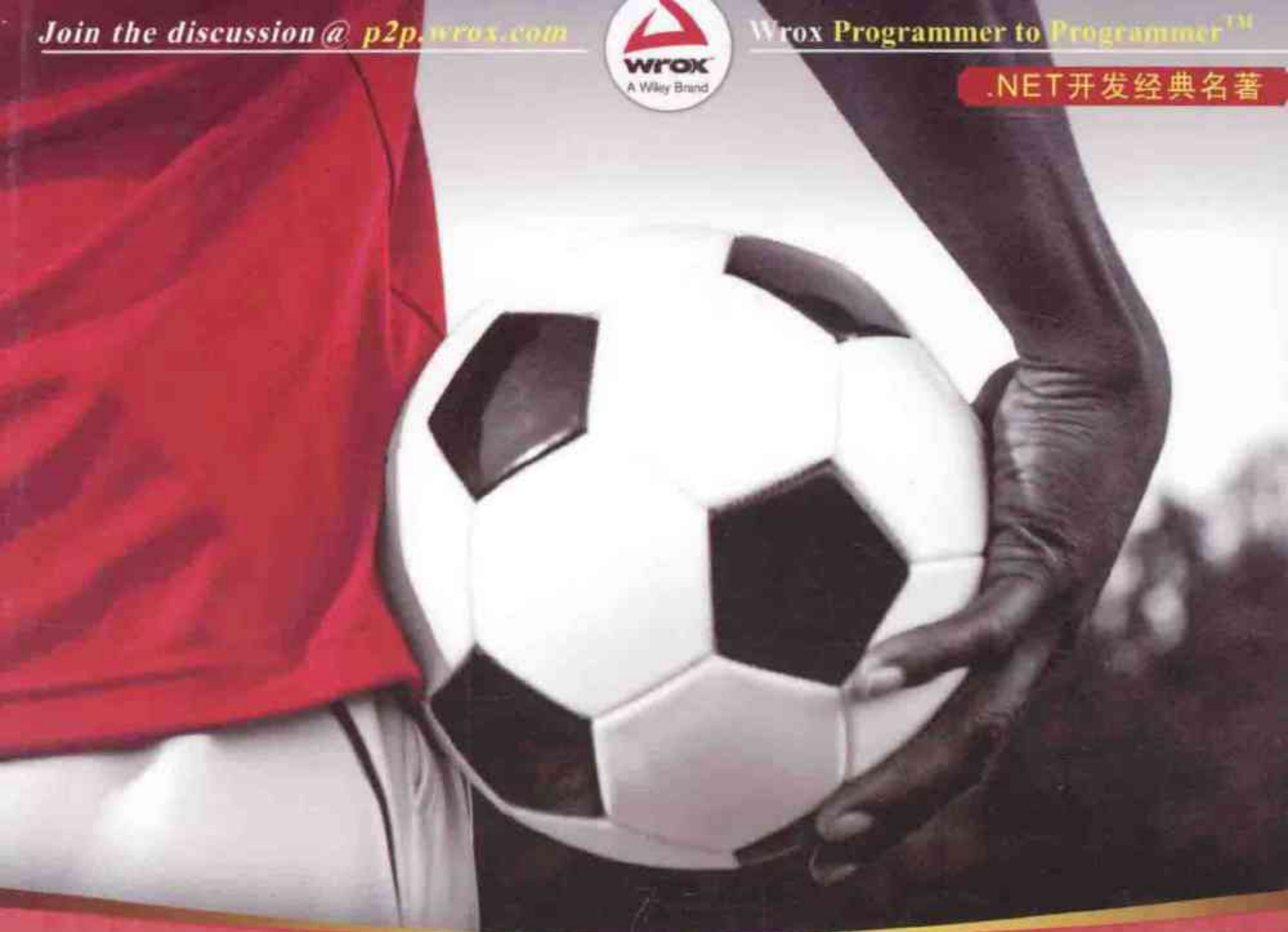


Join the discussion @ p2p.wrox.com



Wrox Programmer to Programmer™

.NET开发经典名著



Professional ASP.NET 4.5 in C# and VB

ASP.NET 4.5

高级编程 (第8版)

微软 Web Platform Team 首席程序经理 Scott Hunter 作序推荐

Jason N. Gaylord
[美] Pranav Rastogi
Scott Hanselman
李增民 苗 荣

Christian Wenz
Todd Miranda 著



译

清华大学出版社

全面涵盖使ASP.NET应用程序高效运行的新工具

ASP.NET是微软提供的免费Web框架,通过使用HTML、CSS和JavaScript创建Web应用程序。本书针对ASP.NET 4.5做了全面修订,解释了ASP.NET的功能,并把One ASP.NET放在更大的环境下讨论。本书融合了专家团队建立和部署站点的丰富经验,以及经过实地测试的专业知识,同时涵盖ASP.NET 4.5提供的一组新工具,从而创建高效运行的ASP.NET应用程序。

主要内容

- ◆ 介绍通过ASP.NET 4.5使用HTML5和CSS3,以及在ASP.NET Web窗体和ASP.NET MVC中合并使用HTML5和各种新技术
- ◆ 讨论Web窗体中的数据和模型绑定、用LINQ查询数据、从Web服务中提取数据等
- ◆ 阐述ASP.NET 4.5的核心功能,包括导航、个性化、成员资格和角色管理、安全性、母版页等
- ◆ 包括ASP.NET 4.5的新增功能,例如ASP.NET Web API,以及通过OAuth和OpenID提供程序(如Facebook和Twitter)进行身份验证
- ◆ 重点介绍async和await关键字,以及利用SignalR进行实时通信

作者简介

Jason N. Gaylord是United One Resources公司的技术主管,是应用开发人员、博主、ASP.NET MVP和ASPInsider。

Christian Wenz是咨询顾问、培训师、作家、ASP.NET MVP和ASPInsider。

Pranav Rastogi是微软ASP.NET团队的成员、Web开发人员和演讲者。

Todd Miranda拥有NxtDimension Solutions公司,是咨询顾问、培训师和微软MVP。

Scott Hanselman是微软内部的Web开发人员,有自己广受欢迎的博客和播客,他撰写了许多图书,还常常在全球会议上发表演讲。

源代码下载及技术支持

<http://www.wrox.com>

<http://www.tupwk.com.cn/downpage>

清华大学出版社数字出版网站

WQBook

www.wqbook.com

Wrox
An Imprint of
WILEY



wrox.com

Programmer Forums

Join our Programmer to Programmer forums to ask and answer programming questions about this book, join discussions on the hottest topics in the industry, and connect with fellow programmers from around the world.

Code Downloads

Take advantage of free code samples from this book, as well as code samples from hundreds of other books, all ready to use.

Read More

Find articles, ebooks, sample chapters and tables of contents for hundreds of books, and more reference resources on programming topics that matter to you.

ISBN 978-7-302-35323-2



9 787302 353232 >

定价: 128.00元

.NET 开发经典名著

ASP.NET 4.5 高级编程

(第 8 版)

Jason N. Gaylord

Christian Wenz

[美] Pranav Rastogi 著

Todd Miranda

Scott Hanselman

李增民 苗 荣 译

清华大学出版社

Jason N. Gaylord, Christian Wenz, Pranav Rastogi, Todd Miranda, Scott Hanselman
Professional ASP.NET 4.5 in C# and VB
EISBN: 978-1-118-31182-0
Copyright © 2013 by John Wiley & Sons, Inc., Indianapolis, Indiana
All Rights Reserved. This translation published under license.

本书中文简体字版由 Wiley Publishing, Inc. 授权清华大学出版社出版。未经出版者书面许可, 不得以任何方式复制或抄袭本书内容。

北京市版权局著作权合同登记号 图字: 01-2013-7010

Copies of this book sold without a Wiley sticker on the cover are unauthorized and illegal.

本书封面贴有 Wiley 公司防伪标签, 无标签者不得销售。

版权所有, 侵权必究。侵权举报电话: 010-62782989 13701121933

图书在版编目(CIP)数据

ASP.NET 4.5 高级编程: 第 8 版 / (美) 盖洛德 (Gaylord, J.N.) 等著; 李增民, 苗荣译. —北京: 清华大学出版社, 2014

(.NET 开发经典名著)

书名原文: Professional ASP.NET 4.5 in C# and VB

ISBN 978-7-302-35323-2

I. ①A… II. ①盖… ②李… ③苗… III. ①网页制作工具—程序设计 IV. ①TP393.092

中国版本图书馆 CIP 数据核字(2014)第 020934 号

责任编辑: 王 军 李维杰

装帧设计: 牛静敏

责任校对: 戚凤进

责任印制: 刘海龙

出版发行: 清华大学出版社

网 址: <http://www.tup.com.cn>, <http://www.wqbook.com>

地 址: 北京清华大学学研大厦 A 座 邮 编: 100084

社 总 机: 010-62770175 邮 购: 010-62786544

投稿与读者服务: 010-62776969, c-service@tup.tsinghua.edu.cn

质 量 反 馈: 010-62772015, zhiliang@tup.tsinghua.edu.cn

印 刷 者: 清华大学印刷厂

装 订 者: 三河市新茂装订有限公司

经 销: 全国新华书店

开 本: 185mm × 260mm 印 张: 78.25 字 数: 2100 千字

版 次: 2014 年 1 月第 1 版 印 次: 2014 年 4 月第 1 次印刷

印 数: 1 ~ 4000

定 价: 128.00 元

产品编号: 071936-01

译者序

ASP.NET 是当前十分成熟的 Web 开发平台，在最新版的 ASP.NET 4.5 中，微软引入了许多新特性，包括设置正则表达式以执行匹配的超时时间、为应用程序域设置区域性信息、支持 Unicode(UTF-16)编码、支持按区域信息进行字符串比较和排序、支持泛型类型、异步文件操作、支持新的 HTML5 表单类型、可以直接把数据访问方法绑定到控件、支持 WebSockets 协议等，让众多的微软平台开发人员能够以更高效的方式开发 Web 应用。

作为 Wrox 经典畅销书之一，本书英文版自出版以来就备受关注，是实实在在的 ASP.NET 参考大全，涵盖了 ASP.NET 编程的所有方面。能够受邀翻译本书，我们感到十分荣幸。在翻译过程中，我们深深感受到了大师的力量，为了追求准确，我们还参考了大量资料，确保尽可能表达清楚原书。

书中不仅涉及 ASP.NET 编程涉及的所有知识点和技术，既包括 ASP.NET Web 窗体的结构、控件、提供程序、母版页、站点导航、成员资格和角色管理等基础知识，也涵盖数据访问、安全性、状态管理、高速缓存、客户端开发等有经验开发人员较为关心的难点问题，对于 ASP.NET 4.5 引入的新特性，本书也都一一涉及。附录对本书内容做了补充，涉及旧项目的迁移、COM 集成、一些有用的 ASP.NET 工具、使用 NuGet 扩展 Visual Studio 等。阅读完本书，相信读者将能够透彻领悟 ASP.NET 平台及相关技术。

对于这本经典之作，我们在翻译过程中力求“信、雅、达”，但鉴于水平有限，错误和失误在所难免，欢迎读者指正。本书由李增民、苗荣翻译，参与翻译的还有孔祥亮、陈跃华、杜思明、熊晓磊、曹汗鸣、张云、王通、方峻、李小凤、洪妍、高娟妮、陈笑、蒋晓冬、邱培强、李亮辉等。

在这里要特别感谢清华大学出版社的李阳和李维杰编辑，他们为本书的翻译投入了巨大的热情并付出许多心血。最后，感谢读者选用本书，希望你们能早日成为 ASP.NET 编程高手，领略 ASP.NET 编程之美！

序

多年来，ASP.NET 一直随着 Web 的发展而发展。Web 窗体在第一次发布时，就是革命性的模型，它使用功能丰富的控件封装 Web 行为，从而把面向对象编程引入 Web。之后，Ajax 开始成为常见术语，ASP.NET 演变为包含丰富的 Ajax 库和 UpdatePanel 控件，从而简化了 Ajax 编程。随后，MVC 模式开始流行，ASP.NET MVC 为使用 MVC 模式创建 ASP.NET 应用程序提供了框架。后来，ASP.NET 回溯到 ASP.NET Web Pages，该框架提供了一种简单的 PHP 编程风格，帮助人们学习为 Web 编程。随着互联设备(如智能手机和平板电脑)的出现，程序员需要一种方式来创建可从这些设备调用的 API，于是 ASP.NET Web API 诞生了。最近的新增内容是 ASP.NET SignalR，它为应用程序提供了一种编程模型，这种应用程序需要实时事件来显示数据的实时更新，例如股票报价。

每次 Web 发生改变时，ASP.NET 也会随之改变，我们继续尽力使每个 ASP.NET 框架与未来的最新发展同步。在 ASP.NET 4.5 中，目标之一是确保 Web 窗体紧跟上述其他框架中的进步。模型绑定、隐含 JavaScript 和 NuGet 库等功能被添加到 Web 窗体中，这样 Web 窗体开发人员就可以访问新框架中创建的新功能。一些工具也被添加到 ASP.NET 4.5 中，这使 ASP.NET Web API 可以添加到任意 ASP.NET 项目中。你将目睹 One ASP.NET 的演化，它允许所有的框架和功能一起工作。使用本书开始 One ASP.NET 和 ASP.NET 4.5 之旅吧！

— Scott Hunter

Microsoft 公司 Web 平台团队的首席项目经理

作者简介



Jason N. Gaylord 是一位应用程序开发人员，是 United One Resources (位于宾夕法尼亚州的威尔克斯-巴里市)的技术主管，他的编程生涯开始于一条简单的 GOTO 语句，之后 15 年都在开发 Windows 和 Web 应用程序。在这段时间，Jason 计划、建立、测试和部署了好几个企业级应用程序，包括与财务和操作系统的集成，以及由美国一些顶级银行使用的 B2B Web 应用程序。

在日常活动以外，Jason 还发表博客、演讲，组织技术会议，包括 TECHbash 和 NEPA GiveCamp。他是 .NET Valley 技术用户组的组长，这个用户组直接给 ASP.NET 团队提供产品反馈。访问他的网站 jasongaylord.com，可以更多地了解他。他的 Twitter 账户是 @jgaylord。



Christian Wenz 在 1993 年时几乎只使用 Web 技术，但是之后就变了，作为开发人员和项目主管，他负责中型企业到大型跨国企业的网站。作为作家，他编写并与他人合作编写了 100 余本书，这些书被翻译为 10 种语言。作为一位顾问，他帮助公司和开发团队使应用程序更可靠、运转得更好、更安全。Christian 常常为各种 IT 杂志写文章，是世界级开发会议上颇受欢迎的演讲者，总是与他人共享令自己激动的技术。他还为各种开源项目贡献力量，是一位 Microsoft ASP.NET MVP

和 ASPInsider，并且是多个开发认证的合作作者。他的 Twitter 账户是 @chwenz。



Pranav Rastogi 是 Microsoft ASP.NET 产品团队的一员，在西雅图市。过去的几年里，Pranav 涉足许多领域，例如 ASP.NET Web 窗体、动态数据以及最近的 ASP.NET SignalR。他积极了解 Web 开发人员在 Microsoft 产品上面临的困难，帮助他们使用有效的工具、包或库，以提升他们在 Microsoft 产品上的 Web 开发体验。Pranav 还定期在各种世界级会议上发表与 ASP.NET 相关的演讲，他在博客 http://blogs.msdn.com/b/prana_rastogi/ 上谈论其经验。在加入 Microsoft 之前，Pranav 毕

业于佛罗里达大学的计算机专业。

Pranav 也是一位美食家，有时在家做大厨。他是一位冒险家，常常去荒地冒险。在闲暇时刻，Pranav 常常为朋友练习做酒吧侍者的技巧。如果你知道有人要聚会，需要一位酒吧侍者，就可以联系他。Pranav 的 Twitter 账户是 @rustd。



Todd Miranda 是 .NET 技术和软件技艺的积极支持者，他有 20 余年的各种平台开发经验，自 .NET 在 2000 年发布以来，他就开始涉足 .NET 了。他使用过许多 .NET 平台，但主要关注 ASP.NET。他拥有 NxtDimension Solutions 公司，该公司为 Microsoft 产品提供咨询和培训。作为 Microsoft MVP，Todd 与 Microsoft 紧密合作，他是伯明翰市 .NET 用户组的合作创始人，在开发社区中一直很活跃。他在 Microsoft 开发议题上记录了数百个在线培训视频，包括安全性、JavaScript、Silverlight、WPF、Expression 和 ASP.NET。他是一位 INETA 演讲者，定期在用户组、社区会议和技术会议上发言。Todd 的 Twitter 账户是 @tmiranda。



Scott Hanselman 是一名 Web 开发人员，他的博客 <http://hanselman.com> 已存在 10 余年。他工作的重点是 Azure 和 ASP.NET，他的家庭办公室在俄勒冈州的波特兰市。Scott 有三个博客，<http://hanselminutes.com> 用于讨论技术，<http://thisdevelopmentslife.com> 用于讨论开发人员的生活和爱情，<http://ratchetandthegEEK.com> 用于讨论大众文化和技术媒介。他编写了许多图书，对全世界近 50 万开发人员发表自己的看法。Scott 的 Twitter 账户是 @shanselman。

技术编辑简介

Ken Cox 自 20 世纪 90 年代中期发布最早的 ASP 传统版本以来，就在用 Microsoft 技术创建 Web 应用程序。他处理过数十个各种规模的 ASP.NET 站点，编写了 *ASP.NET 3.5 For Dummies* (由 John Wiley & Sons 出版) 一书。他在退休前，连续 15 年成为 Microsoft MVP for ASP.NET，且一直是 ASPInsider。Ken 目前在家中做顾问工作，他住在加拿大安大略湖的尼伯斯小镇。

Bipin Joshi 是一位独立的博客主和作家，为明显不相关的主题——瑜伽和技术——撰写文章。作为一位前软件顾问和培训师，他从 1995 年开始编程，自 .NET Framework 诞生以来就一直在使用它。他是一位出版颇丰的作家，编写或与他人合并编写了超过 6 本的图书，并发表了许多 .NET 技术文章。在做软件顾问和培训师期间，Bipin 是一位 Microsoft MVP 和 Microsoft MCT。他还编写了几本有关瑜伽的图书。由于信佛瑜伽，他喜欢沉醉在瑜伽中，编写有关瑜伽、生活和技术的文章。他的网址是 www.bipinjoshi.com。

John Petersen 早期就采用 Microsoft .NET 平台，在构建和开发软件方面有 20 余年的经验。目前，John 是 Neudesic 有限责任公司技术平台组的业务主管、Microsoft 国家系统整合商和 Gold ISV 合作伙伴，他 10 次当选 Microsoft MVP。John 当前是 ASP.NET/IIS MVP。John 在许多业界会议上发言，包括 VSLive 和 DvConnections，以及许多区域会议和 Code Camps。John 还定期给 *CODE Magazine* 撰写文章。

Jeffery Tay 自 2002 年就开始开发和设计 .NET 解决方案，特别擅长教育和医疗领域的解决方案。他在过去三年当选 Microsoft MVP，在 Java 和大多数 Microsoft 技术方面都有丰富的经验，例如 Silverlight、SQL Server 和 Windows Server。他是新加坡国立大学的副校长，并在该大学领导一个小组，管理大学的学习管理系统，提出使用 IT 提升教学效果的新解决方案。Jeffery 的联系邮箱是 taykwama@hotmail.com。

Deepak Verma 在过去 10 年一直负责开发和设计 Web 解决方案。在这些年里，Deepak 涉足的技术包括 Flash 脚本编程、Flex、ColdFusion、Silverlight、Java 和 PHP，现在他主要关注 Microsoft 产品。他目前是一位软件开发工程师，主要负责测试 Microsoft 产品。在 Microsoft，他负责 VS 动态数据工具、多目标代码、Razor 编辑器和工具的编写，以及 Azure SDK for Java、PHP & .NET、VS HTML 和 CSS 编辑器、Azure 移动服务和 NuGet。他与妻子 Madhu、孩子 Adya 和 Lian 住在华盛顿的雷德蒙德。他的联系邮箱是 deepu_verma@yahoo.com。

致 谢

我要感谢 Wrox 出版社和 John Wiley & Sons 为本书花费的时间和努力。没有他们的支持，本书就不可能出版。感谢 Scott Hunter、Scott Hanselman、Damian Edwards 和 ASP.NET 小组的其他成员将 ASP.NET 4.5 推向市场的卓越领导才能。感谢 Scott Guthrie、Steve Smith 和 Rob Howard 允许我加入 ASPInsiders 项目，并帮助我获得许多素材。感谢 ASPInsider 和 Microsoft MVP 的所有成员多年来的支持。感谢 Frank Sorokach 允许我花必要的时间开始我的开发生涯，感谢 Louis Cesare 和 Sean Higgins 支持我的社区活动，在我需要给 ASP.NET 小组提供反馈时给了我时间。感谢 Luzerne County 社区学院和宾夕法尼亚州立大学允许我们的用户组在宾夕法尼亚州东北部发展起来。最后，感谢我的家人，尤其是父母 Deb 和 Tom Gaylord、祖父母、叔叔和嫂子、妻子 Lisa 和孩子们的支持。写书并不容易，他们肯定理解使本书面市所需的时间和精力。

—JASON N. GAYLORD

每本书都是所有编辑精诚合作的结果(获得、开发、技术等)，但就这个主题而言，本书作者团队也十分强大！所以感谢 Jason、Pranav、Scott 和 Todd，能与你们合作真的很荣幸。我还要感谢本书以前版本的作者，感谢 Wiley 参与本书的每个人，以及技术编辑。

7 年前，我在处理 Wrox imprint of Wiley 的另一本书，并答应 Yvonne，在那本书出版后，会邀请她到欧洲的一家顶级饭馆吃饭，但事情并没有像计划的那样进行，包括合作的作者中途退出，所以那本书和那顿饭都没有兑现。现在咱们再试一次——6 月 13 日，行吗？

—CHRISTIAN WENZ

我要感谢西雅图所有咖啡馆的主人，放松的环境和香醇的咖啡是我编写本书的灵感源泉。

—PRANAV RASTOGI

感谢 Kelly 忍受我在那么多个夜晚进行写作，感谢两个公主 Amber 和 Sydney 允许我对她们陪伴和关注得较少。感谢 Kevin、Mary 和 Wiley 小组其他人对本书的帮助和支持。还要感谢本书的合作作者，与你们一起编写本书真的很棒。

—TODD MIRANDA

非常感谢 Pranav Rastogi、Scott Hunter、Damian Edwards、Eilon Lipton，以及整个 ASP.NET 小组所做的实际工作。

—SCOTT HANSELMAN

前言

ASP.NET 4.5 是一种令人惊异的建立 Web 解决方案的技术。ASP.NET 自从诞生之日起，就引入了许多技术，成为当今最有竞争力的 Web 框架。ASP.NET 4.5 建立在 ASP.NET 以前版本的基础之上，但它主要关注的是开发人员的工作效率。

本书将介绍 ASP.NET 的所有内容，除了论述新主题外，还列举了一些有关这些新技术在实际操作中运用的例子。

0.1 ASP.NET 的过去、现在和未来

ASP.NET 4.5 是该产品的另一个主要版本，建立在以前版本的基础之上，并带有新增的功能。该版本继续朝着如下方向前进：使 ASP.NET 开发人员成为 Web 领域最有效率的开发人员。在 ASP.NET 的每个版本中，Microsoft 团队都把其目标设定为开发效率、管理、性能和可伸缩性。本书将讨论 ASP.NET 和 .NET Framework 中的新功能。

ASP.NET 已推出 10 余年。它在第一次发布时，重点关注的是将使用 VBScript 或 JScript 创建 ASP 页面的现有 Web 编程模型转变为面向对象模型。为了帮助进行此转变，ASP.NET 包含了几个服务器控件以封装常见的功能。

随着时间的流逝，开发人员开始需要更多地控制所显示的标记。于是，ASP.NET 有了变化，出现了新的 Web 范型，诞生了 ASP.NET MVC 等技术。另一方面，ASP.NET Web 窗体继续控制着开发的主要过程。因此，以前的几个 ASP.NET 版本把重点放在用服务器控件正确显示标记上。

但是，与大多数 Microsoft 团队一样，VS 团队的发布周期也是 2 到 3 年，但 ASP.NET 团队跟不上 Web 技术的发展步伐。本书的第 1 章将详细解释 ASP.NET 团队如何更频繁地发布产品的带外(Out-Of-Band, OOB)版本，以处理这个两难的问题。该章还讨论 ASP.NET 团队的开源活动。实际上，许多 ASP.NET 技术都已经发布为真正的开源产品。

用 ASP.NET 进行 Web 开发是件很令人兴奋的事。

0.2 使用本书的要求

在使用本书中的示例时，最好安装 Visual Studio 2012，但只安装 Microsoft 的 Notepad 或 WebMatrix，以及 .NET Framework 4.5 附带的命令行编译器也可行。要运行本书的所有示例，需要：

- Windows 8、Windows 7、Windows Vista、Windows Server 2012 或 Windows Server 2008
- Visual Studio 2012(这会安装 .NET Framework 4.5)
- SQL Server 2012、2008、2005 或 SQL Server Express Edition

使用 ASP.NET 4.5 时，并不需要安装 Microsoft Internet Information Services (IIS)，因为 Visual

Studio 2012 包含内置的 Web 服务器 IIS Express。即使没有 SQL Server 的完整版本，也不必担心。使用该数据库的许多例子都可以改为使用 SQL Server Express Edition，可以从 Internet 上免费下载该数据库版本。

0.3 本书读者对象

本书介绍的是 ASP.NET 4.5 提供的新功能，并解释 ASP.NET 的基础知识。因此，读者应大致了解 Web 技术，如 ASP.NET 的以前版本或 PHP 等其他 Web 技术。如果读者具备 Web 编程的基础知识，阅读本书的内容就不会有什么问题。

如果读者是 ASP.NET 新手，就应首先阅读由 Imar Spaanjaars 编写的《ASP.NET 4.5 入门经典(第 7 版)》，掌握 ASP.NET 的基础知识。

除了具备 Web 技术的相关知识之外，读者还应对基本的编程结构有一定的了解，例如变量、For Each 循环、面向对象编程等。

那么，本书到底是适用于 Visual Basic 开发人员还是 C#开发人员呢？这两类开发人员都可以阅读本书！如果示例代码的区别比较大，本书就会提供 VB 和 C#两个版本的代码(VB 版本的代码可从本书支持站点免费下载)。此时，为了简洁起见，会省略 C#示例中的 HTML 标记。

0.4 本书内容

本书主要介绍 ASP.NET 4.5，详细阐述该版本中包含的每个主要新功能。下面是每章的主要内容：

- **第 1 章“One ASP.NET”**：扩展健康的 Web 生态系统，这是本书最有趣的章节之一。ASP.NET 推出 10 余年后，已经成为最丰富的 Web 开发技术之一。本章描述了 ASP.NET 的未来规划，包括一个新的蓝图：One ASP.NET。
- **第 2 章“使用 ASP.NET 进行 HTML5 和 CSS3 设计”**：Visual Studio 2012 比较注重建立基于 HTML5 和 CSS3 的 Web 应用程序。本章将详细介绍如何高效地使用 HTML5 和 CSS3 设计 ASP.NET 应用程序。
- **第 3 章“ASP.NET Web 窗体的结构”**：本章介绍 ASP.NET 应用程序的架构，以及为单个 ASP.NET 页面提供的结构和架构。本章将描述如何使用 Visual Studio 2012 构建 ASP.NET 应用程序，讨论 ASP.NET 中的文件夹和文件，论述编译代码的方式以及如何执行跨页的传送过程。本章最后列出处理 Visual Studio 2012 中的类的简单方法。
- **第 4~7 章“控件”**：将这 4 章放在一起是因为它们讨论的都是服务器控件或用户控件。这几章首先介绍服务器控件的概念及其在 ASP.NET 开发中的重要作用。除了讨论服务器控件的架构之外，还将深入探讨服务器控件在 ASP.NET 开发项目中的作用。第 4 章“ASP.NET 服务器控件和客户端脚本”介绍使用服务器控件的基础知识，第 5 章“ASP.NET Web 服务器控件”介绍最新 ASP.NET 版本中的控件，第 6 章“验证服务器控件”描述了一组特殊的服务器控件——用于验证的服务器控件。使用这些控件可以创建从初级到高级的表单验

证。第7章“用户控件和服务器控件”描述了如何创建自己的服务器控件，以及如何在应用程序中使用它们。

- **第8~13章“数据访问”**：这部分讨论数据访问。目前几乎所有的Web应用程序都至少与一种类型的数据源交互，这些章节讨论了从关系数据到JSON的所有数据访问。第8章“数据绑定”介绍一些在把数据发布给控件之前支持编程处理数据的底层功能。第9章“模型绑定”介绍绑定的基础知识，并将模型绑定应用于Web窗体。第10章“使用LINQ查询”介绍LINQ，以及如何在Web应用程序中高效地使用这个新功能。第11章“Entity Framework”讨论如何把对象从数据库映射到代码中。使用Visual Studio 2012可以可视化地设计实体数据模型。第12章“ASP.NET 动态数据”描述动态数据如何快速、方便地从数据库中建立报表和数据输入应用程序。第13章“使用服务”指出了传统ASMX Web服务和新型Web API之前的区别。
- **第14章“提供程序模型概述”**：ASP.NET 内置的许多系统大大方便了开发人员的工作，显著提高了他们的效率。这些系统以“提供程序模型”为基础，提供程序模型是可扩展的。本章将概述提供程序模型，及其在ASP.NET 4.5中的用法。
- **第15章“扩展提供程序模型”**：在介绍提供程序模型后，本章将探讨扩展ASP.NET 4.5中提供程序模型的方式，并回顾提供程序模型的几个扩展示例。
- **第16章“使用母版页”**：母版页是ASP.NET 技术中的一项强大功能，提供了创建模板页面的方式，该模板页面能用于整个应用程序，而不是单个页面。本章介绍这些模板的创建方式，以及如何把它们应用于ASP.NET 应用程序中的内容页面。
- **第17章“站点导航”**：显然，许多开发人员都不是简单地开发单个页面，而是创建应用程序。因此，他们需要的是能贯穿整个应用程序功能处理的机制，而不是只具有页面处理功能的机制。ASP.NET 4.5 提供的一项应用程序功能就是站点导航系统，本章就介绍这个系统。底层的导航系统允许通过XML 文件定义应用程序的导航结构，该系统引入了一整套全新的导航服务器控件来处理XML 文件中的数据。
- **第18章“个性化”**：开发人员总是在寻找存储与终端用户相关信息的方式。在存储了这些信息之后，这种个性化数据就可以用于用户以后的访问，或者用于获取同一应用程序中的其他页面。ASP.NET 小组开发了一种存储这些信息的方式——ASP.NET 个性化系统。该系统的最大优点是，可以在web.config 文件中配置系统的所有操作。
- **第19章“成员资格和角色管理”**：本章介绍成员资格和角色管理系统，该系统用于简化向ASP.NET 应用程序添加身份验证和授权的过程。这两个系统是可扩展的，使以前较复杂的身份验证和授权方式永远成为历史。本章主要讨论使用web.config 文件控制这些系统的应用方式，以及使用底层系统的新服务器控件。
- **第20章“安全性”**：本章讨论ASP.NET 4.5 中除了成员资格和角色管理功能之外的安全性，深入研究ASP.NET 技术中固有的身份验证和授权机制，以及HTTP 访问类型和模拟。
- **第21章“状态管理”**：ASP.NET 是一种基于请求/响应的技术，因此状态管理和请求、响应的性能非常重要。本章介绍在ASP.NET 开发领域中这两个相互独立并且都非常重要的主题。
- **第22章“高速缓存”**：ASP.NET 的本质是请求/响应，因此服务器上的高速缓存(存储以前生成的结果、图像和页面)对于ASP.NET 应用程序的性能来说非常重要。本章介绍ASP.NET

提供的一些高级高速缓存功能,包括 ASP.NET 4.5 中的 SQL 高速缓存禁用功能。本章还将介绍对象高速缓存以及对象高速缓存的可扩展性。

- **第 23~27 章“客户端开发”**: 这 5 章讨论最热门的技术领域——移动技术。消费者和商家都希望能进行实时的数据传递,设计流畅的应用程序可以在移动电话、平板电脑和功能全面的 PC 上运行。第 23 章“ASP.NET AJAX”介绍如何使用 AJAX 建立应用程序。第 24 章“AJAX 控件工具集”回顾一系列可用于 AJAX 技术的控件。第 25 章“jQuery”概述 JavaScript 库 jQuery。第 26 章“实时通信”比较提供传统实时通信的方式和使用 HTML5 Web 套接字或 SignalR 的现代方式,后者包含一套混合解决方案。第 27 章“开发移动网站”专门讨论更新客户端的方法,以提供更好的可移式客户端支持。
- **第 28~33 章“应用程序的配置和部署”**。现在,读者应牢固地掌握了创建 ASP.NET 应用程序的技巧。这 6 章重点讨论应用程序的配置和优化,最后将部署应用程序。第 28 章“配置”讲述如何使用各种配置文件修改 ASP.NET 的功能和操作系统。第 29 章“调试和错误处理技术”描述如何在应用程序中正确构建结构化错误处理机制,还将说明如何使用各种调试技术查找应用程序可能包含的错误。第 30 章“模块和处理程序”介绍 ASP.NET 处理 HTTP 请求的两种方法——HttpModule 和 IHttpHandler。这两种方法都提供了对 ASP.NET 底层处理过程的独特访问级别。第 31 章“异步通信”讨论了从客户端到服务器上某方法的单向通信,以及从客户端到宿主服务的单向通信。第 32 章“国际化应用程序的建立”介绍构建国际化的 Web 应用程序时应考虑的一些重要事项。第 33 章“打包和部署 ASP.NET 应用程序”进一步介绍构建过程,说明如何打包 ASP.NET 应用程序以便于部署。
- **第 34 章“ASP.NET MVC”**: ASP.NET MVC 使许多开发社区兴奋不已。ASP.NET MVC 提供了使用 MVC(Model-View-Controller, 模型-视图-控制器)模式创建 ASP.NET 应用程序的方法,许多开发人员都对该模式充满期待。ASP.NET MVC 为应用程序提供了可测试性、灵活性和可维护性。注意,ASP.NET MVC 并不是人们熟知和喜欢的 ASP.NET 架构的替代物,而是构建 ASP.NET 应用程序的另一种方式。
- **第 35 章“ASP.NET Web Pages 和 Razor”**: ASP.NET Web Pages 是一项较新的技术,允许开发人员使用 Razor 语法建立交互式的 Web 应用程序。本章概述如何使用 Microsoft WebMatrix 建立 Web Pages 应用程序,还列出了一些较流行的帮助器,它们使用 Razor 语法显示有效的 HTML 和 CSS。
- **附录 A“迁移 ASP.NET 旧项目”**: 在一些情况下,要从头开始构建 ASP.NET 4.5 应用程序——新建所有内容。但在许多情况下,并不需要这样做,而是需要迁移使用 .NET Framework 以前版本构建的 ASP.NET 应用程序,使它们能在 .NET Framework 4.5 上运行。
- **附录 B“COM 集成”**: 肯定有一些组件是用以前的技术创建的,且不希望重建它们,但希望把它们集成到新的 ASP.NET 应用程序中。此时,使用 .NET Framework 可以相当简单地把以前的 COM 组件集成到应用程序中。本附录还介绍如何建立 .NET 组件,而不是采用以前 COM 组件的体系结构。
- **附录 C“ASP.NET 终极工具”**: 本附录介绍 ASP.NET 开发人员可以使用的工具,其中有许多工具有助于加速开发过程。在许多情况下,还可以使读者成为更优秀的开发人员。
- **附录 D“管理”**: ASP.NET 小组除了让开发人员在创建 ASP.NET 应用程序时更轻松、效率更高之外,还花费了大量的精力来简化应用程序的管理。过去,使用 ASP.NET 1.0/1.1 管

理 ASP.NET 应用程序时，必须修改 XML 配置文件中的值。本附录将概述这个版本中的 GUI 工具，利用它们可以方便、高效地管理 Web 应用程序。

- 附录 E“动态类型和语言”：随着 ASP.NET 4.5 的发布，现在可以使用 IronRuby 和 IronPython 来创建应用程序。本附录简要介绍在创建 Web 应用程序的过程中应如何使用动态语言。
- 附录 F“ASP.NET 联机资源”：本附录列出了一些有价值的联机资源，便于读者进一步理解 ASP.NET。
- 附录 G“使用 NuGet 扩展 Visual Studio”：VS 2012 使用 NuGet 的 Microsoft 版本 Package Manager 提供了一个可扩展的模型。NuGet 允许开发人员创建可发布的包来共享文件、文件夹和库，该包在公共种子或私有种子中可用。

0.5 源代码

在读者学习本书中的示例时，可以手动输入所有的代码，也可以使用本书附带的源代码文件。本书使用的所有源代码都可以从本书合作站点 <http://www.wrox.com/> 或 www.tupwk.com.cn/downpage 上下载。登录到站点 <http://www.wrox.com/>，使用 Search 工具或使用书名列表就可以找到本书。接着单击 Download Code 链接，就可以获得所有的源代码。既可以选择下载一个大的包含本书所有代码的 ZIP 文件，也可以只下载某个章节中的代码。



由于许多图书的标题都很类似，因此按 ISBN 搜索是最简单的，本书英文版的 ISBN 是 978-1-118-31182-0。

在下载代码后，只需用解压缩软件对其进行解压缩即可。另外，也可以进入 <http://www.wrox.com/dynamic/books/download.aspx> 上的 Wrox 代码下载主页，查看本书和其他 Wrox 图书的所有代码。记住，可以使用书中列出的程序清单的编号容易地找到所要寻找的代码，如“程序清单 0-1”。

当为大多数可下载的源代码文件命名时，我们会使用这些清单中的数值。对于那些很少的没有用它自己的清单数值命名的程序清单，它们都与文件名匹配，所以很容易就可以在下载源代码文件中找到它们。另外，还可以在 www.wrox.com/go/SQLServer2012DataSets 上下载本书使用的 AdventureWorks 和 NorthWind 数据库。

0.6 勘误表

尽管我们已经尽了各种努力来保证文章或代码中不出现错误，但是错误总是难免的。如果你在本书中找到了错误，例如拼写错误或代码错误，请告诉我们，我们将非常感激。通过勘误表，可以让其他读者避免受挫。当然，这还有助于提供更高质量的信息。

要在网站上找到本书英文版的勘误表，可以登录 <http://www.wrox.com>，通过 Search 工具或书名列表查找本书，然后在本书的细目页面上，单击 Book Errata 链接。在这个页面上可以查看到 Wrox 编辑已提交和粘贴的所有勘误项。完整的图书列表还包括每本书的勘误表，网址是

www.wrox.com/misc-pages/booklist.shtml。

如果你发现的错误在我们的勘误表里还没有出现的话，请登录 www.wrox.com/contact/techsupport.shtml 并完成那里的表格，把你发现的错误发送给我们。我们会检查你的反馈信息，如果正确，我们将在本书的勘误页面张贴该错误信息，并在本书的后续版本中加以修订。

0.7 p2p.wrox.com

要与作者和同行讨论，请加入 p2p.wrox.com 上的 P2P 论坛。这是个基于 Web 的系统，便于你张贴与 Wrox 图书相关的消息和相关技术，与其他读者和技术用户交流心得。该论坛提供了订阅功能，当论坛上有新的消息时，它可以给你传送感兴趣的论题。Wrox 作者、编辑和其他业界专家和读者都会到这个论坛上来探讨问题。

在 <http://p2p.wrox.com> 上，有许多不同的论坛，它们不仅有助于阅读本书，还有助于开发自己的应用程序。要加入论坛，可以遵循下面的步骤：

- (1) 进入 p2p.wrox.com，单击 Register 链接。
 - (2) 阅读使用协议，并单击 Agree 按钮。
 - (3) 填写加入该论坛所需要的信息和自己希望提供的其他可选信息，单击 Submit 按钮。
- 你会收到一封电子邮件，其中的信息描述了如何验证账户，完成加入过程。



不加入 P2P 也可以阅读论坛上的消息，但要张贴自己的消息，就必须先加入该论坛。

加入论坛后，就可以张贴新消息，响应其他用户张贴的消息。可以随时在 Web 上阅读消息。如果能让该网站给自己发送特定论坛中的消息，可以单击论坛列表中该论坛名旁边的 **Subscribe to this Forum** 图标。

要想了解更多的有关论坛软件的工作情况，以及 P2P 和 Wrox 图书的许多常见问题的解答，就一定要阅读 FAQ，只需在任意 P2P 页面上单击 FAQ 链接即可。

目 录

第 I 部分 ASP.NET 基础

第 1 章 One ASP.NET	3
1.1 One ASP.NET 简介	3
1.1.1 ASP.NET 的演变史	4
1.1.2 Web 的演化史	5
1.2 简化复杂的生态系统	6
1.2.1 Web 生态系统	7
1.2.2 起步是很简单的	8
1.2.3 整合生态系统是很简单的	8
1.2.4 真实示例	10
1.3 如何从 One ASP.NET 中获益	13
1.3.1 ASP.NET 使起步很简单	13
1.3.2 ASP.NET 支持 Web 生态系统	13
1.3.3 ASP.NET 更容易查找、添加、 更新站点的各个模块	14
1.3.4 ASP.NET 帮助把一个框架的 概念应用于另一个框架	14
1.3.5 ASP.NET 与 Web 的 发展一样快	14
1.4 本章小结	15
第 2 章 使用 ASP.NET 进行 HTML5 和 CSS3 设计	17
2.1 警告	18
2.2 HTML5 概述	18
2.2.1 HTML5 中的新元素、特性 和值	19
2.2.2 使用新的 HTML5 标记	21
2.2.3 HTML5 中新的 API	23
2.3 CSS3 概述	24

2.3.1 创建样式表	24
2.3.2 CSS 规则	27
2.3.3 CSS 继承	36
2.3.4 元素的布局 and 定位	37
2.3.5 CSS3 中的新功能	43
2.3.6 HTML 和 CSS 的兼容	47
2.4 在 Visual Studio 中使用 HTML 和 CSS	48
2.4.1 在 Visual Studio 中使用 CSS	48
2.4.2 指定 ASP.NET 控件的样式	51
2.4.3 VS 2012 中对 HTML 和 CSS 的改进	53
2.4.4 Page Inspector	54
2.5 本章小结	55
第 3 章 ASP.NET Web 窗体的结构	57
3.1 应用程序的位置选项	57
3.1.1 使用文件系统(IIS Express)	58
3.1.2 使用 IIS	59
3.1.3 使用 FTP	60
3.2 ASP.NET 页面的结构选项	61
3.2.1 内联编码	61
3.2.2 隐藏代码模型	63
3.3 ASP.NET 4.5 的 Page 指令	65
3.3.1 Page 指令	66
3.3.2 Master 指令	68
3.3.3 Control 指令	69
3.3.4 Import 指令	69
3.3.5 Implements 指令	72
3.3.6 Register 指令	72
3.3.7 Assembly 指令	73
3.3.8 PreviousPageType 指令	73

3.3.9	MasterType 指令	74
3.3.10	OutputCache 指令	74
3.3.11	Reference 指令	75
3.4	ASP.NET 的页面事件	75
3.5	处理回送	76
3.6	跨页面传送	77
3.7	ASP.NET 应用程序文件夹	80
3.7.1	App_Code 文件夹	80
3.7.2	App_Data 文件夹	84
3.7.3	App_GlobalResources 文件夹	84
3.7.4	App_LocalResources 文件夹	84
3.7.5	App_WebReferences 文件夹	84
3.7.6	App_Browsers 文件夹	85
3.8	编译	85
3.9	Build Provider	88
3.9.1	使用内置的 Build Provider	90
3.9.2	使用自己的 Build Provider	90
3.10	Global.asax	94
3.11	通过 Visual Studio 2012 使用类	96
3.12	本章小结	99

第 II 部分 控 件

第 4 章 ASP.NET 服务器控件和

客户端脚本 103

4.1	ASP.NET 服务器控件	103
4.1.1	服务器控件的类型	104
4.1.2	使用服务器控件构建页面	105
4.1.3	处理服务器控件的事件	106
4.2	给服务器控件应用样式	107
4.3	CSS 在 ASP.NET 4.5 中的 改变	109
4.4	HTML 服务器控件	109
4.4.1	HtmlControl 基类	110
4.4.2	HtmlContainerControl 类	111
4.4.3	所有的 HTML 类	111

4.4.4	使用 HtmlGenericControl 类	113
4.5	识别 ASP.NET 服务器控件	114
4.6	通过 JavaScript 处理页面和 服务器控件	116
4.6.1	使用 Page.ClientScript. RegisterClientScriptBlock	117
4.6.2	使用 Page.ClientScript. RegisterStartupScript	118
4.6.3	使用 Page.ClientScript. RegisterClientScriptInclude	120
4.7	本章小结	120

第 5 章 ASP.NET Web 服务器控件 121

5.1	Web 服务器控件概述	121
5.2	Label 服务器控件	122
5.3	Literal 服务器控件	124
5.4	TextBox 服务器控件	125
5.4.1	使用 Focus()方法	125
5.4.2	使用 AutoPostBack	126
5.4.3	使用 AutoCompleteType	127
5.5	Button 服务器控件	127
5.5.1	CausesValidation 属性	128
5.5.2	CommandName 属性	128
5.5.3	使用客户端 JavaScript 的 按钮	129
5.6	LinkButton 服务器控件	130
5.7	ImageButton 服务器控件	131
5.8	HyperLink 服务器控件	131
5.9	DropDownList 服务器控件	132
5.10	从集合中可视化地删除 数据项	134
5.11	ListBox 服务器控件	135
5.11.1	允许用户选择多项	136
5.11.2	使用 ListBox 控件的 例子	136
5.11.3	给集合添加项	137
5.12	CheckBox 服务器控件	138
5.12.1	如何确定复选框是否被 选中	139

5.12.2	给复选框赋值	139	5.27.3	使用 StepType 属性	179
5.12.3	排列复选框中的文本	140	5.27.4	给 Wizard 控件添加标题	180
5.13	CheckBoxList 服务器控件	140	5.27.5	使用向导的导航系统	181
5.14	RadioButton 服务器控件	142	5.27.6	使用 Wizard 控件的 事件	181
5.15	RadioButtonList 服务器控件	144	5.27.7	使用 Wizard 控件显示 窗体元素	183
5.16	Image 服务器控件	145	5.28	ImageMap 服务器控件	186
5.17	Table 服务器控件	146	5.29	本章小结	188
5.18	Calendar 服务器控件	148	第 6 章	验证服务器控件	189
5.18.1	从 Calendar 控件中选择 日期	149	6.1	有效性验证	189
5.18.2	从 Calendar 控件中选择 要输出的日期格式	150	6.2	客户端和服务端端的验证	190
5.18.3	选择日期、星期或月份	150	6.3	ASP.NET 验证服务器控件	191
5.18.4	使用日期范围	151	6.3.1	验证原因	192
5.18.5	修改日历的样式和 操作方式	152	6.3.2	ASP.NET 4.5 中的 隐含验证	192
5.19	AdRotator 服务器控件	155	6.3.3	RequiredFieldValidator 服务器控件	195
5.20	Xml 服务器控件	157	6.3.4	CompareValidator 服务器 控件	199
5.21	Panel 服务器控件	157	6.3.5	RangeValidator 服务器 控件	202
5.22	PlaceHolder 服务器控件	159	6.3.6	RegularExpressionValidator 服务器控件	205
5.23	BulletedList 服务器控件	160	6.3.7	CustomValidator 服务器 控件	206
5.24	HiddenField 服务器控件	165	6.3.8	ValidationSummary 服务器 控件	210
5.25	FileUpload 服务器控件	166	6.4	关闭客户端验证功能	213
5.25.1	使用 FileUpload 控件上传 文件	166	6.5	为错误通知使用图像和声音	214
5.25.2	给上传文件指定正确的 ASP.NET 权限	168	6.6	使用验证组	215
5.25.3	理解文件的大小限制	169	6.7	本章小结	218
5.25.4	从同一个页面上传多个 文件	171	第 7 章	用户控件和服务端控件	219
5.25.5	把上传的文件放在 Stream 对象中	173	7.1	用户控件	219
5.25.6	把文件内容从 Stream 对象 移到 Byte 数组中	173	7.1.1	创建用户控件	220
5.26	MultiView 和 View 服务器 控件	174	7.1.2	与用户控件交互	222
5.27	Wizard 服务器控件	177	7.1.3	动态加载用户控件	222
5.27.1	定制边栏导航	178	7.2	服务器控件	226
5.27.2	AllowReturn 属性	179			

7.2.1	Server Control 项目	226
7.2.2	控件的特性	230
7.2.3	控件的显示	231
7.2.4	添加标记特性	234
7.2.5	关于控件 ID	235
7.2.6	给 HTML 设置样式	236
7.2.7	添加客户端功能	238
7.2.8	浏览器功能	244
7.2.9	使用 ViewState	246
7.2.10	引发回送事件	249
7.2.11	处理回送数据	251
7.2.12	复合控件	253
7.2.13	模板化控件	254
7.2.14	在设计期间创建控件	257
7.3	本章小结	268

第III部分 数据访问

第 8 章	数据绑定	271
8.1	数据源控件	271
8.1.1	SqlDataSource 控件	273
8.1.2	AccessDataSource 控件	278
8.1.3	LinqDataSource 控件	278
8.1.4	EntityDataSource 控件	278
8.1.5	给复杂的过滤使用 QueryExtender	280
8.1.6	XmlDataSource 控件	281
8.1.7	ObjectDataSource 控件	282
8.1.8	SiteMapDataSource 控件	283
8.2	数据源控件的高速缓存	284
8.3	数据绑定控件	284
8.3.1	GridView 控件	284
8.3.2	编辑 GridView 中的 数据行	293
8.3.3	删除 GridView 数据	298
8.3.4	DetailsView 控件	300
8.3.5	使用 DetailsView 插入、更新 和删除数据	302
8.3.6	ListView 控件	303
8.3.7	FormView 控件	309

8.4	其他数据绑定控件	312
8.4.1	TreeView 控件	312
8.4.2	Menu 控件	313
8.4.3	Chart 控件	313
8.5	内联数据绑定语法	316
8.5.1	数据绑定语法	317
8.5.2	XML 数据绑定	318
8.6	表达式和表达式生成器	318
8.7	本章小结	321

第 9 章	模型绑定	323
9.1	模型绑定	323
9.1.1	选择数据	324
9.1.2	分页	324
9.1.3	过滤	325
9.1.4	使用值提供程序	325
9.1.5	插入数据	326
9.1.6	更新数据	327
9.2	使用强类型化的控件	329
9.3	扩展模型绑定	330
9.3.1	定制的值提供程序	330
9.3.2	定制的模型绑定器	331
9.3.3	定制的 ModelDataSource	332
9.4	本章小结	334

第 10 章	使用 LINQ 查询	335
10.1	LINQ to Objects	336
10.1.1	传统的查询方法	336
10.1.2	使用 LINQ 替代传统的 查询	338
10.1.3	使用 LINQ 分组数据	345
10.1.4	使用其他 LINQ 运算符	345
10.1.5	建立 LINQ 连接	346
10.1.6	使用 LINQ 分页数据	347
10.2	LINQ to XML	347
10.2.1	一个简单的 LINQ to XML 示例	348
10.2.2	连接 XML 数据	350
10.3	LINQ to SQL	351
10.3.1	使用 O/R 映射器	351

10.3.2	访问和查询数据	353	12.1.3	运行应用程序	397
10.3.3	使用其他 SQL 查询方法	356	12.1.4	运行应用程序的结果	397
10.3.4	通过 LINQ 插入、更新和 删除查询	358	12.1.5	向已有页面中添加动态 数据	400
10.4	LINQ to Entities	361	12.2	理解模型绑定	401
10.4.1	创建 Entity Framework 数据模型	362	12.2.1	特性驱动的 UI	401
10.4.2	访问数据	363	12.2.2	特性驱动的验证	403
10.4.3	编写 LINQ 查询	363	12.3	本章小结	404
10.5	本章小结	365	第 13 章	使用服务	405
第 11 章	Entity Framework	367	13.1	不同系统之间的通信	405
11.1	能否使用同一种语言	368	13.2	建立一个简单的 XML Web 服务	407
11.1.1	开发 workflow 选项	369	13.2.1	WebService 页面指令	408
11.1.2	实体数据模型	369	13.2.2	Web 服务的基类文件	409
11.2	创建第一个实体数据模型	370	13.2.3	把定制的 DataSet 显示为 SOAP	409
11.2.1	使用 EDM 向导	371	13.2.4	XML Web 服务接口	411
11.2.2	使用 Entity Framework 设计器	373	13.3	使用简单的 XML Web 服务	413
11.2.3	使用 EDM 建立 ASP.NET Web 页面	374	13.3.1	添加 Web 引用	413
11.3	理解关系	376	13.3.2	在客户端应用程序中 调用 Web 服务	415
11.3.1	一对一和一对多关系	376	13.4	重载 WebMethod	417
11.3.2	多对一和多对多关系	379	13.5	高速缓存 Web 服务的响应	420
11.4	在 EDM 中使用继承功能	381	13.6	使用 SOAP 报头	421
11.5	使用 EntityDataSource 控件	384	13.6.1	使用 SOAP 报头建立 Web 服务	421
11.5.1	创建基本页面	384	13.6.2	通过 SOAP 报头使用 Web 服务	422
11.5.2	配置数据源控件	385	13.6.3	使用 SOAP 1.2 请求 Web 服务	424
11.6	Entity Framework Code First	387	13.7	异步使用 Web 服务	425
11.6.1	创建 Code First 模型	387	13.8	WCF	427
11.6.2	约定-配置	390	13.8.1	WCF 概述	427
11.6.3	Code First 中的关系	391	13.8.2	建立 WCF 服务	428
11.6.4	Code First Migrations	392	13.8.3	建立 WCF 使用者	433
11.7	本章小结	394	13.8.4	添加服务引用	434
第 12 章	ASP.NET Dynamic Data	395	13.8.5	使用数据协定	436
12.1	Dynamic Data 功能	395	13.8.6	定义名称空间	440
12.1.1	默认应用程序中创建的 核心文件	395			
12.1.2	应用程序特性	396			

13.8.7	使用 WCF 数据服务	440
13.8.8	创建第一个服务	441
13.8.9	查询接口	447
13.8.10	在 ASP.NET 中使用 WCF 数据服务	458
13.9	ASP.NET Web API	460
13.9.1	建立第一个 Web API 项目	461
13.9.2	理解 Web API 路由	464
13.9.3	使用 Web API	465
13.10	本章小结	468

第IV部分 提供程序

第 14 章	提供程序模型概述	471
14.1	提供程序概述	472
14.2	ASP.NET 4.5 中的提供 程序模型	473
14.2.1	设置提供程序以使用 SQL Server 2005、2008 或 2012	474
14.2.2	成员资格提供程序	479
14.2.3	角色提供程序	483
14.2.4	个性化提供程序	486
14.2.5	站点地图提供程序	488
14.2.6	会话状态提供程序	489
14.2.7	Web 事件提供程序	491
14.2.8	提供程序的配置	498
14.3	本章小结	500
第 15 章	扩展提供程序模型	501
15.1	提供程序是较大体系结构中 的一层	501
15.2	通过基于特性的编程修改 提供程序的行为	502
15.2.1	通过 SqlMembership- Provider 建立简单的 密码结构	502
15.2.2	通过 SqlMembership- Provider 建立强类型的 密码结构	505

15.3	ProviderBase 类	506
15.4	建立自己的提供程序	508
15.4.1	创建 CustomProviders 应用程序	508
15.4.2	构造需要的类架构	509
15.4.3	创建 XML 用户数据 存储	512
15.4.4	在 web.config 文件中 定义提供程序实例	512
15.4.5	不实现 MembershipProvider 类的方法和属性	513
15.4.6	实现 MembershipProvider 类的方法和属性	514
15.4.7	使用 XmlMembership- Provider 进行用户登录	519
15.5	扩展预定义的提供程序	520
15.5.1	用新的 LimitedSql- RoleProvider 提供程序 限制角色功能	521
15.5.2	使用新的 LimitedSqlRoleProvider 提供程序	523
15.6	本章小结	526

第V部分 ASP.NET 特性

第 16 章	使用母版页	529
16.1	需要母版页的原因	529
16.2	母版页的基础知识	531
16.3	编写母版页	532
16.4	编写内容页面	534
16.4.1	混合页面类型和语言	538
16.4.2	指定要使用的母版页	539
16.4.3	将母版页应用于 页面子集	540
16.4.4	使用页面标题	540
16.4.5	使用母版页中的控件和 属性	540
16.5	在母版页中指定默认内容	545
16.6	以编程方式将母版页赋予 内容页面	547

16.7	母版页的嵌套	547	17.5.2	StartFromCurrentNode 属性	590
16.8	浏览器特定的母版页	551	17.5.3	StartingNodeOffset 属性	591
16.9	事件的触发顺序	552	17.5.4	StartingNodeUrl 属性	592
16.10	高速缓存母版页	553	17.6	SiteMap API	592
16.11	ASP.NET AJAX 和母版页	554	17.7	URL 映射	594
16.12	本章小结	556	17.8	站点地图的本地化	595
第 17 章	站点导航	557	17.8.1	为本地化构建 Web.sitemap 文件	595
17.1	基于 XML 的站点地图	557	17.8.2	修改 web.config 文件	596
17.2	SiteMapPath 服务器控件	559	17.8.3	创建程序集资源(.resx) 文件	596
17.2.1	PathSeparator 属性	561	17.8.4	测试结果	597
17.2.2	PathDirection 属性	563	17.9	安全补偿	598
17.2.3	ParentLevelsDisplayed 属性	563	17.9.1	启动管理员的角色管理 功能	598
17.2.4	ShowToolTips 属性	564	17.9.2	建立管理员的配置部分	600
17.2.5	SiteMapPath 控件的 子元素	564	17.9.3	启用安全补偿功能	601
17.3	TreeView 服务器控件	565	17.10	嵌套站点地图文件	602
17.3.1	标识 TreeView 控件的 内置样式	568	17.11	本章小结	604
17.3.2	研究 TreeView 控件的 各个部分	569	第 18 章	个性化	605
17.3.3	将 TreeView 控件绑定到 XML 文件	570	18.1	个性化模型	605
17.3.4	在 TreeView 中选择多个 选项	572	18.2	创建个性化属性	606
17.3.5	在 TreeView 控件中指定 定制的图标	575	18.2.1	添加简单的个性化属性	607
17.3.6	指定用于连接节点的线	576	18.2.2	使用个性化属性	607
17.3.7	以编程方式使用 TreeView 控件	577	18.2.3	添加一组个性化属性	610
17.4	Menu 服务器控件	582	18.2.4	使用分组的个性化属性	611
17.4.1	对 Menu 控件应用不同的 样式	583	18.2.5	为个性化属性定义类型	611
17.4.2	Menu 事件	587	18.2.6	使用定制的类型	612
17.4.3	把 Menu 控件绑定到 XML 文件	588	18.2.7	提供默认值	613
17.5	SiteMap 数据提供程序	589	18.2.8	把个性化属性指定为 只读	614
17.5.1	ShowStartingNode 属性	589	18.3	匿名个性化	614
			18.3.1	支持终端用户的 匿名身份	614
			18.3.2	使用匿名身份识别事件	617
			18.3.3	个性化属性的匿名选项	617

18.3.4	有关匿名用户配置信息 的警告.....	618	19.2.3	添加和检索应用程序 角色.....	667
18.4	编程访问个性化功能.....	618	19.2.4	删除角色.....	669
18.4.1	迁移匿名用户.....	619	19.2.5	给角色添加用户.....	669
18.4.2	配置信息的个性化.....	619	19.2.6	获取某个角色的 所有用户.....	670
18.4.3	确定是否继续自动保存 配置信息.....	620	19.2.7	获取包含某个用户的 所有角色.....	671
18.4.4	在 Web Application Projects 中使用配置信息.....	621	19.2.8	从角色中删除用户.....	672
18.5	个性化提供程序.....	622	19.2.9	检查角色中的用户.....	672
18.5.1	使用 SQL Server Express Edition.....	622	19.2.10	角色的高速缓存方式.....	673
18.5.2	使用 SQL Server.....	623	19.3	使用 SimpleMembership API.....	674
18.5.3	使用多个提供程序.....	625	19.4	使用 Web Site Administration Tool.....	675
18.5.4	使用通用提供程序.....	626	19.5	Membership API 的 公有方法.....	675
18.6	管理应用程序的配置信息.....	626	19.6	Roles API 的公有方法.....	676
18.6.1	ProfileManager 类的 属性.....	627	19.7	集成 OAuth 和 OpenID 身份验证.....	677
18.6.2	ProfileManager 类的 方法.....	627	19.7.1	使用 OpenID.....	677
18.6.3	建立配置信息管理页面.....	628	19.7.2	使用 OAuth.....	681
18.6.4	研究配置信息管理页面 的代码.....	630	19.8	本章小结.....	685
18.6.5	运行配置信息管理页面.....	631	第 20 章	安全性.....	687
18.7	本章小结.....	631	20.1	应用身份验证措施.....	688
第 19 章	成员资格和角色管理.....	633	20.1.1	<authentication>节点.....	688
19.1	ASP.NET 4.5 的身份验证.....	634	20.1.2	基于 Windows 的 身份验证.....	689
19.1.1	为成员资格建立 Web 站点.....	634	20.1.3	基于表单的身份验证.....	696
19.1.2	添加用户.....	637	20.2	验证特定的文件和文件夹.....	703
19.1.3	请求凭据.....	649	20.3	以编程方式授权.....	704
19.1.4	处理验证用户.....	655	20.3.1	使用 User.Identity 属性.....	705
19.1.5	显示在线用户数.....	657	20.3.2	使用 IsInRole 方法.....	706
19.1.6	处理密码.....	658	20.3.3	使用 WindowsIdentity 显示更多信息.....	706
19.2	ASP.NET 4.5 的授权.....	662	20.4	身份和模拟.....	708
19.2.1	使用 LoginView 服务器 控件.....	662	20.5	通过 IIS 进行保护.....	710
19.2.2	为角色管理建立 Web 站点.....	664	20.5.1	使用文件扩展名.....	710

20.5.2 使用 IIS 7.x/8 Manager	712	22.2.1 使用 Cache 对象高速缓存数据	754
20.5.3 使用 ASP.NET MMC 管理单元	713	22.2.2 控制 ASP.NET 高速缓存	754
20.6 本章小结	714	22.2.3 高速缓存依赖性	755
第VI部分 应用程序状态		22.2.4 .NET 4.x 中新的对象高速缓存选项	758
第 21 章 状态管理	717	22.3 使用 SQL Server 高速缓存依赖性	761
21.1 如何选择会话状态	717	22.3.1 使数据库支持 SQL Server 高速缓存禁用功能	762
21.2 理解 ASP.NET 中的 Session 对象	719	22.3.2 使表支持 SQL Server 高速缓存禁用功能	762
21.2.1 会话和事件模型	720	22.3.3 对 SQL Server 的影响	763
21.2.2 配置会话状态的管理	721	22.3.4 查看支持 SQL Server 高速缓存禁用功能的表	764
21.2.3 进程中的会话状态	721	22.3.5 使表不支持 SQL Server 高速缓存禁用功能	764
21.2.4 进程外的会话状态	727	22.3.6 使数据库不支持 SQL Server 高速缓存禁用功能	764
21.2.5 SQL 支持的会话状态	732	22.3.7 SQL Server 高速缓存禁用功能	765
21.2.6 使用其他提供程序扩展会话状态	734	22.4 配置 ASP.NET 应用程序	766
21.2.7 无 cookie 的会话状态	735	22.5 测试 SQL Server 高速缓存禁用功能	767
21.2.8 选择维护状态的正确方式	736	22.5.1 给页面添加多个表	769
21.3 Application 对象	737	22.5.2 给 SQL Server 高速缓存依赖性和 Request 对象建立关联	770
21.4 查询字符串	737	22.5.3 给 SQL Server 高速缓存依赖性和 Cache 对象建立关联	770
21.5 cookie	738	22.6 本章小结	773
21.6 回送和跨页面回送	738	第VII部分 客户端开发	
21.7 隐藏字段、ViewState 和 ControlState	740	第 23 章 ASP.NET AJAX	777
21.8 为短时间状态存储应用 HttpContext.Current.Items	743	23.1 理解对 AJAX 的需求	777
21.9 本章小结	744	23.1.1 AJAX 出现之前的请求/响应过程	777
第 22 章 高速缓存	745		
22.1 高速缓存	745		
22.1.1 输出高速缓存	746		
22.1.2 部分页面(UserControl)的高速缓存	749		
22.1.3 Post-Cache Substitution	750		
22.1.4 HttpCachePolicy 和客户端高速缓存	752		
22.2 使用编程方式进行高速缓存	753		

23.1.2	AJAX 改变了请求/响应过程	778	25.3	修改元素	882
23.2	ASP.NET AJAX 和 Visual Studio 2012	780	25.3.1	修改内容	882
23.2.1	客户端技术	781	25.3.2	添加和删除元素	884
23.2.2	服务器端技术	781	25.4	事件处理	886
23.2.3	使用 ASP.NET AJAX 进行开发	782	25.5	AJAX	888
23.3	创建 ASP.NET AJAX 应用程序	782	25.6	jQuery UI	895
23.3.1	建立没有使用 AJAX 的简单 ASP.NET 页面	784	25.7	本章小结	898
23.3.2	建立使用 AJAX 的简单 ASP.NET 页面	785	第 26 章	实时通信	899
23.4	ASP.NET AJAX 的服务器控件	790	26.1	传统的实时通信选项	899
23.4.1	ScriptManager 控件	791	26.1.1	使用 Comet	900
23.4.2	ScriptManagerProxy 控件	793	26.1.2	Polling	901
23.4.3	Timer 控件	794	26.1.3	服务器发送的事件	901
23.4.4	UpdatePanel 控件	795	26.1.4	现有方法的缺点	901
23.4.5	UpdateProgress 控件	799	26.2	HTML5 WebSockets	902
23.5	使用多个 UpdatePanel 控件	801	26.2.1	WebSockets 的概念	902
23.6	使用页面历史记录	804	26.2.2	TCP/IP	903
23.7	脚本合并	809	26.2.3	TCP/HTTP	903
23.8	本章小结	813	26.2.4	WebSockets 协议简介	904
第 24 章	AJAX 控件工具集	815	26.2.5	WebSockets 数据传输	905
24.1	下载和安装 AJAX 控件工具集	816	26.2.6	WebSockets API	906
24.2	ASP.NET AJAX 控件	818	26.2.7	ASP.NET 4.5 中的 WebSockets	907
24.2.1	AJAX 控件工具集的扩展程序	819	26.2.8	使用 WebSockets 的优点	910
24.2.2	AJAX 控件工具集中的服务器控件	863	26.3	SignalR	911
24.3	本章小结	873	26.3.1	什么是 SignalR?	911
第 25 章	jQuery	875	26.3.2	ASP.NET 中的服务器端 SignalR	912
25.1	jQuery 简介	876	26.3.3	ASP.NET 中的客户端 SignalR	913
25.2	选择元素	880	26.4	本章小结	914
			第 27 章	开发移动网站	915
			27.1	移动 Web 设计的挑战	916
			27.2	响应式设计和适应式设计	916
			27.2.1	修改视口	917
			27.2.2	使用 CSS 媒介查询	918
			27.3	ASP.NET 移动应用程序	920
			27.3.1	检测移动浏览器和设备	921

27.3.2	处理移动母版页	921	28.2.12	配置 ASP.NET 工作者 进程	963
27.3.3	创建移动 Web 窗体	923	28.2.13	存储与应用程序相关的 设置	966
27.3.4	ASP.NET Web 窗体中的 FriendlyURLs	924	28.2.14	对配置文件编程	966
27.4	ASP.NET MVC 4 移动 应用程序	925	28.2.15	保护配置设置	970
27.4.1	ASP.NET MVC 4 中的 适应式显示功能	925	28.2.16	编辑配置文件	973
27.4.2	创建移动专用的视图	928	28.3	创建定制部分	975
27.4.3	提供显示模式	929	28.3.1	使用 NameValueFile- SectionHandler 对象	975
27.4.4	包含 jQuery Mobile 和 ViewSwitcher	931	28.3.2	使用 Dictionary- SectionHandler 对象	976
27.4.5	使用 Mobile Application 项目模板	934	28.3.3	使用 SingleTag- SectionHandler 对象	977
27.5	测试移动应用程序	934	28.3.4	使用定制的配置处理 程序	977
27.6	本章小结	935	28.4	使用配置转换	979
第VIII部分 应用程序的配置与开发			28.4.1	添加 web.config 转换	979
第 28 章	配置	939	28.4.2	更新配置转换文件	981
28.1	配置概述	939	28.5	打包和压缩功能	983
28.1.1	服务器配置文件	940	28.5.1	什么是打包和压缩功能	983
28.1.2	应用程序配置文件	943	28.5.2	启用打包和压缩功能	984
28.1.3	应用配置设置	943	28.6	本章小结	985
28.1.4	检测配置文件发生的 改动	944	第 29 章	调试和错误处理技术	987
28.1.5	配置文件的格式	944	29.1	设计期间的支持	987
28.2	公共配置设置	945	29.1.1	语法通知	988
28.2.1	连接字符串	945	29.1.2	Immediate 和 Command 窗口	989
28.2.2	配置会话状态	946	29.1.3	任务列表	989
28.2.3	编译配置	950	29.2	跟踪	990
28.2.4	定制错误	952	29.2.1	System.Diagnostics.Trace 和 ASP.NET 的 Page.Trace	990
28.2.5	身份验证	952	29.2.2	页面级的跟踪	991
28.2.6	匿名身份	955	29.2.3	应用程序的跟踪	991
28.2.7	授权	956	29.2.4	查看跟踪数据	991
28.2.8	锁定配置设置	958	29.2.5	在组件中跟踪	994
28.2.9	ASP.NET 页面配置	958	29.2.6	跟踪的传送	995
28.2.10	包含文件	960	29.2.7	TraceListener	995
28.2.11	配置 ASP.NET 运行时 设置	961			

29.2.8	诊断选项	999	31.2.3	并行	1039
29.2.9	Web 事件	1000	31.2.4	服务器配置	1040
29.3	调试	1001	31.2.5	使用异步模式的缺点	1040
29.3.1	需要的内容	1002	31.3	本章小结	1041
29.3.2	启动调试会话	1003	第 32 章	国际化应用程序的建立	1043
29.3.3	有助于调试的工具	1004	32.1	区域性和地区	1043
29.3.4	使用 IntelliTrace 执行 历史调试	1007	32.1.1	了解区域性类型	1044
29.3.5	调试多个线程	1008	32.1.2	ASP.NET 线程	1045
29.3.6	客户端的 JavaScript 调试	1008	32.1.3	服务器端的区域性 声明	1046
29.3.7	SQL 存储过程的调试	1009	32.1.4	客户端的区域性声明	1048
29.4	异常和错误处理	1010	32.1.5	翻译值和行为	1049
29.4.1	处理页面上的异常	1010	32.2	ASP.NET 4.5 资源文件	1056
29.4.2	处理应用程序异常	1011	32.2.1	使用本地资源	1056
29.4.3	HTTP 状态码	1011	32.2.2	使用全局资源	1061
29.5	用 Page Inspector 进行调试	1012	32.3	本章小结	1064
29.6	本章小结	1015	第 33 章	打包和部署 ASP.NET 应用程序	1065
第 30 章	模块和处理程序	1017	33.1	部署各个部分	1066
30.1	处理 HTTP 请求	1017	33.2	部署之前的准备步骤	1066
30.1.1	IIS 6 和 ASP.NET	1017	33.3	部署 Web 应用程序的方法	1067
30.1.2	IIS 7、IIS 8 和 ASP.NET	1018	33.3.1	使用 XCopy	1067
30.1.3	ASP.NET 请求处理	1019	33.3.2	使用 Copy Web Site 选项	1069
30.2	HttpModule	1019	33.3.3	部署预编译的 Web 应用程序	1072
30.3	HttpHandler	1024	33.3.4	创建 ASP.NET Web Package	1073
30.3.1	一般的处理程序	1024	33.3.5	发布配置文件详解	1076
30.3.2	在 IIS 中映射文件 扩展名	1028	33.4	部署 Windows Azure Web Sites	1082
30.4	本章小结	1029	33.5	本章小结	1084
第 31 章	异步通信	1031	第 IX 部分	其他 ASP.NET 技术	
31.1	异步编程	1031	第 34 章	ASP.NET MVC	1087
31.1.1	使用异步方式的原因	1031	34.1	MVC 的定义	1087
31.1.2	编写异步代码的场合	1032	34.2	当今 Web 上的 MVC	1088
31.1.3	异步简史	1032	34.3	MVC 和 ASP.NET	1089
31.2	ASP.NET 中的异步	1034			
31.2.1	线程池	1035			
31.2.2	编写异步代码	1036			

34.3.1	为方法而不是 文件服务	1089	35.4.1	核心辅助方法	1127
34.3.2	ASP.NET MVC 是 Web Forms 4.5?	1089	35.4.2	使用辅助方法添加 功能	1130
34.3.3	为什么不是 Web 窗体 ...	1090	35.4.3	创建定制的辅助方法 ...	1132
34.3.4	ASP.NET MVC 是完全 不同的	1090	35.5	本章小结	1133
34.3.5	为什么“(ASP.NET>ASP.NET MVC)=True”	1090	第 X 部分 附 录		
34.3.6	约定胜于配置	1092	附录 A	迁移 ASP.NET 旧项目	1137
34.3.7	第三个请求是 Charm ...	1095	附录 B	COM 集成	1145
34.4	理解路由和 URL	1097	附录 C	ASP.NET 终极工具	1155
34.4.1	路由选择与 URL 重写 的比较	1097	附录 D	管理	1165
34.4.2	路由的定义	1098	附录 E	动态类型与语言	1189
34.5	控制器	1103	附录 F	ASP.NET 联机资源	1197
34.5.1	控制器的定义: IController 接口	1103	附录 G	使用 NuGet 扩展 Visual Studio	1201
34.5.2	控制器类和操作	1104			
34.5.3	处理参数	1105			
34.5.4	处理多个参数	1106			
34.6	视图	1106			
34.6.1	指定视图	1108			
34.6.2	ASP.NET MVC 布局 ...	1109			
34.6.3	强类型化视图	1112			
34.6.4	使用 HTML 辅助方法 ...	1113			
34.6.5	HtmlHelper 类和 扩展方法	1113			
34.7	本章小结	1114			
第 35 章 ASP.NET Web Pages 和 Razor			1115		
35.1	ASP.NET Web Pages 概述 ...	1116			
35.2	使用 Razor 创建 HTML 窗体	1116			
35.3	显示数据	1120			
35.3.1	验证	1123			
35.3.2	使用布局	1125			
35.4	使用辅助方法	1127			

第 I 部分

ASP.NET 基础

- 第 1 章 One ASP.NET
- 第 2 章 使用 ASP.NET 进行 HTML5 和 CSS3 设计
- 第 3 章 ASP.NET Web 窗体的结构

第 1 章

One ASP.NET

本章要点

- One ASP.NET 简介
- 研究并简化复杂的 Web 生态系统
- 从 One ASP.NET 中获益

多年来，ASP.NET 有个杰出的 Web 开发架构，一直在支持 .NET 开发人员创建 Web 应用程序。随着互联网的成熟和进步，ASP.NET 架构和围绕它的生态系统也在走向成熟和进步。成千上万个产品、服务和开源项目都把 ASP.NET 当作它们的家。

互联网的前进步伐比 ASP.NET 快得多。HTML、CSS 和 JavaScript 的发展给用户提供了更加丰富的体验，迫使 ASP.NET 支持这些新兴技术。ASP.NET 架构必须支持 Web 开发人员不断增长的需求。于是 ASP.NET 开发了一个健康的生态系统，所有的 Web 开发人员都可以聚集在这里，使 ASP.NET 变得更好，而这也说明 ASP.NET 的核心部分已经有了相当长的历史。

本章介绍 ASP.NET 架构的成长方式，并提出一些方法，用该生态系统来解决问题。ASP.NET 如何成为值得信任的、可靠的架构，使得企业开发人员可以依赖，同时仍能满足高级 Web 开发人员对最新标准的需求？本章的目标是介绍 ASP.NET 的一些变化，以及该架构的目标。

1.1 One ASP.NET 简介

几年前刚刚推出 ASP.NET MVC 时，许多人就告诉我，他们对创建“混合型”应用程序很失望。他们希望 ASP.NET 应用程序带有服务组件、MVC 区域和 Web 窗体部分。一些 Microsoft 员工让他们觉得，Microsoft 不会支持这种应用程序。“为什么你们希望这样？这可不是个好想法。”

但其实这个想法很好。ASP.NET MVC 是 ASP.NET，Web 窗体也是 ASP.NET。10 余年前的流水线系统今天成为现实，提供了许多可扩展点，这些可扩展点不仅 ASP.NET 可以利用，NancyFx 和 ServiceStack 等其他架构也可以利用。

为什么要让开发人员从有相同底层菜单的 6 个不同菜单项中选择？无论应用程序体系结构是什

么，ASP.NET 都提供了需要的核心服务。密钥管理、会话管理、缓存、HTTP、身份验证和授权都是通用的 Web 开发项目。选择如何显示尖括号或花括号是用户的喜好问题，但用户应明白需使用什么 ASP.NET 架构才有助于解决商务问题。用户这么做时，不仅应没有任何负罪感，还应获得完全的支持和鼓励。图 1-1 显示了 ASP.NET 中的不同架构，它们都使用相同的底层 ASP.NET 核心。

站点				服务	
Web 表单	Web Pages	单页面应用程序	MVC	Web API	SignalR
ASP.NET					

图 1-1

One ASP.NET 的目标是使开发人员通过 ASP.NET 和 Visual Studio 创建应用程序更容易。One ASP.NET 是泛称，可以表示许多含义。下面就解释一下。

在现今世界，创建 Web 应用程序有许多选择。每种选择都有其优缺点。开始创建应用程序时，该如何选择架构？

如果选择了一个架构，接着发现这个架构不太好，回过头来选择另一条路有多难？该架构的模块化部分足以换掉有问题的部分吗？还是必须重写所有的内容？代码能在子系统之间共享吗？

架构更新的频率有多快？互联网的发展速度要远远快于大多数架构可以处理的程度，ASP.NET 如何服务两个主人？现在用户希望使用 HTML5 和 CSS3，但不能破坏业务所依赖的应用程序。

One ASP.NET 应让开发人员相信，无论他们选择什么架构，都是在受信任的底层架构——ASP.NET 上进行开发。为了帮助理解 One ASP.NET，下一节将介绍 ASP.NET 的演变史。

1.1.1 ASP.NET 的演变史

Microsoft 的 Web 开发始于 20 世纪 90 年代后期的经典 ASP (Active Server Pages)。开发人员可以根据 VB 脚本语言编写动态页面。

ASP.NET 1.0 于 2002 年推出，建立在 .NET Framework 基础之上。ASP.NET Web 窗体简化了开发人员从 Windows 开发到 Web 开发的转变，允许他们创建包含控件的页面，类似于 Windows 用户界面。它与我们以前看到的都不一样！Visual Basic 开发人员习惯于把数据网格拖放到 Windows 窗体中，现在可以把数据网格拖放到 Web 窗体上了，真神奇！

这是因为它把状态放在无状态的 HTTP 顶部，允许开发人员响应事务级的事件，例如 Button.Click，而不是低级事件，例如 HTTP POST，因此新一代开发人员可以创建强大的 Web 应用程序。

ASP.NET 2.0 与 Visual Studio 2005 在 2005 年一起发布，这个版本进一步加强了控件和集中于数据的改进，例如 SqlDataSource 和 ObjectDataSource。接着 ASP.NET 3.5 在 2008 年推出，带来了 DynamicData，它允许在几分钟内快速生成数据驱动的应用程序。

随后不久就发布了 ASP.NET MVC，第一次在 ASP.NET Web 开发中引入了非常受欢迎的 MVC(Model-View-Controller)模式。ASP.NET MVC 仍旧基于 ASP.NET，提供了当时在 .NET 平台上还没有的可测试性和布局层。开发 ASP.NET MVC 是为了满足开发团体日益增长的如下需求：更容易地对应用程序进行单元测试，更详细地控制 HTML 标记生成的方式。ASP.NET MVC 表示一种绝对控制的级别，能让新一代的 .NET Framework 开发人员(也许还包括某些老的开发人员继续)使用 ASP.NET。

MVC有几个有趣的地方。ASP.NET MVC是ASP.NET框架中第一个在带外(Out Of Band, OOB)发布的产品,即ASP.NET MVC被发布为独立的安装程序,开发人员可以选择下载并添加到它们到已有的Visual Studio安装中。这是巨大的进步,开发人员无须等上2、3年,就可以快速获得新产品。ASP.NET MVC的带外发布标志着,ASP.NET团队开始与Visual Studio团队分道扬镳了,但ASP.NET团队并没有意识到这一点。

过了两年,微软又发布了ASP.NET MVC的两个版本。更重要的是,ASP.NET MVC在Microsoft Public License (MS-PL)下发布,允许开发人员看到他们喜欢的框架的源代码。传言说,也许ASP.NET MVC会发布为正式的开源产品,但Microsoft从来没有这样做。

ASP.NET 4与VS 2010一起发布,同时发布了ASP.NET MVC 4,引入了ASP.NET Web Pages和创建动态页面的全新语法(称为Razor)。ASP.NET Web Pages和新的Razor语法提供了一种快速、可获得的轻型方式,以合并服务器代码和HTML,创建动态的Web内容。Razor成为ASP.NET MVC新的视图引擎。ASP.NET Web Pages将Razor用于一种简单的新编程模型,该模型适用于小型站点或新手。

ASP.NET 4.5与VS 2012在2012年一起发布。这个版本包括ASP.NET MVC 4.5,另外还给ASP.NET家族带来了新成员ASP.NET Web API,以便于编写基于REST的Web服务。

不久,Microsoft在开源许可(Apache License 2.0)下发布了Web Stack的主要内容(包括ASP.NET MVC、Razor和ASP.NET Web API),并宣布Web Stack来自于社团的贡献——是战略上非常重要的旗舰Web框架上的真正开源产品。这艘大船开始转向了。

这就是“开放性开发”的开始。对此感兴趣的社区成员可以切实阅读到已登记的代码,同时开发人员在Microsoft内部工作。更妙的是,贡献代码的人可以修改错误,添加功能,提交请求,了解他们的代码是否可由数以百万计的开发人员使用。

开放性开发最初令人惊慌,但这么做可建立更开放的开发模型,每个人都可以参与其中,对已登记的代码提供反馈,修改错误,开发新功能,每天使用最新的源代码版本和测试程序来创建并测试产品。这是使ASP.NET生态系统汇集在一起的一项巨大进步,从而开发人员可以协同工作,建立更好的框架。

此时,ASP.NET的很大组成部分及其运行库组件在带外发布为开源产品。许多功能都是可选的,或者很容易添加,例如Web Optimization、Universal Membership Providers和测试套装。如何开放这些工具以支持更统一的模块化ASP.NET?

1.1.2 Web的演化史

HTML5还没有完成,除非本书在2017年出版,否则读者阅读本书时,HTML5就仍没有完成。但是,HTML5、CSS3和JavaScript已经在互联网上获得人们的青睐。在每个现代浏览器上都能获得JavaScript的可靠、完整版本,再与紧密结合的文档对象模型(DOM)合并,应用程序就会显著改变原有的体系结构。

JavaScript删除了简单的alert()语句和输入类型验证,逐渐成为浏览器上一种接近完整的虚拟机。HTML5作为一种规范是非常简单的,CSS3使之引人注目。把所有这些要素聚集起来,再与便携超级计算机结合起来,就带来了一场可移式Web革命。

目前,顾客在使用不同形式的产品,包括智能手机、平板电脑、个人电脑。这表示他们可以使用手机、平板电脑、可移式浏览器或传统的桌面浏览器来访问Web应用程序。Web也变得越来越

众化。用户通过 Twitter、Facebook 等社交方式彼此交互，他们希望在所交互的所有 Web 应用程序中带着自己的社交身份。这意味着他们希望使用社交凭证登录，这样在登录到应用程序中后，他们就可以与朋友交互。这是一个为 Web 开发人员提供互联设备和服务的世界。Web 变得越来越普遍，Web 标准一直在演变，所以 ASP.NET 框架一定要确定更新频率，以跟上 Web 开发的发展步伐。

1.2 简化复杂的生态系统

File | New Project 太可怕了，它迫使我们选择一个人为的选项，使开发人员觉得他们被推上了一条不可逆道路上的岔道。也许，不应让道路上的岔道迫使人们选择某个不自然的选项，而是考虑 ASP.NET 的统一标准，允许选择可能的子系统、库或其他应用程序框架。

这样我们就可以做出选择。现在，如何发现并整合库，以方便地在应用程序中编写服务？如果选择安全的选项 Empty Web Application，会怎样？这会便于添加服务吗？如果选择 ASP.NET Web Application，如何给这种应用程序添加 Dynamic Data 功能？

社区方式呢？可以引入社区支持的库，但能很方便地将之整合到 MVC 和 Web 窗体中吗？从开发人员的角度来看，这是个相当复杂的过程，不仅要指出最佳的起步方式，还要确定如何扩展应用程序。图 1-2 显示了创建新项目时的选项。

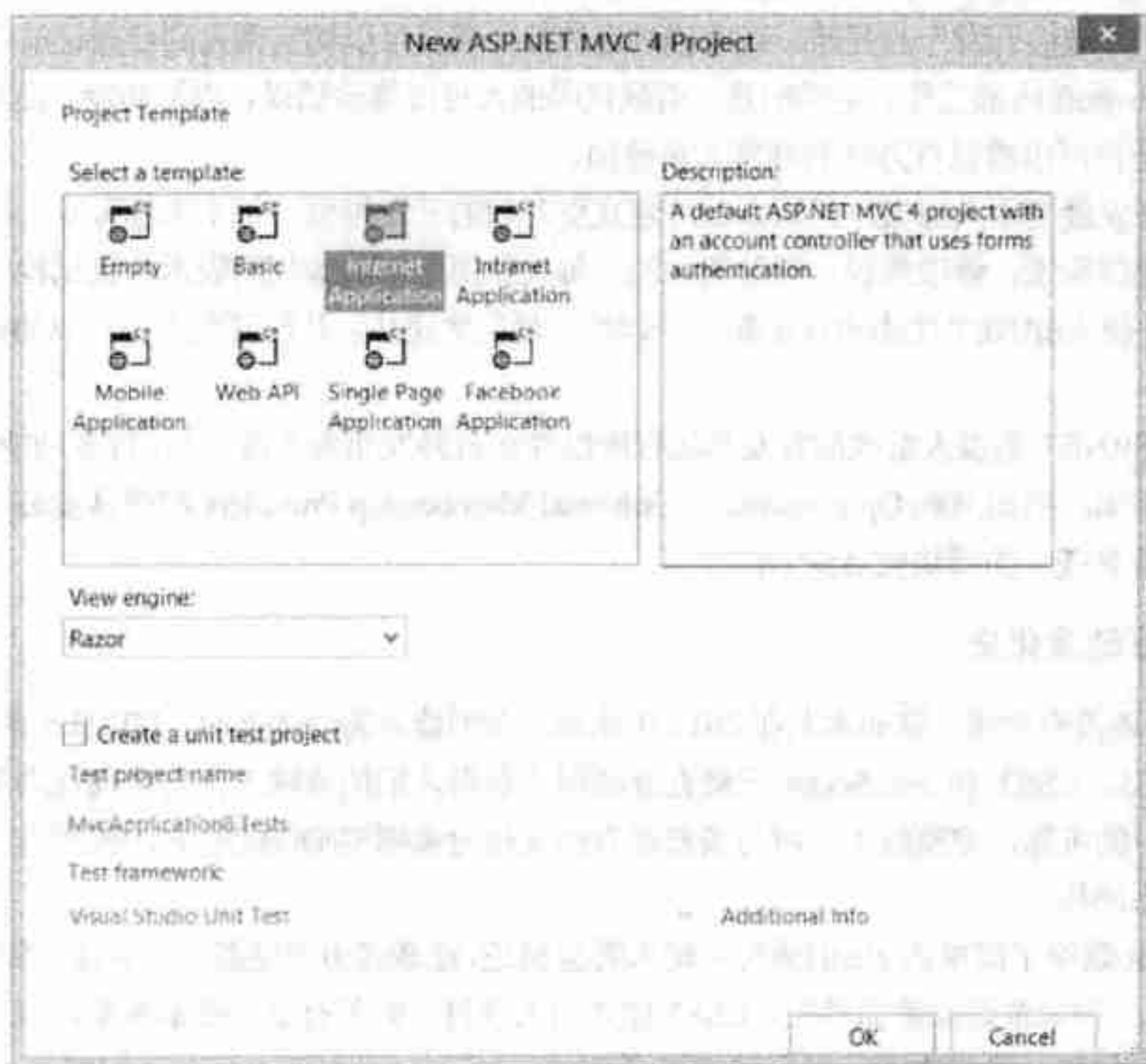


图 1-2

1.2.1 Web 生态系统

Web 开发是个令人激动的复杂领域。有许多因素影响 Web 开发人员的效率。Web 开发人员必须总是精通最新、最重要的 Web 标准，除了了解最新的标准，还必须考虑自己使用的框架和工具是否支持这些标准。还要知道，获得要在应用程序中使用的 FX、工具和库会有多难。

此开发体验的主要部分是不断成长的生态系统。Web 在不断地扩展和成长，有助于开发应用程序的库也呈爆炸性增长。根据要解决的问题，这些库将会变得很流行，被开发社区广泛接受。例如，jQuery 等较通用的库就非常流行，jQuery 是许多使用 JavaScript 的开发人员首选的库。另外，许多专用库可以解决可能不适用于大多数开发人员的独特问题。

有许多这样的库可用，例如 Entity Framework、JSON.NET、ELMAH 等可以在应用程序中使用的开源库。应用程序的主要部分现在由这些开源库和来自 Microsoft 的库组成。所以问题是：在这个正在成长的生态系统中，如何找到这些库，把它们引入应用程序？另外，一旦有了这些库，如何确保它们总是更新为最新版本？图 1-3 显示了带有最流行的包的 NuGet 工具集。这些包中的每个都被下载了超过一百万次，编写本书时，NuGet 工具集中的包的总下载次数已超过 5 千万。



图 1-3

近年来，Web 开发肯定在演变，成为 Web 开发人员是件很令人兴奋的事。Web 开发人员不再需要花时间给应用程序添加基本构建块了，登录等功能非常常见，所以开发人员可以在应用程序中重用许多登录库。很容易先从较小的项目开始，再把这些所谓的 Lego 部分添加到应用程序中，使工作更快完成。现在我们可以重点关注编写应用程序的核心部分，再在需要时添加这些不同的构建块。

下面介绍 ASP.NET 框架和 Visual Studio 工具集如何帮助解决这些问题并提高效率。

1.2.2 起步是很简单的

在 Visual Studio 2012 中选择 File ⇨ New Project 时, 可创建的项目类型有许多种。可以选择创建 Single Page Application (SPA), 或者让应用程序使用 Social Login, 或者让应用程序使用 Windows Authentication。当希望创建的应用程序混合这些情形时, 选择项目类型就成了问题。在 Visual Studio 内置的选项中开始创建应用程序没有简单的方法。模板仅表示一系列静态的选项, 但使用 One ASP.NET 模板会使事情变得很简单, 如图 1-4 所示。

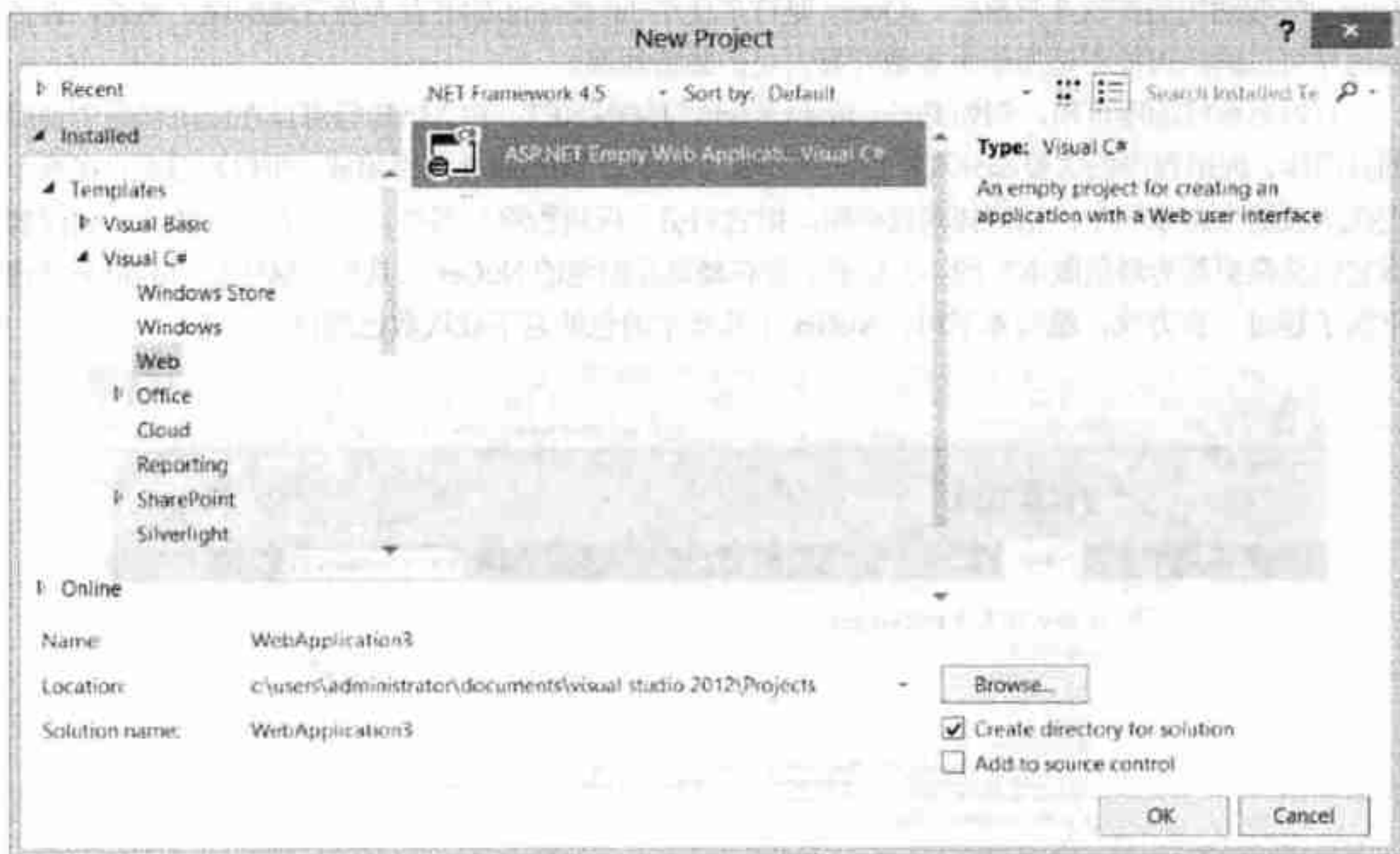


图 1-4



在编写本书时, 这只是个概念。

一旦创建 ASP.NET 项目, Visual Studio 就可以列出一系列步骤, 帮助我们 from 所选的 SPA 框架开始, 允许我们选择如何让用户登录到应用程序。

下一节介绍一旦开始这个应用程序, 如何获得更多的 Lego 部件, 或者更新已有的部件。

1.2.3 整合生态系统是很简单的

上一节介绍了如何创建 SPA 应用程序, 它带有所有需要的库, 例如 Knockout、jQuery、ASP.NET Web API 等。在 Visual Studio 2012 中, 所有的库都安装为 NuGet 包。右击项目, 再单击 Manage NuGet Packages, 就可以看到项目中安装了什么库。图 1-5 显示了这个选项。

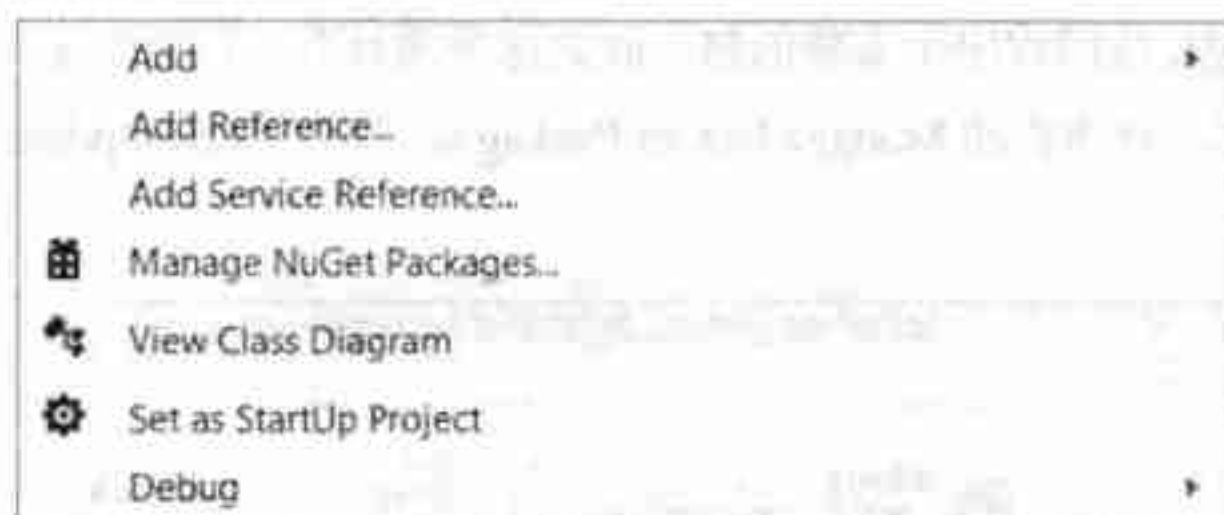


图 1-5

这会打开 Manage NuGet Packages 窗口，在其中可以选择 Installed packages 选项卡，查看项目中已安装的所有包。图 1-6 显示了项目中安装的所有包。



图 1-6

这个窗口显示了创建应用程序时安装的一系列包。该应用程序包含 Microsoft 拥有的包和开源库。本节讨论 NuGet 如何帮助将该生态系统整合到项目中。

NuGet 是 Visual Studio 的扩展，用于在 Visual Studio 项目中添加、删除和更新库及工具。NuGet 在 Visual Studio 2010 和 Visual Studio 2012 中工作，可以从 Visual Studio Extension Gallery 下载。如果所开发的库要与其他开发人员共享，就可以创建 NuGet 包，上传给 NuGet 工具集。如果希望使用其他人开发的库或工具，就可以从 NuGet 工具集中下载该包，安装到自己的 Visual Studio 项目中。简言之，NuGet 正在改变 .NET 开发人员的生态系统，很容易对包重新分布并安装到项目中。

假定需要在应用程序中安装 ELMAH。如果没有 NuGet，就必须下载安装程序，手动添加对安装位置上 ELMAH 库的引用。另外，还必须阅读 ELMAH 网站上的文档，对 web.config 进行所有必需的修改以配置 ELMAH。如果在开发过程中 ELMAH 的新版本发布，将无法确定自己项目中的 ELMAH 版本是否过时，是否需要下载新版本。

NuGet 使查找、安装、配置和更新库的整个过程变得非常容易。它是一站式解决方案，可以解决所有这些问题。

图 1-6 显示了在 SPA 应用程序中安装的包。假定这个项目有一段时间了，希望检查所使用的所有库的更新版本。为此，只须启动 Manage NuGet Packages 窗口，单击 Updates 选项卡。图 1-7 显示了包可用的更新版本。



图 1-7

对要更新的包单击 Update，NuGet 就会下载该包的新版本并更新引用。很容易就快速更新任何包。



在 Visual Studio 2012 中创建新项目时，新项目就已经安装了 NuGet 包，所以很容易获得这些好处。

1.2.4 真实示例

本节介绍如何在真实的应用程序中应用 One ASP.NET。下面的方法并不是什么最佳实践方式，但演示了 One ASP.NET 和 Web 生态系统如何整合在一起，帮助创建 ASP.NET 应用程序。假定希望创建一个简单的应用程序，它只有一个页面，其中包含 to-do 列表，要求如下：

- 用户应使用 Facebook、Twitter 登录，或者在网站上注册账户。
- 用户应创建 to-do 项，并在完成时做标记。
- 管理员应通过管理部分管理用户账户。

首先应选择 ASP.NET SPA 模板，把模板配置为使用 Facebook、Twitter 和 Local 登录。准备好这个模板后，就可以开始创建应用程序了。

先实现 ASP.NET Web API，以便有基于 REST 的服务来管理 to-do 项。右击项目，添加新的项模板 ASP.NET Web API Controller Class，添加 Web API，如图 1-8 所示。

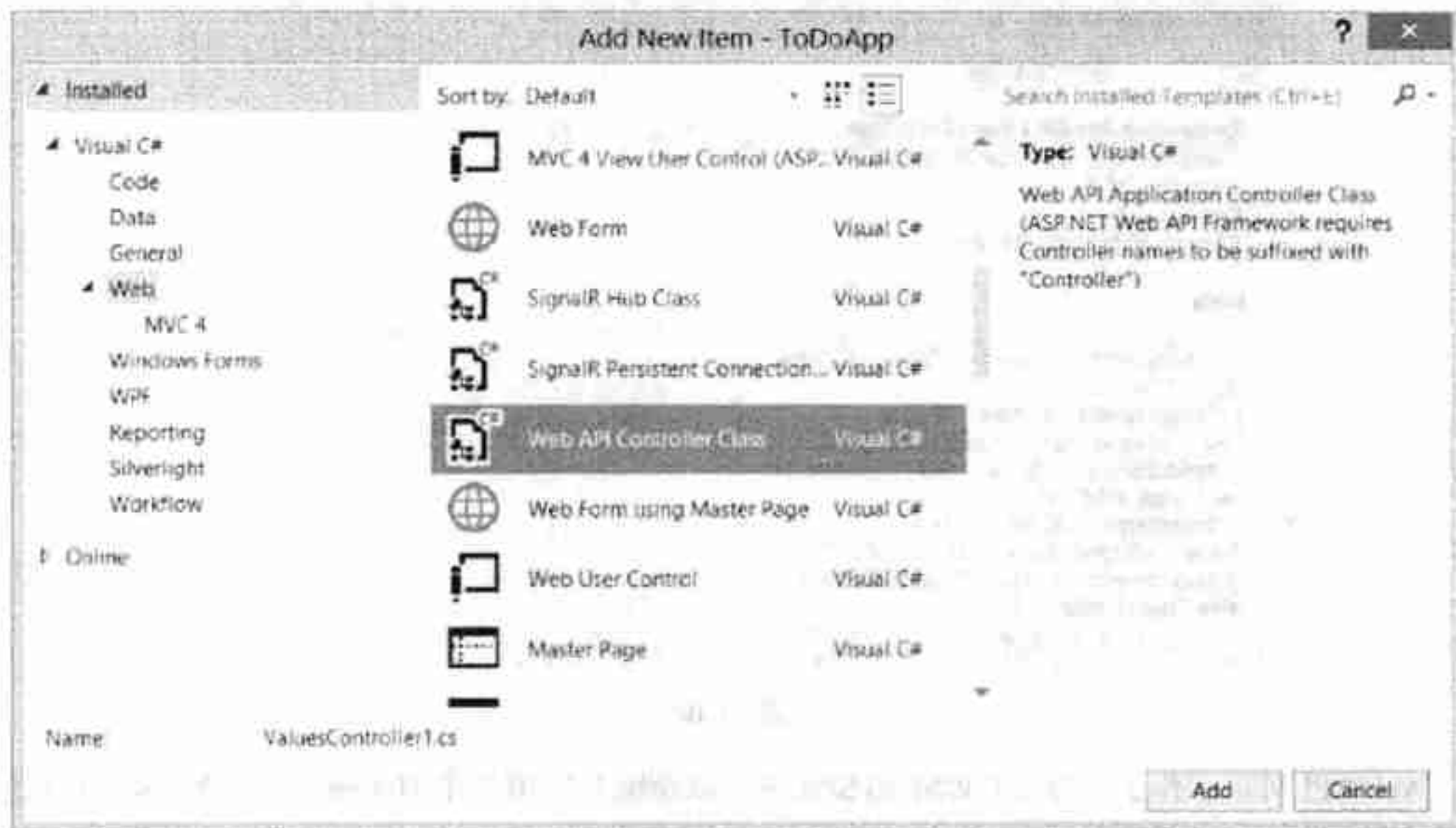


图 1-8

安装这个项模板时，会安装在 ASP.NET 中运行 Web API 所需的所有 NuGet 包。

编写一些代码，实现应用程序的 to-do 功能，包括添加、删除、修改和获得所有 to-do 项。图 1-9 显示了检索 to-do 列表的代码片段。

```
// GET api/TodoList
public IEnumerable<TodoListDto> GetTodoLists()
{
    return db.TodoLists.Include("Todos")
        .Where(u => u.UserId == User.Identity.Name)
        .OrderByDescending(u => u.TodoListId)
        .AsEnumerable()
        .Select(todoList => new TodoListDto(todoList));
}
```

图 1-9

编写这个 To do Web API 时，我们希望测试该功能，但不要运行整个应用程序，因为运行整个应用程序很快会变得非常无聊。使用 NuGet 可以搜索一些测试客户端来测试 To do Web API。此快速搜索可能会返回一系列包，可以根据它们的等级从中选择一个。例如，可以下载 NuGet 包 WebApiTestClient，在编写本章时它很流行。图 1-10 显示了使用这个测试客户端测试 Web API 时的结果。

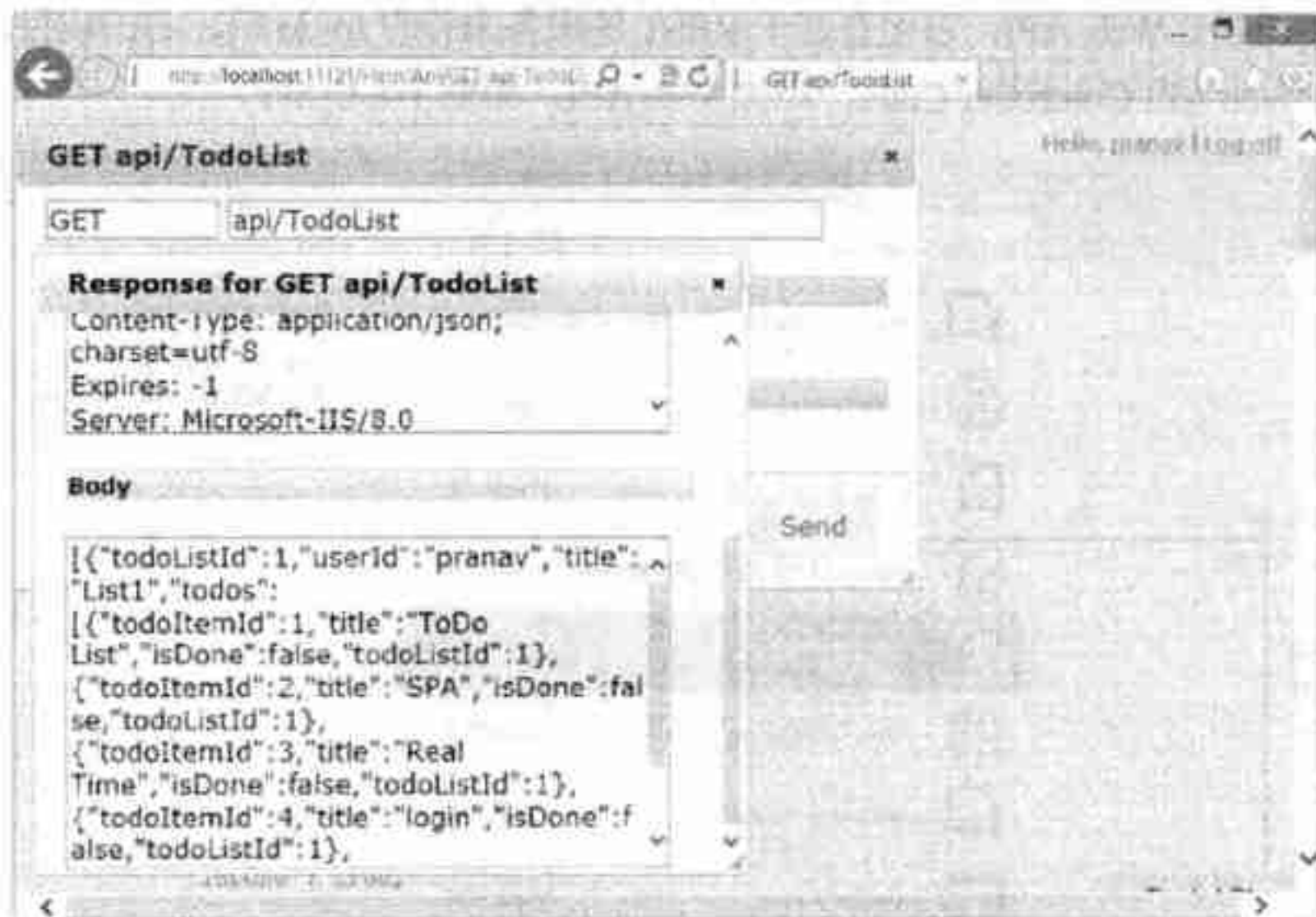


图 1-10

Web API 可以工作了，可以开始添加 SPA 库，以便编写应用程序的前端。使用 NuGet 也可以搜索需要的 SPA 库。对于这个例子，选择 Knockout。安装了这个 NuGet 包后，它就会引入使用 Knockout 所需的所有库。

安装了 Knockout 后，就可以创建 Web 页面，在其中调用 To do Web API，把结果绑定到 Knockout 视图中。Visual Studio 编辑器为 Knockout 语法突出显示功能和 IntelliSense 提供了丰富的支持，更容易使用 Knockout。图 1-11 显示了 Visual Studio 中对 Knockout 语法突出显示功能的支持。

```
<section id="list"
data-bind="foreach: todos, visible: todos().length > 0">
  <article class="todo-item">
    <ul data-bind="foreach: todos">
      <li>
        <input type="checkbox" data-bind="checked: isDone" />
        <input class="todo-item-input" type="text"
          data-bind="value: title, disabled: isDone" />
        <a href="#"
          data-bind="click: $parent.deleteTodo">X</a>
        <p class="error"
          data-bind="visible: errorMessage, text: errorMessage">
        </p>
      </li>
    </ul>
  </article>
</section>
```

图 1-11

使客户端数据绑定能使用 Knockout 后，就有了一个功能完整的 SPA 应用程序。如果希望检索 to-do 项的更新版本，可以安装 ASP.NET SignalR，获得实时更新功能。为此，应下载 NuGet 包 Microsoft.AspNet.SignalR。安装了这个包后，就可以修改应用程序代码，添加实时功能。图 1-12 显示了刚才开发的 to-do 应用程序。

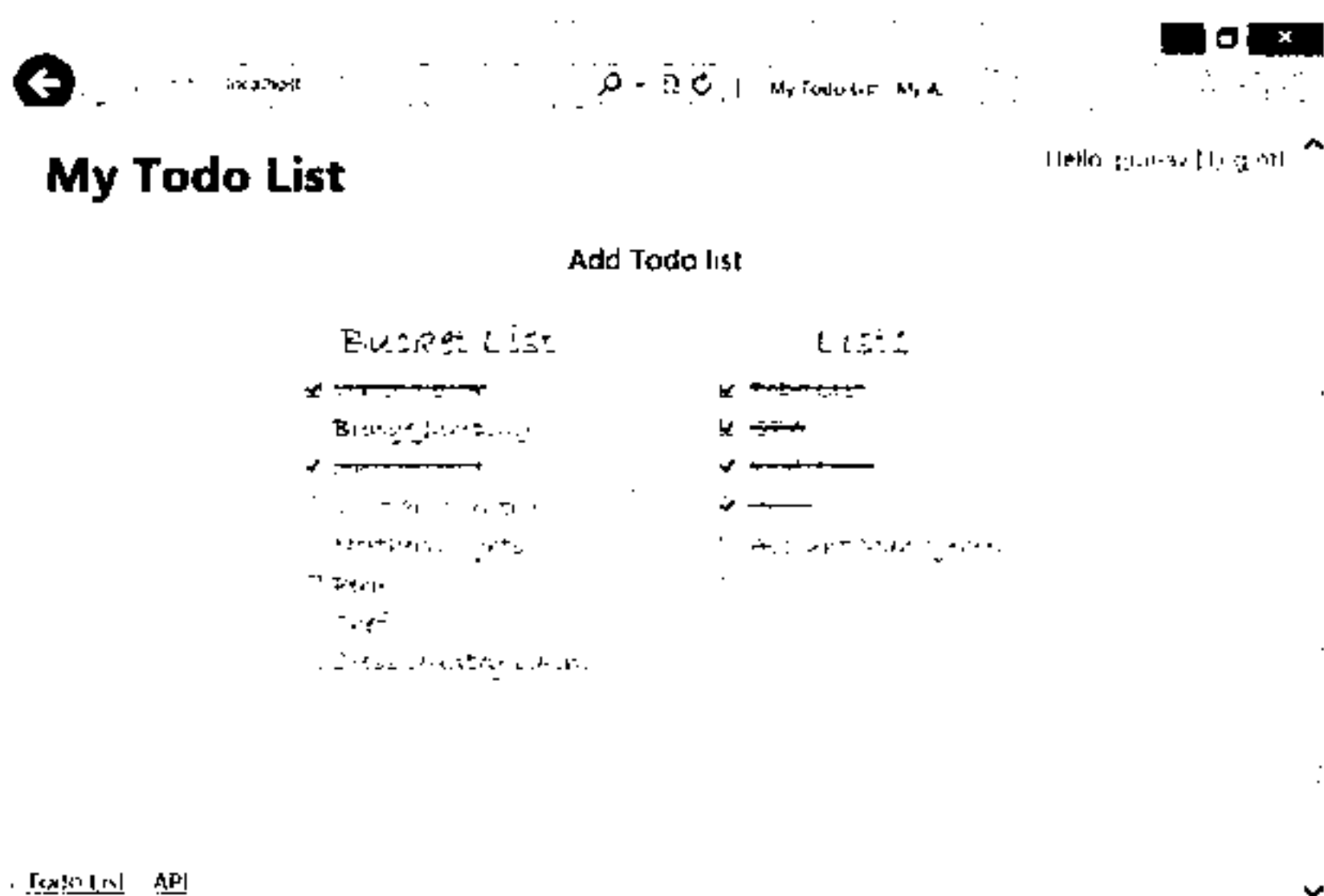


图 1-12

应用程序创建过程的最后阶段是为管理员建立一些用户账户管理功能。使用 ASP.NET Web 窗体和动态数据能很快实现该功能，因为这个框架可以快速创建数据驱动的应用程序。还可以通过 ASP.NET 授权功能，利用角色来保护这个部分，这样就只有管理员能访问网站的这个部分。

最后查看该应用程序，它是 ASP.NET 框架和 ASP.NET Web 窗体、MVC、Web API、SignalR 等其他技术的混合体，这些技术与 Knockout、jQuery、JSON.NET 等开源库一起工作。即使有了这些不同的框架和库，我们也仍旧是在可信的底层框架 ASP.NET 上进行开发。

1.3 如何从 One ASP.NET 中获益

阅读完前面的内容，读者一定会觉得这些都非常好，但这对于开发人员意味着什么？如何从 One ASP.NET 中获益？本节介绍 One ASP.NET 帮助提高开发效率的一些场景，还会讨论目前开发人员面临的常见问题，以及 ASP.NET 和 Visual Studio 如何帮助解决这些问题。

1.3.1 ASP.NET 使起步很简单

有了 ASP.NET 和 Visual Studio，就可以：

- 从空白的项目开始，添加框架和库中需要的部分以创建应用程序。
- 选择创建应用程序所需要的框架和库。

One ASP.NET 使 ASP.NET 框架更模块化，这样就可以开始应用程序，并很容易添加需要的部分以创建应用程序。

1.3.2 ASP.NET 支持 Web 生态系统

我们现在都知道，未来的 Web 应用程序在用户体验上会丰富得多。另外，随着 .NET 开发社区的成长，开发人员创建和发布的库呈爆炸式增长。在 .NET 生态系统中，开发人员现在越来越依赖开

源库，并为这些库作出贡献，使这些库变得更好。

Microsoft Web Stack 的主要部分现在是开源的。ASP.NET MVC、ASP.NET Web Pages、ASP.NET Web API、ASP.NET SignalR 和 Entity Framework 等产品是开源的。这说明，不为 Microsoft 工作的开发人员可以为这些项目作出贡献。



即使这些产品是开源的，Microsoft 也继续全面支持和参与它们。

除了作出贡献之外，还可以浏览源代码，订阅登记通知，浏览这些产品的开发人员提出的问题活动列表。这样，开发人员就会更直接地影响这些产品中正在设计和开发的功能，最终开发出更好的产品。

1.3.3 ASP.NET 更容易查找、添加、更新站点的各个模块

尽管正在成长的生态系统是 Web 开发社区的重要信号，但目前在该生态系统中发现和安装所有这些优秀的库仍是问题。这些库的安装过程主要是下载安装程序，再通过安装步骤来运行。这使添加、配置和更新项目中的库变得非常困难。而 NuGet 是解决这个问题的一大进步。库/框架可以重新发布为 NuGet 包，而 NuGet 包很容易通过 Visual Studio 安装到项目中。

1.3.4 ASP.NET 帮助把一个框架的概念应用于另一个框架

ASP.NET 这把大伞包含在一个框架下开发的许多不同功能，例如 ASP.NET MVC，并且还使这些功能可以进入其他框架，例如 ASP.NET Web Forms，反之亦然。例如：

- 在 ASP.NET Dynamic Data 中引入 Data Annotation Attributes 功能，后来该功能又被引入 ASP.NET MVC。
- 路由和模型绑定是 ASP.NET MVC 的两个亮点，它们最终被引入 ASP.NET Web Forms。
- ASP.NET Web Pages 有个简单的路由模型 Smarty Routes，它允许生成更简洁的 URL，且不需要任何配置。该功能已被引入 ASP.NET Web Forms，现在称为 ASP.NET FriendlyURLs。

这意味着 ASP.NET 开发人员可以使用这些著名的概念或功能，把它们应用于自己选择的任何框架。如果某个 Web 窗体开发人员喜欢模型绑定，现在就可以继续创建 Web 窗体应用程序，同时使用新的、著名的功能，例如模型绑定。不必切换到另一个框架中，例如 ASP.NET MVC，仅仅是因为 ASP.NET MVC 支持模型绑定。

1.3.5 ASP.NET 与 Web 发展得一样快

Web 的发展速度令人惊讶，这是由许多因素造成的。设备和服务的激增，催生了互联越来越紧密的新类型应用程序。Web 变得越来越大众化，用户希望可以彼此实时协作。为了支持所有这些变化，Web 标准联盟必须确保 HTML、JavaScript 和 CSS 的规范快速更新以满足这些要求。由于出现了所有这些变化，开发人员也必须联起手来协同工作，并将他们使用的框架和工具更新为最新标准。

ASP.NET 和 Visual Studio 也在响应这些变化。ASP.NET 2.0 发布时，是与 VS 2005 一起发布的，而且是在 ASP.NET 1.0 发布的 3 年后才发布。这表示开发人员必须等 3 年，才能获得支持最新 Web 标准的最新框架。现在我们进入了一个新世界，ASP.NET 的某些部分可以更频繁地发布，更容易更

新，而无须等待 Visual Studio 的下一个版本。

ASP.NET MVC、Web Pages、Web API 和 SignalR 发布为 NuGet 工具集中的 NuGet 包，所以只要更新项目中安装的 NuGet 包，就可以更新这些框架。Microsoft ASP.NET、Web Frameworks 2012.2 和 Microsoft Web Developer Tools 2012.2 添加了对许多新改进功能的支持，很容易在 VS 中编写 Web 应用程序，并发布到 Azure 中。Visual Studio 2012 带有 Microsoft Web Developer Tools 的 1.0 版本，但本书出版时，Microsoft Web Developer Tools 就已经是 1.2 版本了。这说明在几个月内，开发人员就可以获得最新的工具，提高创建 Web 应用程序的效率。ASP.NET 团队正在缩短发布周期，发布更新过的工具和库，使 Web 开发与当前的标准和库更一致。

1.4 本章小结

对于创建 Web 应用程序而言，ASP.NET 是可靠的、有趣的、模块化的、强大的、可伸缩的平台。无论是在 Web 窗体中重用第三方控件构建器里的报表控件，还是编写自定义的 HTML 帮助程序，都可以自信地使用 ASP.NET 作为底层平台。

也许你的解决方案是使用 TDD 和 Agile with SpecFlow 来开发的，且面向 ASP.NET MVC，或者喜欢 Ember.js 和 ASP.NET Web API，仅仅需要混合并匹配它们，就可以使用 ASP.NET 来进行。ASP.NET 正在成为一种带有各种选项的 Web 框架。

不要害怕尝试新事物，换掉一些组件，或者研究新的开源库。ASP.NET 及其团队会尽力解决可伸缩、安全和性能方面的重大难题。用户选择如何构建应用程序是一种个人喜好，应该不需要考虑 ASP.NET 是否支持你的体系结构。

ASP.NET 和 Web Tools 2012.2 的发布，使 ASP.NET 走上了 One ASP.NET 的梦幻之旅。也许不久之后，Microsoft 项目模板就与社区模板一起使用了，项目模板很容易共享并发布为 NuGet 包。ASP.NET 的最新版本使这个设想实现起来没有那么困难。

第 2 章

使用 ASP.NET 进行 HTML5 和 CSS3 设计

本章要点

- 理解 HTML5 的基础知识
- 理解 CSS3 的基础知识
- 在 ASP.NET 应用程序中使用 HTML5 和 CSS3

HTML 最初是由 Tim Berners-Lee 引入的, 当时它仅仅是研究人员使用 Internet 格式化和交叉链接其研究文档的一种简单方式。当时, Web 仍然主要基于文本, 因此这些文档的格式化要求是相当基本的。HTML 只需要很少的基本布局概念, 如标题、段落、页眉和列表。随着 Web 向一般公众开放以及引入图形浏览器, 格式化 Web 页面的要求也在不断增加, 此时就开发出了 HTML 的新版本。这些新版本扩展了 HTML 的最初功能, 以适应新的、丰富的图形浏览器环境, 允许采用表布局, 并且有更丰富的字体样式、图像和框架(frame)。

对 HTML 的所有这些改进都是有益的, 但 HTML 仍然不允许开发人员创建复杂的、高度样式化的 Web 页面。因此, 在 1994 年引入了一种新技术——层叠样式表(Cascading Style Sheets, CSS)。CSS 是 HTML 的一种补充技术, 为 Web 页面的开发人员提供了他们需要的功能, 可以精细地控制 Web 页面的样式。

随着 Web 的日益成熟, CSS 也变得更常见, 因为开发人员认识到, 与标准的 HTML 样式化功能相比, CSS 有显著的优点。HTML 最初主要用作一种布局机制, 而 CSS 从一开始就为 Web 页面提供了丰富的样式化功能。CSS 的层叠特性便于把宽笔划的样式应用于整个应用程序, 并可以在需要的地方重写该样式。CSS 很容易在外部定义 Web 站点的样式信息, 与 Web 页面的样式和结构清楚地分开。CSS 还允许开发人员极大地减小 Web 页面的文件大小, 缩短页面加载时间, 减少带宽消耗。

自本世纪初以来, 移动设备和多媒体流已经大众化。Web 2.0 应用程序继续在发展, HTML 和 CSS 也在发展。在过去的几年里, HTML5 和 CSS3 越来越常见。本章首先简要介绍 HTML5 和 CSS3, 然后讨论使用 HTML 和 CSS 在 Visual Studio 中创建 Web 站点。最好复习一下这些技术, 因为 VS 2012 中的项目模板已使用 HTML5 和 CSS3 重写过了。

2.1 警告

本章包含 HTML 和 CSS 的许多信息，以及如何在 ASP.NET 和 Visual Studio 中使用它们，但应注意一些事项。

首先，因为无法使用一章的篇幅包含 HTML 和 CSS 的所有内容，所以如果需要深入理解这些主题，可以查阅 Wrox 出版的 *HTML5 24-Hour Trainer* 一书。

其次，因为 CSS 只是个推荐的规范，所以每个浏览器供应商都需要解释和实现该规范。在 Web 开发中，每个浏览器常常采用不同的方式实现(有时不实现)不同的 CSS 功能。本章的示例都在 Internet Explorer 10 中进行了测试，读者应确保在多个平台的多个浏览器上全面测试自己的 Web 站点，使 CSS 在每个目标浏览器上都能正确显示。

最后，HTML5 和 CSS3 尚未完成制订。CSS3 与以前的版本不同，因为它由多个模块组成。在本书出版时，CSS3 已发布了 4 个模块，作为正式的规范。尽管 CSS3 规范还没有完成，但 CSS4 的第一个官方草案已由 W3C 于 2011 年 9 月发布。HTML5 仍在草案阶段。2012 年 7 月，HTML5 规范的编辑 Ian Hickson 宣布，W3C 计划在 2014 年公布 HTML5 规范的快照。这个快照称为 HTML5。在这之后，WHATWG(Web Hypertext Application Technology Working Group)继续维护该规范。

2.2 HTML5 概述

从 Web 诞生一直到现在，HTML 都是定义 Web 页面上内容块的主要机制，也是定义 Web 页面布局的最简单方式。HTML 包含非常多的可用布局标记，包括表、列表和分组元素。可以合并这些元素，在 Web 页面上创建出非常复杂的布局。图 2-1 是一个简单的 Web 页面，它使用各种 HTML 元素定义了页面的基本布局。

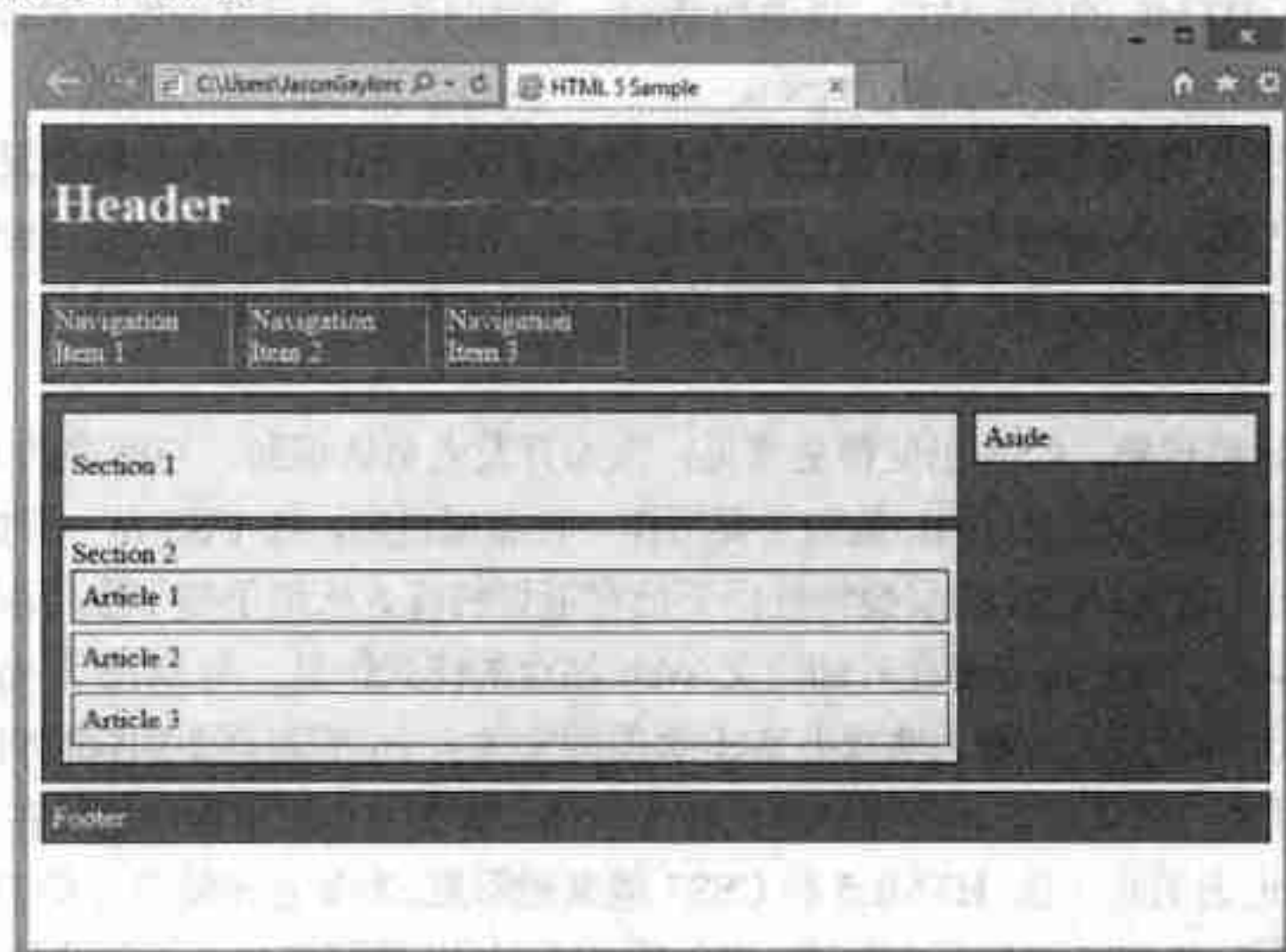


图 2-1

通过图 2-1，你会很快注意到，该 Web 页面包含页面标题、几个小标题、一个列表和几个段落。

但是与典型的 HTML4 不同, 该页面使用了 HTML5 中几个新的语义元素。下面看看这个页面的源代码, 之后详细解释新元素。



如果希望详细了解本章未涉及的 HTML4 和 HTML5 规范之间的区别, 可以访问 W3C 网站 www.w3.org, 并搜索 HTML5。

2.2.1 HTML5 中新的元素、特性和值

W3C 撰写的官方 HTML5 规范包括几个新的语义元素, 表 2-1 列出了其中的一些元素。

表 2-1

元 素	说 明
section	把 Web 页面作为该 Web 页面的全部或一部分, 例如功能化的内容
article	Web 页面上的一些信息, 例如新闻帖子、博客帖子或状态消息
header	Web 页面的顶部, 可以包含徽标或其他页面标题
nav	某个特定的部分, 用于包含导航元素, 例如菜单或面包屑类型的导航
aside	Web 页面的某个部分, 提供与整个内容部分相关的内容, 例如边栏或广告
figure	这个区域用于表示, 一部分内容显示与 Web 页面相关的信息, 例如图表或视频
footer	Web 页面的底部, 可以包含版权信息、隐私通知或作者的其他信息
video	显示多媒体视频
audio	播放声音, 是 bgsound 的替代元素

还有几个新特性也被添加到 HTML5 的已有元素中。表 2-2 列出了所添加的一些最常用特性。

表 2-2

特 性	说 明
autofocus	在加载页面时, 允许 input(类型为 hidden 的除外)、select、textarea 和 button 元素获得焦点
placeholder	提供提示, 为 input 和 textarea 元素输入数据
required	需要为 input(类型为 hidden、image 或按钮类型的除外)、select、textarea 元素输入或选择数据
min、max、step、multiple、pattern	指定 input 元素所受的限制
async	异步加载 script 元素
seamless	为 iframe 元素提供无边框的框架。在带 seamless 特性的 iframe 元素中单击链接时, 内容会加载到父框架中, 除非 target 特性使用了 _blank 值
reversed	翻转 ol 元素中列表项的顺序

HTML5 除了包含语义元素和新特性之外, 还给 input 元素的 type 属性添加了几个新值。在出版本书时, 包含如下值:

- **tel:** 一个字符串值, 它不会强制用户输入有效的电话号码, 因为可能的值会根据位置而改变。但是, 应用 **pattern** 特性, 并给该特性设置一个值, 就可以进行输入验证。
- **search:** 一个非常类似于 **text** 类型的字符串值, 但允许某些浏览器或应用程序用不同的方式识别它。
- **url:** 一个字符串值, 表示一个绝对的 URL。
- **email:** 一个字符串值, 表示一个有效的电子邮件地址。当指定多个特性时, 电子邮件地址用逗号分隔开。
- **datetime:** 一个字符串值, 表示一个有效的全局日期时间值。有效的全局日期时间值包含公历日期和时间。公历日期包含年、月、日, 时间包含小时、分钟、秒、毫秒, 用时区的偏移量来表示。
- **date:** 一个字符串值, 表示一个有效的日期值。有效的日期值包含年、月、日, 并用连字符分隔开。有效的日期不包含时区。
- **month:** 一个字符串值, 表示有效的月份值, 用 4 位数字的年份值、连字符和两个数字的月份值来表示。
- **week:** 一个字符串值, 表示有效的星期, 用 4 位数字的年份值、连字符和两个数字的星期值来表示。如果是闰年, 星期值的最大值就是 53。否则, 星期值的最大值就是 52。如果是从星期一开始计数, 就计入一个星期。
- **time:** 一个字符串值, 表示有效的时间, 有效的时间包含小时、分钟和秒。有时也可以包含至多 3 位数字的微秒数。有效的时间不包含时区。
- **datetime-local:** 一个字符串值, 表示有效的本地日期和时间值。它与 **datetime** 值类似, 也包含公历日期和时间, 但不包含时区。
- **number:** 一个字符串值, 包含正负浮点数。在出版本书时, 这个值的值域是 $2^{1024} \sim -2^{1024}$ 。
- **range:** 一个字符串值, 包含正负浮点数, 如前面的定义所示。但是这个值在使用 **min**、**max** 和 **step** 属性时限制比较多。如果没有提供 **min**、**max** 和 **step** 属性, 其默认的最小值是 0, 最大值是 100。
- **color:** 一个字符串值, 包含一个简单的颜色值。简单的颜色值定义为有效的 **sRGB** 值, 表示为数字符号(#)后跟 **sRGB** 颜色的小写十六进制值。例如, 白色表示为 **#ffffff**。该值的长度必须是 7 个字符。

例如, 程序清单 2-1 显示了图 2-1 中 Web 页面生成的 HTML 源代码。为了简洁起见, 其中删除了样式元素。

程序清单 2-1 图 2-1 中 Web 页面的源代码

```
<!DOCTYPE html>
<html>
<head>
  <title>HTML5 Sample</title>
</head>
<body>
  <header>
    <h1>Header</h1>
  </header>
```

```

<nav>
  <ul>
    <li>Navigation Item 1</li>
    <li>Navigation Item 2</li>
    <li>Navigation Item 3</li>
  </ul>
</nav>
<div id="mainContent">
  <aside>
    Aside
  </aside>
  <section>
    <p>
      Section 1
    </p>
  </section>
  <section>
    Section 2
    <article>
      Article 1
    </article>
    <article>
      Article 2
    </article>
    <article>
      Article 3
    </article>
  </section>
</div>
<footer>
  Footer
</footer>
</body>
</html>

```

查看源代码，可以注意到 HTML5 中的一项变化是，DOCTYPE 名称缩短了。缩短 DOCTYPE 名称的原因是，现代浏览器会加载文档，并根据所加载的内容切换到对应的 DOCTYPE。

接下来注意到的一系列变化是，使用了语义元素，前面介绍了这些元素的作用。但是，在查看 HTML 源代码时，很容易阅读它们。即使其他开发人员对网站访问较少，也能理解其内容布局和标题、部分、文章、边栏和页脚之间的分隔。

2.2.2 使用新的 HTML5 标记

前面学习了 HTML5 中的大多数标记变化。为了更好地理解如何使用这些新元素、特性和值，下面练习两个示例项目。

场景 1：实现倒序的前 10 名

假定第一项任务是提供某篇体育文章的标记。该文章包含处于本区域前 10 位的高中篮球队的列表。但是，不是要将这些篮球队按从 1 到 10 的顺序列出，而是以倒序方式列出。如果使用的是 HTML4，要完成这个任务，就必须违反 HTML 4.01 规范。对于这个示例，生成的 HTML 如程序清

单 2-2 所示(为了便于演示,删减了代码)。

程序清单 2-2 用 HTML4 创建倒序列表

```
<h1>Top Three Teams</h1>
<ol>
  <li value="3">Third Place</li>
  <li value="2">Runner Up</li>
  <li value="1">Top Team</li>
</ol>
```

现在,为有序列表使用 HTML5 中的 `article` 元素和 `reversed` 属性,如程序清单 2-3 所示。

程序清单 2-3 使用 `article` 元素和 `reversed` 属性

```
<article>
  <h1>Top Ten Baseball Teams</h1>
  <ol reversed="reversed">
    <li>Coughlin</li>
    <li>Hazleton Area</li>
    <li>Pittston Area</li>
    <li>Nanticoke Area</li>
    <li>Hanover Area</li>
    <li>Lake Lehman</li>
    <li>Crestwood</li>
    <li>Wyoming Valley West</li>
    <li>Dallas</li>
    <li>Wyoming Area</li>
  </ol>
</article>
```

在 IE 中查看结果,列出的篮球队仍是从 1 到 10。如前所述,HTML5 规范还没有完成。目前,并不是所有的浏览器都支持 `reversed` 属性的布尔值。但是,在 Google Chrome 中,会看到正确的结果,如图 2-2 所示。在本章的后面你将学习如何在大多数浏览器中实现强制兼容性。

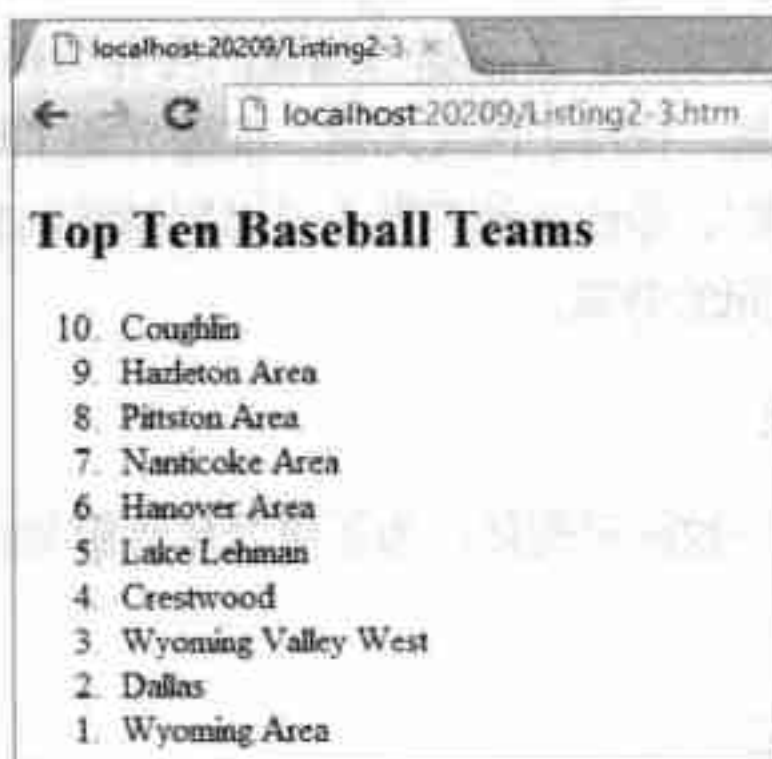


图 2-2

场景 2: 捕获有效的美元值

下面创建一个电子商务应用程序。一项要求是允许产品输入员工输入产品的有效价格。有效价

格有如下要求:

- 产品的最低价可以是\$1.00
- 产品的最高价可以是\$99.95
- 产品价格的增量必须是 5 美分
- 产品价格是必须有的

如果使用纯 HTML4, 就不能完成这个任务。最多只能创建一个下拉列表, 其中的列表项从 1.00 到 99.95。而且, 如果只使用客户端代码, 就需要编写 JavaScript, 强制将值输入到文本域中。

与第一种场景一样, 使用 HTML5 很容易完成这个任务。实际上, 大多数现代浏览器, 包括 IE, 都提供了友好的验证消息, 而无须添加自定义的 JavaScript。添加一个简单的 input 元素, 如下所示, 就可以满足上述所有要求:

```
<input name="price" type="number" min="1" max="99.95" step="0.05" required />
```

可以看出, input 元素的 type 属性被设置为 number。number 类型允许设置为正负浮点数。接着, 给 input 元素添加新属性。为了满足要求, 把 min 属性设置为 1, 把 max 属性设置为 99.95。这会确保数据输入员工只能输入 min 和 max 之间的值(包含 min 值和 max 值)。再添加 step 属性, 把它设置为 0.05。有了这个属性值, 价格就只能每次递增 5 美分。最后, 添加 required 属性, 强制为这个 input 元素输入值。输入不正确的值时, 例如 1.01, 用户就会收到如图 2-3 所示的验证消息。

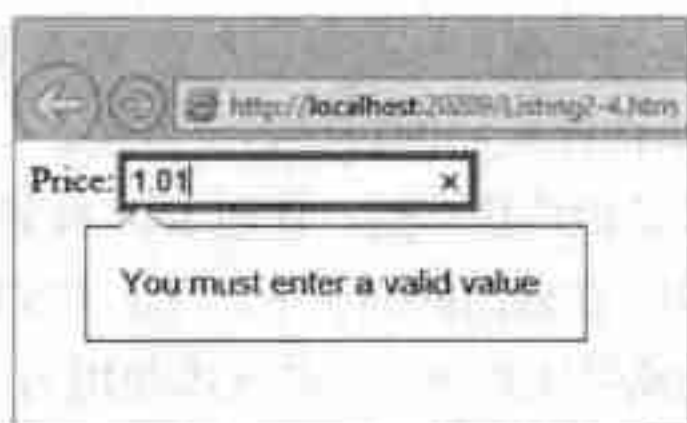


图 2-3

2.2.3 HTML5 中新的 API

除了前面提及的标记语言的变化之外, HTML5 规范还包含许多用 JavaScript 访问的新 API。目前 HTML5 规范包含的一些新 API 如下:

- 多媒体
- 离线 Web 应用程序支持
- 拖放
- 会话浏览器历史
- 打印
- Base64 转换
- 搜索提供程序

HTML5 规范还有一些非官方的其他 API, 其中一些如下:

- Microdata
- Canvas 和 SVG
- 背景脚本

- 客户端数据存储
- WebSocket(客户端-服务器之间的双向通信)
- EventSource(服务器到客户端的通信)
- Geolocation



如果希望进一步学习这些 API, 可以访问 WHATWG 网站 www.whatwg.org 或在线搜索 HTML5 API。

2.3 CSS3 概述

前面的程序清单 2-1 复习了 HTML 标记和语义元素。尽管这个 Web 页面的组织很有趣, 但它只有最基本的样式。为了解决这个问题, 许多开发人员都尝试添加基于 HTML 的格式化标记。例如, 如果要改变第一段文本的字体和颜色, 就需要修改 HTML, 如下所示:

```
<font face="Arial" Color="Maroon">
```

实际上, 在 Web 设计工具发展的早期, 这就是用户在 Web 页面上添加样式而生成的结果。当时, 使用 font 标记似乎是使 Web 页面样式化的极佳解决方案。

但是, Web 开发人员和设计人员很快意识到, 使用 font 标记会很快使 HTML 变得混乱起来, 因为 font 标记散布在整个 HTML 中。在前面的例子中, 假定不仅要设置颜色, 还需要使一些字体变成粗体, 一些字体采用不同的颜色或字体系列, 一些文本使用另一种字体大小, 一些字体要添加下划线, 一些字体要显示为上标。需要这么多 font 标记, 它们会增大 Web 页面, 降低其可维护性。使用 font 标记(和其他与样式相关的标记), Web 页面的结构和内容就不再能清楚地分开, 它们会混合在一起, 成为复杂的文档。

在 Web 开发和设计领域中引入 CSS, 使得为 Web 页面的样式化重新找到了一种清晰、简洁的解决方案。CSS 意味着可以在一个地方定义用于整个 Web 站点的样式, 之后再在需要该样式的元素上引用。使用 CSS 可以使 Web 页面的内容和用于显示内容的样式再次从逻辑上分开。



CSS 在 Web 开发中非常普遍, 所以 HTML5 规范废弃了一些过去表示样式的 HTML 元素, 例如 font、center 和 strike 元素。

2.3.1 创建样式表

与 HTML 一样, CSS 也是一种解释性语言。Web 服务器处理 Web 页面请求时, 服务器的响应可以包含样式表, 样式表就是 CSS 指令的简单集合。样式表包含在服务器响应中的方式有 3 种: 外部样式表文件、直接内嵌在 Web 页面中的内部样式表或内联样式表。

1. 外部样式表

外部样式表是在将使用这些样式的 Web 页面外部存储的 CSS 样式集合，一般是扩展名为.css 的文件。Visual Studio 很容易在如图 2-4 所示的 Add New Item 对话框中包含 Style Sheet 文件模板，从而把外部样式表文件添加到应用程序中。



右击项目，选择 Add | Add New Item，或者选择 File | New | File，或者按下 Ctrl+N 快捷键，就可以访问 Add New Item 对话框。

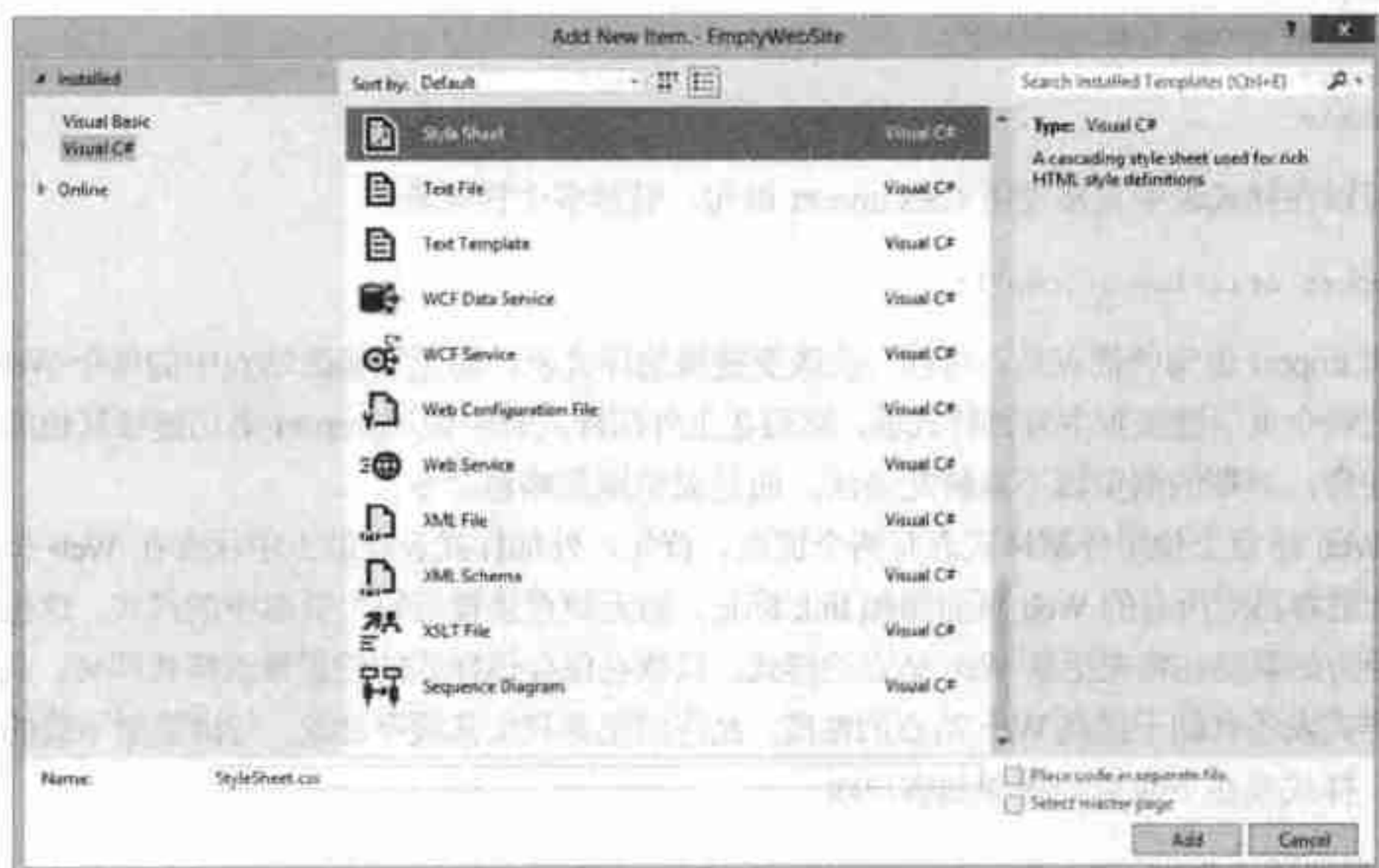


图 2-4

在 Visual Studio 创建完样式表后，就很容易插入新样式。Visual Studio 甚至在处理文档中的样式时提供了 CSS IntelliSense，如图 2-5 所示。

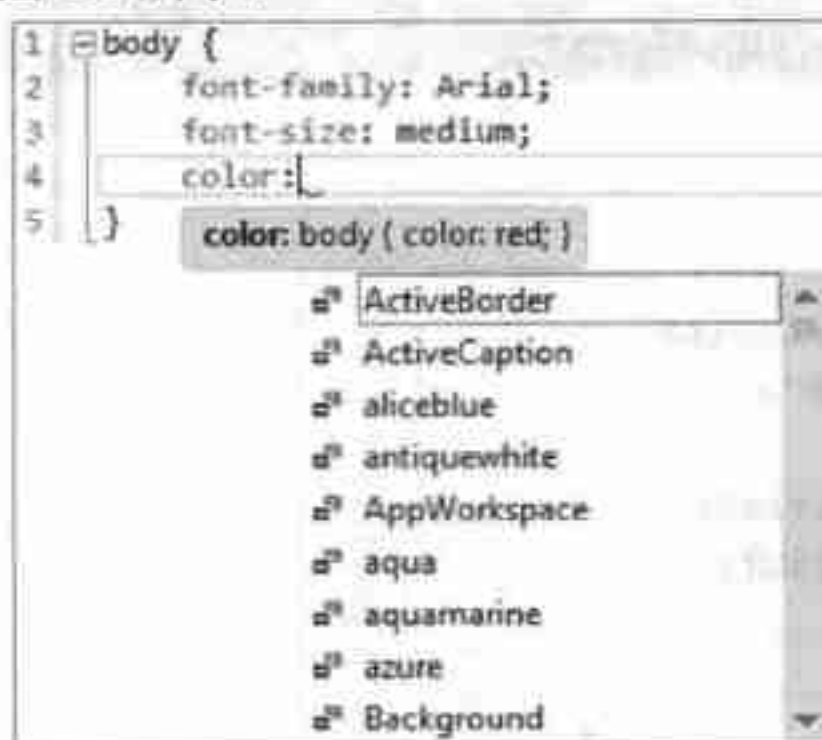


图 2-5

使用 HTML `<link>` 标记将外部样式表链接到 Web 页面。一个 Web 页面可以包含多个样式表引用，如程序清单 2-4 所示。

程序清单 2-4 在 Web 页面中使用外部样式表

```
<!DOCTYPE html>
<html>
<head>
  <title>HTML 5 Sample</title>
  <link href="primary.css" type="text/css" rel="stylesheet" />
  <link href="secondary.css" type="text/css" rel="stylesheet" />
</head>
<body>
  <div>Lorem Ipsum</div>
</body>
</html>
```

也可以在样式表中直接使用 CSS `import` 语句，链接多个样式表：

```
@import url("layout.css");
```

使用 `import` 语句的优点是，可以一次改变链接的样式表，而无须修改站点中的每个 Web 页面。也可以把每个页面链接到主外部样式表，然后在主外部样式表中使用 `import` 语句链接其他的外部样式表。注意，早期的浏览器不理解此语法，而是简单地忽略该命令。

在 Web 站点上使用外部样式表有两个优点。首先，外部样式表在站点中保存在 Web 页面的外部，因此更容易给所有的 Web 页面添加 `link` 标记，而无须直接管理每个页面中的样式。这也更便于维护，因为如果要在将来更新 Web 站点的样式，只须在保存该样式的位置修改样式即可。其次，使用外部样式表还有助于提高 Web 站点的性能，允许浏览器利用其缓存功能。与浏览器下载的其他文件一样，样式表在下载后会缓存到客户端。

2. 内部样式表

内部样式表是在单个 Web 页面内部存储的 CSS 样式集合。这些样式位于 HTML `<style>` 标记中，这个标记一般位于 Web 页面的 `<head>` 部分。内部样式表的示例如程序清单 2-5 所示。

程序清单 2-5 在 Web 页面中使用内部样式表

```
<!DOCTYPE html>
<html>
<head>
  <title>HTML5 Sample</title>
  <style type="text/css">
    body {
      font-family: Arial;
      font-size: medium;
    }
  </style>
</head>
<body>
  <div>Lorem Ipsum</div>
```

```
</body>
</html>
```

在创建内部样式表的样式块时，一定要在样式标记中包含 `type` 特性，这样浏览器才知道如何正确解释该样式块。另外，与外部样式表一样，Visual Studio 也提供了 IntelliSense 支持，以便于添加属性。

3. 内联样式

内联样式是将大多数 HTML 元素都有的 `style` 特性直接应用于单个 HTML 元素的 CSS 样式。内联样式的示例如程序清单 2-6 所示。

程序清单 2-6 在 Web 页面中使用内联样式

```
<!DOCTYPE html>
<html>
<head>
  <title>Inline CSS Sample</title>
</head>
<body>
  <div style="font-family: Arial;">Lorum Ipsum</div>
</body>
</html>
```

2.3.2 CSS 规则

无论 CSS 样式如何存储，一旦从服务器将它们发送到客户端，浏览器就负责解析样式，把它们应用于 Web 页面中相应的 HTML 元素。如果样式存储在外部或内部样式表中，样式就定义为 CSS 规则。浏览器使用该规则确定应用什么样式，以及应用于哪些 HTML 元素。



内联样式不需要定义为规则，因为它们会自动应用于包含它们的元素。因此，浏览器不需要选择要应用该样式的元素。

规则由两部分组成：选择器和属性。图 2-6 是 CSS 规则的一个例子。

```
p
{
  font-family: Arial;
  font-size: 12pt;
  color: Maroon;
  padding: 5px;
  letter-spacing: 3pt;
  text-decoration: underline;
  font-weight: bold;
  line-height: 1px;
  text-align: center;
}
```

选择器

属性

图 2-6

1. 选择器

选择器是 CSS 规则中明确指定 Web 浏览器如何选择要应用样式的元素的部分。CSS 包含许多类型的选择器，每种选择器都定义了不同的元素选择技术。

通用选择器

通用选择器指定样式应用于 Web 页面的所有元素。下面的示例演示了一个通用选择器，它把支持 font-family 属性的元素的字体改为 Arial：

```
*
{
    font-family:Arial;
}
```

类型选择器

类型选择器可以创建样式，应用于指定类型的 HTML 元素。接着把该样式应用于 Web 页面上该类型的所有元素。下面的示例演示了一个为 HTML 段落标记配置的类型选择器，它会把 Web 页面上所有<p>标记的字体系列改为 Arial：

```
p
{
    font-family:Arial;
}
```

派生选择器

派生选择器可以创建样式，其目标 HTML 元素是指定类型的元素的派生元素。下面示例中的样式应用于派生自<div>标记的所有标记：

```
div span
{
    font-family:Arial;
}
```

子选择器

子选择器类似于派生选择器，但派生选择器会搜索元素的整个派生层次结构，而子选择器把元素搜索范围限制在父元素的直接子元素上。下面的代码修改了派生选择器示例，使其变成子选择器示例：

```
div > span
{
    font-family:Arial;
}
```

属性选择器

属性选择器可以定义样式，根据元素是否有某个属性(而不是元素名称)，把该样式应用于元素。例如，下面的示例创建了一个样式，它可应用于 Web 页面上所有设置了 href 属性的元素：

```
*[href]
{
```



```
font-family:Arial;
}
```

CSS3 给属性选择器添加了一些额外的行为。从 CSS3 开始,不仅可以指定属性,还可以指定属性值。例如,假定要显示如图 2-7 所示的 Web 链接列表。对于每个以 `http://` 开头的链接,需要在其左边放置一个锁定图标:

```
div a[href^="https://"] {
    background-image: url(img/lock-icon.png);
    background-repeat: no-repeat;
    background-position: left;
}
```

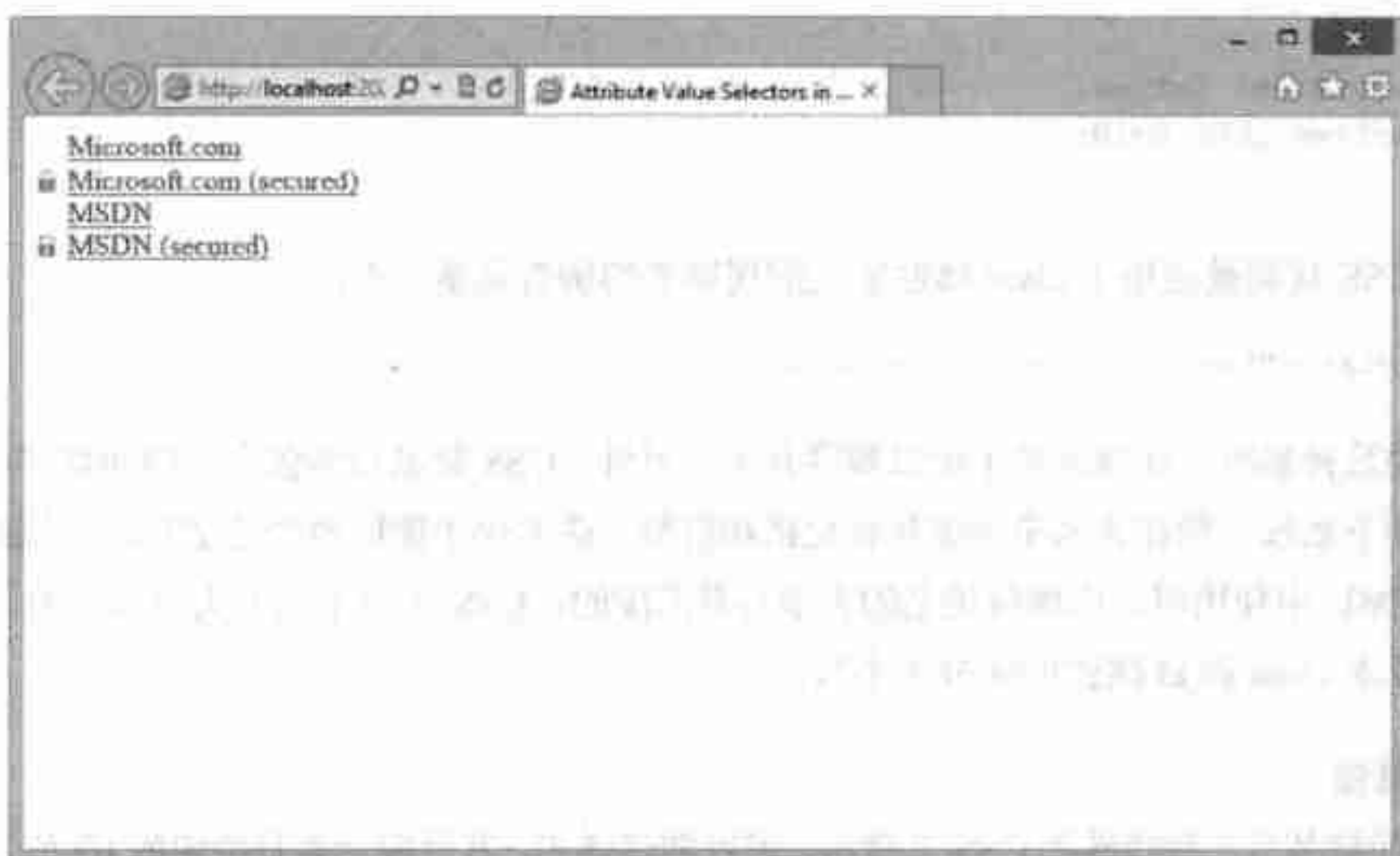


图 2-7



Internet Explorer 6 等旧浏览器不支持属性选择器。但 Web 开发人员可以确定是否要支持旧浏览器。访问 www.ie6countdown.com, 可以看到 PC 安装 IE6 的当前百分比。

相邻选择器

相邻选择器可以选择直接相邻另一元素类型的 HTML 元素。例如,在无序列表中,需要突出显示第一个列表项,再对其后的所有项使用另一种样式。此时就可以使用相邻选择器,如下面的示例所示:

```
li
{
    color: maroon;
}
li+li
{
    color: silver;
}
```

在这个例子中，为列表项元素创建了一个默认的类型选择器，它会把元素中文本的颜色改为栗色。然而，还创建了第二个相邻选择器，它将覆盖第一个列表项后面的所有列表项的类型选择器，把颜色改为银色。

类选择器

类选择器是一种特殊类型的 CSS 选择器，可以把样式应用于有指定类名的元素，类在 HTML 中使用几乎每个元素都有的 class 属性定义类名。类选择器与其他选择器类型的区别在于有前缀——句点(.):

```
.title
{
    font-size: larger;
    font-weight: bold;
}
```

这个 CSS 规则被应用于 class 属性值匹配规则名的所有元素，例如：

```
<div class="title">Lorum Ipsum</div>
```

创建类选择器时，注意类名不能以数字开头。另外，CSS 类名只能包含字母和数字，也可以包含连字符和下划线，但在类名中不能包含空格和符号。类名也不能以两个连字符或下划线开头。最后，在 HTML 中使用时，应确保类名的大小写是匹配的。CSS 本身不区分大小写，但一些 HTML DocType 要求 class 和 id 属性应区分大小写。

ID 选择器

ID 选择器是另一种特殊的 CSS 选择器，可以创建样式，其目标元素有特定的 ID 值。ID 选择器与其他选择器的区别在于有前缀——散列符号(#):

```
#title
{
    font-size: larger;
    font-weight: bold;
}
```

这个 CSS 规则会被应用于 id 属性值匹配规则名的所有元素，例如：

```
<div id="title">Lorum Ipsum</div>
```

2. 伪类

CSS 还包含一系列伪类选择器，在创建样式规则时提供了额外的选项。伪类可以添加到其他选择器以创建更复杂的规则。

第一个子元素伪类

第一个子元素伪类可以指定，规则应选择元素 N 的第一个子元素 M。下面是使用第一个子元素伪类的例子：

```
#title p:first-child
```

```
{
    font-size: xx-small;
}
```

上面定义的规则表明，样式应应用于 `id` 属性值为 `title` 的元素中的第一个段落标记。在下面的 HTML 中，这表示文本 `First Child` 要应用该样式：

```
<div id="title">
    Lorum
    <p>First Child</p>
    <p>Second Child</p>
    Ipsum
</div>
```

CSS 包含与锚点标记相关的许多伪类，这些特殊的伪类可以为锚点标记的不同状态定义样式：

```
a:link
{
    color: maroon;
}
a:visited
{
    color: silver;
}
```

在这个例子中，创建了两个规则：第一个规则把样式应用于页面上未访问的链接，第二个规则把另一个样式应用于已访问的链接。

动态伪类

动态伪类是浏览器根据终端用户执行的操作(比如把光标停放在元素上、激活某个元素或给元素指定焦点)应用样式的特殊 CSS 类。

```
a:hover
{
    color: maroon;
}
a:active
{
    color: silver;
}
a:focus
{
    color: olive;
}
```

该例演示了如何给锚点标记使用动态伪类，这些伪类也可以用于其他 HTML 元素。但是需要注意，对动态伪类的支持在不同的浏览器中是有所区别的。

语言伪类

语言伪类可以根据终端用户的语言设置来定义特殊的规则。

```
:lang(de)
{
```



```
quotes: '<<' '>>' '\2039' '\203A'
```

在这个例子中，`lang` 伪类用于设置德语版的 Web 页面的引号。



Internet Explorer 7 不支持 `lang` 伪类。

3. 伪元素

CSS 还包含几个伪元素，它们可以在 Web 页面上选择不是真正元素的项。

可用的伪元素有 `first-line`、`first-letter`、`before` 和 `after`。下面的示例演示了这些伪元素的使用法：

```
p:first-line
{
    font-style: italic;
}
p:first-letter
{
    font-size: xx-large;
}
```

伪元素 `first-line` 和 `first-letter` 可以把特定的样式应用于内容块的第一行和第一个字母。

```
p:before
{
    content: url(images/quote.gif);
}
p:after
{
    content: '<<end>>';
}
```

伪元素 `before` 和 `after` 可以在目标元素的前面或后面插入内容，在上面的例子中就插入了一个段落元素。插入的内容可以是 URL、字符串、引号字符、计数器或元素的属性值。

4. 选择器的分组

创建 CSS 规则时，CSS 允许把多个选择器分组到单条规则中。下面的示例演示了合并 3 个类型选择器的规则：

```
h1, h2, h3
{
    color: maroon;
}
```

该规则把 `h1`、`h2` 或 `h3` 标记的文本内容的前景色改为栗色。

5. 选择器的合并

CSS 还允许合并多种选择器类型，如程序清单 2-7 所示。例如，可以创建类选择器，应用于特

定的 HTML 元素, 并且匹配 class 属性值。

程序清单 2-7 在 CSS 规则中合并多种选择器类型

```
<!DOCTYPE html>
<html>
<head>
  <title>Combining Selector Types</title>
  <style type="text/css">
    .title
    {
      font-family:'Courier New';
    }

    div.title
    {
      font-family:Arial;
    }
  </style>
</head>
<body>
  <article>
    <p class="title">Lorum Ipsum</p>
    <div class="title">Lorum Ipsum</div>
  </article>
</body>
</html>
```

6. 样式的合并

在定义应用于给定 HTML 元素的样式规则时, CSS 还会合并样式。例如, 在程序清单 2-8 所示的示例代码中, 定义了一个类选择器和一个类型选择器, 这两个选择器都应用于 HTML 中的段落元素。当浏览器解释这些样式时, 会把它们合并到元素上。

程序清单 2-8 把多个规则中的样式合并到单个元素上

```
<!DOCTYPE html>
<html>
<head>
  <title>Merging Styles</title>
  <style type="text/css">
    .title
    {
      text-decoration:underline;
    }

    p
    {
      font-family:'Courier New';
    }
  </style>
</head>
<body>
```

```
<article>
  <p class="title">Lorum Ipsum</p>
</article>
</body>
</html>
```

如图 2-8 所示, 单个段落元素的字体和文本修饰都应用了样式, 但实际上有两个样式规则定义了该样式。



图 2-8

还可以使用不同的选择器类型定义多个规则, 以此来合并多个样式。如果单个 HTML 元素匹配所有的规则, 就把每个规则中的样式合并在一起。在程序清单 2-9 所示的例子中, 单个元素匹配多个规则。

程序清单 2-9 多个选择器匹配单个元素

```
<!DOCTYPE html>
<html>
<head>
  <title>Multiple Selector matches</title>
  <style type="text/css">
    p
    {
      font-family: Arial;
      color: blue;
    }

    p#book
    {
      font-size: xx-large;
    }

    p.title
```



```

        {
            font-family: 'Courier New';
        }
    </style>
</head>
<body>
    <article>
        <p id="book" class="title" style="letter-spacing:5pt;">Lorum Ipsum</p>
    </article>
</body>
</html>

```

在这个例子中，由于段落标记定义了 `id`、`class` 和 `style` 属性，每个样式规则都匹配，因此这些样式被合并到元素上。

最后，`class` 属性本身也可以用于把多个样式合并到同一元素上，如程序清单 2-10 所示。`class` 属性可以在以空格分隔的字符串中指定多个类名。

程序清单 2-10 把多个类选择器赋给单个元素

```

<!DOCTYPE html>
<html>
<head>
    <title>Multiple Class Selectors</title>
    <style type="text/css">
        p.title
        {
            font-family: 'Courier New';
            letter-spacing: 5pt;
        }

        p.summer
        {
            color: Blue;
        }

        p.newproduct
        {
            font-weight: bold;
            color: red;
        }
    </style>
</head>
<body>
    <article>
        <p class="title newproduct summer">Lorum Ipsum</p>
    </article>
</body>
</html>

```

在这个例子中，在 `class` 属性中定义类 `title`、`summer` 和 `newproduct`，这就表示这 3 个样式会合并到段落元素上。

注意在这种情况下，CSS 类在内部样式表中的定义顺序也会影响样式合并到段落标记上的方式。

即使 summer 类位于 class 属性定义的类列表中的最后, newproduct 规则也会重写 summer 规则的颜色属性, 因为在内部样式表中, newproduct 规则是在 summer 规则的后面定义的。

2.3.3 CSS 继承

CSS 包含样式继承的概念, 这是有效的方式, 因为浏览器把可以定义样式的位置(外部、内部和内联)看作层次结构。图 2-9 显示了这种继承关系, 演示了在 3 个不同位置定义的段落类型选择器规则的 font-family 属性可以被其他样式规则重写。



图 2-9

从图 2-9 中可以看出, 样式的定义越接近应用样式的元素, 其优先级就越高。在这个例子中, 最终应使用 Courier New 字体显示段落文本, 因为已在内联样式中定义该字体。

继承关系不仅应用于在不同文件位置中保存的样式, 还应用于同一位置中的样式。这就说明, 有时还需要考虑定义样式的顺序。例如, 程序清单 2-11 显示了一个样式表, 它包含两个类型选择器, 这两个选择器都应用于段落元素, 并且都设置了 font-family 样式属性。显然, 它们不可能应用于同一个元素, 因此 CSS 选择最接近段落标记的选择器。

程序清单 2-11 在同一个内部样式表中使用样式重写功能

```
<!DOCTYPE html>
<html>
<head>
  <title>Styling Overriding</title>
  <style type="text/css">
    p
    {
      font-family: Arial;
    }
    p
    {
      font-family: 'Courier New';
    }
  </style>
</head>
<body>
  <article>
    <p>Lorum Ipsum</p>
  </article>
</body>
</html>
```

运行这个示例，你会看到所应用的字体是 Courier New。

注意，应谨慎合并外部样式表和内部样式表中的样式。浏览器最终会选择最接近特定元素的样式，这说明浏览器开始分析 Web 页面时，在外部样式之前定义的内部样式应远离 HTML 元素。因此，浏览器会使用外部样式表中的样式。如果打算在内部和外部样式表中存储样式规则，应在 Web 页面的内部样式表的<style>块之前包含外部样式表<link>标记。

2.3.4 元素的布局 and 定位

CSS 不仅可用于使页面上的元素样式化，还可以定位元素。CSS 为定位元素提供了比 HTML 更灵活的系统。CSS 把 Web 页面上的元素定位建立在“框模型”的基础上。确定元素的框行为后，就可以使用几种不同的技术定位元素。

1. CSS 的框模型

在 CSS 中，定位的核心元素是框模型。框模型定义了浏览器如何把 HTML 中的每个元素看作矩形方框。该方框由不同的部分组成，包括页边距、内边距、边框和内容。图 2-10 显示了如何合并这些元素以构成该方框。



图 2-10

构成该方框的所有元素可以影响元素在 Web 页面中的位置，除非特别指定，否则每个元素的默认值都是 0。元素的高度和宽度等于页边距外边界的高度和宽度，如图 2-7 所示，它们不一定等于内容的高度和宽度。

HTML 提供了两种不同类型的方框：块框(block box)和内联框(inline box)。块框一般使用<p>、<div>或<table>等标记表示，或者用新的 HTML5 语义元素<article>、<section>、<header>、<aside>、<nav>或<footer>等表示，块框的包含块用于确定子块的位置。另外，块框可以包含内联框或块框，但不能同时包含两者。

程序清单 2-12 显示的页面包含一个父块和两个子块元素。

程序清单 2-12 创建块框元素

```
<div>
  Lorem ipsum dolor sit amet, consectetur adipiscing elit.
  <div>
    Donec et velit a risus convallis porttitor.
    Vestibulum nisi metus, imperdiet sed, mollis
    condimentum, nonummy eu, magna.
  </div>
  <div>
    Duis lobortis felis in est. Nulla eu velit ut nisi
```



```

        consequat vulputate.
    </div>
    Vestibulum vel metus. Integer ut quam. Ut dignissim, sapien
    sit amet malesuada aliquam, quam quam vulputate nibh, ut
    pulvinar velit lorem at eros. Sed semper lacinia diam. In
    faucibus nonummy arcu. Duis venenatis interdum quam. Aliquam
    ut dolor id leo scelerisque convallis. Suspendisse non velit.
    Quisque nec metus. Lorem ipsum dolor sit amet, consectetur
    adipiscing elit. Praesent pellentesque interdum magna.
</div>

```

第二种框类型是内联框。一般使用、<i>、等标记以及实际的文本和内容表示内联框。程序清单 2-13 说明了如何将程序清单 2-12 修改为包含内联框。

程序清单 2-13 创建内联框元素

```

<div>
    Lorem <strong>ipsum</strong> dolor sit amet, consectetur adipiscing elit.
    <div>
        Donec et velit a risus <strong>convallis</strong> porttitor.
        Vestibulum nisi metus, imperdiet sed, mollis
        condimentum, nonummy eu, magna.
    </div>
    <div>Duis lobortis felis in est.
        <span>Nulla eu velit ut nisi consequat vulputate.</span>
    </div>
    <i>Vestibulum vel metus.</i> Integer ut quam. Ut dignissim, sapien
    sit amet malesuada aliquam, quam quam vulputate nibh, ut
    pulvinar velit lorem at eros. Sed semper lacinia diam. In
    faucibus nonummy arcu. Duis venenatis interdum quam. Aliquam
    ut dolor id leo scelerisque convallis. Suspendisse non velit.
    Quisque nec metus. Lorem ipsum dolor sit amet, consectetur
    adipiscing elit. Praesent pellentesque interdum magna
</div>

```

显示这个页面，会使每个块从一个新行开始显示。图 2-11 是在浏览器中显示的标记。

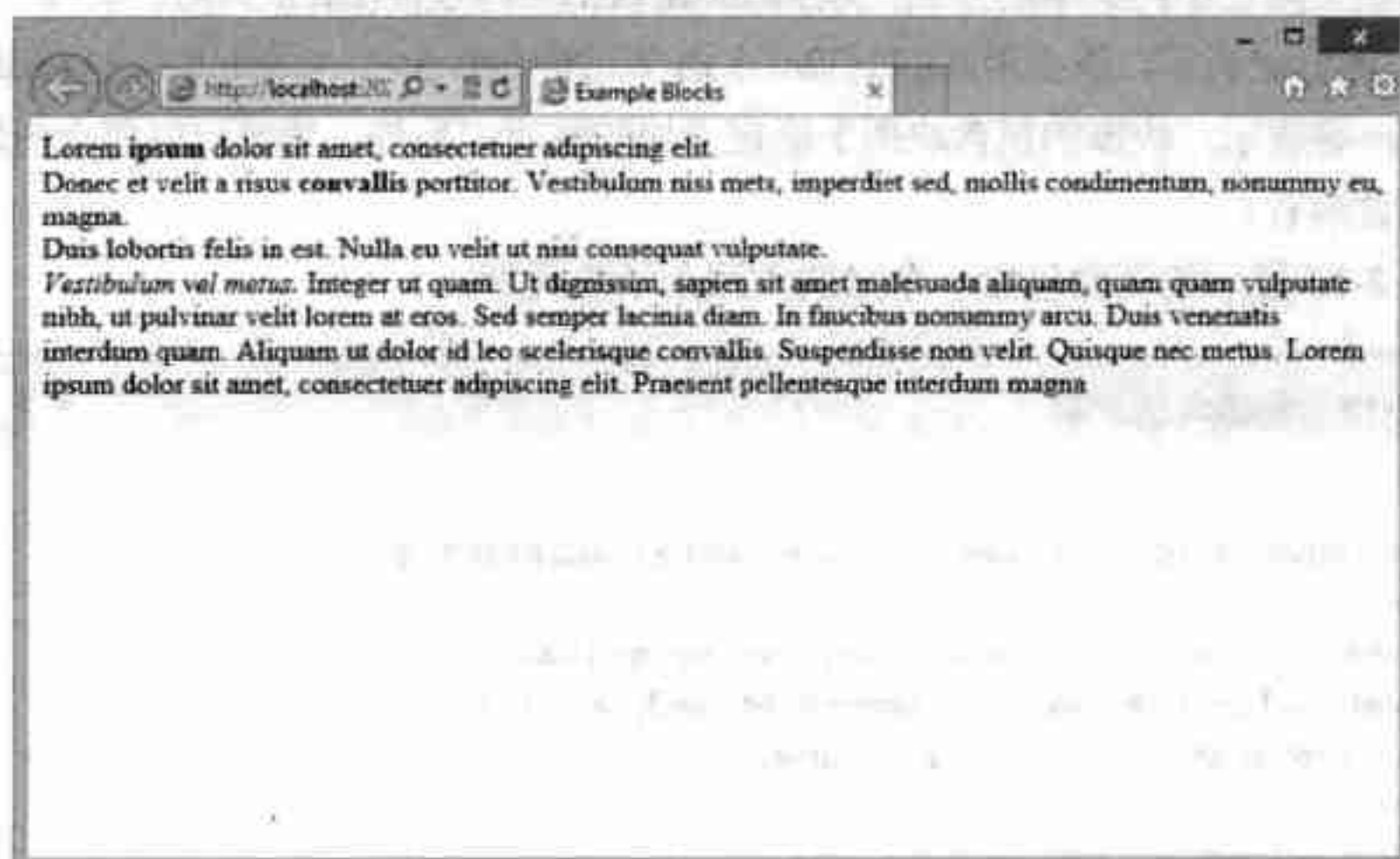


图 2-11

如图 2-15 所示，在 View 菜单上有两个选项，用于使设计界面切换设置了这些属性的元素的可见性。

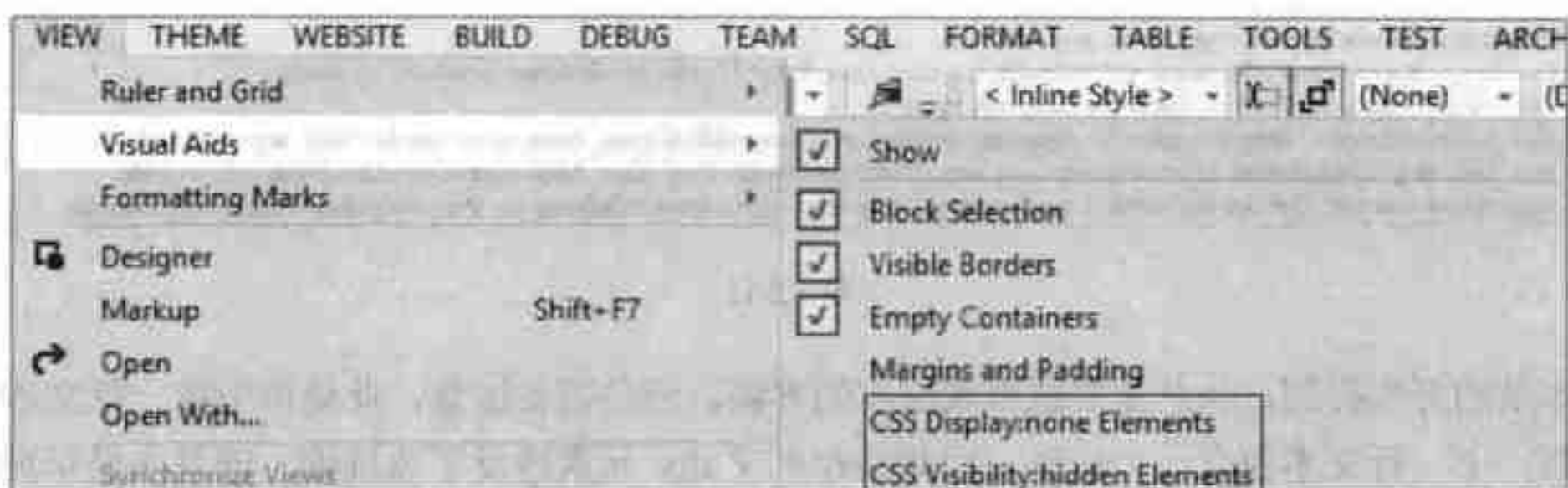


图 2-15

2. 定位 CSS 元素

CSS 提供了 3 种主要的定位机制：一般、绝对和相对。每种机制都提供了不同的行为，可以用于定位页面上的元素。要指定元素使用的布局类型，可以设置 CSS 的 `position` 属性。每个元素都可以设置自己的 `position` 属性，从而在同一 Web 页面上使用多种定位方案。

一般定位机制

使用一般定位机制，块项会垂直布置，内联项会从左到右水平布置。这是默认定位方式，如果没有给 `position` 属性提供其他值，就使用该方式。图 2-16 演示了 4 个块使用一般定位机制的布局。可以看出，每个块项都垂直布置。

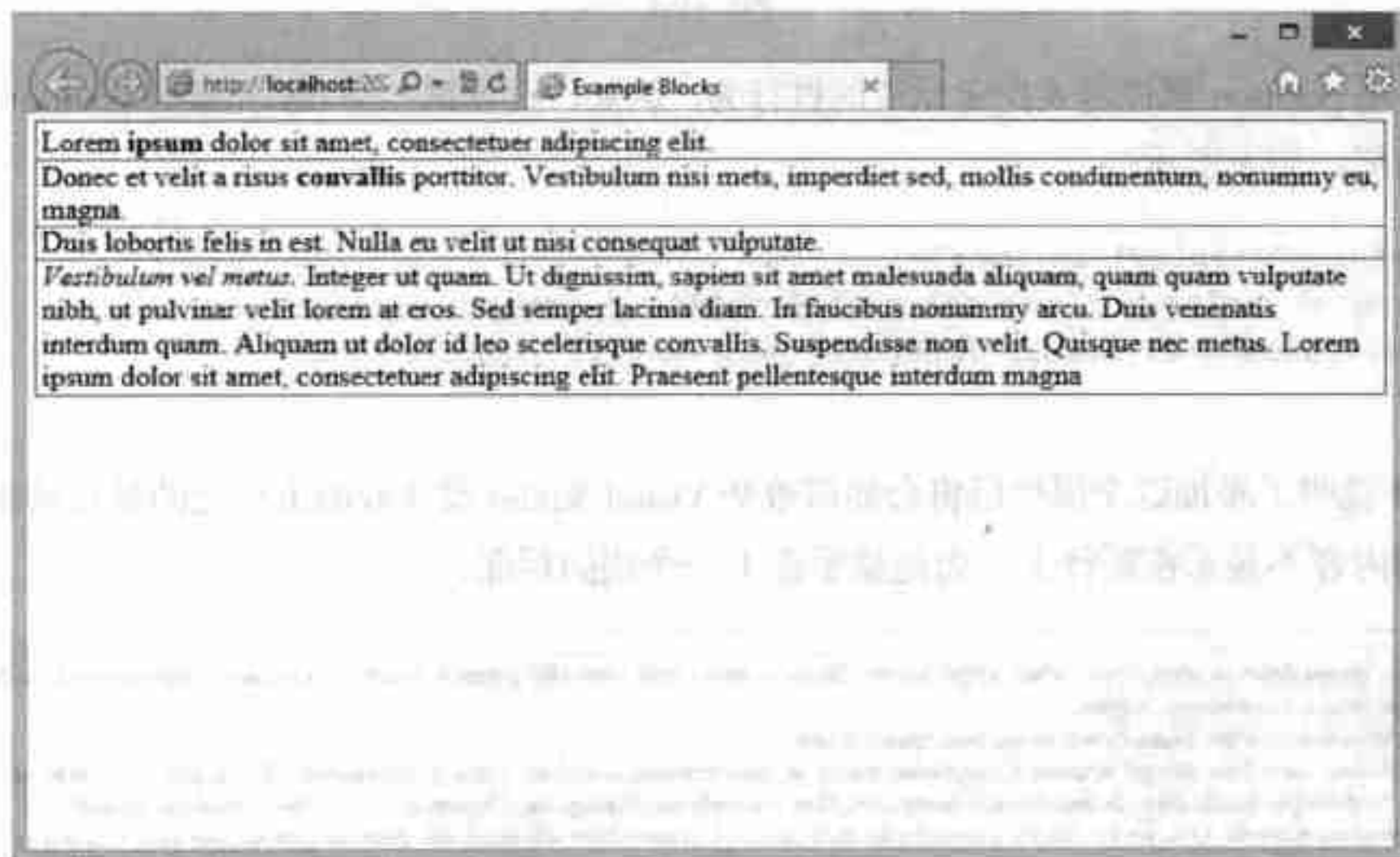


图 2-16

相对定位机制

使用相对定位机制，最初仍然使用一般布局方式来定位元素。首先定位包围框，再根据偏移属性 `top`、`bottom`、`left` 和 `right` 移动该框。图 2-17 显示了与前面相同的内容，但现在第三个块框设置

为使用相对定位机制。Visual Studio 可以提供块的定位线，显示其顶部的偏移量是根据块的正常顶部位置计算出来的，而左边的偏移量是根据正常的左边位置计算出来的。Visual Studio 甚至允许抓取元素的标签标记，在设计界面上拖动它们，从而可视化地定位块。

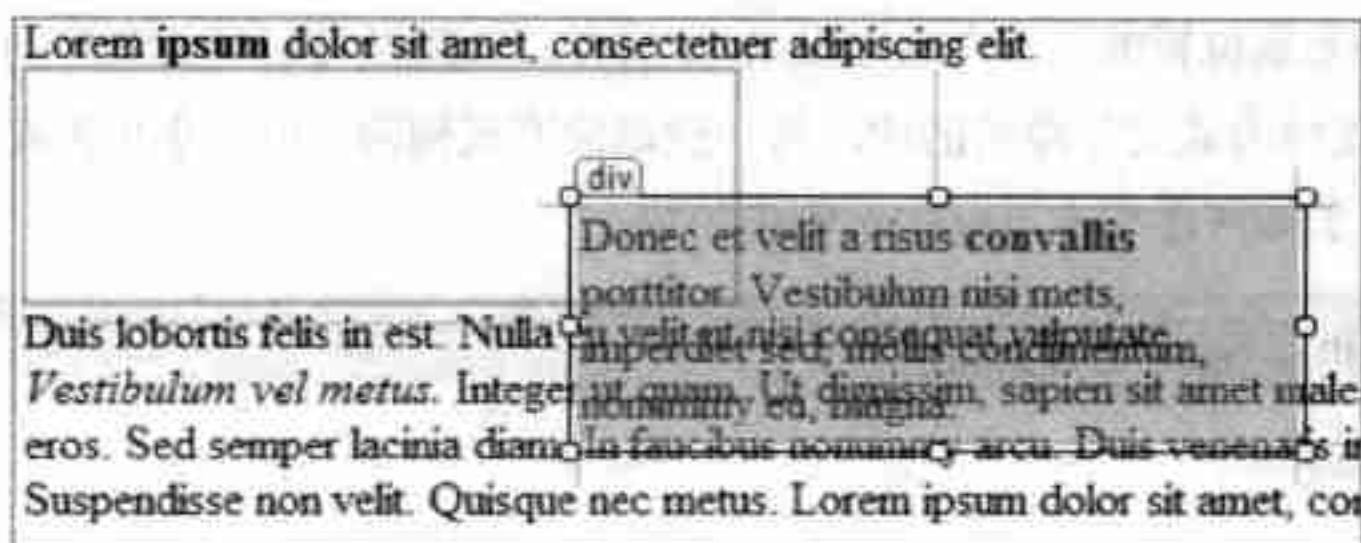


图 2-17

在设计界面上定位元素时，元素的 top 和 left 值会更新。最后，元素如下所示：

```
<div style="position: relative; top: 214px; left: 62px;
width: 239px; height: 81px">
  Donec et velit a risus <strong>convallis</strong> porttitor. Vestibulum
  nisi metus, imperdiet sed, mollis condimentum, nonummy eu, magna.
</div>
```

如果使用相对定位机制，并指定左边和右边的偏移量，一般会忽略右边的偏移量。

绝对定位机制

绝对定位机制类似于相对定位机制，但元素的偏移位置是根据它们在一般定位机制中的位置来计算的，而偏移量则是根据距离元素最近的绝对定位祖先元素的位置来计算的。如果不存在祖先元素，就把浏览器窗口作为祖先元素。

图 2-18 说明了使用绝对定位机制的块如何显示在 Visual Studio 设计界面上。可以看出，与前面相对定位的元素的显示不同，这次定位线延伸到设计界面的边界处，这是因为块使用浏览器窗口来计算其偏移量。

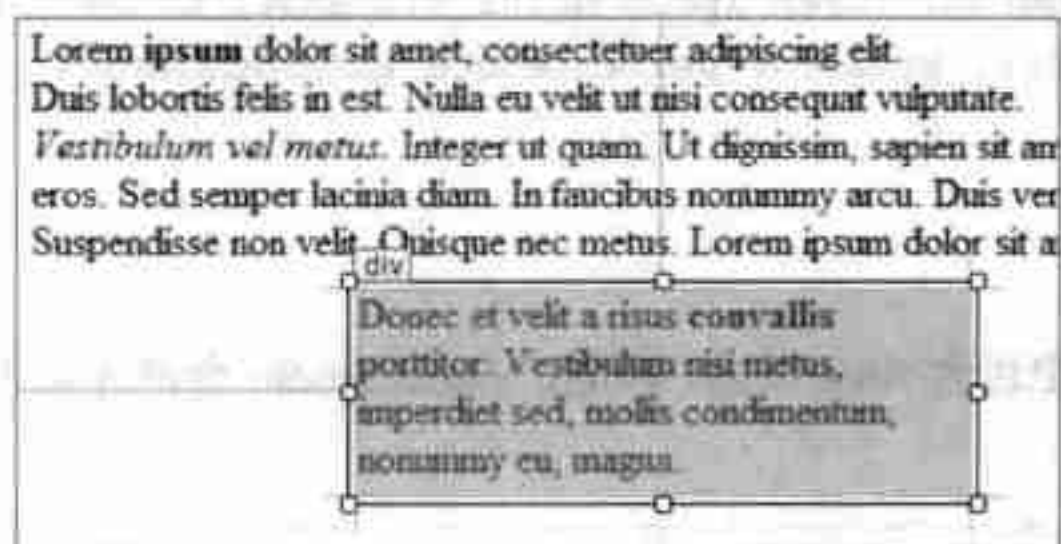


图 2-18

与相对定位的块一样，也可以使用元素的标签标记在页面上定位元素，Visual Studio 会自动更新偏移值。图 2-18 中的块会输出如下元素：

```
<div style="position: absolute; top: 180px; left: 94px;
width: 239px; height: 81px;">
  Donec et velit a risus <b>convallis</b> porttitor. Vestibulum
```

```
nisi metus, imperdiet sed, mollis condimentum, nonummy eu, magna.
</div>
```

元素的浮动

使用 CSS 控制元素位置的另一个选项是 `float` 属性, 该属性可以使元素浮动在块的左边界或右边界上。采用垂直定位方式定位浮动的块, 与一般定位方式相同, 但在水平方向上可以左右移动。程序清单 2-14 演示了如何浮动本节前面示例中的块。

程序清单 2-14 把一个块元素浮动到右边

```
<!DOCTYPE html>
<html>
<head>
  <title>Box Elements</title>
</head>
<body>
  <div id="asdas" class="werwer">
    Lorem ipsum dolor sit amet, consectetur adipiscing elit.
  </div>
  <div style="float: right; width: 236px;">
    Donec et velit a risus <strong>convallis</strong> porttitor.
    Vestibulum nisi metus, imperdiet sed, mollis
    condimentum, nonummy eu, magna.
  </div>
  <div>
    Duis lobortis felis in est. Nulla eu velit ut nisi consequat.
  </div>
  <div>
    Sit amet malesuada aliquam, quam quam vulputate nibh, ut
    pulvinar velit lorem at eros. Sed semper lacinia diam. In
    faucibus nonummy arcu. Duis venenatis interdum quam. Aliquam
    ut dolor id leo scelerisque convallis. Suspendisse non velit.
    Quisque nec metus. Lorem ipsum dolor sit amet, consectetur
    adipiscing elit. Praesent pellentesque interdum magna.
  </div>
</body>
</html>
```

将块修改为在其样式中包含 `float` 属性。因此, Visual Studio 会把元素正确定位到页面的最右端, 如图 2-19 所示。

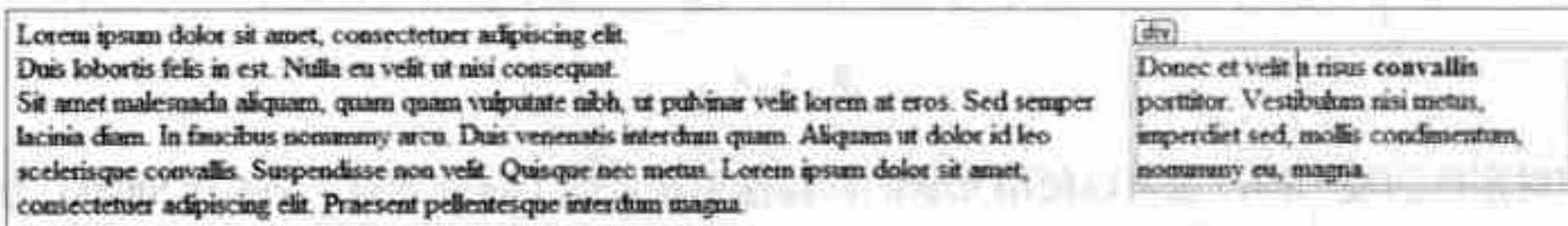


图 2-19

3. !important 特性

如本章前面所述,浏览器选择把最接近的样式应用于元素,这表示可以重写其他已应用的样式的属性。与 CSS 中的许多其他规则相同,CSS 也提供了一种机制来绕过这个规则,该机制称为 !important 特性。应用了这个特性的属性可以避免其他 CSS 规则重写其值。程序清单 2-11 说明了如何重写 font-family 属性。可以修改这个示例以使用 !important 特性,从而了解其工作原理,如程序清单 2-15 所示。

程序清单 2-15 使用 !important 特性控制样式的重写

```
<!DOCTYPE html>
<html>
<head>
  <title>Box Elements</title>
  <style type="text/css">
    p {
      font-family: Arial !important;
    }
    p {
      font-family: 'Courier New';
    }
  </style>
</head>
<body>
  <p>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit.
  </p>
</body>
</html>
```

在这个例子中,没有使用 Courier New 字体显示段落,而是使用 Arial 字体,因为此处使用 !important 特性标记该字体。

2.3.5 CSS3 中的新功能

如前所述,CSS3 规范已推出了几个模块。其中的许多模块已经在大多数现代浏览器中得以实现。将 CSS3 和 HTML5 合并使用,就允许 Web 开发人员实现更高级的设计,而无需 ActiveX 插件或 JavaScript 库。

多年来,开发人员必须添加浏览器专用的 CSS,才能完成特定的任务。例如,要在大多数浏览器中给元素添加不透明度,可以使用如下 CSS:

```
opacity:.8;
```

但是,IE 不能用指定的不透明度显示该元素。要为 IE 用户实现这个效果,开发人员需要使用:

```
-ms-filter:"progid:DXImageTransform.Microsoft.Alpha(opacity=80)";
filter:alpha(opacity=80);
```

CSS3 的目标之一是帮助迁移浏览器专用的 CSS。尽管仍有几个浏览器专用的元素,但许多浏览器都同意,在正式化时会遵循 CSS3 规范。



CSS3 有许多功能和样式规则。本章会介绍一些新功能和规则，但读者最好在互联网上搜索 CSS3，以了解其他功能和规则。

1. 使用新的边框功能

CSS3 中最常见的功能是实现边框模块。过去，开发人员需要使用多个 div 元素或圆角图像来实现圆角。在 CSS3 中，只需添加几个样式原则，就可以实现圆角：

```
div {
    border-radius: 15px;
    box-shadow: 5px 5px rgba(0,0,0,0.2);
}
```

除了添加 15 像素的圆角外，div 元素还有 5 像素的阴影。程序清单 2-16 进一步改进了这个功能，它仅在右下角定义了 border-radius，并在底部放置了一块内部阴影，生成的效果如图 2-20 所示。

程序清单 2-16 使用 CSS3 边框的改进规则

```
<!DOCTYPE html>
<html>
<head>
    <title>CSS3 Borders</title>
    <style type="text/css">
        .pageFeel {
            padding: 5px;
            height: 240px;
            border: 1px solid #cccccc;
            background-color: #eeeeee;
            width: 240px;
            -moz-border-radius-bottomright: 50px 25px;
            border-bottom-right-radius: 50px 25px;
            -moz-box-shadow: inset -5px -5px 5px #aaa;
            -webkit-box-shadow: inset -5px -5px 5px #aaa;
            box-shadow: inset -3px -3px 20px #aaa;
        }
    </style>
</head>
<body>
    <div class="pageFeel">
        Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    </div>
</body>
</html>
```

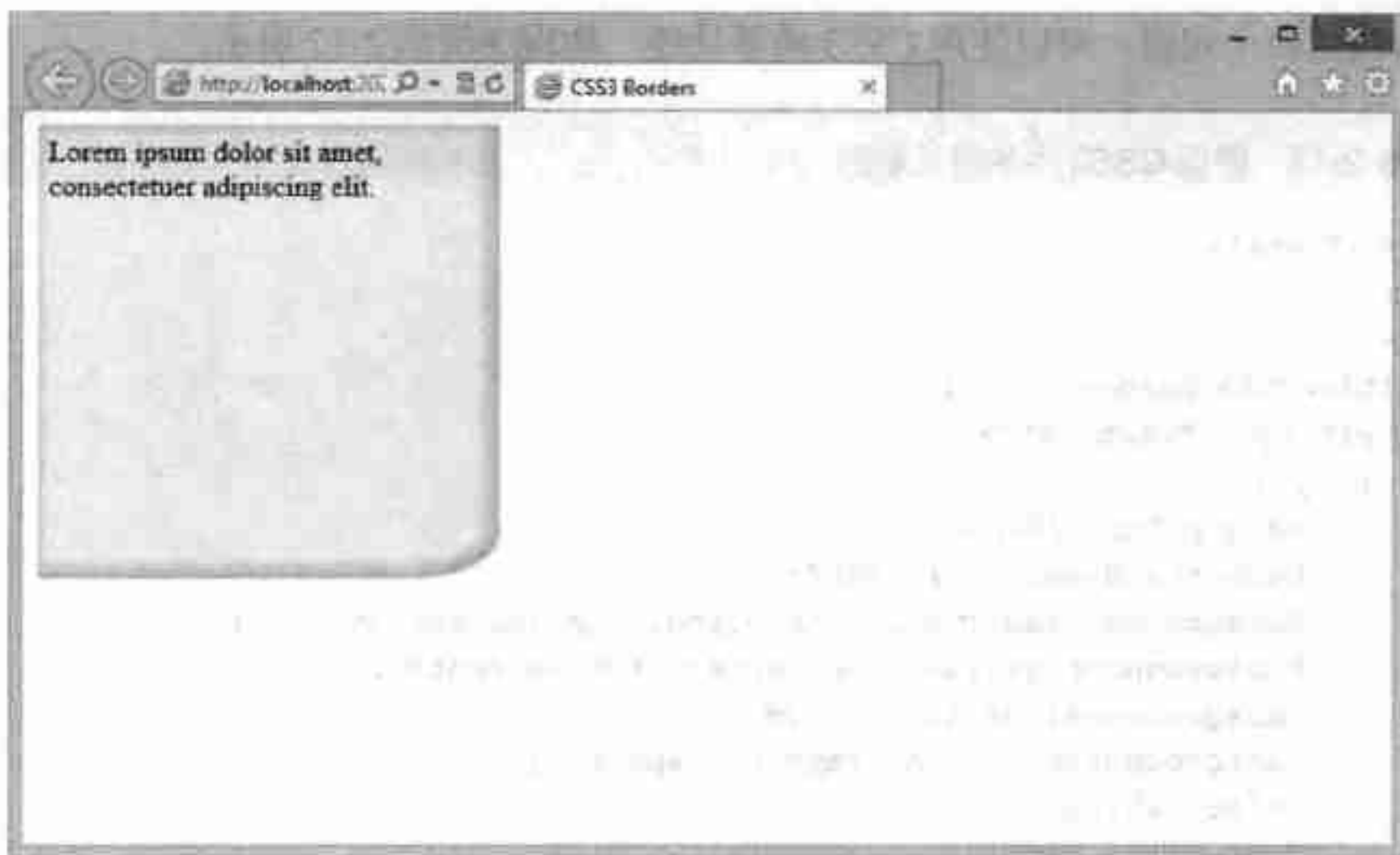


图 2-20

注意,对于 border-radius 规则,有一条用-moz-前缀指定的特殊规则,允许提供对 Mozilla FireFox 3.5 的支持。对于 box-shadow 规则,除了用-moz-前缀指定的规则之外,还有一条用-webkit-前缀指定的规则,该规则允许提供对 Safari 和 Google Chrome 的支持。

2. 改进的背景

以前,开发人员要面对的另一样式问题是使用多个背景或给背景分层。这个问题的常见解决方法是,让多个 div 元素有不同的 z-index 值。在 CSS3 中,多个背景可以在同一条样式规则中应用。

开发人员的另一个常见问题是为站点创建背景图像,可以根据不同的屏幕尺寸平铺或缩放。这常常要使用独立的 CSS 文件来捕获屏幕的分辨率。否则,就必须设计一幅图像,进行平铺,而不让用户注意到图像的边缘。图 2-21 是个例子。

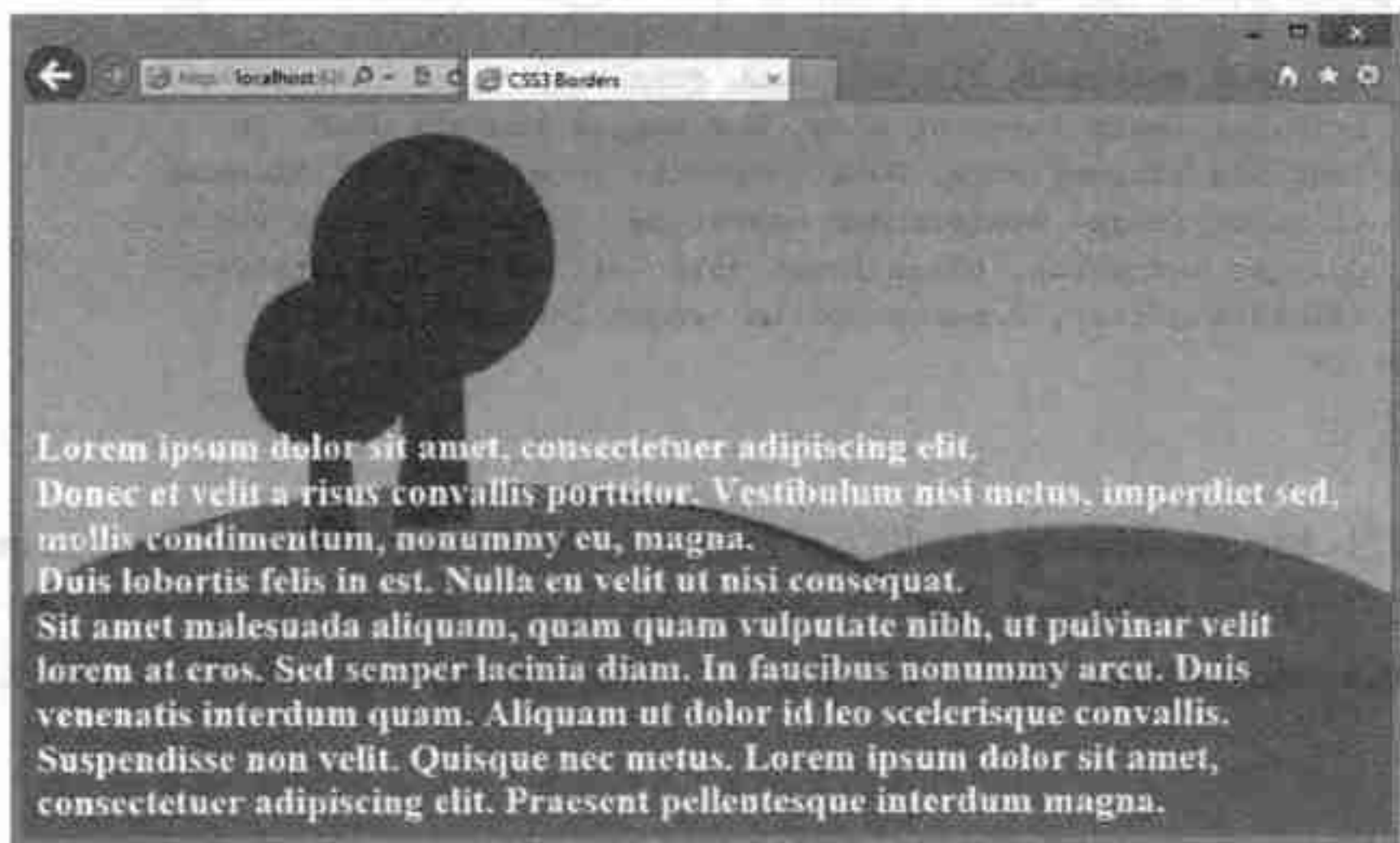


图 2-21

为了解决这两个问题，可以使用 CSS3 的新功能，如程序清单 2-17 所示。

程序清单 2-17 使用 CSS3 中的背景规则

```
<!DOCTYPE html>
<html>
<head>
  <title>CSS3 Borders</title>
  <style type="text/css">
    body {
      margin-top: 200px;
      background-color: #33BEF2;
      background-image: url(clouds.png), url(background.png);
      background-position: top center, bottom center;
      background-size: auto, cover;
      background-repeat: no-repeat, repeat-y;
      color: white;
      font-size: 18pt;
      font-weight: bold;
    }
  </style>
</head>
<body>
  <div>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit.
  </div>
  <div>
    Donec et velit a risus <strong>convallis</strong> porttitor.
    Vestibulum nisi metus, imperdiet sed, mollis
    condimentum, nonummy eu, magna.
  </div>
  <div>
    Duis lobortis felis in est. Nulla eu velit ut nisi consequat.
  </div>
  <div>
    Sit amet malesuada aliquam, quam quam vulputate nibh, ut
    pulvinar velit lorem at eros. Sed semper lacinia diam. In
    faucibus nonummy arcu. Duis venenatis interdum quam. Aliquam
    ut dolor id leo scelerisque convallis. Suspendisse non velit.
    Quisque nec metus. Lorem ipsum dolor sit amet, consectetur
    adipiscing elit. Praesent pellentesque interdum magna.
  </div>
</body>
</html>
```

可以看出，`background-image` 规则指定了两幅图像。每幅图像都单独显示。`background.png` 文件显示在页面的背景上，`clouds.png` 放在背景的上部。`background-size` 规则也有两个值，第一个值应用于第一个 `background-image`，第二个值应用于第二个 `background-image`，如图 2-22 所示。

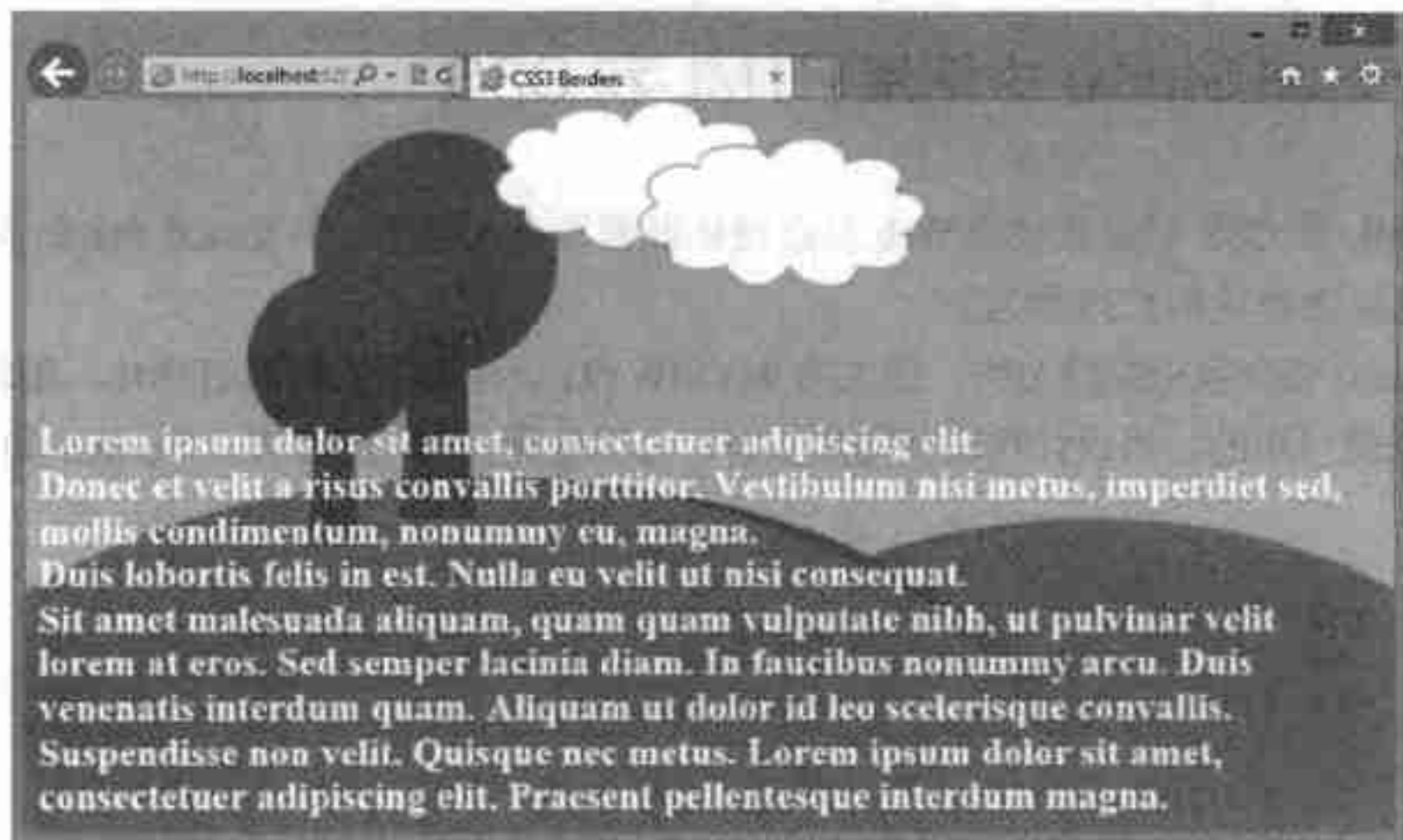


图 2-22

3. 给文本添加阴影

在 CSS2 推出之前,给文本添加阴影的最佳方式是生成一幅图像来获得需要的结果。CSS2 引入了一条可以在文本上指定的新规则 `text-shadow`。尽管这条规则是在 CSS2 中引入的,但现代浏览器并没有广泛采用。该规则接收 4 个值:

```
text-shadow: 5px 5px 2px #000000;
```

第 1 个值是向右偏移的像素数,第 2 个值是向左偏移的像素数,第 3 个值把像素数设置为模糊度,最后一个值设置阴影的颜色。

2.3.6 HTML 和 CSS 的兼容

前面讨论的许多内容都不适用于 IE8 等旧浏览器,而 IE8 是运行 Windows 7 的大多数计算机所安装的 Web 浏览器。为了帮助开发人员确保兼容性,ASP.NET 团队在所有新的 ASP.NET 项目模板中包含了 JavaScript 库 `Modernizr.js`。尽管这个 JavaScript 库没有添加旧浏览器不支持的 HTML5 元素,但它有助于缓解新旧浏览器之间的兼容性问题。

如果开发人员希望添加更多的 HTML5 支持,使页面显示在旧的 IE 浏览器中,就应使用 `HTML5Shiv.js` 文件,它不是 `Modernizr` 库的一部分。这个文件也称为 HTML5 shim 文件。



要了解 `Modernizr` 库的更多信息,可访问 <http://modernizr.com/>。如果已经有了网站或 Web 应用程序,就可以使用 NuGet 安装 `Modernizr` 来更新应用程序,使之包含 `Modernizr`。附录 G 详细介绍了 NuGet。要了解 HTML5 shim 文件的更多信息,可以在线搜索 `HTML5shiv` 或 `HTML5shim`。

2.4 在 Visual Studio 中使用 HTML 和 CSS

使用 HTML 和 CSS 创建漂亮的 Web 站点设计很容易令人畏缩, 但 Visual Studio 提供了许多工具, 帮助简化页面布局和 CSS 管理。

Visual Studio 包含我们很熟悉的、强大的 WYSIWYG 设计界面来编辑 HTML。编辑 HTML 时, Format 菜单就是可用的。当设计视图获得焦点时, 可以使用另外两个菜单: Format 和 Table, 如图 2-23 所示。

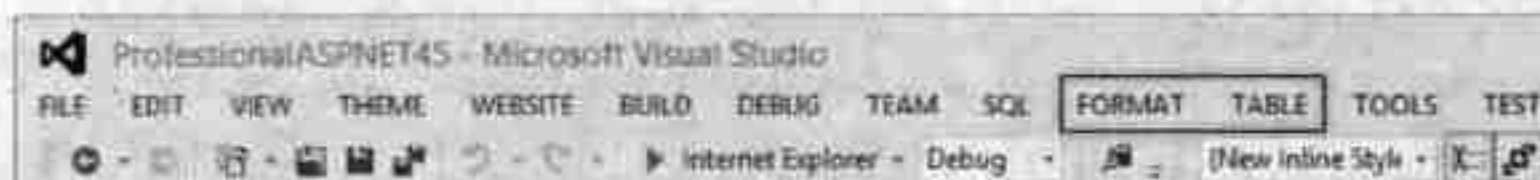


图 2-23

很容易猜出, Table 菜单包含的工具可以在 Web 页面上插入、删除、选择和修改 HTML 表。从 Table 菜单中选择 Insert Table 选项, 会打开如图 2-24 所示的 Insert Table 对话框, 使用该对话框可以方便地指定表的属性。我们可以定义表的行数和列数, 单元格的内边距、间距以及边框属性。单击 OK 按钮后, Visual Studio 就会在 Web 页面上自动生成相应的 HTML 表。



图 2-24

在 Web 页面上选择已有的表时, Table 菜单就允许插入和删除表中的行、列和单元格。Modify 菜单项还可以把已有的单元格拆分为两个单元格, 把两个单元格合并为一个单元格, 配置行和列的大小。

Format 菜单包含基本的元素格式化选项, 例如访问元素的 CSS 类, 设置前景色和背景色、字体和位置, 把内容转换为不同类型的列表。

2.4.1 在 Visual Studio 中使用 CSS

Visual Studio 提供了许多工具, 使得 CSS 的使用更为方便。要为 Web 页面创建新样式, 只需从 Format 菜单中选择 New Style 选项即可, 该操作将打开如图 2-25 所示的 New Style 对话框。



图 2-25

使用这个对话框可以很快地创建新样式。首先，从 Selector 下拉列表中选择要创建的选择器类型，该列表包含所有的元素类型。如果要创建类选择器或 ID 选择器，只需在 Selector 组合框中输入样式名称即可。

其次，从 Define in 组合框中选择创建样式的位置。可以选择 Current Page 选项以创建内部样式表，选择 New Style Sheet 选项以创建新的外部样式表，选择 Existing Style Sheet 选项以把样式插入已有的样式表文件中。如果选择了 New Style Sheet 选项或 Existing Style Sheet 选项，就需要为 URL 组合框提供值。

最后，为了能把样式应用到在当前设计界面中选择的元素上，需要选中“Apply new style to document selection”复选框。

输入要使用的选择器并选择定义样式的位置后，就可以开始设置样式的属性。只需从 Category 列表框中选择属性类别，设置属性值即可。Preview 区域会给出新样式的实时预览。另外，Description 区域显示了 Visual Studio 创建的属性语法。单击 OK 按钮以关闭对话框。

为应用程序创建了样式后，还需要对它们进行管理并把它们应用到应用程序的元素上。为此，Visual Studio 引入了 3 个工具窗口，用于管理样式表、把样式应用于元素以及查看应用于元素的样式属性。

Manage Styles 工具窗口

此处探讨的第一个工具是 Manage Styles 工具窗口，在 View 菜单的 CSS Styles 子菜单中选择 Manage Styles 命令，就可以打开该窗口。这个工具窗口如图 2-26 所示，通过该窗口可以查看当前在 Visual Studio 中打开的 Web 页面上能使用的所有样式。

如果查看这个工具的内容，就会发现其顶部包含两个重要的链接：New Style 和 Attach Style Sheet。第一个链接可以打开 New Style 对话框，创建新的 CSS 样式，如本节前面所述。第二个链接可以把新样式表导入 Web 页面。使用这个选项把样式表关联到 Web 页面上，会使 Visual Studio 在 Web 页面上插入<link>标记。

必须小心地安排 link 标记和样式块的顺序，确保样式得到正确的应用。

该工具窗口的下一部分是预览区域，可以查看每个样式的实时预览。

最后，该工具窗口的底部显示了页面可用的所有样式。根据选择器类型，使用彩色的项目列表对样式进行了彩色编码：蓝色表示类型选择器，绿色表示类选择器，红色表示 ID 选择器。使用包围彩色项目列表的灰色圆圈包围页面中使用的样式。如果 Web 页面包含多个链接的样式表或内联样式表，这些样式会组合在一起，以便于确定在何处定义样式。

另外，如图 2-26 所示，该工具窗口还允许通过 CSS Imports 语句查看与 Web 页面关联的样式表。在图 2-26 中展开 light.css 节点，就会列出该样式表包含的所有样式。

Apply Styles 工具窗口

在 Visual Studio 中，帮助使用 CSS 的第二个工具是 Apply Styles 工具窗口。与 Manage Styles 工具窗口一样，Apply Styles 工具窗口也提供了一种简单的方式，可以查看应用程序中可用的 CSS 样式，并把这些样式应用于 Web 页面中的元素上。在这个工具窗口中，可以把 CSS 文件关联到 Web 页面上，使外部的 CSS 样式可用，选择要应用的页面样式或要从元素中删除的页面样式，以及修改样式等。与其他 CSS 工具窗口一样，Apply Styles 工具窗口也根据 CSS 继承顺序显示可用的样式，首先显示外部样式，然后显示页面样式部分，最后显示内联样式。Apply Styles 工具窗口还可以根据当前选择的元素，显示应用程序中可以应用于该元素类型的样式。可以根据 CSS 选择器样式来组合样式，使每种选择器类型有不同外观的指示器。

如图 2-27 所示的 Apply Styles 工具窗口显示了锚点标记<a>的可用样式。该工具窗口首先显示了关联文件 layout.css 中的所有样式，接着显示了当前页面中的样式，最后显示了已对该元素应用的内联样式。单击这些部分中的样式，就会把它们应用于元素。

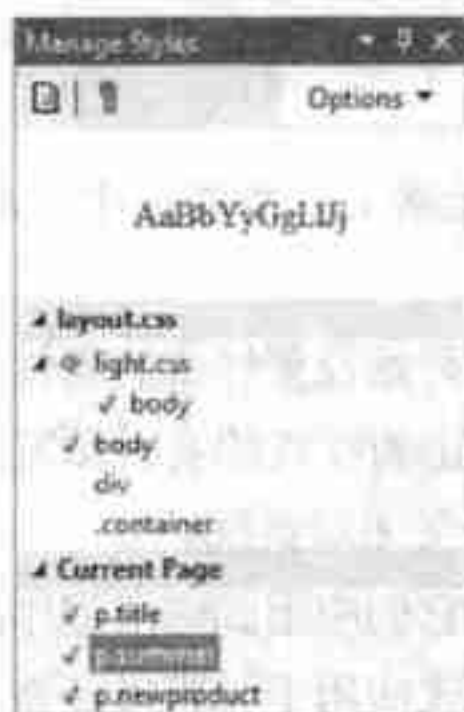


图 2-26

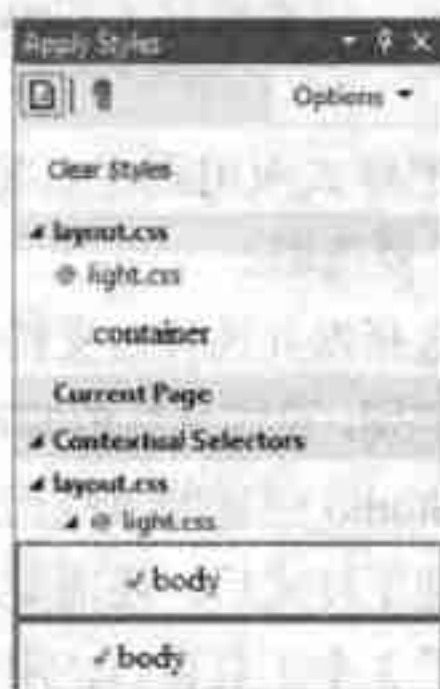


图 2-27

Apply Styles 工具窗口还提供了用于正确应用多个类选择器的 Intelligence 工具(单击列表中的类选择器时按住 Ctrl 键)，但不能选择多个 ID 选择器，因为这会使 CSS 无效。该工具也不允许取消对类型选择器或内联样式的选择。

CSS Properties 工具窗口

最后一个工具是 CSS Properties 工具窗口，如图 2-28 所示。这个方便的工具窗口显示了已应用于当前所选元素的所有 CSS 属性。该工具窗口由两部分组成：Applied Rules 列表和 CSS Properties 网格。

Applied Rules 列表显示了应用于所选元素的所有 CSS 规则。该列表自动排序，显示已应用规则

的继承链,最外层的规则在顶部,最内层的规则在底部。这表示包含在外部 CSS 文件中的规则自动排在列表的顶部,内联样式排在底部。可以单击列表中的每条规则,修改显示在 CSS Properties 网格中的属性。

CSS Properties 网格类似于标准的.NET 属性网格,显示了元素可用的所有 CSS 属性,其中设置了值的属性显示为粗体。另外,还可以在 CSS Properties 网格中直接给 CSS 规则设置属性值。在 CSS Properties 工具窗口中还有一个 Summary 按钮,可以改变 CSS Properties 网格的显示,只列出已设置了值的属性。这是非常有用的功能,因为 HTML 元素可能包含非常多的 CSS 属性。

因为 CSS 还包含属性继承的概念,而在标准的.NET 对象中没有这个概念,所以 CSS Rules 列表和 CSS Properties 网格有助于全面理解应用于元素的某个属性值是在哪里定义的。在 CSS Rules 列表中单击某个规则,CSS Properties 网格就会更新以显示该规则的属性。但要注意,某些属性有红色的删除线,如图 2-29 所示。



图 2-28

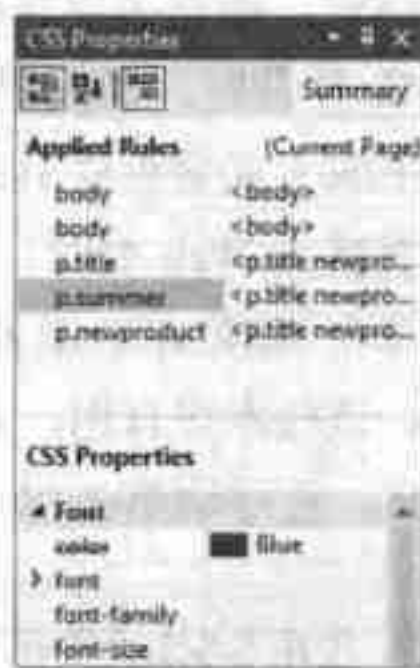


图 2-29

属性的删除线表示,该属性的值被更接近元素的规则重写。

2.4.2 指定 ASP.NET 控件的样式

因为 ASP.NET 控件只是简单地显示 HTML 标记,所以很容易使用 CSS 给它们指定样式。实际上,这些控件默认使用内联 CSS 样式,查看标准的 ASP.NET 按钮控件就可以看出这一点。给 ASP.NET 控件(如 Button 控件)指定样式的标准方法是为控件中与样式相关的属性指定值,如下所示:

```
<asp:Button ID="Button1" runat="server" BackColor="#3333FF"
    BorderColor="Silver" BorderStyle="Double" BorderWidth="3px"
    Font-Bold="True" Font-Size="Large" ForeColor="White"
    Text="Button" />
```


ASP.NET 处理包含这个控件的 Web 页面时，会把按钮转换为标准的 HTML input 标记，并把已设置的样式属性转换为 CSS 样式，将它们应用于 input 标记。该按钮显示的 HTML 和 CSS 如下：

```
<input type="submit" name="Button1" value="Button" id="Button1"
    style="color:White;background-color:#3333FF;border-color:Silver;
    border-width:3px;border-style:Double;font-size:Large;font-weight:bold;" />
```

直接在 ASP.NET 控件上设置样式属性是为应用程序中的 ASP.NET 控件指定样式的一种快捷方法。另外，这些属性都是控件上的标准属性，因此也可以在运行期间使用代码设置它们：

```
protected void Page_Load(object sender, EventArgs e)
{
    this.Button1.BackColor =
        System.Drawing.ColorTranslator.FromHtml("#3333FF");
    this.Button1.BorderColor = System.Drawing.Color.Silver;
    this.Button1.BorderStyle= BorderStyle.Double;
    this.Button1.BorderWidth = Unit.Pixel(3);
    this.Button1.Font.Bold=true;
    this.Button1.Font.Size=FontUnit.Large;
    this.Button1.ForeColor=System.Drawing.Color.White;
}
```

使用属性设置样式信息很简单、方便，但存在一些缺点，尤其是在对较多的相同控件使用相同的技术时，缺点就更明显。首先，使用内联样式将很难在较高的、较抽象的层面上控制 Web 站点的样式。如果希望 Web 站点上的每个按钮都使用指定的样式，那么一般需要在整个站点上手动为每个按钮控件设置该样式。主题可以帮助完成这项任务，但并不总是有效，尤其是在混合了 ASP.NET 控件和标准 HTML 控件时，情况更是如此。

其次，对于生成了大量重复 HTML 的控件(如 GridView)，在每次遍历 HTML 时给每个元素使用内联样式会显著增大 Web 页面。

幸运的是，每个 ASP.NET 服务器控件都有 CssClass 属性。这个属性可以为控件提供一条或多条类选择器规则。该技术很有用，并且通常好于让控件显示内联样式，但仍然需要熟悉控件在运行期间显示的 HTML。程序清单 2-18 说明了如何使用 CssClass 属性设置 ASP.NET Button 控件的样式。

程序清单 2-18 使用 CSS 设置标准 ASP.NET Button 控件的样式

```
<%@ Page Language="C#" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <link href="SpringStyles.css" rel="stylesheet" type="text/css" />
    <style>
        .search
        {
            color:White;
            font-weight:bold;
            background-color:Green;
        }
    </style>
```



```

</head>
<body>
  <form id="form1" runat="server">
    <asp:Button ID="Button1"
      CssClass="search" runat="server" Text="Button" />
  </form>
</body>
</html>

```

在这个例子中，对 Button 控件应用了 search 类。

2.4.3 VS 2012 中对 HTML 和 CSS 的改进

在 VS 的每个版本中，Microsoft 都改进了软件中的 HTML 开发功能。VS 2012 也不例外。在 VS 2012 中，给 HTML 添加的几个改进如下：

- WAI-ARIA 支持，它允许元素包含角色属性和其他以 aria 开头的属性。
- 在其他标记重命名时，可以匹配 HTML，并重命名 ASP.NET 服务器控件标记。
- HTML5 元素现在可以用作 VS 中的片段。
- 更新了 ASP.NET 服务器控件 TextBox，它现在支持新的 HTML5 输入类型，例如 email 和 datetime。
- ASP.NET 服务器控件 FileUpload 支持在现代浏览器中上传多个文件。

在 VS 2012 中，给 CSS 添加的几个改进如下：

- 改进的 IntelliSense 只显示可用的项以及新的 CSS3 规则，例如以-moz-和-webkit 开头的浏览器专用规则。
- 分层缩进，显示 CSS 规则的逻辑分组，如图 2-30 所示。
- 注释(使用快捷键)、区域和代码片段可用于 CSS，就像它们用于源代码一样。
- 新的调色板帮助选择需要的颜色，如图 2-31 所示。

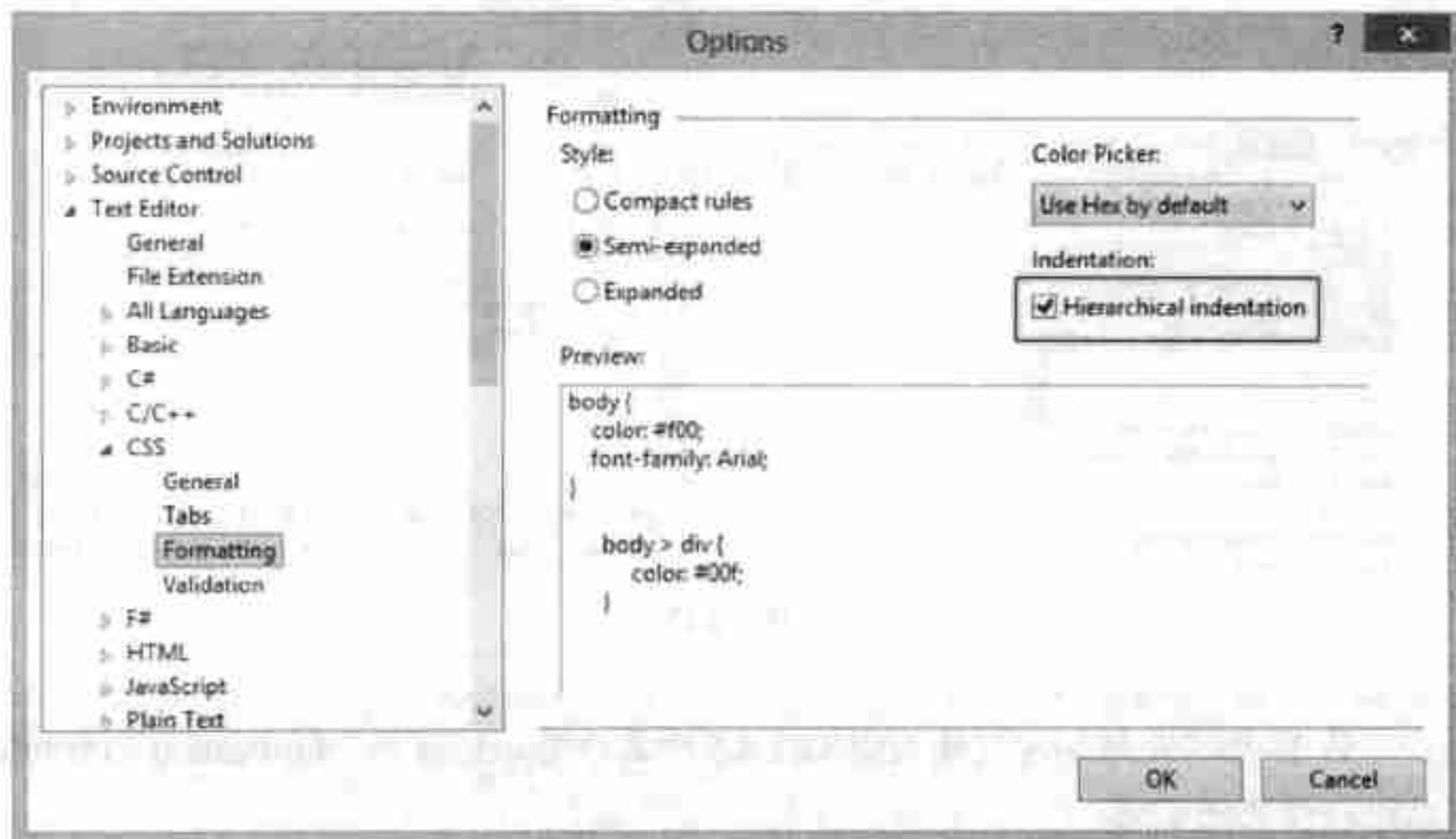


图 2-30

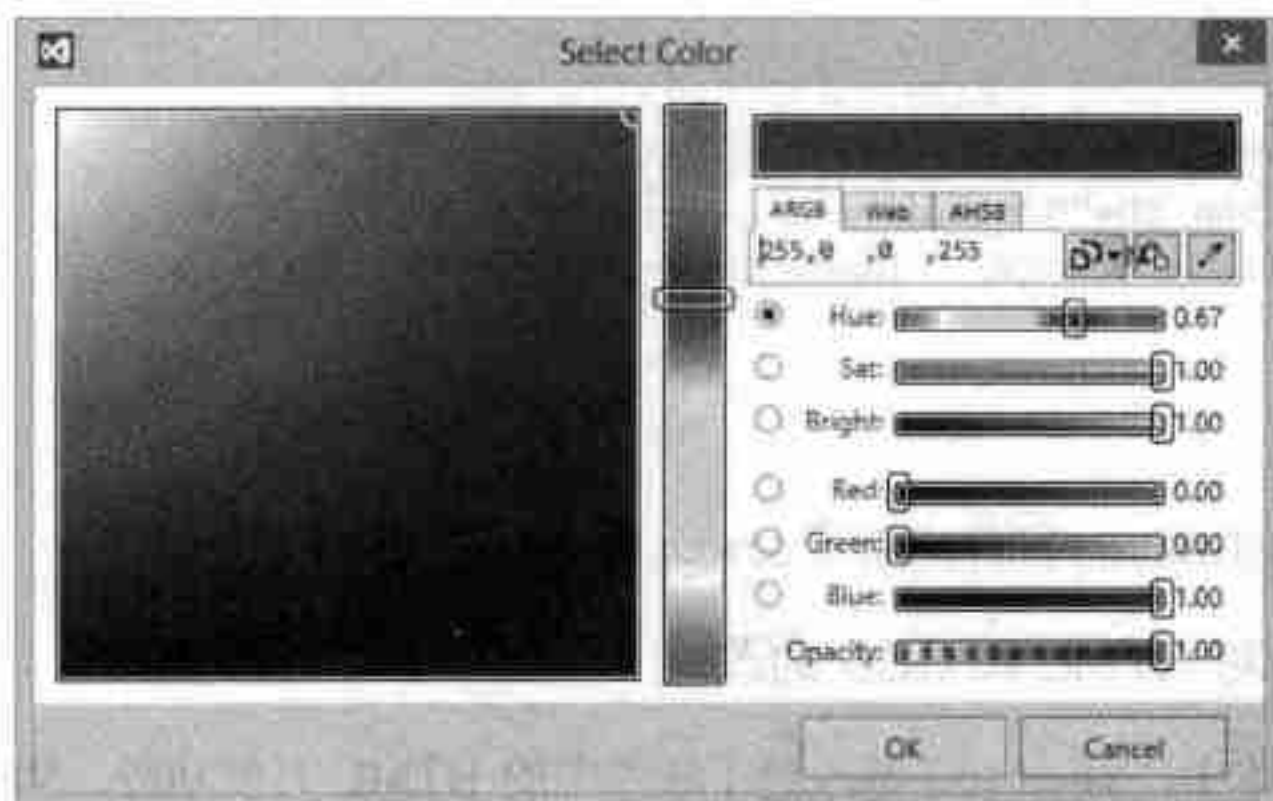


图 2-31

2.4.4 Page Inspector

Page Inspector 是 VS 2012 附带的新工具。该工具允许开发人员查看相关服务器端代码的标记。换言之,选择 Web 窗体或 MVC 页面中的控件时,在预览窗口中就会突出显示对应的标记,如图 2-32 所示。目前,Page Inspector 使用 IE9 或更新版本来显示标记。



图 2-32



第 28 章将介绍如何利用 ASP.NET 4.5 中名为 Bundling 和 Minification 的新功能,以进一步优化 CSS。

2.5 本章小结

CSS 是给 Web 站点添加样式的一种极佳方式，是一种强大而方便的机制，允许为 Web 站点创建复杂的样式和布局，尤其是在与 HTML5 一起使用时，就更是如此。完整讨论 CSS 需要的时间和篇幅远远超出了这里的范围，因此本章仅介绍了 HTML 和 CSS 的一些基本概念，以及 Visual Studio 2012 和 ASP.NET 4.5 中的新功能。

本章概述了 HTML5，介绍了新元素、特性、特性值和 API，讨论了如何在实际场景中使用新的 HTML5 元素和特性。

接着概述了 CSS，介绍了外部样式表、内部样式表和内联样式表。本章也讨论了 CSS 提供的各种选择器类型，以及 CSS 元素的基本布局和定位，包括框模型如何影响 Web 页面上元素的位置，还学习了 HTML5 和 CSS3 的向后兼容性。

最后，本章复习了 Visual Studio 中便于使用 CSS 的工具，包括 Manager Styles 和 CSS Properties 工具窗口，阐述了如何把 CSS 用于 ASP.NET 服务器控件，还学习了 VS 2012 中 HTML 和 CSS 的几项新功能，以及 Page Inspector。

第 3 章

ASP.NET Web 窗体的结构

本章要点

- 选择应用程序的位置选项和页面的结构选项
- 使用 page 指令、页面事件和应用程序文件夹
- 选择编译选项

从 ASP 3.0 到 ASP.NET 3.0 的进步是革命性的，或者说至少是变化非常大的。现在 ASP.NET 的最新版本 4.5 仍在不断变化。最初，ASP.NET 1.0 的引入基本改变了 Web 站点和应用程序的开发方式，而 ASP.NET 4.5 只是改革了提高工作效率的方式。直到最近，ASP.NET 的主要目标仍是使用尽可能少的代码，建立强大、安全、动态的应用程序。本书涵盖了 ASP.NET 4.5 提供的新功能，同时也讨论 ASP.NET 技术的所有内容。

如果是 ASP.NET 的初学者，那么当使用 ASP.NET 4.5 构建第一组应用程序时，你可能会对 ASP.NET 4.5 提供的海量服务器控件感到惊讶，会对使用一系列数据提供程序更高效地处理数据感到惊奇，还会对可以轻松构建安全措施以及进行个性化配置留下深刻印象。

但 ASP.NET 4.5 的功能绝不仅于此。本章将介绍便于处理 ASP.NET 页面和应用程序的许多功能。在启动项目时，开发人员要进行的第一步是熟悉正在构建的基础架构和定制该基础架构的选项。

3.1 应用程序的位置选项

在 ASP.NET 4.5 中，使用 Visual Studio 2012 可以在映射到 IIS 的虚拟目录下创建应用程序，或者在 IIS 的限制之外创建独立的应用程序。早期的 Visual Studio .NET 2002/2003 IDE 要求开发人员为所有的 Web 应用程序使用 IIS，但 Visual Studio 2008/2010(以及 Visual Web Developer 2008/2010 Express Edition)包含内置的 Web 服务器 Cassini，可以用于开发工作，它类似于过去使用的 ASP.NET Web Matrix。在 Visual Studio 2012 中，内置的 Web 服务器是 IIS Express。



IIS Express 是轻型的自包含 IIS 版本，为开发和 VS 进行了优化。访问 www.iis.net 可以获得 IIS Express 的更多信息。如果按照第 35 章所述下载了 WebMatrix，那么也就安装好了 IIS Express。

下一节将介绍如何使用 Visual Studio 2012 附带的 IIS Express。

3.1.1 使用文件系统(IIS Express)

默认情况下，Visual Studio 2012 使用 IIS Express 构建应用程序。在该 IDE 中选择 File | New | Web Site 命令时，就可以看出这一点。为应用程序提供的位置默认为 C:\Users\JasonGaylord\Documents\Visual Studio 2012\WebSites(假定使用 Windows 8)，如图 3-1 所示，而在 Visual Studio .NET 2002/2003 中使用的是 C:\inetpub\wwwroot\。在 C:\Users\JasonGaylord\Documents\Visual Studio 2012\WebSites(或创建的其他文件夹)中建立与管理的所有站点都默认使用 Visual Studio 2012 中内置的 IIS Express。如果使用 Visual Studio 2012 中内置的 Web 服务器，就不会被锁定在 WebSites 文件夹中，而是可以在系统中创建任意文件夹。

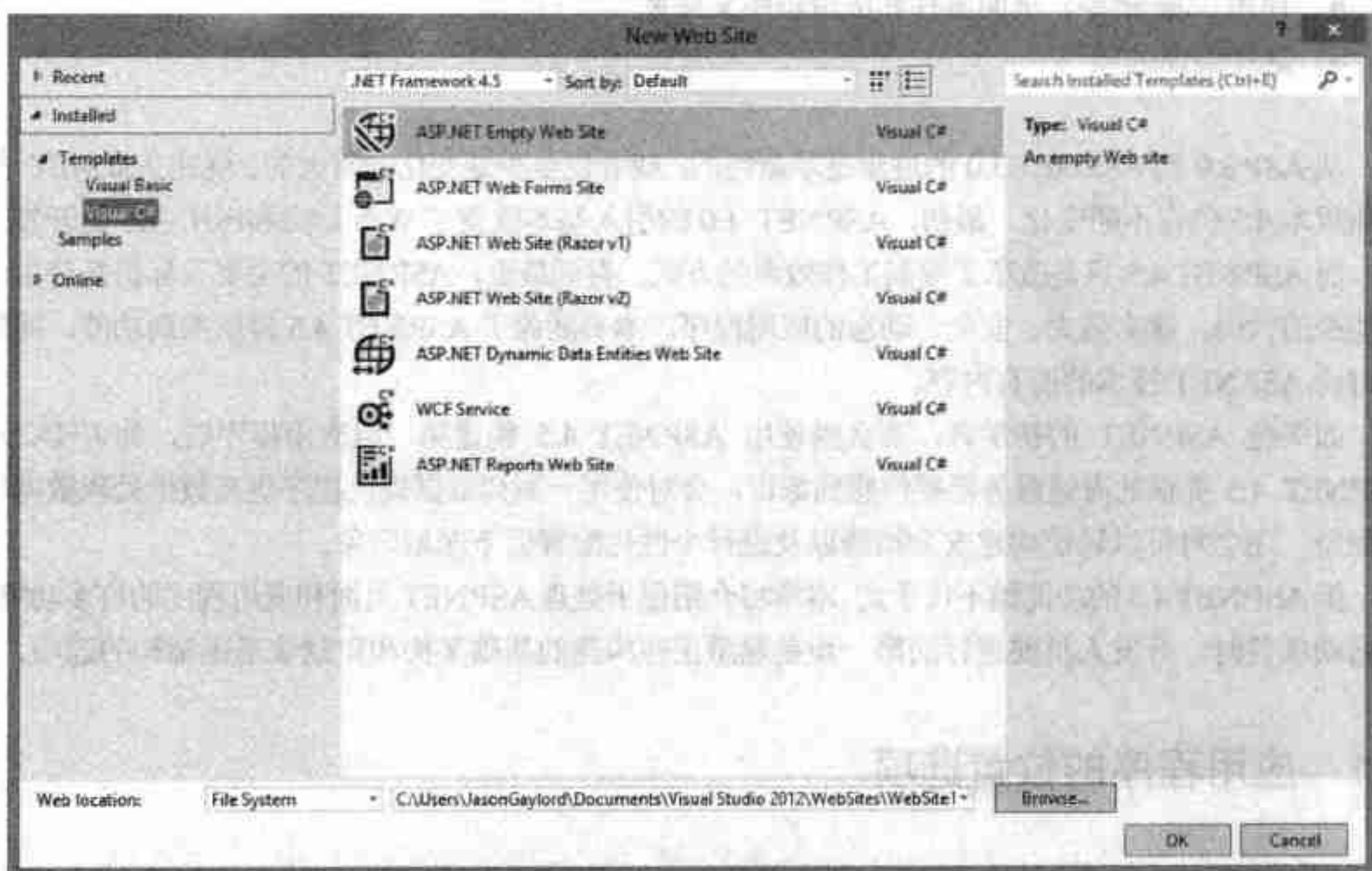


图 3-1

为了改变此默认设置，可以使用一些选项。单击 New Web Site 对话框中的 Browse 按钮，打开 Choose Location 对话框，如图 3-2 所示。



图 3-2

如果继续使用 Visual Studio 2012 提供的内置 IIS Express, 就可以在这个对话框中为 Web 应用程序选择新的位置。要选择新位置, 可以选择一个新文件夹, 把.aspx 页面和其他相关的所有文件都保存到这个目录下。在使用 Visual Studio 2012 时, 可以从这个位置运行应用程序。即使不能访问某台 Web 服务器, 这种处理 ASP.NET 页面的新方式也非常理想, 因为它允许在不带 IIS 完整版本的计算机上创建应用程序。

创建新网站后, 就可以在 VS 属性面板上修改 IIS Express 设置, 如图 3-3 所示。



图 3-3

3.1.2 使用 IIS

在 Choose Location 对话框(如图 3-4 所示)中, 还可以改变应用程序的存储位置和应用程序所使用的 Web 服务器类型。要使用 IIS, 可以在对话框中单击 Local IIS 按钮。这会改变文本区域中的内容, 显示计算机上所有虚拟应用程序根目录的列表。



如果要查看本地的 IIS 实例，那么需要以管理员用户身份运行 Visual Studio。

要为应用程序创建新的虚拟根目录，可突出显示 Default Web Site 选项。在对话框的顶部会显示两个可访问的按钮，如图 3-4 所示。从左向右看，对话框右上角的第 1 个按钮用于创建新站点，把新站点添加到 IIS Express 中。第 2 个按钮用于创建新的 Web 应用程序或虚拟根目录。第 3 个按钮允许为所建立的虚拟根目录创建虚拟目录。最后一个按钮是 Delete 按钮，它允许删除服务器上选中的站点、虚拟目录或虚拟根目录。

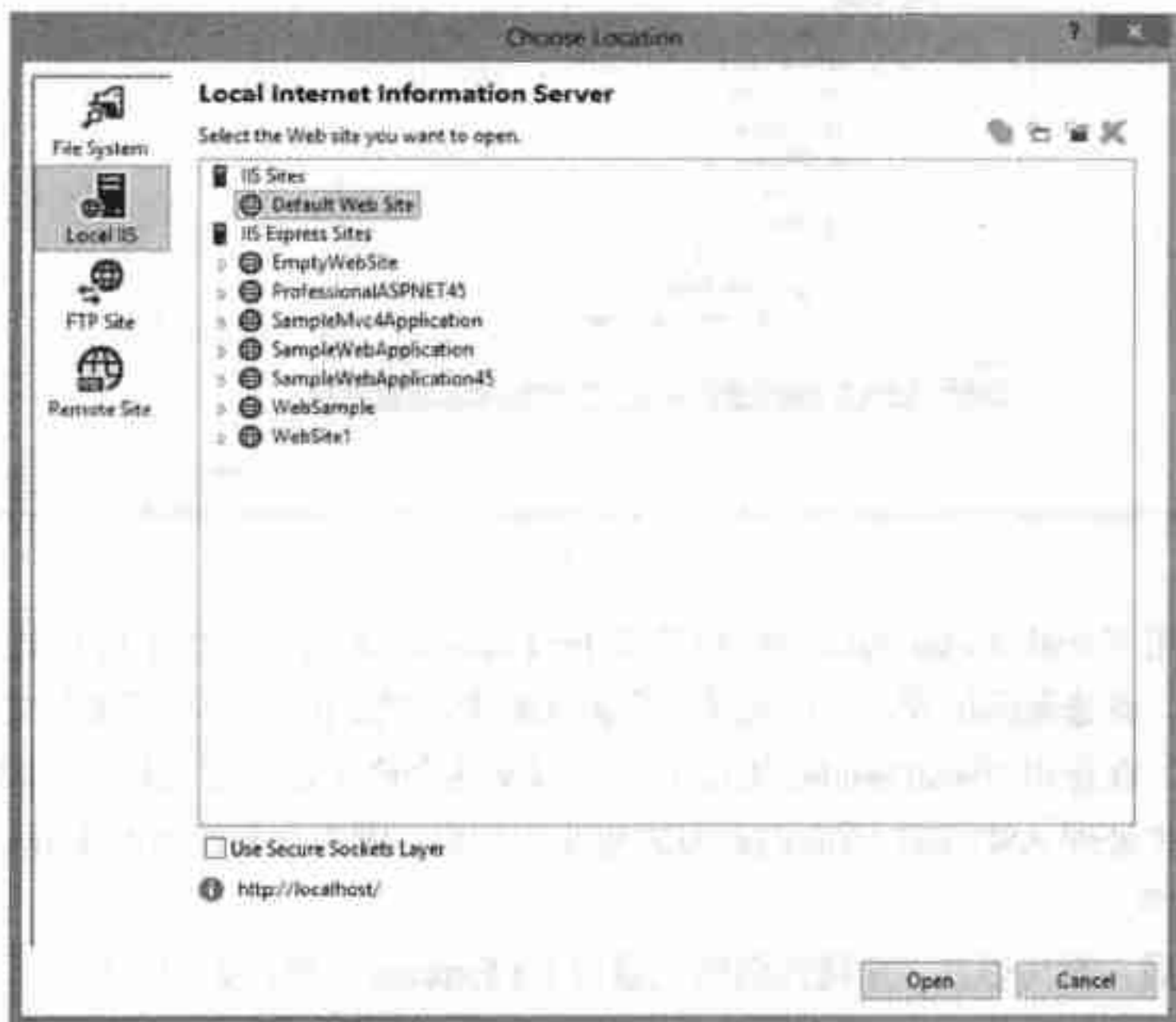


图 3-4



在如图 3-4 所示的 IIS 选项中，所有站点都是为 IIS Express 建立的。在这个对话框中，还可以删除不再有效的站点。也可以创建新站点、Web 应用程序或每个站点下的虚拟目录。

在创建需要的虚拟目录之后，单击 Open 按钮。接着，Visual Studio 2012 会进入创建应用程序的标准过程。但是，应用程序现在不使用 IIS Express，而是使用 IIS 的完整版本。在调用应用程序时，URL 会类似于 `http://localhost/MyWeb/Default.aspx`，这意味着它在使用 IIS。

3.1.3 使用 FTP

在使用 Choose Location 对话框创建 Web 应用程序时，不仅可以决定 Web 应用程序使用什么类型的 Web 服务器，还可以决定应用程序位于什么地方。使用前面的选项会在本地服务器上建立应用程序，而 FTP 选项允许在企业其他地方或者地球另一边的服务器上存储甚至编写应用程序，还可以

使用 FTP 功能在同一台服务器的不同地方工作。使用这个功能可以提供范围广泛的选项，如图 3-5 所示。

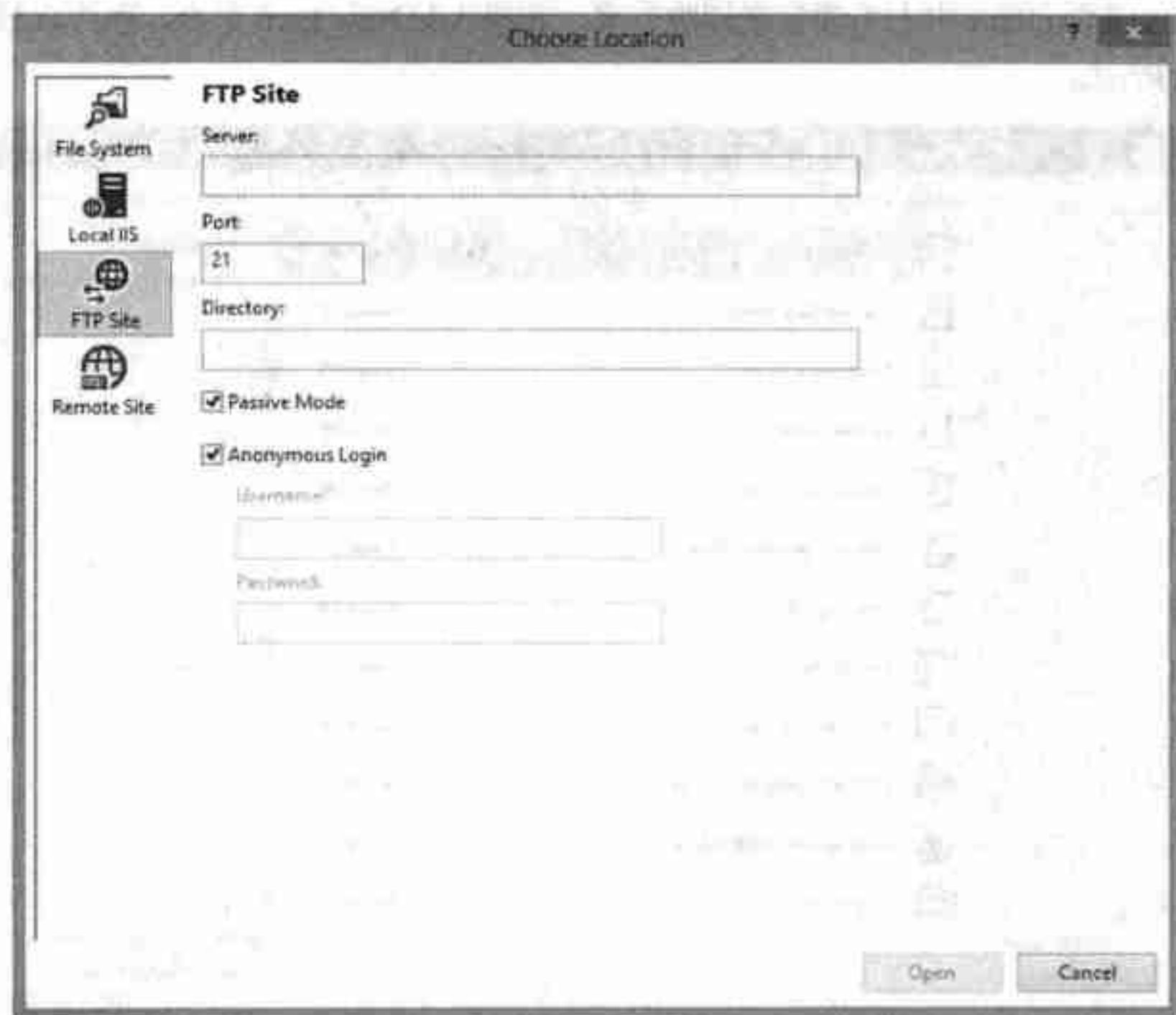


图 3-5

要使用 FTP 在远程服务器上创建应用程序，只需提供服务器名、要使用的端口和目录，以及所需的证书。如果提供的信息正确，Visual Studio 2012 就会连接远程服务器，为应用程序的启动创建合适的文件，就好像是在本地完成这些工作一样。从现在开始就可以打开项目，使用 FTP 连接远程服务器。

3.2 ASP.NET 页面的结构选项

ASP.NET 为构造 ASP.NET 页面的代码提供了两条途径。第一条途径是利用内联编码模型。这个模型对传统 ASP 开发人员来说很熟悉，因为所有的代码都包含在单个.aspx 页面中。第二条途径是使用 ASP.NET 的隐藏代码模型，它允许把页面的业务逻辑代码与表示逻辑代码分开。在这个模型中，页面的表示逻辑存储在.aspx 页面中，业务逻辑存储在单独的类文件.aspx.vb 或.aspx.cs 中。最好使用隐藏代码模型，因为它提供了一个整洁的模型，可以把纯 UI 元素与处理这些元素的代码分开，这也是维护代码的较好方式。

3.2.1 内联编码

在 .NET Framework 1.0/1.1 中，开发人员可以在 Visual Studio .NET 的外部建立内联的 ASP.NET 页面，避免使用 Microsoft 和其他公司强烈推荐的隐藏代码模型。Visual Studio 2012(和 Visual Studio

Express 2012)允许使用这种编码样式方便地构建页面。要构建内联的 ASP.NET 页面而不使用隐藏代码模型,只需从 Add New Item 对话框中选择页面类型,并取消选中 Place code in separate file 复选框即可。在 Solution Explorer 中右击项目或解决方案,选择 Add New Item 命令,就可以打开这个对话框,如图 3-6 所示。



图 3-6

在这个对话框中,可以看到想要构建内联的 ASP.NET 页面时需要取消选中的复选框。实际上,许多页面类型都有用于内联编码样式和隐藏代码样式的选项。表 3-1 显示了从这个对话框中选择文件时的内联编码选项。

表 3-1

使用内联编码时的文件选项	所创建的文件
Web Form	.aspx 文件
Master Page	.master 文件
Web User Control	.ascx 文件
Web Service	.asmx 文件



除了表 3-1 中的文件类型之外,名称后跟(Razor)的文件类型也支持内联编码选项。这些文件在第 35 章介绍。

使用 Web Form 选项和一些控件,页面可以封装表示逻辑和业务逻辑,如程序清单 3-1 所示(本章下载代码中的 Listing03-01.aspx 文件)。

程序清单 3-1 使用内联编码模型的简单页面

```

<%@ Page Language="C#" %>
<script runat="server">
    protected void Button1_Click1(object sender, EventArgs e)
    {
        Literall1.Text = "Hello " + TextBox1.Text;
    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Simple Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            What is your name?<br />
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
            <asp:Button ID="Button1" runat="server" Text="Submit"
                OnClick="Button1_Click" />
        </div>
        <div>
            <asp:Literal ID="Literall1" runat="server" />
        </div>
    </form>
</body>
</html>

```

在这个例子中，所有的业务逻辑都封装在<script>标记之间。内联模型的优点是，业务逻辑和表示逻辑都包含在同一文件中。一些开发人员发现，把所有内容都放在单个可访问的实例中，可以使 ASP.NET 页面的处理更简单。另一个优点是，Visual Studio 2012 在处理内联编码模型和 ASP.NET 4.5 时提供了 IntelliSense。在 Visual Studio 2005 之前，这个功能是不存在的。

3.2.2 隐藏代码模型

构造 ASP.NET 4.5 页面的另一个选项是使用隐藏代码模型构建文件。使用隐藏代码模型思路就是把业务逻辑和表示逻辑分别放在不同的文件中。这样做可以简化页面的处理，尤其是在团队环境下工作时，其中可视化界面的设计人员处理页面的 UI，编码人员处理位于表示部分后台的业务逻辑。



较佳的方法是隐藏代码模型，而不是内联模型。本书的许多例子都使用内联编码模型，因为该模型适合于在单个程序清单中显示例子。尽管例子使用的是内联编码模型，但是推荐移植代码以使用隐藏代码模型。

要在使用隐藏代码模型的 ASP.NET 解决方案中创建新页面，可以从 New File 对话框中选择需要的页面类型。要建立使用隐藏代码模型的页面，首先在 Add New Item 对话框中选择该页面，并确保选中 Place code in separate file 复选框。表 3-2 列出了使用隐藏代码模型的页面选项。

表 3-2

使用隐藏代码模型的文件选项	所创建的文件
Web Form	.aspx 文件、.aspx.vb 或.aspx.cs 文件
Master Page	.master 文件、.master.vb 或.master.cs 文件
Web User Control	.ascx 文件、.ascx.vb 或.ascx.cs 文件
Web Service	.asmx 文件、.vb 或.cs 文件

程序清单 3-1 说明了如何使用内联编码模型创建页面。程序清单 3-2 和 3-3 说明了如何把内联模型转换为隐藏代码模型。程序清单 3-2 的代码在本章下载代码的 Listing03-02.aspx 文件中，程序清单 3-3 的代码在本章下载代码的 Listing03-02.aspx.cs 文件中。

程序清单 3-2 使用 ASP.NET 4.5 隐藏代码模型的.aspx 页面

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Listing03-02.aspx.cs"
    Inherits="ProfessionalASPNet45_03CS.Listing03_02" %>
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Simple Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            What is your name?<br />
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
            <asp:Button ID="Button1" runat="server" Text="Submit"
                OnClick="Button1_Click" />
        </div>
        <div>
            <asp:Literal ID="Literal1" runat="server" />
        </div>
    </form>
</body>
</html>
```

程序清单 3-3 隐藏代码页面

```
public partial class Listing03_02 : System.Web.UI.Page
{
    protected void Button1_Click1(object sender, EventArgs e)
    {
        Literal1.Text = "Hello " + TextBox1.Text;
    }
}
```

使用该 ASP.NET 隐藏代码模型的.aspx 页面在 Page 指令中有一些特性值得注意。第一个特性是 CodeFile，这是 Page 指令中的特性，用于指向与这个表示页面一起使用的隐藏代码页面。在这个例子中，给该特性赋予的值是 Listing03-02.aspx.cs。第二个特性是 Inherits，这个特性在 ASP.NET 的以

前版本中已经存在,但在 ASP.NET 2.0 之前很少使用,用于指定编译页面时绑定到页面的类名。Page 指令在 ASP.NET 4.5 中非常简单。请参见程序清单 3-3 的隐藏代码页面。

隐藏代码页面看起来很简单,因为使用了 .NET 4.5 提供的部分类功能。在隐藏代码文件中创建的类通过 C# 2012 中的 `partial` 关键字使用部分类。这样就可以把需要的方法放在页面类中。在这个例子中,只有一个按钮单击事件。

本章的后面将介绍这两个模型的编译过程。

3.3 ASP.NET 4.5 的 Page 指令

ASP.NET 指令在每个 ASP.NET 页面中都存在,使用这些指令可以控制 ASP.NET 页面的行为。下面是 Page 指令的一个例子:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Listing03-02.aspx.cs"
    Inherits="ProfessionalASPN45_03cs.Listing03_02" %>
```

ASP.NET 页面或用户控件中有 12 个指令,如表 3-3 所示。无论页面使用隐藏代码模型还是内联编码模型,都可以在应用程序中使用这些指令。

基本上,这些指令都是编译器在编译页面时使用的命令。把指令合并到页面中是很简单的操作。指令的格式如下:

```
<%@ [Directive] [Attribute=Value] %>
```

从上面的代码行可以看出,指令以 `<%@` 开头,以 `%>` 结束。最好把这些指令放在页面或控件的顶部,因为开发人员都习惯把指令放在此处(但如果把指令放在其他地方,页面也仍然能编译)。当然,也可以把多个特性添加到指令语句中,如下所示:

```
<%@ [Directive] [Attribute=Value] [Attribute=Value] %>
```

表 3-3 描述了 ASP.NET 4.5 中可供使用的指令。

表 3-3

指 令	说 明
Assembly	把程序集链接到与之相关的页面或用户控件
Control	用户控件(.ascx)使用的 Page 指令
Implements	实现指定的 .NET Framework 接口
Import	在页面或用户控件中导入指定的名称空间
Master	允许指定母版页,即在分析或编译页面时使用的特定特性和值。这个指令只能与母版页(.master)一起使用
MasterType	把类名与页面关联起来,获得包含在特定母版页中的强类型化的引用或成员
OutputCache	控制页面或用户控件的输出高速缓存策略
Page	允许指定在分析或编译页面时使用的页面特定特性和值。这个指令只能用于 ASP.NET 页面(.aspx)
PreviousPageType	允许 ASP.NET 页面处理应用程序中另一页面的回送信息

(续表)

指 令	说 明
Reference	把页面或用户控件链接到当前页面或用户控件
Register	把别名与名称空间和类名关联起来，作为自定义服务器控件语法中的记号
Webhandler	允许把页面用作 <code>HttpHandler</code> ，详见第 30 章

下面简要介绍这些指令。其中一些指令仅在特定的页面类型中有效。

3.3.1 Page 指令

`Page` 指令允许为 ASP.NET 页面(.aspx)指定分析和编译页面时使用的特性和值，这是表 3-3 中最常用的一个指令。ASP.NET 页面是 ASP.NET 的一个重要部分，它包含多个特性。表 3-4 总结了 `Page` 指令的可用特性。

表 3-4

特 性	说 明
<code>AspCompat</code>	如果值为 <code>True</code> ，就允许页面在单线程的单元中执行。这个特性的默认设置是 <code>False</code>
<code>Async</code>	指定 ASP.NET 页面是同步处理还是异步处理
<code>AsyncTimeout</code>	指定等待异步任务完成的时间(秒)，默认设置为 45 秒
<code>AutoEventWireUp</code>	设置为 <code>True</code> 时，指定页面事件是否自动触发。这个特性的默认设置是 <code>True</code>
<code>Buffer</code>	设置为 <code>True</code> 时，支持 HTTP 响应缓冲。这个特性的默认设置是 <code>True</code>
<code>ClassName</code>	指定编译页面时绑定到页面的类名
<code>ClientIDMode</code>	指定为页面上的服务器控件生成 <code>ClientID</code> 值时该页面使用的模式。这个特性的默认值是 <code>AutoID</code> (该模式也用于 ASP.NET 4 之前的 ASP.NET 页面)
<code>ClientTarget</code>	给目标用户代理指定控件应显示的内容。这个特性需要与 <code>web.config</code> 文件中 <code><clientTarget></code> 部分定义的别名关联起来
<code>CCodeBehind</code>	引用与页面相关的已编译的隐藏代码文件。这个特性用于 Web 应用程序项目
<code>CodeFile</code>	引用与页面相关的隐藏代码文件。这个特性用于网站项目
<code>CodeFileBaseClass</code>	指定与隐藏代码类一起使用的基类名，由 <code>CodeFile</code> 特性使用
<code>CodePage</code>	指定响应的代码页面值
<code>CompilationMode</code>	指定 ASP.NET 是否应编译页面，可用的选项包括 <code>Always</code> (默认)、 <code>Auto</code> 或 <code>Never</code> 。设置为 <code>Auto</code> 则表示，在可能的情况下 ASP.NET 将不编译页面
<code>CompilerOptions</code>	编译器字符串，指示页面的编译选项
<code>ContentType</code>	把响应的 HTTP 内容类型定义为标准的 MIME 类型
<code>Culture</code>	指定页面的区域性设置。ASP.NET 3.5 及以后版本允许把 <code>Culture</code> 特性的值设置为 <code>Auto</code> ，从而支持自动检测所需要的区域性
<code>Debug</code>	设置为 <code>True</code> 时，使用调试符号编译页面
<code>Description</code>	提供页面的文本描述。ASP.NET 分析器忽略这个特性和它的值
<code>EnableEventValidation</code>	指定是否在回送和回调时启动事件的有效性验证。该特性的默认设置为 <code>True</code> ，表示验证事件

(续表)

属 性	说 明
EnableSessionState	设置为 True 时, 支持页面的会话状态。默认设置是 True
EnableTheming	设置为 True 时, 页面可以使用主题。默认设置是 True
EnableViewState	设置为 True 时, 在页面中维护视图状态。默认设置是 True
EnableViewStateMac	设置为 True 时, 当用户回送页面时, 页面会对视图状态进行机器验证检查。默认设置是 False
ErrorPage	为所有未处理的页面异常指定用于发送信息的 URL
Explicit	设置为 True 时, 支持 Visual Basic 的 Explicit 选项。默认设置是 False
Language	定义所有内联显示和脚本块使用的语言
LCID	为 Web 窗体的页面定义本地标识符
LinePragmas	Boolean 值, 指定得到的程序集是否使用行附注
MasterPageFile	其值为 String, 指向页面所使用的母版页的地址。这个特性在内容页面中使用
MaintainScrollPositionOnPostback	其值为 Boolean, 表示在回送页面时, 页面是位于相同的滚动位置, 还是在最顶端的位置重新生成页面
MetaDescription	允许在 meta 标记中指定页面描述以达到 SEO 目的
MetaKeywords	允许在 meta 标记中指定页面关键字以达到 SEO 目的
ResponseEncoding	指定页面内容的响应编码
SmartNavigation	指定是否为功能更丰富的浏览器激活 ASP.NET 智能导航功能。这种功能可以把回送信息返回到页面的当前位置。该特性的默认值是 False。从 ASP.NET 2.0 开始不再使用 SmartNavigation, 而是使用 SetFocus() 方法和 MaintainScrollPositionOnPostback 属性
Src	指向类的源文件, 用于所显示页面的隐藏代码
Strict	设置为 True 时, 使用 Visual Basic 的 Strict 模式编译页面。默认值是 False
StylesheetTheme	使用 ASP.NET 的主题功能, 把指定的主题应用于页面。StylesheetTheme 特性和 Theme 特性的区别是, 使用 StylesheetTheme 特性不会重写控件中预先存在的样式设置, 而使用 Theme 特性则会删除这些设置
Theme	使用 ASP.NET 的主题功能, 把指定的主题应用于页面
Title	应用页面的标题。这个特性主要用于内容页面, 该页面必须应用该特性指定的页面标题, 而不是应用母版页中指定的页面标题
Trace	设置为 True 时, 激活页面跟踪功能。默认值是 False
TraceMode	指定激活跟踪功能时如何显示跟踪消息。这个特性的设置可以是 SortByTime 或 SortByCategory。默认设置是 SortByTime
Transaction	指定页面上是否支持事务。这个特性的设置可以是 Disabled、NotSupported、Supported、Required 和 RequiresNew。默认设置是 Disabled
UICulture	UICulture 特性的值指定 ASP.NET 页面使用什么 UI Culture。ASP.NET 3.5 及以后版本允许 UICulture 特性使用 Auto 值, 从而支持自动检测 UICulture
ValidateRequest	设置为 True 时, 可根据一组潜在危险的值检查窗体输入值, 这有助于防止 Web 应用程序受到恶意攻击, 如 JavaScript 攻击。默认值是 True
ViewStateEncryptionMode	指定如何加密页面上的 ViewState, 它的选项包括 Auto、Always 和 Never。默认值是 Auto

(续表)

属 性	说 明
ViewStateMode	指定是否给页面上的控件维护 ViewState
WarningLevel	指定停止编译页面时的编译器警告级别，其值可以是 0~4

下面是使用 Page 指令的一个示例：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Listing03-02.aspx.cs"
    Inherits="ProfessionalASPNet45_03CS.Listing03_02" %>
```

3.3.2 Master 指令

Master 指令非常类似于 Page 指令，但 Master 指令用于母版页(.master)。在使用 Master 指令时，需要指定与站点上的所有内容页面一起使用的模板页面的属性。任何内容页面(使用 Page 指令建立)都可以继承母版页上的所有主要内容(在母版页上使用 Master 指令定义的内容)。尽管这两个指令是类似的，但 Master 指令的特性少于 Page 指令的特性。Master 指令的可用特性如表 3-5 所示。

表 3-5

特 性	说 明
AutoEventWireUp	设置为 True 时，指定母版页的事件是否自动触发。默认设置为 True
ClassName	指定编译页面时绑定到母版页的类名
CodeBehind	引用与母版页相关的已编译隐藏代码文件
CodeFile	引用与母版页相关的隐藏代码文件
CompilationMode	指定 ASP.NET 是否应编译页面。可用的选项包括 Always(默认)、Auto 或 Never。设置为 Auto 时，表示在可能的情况下 ASP.NET 将不编译页面
CompilerOptions	编译器字符串，表示母版页的编译选项
Debug	设置为 True 时，使用调试符号编译母版页
Description	提供母版页的文本描述。ASP.NET 分析器会忽略这个特性和分配给它的值
EnableTheming	设置为 True 时，表示母版页可以使用主题功能。默认设置是 True
EnableViewState	设置为 True 时，维护母版页的视图状态。默认设置是 True
Explicit	设置为 True 时，表示 Visual Basic 的 Explicit 选项被激活。默认设置是 False
Inherits	指定母版页要继承的 CodeBehind 类
Language	定义内联显示和脚本块使用的语言
LinePragmas	Boolean 值，指定得到的程序集是否使用行附注
MasterPageFile	为 String 值，指向某个母版页所使用的另一个母版页的地址。一个母版页可以使用另一个母版页，从而创建嵌套的母版页
Src	指向类的源文件，用于要显示的母版页的隐藏代码
Strict	设置为 True 时，使用 Visual Basic 的 Strict 模式编译母版页。默认设置是 False
WarningLevel	指定停止编译页面时的编译器警告级别，其值可以是 0~4

下面是使用 Master 指令的一个例子：

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeBehind="ProfessionalASPNet45-Layout.master.cs"
Inherits="ProfessionalASPNet45_03CS.ProfessionalASPNet45_Layout" %>
```

3.3.3 Control 指令

Control 指令类似于 Page 指令，但是在构建 ASP.NET 用户控件时使用 Control 指令。Control 指令允许定义用户控件要继承的属性。这些属性值会在分析和编译页面时赋给用户控件。Control 指令的可用特性少于 Page 指令的特性，但其中有许多特性都可以在构建用户控件时根据需要进行修改。表 3-6 详细介绍了这些可用特性。

表 3-6

特 性	说 明
AutoEventWireUp	设置为 True 时，指定用户控件的事件是否自动触发。默认设置为 True
ClassName	指定编译页面时绑定到用户控件的类名
ClientIDMode	指定为页面上的服务器控件生成 ClientID 值时该页面所使用的模式。这个特性的默认值是 AutoID(该模式也用于 ASP.NET 4 之前的 ASP.NET 页面)
CodeBehind	引用与母版页相关的、已编译的隐藏代码文件。在 ASP.NET 2.0 和以前版本中，CodeFile 应与 Inherits 一起使用
CodeFile	引用与用户控件相关的隐藏代码文件
CodeFileBaseClass	指定与隐藏代码类一起使用的基类名，由 CodeFile 特性使用
CompilationMode	指定是否编译控件
CompilerOptions	编译器字符串，表示用户控件的编译选项
Debug	设置为 True 时，使用调试符号编译用户控件
Description	提供用户控件的文本描述。ASP.NET 分析器会忽略这个特性和分配给它的值
EnableTheming	设置为 True 时，表示用户控件可以使用主题功能。默认设置是 True
EnableViewState	设置为 True 时，维护用户控件的视图状态。默认设置是 True
Explicit	设置为 True 时，表示激活 Visual Basic 的 Explicit 选项。默认设置是 False
Inherits	指定用户控件要继承的 CodeBehind 类
Language	定义所有内联显示和脚本块使用的语言
LinePragmas	Boolean 值，指定得到的程序集是否使用行附注
Src	指向类的源文件，用于要显示的用户控件的隐藏代码
Strict	设置为 True 时，使用 Visual Basic 的 Strict 模式编译用户控件。默认设置是 False
WarningLevel	指定停止编译页面时的编译器警告级别，其值可以是 0-4

Control 指令用于 ASP.NET 用户控件。下面是使用该指令的一个例子：

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="RegistrationUserControl.ascx.cs"
Inherits="ProfessionalASPNet45_03CS.RegistrationUserControl" %>
```

3.3.4 Import 指令

Import 指令允许指定要导入 ASP.NET 页面或用户控件的名称空间。通过导入名称空间，该名称空间中的所有类和接口就可以在页面和用户控件中使用。这个指令只支持特性 Namespace。

Namespace 特性为 String 值, 可以指定要导入的名称空间。Import 指令不能包含多个特性/值对。因此, 必须把多个名称空间导入指令放在多行代码, 如下所示:

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
```

应用程序已经引用了多个程序集。如果查看 C:\Windows\Microsoft.NET\Framework\v4.0.xxxxx\CONFIG 下的 web.config 根文件, 就可以找到这些已导入的名称空间的列表。这个程序集列表从 <compilation> 元素的 <assemblies> 子元素引用而来。web.config 根文件中的设置如下所示:

```
<assemblies>
  <add assembly="Microsoft.VisualStudio.Web.PageInspector.Loader,
    Version=1.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
  <add assembly="mscorlib" />
  <add assembly="Microsoft.CSharp, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b03f5f7f11d50a3a" />
  <add assembly="System, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.Configuration, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b03f5f7f11d50a3a" />
  <add assembly="System.Web, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b03f5f7f11d50a3a" />
  <add assembly="System.Data, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.Web.Services, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b03f5f7f11d50a3a" />
  <add assembly="System.Xml, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.Drawing, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b03f5f7f11d50a3a" />
  <add assembly="System.EnterpriseServices, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b03f5f7f11d50a3a" />
  <add assembly="System.IdentityModel, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.Runtime.Serialization, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.ServiceModel, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.ServiceModel.Activation, Version=4.0.0.0,
    Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
  <add assembly="System.ServiceModel.Web, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=31bf3856ad364e35" />
  <add assembly="System.Activities, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=31bf3856ad364e35" />
  <add assembly="System.ServiceModel.Activities, Version=4.0.0.0,
    Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
  <add assembly="System.WorkflowServices, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=31bf3856ad364e35" />
  <add assembly="System.Core, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
```



```

<add assembly="System.Web.Extensions, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=31bf3856ad364e35" />
<add assembly="System.Data.DataSetExtensions, Version=4.0.0.0,
    Culture=neutral, PublicKeyToken=b77a5c561934e089" />
<add assembly="System.Xml.Linq, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
<add assembly="System.ComponentModel.DataAnnotations, Version=4.0.0.0,
    Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
<add assembly="System.Web.DynamicData, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=31bf3856ad364e35" />
<add assembly="System.Web.ApplicationServices, Version=4.0.0.0,
    Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
<add assembly="*" />
<add assembly="System.Web.WebPages.Deployment, Version=1.0.0.0,
    Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
<add assembly="System.Web.WebPages.Deployment, Version=2.0.0.0,
    Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
</assemblies>

```

因为 web.config 根文件中有这个引用, 所以这些程序集不必像在 ASP.NET 1.0/1.1 中那样在 References 文件夹中引用。事实上, 可以添加或删除在这个列表中引用的程序集。例如, 如果服务器上的每个应用程序都引用了一个自定义程序集, 就可以在其他程序集的下面添加对该自定义程序集的类似引用。注意, 还可以通过应用程序专有的 web.config 文件完成这项任务。



尽管可以编辑 C:\Windows\Microsoft.NET\Framework\v4.0.xxxx\CONFIG 下的 web.config 根文件, 但最好不要修改这些文件, 因为将来的升级可能删除自定义程序集的引用。另外, 如果把应用程序部署到 web farm 中, 就必须对每台服务器进行所有针对机器级别的修改。

尽管已引用了程序集, 但是仍然必须在页面中导入这些程序集的名称空间。web.config 根文件包含自动导入应用程序每个页面的名称空间列表, 通过 <pages> 元素的 <namespaces> 子元素指定该列表:

```

<namespaces>
  <add namespace="System" />
  <add namespace="System.Collections" />
  <add namespace="System.Collections.Generic" />
  <add namespace="System.Collections.Specialized" />
  <add namespace="System.ComponentModel.DataAnnotations" />
  <add namespace="System.Configuration" />
  <add namespace="System.Linq" />
  <add namespace="System.Text" />
  <add namespace="System.Text.RegularExpressions" />
  <add namespace="System.Web" />
  <add namespace="System.Web.Caching" />
  <add namespace="System.Web.DynamicData" />

```

```

<add namespace="System.Web.SessionState" />
<add namespace="System.Web.Security" />
<add namespace="System.Web.Profile" />
<add namespace="System.Web.UI" />
<add namespace="System.Web.UI.WebControls" />
<add namespace="System.Web.UI.WebControls.WebParts" />
<add namespace="System.Web.UI.HtmlControls" />
<add namespace="System.Xml.Linq" />
</namespaces>

```

从这个 XML 列表中可以看出, 每个 ASP.NET 页面中都导入了许多名称空间。可以在 web.config 根文件中自由修改这个列表, 甚至可以在应用程序的 web.config 文件中包含类似的名称空间列表。

例如, 可以在应用程序的 web.config 文件中导入自己的名称空间, 使该名称空间可用于所有的页面。

```

<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.5" />
    <httpRuntime targetFramework="4.5" />
    <pages>
      <namespaces>
        <add namespace="Widgets" />
      </namespaces>
    </pages>
  </system.web>
</configuration>

```

把名称空间导入 ASP.NET 页面或用户控件之后, 在使用类时就不必完全限定类名。例如, 在 ASP.NET 页面中导入 System.IO 名称空间, 就可以使用单个类名来引用这个名称空间中的类(即使用 Directory, 而不使用 System.IO.Directory)。

3.3.5 Implements 指令

Implements 指令允许 ASP.NET 页面实现特定的 .NET Framework 接口。这个指令只支持 Interface 特性。

Interface 特性直接指定 .NET Framework 接口。当 ASP.NET 页面或用户控件实现接口时, 可以直接访问其中所有的事件、方法和属性。

下面是 Implements 指令的一个例子:

```
<%@ Implements Interface="System.Web.UI.IValidator" %>
```

3.3.6 Register 指令

Register 指令把别名与名称空间和类名关联起来, 作为自定义服务器控件语法中的记号。把用户控件拖放到.aspx 页面上时, 就使用了 Register 指令。把用户控件拖放到.aspx 页面上后, Visual Studio 2012 就会在页面的顶部创建一条 Register 指令。这样就在页面上注册了用户控件, 从而可以通过特定的名称在.aspx 页面上访问该控件。

Register 指令支持 5 个特性，如表 3-7 所示。

表 3-7

特 性	说 明
Assembly	与 TagPrefix 关联的程序集
Namespace	与 TagPrefix 关联的名称空间
Src	用户控件的位置
TagName	与类名关联的别名
TagPrefix	与名称空间关联的别名

下面是使用 Register 指令把用户控件导入 ASP.NET 页面的一个例子：

```
<%@ Register TagPrefix="MyTag" Namespace="MyNamespace"
    Assembly="MyAssembly" %>
```

3.3.7 Assembly 指令

Assembly 指令在编译时把程序集(.NET 应用程序的构建块)附加到 ASP.NET 页面或用户控件上，从而使该程序集中的所有类和接口都可用于页面。这个指令支持两个特性，如表 3-8 所示。

表 3-8

特 性	说 明
Name	允许指定用于附加到页面文件的程序集名称。程序集名称应该只包含文件名，不包含文件的扩展名。例如，如果文件是 MyAssembly.cs，那么 Name 特性的值就是 MyAssembly
Src	允许指定编译时使用的程序集文件的来源

下面是使用 Assembly 指令的一些例子：

```
<%@ Assembly Name="MyAssembly" %>
<%@ Assembly Src="MyAssembly.vb" %>
```

3.3.8 PreviousPageType 指令

这个指令用于指定跨页面的传送过程起始于哪个页面。在 ASP.NET 页面之间的跨页面传送过程详见后面的 3.6 节。

PreviousPageType 指令用于处理 ASP.NET 4.5 提供的跨页面传送功能。这个简单的指令只支持两个特性，如表 3-9 所示。

表 3-9

特 性	说 明
TypeName	设置回送时派生类的名称
VirtualPath	设置回送时传送页面的位置

3.3.9 MasterType 指令

MasterType 指令把一个类名关联到 ASP.NET 页面,以获得指定母版页中包含的强类型化的引用或成员。这个指令支持两个特性,如表 3-10 所示。

表 3-10

特 性	说 明
TypeName	设置从中获得强类型化的引用或成员的派生类名称
VirtualPath	设置从中检索这些强类型化的引用或成员的页面位置

使用 MasterType 指令的细节请参阅第 16 章。下面是使用该指令的一个例子:

```
<%@ MasterType VirtualPath="~/Wrox.master" %>
```

3.3.10 OutputCache 指令

OutputCache 指令用于控制 ASP.NET 页面或用户控件的输出高速缓存策略。这个指令支持的特性如表 3-11 所示。

表 3-11

特 性	说 明
CacheProfile	允许使用集中式方法管理应用程序的高速缓存配置文件。使用 CacheProfile 特性可指定在 web.config 文件中详细说明的高速缓存配置文件的名称
Duration	高速缓存 ASP.NET 页面或用户控件的持续时间,单位是秒
Location	位置枚举值,默认值为 Any。它只对.aspx 页面有效,不能用于用户控件(.ascx)。其他值有 Client、Downstream、None、Server 和 ServerAndClient
NoStore	指定是否随页面发送没有存储的报头
Shared	指定用户控件的输出是否可以在多个页面之间共享,这个特性使用 Boolean 值,默认设置为 false
SqlDependency	支持某个特定页面使用 SQL Server 高速缓存禁用功能
VaryByContentEncodings	使用分号分隔的字符串列表,用于根据内容编码改变输出高速缓存
VaryByControl	使用分号分隔的字符串列表,用于改变用户控件的输出高速缓存
VaryByCustom	值为字符串,指定自定义的输出高速缓存需求
VaryByHeader	使用分号分隔的 HTTP 报头列表,用于改变输出高速缓存
VaryByParam	使用分号分隔的字符串列表,用于改变输出高速缓存

下面是使用 OutputCache 指令的一个例子:

```
<%@ OutputCache Duration="180" VaryByParam="None" %>
```

Duration 特性用于指定页面在系统高速缓存中存储的时间(单位为秒)。

3.3.11 Reference 指令

Reference 指令声明, 另一个 ASP.NET 页面或用户控件应该与当前活动的页面或控件一起编译。这个指令只支持一个特性: VirtualPath 设置从中引用活动页面的页面或用户控件的位置。

下面是使用 Reference 指令的一个例子:

```
<%@ Reference VirtualPath="~/MyControl.ascx" %>
```

3.4 ASP.NET 的页面事件

ASP.NET 开发人员一直在服务器端代码中使用各种事件。他们使用的许多事件都专门用于特定的服务器控件。例如, 如果要在终端用户单击 Web 页面上的某个按钮时执行某项操作, 就要在服务器端代码中创建按钮单击事件, 如程序清单 3-4 所示(本章下载代码中的 Listing03-04.aspx.cs)。

程序清单 3-4 C#中的按钮单击事件示例

```
protected void VerifyDataButton_Click(object sender, EventArgs e)
{
    // Verify Data
}
```

除了服务器控件之外, 开发人员还希望在创建或销毁 ASP.NET 页面时初始化操作。ASP.NET 页面有许多事件可以完成这些任务。自从 ASP.NET 诞生以来, 就包含如下页面事件:

- AbortTransaction
- CommitTransaction
- DataBinding
- Disposed
- Error
- Init
- Load
- PreRender
- Unload

这个列表中较常用的一个页面事件是 Load 事件, 它在 C#中的用法如程序清单 3-5 所示(本章下载代码中的 Listing03-05.aspx.cs)。

程序清单 3-5 使用 Page_Load 事件

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write("This is the Page_Load event");
}
```

除了上面的页面事件之外, ASP.NET 4.5 中还添加了下述事件:

- InitComplete: 表示完成了页面初始化。
- LoadComplete: 表示页面已完全加载到内存中。

- PreInit: 表示页面初始化前的那一刻。
- PreLoad: 表示页面加载到内存前的那一刻。
- PreRenderComplete: 表示页面显示在浏览器中之前的那一刻。

PreInit 事件的用法如程序清单 3-6 所示(本章下载代码中的 Listing03-06.aspx.cs)。

程序清单 3-6 使用页面事件

```
protected void Page_PreInit(object sender, EventArgs e)
{
    Page.Theme = Request.QueryString["ThemeChange"];
}
```

如果创建 ASP.NET 4.5 页面并启用跟踪功能, 就会看到主页面事件的初始化顺序。这些事件的引发顺序如图 3-7 所示。

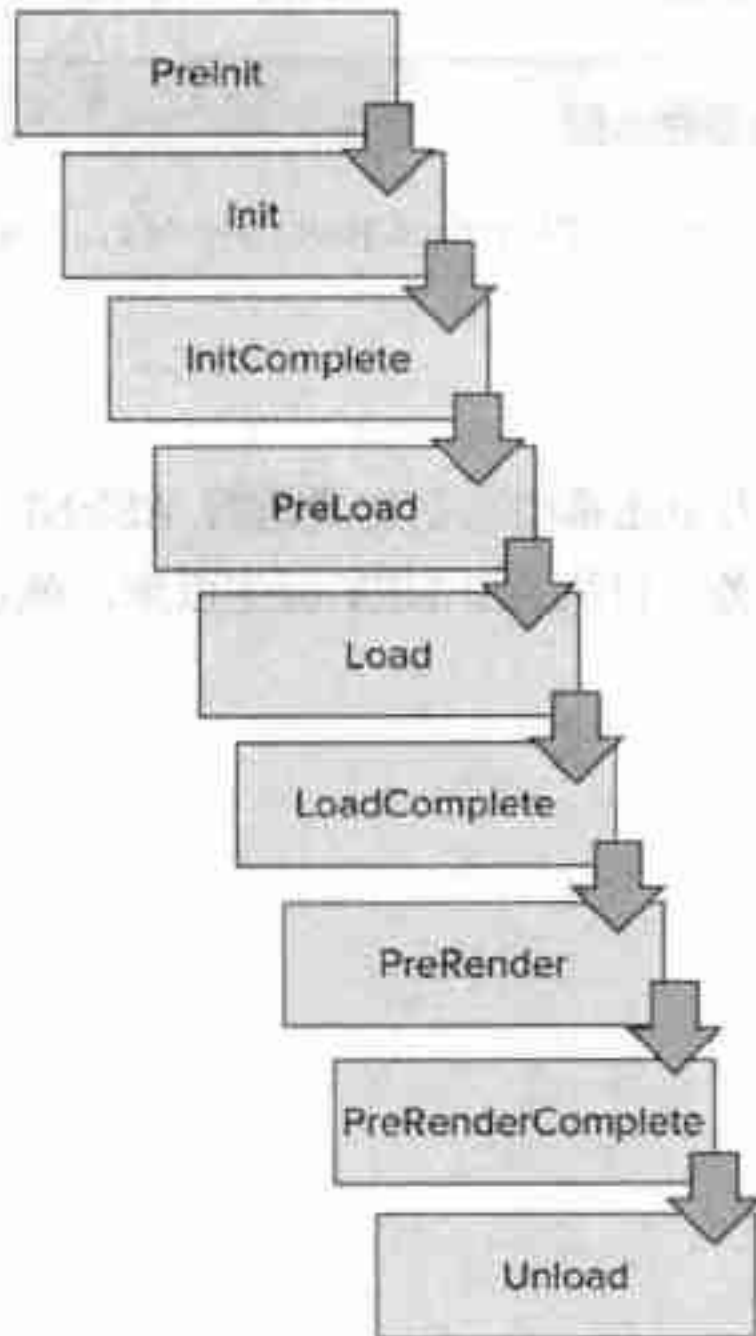


图 3-7

添加完这些新的页面事件, 我们就可以在页面编译期间在许多不同的位置处理页面和页面上的控件。本书的代码示例将多次用到这些有用的新页面事件。

3.5 处理回送

在使用 ASP.NET 页面时, 一定要理解刚才介绍的页面事件。它们非常重要, 因为在页面生存期的某些特定位置, 需要在这些事件中放置许多页面代码。

在 ASP 3.0 中, 开发人员把页面传送给应用程序中的其他页面。ASP.NET 页面一般要回送给自

已以处理事件(如按钮单击事件)。

因此,必须区分终端用户在第·一次加载页面时传送的内容和回送的内容。回送就是传送回原来的页面。回送包含在初始页面上收集的所有窗体信息,以备在需要时处理。



回送使用 HTTP POST 操作,而不是 HTTP GET。HTTP POST 在 HTTP 请求的消息主体中发送窗体数据,而 HTTP GET 在 HTTP 请求的 URL 中发送窗体数据。

所有的回送都在 ASP.NET 页面上进行,所以要弄清楚是请求页面的第一个实例,还是请求回送原来的页面。使用 Page 类的 IsPostBack 属性可以进行此项检查,如下所示:

```
if(!Page.IsPostBack) {
    // Do processing
}
```

3.6 跨页面传送

ASP 3.0 中有个常见功能在 ASP.NET 3.0/3.1 中很难实现,即进行跨页面传送。跨页面传送就是提交窗体(如 Page1.aspx),并把这个窗体和所有的控件值都传送给另一个页面(Page2.aspx)。

传统上,在 ASP.NET 1.0/1.1 中创建的页面都只传送给自已,并且在这个页面实例中处理控件值。要区分页面的第一次请求和回送,可以使用 Page.IsPostBack 属性,如本章前面所示。

尽管有了这个功能,但是开发人员仍然希望能把数据传送给另一个页面,并在该页面上处理第一个页面的控件值。ASP.NET 4.5 已实现了这个功能,过程非常简单。

例如,创建页面 Page1.aspx,其中包含·一个简单的窗体。这个页面如程序清单 3-7 所示。这个程序清单中的代码位于本章下载代码的 Listing03-07 (Page 1).aspx 文件中。

程序清单 3-7 Page1.aspx

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Button1_Click(Object sender, EventArgs e)
    {
        Label1.Text = "Hello " + TextBox1.Text + "<br />" +
            "Date Selected: " + Calendar1.SelectedDate.ToShortDateString();
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>First Page</title>
</head>
<body>
    <form id="form1" runat="server">
        Enter your name:<br />
        <asp:Textbox ID="TextBox1" Runat="server"></asp:Textbox>
        <p>
            When do you want to fly?<br />
```

```

        <asp:Calendar ID="Calendar1" Runat="server"></asp:Calendar>
    </p>
    <br />
    <asp:Button ID="Button1" Runat="server" Text="Submit page to itself"
        OnClick="Button1_Click" />
    <asp:Button ID="Button2" Runat="server" Text="Submit page to Page2.aspx"
       PostBackUrl="Page2.aspx" />
    <p>
        <asp:Label ID="Label1" Runat="server"></asp:Label>
    </p>
</form>
</body>
</html>

```

如程序清单 3-7 所示, Page1.aspx 的代码非常有趣。首先,在页面上显示两个按钮。这两个按钮都提交窗体,但是各自把窗体提交到不同的位置。第 1 个按钮把窗体提交给自己,这是 ASP.NET 1.0/1.1 的默认操作。实际上,Button1 并没有什么不同,它把 Page1.aspx 提交为回送,因为在按钮控件上使用了 OnClick 属性。Page1.aspx 上的 Button1_Click 方法处理页面上服务器控件包含的值。

第 2 个按钮 Button2 完全不同。与第 1 个按钮不同的是,这个按钮不包含 OnClick 事件,它使用的是 PostBackUrl 属性。这个属性带有一个字符串值,指向页面所要传送到的文件的位置,在本例中是 Page2.aspx。这说明现在 Page2.aspx 接收回送的内容和包含在 Page1.aspx 控件中的所有值。Page2.aspx 的代码如程序清单 3-10 所示(本章下载代码中的 Listing03-08(Page 2).aspx)。

程序清单 3-8 Page2.aspx

```

<%@ Page Language="C#" %>
<script runat="server">
    protected void Page_Load(Object sender, EventArgs e)
    {
        TextBox pp_TextBox1;
        Calendar pp_Calendar1;
        pp_TextBox1 = (TextBox)PreviousPage.FindControl("TextBox1");
        pp_Calendar1 = (Calendar)PreviousPage.FindControl("Calendar1");
        Label1.Text = "Hello " + pp_TextBox1.Text + "<br />" +
            "Date Selected: " + pp_Calendar1.SelectedDate.ToShortDateString();
    }
</script>
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Second Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Label ID="Label1" Runat="server"></asp:Label>
    </form>
</body>
</html>

```

要从第二个页面获得 Page1.aspx 的控件值,可以采用两种方式。第一种方式如程序清单 3-8 所示。要获得从前一个页面传送过来的控件值,只需创建该控件类型的一个实例,并使用 PreviousPage 属性的 FindControl()方法填充该实例即可。赋予 FindControl()方法的 String 值是 Id,它用于前一个页面上的服务器控件。赋值之后,就可以处理该服务器控件及其值,就好像它一开始就位于当前页面上一样。从例子中可以看出,可以从控件中提取 Text 和 SelectedDate 属性。

从第一个页面(Page1.aspx)获取控件值的另一种方式是创建该控件的属性,如程序清单 3-9 所示(本章下载代码中的 Listing03-09(Page 1).aspx)。

程序清单 3-9 从属性获得控件的值

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    public TextBox pp_TextBox1 { get { return TextBox1; } }
    public Calendar pp_Calendar1 { get { return Calendar1; } }
</script>
```

由于这些属性都显示在传送的页面中,因此第二个页面(Page2.aspx)可以轻松地处理从第一个页面获得的服务器控件的属性。程序清单 3-10 说明了 Page2.aspx 如何处理这些属性(本章下载代码中的 Listing03-10(Page 2).aspx)。

程序清单 3-10 使用从第一个页面获得的属性

```
<%@ Page Language="C#" %>
<%@ PreviousPageType VirtualPath="Page1.aspx" %>
<script runat="server">
    protected void Page_Load(Object sender, EventArgs e)
    {
        Label1.Text = "Hello " + PreviousPage.pp_TextBox1.Text + "<br />" +
            "Date Selected: " +
            PreviousPage.pp_Calendar1.SelectedDate.ToShortDateString();
    }
</script>
```

为了使用 Page1.aspx 中的属性,必须把 PreviousPage 属性强类型化为 Page1.aspx。为此,要使用 PreviousPageType 指令。这个指令允许使用 VirtualPath 特性指向 Page1.aspx。

注意,在从一个页面传送到另一个页面时,并不仅限于在第二个页面上处理回送的内容。实际上,还可以在 Page1.aspx 上创建方法来处理回送的内容,然后再移到 Page2.aspx。为此,只需在 Page1.aspx 上为按钮添加一个 OnClick 事件和一个方法,再为 PostBackUrl 属性分配一个值。然后,就可以处理 Page1.aspx 上的回送,之后再移到 Page2.aspx。

如果在未处理 Page1.aspx 之前有人请求 Page2.aspx,该怎么办? 实际上,很容易确定这个请求是来自于 Page1.aspx,还是有人直接单击了 Page2.aspx。使用 IsCrossPagePostBack 属性就可以处理该请求,IsCrossPagePostBack 属性与 IsPostBack 属性很类似,使用它可以确定请求是否来自于 Page1.aspx。程序清单 3-11 显示了使用该属性的一个例子(本章下载代码中的 Listing03-11.aspx)。

程序清单 3-11 使用 IsCrossPagePostBack 属性

```
<%@ Page Language="C#" %>
<%@ PreviousPageType VirtualPath="Pagel.aspx" %>
<script runat="server">
    protected void Page_Load(Object sender, EventArgs e)
    {
        if(PreviousPage != null && PreviousPage.IsCrossPagePostBack)
        {
            Label1.Text = "Hello " + PreviousPage.pp_TextBox1.Text + "<br />" +
                "Date Selected: " +
                PreviousPage.pp_Calendar1.SelectedDate.ToShortDateString();
        }
        else
        {
            Response.Redirect("Pagel.aspx");
        }
    }
</script>
```

3.7 ASP.NET 应用程序文件夹

在创建 ASP.NET 应用程序时, 注意 ASP.NET 4.5 使用的是基于文件的方法。在使用 ASP.NET 时, 可以在应用程序中增加任意多个文件和文件夹, 而无须每次在给解决方案添加新文件时重新编译它们。ASP.NET 4.5 能够自动、动态地预编译 ASP.NET 应用程序。

ASP.NET 1.0/1.1 把解决方案中的所有内容都编译到 DLL 中。现在不再需要这样操作, 因为 ASP.NET 应用程序有定义好的文件夹结构。使用 ASP.NET 定义好的文件夹, 就可以自动编译代码, 在整个应用程序中访问应用程序主题, 并在需要时可以使用全局资源。下面介绍这些定义好的文件夹以及它们的工作方式。首先介绍 App_Code 文件夹。

3.7.1 App_Code 文件夹

App_Code 文件夹用于存储类、.wsdl 文件和类型化的数据集。存储在这个文件夹中的所有项都可以自动用于解决方案中的所有页面。App_Code 文件夹的优点是在把某个对象放在这个文件夹之后, Visual Studio 2012 就会自动检测它, 如果该对象是类(.vb 或.cs), 就会编译它, 从.wsdl 文件中自动创建 XML Web 服务代理类, 或从.xsd 文件中自动创建类型化的数据集。文件自动编译后, 这些项能够立即用于同一个解决方案中的任意 ASP.NET 页面。下面看看如何通过 App_Code 文件夹使用解决方案中的一个简单类。

第一步是创建 App_Code 文件夹。为此, 右击解决方案并选择 Add | Add ASP.NET Folder | App_Code 命令。创建好 App_Code 文件夹之后, 右击该文件夹并选择 Add New Item 命令, 就会打开 Add New Item 对话框, 如图 3-8 所示。该对话框给出了可以放在这个文件夹中的一些文件类型的选项。如果使用 Visual Studio 2012, 可用的选项有支持 AJAX 的 WCF 服务、Class 文件、LINQ to SQL 类、ADO.NET 实体数据模型、ADO.NET 实体对象生成器、Sequence Diagram、Text 模板、Text 文件、DataSet、Report 和类图。Visual Web Developer 2012 Express Edition 只提供这些文件的一部分。

下面的第一个例子选择 Class 文件类型,并将类命名为 Calculator.cs。程序清单 3-12 显示了 Calculator 类(本章下载代码中的 Calculator.cs)。

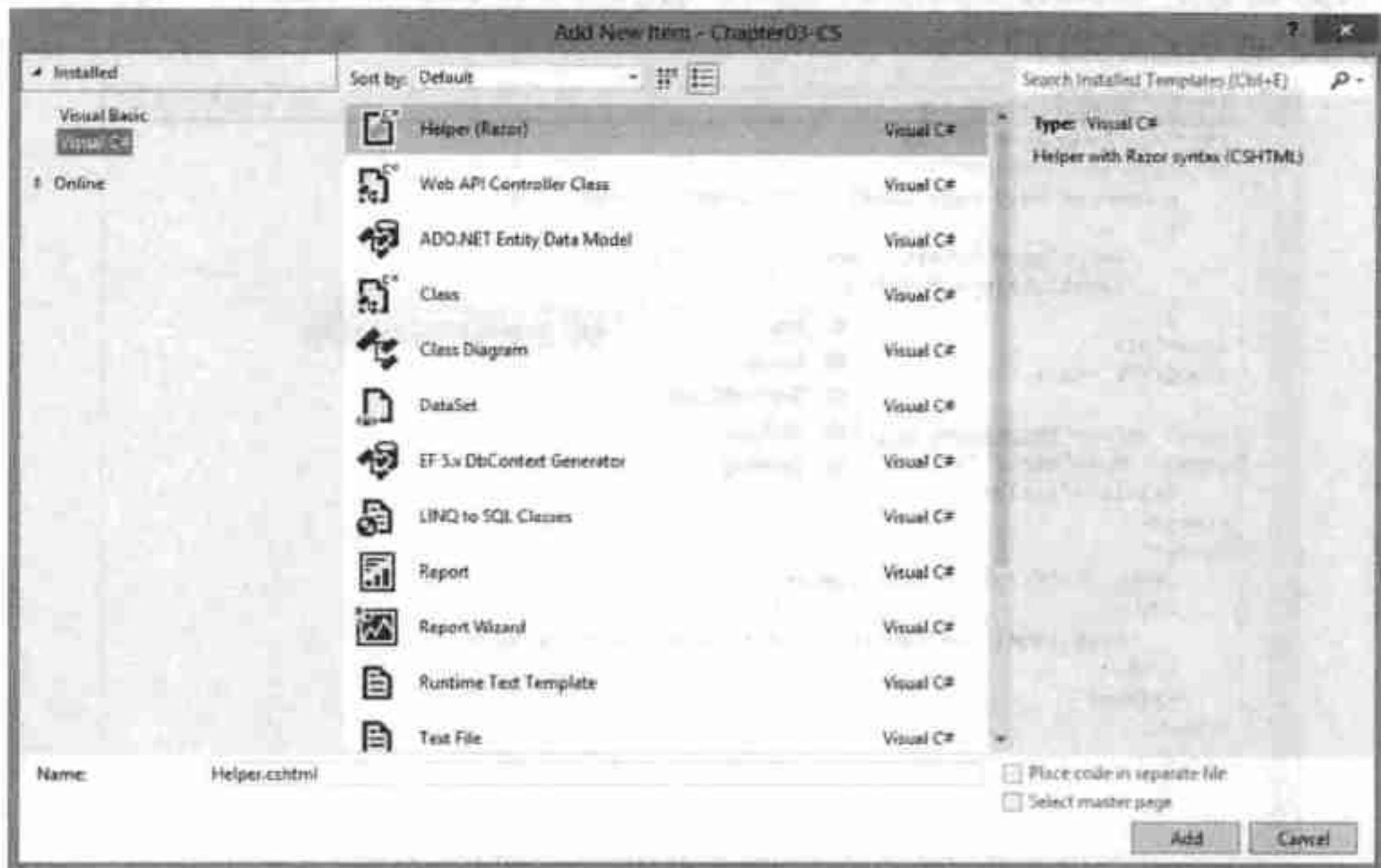


图 3-8

程序清单 3-12 Calculator 类

```
public class Calculator
{
    public int Add(int a, int b)
    {
        return (a + b);
    }
}
```

下一步是保存这个文件,现在它就可以用于解决方案中的任意页面。为了证明这一点,可以创建一个简单的.aspx 页面,其中只有一个 Label 服务器控件。程序清单 3-13(本章下载代码中的 Calculator.aspx)把代码放在 Page_Load 事件中,使这个新类可用于页面。

程序清单 3-13 使用了 Calculator 类的.aspx 页面

```
<%@ Page Language="C#" %>
<script runat="server">
    protected void Page_Load(Object s, EventArgs e)
    {
        Calculator myCalc = new Calculator();
        Label1.Text = myCalc.Add(12, 12).ToString();
    }
</script>
```

在运行这个.aspx 页面时, 注意它使用了 Calculator 类, 并且在使用前不需要编译该类。实际上, 在解决方案中保存 Calculator 类或把这个类移到 App_Code 文件夹之后, 就能够立即在 IntelliSense 中看到该类的方法, 如图 3-9 所示。



图 3-9

为了说明 Visual Studio 2012 是如何使用 App_Code 文件夹的, 请再次在 IDE 中打开 Calculator 类并添加 Subtract 方法。Calculator 类现在应该如程序清单 3-16 所示(本章下载代码中的 Calculator.cs)。

程序清单 3-14 给 Calculator 类添加 Subtract 方法

```
public class Calculator
{
    public int Add(int a, int b)
    {
        return (a + b);
    }
    public int Subtract(int a, int b)
    {
        return (a - b);
    }
}
```

给 Calculator 类添加了 Subtract 方法之后, 保存文件, 然后返回到.aspx 页面。注意, 此时 IDE 已经重新编译了 Calculator 类, 新的方法现在可用于页面。可以在 IntelliSense 中直接看到这个方法, 如图 3-10 所示。



图 3-10

App_Code 文件夹中的所有内容都被编译到单个程序集中。放在 App_Code 文件夹中的类文件不需要使用特定的语言。也就是说,即使解决方案中的所有页面都是用 Visual Basic 编写的,解决方案的 App_Code 文件夹中的 Calculator 类也可以用 C# 创建(Calculator.cs)。

由于这个文件夹包含的所有类都构建在单个程序集中,因此在 App_Code 根文件夹中不能有不同语言的类,如下面的代码所示:

```
\App_Code
  Calculator.cs
  AdvancedMath.vb
```

如果 App_Code 文件夹中的两个类是用不同的语言创建的(如上所示),就会产生错误。不能让指定的编译器处理两种不同的语言。因此,为了能够在 App_Code 文件夹中使用多种语言,必须对文件夹结构和 web.config 文件进行一些修改。

第一步是给 App_Code 文件夹添加两个子文件夹: VB 文件夹和 CS 文件夹,从而文件夹结构如下所示:

```
\App_Code
  \VB
    Add.vb
  \CS
    Subtract.cs
```

但是,这样仍然不能把这些类文件正确编译到不同的程序集中,至少在给 web.config 文件添加一些内容之前不能这样操作。此时解决方案中还没有 web.config 文件,所以通过 Solution Explorer 添加该文件。之后,修改<compilation>节点,使其结构如程序清单 3-15 所示(本章下载代码中的 web.config)。

程序清单 3-15 构建 web.config 文件,使 App_Code 文件夹中的类可以使用不同的语言

```
<compilation debug="false" targetFramework="4.5">
  <codeSubDirectories>
```

```
<add directoryName="VB"></add>
<add directoryName="CS"></add>
</codeSubDirectories>
</compilation>
```

在 web.config 文件中进行修改后, 就可以处理 ASP.NET 页面中的各个类。此外, 与 VB 文件夹中的类一样, CS 文件夹中的 C# 类现在会自动编译。把这些目录添加到 web.config 文件中后, 就不需要像以前那样将它们命名为 VB 和 CS, 而可以使用任意名称。

3.7.2 App_Data 文件夹

App_Data 文件夹中存放的是应用程序使用的数据, 是集中存储应用程序所用数据的地方。App_Data 文件夹可以包含 Microsoft SQL Express 文件(.mdf 文件)、Microsoft Access 文件(.mdb 文件)、XML 文件等。

应用程序使用的用户账户具有对 App_Data 文件夹中任意文件的读写权限。在这个文件夹中存储所有数据文件的另一个原因是, 许多 ASP.NET 系统, 从成员关系和角色管理系统到 GUI 工具(如 ASP.NET MMC 管理单元和 ASP.NET Web 站点管理工具), 都构建为使用 App_Data 文件夹。

3.7.3 App_GlobalResources 文件夹

资源文件是一些字符串表, 当应用程序需要根据某些事情(如区域性的改变)修改内容时, 资源文件可用作这些应用程序的数据字典。可以在 App_GlobalResources 文件夹中添加程序集资源文件(.resx), 它们会动态编译并成为解决方案的一部分, 供应用程序中的所有.aspx 页面使用。在使用 ASP.NET 1.0/1.1 时, 必须使用 resgen.exe 工具, 把资源文件编译为.dll 或.exe 文件, 这样才能在解决方案中使用。而在 ASP.NET 4.5 中, 资源文件的处理就容易很多。只要把在应用程序范围内使用的资源放在 App_GlobalResources 文件夹中, 就能立即使用它们。有关本地化的内容, 详见第 32 章。

3.7.4 App_LocalResources 文件夹

如果对使用 App_GlobalResources 文件夹构造应用程序范围内的资源不感兴趣, 而对构造只能用于单个.aspx 页面的资源感兴趣, 就可以使用 App_LocalResources 文件夹。

可以把专门用于页面的资源文件添加到 App_LocalResources 文件夹中, 方法是构造.resx 文件名, 如下所示:

- Default.aspx.resx
- Default.aspx.fi.resx
- Default.aspx.ja.resx
- Default.aspx.en-gb.resx

现在, 可以从 App_LocalResources 文件夹的相应文件检索在 Default.aspx 页面上使用的资源声明。如果没有找到匹配的资源, 就默认使用 Default.aspx.resx 资源文件。如果客户使用 fi-FI(芬兰)区域性设置, 就使用 Default.aspx.fi.resx 文件。有关本地资源的本地化内容, 详见第 32 章。

3.7.5 App_WebReferences 文件夹

App_WebReferences 文件夹是 ASP.NET 3.5 之前版本中使用的 Web References 文件夹的新名称。

使用 App_WebReferences 文件夹可以自动访问应用程序引用的远程 Web 服务。有关 ASP.NET 中的 Web 服务的内容, 详见第 13 章。

3.7.6 App_Browsers 文件夹

App_Browsers 文件夹包含 .browser 文件, 这些 .browser 文件是 XML 文件, 用于标识向应用程序发出请求的浏览器, 并识别这些浏览器具备的功能。C:\Windows\Microsoft.NET\Framework\v4.0.xxxx\Config\Browsers 上有可全局访问的 .browser 文件列表。另外, 要想修改这些默认浏览器定义文件中的任意部分, 只需把相应的 .browser 文件从 Browsers 文件夹复制到应用程序的 App_Browsers 文件夹中, 并且修改定义即可。

3.8 编译

如前所述, 在开发人员处理应用程序的各个部分时, Visual Studio 2012 就会编译它们(例如, 通过将一个类放在 App_Code 文件夹中)。应用程序的其他部分, 如 .aspx 页面, 可以像在 ASP.NET 的早期版本中那样通过在浏览器中引用它们对其进行编译。

在浏览器中第一次引用 ASP.NET 页面时, 请求会传送给 ASP.NET 分析器。ASP.NET 分析器使用该页面的语言创建类文件, 该类文件会根据其扩展名(.aspx)传送给 ASP.NET 分析器, 因为 ASP.NET 意识到这个文件扩展名类型对其处理是有意义的。在创建类文件后, 类文件就编译为 DLL, 然后写入 Web 服务器的磁盘。此时, 实例化 DLL 并处理, 从而为 ASP.NET 页面的最初请求者生成输出。上述详细过程如图 3-11 所示。

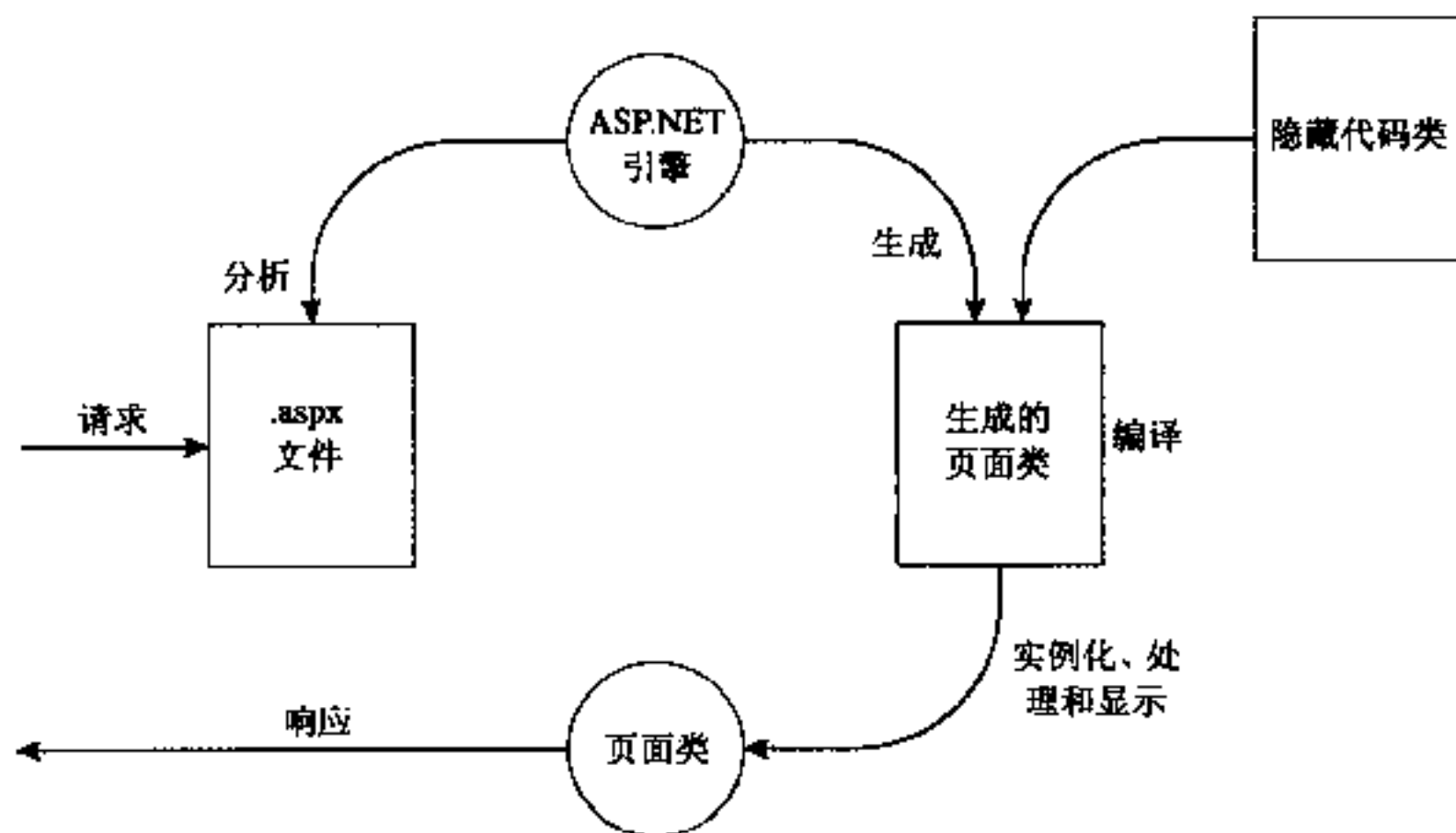


图 3-11

下一个请求到来时, 情况有了一些变化。没有再为第二个请求和以后的各个请求重复上述整个过程, 该请求只是实例化已创建的 DLL, 该 DLL 将响应发送给请求者, 如图 3-12 所示。

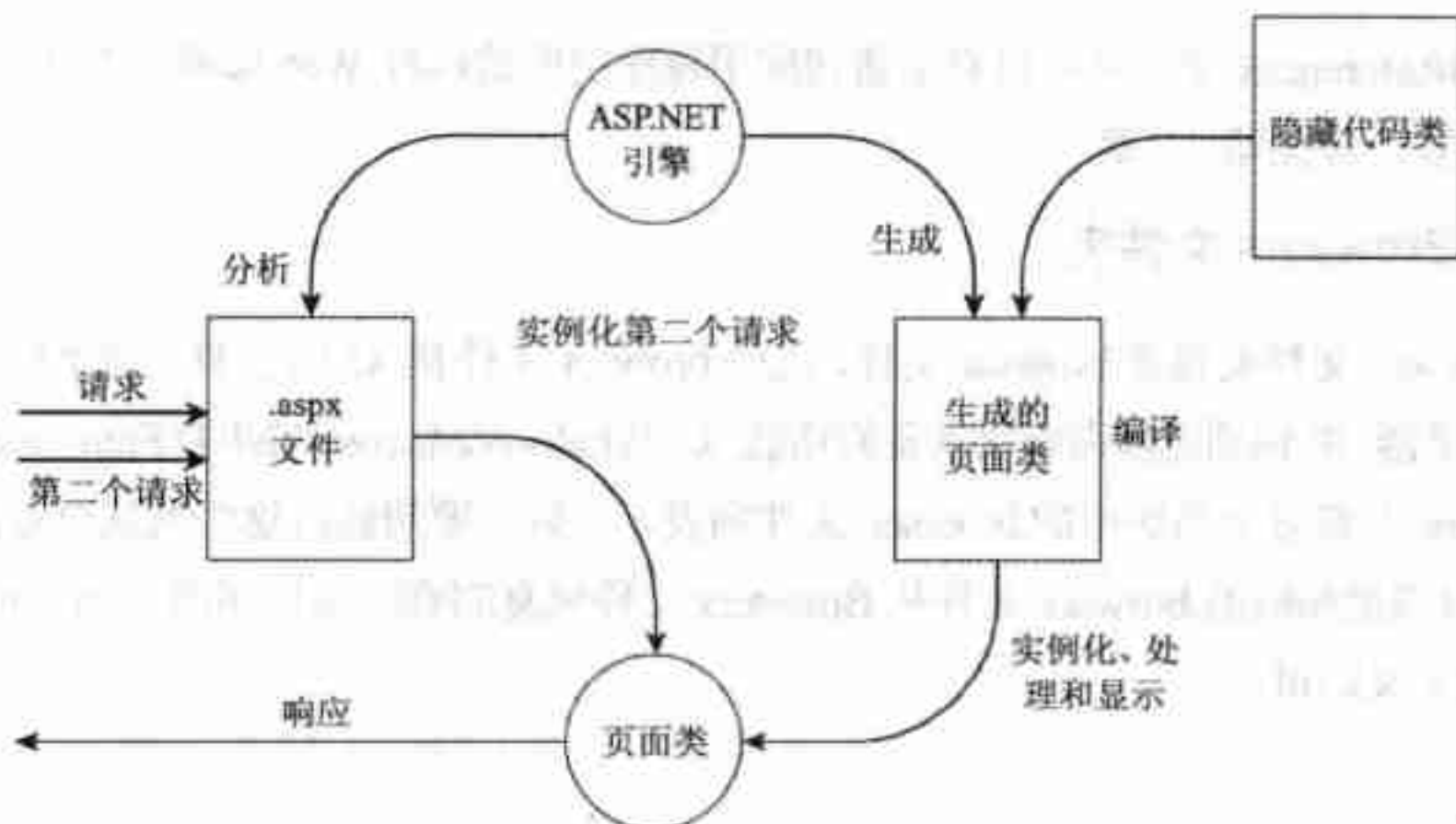


图 3-12

由于采用这个过程机制，如果对.aspx 隐藏代码页面进行了修改，就需要重新编译应用程序。如果站点很大，这就是个很痛苦的过程，在编译后第一次引用.aspx 页面时，我们常常不希望终端用户感觉有很大的延迟。于是许多开发人员开始开发自己的工具，以自动检查应用程序中的每个页面，使终端用户在浏览时没有第一次引用页面的延迟感。

ASP.NET 引入了一些技术，可以使用一个命令对整个应用程序进行预编译，该命令可以直接在命令行上调用。其中一种类型的编译称为原地预编译。为了预编译整个 ASP.NET 应用程序，必须使用 ASP.NET 附带的 `aspnet_compiler.exe` 工具。使用命令窗口可以打开这个工具。打开命令窗口，进入 `C:\Windows\Microsoft.NET\Framework\v4.0.xxxxx\`，此时就可以使用 `aspnet_compiler` 工具。也可以通过 Visual Studio 2012 Developer Command Prompt 直接打开这个工具。在 Windows 8 中，选择 Search 面板，进入 Developer Command Prompt，然后再从搜索结果中选择合适的应用程序。

打开命令提示窗口后，通过 `aspnet_compiler.exe` 工具使用下面的命令执行原地预编译：

```
aspnet_compiler -p "C:\Inetpub\wwwroot\WROX" -v none
```



Windows 8 和 Windows Server 2012 上的 IIS 8 包含了 Application Initialization 功能，这非常类似于使用 `aspnet_compiler.exe` 实用工具预编译应用程序。更多信息可在线搜索 IIS8 Application Initialization。

我们会得到一条消息，指示预编译成功完成。预编译功能的另一项优点是，可以用来查找应用程序中任意 ASP.NET 页面上的错误。由于要调用每个页面，因此如果某个页面包含直到运行时才触发的错误，在使用这个预编译方法时会立即接到错误通知。

下一个预编译选项常常称为用于部署的预编译。这是 ASP.NET 的一项杰出功能，允许把应用程序编译为某些 DLL，然后部署给客户、伙伴或其他位置。这么做不但需要的步骤非常少，而且应用程序在编译后，只需移动这些 DLL 和某些占位符文件，就可以让站点运转起来。也就是说，在部署时 Web 站点的代码都放在 DLL 中。

但是在此之前，应该在根驱动器上创建一个文件夹，例如 Wrox。这个文件夹是指示编译器输出

内容的文件夹。创建该文件夹后，就可以返回到编译器工具，并给出如下命令：

```
aspnet_compiler -v [Application Name] -p [Physical Location] [Target]
```

因此，假设应用程序 ThomsonReuters 位于 C:\Websites\ThomsonReuters 下，就可以使用如下命令：

```
aspnet_compiler -v /ThomsonReuters -p C:\Websites\ThomsonReuters C:\Wrox
```

按 Enter 键，编译器就会告诉我们，要么其中一个命令参数有问题，要么编译成功，如图 3-13 所示。如果编译成功，就会在目标目录下看到输出。



图 3-13

在上面的例子中，-v 是指定应用程序虚拟路径的命令，这里使用 /ThomsonReuters 提供该路径。命令 -p 指向应用程序的物理路径，这里是 C:\Websites\ThomsonReuters。最后是 C:\Wrox，表示编译器输出的位置。表 3-12 描述了 aspnet_compiler.exe 工具的一些命令。

表 3-12

命 令	说 明
-m	指定应用程序的完整 IIS 元数据库路径。如果使用 -m 命令，就不能使用 -v 或 -p 命令
-v	指定要编译的应用程序的虚拟路径。如果还使用了 -p 命令，就使用物理路径查找应用程序的位置
-p	指定要编译的应用程序的物理路径。如果未指定该命令，就使用 IIS 元数据库查找应用程序
-u	如果使用了这个命令，就指定应用程序是可更新的
-f	指定覆盖目标目录(假定该目录存在)
-d	指定调试信息应排除在编译过程之外
-c	指定要编译的应用程序应完全重建
[目标目录]	指定编译后的文件放在哪个目标目录下。如果未指定该命令，就把输出文件放在应用程序目录下

编译应用程序之后，就可以进入 C:\Wrox 目录查看输出，其中包含了原始应用程序中的所有文件和文件结构。但是，如果查看某个文件的内容，就会注意该文件只是个占位符。在该文件中包含如下注释：

```
This is a marker file generated by the precompilation tool
and should not be deleted!
```


实际上, bin 文件夹中有一个或多个 App_Web_XXXXXXXXX.dll 文件, 其中包含了所有的页面代码。因为是在 DLL 文件中, 所以还提供了一个重要的代码平台。之后, 只需使用 FTP 或 Windows 资源管理器把这些文件移到另一台服务器上, 就可以从这些文件中运行整个 Web 应用程序。在更新应用程序后, 只需提供一组新的已编译文件即可。示例输出如图 3-14 所示。

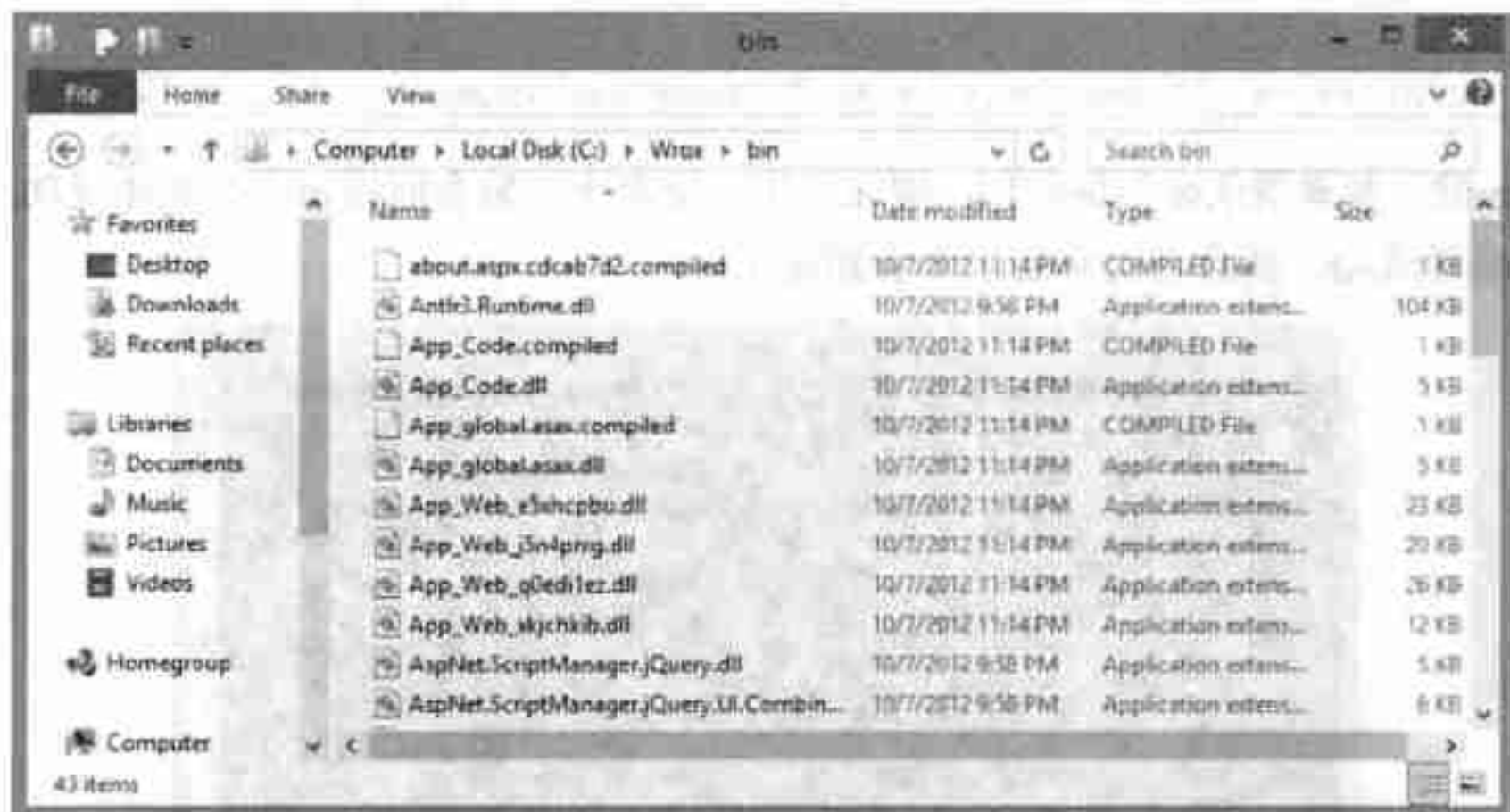


图 3-14

注意, 这个编译过程并不编译每种类型的 Web 文件。实际上, 只编译与 ASP.NET 相关的文件类型, 没有编译如下类型的文件:

- HTML 文件
- XML 文件
- XSD 文件
- web.config 文件
- 文本文件

这个问题暂时无法解决, 但是 HTML 文件和文本文件除外。对于这些文件类型, 只需把文件类型的扩展名改为.aspx, 它们就会像其他 ASP.NET 文件那样编译到 App_Web_XXXXXXXXX.dll 中。

3.9 Build Provider

在学习各个 ASP.NET 文件夹时, 可以注意到 App_Code 文件夹是比较有趣的。可以直接把代码文件、XSD 文件、甚至 WSDL 文件放在该文件夹中, 以自动编译它们。把类文件放在 App_Code 文件夹中时, 该类可以自动地被正在运行的应用程序使用。在 ASP.NET 的早期版本中, 如果希望部署自定义的组件, 就必须预编译这些组件, 再在应用程序中使用它们。现在, ASP.NET 会自动完成预编译的工作, 我们不需要执行任何编译例程。

在 App_Code 文件夹中, 哪些文件类型会自动编译呢? 与 ASP.NET 中的大多数对象一样, 这是通过配置文件中应用的设置来确定的。程序清单 3-16 列出了一段配置代码, 这段代码取自 ASP.NET 4.5 的主 web.config 文件。注意这个程序清单没有包含在下载代码中, 因为这里列出的 web.config 是随 .NET 4.5 Framework 一起安装的。

程序清单 3-16 Build Provider 的配置代码

```

<buildProviders>
  <add extension=".aspx" type="System.Web.Compilation.PageBuildProvider" />
  <add extension=".ascx"
    type="System.Web.Compilation.UserControlBuildProvider" />
  <add extension=".master"
    type="System.Web.Compilation.MasterPageBuildProvider" />
  <add extension=".asmx" type="System.Web.Compilation.WebServiceBuildProvider" />
  <add extension=".ashx" type="System.Web.Compilation.WebHandlerBuildProvider" />
  <add extension=".soap" type="System.Web.Compilation.WebServiceBuildProvider" />
  <add extension=".resx" type="System.Web.Compilation.ResXBuildProvider" />
  <add extension=".resources"
    type="System.Web.Compilation.ResourcesBuildProvider" />
  <add extension=".wsdl" type="System.Web.Compilation.WsdlBuildProvider" />
  <add extension=".xsd" type="System.Web.Compilation.XsdBuildProvider" />
  <add extension=".js" type="System.Web.Compilation.ForceCopyBuildProvider" />
  <add extension=".lic" type="System.Web.Compilation.IgnoreFileBuildProvider" />
  <add extension=".lick" type="System.Web.Compilation.IgnoreFileBuildProvider" />
  <add extension=".exclude"
    type="System.Web.Compilation.IgnoreFileBuildProvider" />
  <add extension=".refresh"
    type="System.Web.Compilation.IgnoreFileBuildProvider" />
  <add extension=".edmx"
    type="System.Data.Entity.Design.AspNet.EntityDesignerBuildProvider" />
  <add extension=".xoml"
    type="System.ServiceModel.Activation.WorkflowServiceBuildProvider,
      System.WorkflowServices, Version=4.0.0.0, Culture=neutral,
      PublicKeyToken=31bf3856ad364e35" />
  <add extension=".svc"
    type="System.ServiceModel.Activation.ServiceBuildProvider,
      System.ServiceModel.Activation, Version=4.0.0.0, Culture=neutral,
      PublicKeyToken=31bf3856ad364e35" />
  <add extension=".xamlx" type="System.Xaml.Hosting.XamlBuildProvider,
      System.Xaml.Hosting, Version=4.0.0.0, Culture=neutral,
      PublicKeyToken=31bf3856ad364e35" />
</buildProviders>

```

这段代码包含一组 Build Provider，它们可以在开发过程中由两个实体使用。在开发过程中，在 Visual Studio 2012 中构建解决方案时，首先会使用 Build Provider。例如，在开发过程中把 .wsdl 文件放在 App_Code 文件夹中，IDE 就会自动访问 .wsdl 文件中动态编译的代理类。使用 Build Provider 的另一个实体是 ASP.NET 本身。前面提到过，把 .wsdl 文件拖放到已部署的应用程序的 App_Code 文件夹中，会使 ASP.NET 应用程序自动访问已创建的代理类。

Build Provider 只是一个继承了 System.Web.Compilation.BuildProvider 的类。web.config 文件中的 <buildProviders> 段用于列出要使用的 Build Provider 类。动态编译 WSDL 文件的功能是在配置文件中通过下面的代码定义的：

```

<add extension=".wsdl" type="System.Web.Compilation.WsdlBuildProvider" />

```

也就是说，使用继承了 BuildProvider 的 WsdlBuildProvider 类编译扩展名为 .wsdl 的任何文件。Microsoft 提供了一组 Build Provider 以供用户使用。从程序清单 3-16 中可以看出，除了 WsdlBuildProvider

之外,还有许多其他的提供程序,包括 XsdBuildProvider、PageBuildProvider、UserControlBuildProvider 和 MasterPageBuildProvider 等。从这些提供程序的名称就可以看出它们的作用。下一节将讨论不太能够从名称猜测出作用的其他 Build Provider。

3.9.1 使用内置的 Build Provider

本节介绍两个 Build Provider: ForceCopyBuildProvider 和 IgnoreFileBuildProvider, 它们都包含在提供程序的默认列表中。

ForceCopyBuildProvider 基本上只复制用于部署的文件, 这些文件使用指定的扩展名, 它们不包含在编译过程中。使用 ForceCopyBuildProvider 的扩展名如程序清单 3-16 中的预定义列表所示, 是 .js 文件类型(JavaScript 文件扩展名)。.js 文件只会被简单地复制, 并不会包含在编译过程中(这对 JavaScript 文件来说是有意义的)。使用下面的命令, 可以把其他需要的文件类型添加到这个复制过程中:

```
<add extension=".chm" type="System.Web.Compilation.ForceCopyBuildProvider" />
```

除了 ForceCopyBuildProvider 之外, 还可以使用 IgnoreFileBuildProvider 类。这个 Build Provider 会在部署或编译过程中忽略指定的文件类型。也就是说, 使用 IgnoreFileBuildProvider 定义的文件类型会被忽略。Visual Studio 不会复制、编译或部署这些文件类型。因此, 如果在项目中包含了 Visio 图, 就可以在 web.config 文件中添加<add>元素, 以忽略相应的文件类型。示例如下:

```
<add extension=".vsd" type="System.Web.Compilation.IgnoreFileBuildProvider" />
```

添加了这个<add>元素后, 所有.vsd 文件就会被忽略。

3.9.2 使用自己的 Build Provider

除了使用预定义的 Build Provider 之外, 还可以进一步构建自定义 Build Provider, 以便在应用程序中使用。

例如, 假设要根据自定义 .car 文件中应用的设置动态地构建 Car 类, 以便在多个项目中使用 .car 定义文件, 或在同一个项目中多次使用该定义文件。通过 Build Provider 可以更方便地定义 Car 类的多个实例。

.car 文件类型的示例如程序清单 3-17 所示(本章下载代码中的 Listing03-17.car)。

程序清单 3-17 .car 文件的示例

```
<?xml version="1.0" encoding="utf-8" ?>
<car name="AspNetCar">
  <color>White</color>
  <door>4</door>
  <seats>8</seats>
</car>
```

该 XML 声明指定了要编译的类名、类中各个属性的值和一个方法。这些元素构成了 Car 类。理解了 .car 文件类型的结构之后, 下一步就是构建 Build Provider。为此, 使用选择的语言在 Visual Studio 中创建新的类库项目, 命名为 CarBuildProvider。CarBuildProvider 包含类 Car.cs, 该类继承了基类 BuildProvider, 重写了 BuildProvider 类的 GenerateCode() 方法, 如程序清单 3-18 所示(本章下载

代码中的 Listing03-18.cs)。

程序清单 3-18 CarBuildProvider

```
using System.IO;
using System.Web.Compilation;
using System.Xml;
using System.CodeDom;
namespace CarBuildProvider
{
    class Car : BuildProvider
    {
        public override void GenerateCode(AssemblyBuilder myAb)
        {
            XmlDocument carXmlDoc = new XmlDocument();
            using (Stream passedFile = OpenStream())
            {
                carXmlDoc.Load(passedFile);
            }
            XmlNode mainNode = carXmlDoc.SelectSingleNode("/car");
            string selectionMainNode = mainNode.Attributes["name"].Value;
            XmlNode colorNode = carXmlDoc.SelectSingleNode("/car/color");
            string selectionColorNode = colorNode.InnerText;
            XmlNode doorNode = carXmlDoc.SelectSingleNode("/car/door");
            string selectionDoorNode = doorNode.InnerText;
            XmlNode seatsNode = carXmlDoc.SelectSingleNode("/car/seats");
            string selectionseatsNode = seatsNode.InnerText;
            CodeCompileUnit ccu = new CodeCompileUnit();
            CodeNamespace cn = new CodeNamespace();
            CodeMemberProperty cmp1 = new CodeMemberProperty();
            CodeMemberProperty cmp2 = new CodeMemberProperty();
            CodeMemberMethod cmml = new CodeMemberMethod();
            cn.Imports.Add(new CodeNamespaceImport("System"));
            cmp1.Name = "Color";
            cmp1.Type = new CodeTypeReference(typeof(string));
            cmp1.Attributes = MemberAttributes.Public;
            cmp1.GetStatements.Add(new CodeSnippetExpression("return \"\" +
                selectionColorNode + \"\""));
            cmp2.Name = "Doors";
            cmp2.Type = new CodeTypeReference(typeof(int));
            cmp2.Attributes = MemberAttributes.Public;
            cmp2.GetStatements.Add(new CodeSnippetExpression("return " +
                selectionDoorNode));
            cmml.Name = "Go";
            cmml.ReturnType = new CodeTypeReference(typeof(int));
            cmml.Attributes = MemberAttributes.Public;
            cmml.Statements.Add(new CodeSnippetExpression("return " +
                selectionseatsNode));
            CodeTypeDeclaration ctd = new CodeTypeDeclaration(selectionMainNode);
            ctd.Members.Add(cmp1);
            ctd.Members.Add(cmp2);
            ctd.Members.Add(cmml);
            cn.Types.Add(ctd);
            ccu.Namespaces.Add(cn);
        }
    }
}
```



```

        myAb.AddCodeCompileUnit(this, ccu);
    }
}

```

`GenerateCode()`方法使用了 `AssemblyBuilder` 类的一个实例。这个 `AssemblyBuilder` 对象来自于 `System.Web.Compilation` 名称空间，因此本例的类库项目需要引用 `System.Web` 程序集。对于在 `Car` 类中使用的各个对象，还必须导入下述名称空间：

```

using System.IO;
using System.Web.Compilation;
using System.Xml;
using System.CodeDom;

```

之后，`GenerateCode()`方法还剩下一项任务没有完成：加载 `.car` 文件。由于 `.car` 文件使用特定格式的 XML，因此可以方便地使用 `XmlDocument` 对象加载该文档。接着，可以使用 `CodeDom` 创建一个类，该类动态地包含两个属性和一个方法。所生成的类是 `.car` 文件定义的内容的抽象表示。最后，该类的名称也是通过 `.car` 文件中主节点 `<Car>` 的 `name` 特性值动态派生的。

然后，用作输入对象的 `AssemblyBuilder` 实例把生成的代码和其他内容编译到一个程序集中。

ASP.NET 项目包含对 `CarBuildProvider` 程序集的引用，其含义是什么？它表示可以创建自己定义的 `.car` 文件，把该文件拖放到 `App_Code` 文件夹中，这样就可以立即编程访问该文件中指定的定义。

为了证明这一点，首先在服务器的 `machine.config` 文件或应用程序的 `web.config` 文件中添加对 `Build Provider` 的引用，如程序清单 3-21 所示(本章下载代码中的 `web.config`)。

程序清单 3-19 在 `web.config` 文件中引用 `Build Provider`

```

<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.5">
      <buildProviders>
        <add extension=".car" type="CarBuildProvider.Car" />
      </buildProviders>
    </compilation>
    <httpRuntime targetFramework="4.5" />
  </system.web>
</configuration>

```

`<buildProviders>`元素是`<compilation>`元素的子元素，并且又带有两个子元素，用于添加或删除提供程序。在本例中，因为要添加对自定义对象 `CarBuildProvider` 的引用，所以使用`<add>`元素。`<add>`元素可以带两个特性——`extension` 和 `type`，必须同时使用这两个特性。在 `extension` 特性中，定义与 `Build Provider` 相关的文件扩展名，这里使用扩展名 `.car`。也就是说，使用该扩展名的所有文件都与 `type` 特性中定义的类相关。`type` 特性指定已建立的 `CarBuildProvider` 类的引用——`CarBuildProvider.Car`。

有了这个引用后，就可以创建程序清单 3-17 中的 `.car` 文件。把创建好的 `.car` 文件放到 `App_Code` 文件夹中，就可以立即访问通过 `.car` 文件中的定义动态生成的类。例如，把 `AspNetCar` 用作 `<Car>` 元素的 `name` 特性值，这就是生成的类名。在 Visual Studio 中输入代码时，这个类名会在 IntelliSense

中显示出来。

只要创建 `AspNetCar` 类的一个实例，就可以访问该类的属性和方法，如图 3-15 所示。

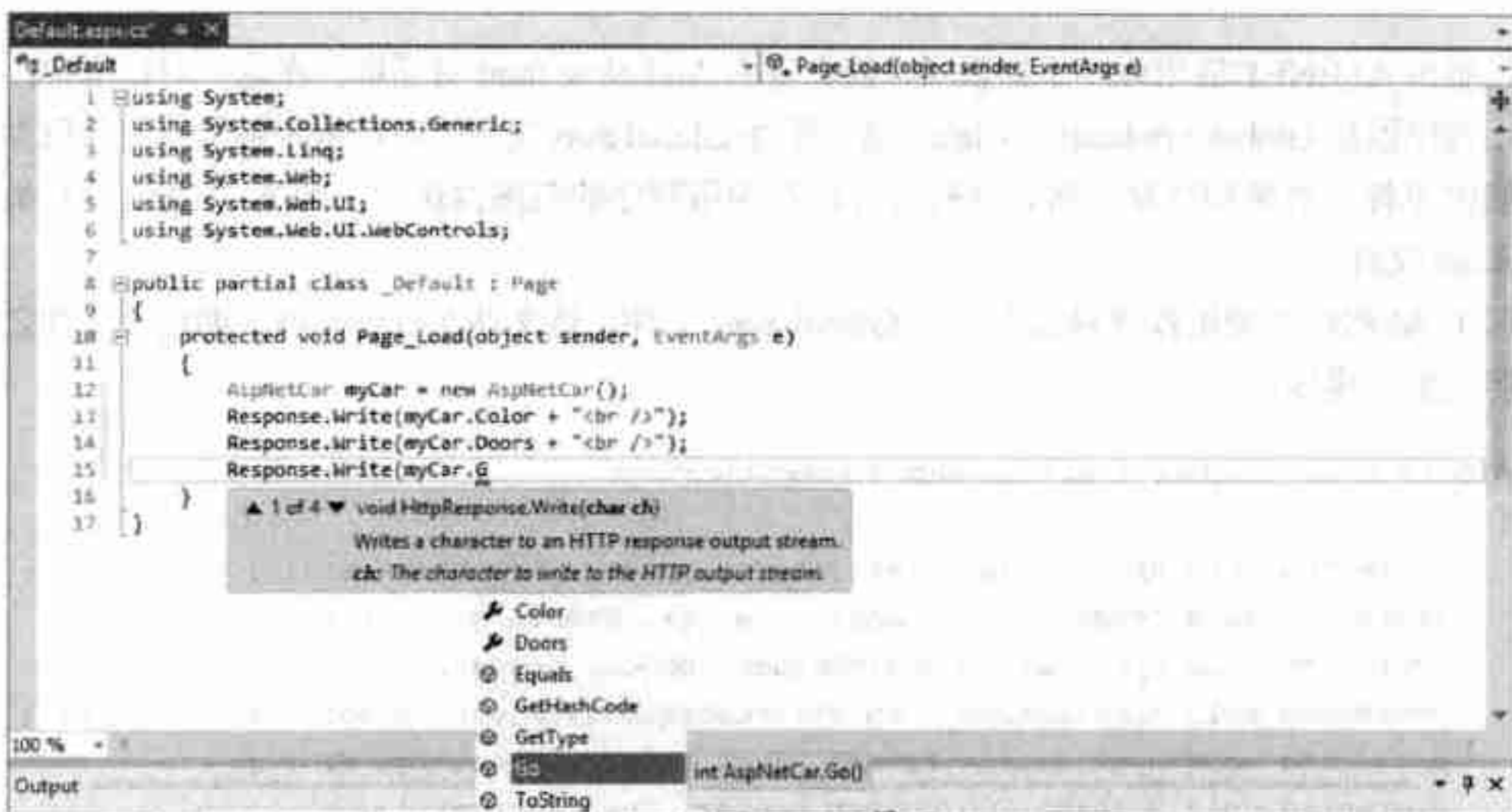


图 3-15

除了访问类的属性和方法之外，还可以访问在 `car` 文件中定义的值，如图 3-16 所示。运行图 3-15 中所示的代码示例，就会在浏览器上得到这个输出。

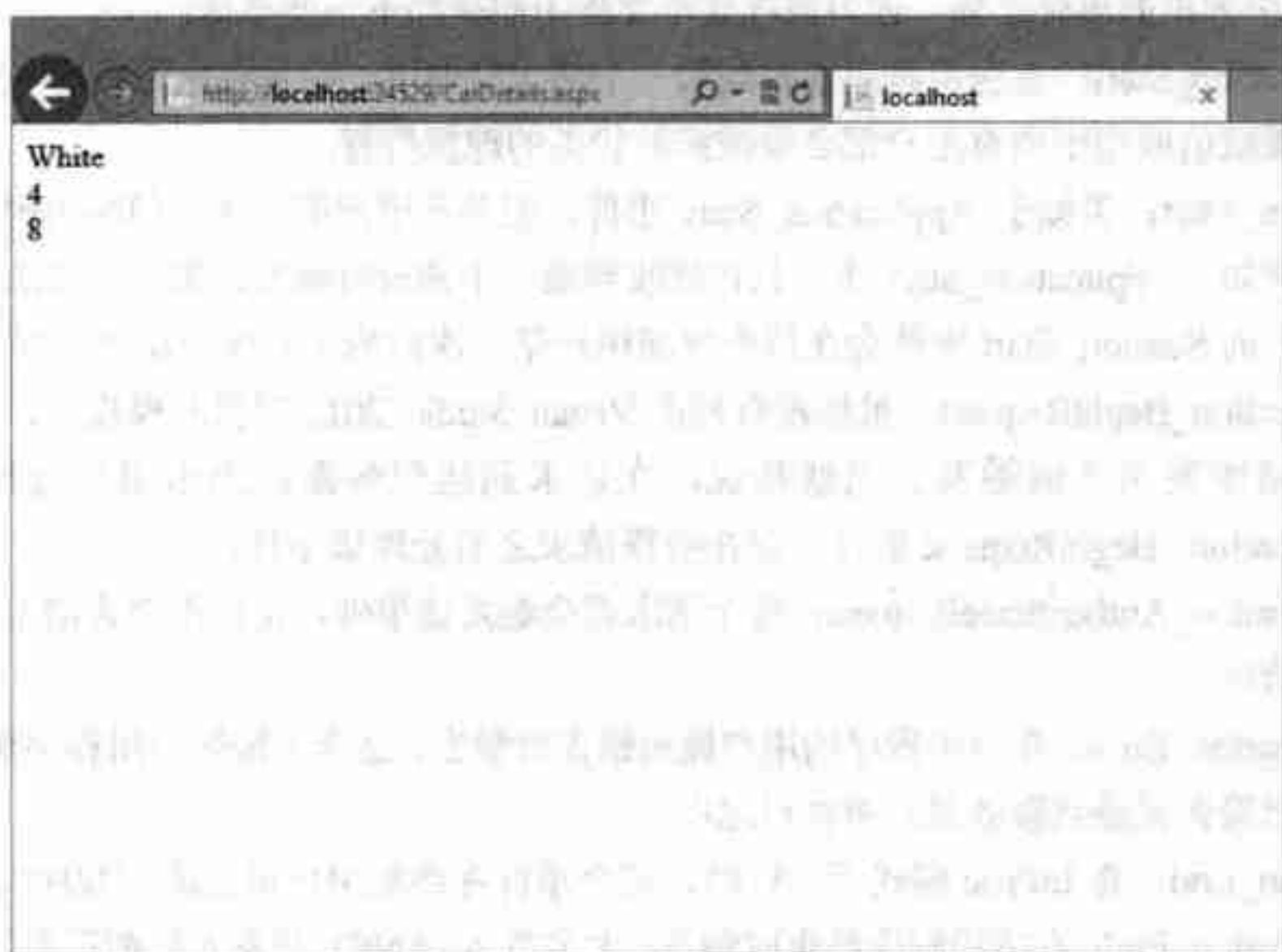


图 3-16

尽管 `car` 类的实际用处不大，但这个例子说明了如何使用 Build Provider 机制，以扩展应用程序的功能。

3.10 Global.asax

只要为 ASP.NET 应用程序添加新项，就会进入 Add New Item 对话框。在这个对话框中，可以给应用程序添加 Global Application Class。这会添加 Global.asax 文件，应用程序使用该文件保存应用程序级的事件、对象和变量，所有这些都可以在应用程序的范围内访问。ASP 开发人员有类似的 Global.asa 文件。

每个 ASP.NET 应用程序只能有一个 Global.asax 文件，该文件支持许多项。创建该文件之后，就会得到如下模板：

```
public class Global : System.Web.HttpApplication
{
    protected void Application_Start(object sender, EventArgs e) { }
    protected void Session_Start(object sender, EventArgs e) { }
    protected void Application_BeginRequest(object sender, EventArgs e) { }
    protected void Application_AuthenticateRequest(object sender, EventArgs e) { }
    protected void Application_Error(object sender, EventArgs e) { }
    protected void Session_End(object sender, EventArgs e) { }
    protected void Application_End(object sender, EventArgs e) { }
}
```

与处理.aspx 页面中的页面级事件一样，也可以在 Global.asax 文件中处理应用程序事件。除了这个代码示例中列出的事件之外，还可以在这个文件中构建如下一些事件：

- **Application_Start**：在应用程序接收到第一个请求时调用，这是在应用程序中给应用程序级的变量赋值或指定所有用户都必须维护的状态的理想位置。
- **Session_Start**：类似于 Application_Start 事件，但是在用户第一次访问应用程序时调用该事件。例如，Application_Start 事件只在接收到第一个请求时触发，第一个请求会让应用程序运行，而 Session_Start 事件会在每个终端用户第一次向应用程序发出请求时调用。
- **Application_BeginRequest**：虽然没有列在 Visual Studio 2012 提供的模板中，但该事件会在每个请求发出之前触发。也就是说，在请求到达服务器但尚未得到处理时，会触发 Application_BeginRequest 事件，并在处理请求之前处理该事件。
- **Application_AuthenticateRequest**：每个请求都会触发该事件，允许用户为请求建立自定义的身份验证。
- **Application_Error**：在应用程序的用户抛出错误时触发。适合于提供应用程序级的错误处理，或者把错误记录到服务器的事件日志中。
- **Session_End**：在 InProc 模式下运行时，这个事件在终端用户退出应用程序时触发。
- **Application_End**：在应用程序结束时触发。大多数 ASP.NET 开发人员都不常使用这个事件，因为 ASP.NET 很好地完成了关闭和清理剩余对象的任务。

除了提供对全局应用程序事件的访问之外，还可以在 Global.asax 文件中使用指令，就像在其他 ASP.NET 页面中一样。Global.asax 文件允许使用如下指令：

- Application
- Assembly

- Import

这些指令的执行方式与在其他 ASP.NET 页面类型中相同。

使用 Global.asax 文件的示例如程序清单 3-20 所示(本章下载代码中的 Global.asax), 该例演示了如何记录 ASP.NET 应用程序域退出的时间。ASP.NET 应用程序域退出时, ASP.NET 应用程序会立即结束。因此, 应该在 Global.asax 文件的 Application_End 方法中编写一些记录代码。

程序清单 3-20 使用 Global.asax 文件的 Application_End 事件

```
<%@ Application Language="C#" %>
<%@ Import Namespace="System.Reflection" %>
<%@ Import Namespace="System.Diagnostics" %>
<script runat="server">
    void Application_End(object sender, EventArgs e)
    {
        HttpRuntime runtime =
            (HttpRuntime)typeof(System.Web.HttpRuntime)
                .InvokeMember("_theRuntime", BindingFlags.NonPublic |
                    BindingFlags.Static | BindingFlags.GetField,
                    null, null, null);
        if (runtime == null)
        {
            return;
        }
        string shutDownMessage =
            (string)runtime.GetType().InvokeMember("_shutDownMessage",
                BindingFlags.NonPublic | BindingFlags.Instance |
                BindingFlags.GetField, null, runtime, null);
        string shutDownStack =
            (string)runtime.GetType().InvokeMember("_shutDownStack",
                BindingFlags.NonPublic | BindingFlags.Instance |
                BindingFlags.GetField, null, runtime, null);
        if (!EventLog.SourceExists(".NET Runtime"))
        {
            EventLog.CreateEventSource(".NET Runtime", "Application");
        }
        EventLog logEntry = new EventLog();
        logEntry.Source = ".NET Runtime";
        logEntry.WriteEntry(String.Format("\r\n\r\n" +
            "shutDownMessage={0}\r\n\r\n_shutDownStack={1}",
            shutDownMessage, shutDownStack), EventLogEntryType.Error);
    }
</script>
```

在为 Global.asax 文件添加上述代码后, 启动 ASP.NET 应用程序。然后执行一些操作, 使应用程序重新启动。例如, 可以在应用程序运行的过程中修改 web.config 文件。这会触发 Application_End 事件, 并在事件日志中添加如图 3-17 所示的内容。

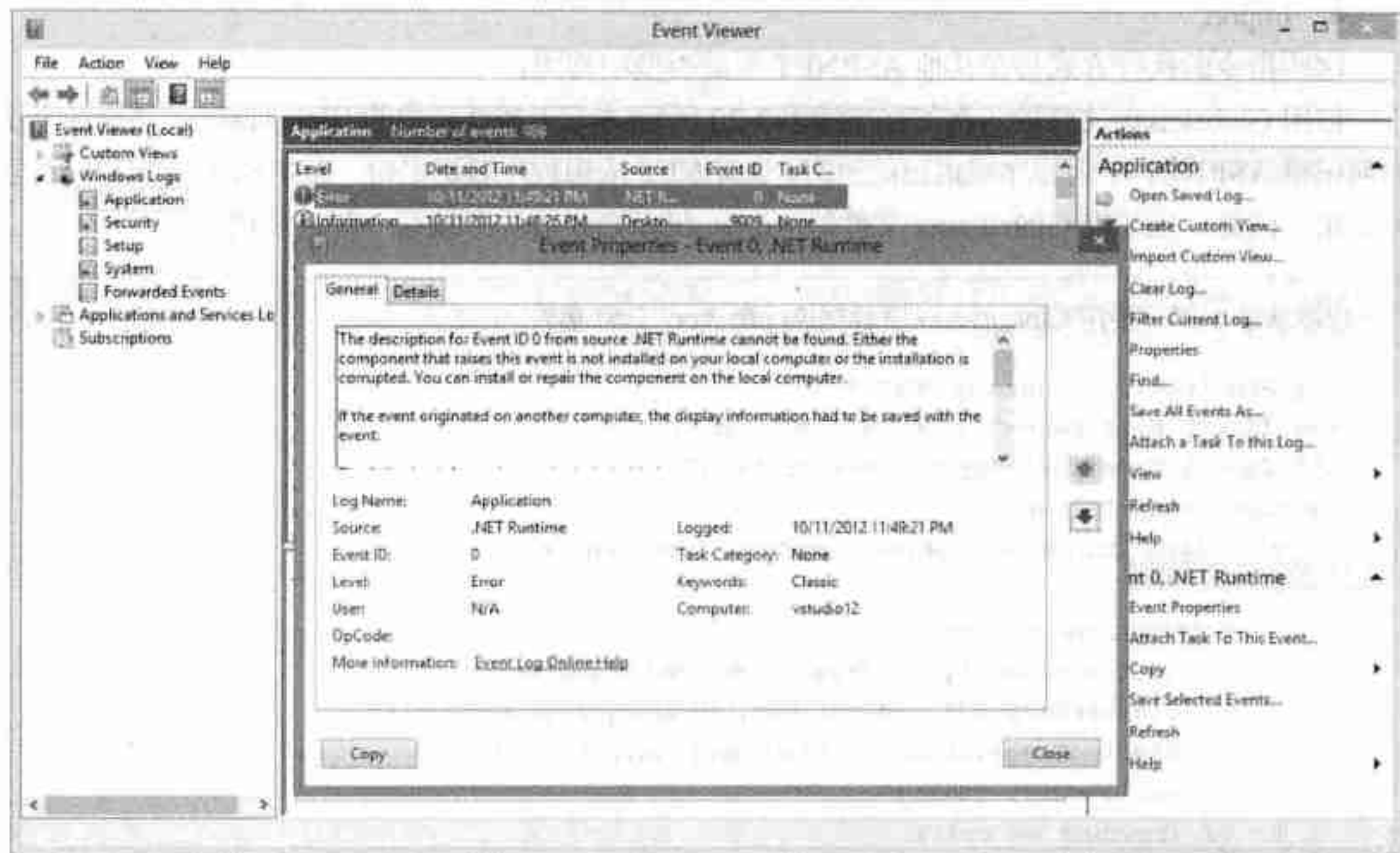


图 3-17

3.11 通过 Visual Studio 2012 使用类

本章前面已经介绍了如何在 ASP.NET 项目中使用类。在构造和使用类时，Visual Studio 2012 是相当有用的。一个特别有用的项是类设计器文件。类设计器文件的扩展名是.cd，提供了查看类、方法、属性和类包含的其他项的可视化方式。

为了查看类设计器，可以使用选择的语言创建一个新的类库项目。该类库项目含有类文件 Class1.cs。删除这个类文件，创建新的类文件 Calculator.cs。之后，创建简单的 Add()和 Subtract()方法，完成这个类的创建。这两个方法都带有两个参数(类型为 Integer)，并返回执行了相应计算的整数结果。

创建完 Calculator 类后，为这个类创建类设计器文件的最简单方式是直接在 Solution Explorer 中右击 Calculator.cs 文件，从弹出菜单中选择 View Class Diagram 命令，这会在解决方案中创建 ClassDiagram1.cd 文件。

可视化文件 ClassDiagram1.cd 如图 3-18 所示。

新的类设计器文件显示了类的设计视图。在 Visual Studio 的文档窗口中，可以看到 Calculator 类的可视化表示。这个类显示在方框中，提供了类的名称以及类中的两个方法。这个类非常简单，因此可视化视图中的内容比较有限。

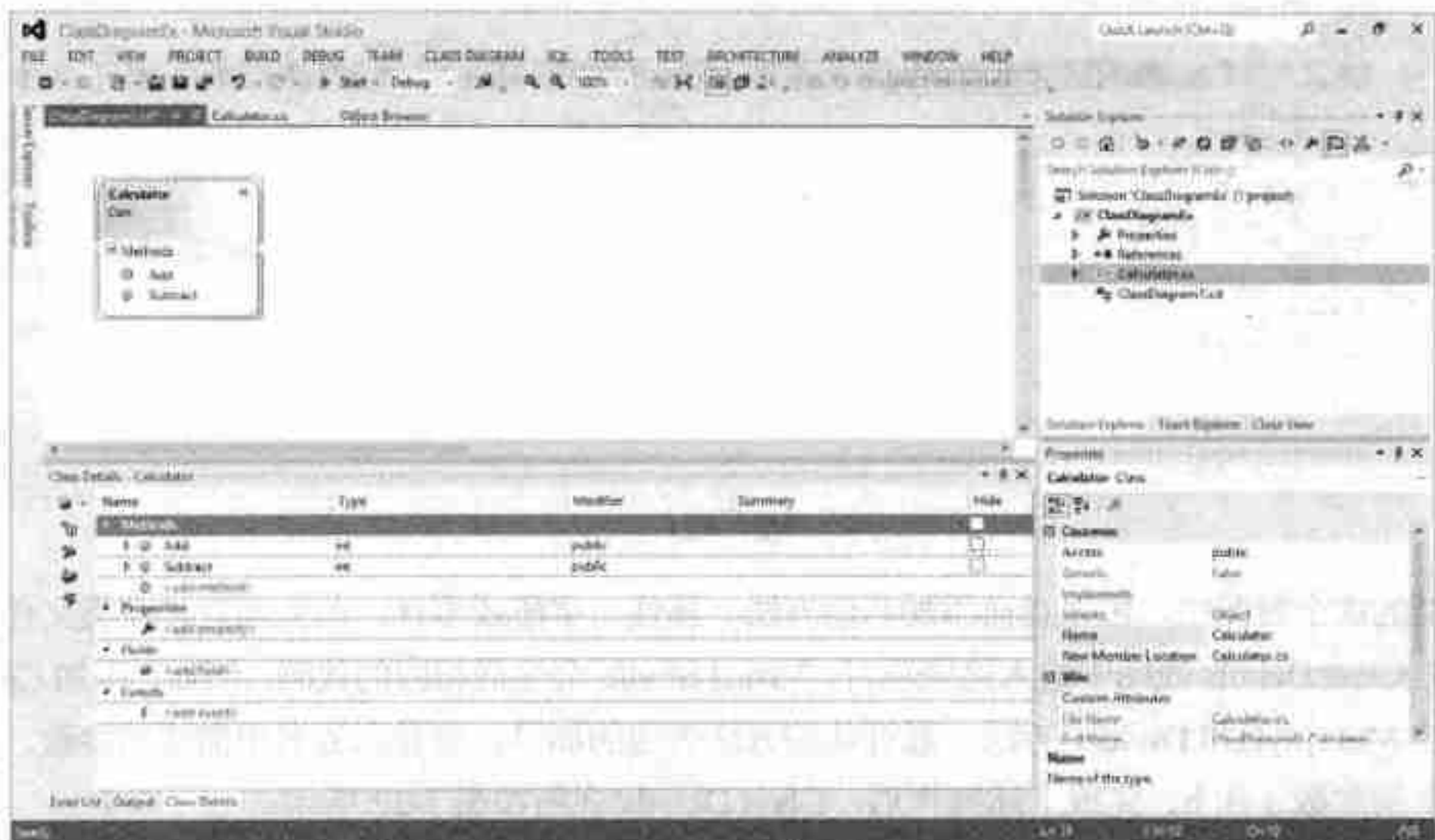


图 3-18

把类文件拖放到设计视图，就可以把其他类添加到这个类图中。接着可以根据需要在设计视图中安排类文件。还可以为继承自其他类文件的类以及派生自接口或抽象类的类建立连接。实际上，可以在类设计器中直接从刚才创建的类提取接口，方法是右击 **Calculator** 类的方框，从弹出的菜单中选择 **Refactor | Extract Interface** 命令，打开 **Extract Interface** 对话框，在该对话框中可以自定义接口的创建。这个对话框如图 3-19 所示。

单击 **OK** 按钮后，就创建了 **ICalculator** 接口，并可可视化地显示在类图中，如图 3-20 所示。

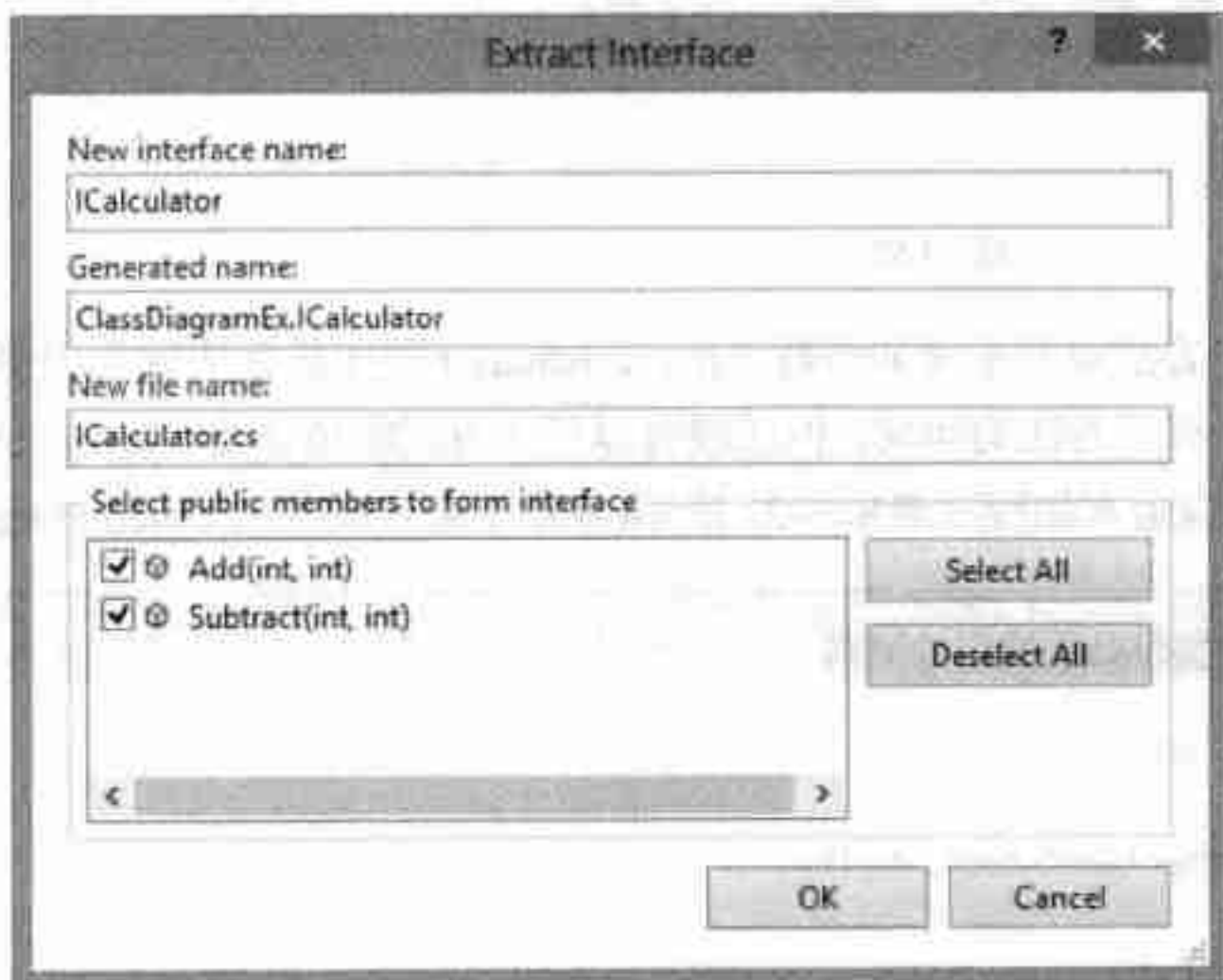


图 3-19

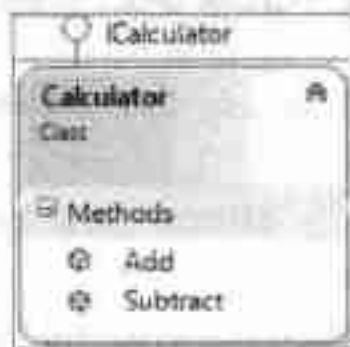


图 3-20

除了创建接口之外，还可以通过 Visual Studio 中的 **Class Details** 面板修改 **Calculator** 类，添加其他方法、属性、事件等。**Class Details** 面板如图 3-21 所示。

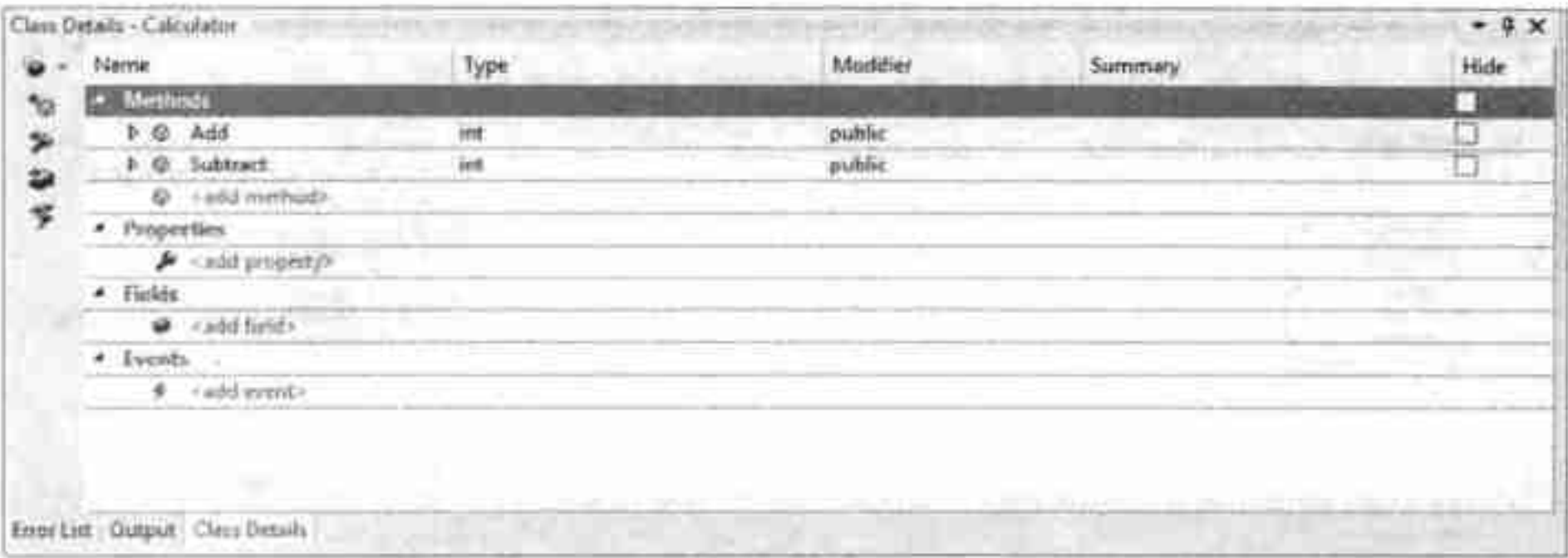


图 3-21

在类的这个视图中，可以直接添加其他方法、属性、字段或事件，而无须直接在类文件中输入代码。在 Class Details 视图中输入这些项时，Visual Studio 会生成相应的代码。例如，添加 Calculator 类需要的 Multiply()和 Divide()方法。展开这些方法旁边的加号，会显示签名中需要的参数。在这里添加需要的参数 a 和 b。完成上述操作后，Class Details 视图如图 3-22 所示。

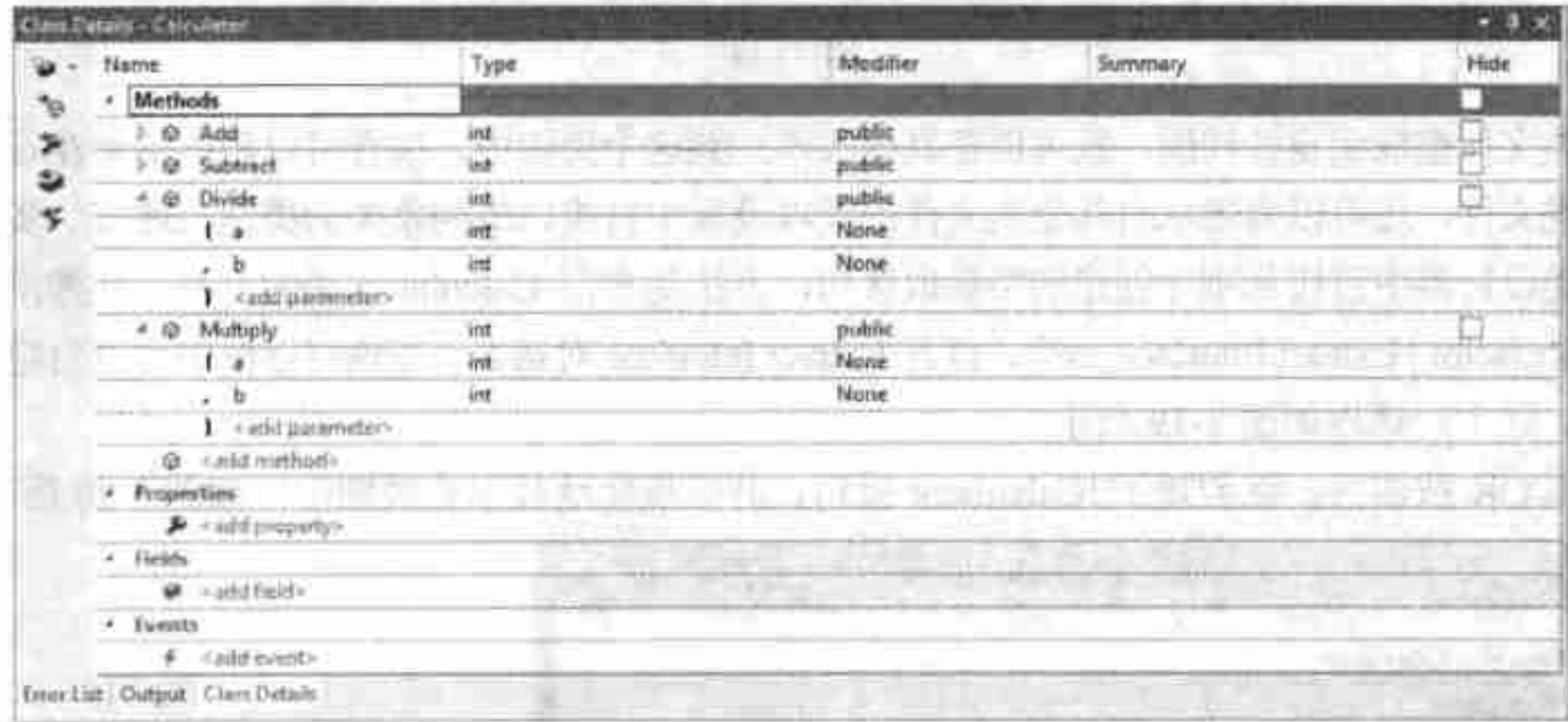


图 3-22

添加了新的 Multiply()和 Divide()方法以及所需的参数之后，Calculator 类中的代码就会改变，以指示已经包含这些新方法。现在已经有了方法的架构，但还没有以任何方式实现该类。Visual Studio 创建的 Multiply()和 Divide()方法的 C#版本如程序清单 3-21 所示(本章下载代码中的 Calculator.cs)。

程序清单 3-21 由 Visual Studio 的类设计器提供的架构

```
public int Divide(int a, int b)
{
    throw new System.NotImplementedException();
}
public int Multiply(int a, int b)
{
    throw new System.NotImplementedException();
}
```

新的类设计器提供了一种强大的方式，可以更好地查看和理解类——有时一图胜千言。cd 文件的最后一个要点是：Visual Studio 完成了处理该文件的所有工作。在 Notepad 中打开 Class-

Designer1.cd 文件，结果如程序清单 3-22 所示(本章下载代码中的 ClassDesigner1.cd)。

程序清单 3-22 在 Notepad 中打开的 ClassDesigner1.cd 文件

```
<?xml version="1.0" encoding="utf-8"?>
<ClassDiagram MajorVersion="1" MinorVersion="1">
  <Class Name="ClassDiagramEx.Calculator">
    <Position X="0.5" Y="0.5" Width="1.5" />
    <TypeIdentifier>
      <HashCode>AAIAAAAAAQAAAAAAAAADAAAAAAAAAAAAAAAAAAAA= </HashCode>
      <FileName>Calculator.cs</FileName>
    </TypeIdentifier>
    <Lollipop Position="0.2" />
  </Class>
  <Font Name="Segoe UI" Size="9" />
</ClassDiagram>
```

这是相当简单的 XML 文件，定义了类和与该类连接的项的位置。

3.12 本章小结

本章介绍了许多基础知识，讨论了一些与整个 ASP.NET 应用程序相关的问题，以及建立和部署新应用程序的一些方法。有了 Visual Studio 2012 的帮助，我们可以在创建应用程序时选择使用哪个 Web 服务器；也可以通过内置的 FTP 功能，确定是在本地工作，还是在远程工作。

通过 ASP.NET 4.5 和 Visual Studio 2012 可以轻松地使用内联编码模型来创建页面，或者选择比过去更易于使用和部署的隐藏代码模型。本章还介绍了跨页面传送功能和 ASP.NET 4.5 中新的文件夹，它们简化了开发人员的日常工作。这些文件夹动态地使其资源可用，而我们无须进行任何操作。本章另外介绍了一些杰出的新编译选项。最后，本章探讨了 Visual Studio 2012 使开发人员更便于使用项目中的类的方式。

第 II 部分

控 件

- 第 4 章 ASP.NET 服务器控件和客户端脚本
- 第 5 章 ASP.NET Web 服务器控件
- 第 6 章 验证服务器控件
- 第 7 章 用户控件和服务端控件

第 4 章

ASP.NET 服务器控件和客户端脚本

本章要点

- 使用服务器控件构建 ASP.NET 页面
- 使用 HTML 服务器控件
- 识别服务器控件
- 使用 JavaScript 修改服务器控件

如前一章所述, ASP.NET 是从 Microsoft 早期的 Web 技术 Active Server Pages(称为 ASP, 现在称为传统的 ASP)演化而来的。这个模型与现在的 ASP.NET 完全不同。传统的 ASP 使用解释性语言完成最终的 HTML 文档的构建, 之后把它发送到浏览器上。而 ASP.NET 使用真正的编译语言完成这项任务。本章主要讨论在编译环境下创建基于对象的 Web 页面的理念。

本章还将介绍如何在 ASP.NET 页面中使用一种特定类型的对象, 即服务器控件, 以及如何用好这种控件。同时还将介绍一种特殊类型的服务器控件: HTML 服务器控件。本章也将描述如何在 ASP.NET 页面中使用 JavaScript 改变服务器控件的行为。

本章的剩余部分将讨论如何以可视化和编程的方式使用和操作服务器控件, 以帮助创建 ASP.NET 页面。

4.1 ASP.NET 服务器控件

过去, 使用传统 ASP 的一个难点是, 必须根据所编写的服务器端代码对浏览器输出的所有 HTML 结果负全责。这看起来很理想, 但是存在一个问题, 因为每个浏览器都以略微不同的方式来解释提供给它的 HTML。

目前的两种主流浏览器是 Internet Explorer 和 Netscape Navigator。这意味着开发人员不仅要确定输出 HTML 的浏览器类型, 还必须考虑对应用程序发出请求的浏览器的版本。一些开发人员为了解决这个问题, 创建了两个不同的应用程序。当终端用户第一次对应用程序发出请求时, 代码就检查浏览器, 确定发出请求的浏览器的类型。接着, ASP 页面把 IE 用户重定向到一条路径上, 而把 Netscape 用户重定向到另一条路径上。

由于请求可能来自同一个浏览器的许多不同版本，因此开发人员常常为用于浏览站点的最低版本开发应用程序。基本上，每个人都把使用最低版本作为目标。这种技术可确保页面在大多数发出请求的浏览器上正确显示，但它迫使开发人员向下兼容其应用程序。如果应用程序总是为最低版本开发，开发人员就不能利用新浏览器版本所提供的高级功能。

ASP.NET 服务器控件克服了这些弊端。在使用 ASP.NET 提供的服务器控件时，不必在服务器端代码中指定要输出的 HTML，而是指定要在浏览器上显示的功能，让 ASP.NET 决定发送到浏览器上的输出。

在收到一个请求时，ASP.NET 会检查这个请求，确定发出该请求的浏览器的类型以及版本，然后生成特定于该浏览器的 HTML 输出。处理从 HTTP 请求获取的用户代理报头以确定要发送给浏览器的内容，以此来完成这一过程。也就是说，可以为最好的浏览器建立应用程序，而不必担心给应用程序发出请求的浏览器是否支持该功能。有了前面描述的功能，就可以将这些控件称为智能控件。

4.1.1 服务器控件的类型

ASP.NET 提供了两种不同的服务器控件：HTML 服务器控件和 Web 服务器控件。这两种类型的控件大不相同，在使用 ASP.NET 时就会看出最有用的是 Web 服务器控件。这并不是说 HTML 服务器控件没有用，它们也提供了许多功能，并且其中一些功能 Web 服务器控件并不具备。

那么，哪种控件比较好呢？答案完全取决于所要获得的结果。HTML 服务器控件会映射为特定的 HTML 元素。可以在 ASP.NET 页面上放置 HtmlTable 服务器控件，动态地处理<table>元素。另一方面，Web 服务器控件映射为 ASP.NET 页面上需要的特定功能。也就是说，<asp:Panel>控件可以使用<table>或另一个元素，这实际上取决于发出请求的浏览器的功能。

表 4-1 总结了使用 HTML 服务器控件和 Web 服务器控件的场合。

控 件 类 型	使用该控件类型的场合
HTML 服务器控件	将传统的 ASP 3.0 Web 页面转换为 ASP.NET Web 页面并且转换速度要求比较高时，可以使用该类型的控件。将 HTML 元素转换为 HTML 服务器控件，比将它们转换为 Web 服务器控件容易得多。在比较喜欢 HTML 类型的编程模型时使用，以及在希望显式地控制为浏览器生成的代码时使用。但是，简单地运用 ASP.NET MVC(将会在第 34 章讲述)解决这个问题可能会更好。
Web 服务器控件	在需要更丰富的功能集来执行复杂的页面请求时使用。 开发可使用多种浏览器类型查看的 Web 页面并且需要根据不同类型使用不同代码时使用。 在比较喜欢 Visual Basic 类型的编程模型时使用，该编程模型使用控件和控件属性。

当然，有些开发人员喜欢把一些控件与其他控件分开，将它们放在各自的类别中。例如，引用下述类型的控件：

- 列表控件：这种控件允许绑定数据，以达到某种显示目的。
- Rich 控件：像 Calendar 这样的控件，其显示的内容和功能比其他控件丰富。
- 验证控件：这些控件与其他窗体控件交互，以验证它们包含的数据。
- 用户控件：它们并不是真正的控件，而是页面模板，可以像使用 ASP.NET 页面上的服务器控件那样使用它们。

- **自定义控件**: 自己建立的控件, 其使用方式与 ASP.NET 4.5 默认安装所提供的 ASP.NET 服务器控件的使用方式相同。

在决定是使用 HTML 服务器控件还是 Web 服务器控件时, 并没有什么硬性规则。可能一种控件使用得比较多, 另一种控件使用得比较少, 但是一种控件提供的某些功能可能在另一种控件中并不存在。如果要完成特定的任务, 而当前使用的控件类型无法完成, 就可以试着使用另一种控件, 因为这样很可能完成该任务。还要注意, 可以混合和匹配这些控件类型。在同一页面或应用程序中完全可以同时使用 HTML 服务器控件和 Web 服务器控件。

4.1.2 使用服务器控件构建页面

使用服务器控件构建 ASP.NET 页面有两种方式。可以使用专门为处理 ASP.NET 4.5 而设计的工具, 这些工具允许可视化地将控件拖放到设计界面上, 并操纵该控件的行为。也可以直接通过输入代码来处理服务器控件。

1. 在设计界面上使用服务器控件

Visual Studio 2012 允许将控件拖放到设计界面上, 可视化地创建 ASP.NET 页面。要获得这个可视化的设计选项, 可以在查看 ASP.NET 页面时单击 IDE 底部的 Design 选项卡。也可以在同一个文档窗口中显示设计(Design)视图和源代码(Source)视图, 这个功能在 Visual Studio 2008 中引入。在设计视图中, 可以把光标放在页面上希望控件出现的任何地方, 再在 Visual Studio 的 Toolbox 窗口中双击需要的控件。

在页面的设计视图中, 可以突出显示一个控件, 该控件的属性会显示在 Properties 窗口中。如图 4-1 所示, 在设计视图中选择 Button 控件, 其属性就显示在右下角的 Properties 窗口中。



图 4-1

在这个窗口中修改属性, 就会改变突出显示的控件的外观或行为。因为所有的控件都继承自一个特定的基类(WebControl), 所以还可以同时突出显示多个控件, 并一次性改变这些控件的基本属性。在选择多个控件时, 需要按住 Ctrl 键。

2. 编写服务器控件

还可以直接在源代码视图中工作。许多开发人员都喜欢这么做，因此这是第一次创建 ASP.NET 页面时的默认方式。手动编写自己的 ASP.NET 页面似乎比把控件拖放到设计视图中慢一些，但这并不像我们想象得那么慢。在 Visual Studio 2012 中编写应用程序时会得到许多帮助。当在 Visual Studio 中输入代码时，IntelliSense 功能就会帮助我们自动完成代码的输入。例如，图 4-2 显示了在输入代码时出现的 IntelliSense 下拉列表，其中包含可能的完整代码语句。

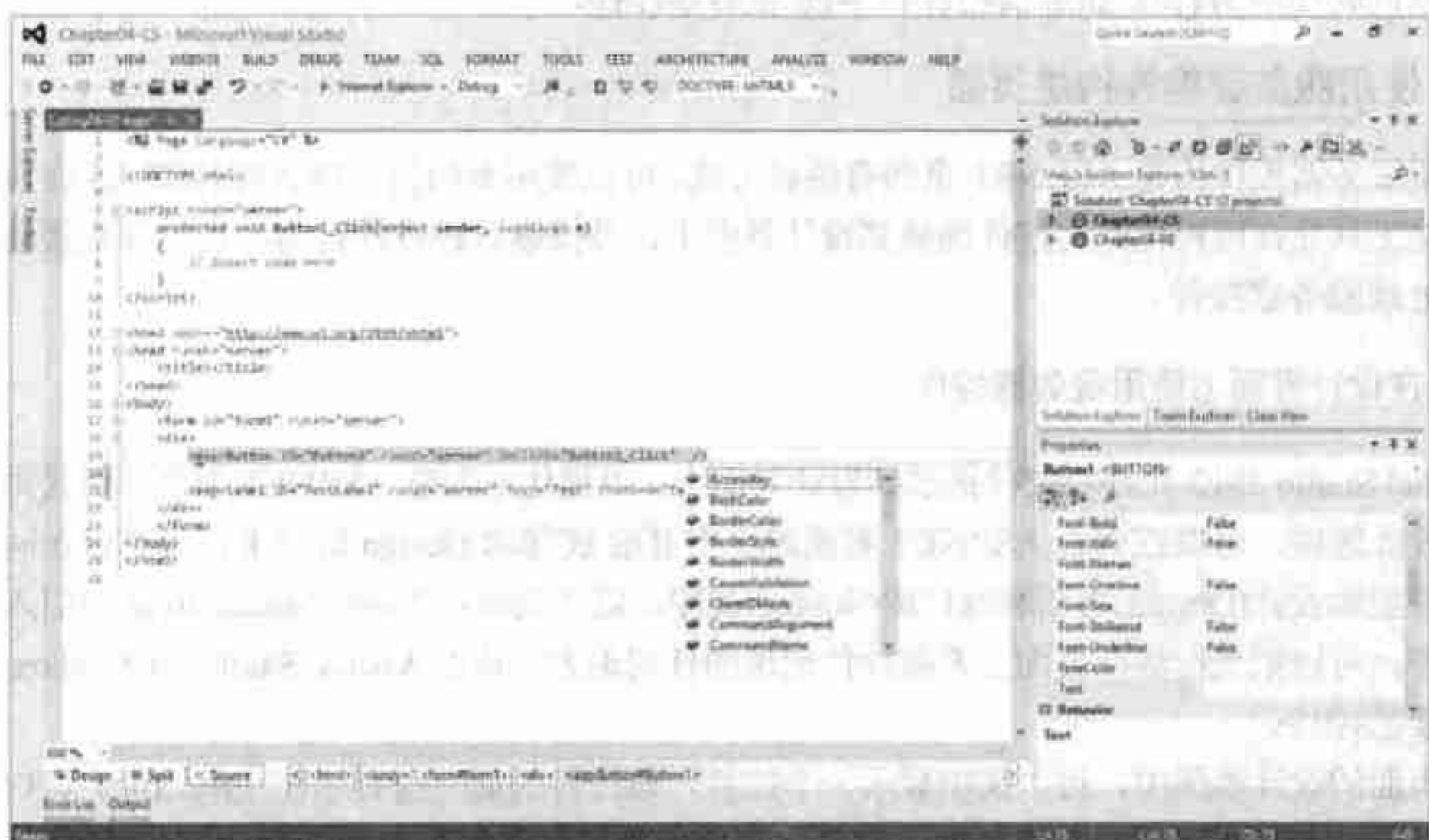


图 4-2

IntelliSense 会列出当前处理的控件或代码的最常用属性或语句。在工作时高效地使用 IntelliSense，可以大大加快编码的速度。

与设计视图一样，页面的源代码视图也允许把控件从 Toolbox 窗口拖放到代码页面上。例如，把一个 TextBox 控件拖放到代码页面上，与把它拖放到设计视图中的效果相同：

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
```

也可以在源代码视图中突出显示一个控件，或者只是把光标放在该控件的代码语句上，Properties 窗口就会显示该控件的属性。现在就可以直接应用 Visual Studio 的 Properties 窗口中的属性，这些属性会动态地添加到控件的代码中。

4.1.3 处理服务器控件的事件

ASP.NET 使用事件驱动模型。项或编码任务只在特定事件发生时执行。ASP.NET 编程模型中的常见事件是 Page_Load，如程序清单 4-1 所示。

程序清单 4-1 处理特定的页面事件

```
protected void Page_Load(Object sender, EventArgs e)
{
    // Insert code here
}
```


其实, 不仅可以使整个页面及其在页面事件的某个特定时刻的属性和方法, 而且可以通过特定的控件事件使用页面上包含的服务器控件。例如, 窗体上按钮的一个常见事件是 `Button_Click`, 如程序清单 4-2 所示。

程序清单 4-2 处理按钮单击事件

```
protected void Button1_Click(object sender, EventArgs e)
{
    // Insert code here
}
```

程序清单 4-2 中所示的事件仅在终端用户单击了窗体上的按钮时才触发, 该按钮的 `OnClick` 属性值是 `Button1_Click`。因此, 事件处理程序不仅存在于 ASP.NET 页面的服务器端代码中, 还使用相关的 ASP.NET 页面标记中的服务器控件的 `OnClick` 属性关联起来, 如下面的代码所示:

```
<asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1_Click" />
```

如何触发服务器控件的这些事件呢? 有两种方式。第一种方式是在设计视图中打开 ASP.NET 页面, 双击要创建服务器端事件的控件。例如, 双击设计视图中的 `Button` 服务器控件, 无论代码是在隐藏代码文件中还是内联编码, 都会在服务器端代码中创建 `Button1_Click` 事件的结构。这会为该服务器控件最常用的事件创建处理程序架构。

如前所述, 注意 `Button` 控件有非常多的事件, 双击该控件并不能得到这些事件。为了能够从 IDE 内的所有视图中访问这些事件, 需要首先在 `Properties` 窗口中选择该控件。稍后会看到一个闪电状的图标, 它会提供一个带有所有控件事件的下拉列表, 之后就可以双击感兴趣的事件, 并且 `Visual Studio` 会创建所需要的功能架构。图 4-3 显示了该事件下拉列表。例如, 处理 `Button` 控件的 `PreRender` 事件, 而不是其 `Click` 事件。该事件的处理程序会放在服务器端代码中。

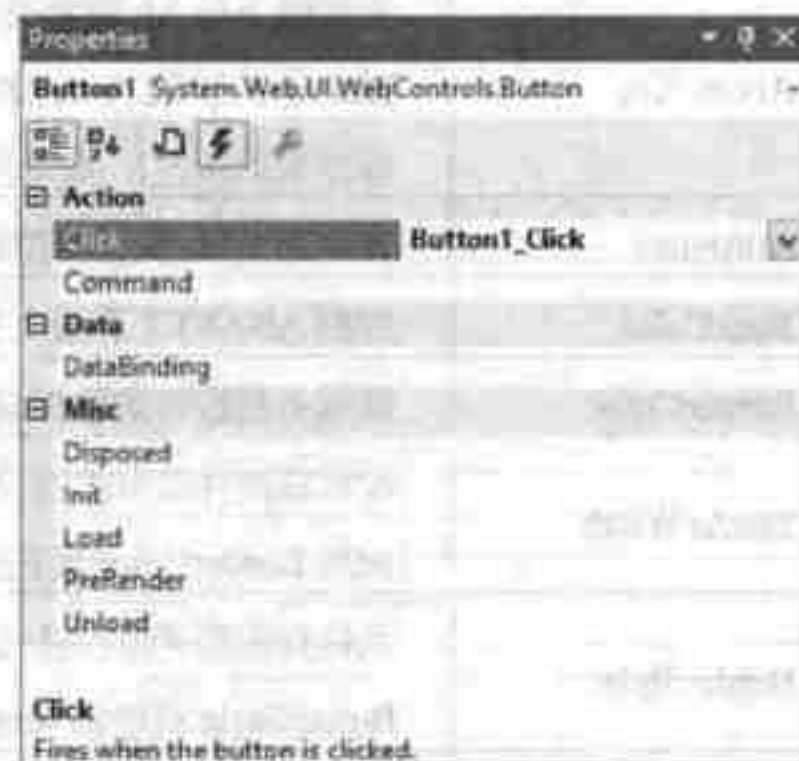


图 4-3

有了事件的结构后, 就可以编写触发事件时希望发生的特定操作。

4.2 给服务器控件应用样式

我们常常希望改变应用程序中实现的服务器控件的默认样式(它们基本上没有样式), 并且希望所建立的应用程序能反映自己的外观和操作方式。自定义页面控件外观的一种方式就是更改控件的属性。

如本章前面所述, 要获得某个控件的属性, 只需在 `Visual Studio` 的页面的设计视图中突出显示该控件即可。如果位于源代码视图中, 把光标放在该控件的代码上即可。`Properties` 窗口中显示的属性允许控制所选控件的外观和行为。



目前的大多数网站和应用程序都使用 CSS 代替内联样式。CSS 的更多信息可参阅第 2 章。

研究控件的常见属性

ASP.NET 4.5 附带的许多默认服务器控件都派生于 WebControl 类，并且拥有类似的属性，这些属性可以用于改变它们的外观和行为。并不是所有的派生控件都可以使用所有的可用属性(尽管许多控件都拥有这些属性)。另一个要点是并不是所有的服务器控件都是从 WebControl 类实现的。例如，Literal、Placeholder、Repeater 和 XML 服务器控件就不是派生自 WebControl 基类，而是派生自 Control 类。

HTML 服务器控件也不是派生自 WebControl 基类，因为它们主要用于设置 HTML 元素的特性。表 4-2 列出了服务器控件具有的公共属性。

表 4-2

属 性	说 明
AccessKey	允许赋予与 Alt 键相关的一个字符，这样终端用户就可以使用键盘上的快捷键激活控件。例如，给 Button 控件的 AccessKey 属性赋值 K。这样终端用户就不需要单击 ASP.NET 页面上的按钮(使用鼠标控制的指针)，而是简单地按下 Alt+K 快捷键即可
Attributes	允许为 Web 服务器控件定义公共属性未定义的额外属性
BackColor	控制 ASP.NET 页面上控件的背景色
BorderColor	给服务器控件的边框设置颜色
BorderWidth	给组成控件边框的线设置线宽值。把一个数字设置为该值，就是把该数字设置为边框的宽度像素值。如果 BorderColor 属性没有与 BorderWidth 属性设置一起使用，默认的边框颜色就是黑色
BorderStyle	允许指定服务器控件边框的设计样式。边框默认创建为直线，但可以给边框设置许多不同的样式。BorderStyle 属性的其他值包括 Dotted、Dashed、Solid、Double、Groove、Ridge、Inset 和 Outset
ClientIDMode	允许获得或设置用于生成 ClientID 属性值的算法
CssClass	给控件指定自定义的 CSS(Cascading Style Sheet，层叠样式表)类
Enabled	允许在把这个属性的值设置为 False 时关闭控件的功能。默认设置为 True
EnableTheming	允许为所选的服务器控件打开主题功能。默认值是 True
EnableViewState	指定是否为该控件保持视图状态
Font	设置控件中所有文本的字体
ForeColor	设置控件中所有文本的颜色
Height	设置控件的高度
SkinID	给控件应用主题时设置要使用的外观
Style	允许把 CSS 样式应用于控件
TabIndex	设置控件在 ASP.NET 页面中的制表符位置。这个属性要与页面上的其他控件一起使用
ToolTip	设置当鼠标指针停留在控件上一小段时间时，指定出现在浏览器内的一个黄色框中的文本，它可以为终端用户提供更多的指示
Width	设置控件的宽度

许多服务器控件都包含这些公共属性。WebControl 类的一些属性可以直接操纵 ASP.NET 内置的主题系统,如 EnableTheming 和 SkinID 属性。一些控件还有其他特定的属性。掌握表 4-2 中列出的属性有助于快速使用 Web 服务器控件,并根据需要修改它们。

4.3 CSS 在 ASP.NET 4.5 中的改变

在 ASP.NET 3.5 和以前版本中,ASP.NET 服务器控件显示的 HTML 并非总是遵循最新的 HTML 标准。例如,在 ASP.NET 3.5 中禁用服务器控件时,只需将服务器控件的 Enabled 属性设置为 False。这样就可以在页面上显示控件,但是其 disabled 属性却如下所示:

```
<span id="Label1" disabled="disabled">Hello there!</span>
```

这不是有效的 HTML,因为 span 元素没有 disabled 属性。

在 ASP.NET 4 和 4.5 中,web.config 文件中的<pages>元素现在有一个属性,它可以在显示控件时指示 ASP.NET 使用何种版本样式。其默认值是 4.0,如下所示:

```
<pages controlRenderingCompatibilityVersion="4.0" />
```

如果使用上面的代码行将该属性设置为 4.0,那么 ASP.NET 将会正确地使用 CSS 禁用该控件,如下所示:

```
<span id="Label1" class="aspNetDisabled">Hello there!</span>
```



controlRenderingCompatibilityVersion 属性值的格式是一个数字和一个小数位。小于 4.0 的值会显示 disabled 属性,大于等于 4.0 的值会显示 class 属性。

此时,ASP.NET 设置的是 class 属性,而不是 disabled 属性。但是,如果想改为使用原有的控件显示方式,可以把 controlRenderingCompatibilityVersion 属性的值设置为 3.5。

4.4 HTML 服务器控件

ASP.NET 允许用户提取 HTML 元素,通过少量的工作,把它们转换为服务器控件。之后,就可以使用它们控制在 ASP.NET 页面中实现的元素的行为和操作。

当然,可以把需要的任意 HTML 元素都放在页面上。也可以把页面上的 HTML 元素用作服务器控件。在 Visual Studio 的 Toolbox 中包含了一个 HTML 元素列表,如图 4-4 所示。

在 Document 窗口中,把这些 HTML 元素中的任何一个从 Toolbox 拖放到 ASP.NET 页面的设计或源代码视图中,就可以生成相应的 HTML 元素。例如,把一个 HTML Button 控件拖放到页面上,就会在代码中生成如下结果:



图 4-4


```
<input id="Button1" type="button" value="button" />
```

在此状态下, Button 控件不是服务器端控件, 而只是 HTML 元素。可以轻易地把它转换为 HTML 服务器控件。在源代码视图中, 只需添加 `runat="server"`, 就可以把 HTML 元素转换为控件:

```
<input id="Button1" type="button" value="button" runat="server" />
```

元素转换为服务器控件之后(通过添加 `runat="server"` 属性和值), 就可以像处理任何 Web 服务器控件那样处理所选的元素。程序清单 4-3 是一些 HTML 服务器控件的示例。

程序清单 4-3 使用 HTML 服务器控件

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Button1_ServerClick(object sender, EventArgs e)
    {
        Response.Write("Hello " + Text1.Value);
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Using HTML Server Controls</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            What is your name?<br />
            <input id="Text1" type="text" runat="server" />
            <input id="Button1" type="button" value="Submit" runat="server"
                onclick="Button1_ServerClick" />
        </div>
    </form>
</body>
</html>
```

在这个例子中, 页面上有两个 HTML 服务器控件。它们都是一般的 HTML 元素, 只是添加了 `runat="server"` 属性。如果要把 HTML 元素用作服务器控件, 就必须包含 `id` 属性, 这样才能在服务器端代码中识别出服务器控件。

Button 控件使用 `OnClick` 属性包含对服务器端事件的引用。这个属性指向的服务器端事件在终端用户单击按钮时触发, 在这里是 `Button1_ServerClick` 事件。在 `Button1_ServerClick` 方法中, 使用 `Value` 属性输出文本框中的值。

4.4.1 HtmlControl 基类

所有的 HTML 服务器控件都使用一个派生自 `HtmlControl` 基类(完整名称是 `System.Web.UI.HtmlControls.HtmlControl`)的类。这些类从控件的派生类继承了许多属性。表 4-3 列出了从这个基类继承的一些属性, 其中一些属性本身派生自 `Control` 基类。

表 4-3

方法或属性	说 明
Attributes	为控件中指定的所有可用属性提供名称/值对的集合, 包括自定义属性
Disabled	允许使用 Boolean 值设置是否禁用控件
EnableTheming	允许使用 Boolean 值设置控件是否参与页面主题功能
EnableViewState	允许使用 Boolean 值设置控件是否参与页面的视图状态功能
ID	获取或设置控件的唯一标识符
Page	获取包含特定服务器控件的 Page 对象的引用
Parent	在页面控件层次结构中获取对父控件的引用
Site	提供服务器控件所属的 Web 站点的信息
SkinID	EnableTheming 属性设置为 True 时, SkinID 属性指定在设置主题时使用的外观文件
Style	引用应用于特定控件的 CSS 样式集合
TagName	提供从指定控件生成的元素名
Visible	指定控件在生成的页面上是否可见



还有更多的属性。更完整的列表请参阅 MSDN 文档。

4.4.2 HtmlContainerControl 类

基类 HtmlContainerControl 用于一些 HTML 类, 这些类主要包括可以包含在单个节点中的 HTML 元素。例如, 、<input> 和 <link> 元素用于派生自 HtmlControl 类的类。

其他 HTML 元素, 例如 <a>、<form> 和 <select>, 需要一对起始和结束标记。这些元素使用派生自 HtmlContainerControl 类的类, HtmlContainerControl 类是为专门处理需要结束标记的 HTML 元素而设计的。

HtmlContainerControl 类因为派生自 HtmlControl 类, 所以拥有 HtmlControl 类的所有属性和方法, 还拥有一些在自身中声明的新项。其中最重要的是 InnerText 和 InnerHtml 属性。

- InnerHtml: 允许用户指定包含 HTML 元素的内容, 这些 HTML 元素放在特定控件的起始和结束标记之间。
- InnerText: 允许用户指定要放在特定控件的起始和结束标记之间的纯文本。

4.4.3 所有的 HTML 类

可以使用所有的 HTML 元素, 因为它们都有对应的类。表 4-4 列出了部分用来处理 HTML 的服务器控件类:

表 4-4

类	显示的 HTML 元素
HtmlAnchor	<a>
HtmlArea	<area>
HtmlAudio	<audio>
HtmlButton	<button>
HtmlForm	<form>
HtmlHead	<head>
HtmlIframe	<iframe>
HtmlImage	
HtmlInputButton	<input type="button">
HtmlInputCheckBox	<input type="checkbox">
HtmlInputFile	<input type="file">
HtmlInputHidden	<input type="hidden">
HtmlInputImage	<input type="image">
HtmlInputPassword	<input type="password">
HtmlInputRadioButton	<input type="radio">
HtmlInputReset	<input type="reset">
HtmlInputSubmit	<input type="submit">
HtmlInputText	<input type="text">
HtmlLink	<link>
HtmlMeta	<meta>
HtmlSelect	<select>
HtmlSource	<source>
HtmlTable	<table>
HtmlTableCell	<td>
HtmlTableRow	<tr>
HtmlTextArea	<textarea>
HtmlTitle	<title>
HtmlTrack	<track>
HtmlVideo	<video>

把 HTML 元素转换为 HTML 服务器控件时，可以访问上述列表中的类。例如，按照以下方式把<title>元素转换为服务器控件：

```
<title id="Title1" runat="server"/>
```

此时可以访问这个 HTML 元素的 HtmlTitle 类。使用这个类实例，可以执行许多任务，包括给页面标题动态地提供文本值：


```
Title1.Text = DateTime.Now.ToString();
```

使用这些类可以获得所需要的大多数 HTML 元素,但这些 HTML 类并没有显式包含相当多的其他 HTML 元素。例如,HtmlGenericControl 类可以对任意 HTML 元素进行服务器端的访问。

4.4.4 使用 HtmlGenericControl 类

注意 HtmlGenericControl 类的重要性:它的一些功能不能从 ASP.NET 提供的其他服务器控件获得。例如,使用 HtmlGenericControl 类可以对<meta>、<p>、或其他元素进行服务器端的访问,但使用其他类不能进行这样的访问。

程序清单 4-4 说明了如何使用 HtmlGenericControl 类修改页面中新的 HTML5 元素<progress>。

程序清单 4-4 使用 HtmlGenericControl 类修改<progress>元素

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        Progress1.Attributes["max"] = "100";
        Progress1.Attributes["value"] = "25";
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Using the HtmlGenericControl class</title>
</head>
<body>
    <form id="form1" runat="server">
        <progress id="Progress1" runat="server"></progress>
    </form>
</body>
</html>
```

在这个例子中,为页面的<meta>元素添加了 id 和 runat 属性,从而将其转换为 HTML 服务器控件。因为 HtmlGenericControl 类(派生自 HtmlControl 类)可以处理许多 HTML 元素,所以不能用处理其他 HTML 类(如 HtmlInputButton)的方式给 HTML 属性赋值,而必须使用 HtmlGenericControl 类的 Attributes 属性给 HTML 元素的属性赋值,把要处理的属性指定为字符串值。

下面是运行示例页面的部分结果:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta id="Meta1" Name="description"
        CONTENT="Generated on: 2/5/2010 2:42:52 PM"></meta>
    <title>Using the HtmlGenericControl class</title>
</head>
```

使用 HtmlGenericControl 类和其他 HTML 类,可以通过服务器端代码操作 ASP.NET 页面上的所有元素。

4.5 识别 ASP.NET 服务器控件

当使用一系列控件构建ASP.NET页面时，许多控件的嵌套，甚至许多控件在页面上的动态布局都是由ASP.NET自身完成的。例如，在使用用户控件(如GridView、ListView、Repeater等)时，ASP.NET会构建复杂的控件树，从而在页面上显示这些控件。

当发生这种情况时，ASP.NET 需要为这些动态控件提供 ID，此时会生成 GridView1\$ctl02\$ctl00 这样的 ID。与控件一样，ID 也是动态的，因为 ID 难以预测，生成的 ID 难以通过客户端代码使用这些控件。

为了解决这个问题，ASP.NET 4.5 可以控制控件所使用的 ID。为了说明这个功能，程序清单 4-5 给出了一些代码，这些代码将为控件产生一些难以预测的客户 ID。首先，创建用户控件。

程序清单 4-5 包含一些简单控件的用户控件

```
<%@ Control Language="C#" ClassName="Listing04_05" %>
<asp:TextBox ID="TextBox1" runat="server" />
<br />
<asp:Button ID="Button1" runat="server" Text="Button" />
```

之后，在 ASP.NET 页面中使用该用户控件，如程序清单 4-6 所示。

程序清单 4-6 在简单的 ASP.NET 页面中使用用户控件

```
<%@ Page Language="C#" Trace="true" %>
<%@ Register src="Listing04-05.ascx" tagname="MyUserControl" tagprefix="muc" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Working with Control IDs</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <muc:MyUserControl ID="MyUserControl1" runat="server" />
        </div>
    </form>
</body>
</html>
```

此时，页面上的该用户控件仅包含两个简单的服务器控件，它们之后会在页面上显示出来。从程序清单4-5中可以看出，它们的控件ID非常简单。其中，TextBox服务器控件的ID值是TextBox1，Button服务器控件的ID值是Button1。

从程序清单 4-6 的页面代码中可以看到，在 page 指令中将 Trace 特性的值设置为 true，这就可以看到在页面控制树中产生的 ClientID。运行该页面，效果如图 4-5 所示。

Control Tree	
Control UniqueID	Type
Page	ASP.listing04_06.aspx
ctl02	System.Web.UI.LiteralControl
ctl00	System.Web.UI.HtmlControls.HtmlHead
ctl01	System.Web.UI.HtmlControls.HtmlTitle
ctl03	System.Web.UI.LiteralControl
form1	System.Web.UI.HtmlControls.HtmlForm
ctl04	System.Web.UI.LiteralControl
MyUserControl1	ASP.listing04_05
MyUserControl1\$TextBox1	System.Web.UI.WebControls.TextBox
MyUserControl1\$ctl00	System.Web.UI.LiteralControl
MyUserControl1\$Button1	System.Web.UI.WebControls.Button
ctl05	System.Web.UI.LiteralControl
ctl06	System.Web.UI.LiteralControl

图 4-5

如果检查页面的源代码，就会看到下面的代码片段：

```
<div>
<input name="MyUserControl1$TextBox1" type="text" id="MyUserControl1_TextBox1" />
<br />
<input type="submit" name="MyUserControl1$Button1" value="Button"
       id="MyUserControl1_Button1" />
</div>
```

从中可以看出，ASP.NET 分配的控件 ID 较长，一般不会选择使用。TextBox 服务器控件的输出包含 name 值 WebUserControl1\$TextBox1 和 id 值 WebUserControl1_TextBox1。所做的这些操作都是为了确保控件最终得到唯一的 ID。

ASP.NET 4.5 可以使用 ClientIDMode 属性来控制这些赋值。ClientIDMode 属性的值可以是 AutoID、Inherit、Predictable 或 Static。下面是设置 ClientIDMode 属性值的一个示例：

```
<muc:MyUserControl ID="MyUserControl1" runat="server" ClientIDMode="AutoID" />
```

这个示例将 ClientIDMode 属性的值设为 AutoID，迫使命名操作遵守 .NET Framework 3.5 及其以前版本中的命名方式。使用 AutoID 可以得到以下结果：

- name: MyUserControl1\$TextBox1 MyUserControl1\$Button1
- id: MyUserControl1_TextBox1 MyUserControl1_Button1

如果将该属性设置为 Inherit，那么仅仅是继承容器控件、页面或应用程序使用的命名方式，这样可以得到与使用 AutoID 时一样的值。Inherit 是所有控件的默认值。

Predictable 通常用于嵌套其他控件的数据绑定控件(如 Repeater 控件)。当结合使用 ClientIDRowSuffix 属性值时，将附加这个值，而不是递增数字(例如，ctrl1、ctrl2)。

Static 可以给出赋值的控件的名称，开发人员应确保标识符的唯一性。当把用户控件的 ClientIDMode 属性设置为 Static 时，将会得到以下结果：

- name: MyUserControl1\$TextBox1 MyUserControl1\$Button1
- id: TextBox1 Button1

还可以通过 machine.config 或 web.config 文件中的 <pages> 元素，在控件、容器控件、用户控件、页面，甚至应用程序级设置 ClientID 属性。

有了这一功能，使用像 JavaScript 这样的技术在客户端处理服务器控件，就会比以前容易很多。下一节将讨论 JavaScript 在 ASP.NET 中的应用。

4.6 通过 JavaScript 处理页面和服务器控件

开发人员一般喜欢在 ASP.NET 页面上包含一些自定义的 JavaScript 函数。完成该操作有两种方式。第一种方式是把 JavaScript 直接应用于 ASP.NET 页面上的控件。例如,程序清单 4-7 中的 TextBox 服务器控件显示了当前的日期和时间。

程序清单 4-7 显示当前的日期和时间

```
protected void Page_Load(object sender, EventArgs e)
{
    TextBox1.Text = DateTime.Now.ToString();
}
```



本章只介绍了 JavaScript 的一点儿内容,本书第 VII 部分详细探讨了 ASP.NET 中的 JavaScript。

这几行代码在终端用户的页面上显示了当前的日期和时间。问题是所显示的日期和时间对于生成页面的 Web 服务器来说是正确的。如果用户位于美国西部标准时区(PST),而 Web 服务器位于东部时区(EST),页面对于访问者来说就是不正确的。如果希望时间对于浏览站点的任何人来说都是正确的,而无论他们在世界的哪个角落,可以使用 JavaScript 处理 TextBox 控件,如程序清单 4-8 所示。

程序清单 4-8 使用 JavaScript 给终端用户显示当前时间

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Using JavaScript</title>
</head>
<body onload="javascript:document.forms[0]['TextBox1'].value=Date();">
    <form id="form1" runat="server">
        <div>
            <asp:TextBox ID="TextBox1" runat="server" Width="300"></asp:TextBox>
        </div>
    </form>
</body>
</html>
```

在这个例子中,即使使用的是 Web 服务器控件系列中的标准 TextBox 服务器控件,也仍然能使用 JavaScript 在<body>元素的 onload 特性中访问这个控件。通过匿名函数使用服务器控件的 ID 属性值 TextBox1,使 onload 特性的值实际指向特定的服务器控件 TextBox1。采用这个方法还可以访问页面上的其他服务器控件。这几行代码生成了如图 4-6 所示的结果。

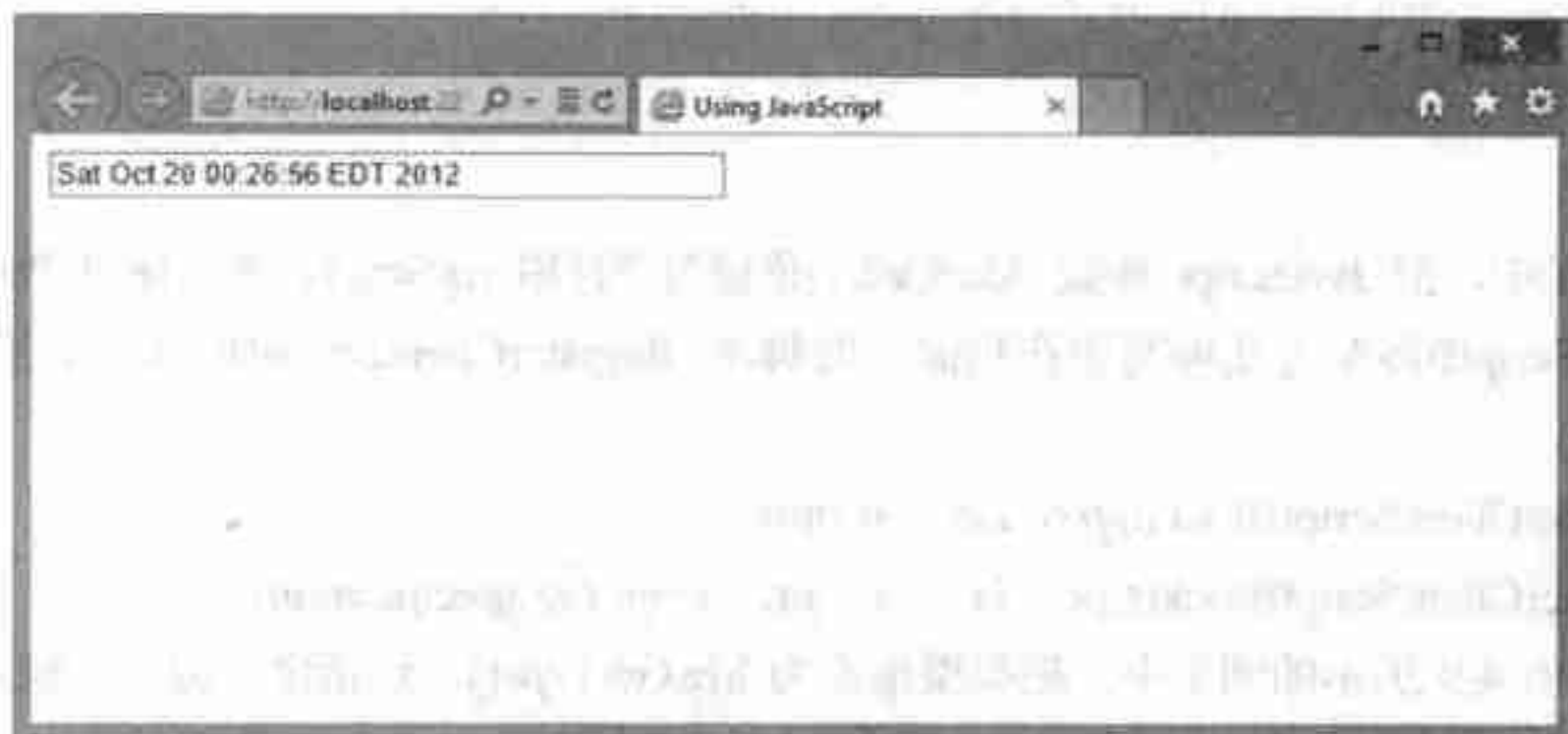


图 4-6

ASP.NET 使用 `Page.ClientScript` 属性在 ASP.NET 页面上注册和使用 JavaScript 函数。这里介绍其中的 3 个方法。更多的方法和属性可通过 `ClientScript` 对象(它引用了 `System.Web.UI.ClientScriptManager` 的一个实例)获得,但是这 3 个方法更加有用。其他的方法和属性可参阅在线文档。



.NET Framework 1.0/1.1 中的 `Page.RegisterStartupScript` 和 `Page.RegisterClientScriptBlock` 方法现在已被弃用。这两个用于注册脚本的方法均需要一组键/脚本参数。因为这两个方法比较复杂,所以很可能会发生键名冲突。`Page.ClientScript` 属性把所有的脚本注册都放在一起,从而使代码不容易出错。

4.6.1 使用 `Page.ClientScript.RegisterClientScriptBlock`

`RegisterClientScriptBlock` 方法可以把 JavaScript 函数放在页面的顶部。也就是说,该脚本用于在浏览器中启动页面。其用法如程序清单 4-9 所示。

程序清单 4-9 使用 `RegisterClientScriptBlock` 方法

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        string myScript = @"function AlertHello() { alert('Hello ASP.NET'); }";
        Page.ClientScript.RegisterClientScriptBlock(this.GetType(),
            "MyScript", myScript, true);
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Adding JavaScript</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Button ID="Button1" runat="server" Text="Button"
            OnClientClick="AlertHello()" />
    </div>
    </form>
</body>
</html>
```

```

    </div>
  </form>
</body>
</html>

```

在这个例子中, 把 JavaScript 函数 `AlertHello()` 创建为字符串 `myScript`。然后使用 `Page.ClientScript.RegisterClientScriptBlock` 方法编写放在页面上的脚本。`RegisterClientScriptBlock` 方法的两种构造方式如下:

- `RegisterClientScriptBlock(type, key, script)`
- `RegisterClientScriptBlock(type, key, script, script tag specification)`

在程序清单 4-9 所示的例子中, 把类型指定为 `Me.GetType()`, 还指定了键、要包含的脚本以及一个设置为 `True` 的 `Boolean` 值, 这样 .NET 就会自动把脚本放在 ASP.NET 页面的 `<script>` 标记中。在运行页面时, 可以查看页面的源代码以了解结果, 如下所示:

```

<!DOCTYPE html
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>
    Adding JavaScript
</title></head>
<body>
    <form method="post" action="Listing04-09.aspx" id="form1">
    <div class="aspNetHidden">
    <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="hbVKLJCpbKtClOpVO6
+dE5HkSYJIxqrLCZF0BMbiZhhdOepUjRfUukoZ7ZEq0BGQiZYBg8umPQQs/DvUX212QbHL3D4wLmJRxiVN
pCKLY=" />
    </div>
    <script type="text/javascript">
    //
    function AlertHello() { alert('Hello ASP.NET'); }//]]&gt;
    &lt;/script&gt;
    &lt;div class="aspNetHidden"&gt;
    &lt;input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION" value="Sbe6uw
GDqPTxBfbGDMsls8mWmBt0+sJwDh9YY6hS0PUZGqpOraDflMdrhkd3MgX5444UcMM5yv+j6yZi+XNuseFXe
9L9qv+l7YKNOLDHUJHJaibcCyLDb3qp0dOPcJkv" /&gt;
    &lt;/div&gt;
    &lt;div&gt;
    &lt;input type="submit" name="Button1" value="Button" onclick="AlertHello();"
    id="Button1" /&gt;
    &lt;/div&gt;
    &lt;/form&gt;
&lt;/body&gt;
&lt;/html&gt;
</pre>
</div>
<div data-bbox="81 778 910 816" data-label="Text">
<p>从上面的输出中可以看出, 指定的脚本的确包含在 ASP.NET 页面的相关页面代码之前。不仅包含 <code>&lt;script&gt;</code> 标记, 还在脚本外部添加了一些适当的注释标记(从而旧式浏览器不会崩溃)。</p>
</div>
<div data-bbox="81 830 574 847" data-label="Section-Header">
<h4>4.6.2 使用 <code>Page.ClientScript.RegisterStartupScript</code></h4>
</div>
<div data-bbox="81 860 910 918" data-label="Text">
<p><code>RegisterStartupScript</code> 方法与 <code>RegisterClientScriptBlock</code> 方法的区别不大。最大的区别在于 <code>RegisterStartupScript</code> 把脚本放在 ASP.NET 页面的底部而不是顶部。实际上, <code>RegisterStartupScript</code> 方法甚至使用与 <code>RegisterClientScriptBlock</code> 方法相同的构造方式:</p>
</div>
<div data-bbox="51 958 85 971" data-label="Page-Footer">118</div>
```


- RegisterStartupScript(type, key, script)
- RegisterStartupScript(type, key, script, script tag specification)

那么,这两个方法在页面上注册脚本的过程有什么区别?实际上区别很大!

如果在页面上有一些处理控件的 JavaScript 代码,那么在大多数情况下应该使用 RegisterStartupScript 方法,而不是 RegisterClientScriptBlock 方法。例如,应该使用下面的代码创建包含简单 <asp: TextBox> 控件的页面,该控件包含默认值 Hello ASP.NET:

```
<asp:TextBox ID="TextBox1" Runat="server">Hello ASP.NET</asp:TextBox>
```

然后使用 RegisterClientScriptBlock 方法把脚本放在页面上,其中页面利用了 TextBox1 控件中的值,如程序清单 4-10 所示。

程序清单 4-10 不正确地使用 RegisterClientScriptBlock 方法

```
protected void Page_Load(object sender, EventArgs e)
{
    string myScript = @"alert(document.getElementById('TextBox1').value);";
    Page.ClientScript.RegisterClientScriptBlock(this.GetType(),
        "MyScript", myScript, true);
}
```

运行这个页面会产生 JavaScript 错误,如图 4-7 所示(取决于所使用的 IE 版本)。

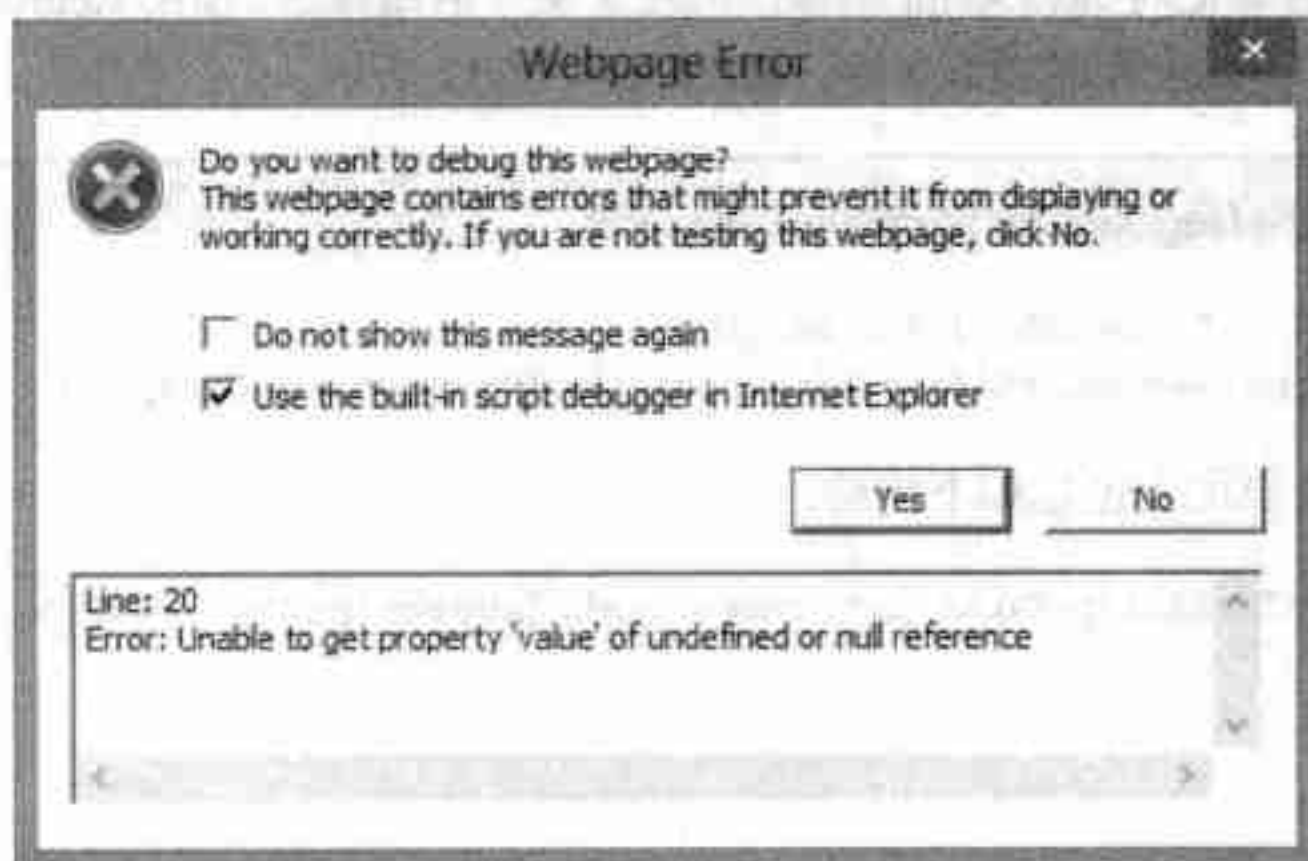


图 4-7

出错的原因是: JavaScript 函数在文本框放到屏幕之前就已触发。因此, JavaScript 函数找不到 TextBox1, 导致页面抛出错误。现在试用 RegisterStartupScript 方法, 如程序清单 4-11 所示。

程序清单 4-11 使用 RegisterStartupScript 方法

```
protected void Page_Load(object sender, EventArgs e)
{
    string myScript = @"alert(document.forms[0]['TextBox1'].value);";
    Page.ClientScript.RegisterStartupScript(this.GetType(),
        "MyScript", myScript, true);
}
```

这个方法把 JavaScript 函数放在 ASP.NET 页面的底部，因此在实际启动 JavaScript 时，该函数就会找到 TextBox1 元素，并按照预期的方式操作。结果如图 4-8 所示。



图 4-8

4.6.3 使用 Page.ClientScript.RegisterClientScriptInclude

最后一个方法是 RegisterClientScriptInclude。许多开发人员都把 JavaScript 放在.js 文件中，这是最好的方式，因为很容易把对 JavaScript 的修改应用于整个应用程序。使用 RegisterClientScriptInclude 方法可以在 ASP.NET 页面上注册脚本文件，如程序清单 4-12 所示。

程序清单 4-12 使用 RegisterClientScriptInclude 方法

```
string myScript = "myJavaScriptCode.js";  
Page.ClientScript.RegisterClientScriptInclude("myKey", myScript);
```

这会在 ASP.NET 页面上创建如下构造：

```
<script src="myJavaScriptCode.js" type="text/javascript"></script>
```

4.7 本章小结

本章介绍了 ASP.NET 页面的核心构建块——服务器控件。服务器控件是一种用于页面开发的面向对象方法，用于把页面元素封装到可修改和可扩展的组件中。最后，本章介绍了 HTML 服务器控件的使用，以及如何把 JavaScript 添加到页面中以修改控件的行为。

第 5 章

ASP.NET Web 服务器控件

本章要点

- 回顾关键的 Web 服务器控件
- 区分 Web 服务器控件的功能
- 从集合中删除项

在 HTML 服务器控件和 Web 服务器控件这两种服务器控件中，后者比较强大和灵活。第 4 章介绍了如何在应用程序中使用 HTML 服务器控件。HTML 服务器控件可以操作服务器端代码中的 HTML 元素。然而 Web 服务器控件比较强大，因为它们不是与特定的 HTML 元素明确关联，而是与某些要生成的功能紧密相关。从本章中可以看出，根据所使用的控件，Web 服务器控件可以非常简单，也可以非常复杂。

许多控件的作用是提高效率。这些控件提供了大量的高级功能，而这些功能在过去需要编写大量的程序，或者简单地忽略。例如，在以前使用传统的 ASP 时，很少在 Internet 网站上使用日历。而自从在 ASP.NET 1.0 中引入 Calendar 服务器控件以来，在站点上创建日历就是一项很简单的任务。在图像上建立图像映射是另一项在 ASP.NET 1.x 中难以完成的任务，而这个功能已作为一个新的服务器控件内置于 ASP.NET 2.0 中。随着 ASP.NET 版本的不断升级，新的控件和功能不断添加进来，从而使开发人员的 Web 开发更加高效。

本章将介绍一些可用的 Web 服务器控件。第一部分主要讨论 ASP.NET 早期版本中使用的 Web 服务器控件，接着介绍 ASP.NET 后面版本中新增的服务器控件。本章不讨论所有的服务器控件，因为一些服务器控件会在本书的其他章节介绍。

5.1 Web 服务器控件概述

Web 服务器控件是 ASP.NET 中最常用的组件。读者可能已经了解了第 4 章介绍的 HTML 服务器控件的许多潜在用途，但 Web 服务器控件在功能上更胜一筹。它们在使用时将呈现更高级的功能。

ASP.NET 提供的 HTML 服务器控件在工作时会映射为特定的 HTML 元素，通过处理 HTML 元

素提供的 HTML 属性来控制输出。这些属性可以在服务器端动态修改,之后输出到客户端。HTML 服务器控件的功能非常强大,一些功能是 Web 服务器控件所没有的。

Web 服务器控件的工作方式与此不同,它们不映射为特定的 HTML 元素,而是用于定义页面的功能、性能和外观,而且不需要通过一组 HTML 元素的属性来实现。在构造由 Web 服务器控件组成的 Web 页面时,可以描述页面元素的功能、外观和行为,然后让 ASP.NET 确定如何输出该页面。当然,其结果取决于发出请求的容器的功能。也就是说,每个请求人员都可能得到不同的 HTML 输出,因为每个请求人员都使用不同的浏览器类型或版本请求同一页面。ASP.NET 会检测浏览器的类型和版本,并完成相应的工作。

与 HTML 服务器控件不同,Web 服务器控件不仅可以处理常见的 Web 页面窗体元素(如文本框和按钮),还可以给 Web 页面增添一些高级功能。例如,许多 Web 应用程序的一个常见功能是带有日历。HTML 窗体元素不能把日历放在 Web 窗体上,但 ASP.NET 的 Web 服务器控件可以为应用程序提供功能全面的日历,包括一些高级功能。过去,在 Web 页面上添加日历不是一项简单的编程任务。而现在,使用 ASP.NET 添加日历是相当简单的,通过一行代码即可完成!

构造 Web 服务器控件就是在构造控件(即一组指令),只是该控件用于服务器(而不是客户端)。默认情况下,ASP.NET 提供的所有 Web 服务器控件都在控件声明的开头使用“asp:”。下面是一个典型的 Web 服务器控件:

```
<asp:Label ID="Label1" runat="server" Text="Hello World"></asp:Label>
```

与 HTML 服务器控件一样,Web 服务器控件也需要 ID 属性来引用服务器端代码中的控件,还需要 runat="server" 属性声明。与处理其他基于 XML 的元素一样,也需要使用 XML 语法规则正确打开和关闭 Web 服务器控件。在上面的例子中,<asp:Label>控件有一个关联的</asp:Label>结束元素。还可以使用下面的语法关闭该元素:

```
<asp:Label ID="Label1" runat="server" Text="Hello World" />
```

本章接下来将详细介绍 ASP.NET 中一些可用的 Web 服务器控件。

5.2 Label 服务器控件

Label 服务器控件用于在浏览器上显示文本。因为是服务器控件,所以可以在服务器端代码中动态地修改文本。在上面使用<asp:Label>控件的示例中,该控件使用 Text 属性来指定控件的内容,如下所示:

```
<asp:Label ID="Label1" runat="server" Text="Hello World" />
```

如果不使用 Text 属性,还可以把要显示的内容放在<asp:Label>元素之间,如下所示:

```
<asp:Label ID="Label1" runat="server">Hello World</asp:Label>
```

也可以通过编程方式提供控件的内容,如程序清单 5-1 所示。

程序清单 5-1 通过编程提供 Label 控件的文本

```
Label1.Text = "Hello ASP.NET!";
```

Label 服务器控件是一个仅显示文本的控件。但自从 ASP.NET 2.0 以来,它还有一个额外的功能。自这个架构版本发布以来,最大的变化是可以给窗体中的项添加热键功能(也称为快捷键)。这样,页面就可以激活某个声明为赋予指定热键的服务器控件(例如,使用 Alt+N 快捷键激活窗体上的第一个文本框)。



热键不一定能用于所有的浏览器和浏览器版本。

热键是终端用户在页面上执行操作的一种快捷方式。例如,如果使用的是 Internet Explorer,就可以按下 Ctrl+N 快捷键来打开 IE 的一个新实例。热键在胖客户端应用程序(Windows 窗体应用程序)中非常常见,而现在则可以在 ASP.NET 中使用它们。程序清单 5-2 是把热键功能应用于窗体中两个文本框的例子。

程序清单 5-2 使用 Label 服务器控件提供热键功能

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Label Server Control</title>
</head>
<body>
<form id="form1" runat="server">
<p>
<asp:Label ID="Label1" runat="server" AccessKey="N"
AssociatedControlID="Textbox1">User<u>n</u>ame</asp:Label>
<asp:TextBox ID="Textbox1" runat="server"></asp:TextBox>
</p>
<p>
<asp:Label ID="Label2" runat="server" AccessKey="P"
AssociatedControlID="Textbox2"><u>P</u>assword</asp:Label>
<asp:TextBox ID="Textbox2" runat="server"></asp:TextBox>
</p>
<p>
<asp:Button ID="Button1" runat="server" Text="Submit" />
</p>
</form>
</body>
</html>
```

使用 AccessKey 属性指定热键。在本例中,Label1 使用 N, Label2 使用 P。Label 控件的第二个属性是 AssociatedControlID,它的字符串值把 Label 控件与窗体中的另一个服务器控件关联起来。该值必须是窗体上的另一个服务器控件,否则,在调用页面时就会生成错误。

有了这两个控件后,在浏览器中调用页面时,就可以按下 Alt+N 或 Alt+P 快捷键,自动激活窗体上的特定文本框。在图 5-1 中,HTML 文本中带有下划线的字母与 Alt 键一起按下,就会激活与该文本相关的控件。这不是必需的操作,但强烈推荐,因为这是终端用户在使用热键时希望具有的操作方式。在这个例子中,对 Username 中的字母 n 和 Password 中的字母 P 添加了下划线。

在使用热键时,注意并不是所有的字母都可以与 Alt 键一起使用。Internet Explorer 已经使用了

字母 F、E、V、I、O、T、A、W 和 H。如果使用这些字母，IE 操作就会替代我们在页面上进行的操作。

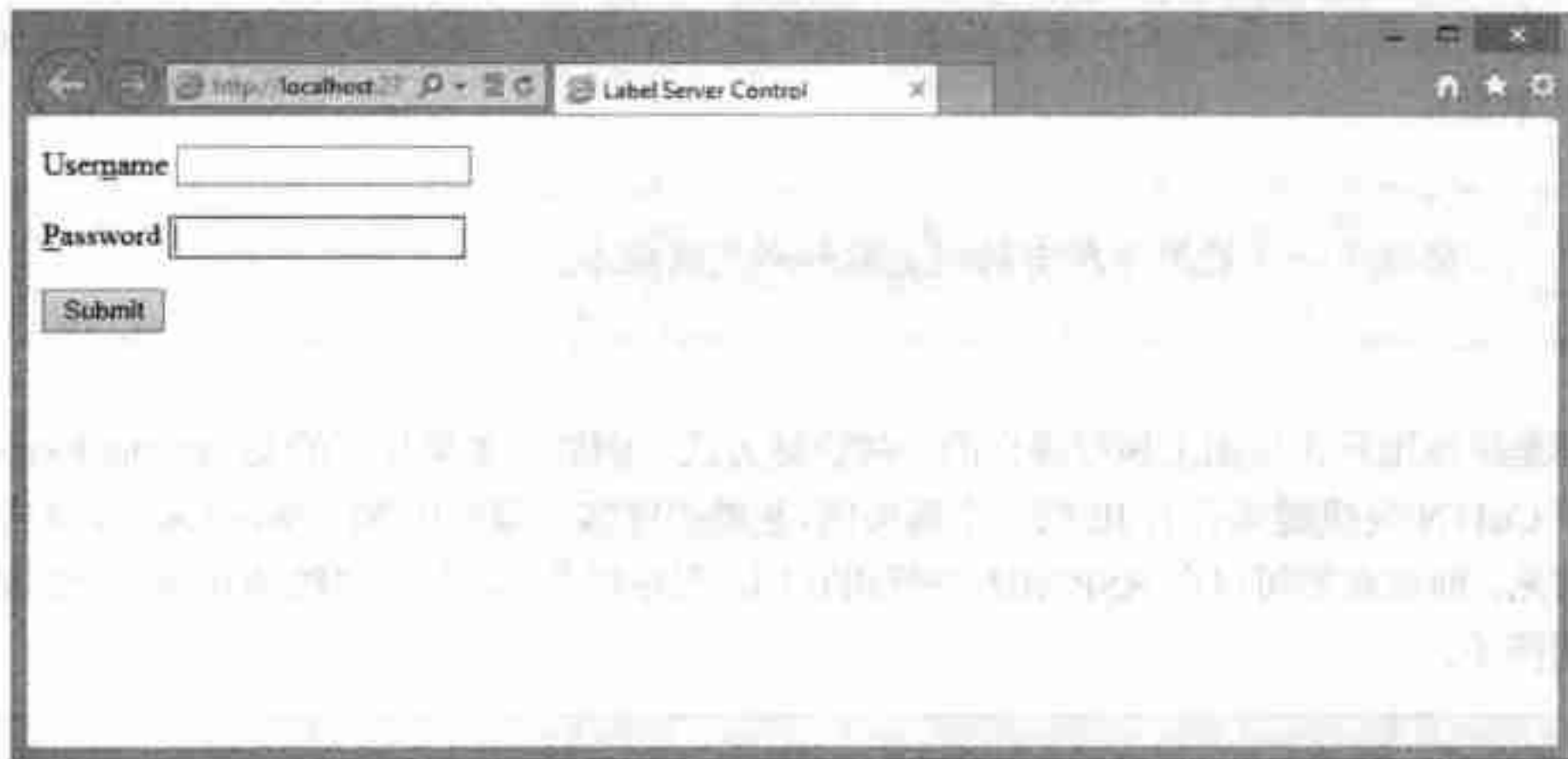


图 5-1

5.3 Literal 服务器控件

Literal 服务器控件的工作方式非常类似于 Label 服务器控件。这个控件过去总是用于在浏览器上显示在整个过程中不发生变化的文本(字面状态)。Label 控件通过在文本的外部加上 `` 元素来改变输出，如下所示：

```
<span id="Label1">Here is some text</span>
```

Literal 控件只输出文本，不输出 `` 元素。该服务器控件有一个属性 `Mode`，该属性可以指定 ASP.NET 引擎如何解释赋予控件的文本。

如果把一些 HTML 代码放在输出的字符串中(如 `Here is some text`)，Literal 控件就输出这些 HTML 代码，所用的浏览器会把文本显示为黑体：

Here is some text

试着使用 `Mode` 属性，如下所示：

```
<asp:Literal ID="Literal1" runat="server" Mode="Encode"
Text="<b>Here is some text</b>"></asp:Literal>
```

添加 `Mode="Encode"`，ASP.NET 将对输出进行编码，再由应用程序接收该输出：

```
&lt;b&gt;Label&lt;/b&gt;
```

现在不是把文本转换为黑体，而是显示 `` 元素：

```
<b>Here is some text</b>
```

如果希望在应用程序中显示代码，这就是很理想的方式。`Mode` 属性的其他值有 `Transform` 和 `PassThrough`。`Transform` 会考虑到用户，根据需要包含或删除元素。例如，并不是所有的设备都接受

HTML 元素,因此如果将 `Mode` 属性的值设置为 `Transform`,在把字符串发送给应用程序之前,就会从字符串中删除这些元素。`Mode` 属性的 `PassThrough` 值表示,在将文本发送到应用程序时,不对字符串进行任何修改。

5.4 TextBox 服务器控件

Web 页面的一项主要功能是提供窗体,让终端用户使用它们提交信息。`TextBox` 服务器控件就是用于这种场合的最常用控件之一。顾名思义,该控件在窗体上提供文本框,让终端用户输入文本。可以把 `TextBox` 控件映射为窗体上使用的 3 个不同的 HTML 元素。

首先,`TextBox` 控件可以用作标准的 HTML 文本框,如下面的代码所示:

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
```

这行代码在窗体上创建了一个文本框,如图 5-2 所示。

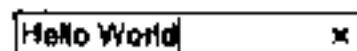


图 5-2

其次,`TextBox` 控件允许终端用户把密码输入到窗体上。这只需把 `TextBox` 控件的 `TextMode` 属性改为 `Password` 即可,如下所示:

```
<asp:TextBox ID="TextBox1" runat="server" TextMode="Password"></asp:TextBox>
```

要求终端用户通过浏览器输入密码时,最好提供一个文本框,并对这个窗体元素中的内容进行编码。使用 `TextMode="Password"` 属性值,就可以确保文本以星号(*)或圆点显示,如图 5-3 所示。



图 5-3

最后,`TextBox` 服务器控件可以用作多行文本框。完成这项任务的代码如下所示:

```
<asp:TextBox ID="TextBox1" runat="server" TextMode="MultiLine"
Width="300px" Height="150px"></asp:TextBox>
```

给 `TextMode` 属性赋予 `MultiLine` 值,就会创建一个多行文本框,终端用户可以在该窗体元素中输入大量的文本。`Width` 和 `Height` 属性设置了文本区域的尺寸,但它们是可选的属性,如果没有它们,文本区域就使用最小尺寸。图 5-4 显示了为上述代码添加一些文本后的结果。

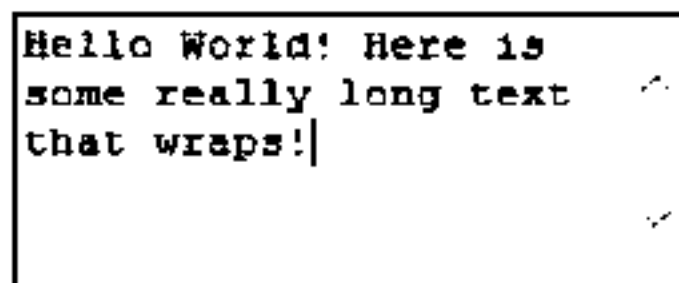


图 5-4

在使用多行文本框时,要注意 `Wrap` 属性。该属性设置为 `True`(默认值)时,输入到文本区域中的文本会根据需要自动换行。如果设置为 `False`,终端用户就可以一直在一行中输入文本,按下 `Enter` 键后,光标才会跳到下一行。

5.4.1 使用 `Focus()` 方法

`TextBox` 服务器控件派生自 `WebControl` 基类,因此可以使用该基类的 `Focus()` 方法。`Focus()` 方法可以把终端用户的光标动态地放置在某个指定的窗体元素上(不仅仅是 `TextBox` 控件,还可以是派生自 `WebControl` 类的任何服务器控件)。因此,它是 `TextBox` 控件最常用的一个方法,如程序清单 5-3 所示。

程序清单 5-3 使用 TextBox 控件的 Focus()方法

```
protected void Page_Load(Object sender, EventArgs e)
{
    FirstNameTextBox.Focus();
}
```

当使用了这个方法的页面加载到浏览器中时,光标已经位于文本框中,准备接受用户的输入。此时不需要移动鼠标以将光标放在正确的位置上,就可以开始在窗体上输入信息。这是使用键盘控制窗体的理想方式。

5.4.2 使用 AutoPostBack

ASP.NET 页面以事件驱动的方式工作。当 Web 页面上的一项操作触发了事件时,就执行服务器端代码。一个较常见的事件是终端用户单击了窗体上的一个按钮。如果在 Visual Studio 2012 的设计视图中双击该按钮,就可以看到代码页面上包含了 Button1_Click 事件的结构。这是因为 OnClick 是 Button 控件最常用的事件。双击 TextBox 控件会构造 OnTextChanged 事件,该事件在终端用户把光标移出文本框时触发。要退出文本框,可以在文本框中输入一些内容后单击页面上的另一个元素,也可以按 Tab 键。这个事件的使用如程序清单 5-4 所示。

程序清单 5-4 改变文本框时触发一个事件

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void TextBox1_TextChanged(object sender, EventArgs e)
    {
        Response.Write("OnTextChanged event triggered");
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        Response.Write("OnClick event triggered");
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Triggering an event when a TextBox</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:TextBox ID="TextBox1" runat="server" AutoPostBack="True"
                OnTextChanged="TextBox1_TextChanged"></asp:TextBox>
            <asp:Button ID="Button1" runat="server" Text="Button"
                OnClick="Button1_Click" />
        </div>
    </form>
</body>
</html>
```

注意，在编译并运行这个页面时，可以在文本框中输入一些内容。但使用 Tab 键退出文本框之后，就会触发 OnTextChanged 事件，并运行 TextBox1_TextChanged 事件包含的代码。为此，必须给 TextBox 控件添加 AutoPostBack 属性，并把它设置为 True。这样 Web 页面才会在回送之前查找已改变的文本。为了使 AutoPostBack 属性能工作，用于查看页面的浏览器必须支持 ECMAScript。

5.4.3 使用 AutoCompleteType

我们希望为 Web 应用程序建立的窗体使用起来尽可能简单，使用这些窗体的终端用户能够方便快捷地填充信息。如果窗体的构建非常耗时，访问站点的用户就会很快退出。

Web 窗体的一项主要功能是智能、自动地完成窗体。在第一次访问站点时就会看到这个功能。在开始给窗体填充信息时，在文本框的下面会显示一个下拉列表，其中显示了在前一个窗体中输入的值。当前处理的纯文本框就变成了智能文本框，如图 5-5 所示。

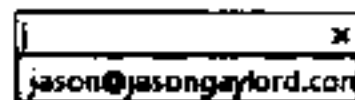


图 5-5

TextBox 控件提供的 AutoCompleteType 属性允许对窗体应用自动完成功能。但是，用户必须帮助窗体上的文本框识别出它们需要的信息类型。怎么做呢？首先查看 AutoCompleteType 属性的值：

Disabled	Company	HomeFax
BusinessCity	DisplayName	Homepage
BusinessCountryRegion	HomeState	HomePhone
BusinessFax	HomeStreetAddress	JobTitle
BusinessPhone	HomeZipCode	LastName
BusinessState	Email	MiddleName
BusinessStreetAddress	Enabled	Notes
BusinessUrl	FirstName	Office
BusinessZipCode	Gender	Pager
Cellular	HomeCity	Search
Department	HomeCountryRegion	

从这个列表中可以看出，如果文本框请求终端用户输入家庭地址的街道部分，就要在 TextBox 控件中使用如下代码：

```
<asp:TextBox ID="TextBox1" runat="server"
    AutoCompleteType="HomeStreetAddress"></asp:TextBox>
```

查看所创建的文本框的源代码，会发现如下构造代码：

```
<input name="TextBox1" type="text" vcard_name="vCard.Home.StreetAddress"
    id="TextBox1" />
```

这个功能使窗体更容易处理。这是很简单的操作，但有时可以为 Web 站点赢得许多回头客。

5.5 Button 服务器控件

Web 窗体的另一个常见控件是按钮，可以使用 Button 服务器控件构造按钮。按钮是提交窗体的

常用元素。在大多数情况下，我们都是通过 Button 控件的 OnClick 事件来处理包含在窗体中的各项，如程序清单 5-5 所示。

程序清单 5-5 Button 控件的 OnClick 事件

```
protected void Button1_Click(object sender, EventArgs e)
{
    // Code here
}
```

Button 控件是较容易使用的控件之一，但有两个属性必须掌握：CausesValidation 和 CommandName。下面几节将讨论它们。

5.5.1 CausesValidation 属性

如果 Web 页面上有多个按钮，而且当前处理的是验证服务器控件，就不能对窗体上的每个按钮进行验证。把 CausesValidation 属性设置为 False，就是使用不引发验证过程的按钮的一种方式，该方式详见第 6 章。

5.5.2 CommandName 属性

可以让窗体上的多个按钮处理单个事件。我们可以标记按钮，使代码能根据窗体上单击的按钮进行逻辑判断。必须按照程序清单 5-6 给出的方式构建 Button 控件，这样才能利用多个按钮处理单个事件。

程序清单 5-6 构建多个 Button 控件以使用单个函数

```
<asp:Button ID="Button1" runat="server" OnCommand="Button_Command"
    CommandName="DoSomething1" Text="Button 1" />
<asp:Button ID="Button2" runat="server" OnCommand="Button_Command"
    CommandName="DoSomething2" Text="Button 2" />
```

下面查看 Button 控件的这两个实例，需要注意几个地方。首先要注意的是哪个元素不存在，即具有 OnClick 事件的特性。相反，这里使用的是 OnCommand 事件，它指向 Button_Command 事件。这两个 Button 控件处理同一个事件。事件应该如何确定单击哪个按钮呢？可以通过 CommandName 属性的值来确定这一点。在这个例子中，它们拥有不同的值——DoSomething1 和 DoSomething2。

下一步是创建这两个按钮要处理的 Button_Command 事件，方法是输入事件，或者在 Visual Studio 的源代码视图中，从 Button 控件的可用事件下拉列表中选择 Command 事件。无论采用什么方法，都会得到程序清单 5-7 所示的事件。

程序清单 5-7 Button_Command 事件

```
protected void Button_Command(object sender,
    System.Web.UI.WebControls.CommandEventArgs e)
{
    switch (e.CommandName)
    {
        case ("DoSomething1"):
            Response.Write("Button 1 was selected");
    }
```

```

        break;
    case ("DoSomething2"):
        Response.Write("Button 2 was selected");
        break;
    }
}

```

注意,这种方法使用的是 `System.Web.UI.WebControls.CommandEventArgs`,而不是常用的 `System.EventArgs`。以便把 `Select Case(switch)`语句中使用的成员 `CommandName` 作为 `e.CommandName` 来访问。使用这个对象,可以检查窗体上被单击按钮使用的 `CommandName` 属性值,并根据该值执行特定的操作。

除了 `CommandName` 属性中定义的内容之外,还可以在 `Command` 事件中添加要传入的参数。为此,可以使用 `Button` 控件的 `CommandArgument` 属性。给属性添加值可以更精细地定义需要的项。在服务器端代码中使用 `CommandEventArgs` 对象的 `e.CommandArgument`,就可以获得这个值。

5.5.3 使用客户端 JavaScript 的按钮

按钮常常用于提交信息,并引发 Web 页面上的操作。在 ASP.NET 1.0/1.1 推出以前,开发人员要在页面中混合许多 JavaScript 才能实现在单击按钮时触发 JavaScript 事件。这个过程在 ASP.NET 1.0/1.1 中比较繁琐,但在 ASP.NET 2.0 以后的版本中就容易很多。

创建一个页面,其中包含一个 JavaScript 事件和一个服务器端事件,它们在单击按钮时触发,如程序清单 5-8 所示。

程序清单 5-8 两种类型的按钮事件

```

<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Button1_Click(Object sender, EventArgs e)
    {
        Response.Write("Postback!");
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Button Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Button ID="Button1" runat="server" Text="Button"
            OnClientClick="AlertHello()" OnClick="Button1_Click" />
    </form>
    <script type="text/javascript">
        function AlertHello() {
            alert('Hello ASP.NET');
        }
    </script>
</body>
</html>

```

首先要注意 Button 服务器控件的属性 OnClientClick。它指向客户端函数,这一点与指向服务器端事件的 OnClick 属性不同。这个例子使用了 JavaScript 函数 AlertHello()。

Visual Studio 2012 的一个优点是,它可以同时处理服务器端脚本标记与客户端脚本标记,它们能无缝地工作。在这个例子中,打开 JavaScript 警告对话框(如图 5-6 所示),终端用户单击 OK 按钮之后,页面将在触发服务器端事件时回送。



图 5-6

Button 服务器控件的另一个有趣属性是 PostBackUrl,它可以进行跨页面传送,而不是把 ASP.NET 页面的内容传送回原来的页面,如下面的例子所示:

```
<asp:Button ID="Button1" runat="server" Text="Submit to Another Page"
    PostBackUrl="Page2.aspx" />
```

有关跨页面传送的内容详见第 3 章。

5.6 LinkButton 服务器控件

LinkButton 服务器控件是 Button 服务器控件的一个变体,它基本上与 Button 控件相同,但 LinkButton 控件采用的是超链接的形式,而且这不是一般的超链接。终端用户单击该链接时,它的行为与按钮类似。

LinkButton 服务器控件的构造代码如下所示:

```
<asp:LinkButton ID="LinkButton1" runat="server" OnClick="LinkButton1_Click">
    Submit your name to our database
</asp:LinkButton>
```

使用 LinkButton 控件的结果如图 5-7 所示。

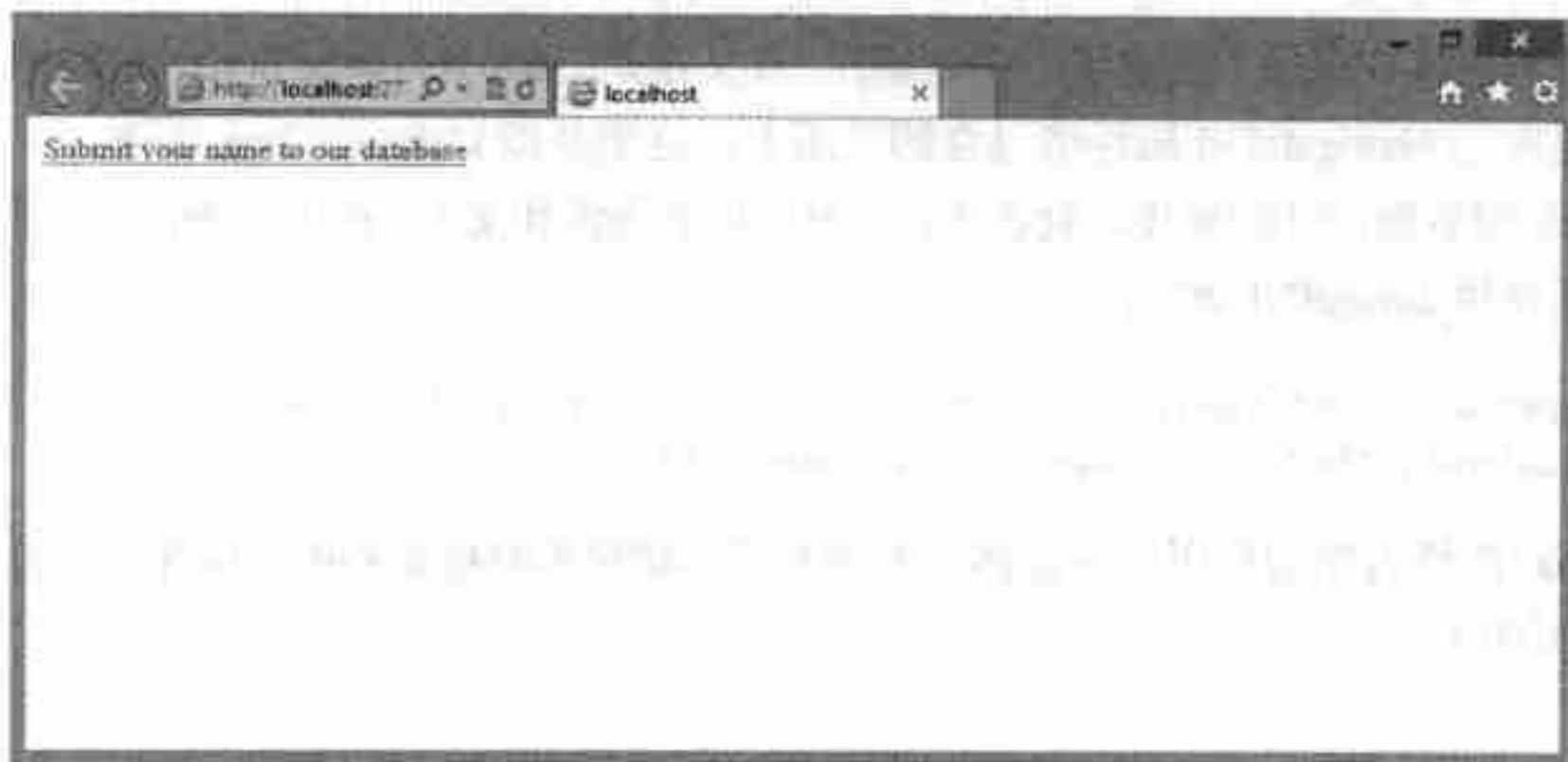


图 5-7

5.7 ImageButton 服务器控件

ImageButton 控件也是 Button 控件的一个变体，它几乎与 Button 控件完全相同，但是可以使用定制图像作为窗体的按钮，而不是使用大多数窗体上的常见按钮。也就是说，可以把自己的按钮创建为图像，终端用户可以单击该图像来提交窗体数据。ImageButton 控件的构造代码如下所示：

```
<asp:ImageButton ID="ImageButton1" runat="server"
    OnClick="ImageButton1_Click" ImageUrl="search.jpg" />
```

ImageButton 控件使用 ImageUrl 属性指定所用图像的位置。在这个例子中，ImageUrl 指向 search.jpg。ImageButton 控件与 LinkButton 或 Button 控件的最大区别是，ImageButton 控件的 OnClick 事件有不同的构造，如程序清单 5-9 所示。

程序清单 5-9 ImageButton 控件的 Click 事件

```
protected void ImageButton1_Click(object sender, ImageClickEventArgs e)
{
    // Code here
}
```

该构造代码使用了 ImageClickEventArgs 对象，而不是 LinkButton 和 Button 控件通常使用的 System.EventArgs 对象。可以使用这个对象的 e.X 和 e.Y 坐标确定终端用户单击了图像中的什么位置。

5.8 HyperLink 服务器控件

HyperLink 服务器控件可以通过编程方式处理 Web 页面上的任意超链接。超链接允许终端用户从一个页面移动到另一个页面。可以使用该控件的 Text 属性设置超链接的文本：

```
<asp:HyperLink ID="HyperLink1" runat="server" Text="Go to this page here"
    NavigateUrl="Default2.aspx"></asp:HyperLink>
```

这个服务器控件在页面上创建了一个超链接,其文本是 `Go to this page here`。在单击这个链接时,用户会被重定向到 `NavigateUrl` 属性值包含的页面上,这里是指 `Default2.aspx` 页面。

`HyperLink` 服务器控件的有趣之处在于,它可以用于图像和文本。在用于图像时,不应使用 `Text` 属性,而需要使用 `ImageUrl` 属性:

```
<asp:HyperLink ID="HyperLink2" runat="server" ImageUrl="MyLinkImage.gif"
    NavigateUrl="Default2.aspx"></asp:HyperLink>
```

`HyperLink` 控件可以根据用户在窗体上的输入或加载页面时检索到的数据库值,把超链接动态地放在 Web 页面上。

5.9 DropDownList 服务器控件

`DropDownList` 服务器控件可以把 HTML 选择框放在 Web 页面上,并对它进行编程操作。如果集合中有许多项,希望终端用户从中选择一项,那么使用 `DropDownList` 服务器控件就是很理想的方式。该控件通常用于中型或大型集合。如果集合比较小,应考虑使用 `RadioButtonList` 服务器控件(详见本章后面的内容)。

`DropDownList` 控件生成的选择框会显示一项,允许终端用户从较大的项列表中选择一项。根据选择框中的选项数,终端用户可能需要在一系列选项中滚动。注意下拉列表中的滚动栏是浏览器根据其版本和列表中包含的项数自动创建的。

下面是 `DropDownList` 控件的代码:

```
<asp:DropDownList ID="DropDownList1" runat="server">
    <asp:ListItem>Select an Item</asp:ListItem>
    <asp:ListItem>Car</asp:ListItem>
    <asp:ListItem>Airplane</asp:ListItem>
    <asp:ListItem>Train</asp:ListItem>
</asp:DropDownList>
```

这段代码在浏览器上生成了一个下拉列表,如图 5-8 所示。

很容易把 `DropDownList` 控件绑定到各种数据存储。数据存储可以是数组、数据库值、XML 文件值或其他值。程序清单 5-10 是绑定 `DropDownList` 控件的一个例子,它从 3 个可用数组中选取一个,并动态生成一个 `DropDownList` 控件。

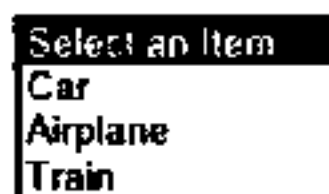


图 5-8

程序清单 5-10 从数组中动态生成 `DropDownList` 控件

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
    {
        string[] carArray = new[] { "Ford", "Honda", "BMW", "Dodge" };
        string[] airplaneArray = new[] { "Boeing 777", "Boeing 747",
            "Boeing 737" };
        string[] trainArray = new[] { "Bullet Train", "Amtrack", "Tram" };
    }
}
```

```

        if(DropDownList1.SelectedValue == "Car")
        {
            DropDownList2.DataSource = carArray;
        }
        else if(DropDownList1.SelectedValue == "Airplane")
        {
            DropDownList2.DataSource = airplaneArray;
        }
        else if(DropDownList1.SelectedValue == "Train")
        {
            DropDownList2.DataSource = trainArray;
        }
        DropDownList2.DataBind();
        DropDownList2.Visible = DropDownList1.SelectedValue != "Select an Item";
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        Response.Write("You selected <b>" +
            DropDownList1.SelectedValue.ToString() + ": " +
            DropDownList2.SelectedValue.ToString() + "</b>");
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>DropDownList Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            Select transportation type:<br />
            <asp:DropDownList ID="DropDownList1" runat="server"
                OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged"
                AutoPostBack="true">
                <asp:ListItem>Select an Item</asp:ListItem>
                <asp:ListItem>Car</asp:ListItem>
                <asp:ListItem>Airplane</asp:ListItem>
                <asp:ListItem>Train</asp:ListItem>
            </asp:DropDownList>&nbsp;
            <asp:DropDownList ID="DropDownList2" runat="server" Visible="false">
            </asp:DropDownList>
            <asp:Button ID="Button1" runat="server" Text="Select Options"
                OnClick="Button1_Click" />
        </div>
    </form>
</body>
</html>

```

在这个例子中，第二个下拉列表是根据在第一个下拉列表中选择的值动态生成的。例如，从第一个下拉列表中选择 Car，就会在窗体上动态创建第二个下拉列表，其中包含一组可用的汽车选项。

这是允许的操作，因为使用了 DropDownList 控件的 AutoPostBack 属性。AutoPostBack 属性设置为 True 时，如果做出了选择，就触发通过 OnSelectedIndexChanged 事件提供的方法。在该例中，触发了 DropDownList1_SelectedIndexChanged 事件，动态创建了第二个下拉列表。

在这个方法中，在一个字符串数组中创建了第二个下拉列表的内容，然后通过 DataBind() 方法

和 DataSource 属性绑定到第二个 DropDownList 控件上。

编译并运行该页面，结果如图 5-9 所示。

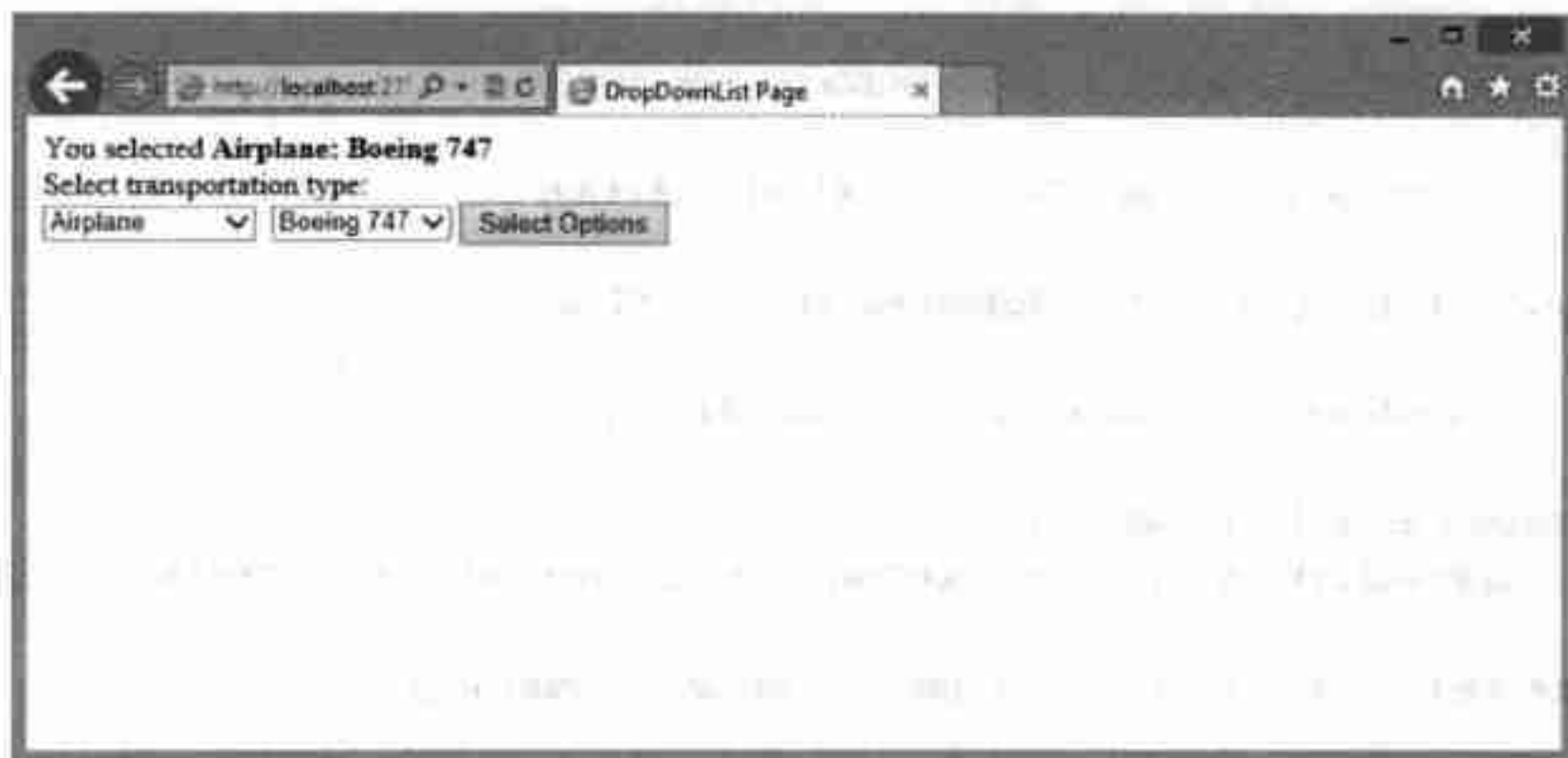


图 5-9

5.10 从集合中可视化地删除数据项

DropDownList、ListBox、CheckBoxList 和 RadioButtonList 服务器控件可以从控件的集合中可视化地删除数据项，而且可以操作未显示在服务器端代码中的各项。



ListBox、CheckBoxList 和 RadioButtonList 控件将在稍后讨论。

在下面的删除数据项例子中，要创建一个包含 3 项的下拉列表，其中有一项是不显示出来的。但是在回送时，仍然可以操作 ListItem 的 Value 或 Text 属性，如程序清单 5-11 所示。

程序清单 5-11 禁用集合中的某些 ListItem

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
    {
        Response.Write("You selected item number " +
            DropDownList1.SelectedValue + "<br>");
        Response.Write("You didn't select item number " +
            DropDownList1.Items[1].Value);
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>DropDownList Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
```

```

<div>
    <asp:DropDownList ID="DropDownList1" runat="server"
        AutoPostBack="True"
        OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
        <asp:ListItem Value="1">First Choice</asp:ListItem>
        <asp:ListItem Value="2" Enabled="False">Second
            Choice</asp:ListItem>
        <asp:ListItem Value="3">Third Choice</asp:ListItem>
    </asp:DropDownList>
</div>
</form>
</body>
</html>

```

从代码中可以看出, `<asp:ListItem>` 元素有个属性 `Enabled`, 这个属性的布尔值决定了是否显示集合中的项。如果指定 `Enabled="False"`, 就不显示这一项, 但仍然可以在 `DropDownList1_SelectedIndexChanged` 事件的服务器端代码中操作该项。这些 `Response.Write` 语句的输出结果如图 5-10 所示。

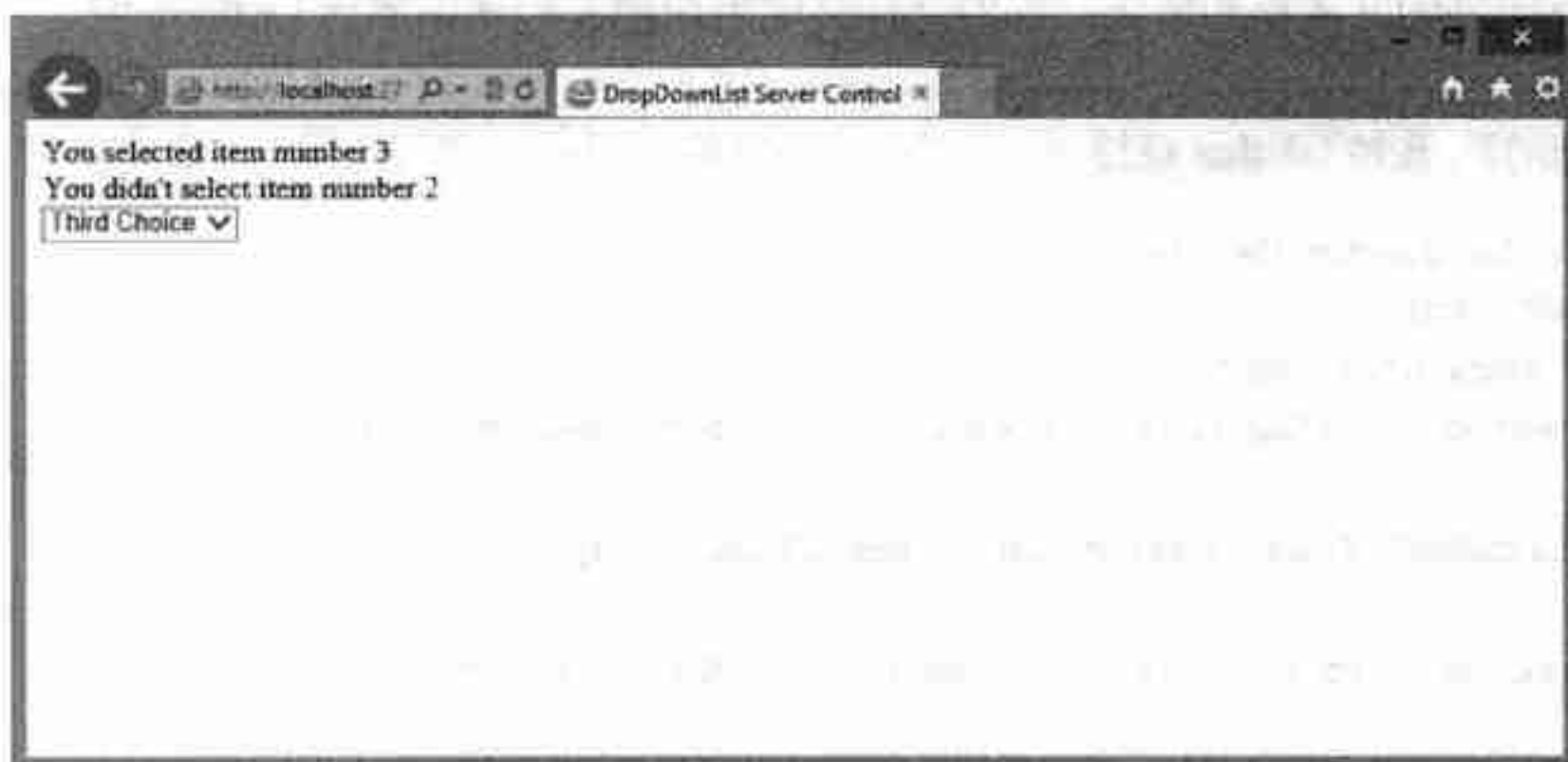


图 5-10

5.11 ListBox 服务器控件

`ListBox` 服务器控件的功能类似于 `DropDownList` 控件, 它也显示一个数据项集合。但是, `ListBox` 控件的操作不同于 `DropDownList` 控件, 它可以为终端用户显示集合中的更多内容, 并允许终端用户在集合中选择多项, 而 `DropDownList` 控件不可能做到这一点。

`ListBox` 控件的代码如下所示:

```

<asp:ListBox ID="ListBox1" runat="server">
    <asp:ListItem>ASP.NET 4.5</asp:ListItem>
    <asp:ListItem>ASP.NET MVC 4</asp:ListItem>
    <asp:ListItem>jQuery 1.8.x</asp:ListItem>
    <asp:ListItem>Visual Studio 2012</asp:ListItem>
</asp:ListBox>

```

```

ASP.NET 4.5
ASP.NET MVC 4
jQuery 1.8.x
Visual Studio 2012

```

图 5-11

在浏览器上生成的结果如图 5-11 所示。

5.11.1 允许用户选择多项

可以使用 `SelectionMode` 属性让终端用户从 `ListBox` 控件显示的内容中选择多项。下面是一个例子：

```
<asp:ListBox ID="ListBox1" runat="server" SelectionMode="Multiple">
  <asp:ListItem>ASP.NET 4.5</asp:ListItem>
  <asp:ListItem>ASP.NET MVC 4</asp:ListItem>
  <asp:ListItem>jQuery 1.8.x</asp:ListItem>
  <asp:ListItem>Visual Studio 2012</asp:ListItem>
</asp:ListBox>
```

`SelectionMode` 属性的值可以是 `Single` 或 `Multiple`。把该属性的值设置为 `Multiple`，将允许终端用户在列表框中选择多项。用户在选择时必须按住 `Ctrl` 或 `Shift` 键。按住 `Ctrl` 键可以从列表中选择一项，并且原来的项仍被选中。按住 `Shift` 键可以选择某个范围内的多项。

5.11.2 使用 `ListBox` 控件的例子

程序清单 5-12 中的 `ListBox` 控件允许在用户单击 `Submit` 按钮时，在浏览器上显示多个选项。窗体的顶部还有额外的文本框和按钮，它们允许终端用户把其他项添加到 `ListBox` 中。

程序清单 5-12 使用 `ListBox` 控件

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
  protected void Button1_Click(object sender, EventArgs e)
  {
    ListBox1.Items.Add(TextBox1.Text.ToString());
  }
  protected void Button2_Click(object sender, EventArgs e)
  {
    Label1.Text = "You selected from the ListBox:<br>";
    foreach(ListItem li in ListBox1.Items)
    {
      if(li.Selected)
      {
        Label1.Text += li.Text + "<br>";
      }
    }
  }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Using the ListBox</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
      <asp:Button ID="Button1" runat="server" Text="Add an additional item"
        OnClick="Button1_Click" /><br />
```



```

<br />
<asp:ListBox ID="ListBox1" runat="server" SelectionMode="Multiple">
    <asp:ListItem>ASP.NET 4.5</asp:ListItem>
    <asp:ListItem>ASP.NET MVC 4</asp:ListItem>
    <asp:ListItem>jQuery 1.8.x</asp:ListItem>
    <asp:ListItem>Visual Studio 2012</asp:ListItem>
</asp:ListBox><br />
<br />
<asp:Button ID="Button2" runat="server" Text="Submit"
    OnClick="Button2_Click" /><br />
<br />
<asp:Label ID="Label1" runat="server"></asp:Label>
</div>
</form>
</body>
</html>

```

这是个很有趣的例子。首先，在 `ListBox` 控件中已经放置了一些默认项(ASP.NET Web 开发中的 4 个常用工具)。但是，窗体顶部的文本框和按钮允许终端用户给列表添加更多的工具。用户可以从 `ListBox` 中选择一项或多项，包括动态添加到集合中的项。当用户做出选择并单击按钮后，`Button2_Click` 事件就遍历集合中的 `ListItem` 实例，只显示已选中的项。

这个控件会创建一个 `ListItem` 对象，使用其 `Selected` 属性确定集合中的某项是否被选中。`ListItem` 对象不仅可用于 `ListBox` 控件(尽管以前只能用于 `ListBox` 控件)，还能用于其他控件。可以在集合中动态添加或删除项，使用 `DropDownList`、`CheckBoxList` 和 `RadioButtonList` 中的 `ListItem` 对象获取项及其值。它是每个列表控件都可以使用的对象。编译并运行页面，结果如图 5-12 所示。

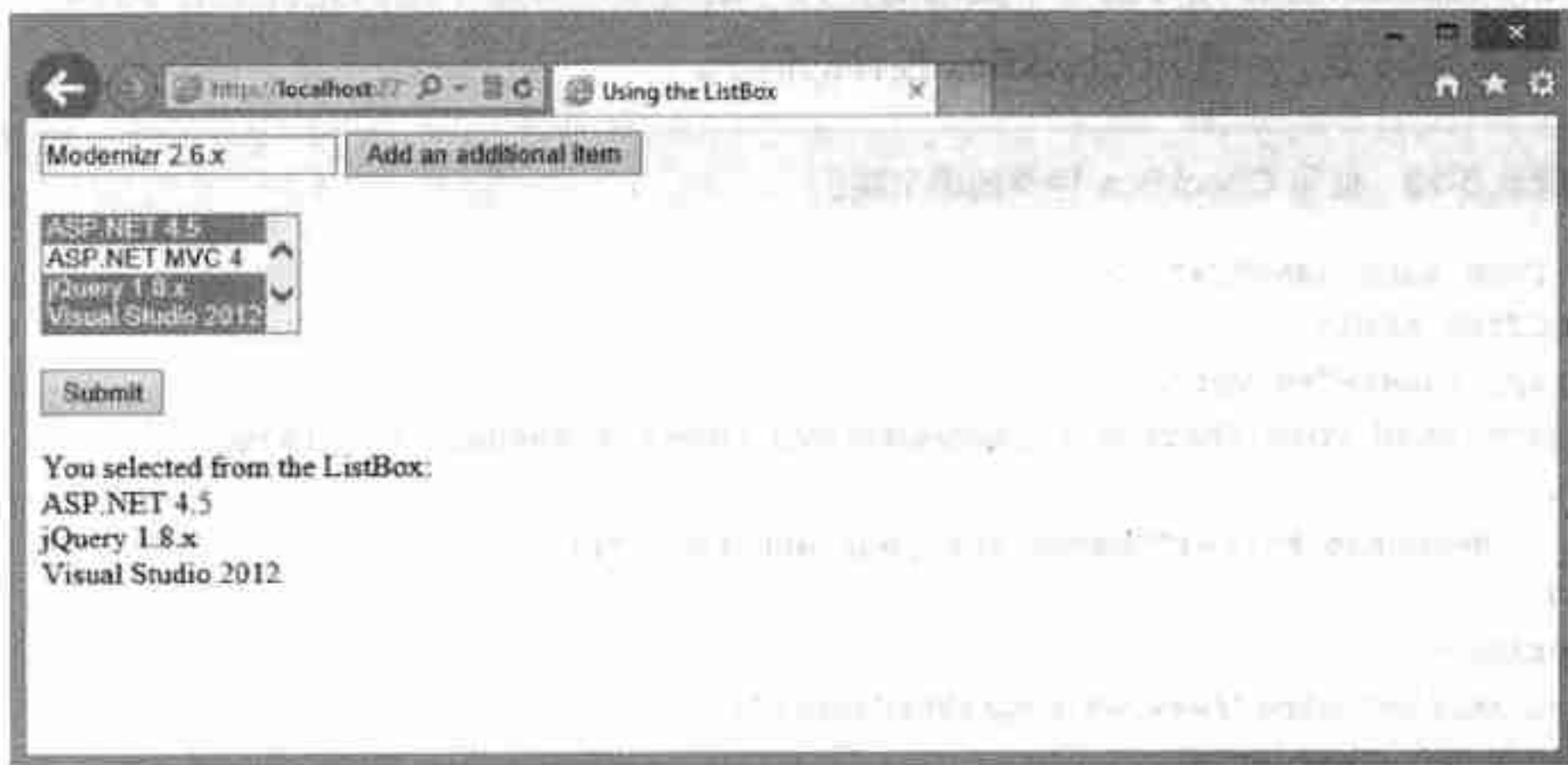


图 5-12

5.11.3 给集合添加项

要给集合添加项，可以使用下面的语法：

```
ListBox1.Items.Add(TextBox1.Text)
```

在浏览器中创建的源代码包含如下动态生成的代码：

```
<select size="4" name="ListBox1" multiple="multiple" id="ListBox1">
  <option value="ASP.NET 4.5">ASP.NET 4.5</option>
  <option value="ASP.NET MVC 4">ASP.NET MVC 4</option>
  <option value="jQuery 1.8.x">jQuery 1.8.x</option>
  <option value="Visual Studio 2012">Visual Studio 2012</option>
  <option value="Modernizr 2.6.x">Modernizr 2.6.x</option>
</select>
```

可以看出，动态添加的值是文本项，还可以看到项值。另外，也可以添加 `Listitem` 对象以获取不同的项名和项值：

```
ListBox1.Items.Add(new ListItem("Modernizr 2.6.x", "Modernizr"));
```

这个例子添加了一个新的 `Listitem` 对象——不仅添加了项名，还添加了项值，在浏览器上生成的结果如下：

```
<option value="Modernizr">Modernizr 2.6.x</option>
```

5.12 CheckBox 服务器控件

Web 窗体上的复选框允许用户选择项集合中的项，或者把某项的值指定为 `yes/no`、`on/off` 或 `true/false`。使用 `CheckBox` 或 `CheckBoxList` 控件可以在 Web 窗体上包含复选框。

`CheckBox` 控件允许把一个复选框放在窗体上；`CheckBoxList` 控件允许把一组复选框放在窗体上。在 ASP.NET 页面上可以使用多个 `CheckBox` 控件，但要把每个复选框看作其相关事件的单一元素。而 `CheckBoxList` 控件允许把多个复选框作为一组对象，为整个组创建特定的事件。

程序清单 5-13 是一个使用 `CheckBox` 控件的例子。

程序清单 5-13 使用 `CheckBox` 控件的单个实例

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void CheckBox1_CheckedChanged(object sender, EventArgs e)
    {
        Response.Write("Thanks for your donation!");
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>CheckBox control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:CheckBox ID="CheckBox1" runat="server" Text="Donate $10 to my cause!"
                OnCheckedChanged="CheckBox1_CheckedChanged" AutoPostBack="true" />
        </div>
    </form>
```

```
</body>
</html>
```

这段代码会生成一个页面, 其中包含一个复选框, 用于请求用户输入捐款。使用 `CheckedChanged` 事件, 就会使用 `CheckBox` 控件的 `OnCheckedChanged` 属性。该属性的值指向 `CheckBox1_CheckedChanged` 事件, 当用户选中该复选框时就会触发该事件。只有当 `AutoPostBack` 属性设置为 `True` (这个属性默认设置为 `False`) 时, 该事件才会触发。运行这个页面, 结果如图 5-13 所示。



图 5-13

5.12.1 如何确定复选框是否被选中

我们可能不想使用复选框的 `AutoPostBack` 属性, 但希望确定在窗体回送给服务器后复选框是否被选中。通过 `If Then` 语句可以做出这个判断, 如下面的例子所示:

```
if (CheckBox1.Checked == true)
{
    Response.Write("Thanks for your donation!");
}
```

使用控件的 `Checked` 属性对 `CheckBox` 的值进行判断。该属性的值是个布尔值, 因此是 `True`(选中)或 `False`(未选中)。



在 VB 和 C# 中, 可以去掉 `true` 值, 仅编写如下代码(这里是 C# 代码):

```
if (CheckBox1.Checked) {...}
```

5.12.2 给复选框赋值

也可以使用 `Checked` 属性, 根据其他动态值确保复选框被选中:

```
if (Member == true)
{
    CheckBox1.Checked = true;
}
```


5.12.3 排列复选框中的文本

在前面的复选框示例中，文本位于复选框的右边，如图 5-14 所示。

使用 CheckBox 控件的 TextAlign 属性，可以重新排列文本，使之显示在复选框的其他位置：

```
<asp:CheckBox ID="CheckBox1" runat="server" Text="Donate $10 to my cause!"
    OnCheckedChanged="CheckBox1_CheckedChanged" AutoPostBack="true"
    TextAlign="Left" />
```

TextAlign 属性的值可以是 Right(默认设置)或 Left。这个属性也可用于 CheckBoxList、RadioButton 和 RadioButtonList 控件。选择 Left 值会得到如图 5-15 所示的结果。



图 5-14



图 5-15

5.13 CheckBoxList 服务器控件

CheckBoxList 服务器控件非常类似于 CheckBox 控件，但前者允许操作一组项，而不是一项。CheckBoxList 服务器控件是相关项的集合，每项都有对应的复选框。

为了说明 CheckBoxList 控件的用法，下面举一个例子，使用 SQL Server 输出 AdventureWorks 示例数据库的 Customer 表中的信息，如程序清单 5-14 所示。



本书使用的 AdventureWorks 数据库版本可以从 www.wrox.com/go/SQLServer2012/DataSets 上下载。

程序清单 5-14 动态填充 CheckBoxList

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = "You selected:<br>";
        foreach(ListItem li in CheckBoxList1.Items)
        {
            if(li.Selected == true)
            {
                Label1.Text += li.Text + "<br>";
            }
        }
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
```

```

<title>CheckBoxList control</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:Button ID="Button1" runat="server" Text="Submit Choices"
        OnClick="Button1_Click" /><br />
      <br />
      <asp:Label ID="Label1" runat="server"></asp:Label>
      <br />
      <asp:CheckBoxList ID="CheckBoxList1" runat="server"
        DataSourceID="SqlDataSource1" DataTextField="CompanyName"
        RepeatColumns="3" BorderColor="Black"
        BorderStyle="Solid" BorderWidth="1px">
      </asp:CheckBoxList>
      <asp:SqlDataSource ID="SqlDataSource1" runat="server"
        SelectCommand="SELECT DISTINCT TOP 12 [CompanyName]
        FROM [SalesLT].[Customer] ORDER BY [CompanyName]"
        ConnectionString="
          <%%$ ConnectionStrings:AdventureWorksConnectionString %>"
      </asp:SqlDataSource>
    </div>
  </form>
</body>
</html>

```

这个 ASP.NET 页面上有一个 `SqlDataSource` 控件，它从 AdventureWorks 数据库中提取需要的信息。在这个控件中使用了 `SELECT` 语句，从 Customer 表的前 10 个不同列表中提取 `CompanyName` 字段。

`CheckBoxList` 控件通过以下几个属性绑定到 `SqlDataSource` 控件上：

```

<asp:CheckBoxList ID="CheckBoxList1" runat="server"
  DataSourceID="SqlDataSource1" DataTextField="CompanyName"
  RepeatColumns="3" BorderColor="Black"
  BorderStyle="Solid" BorderWidth="1px">
</asp:CheckBoxList>

```

`DataSourceID` 属性用于把 `CheckBoxList` 控件与 `SqlDataSource` 控件返回的结果关联起来。`DataTextField` 属性用于从结果中提取需要处理的字段名。在这个例子中，只有字段 `CompanyName` 是可用的。`CheckBoxList` 生成了需要的结果。

剩余的代码包含一些有趣的样式化属性。`BorderColor`、`BorderStyle` 和 `BorderWidth` 属性可以为整个复选框列表绘制边框。更有趣的属性是 `RepeatColumns`，它指定使用多少列来显示结果(本例中为 3 列)。

运行页面，结果如图 5-16 所示。

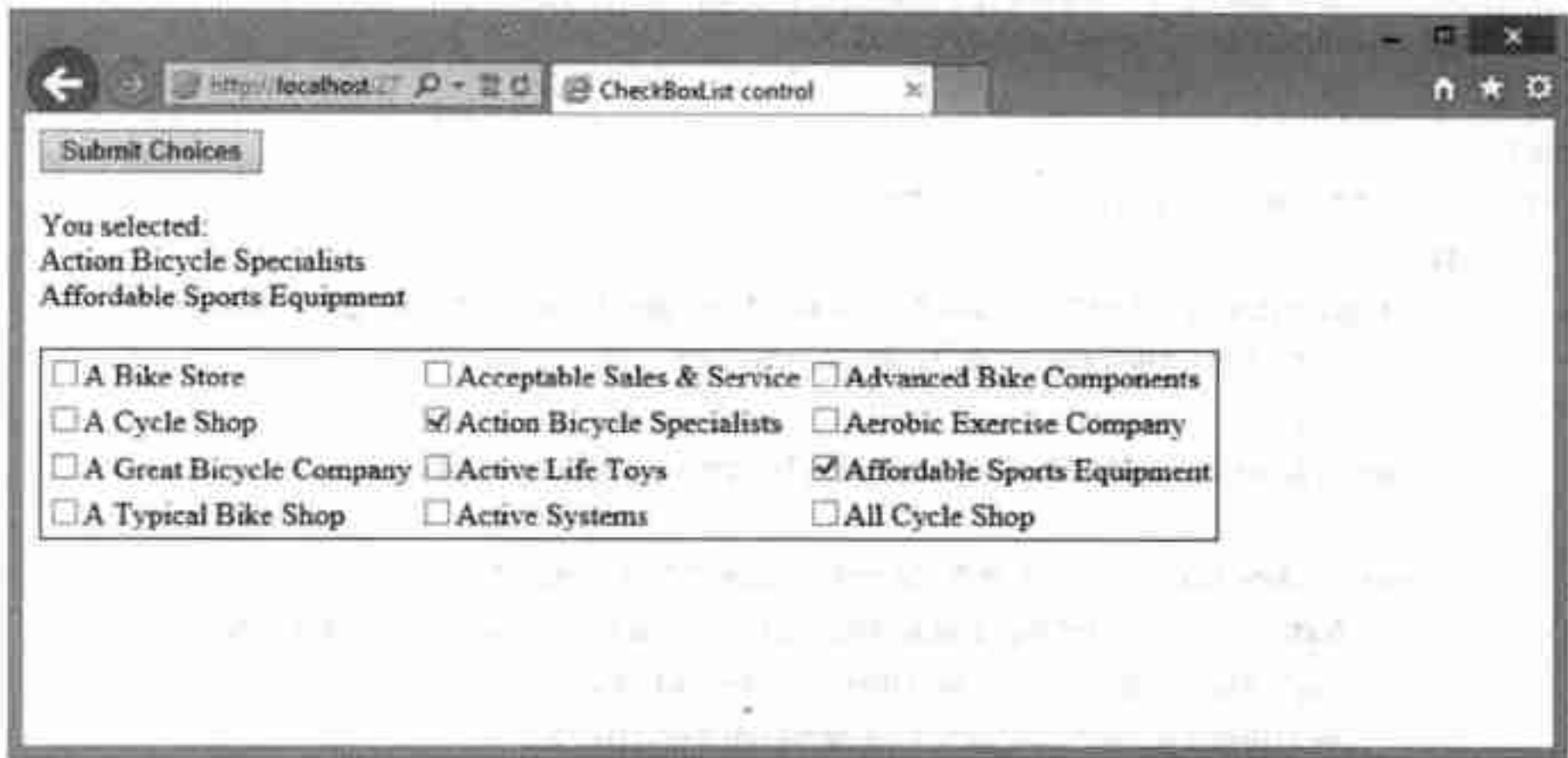


图 5-16

RepeatDirection 属性告诉 CheckBoxList 控件如何在 Web 页面上布置绑定到控件的各项。该属性的值可以是 Vertical 和 Horizontal，默认值是 Vertical。将其值设置为 Vertical，将 RepeatColumn 设置为 3，结果如下所示：

CheckBox1	CheckBox4	CheckBox7
CheckBox2	CheckBox5	CheckBox8
CheckBox3	CheckBox6	CheckBox9

将 RepeatDirection 属性设置为 Horizontal，这些复选框就按照水平方式排列：

CheckBox1	CheckBox2	CheckBox3
CheckBox4	CheckBox5	CheckBox6
CheckBox7	CheckBox8	CheckBox9

5.14 RadioButton 服务器控件

RadioButton 服务器控件非常类似于 CheckBox 服务器控件。它在 Web 页面上放置一个单选按钮。但与复选框不同，窗体上的一个单选按钮没有什么意义。单选按钮一般至少需要两个。通常情况下，页面上的一组 RadioButton 控件采用的是如下构造方式：

```
<asp:RadioButton ID="RadioButton1" runat="server" Text="Yes" GroupName="Set1" />  
<asp:RadioButton ID="RadioButton2" runat="server" Text="No" GroupName="Set1" />
```

图 5-17 显示了结果。

在 RadioButton 控件的代码中，注意标准的 Text 属性把文本放在 Web 窗体上单选按钮的旁边。这里较为重要的属性是 GroupName，它可以在一个 RadioButton 控件上设置，以匹配在其他 RadioButton 控件上设置的该属性。这样，Web 窗体上的单选按钮就可以协同工作。但是，它们到底是如何协同工作的呢？当选中窗体上的一个单选按钮时，就会填满与选中项相关的圆。集合中同一组的其他选中项就会被取消

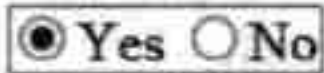


图 5-17

选中，确保集合中只选中一个单选按钮。

程序清单 5-15 是使用 RadioButton 控件的一个例子。

程序清单 5-15 使用 RadioButton 服务器控件

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void RadioButton_CheckedChanged(object sender, EventArgs e)
    {
        if(RadioButton1.Checked == true)
        {
            Response.Write("You selected Visual Basic");
        }
        else
        {
            Response.Write("You selected C#");
        }
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>RadioButton control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:RadioButton ID="RadioButton1" runat="server" Text="Visual Basic"
                GroupName="LanguageChoice"
                OnCheckedChanged="RadioButton_CheckedChanged"
                AutoPostBack="True" />
            <asp:RadioButton ID="RadioButton2" runat="server" Text="C#"
                GroupName="LanguageChoice"
                OnCheckedChanged="RadioButton_CheckedChanged"
                AutoPostBack="True" />
        </div>
    </form>
</body>
</html>
```

与 CheckBox 控件一样，RadioButton 控件也有 CheckedChanged 事件，它把 OnCheckedChanged 属性放在控件上。该属性的值指向服务器端事件，当选中窗体上的两个单选按钮之一时就触发这个事件。AutoPostBack 属性必须设置为 True，该事件才能正常工作。

图 5-18 显示了结果。

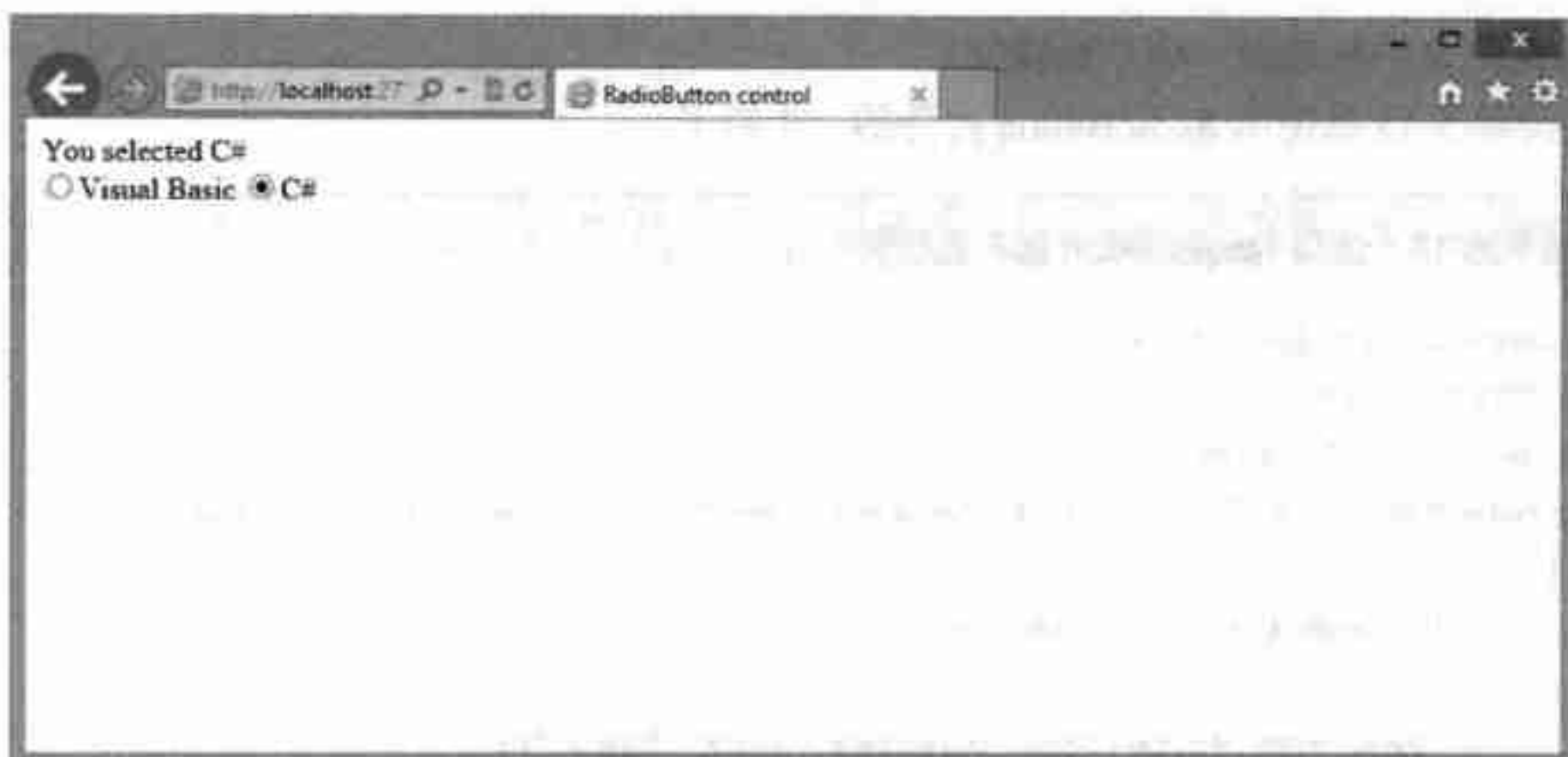


图 5-18

RadioButton 控件优于 RadioButtonList 控件(后面将讨论)的一个方面是, 可以在 RadioButton 控件之间放置其他项(文本、控件或图像), 但 RadioButtonList 控件总是 Web 页面上的单选按钮列表。

5.15 RadioButtonList 服务器控件

RadioButtonList 服务器控件允许在 Web 页面上显示一组单选按钮。RadioButtonList 控件非常类似于 CheckBoxList 和其他列表控件, 因为它允许遍历用户所选择的项, 以进行计数或执行其他操作。通常, RadioButtonList 控件以如下方式写到页面上:

```
<asp:RadioButtonList ID="RadioButtonList1" runat="server">
  <asp:ListItem Selected="True">English</asp:ListItem>
  <asp:ListItem>Russian</asp:ListItem>
  <asp:ListItem>Finnish</asp:ListItem>
  <asp:ListItem>Swedish</asp:ListItem>
</asp:RadioButtonList>
```

与其他列表控件一样, 这个控件也为集合包含的每一项使用 ListItem 对象。在这个例子中, 如果 Selected 属性设置为 True, 那么在第一次生成页面时将默认选择 ListItem 对象。生成的结果如图 5-19 所示。

Selected 属性不是必需的, 但如果希望终端用户在这个集合中选择一些项, 那么最好使用这个属性。使用该属性时, 集合不能为空。

可以使用 RadioButtonList 控件在任意页面方法中检查终端用户选中项的值。程序清单 5-16 中的 Button1_Click 事件提取出了在 RadioButtonList 集合中选中项的值。

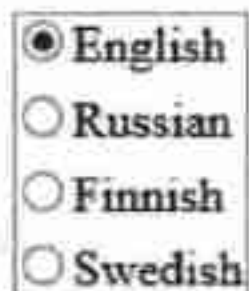


图 5-19

程序清单 5-16 查看在 RadioButtonList 控件中选中项的值

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
  protected void Button1_Click(object sender, EventArgs e)
```

```

    {
        Label1.Text = "You selected: " + RadioButtonList1.SelectedItem.ToString();
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>RadioButtonList Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="Label1" runat="server" />
            <asp:RadioButtonList ID="RadioButtonList1" runat="server">
                <asp:ListItem Selected="True">English</asp:ListItem>
                <asp:ListItem>Russian</asp:ListItem>
                <asp:ListItem>Finnish</asp:ListItem>
                <asp:ListItem>Swedish</asp:ListItem>
            </asp:RadioButtonList>
            <asp:Button ID="Button1" runat="server" Text="Submit"
                OnClick="Button1_Click" />
        </div>
    </form>

```

这段代码获取从 ListItem 对象的 RadioButtonList 集合中选中项的值。使用 ASP.NET 提供的其他列表控件时,也采用这种方式。RadioButtonList 还可以访问 RepeatColumns 和 RepeatDirection 属性(详见 5.13 节)。可以把这个控件绑定到数据源控件提供的项上,以便在 Web 页面上动态创建单选按钮列表。

5.16 Image 服务器控件

Image 服务器控件可以在服务器端代码中操作显示在 Web 页面上的图像。这虽然是一个简单的服务器控件,但可以确定图像如何显示在浏览器屏幕上。Image 控件的构造代码如下所示:

```
<asp:Image ID="Image1" runat="server" ImageUrl="~/Images/Windows.jpg" />
```

其中,重要的属性是 ImageUrl,它指向图像文件的位置。在这里,其位置指定为文件 Windows.jpg。程序清单 5-17 是动态改变 ImageUrl 属性的一个例子。

程序清单 5-17 动态改变 ImageUrl 属性

```

<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Image1.ImageUrl = "~/Images/Windows8.jpg";
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Image Control</title>

```



```

</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:Image ID="Image1" runat="server"
        ImageUrl="~/Images/Windows.jpg" /><br />
      <br />
      <asp:Button ID="Button1" runat="server" Text="Change Image"
        OnClick="Button1_Click" />
    </div>
  </form>
</body>
</html>

```

在这个例子中，当第一次加载页面时，图像(Windows.jpg)就显示在浏览器上。终端用户单击页面上的按钮时，就会在回送过程中加载一幅新图像(Windows8.jpg)。

特定的环境会阻止终端用户查看 Web 页面上的图像，终端用户可能由于自身原因不能看到图像，或者使用了只支持文本的浏览器。在这些情况下，浏览器会查找元素的 longdesc 属性，它指向包含图像描述的文件，并显示该文件的内容。

对于这些情况，Image 服务器控件包含属性 DescriptionUrl，其值是一个文本文件，其中包含所关联图像的完整描述。下面是它的使用方式：

```

<asp:Image ID="Image1" runat="server" ImageUrl="~/Images/Windows.jpg"
  DescriptionUrl="~/WindowsImage.txt" />

```

这段代码在浏览器上生成了如下结果：

```



```

Image 服务器控件不支持用户单击图像的事件。如果希望根据按钮单击来编写事件，应使用本章前面讨论的 ImageButton 服务器控件。

5.17 Table 服务器控件

表是 Web 页面的最常用元素之一，因为<table>元素是用于显示大量信息记录的最常用格式之一。Table 服务器控件的构造代码如下所示：

```

<asp:Table ID="Table1" runat="server">
  <asp:TableRow ID="TableRow1" runat="server" Font-Bold="True"
    ForeColor="White" BackColor="DarkGray">
    <asp:TableHeaderCell>First Name</asp:TableHeaderCell>
    <asp:TableHeaderCell>Last Name</asp:TableHeaderCell>
  </asp:TableRow>
  <asp:TableRow>
    <asp:TableCell>Jason</asp:TableCell>
    <asp:TableCell>Gaylord</asp:TableCell>
  </asp:TableRow>
  <asp:TableRow>
    <asp:TableCell>Scott</asp:TableCell>
    <asp:TableCell>Hanselman</asp:TableCell>
  </asp:TableRow>
</asp:Table>

```

```

<asp:TableRow>
    <asp:TableCell>Todd</asp:TableCell>
    <asp:TableCell>Miranda</asp:TableCell>
</asp:TableRow>
<asp:TableRow>
    <asp:TableCell>Pranav</asp:TableCell>
    <asp:TableCell>Rastogi</asp:TableCell>
</asp:TableRow>
</asp:Table>

```

这段代码会生成一个两列的表，如图 5-20 所示。

可以对 Table 服务器控件进行许多操作。例如，可以动态地给表添加行，如程序清单 5-18 所示。

First Name	Last Name
Jason	Gaylord
Scott	Hanselman
Todd	Miranda
Pranav	Rastogi

图 5-20

程序清单 5-18 动态地给表添加行

```

<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        TableRow tr = new TableRow();
        TableCell fname = new TableCell();
        fname.Text = "Christian";
        tr.Cells.Add(fname);
        TableCell lname = new TableCell();
        lname.Text = "Wenz";
        tr.Cells.Add(lname);
        Table1.Rows.Add(tr);
    }
</script>

```

要为 Table 控件添加一行，就必须创建 TableRow 和 TableCell 对象。首先创建 TableCell 对象，再把它们放在 TableRow 对象中，最后把 TableRow 对象添加到 Table 对象中。

Table 服务器控件显然包含一些前面没有介绍的功能。一个较简单的功能是在 Web 页面上为表添加标题。图 5-21 显示了带标题的表。

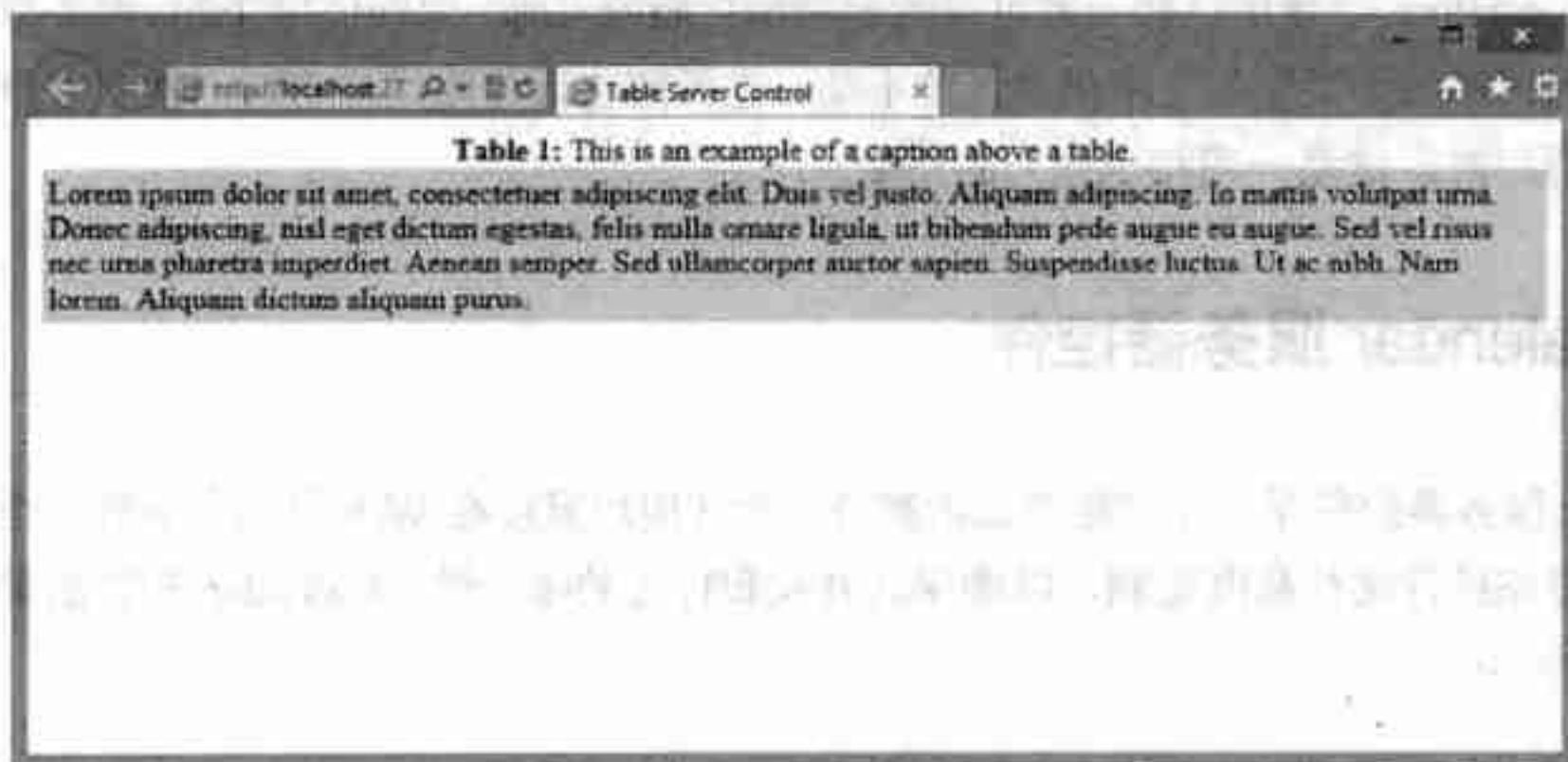


图 5-21

为了给表添加标题，只需使用 Table 控件的 Caption 属性，如程序清单 5-19 所示。

程序清单 5-19 使用 Caption 属性

```
<%@ Page Language="C#" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Table Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Table ID="Table1" runat="server"
            Caption="<b>Table 1:</b> This is an example of a caption above a table."
            BackColor="Gainsboro">
            <asp:TableRow ID="Tablerow1" runat="server">
                <asp:TableCell ID="Tablecell1" runat="server">
                    Lorem ipsum dolor sit
                    amet, consectetur adipiscing elit. Duis vel justo. Aliquam
                    adipiscing. In mattis volutpat urna. Donec adipiscing, nisl ege
                    dictum egestas, felis nulla ornare ligula, ut bibendum pede aug
                    eu augue. Sed vel risus nec urna pharetra imperdiet. Aenean
                    semper. Sed ullamcorper auctor sapien. Suspendisse luctus. Ut a
                    nibh. Nam lorem. Aliquam dictum aliquam purus.
                </asp:TableCell>
            </asp:TableRow>
        </asp:Table>
    </form>
</body>
</html>
```

标题默认放在表顶部的中央，但可以使用另一个属性 CaptionAlign 来控制标题的位置。该属性的值可以是 Bottom、Left、NotSet、Right 和 Top。

在以前的 ASP.NET 版本中，<asp:Table>元素可以包含任意数量的<asp:TableRow>元素。在 ASP.NET 4.5 中，还有一些其他元素可以嵌入到<asp:Table>元素中。这些新元素包括<asp:TableHeaderRow>和<asp:TableFooterRow>。使用这些元素可以给表添加页眉或页脚，以便使用 Table 服务器控件给大量数据分页，而表示数据类型的文本仍然保留不动。在处理告诉终端用户有时一次只能查看几条记录的移动应用程序时，这是一项相当强大的功能。

5.18 Calendar 服务器控件

Calendar 服务器控件是一个功能丰富的控件，允许用户直接在 Web 页面上放置一个功能完善的日历。可以对该控件进行高度定制，以确保其外观和行为的唯一性。Calendar 控件的最简单形式如下面的代码所示：

```
<asp:Calendar ID="Calendar1" runat="server"></asp:Calendar>
```


这段代码在 Web 页面上生成一个日历，并且没有添加任何样式，如图 5-22 所示。

October 2012						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

图 5-22

5.18.1 从 Calendar 控件中选择日期

Calendar 控件允许在月份中滚动，选择所显示月份中的特定日期。程序清单 5-20 中的简单应用程序允许用户选择某个月份中的某一天。

程序清单 5-20 在 Calendar 控件中选择某一天

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Calendar1_SelectionChanged(object sender, EventArgs e)
    {
        Response.Write("You selected: " +
            Calendar1.SelectedDate.ToShortDateString());
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Using the Calendar Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Calendar ID="Calendar1" runat="server"
                OnSelectionChanged="Calendar1_SelectionChanged"></asp:Calendar>
        </div>
    </form>
</body>
</html>
```

运行这个应用程序会在浏览器上显示日历，终端用户可以选择日期。选择了日期后，使用 OnSelectionChange 属性触发 Calendar1_SelectionChanged 事件。这个事件把选中的日期值显示到屏幕上，结果如图 5-23 所示。

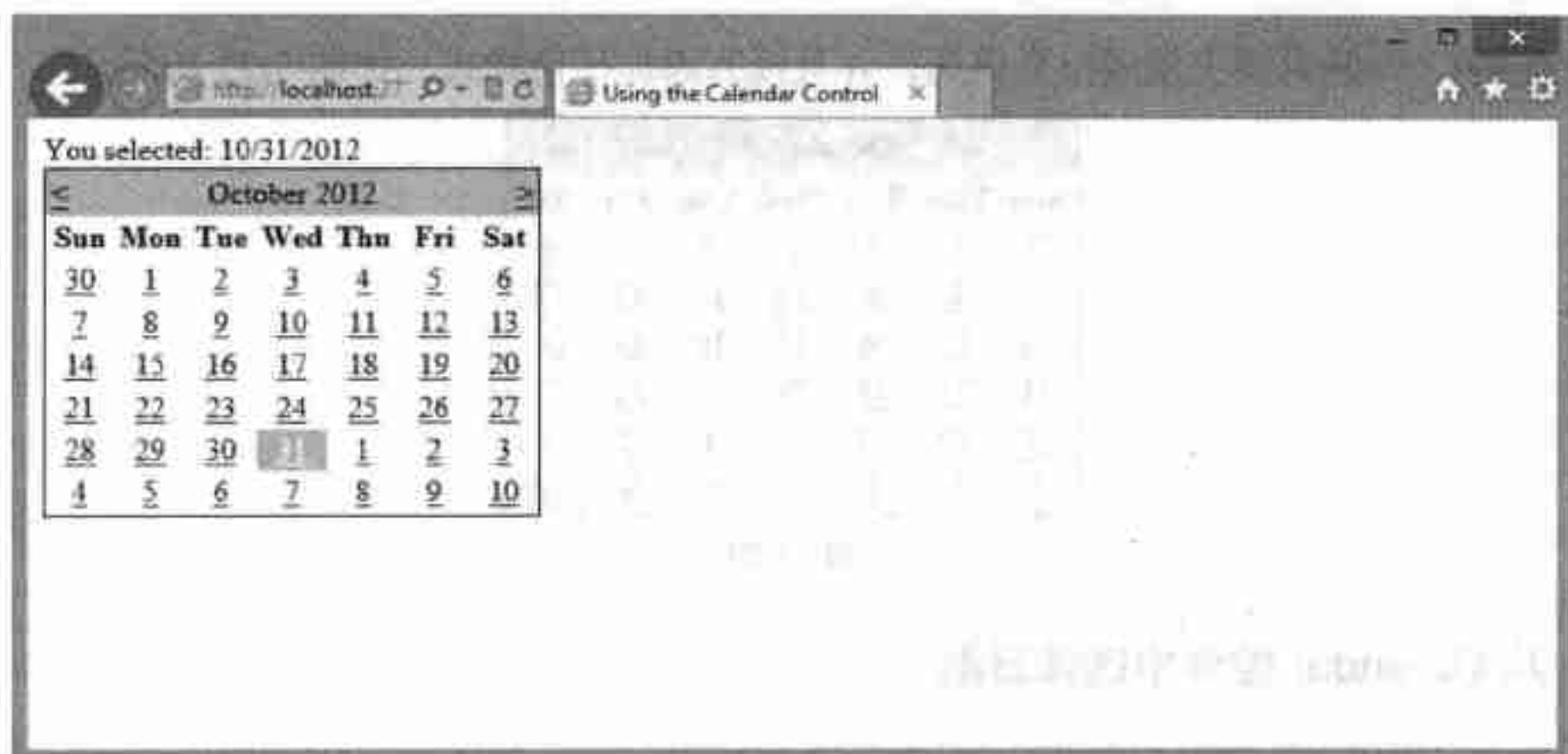


图 5-23

5.18.2 从 Calendar 控件中选择要输出的日期格式

使用 `Calendar1_SelectionChanged` 事件时, 选中的日期通过 `ToShortDateString()` 方法来输出。

Calendar 控件也允许以许多其他格式输出日期, 如下所示:

- `ToFileTime`: 把选中的项转换为本地操作系统的文件时间 1299612960000000000。
- `ToFileTimeUtc`: 把选中的项转换为操作系统的文件时间, 但不使用本地时间, 而使用 UTC 时间 12996115260000000000。
- `ToLocalTime`: 把当前的国际标准时间(UTC)转换为本地时间 10/30/2012 8:00:00 PM。
- `ToLongDateString`: 把选中的项转换为长格式的可读字符串 Wednesday, October 31, 2012。
- `ToLongTimeString`: 把选中的项转换为长格式的时间值(不包括日期)12:00:00 AM。
- `ToOADate`: 把选中的项转换为 OLE 自动日期 41213。
- `ToShortDateString`: 把选中的项转换为短格式的可读字符串 10/31/2012。
- `ToShortTimeString`: 把选中的项转换为短格式的时间值(不包括日期)12:00 AM。
- `ToString`: 把选中的项转换为格式 10/31/2012 12:00:00 AM。
- `ToUniversalTime`: 把选中的项转换为国际标准时间(UTC)10/31/2012 4:00:00 AM。

5.18.3 选择日期、星期或月份

Calendar 控件默认可以选择日期。可以使用 `SelectionMode` 属性修改该选择操作, 以允许用户从日历中选择星期或月份。这个属性的值可以是 `Day`、`DayWeek`、`DayWeekMonth` 和 `None`。

`Day` 设置允许在日历中单击某个日期以突出显示(这是默认设置)。使用 `DayWeek` 设置仍然可以选择某个日期, 还可以单击星期旁边的箭头(如图 5-24(b)所示), 选择整个星期。使用 `DayWeekMonth` 设置可以选择日期或星期。日历左上角的新箭头允许用户选择整个月份(如图 5-24(c)所示)。None 设置表示终端用户不能进行任何选择, 这种日历适合于只提供信息的站点。

< October 2012 >						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

(a)

< October 2012 >						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

(b)

< October 2012 >						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

(c)

图 5-24

5.18.4 使用日期范围

即使终端用户选择的项包含整个星期或月份，选中的项也只能是该日期范围中的第一天。例如，如果允许用户选择整个月份，而用户选择了2012年10月，那么选中的日期(使用 `ToShortDateString()`)就是2012年10月1日，即选中的是日期范围中的第一天。这虽然可能是合适的行为，但如果需要所选范围的所有日期，该怎么办呢？程序清单5-21说明了如何实现这一功能。

程序清单 5-21 从选择的项获取日期范围

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Calendar1_SelectionChanged(object sender, EventArgs e)
    {
        Label1.Text = "<b><u>You selected the following date/dates:</u></b><br>";
        for(int i = 0; i < Calendar1.SelectedDates.Count; i++)
        {
            Label1.Text += Calendar1.SelectedDates[i].ToShortDateString() + "<br>";
        }
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Using the Calendar Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Calendar ID="Calendar1" runat="server"
                OnSelectionChanged="Calendar1_SelectionChanged"
                SelectionMode="DayWeekMonth"></asp:Calendar>
            <p>
                <asp:Label ID="Label1" runat="server"></asp:Label>
            </p>
        </div>
    </form>
</body>
</html>
```



```

        </div>
    </form>
</body>
</html>

```

在这个例子中，Calendar 控件允许用户选择一天、一星期甚至一个月。使用 For Next 循环和 SelectedDates.Count 属性就可以遍历选项。代码生成的结果如图 5-25 所示。

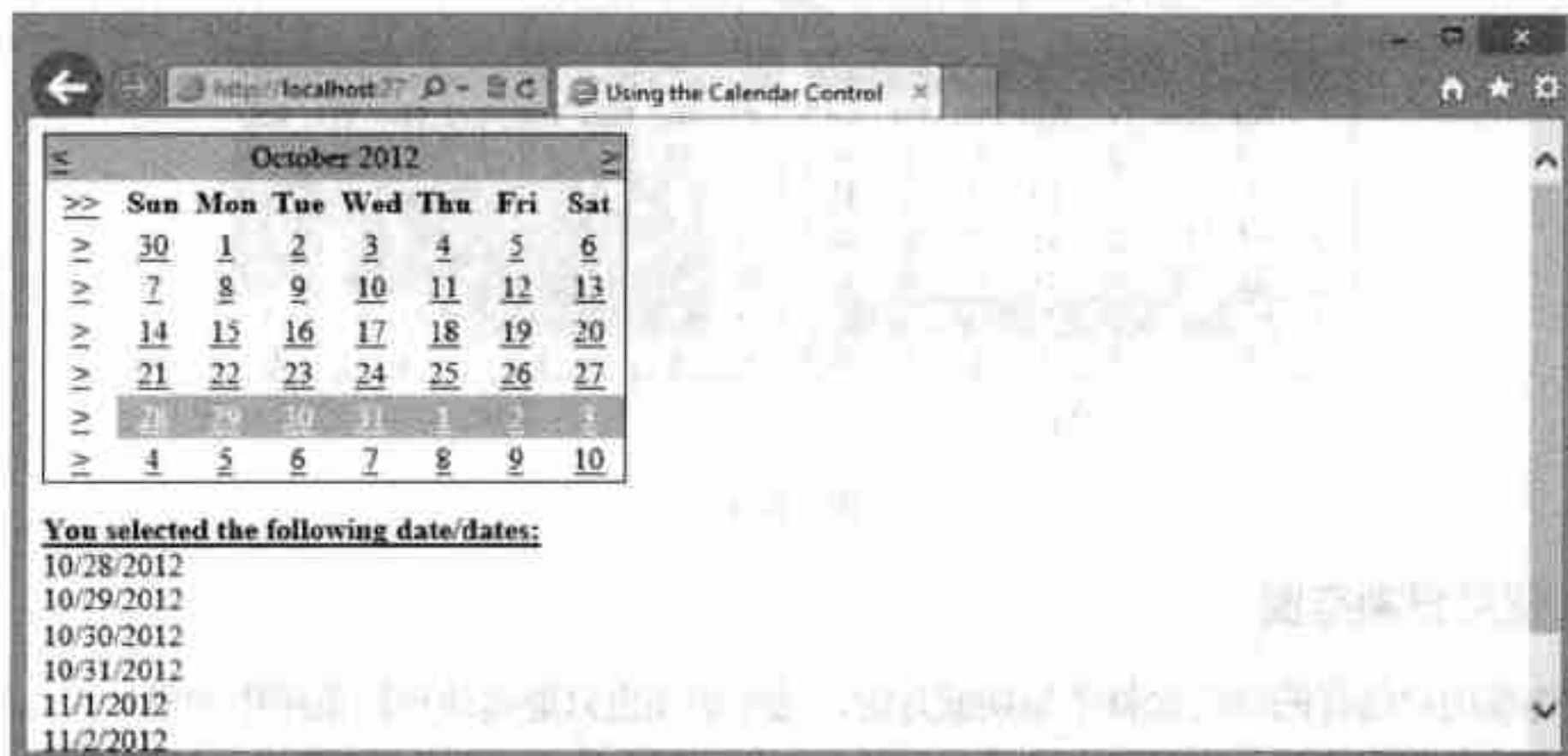


图 5-25

使用下面的代码可以获得选中日期范围中的第一天：

```
Calendar1.SelectedDates[0].ToShortDateString();
```

使用下面的代码可以获得选中日期范围中的最后一天：

```
Calendar1.SelectedDates[Calendar1.SelectedDates.Count-1].ToShortDateString();
```

可以看出，使用 SelectedDates 对象的 Count 属性即可获得需要的结果。

5.18.5 修改日历的样式和操作方式

Calendar 控件有许多内容，而且多于本章介绍的内容。Calendar 控件的一个优点是易于扩展。下面开始研究进一步定制这个控件的新方式，首先查看修改它的最简单方式：给该控件应用样式。

使用 Visual Studio，可以在所操作页面的设计视图中给控件指定新的外观和操作方式。突出显示 Calendar 控件，打开该控件的智能标记，就会看到 Auto Format 链接。其中有一列可用样式，它们可以应用于 Calendar 控件。



这个功能并非 Calendar 控件独有，其他许多控件也提供了样式列表。该功能总是位于控件的智能标记中。

图 5-26 显示了一些样式。



图 5-26

除了改变 Calendar 控件的样式之外,还可以在其显示过程中操作该控件。Calendar 控件支持 DayRender 事件,该事件允许控制日历中的某个日期或所有日期如何显示。程序清单 5-22 中的例子说明了如何改变日历中显示的某个日期。

程序清单 5-22 控制日历中的日期如何显示

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Calendar1_DayRender(object sender, DayRenderEventArgs e)
    {
        e.Cell.VerticalAlign = VerticalAlign.Top;
        if(e.Day.DayNumberText == "18")
        {
            e.Cell.Controls.Add(new LiteralControl("<p>User Group Meeting!</p>"));
            e.Cell.BorderColor = System.Drawing.Color.Black;
            e.Cell.BorderWidth = 1;
            e.Cell.BorderStyle = BorderStyle.Solid;
            e.Cell.BackColor = System.Drawing.Color.LightGray;
        }
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Using the Calendar Control</title>
```

```

</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:Calendar ID="Calendar1" runat="server"
        OnDayRender="Calendar1_DayRender" Height="190px"
        BorderColor="White"
        Width="350px" ForeColor="Black" BackColor="White" BorderWidth="1px"
        NextPrevFormat="FullMonth" Font-Names="Verdana" Font-Size="9pt">
        <SelectedDayStyle ForeColor="White"
          BackColor="#333399"></SelectedDayStyle>
        <OtherMonthDayStyle ForeColor="#999999"></OtherMonthDayStyle>
        <TodayDayStyle BackColor="#CCCCCC"></TodayDayStyle>
        <NextPrevStyle ForeColor="#333333" VerticalAlign="Bottom"
          Font-Size="8pt" Font-Bold="True"></NextPrevStyle>
        <DayHeaderStyle Font-Size="8pt" Font-Bold="True"></DayHeaderStyle>
        <TitleStyle ForeColor="#333399" BorderColor="Black"
          Font-Size="12pt" Font-Bold="True"
          BackColor="White" BorderWidth="4px">
        </TitleStyle>
      </asp:Calendar>
    </div>
  </form>
</body>
</html>

```

在这个例子中，使用了 Calendar 控件和一些样式。在浏览器上编译和运行页面时，Calendar1_DayRender 事件中的代码修改了日历中每月的 18 号。作为结果的日历如图 5-27 所示。

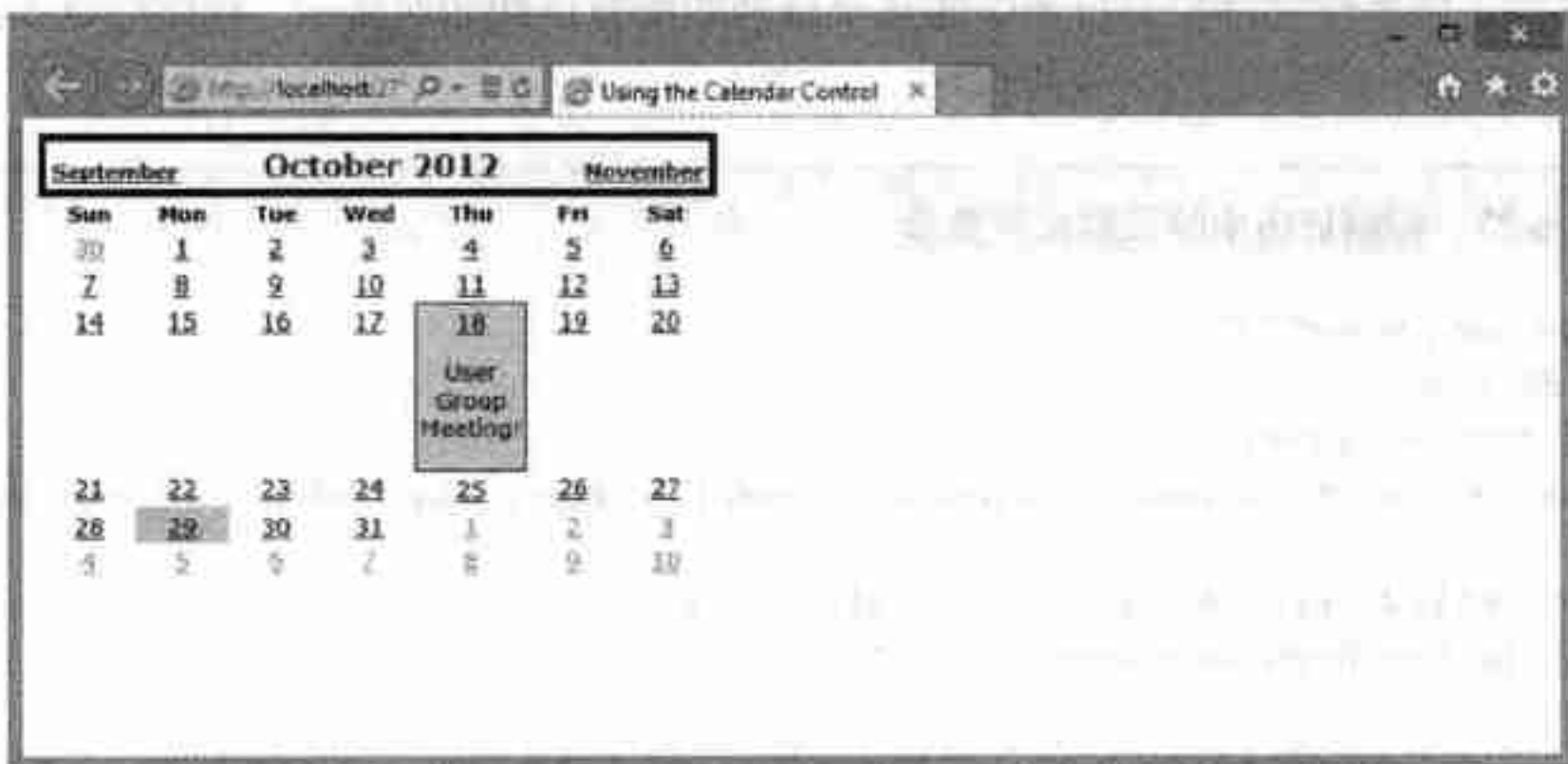


图 5-27

这个例子中的 Calendar 控件增加了属性 OnDayRender，它指向 Calendar1_DayRender 事件。针对日历中显示的每一天都会运行该事件。类构造函数说明，这里并没有使用 System.EventArgs 类，而是使用了 DayRenderEventArgs 类。它允许访问日历中显示的每一天。

DayRenderEventArgs 类中的两个主要属性是 Cell 和 Day。Cell 属性允许访问显示日期的单元格，Day 属性允许访问单元格中显示的特定日期。

从 Calendar1_DayRender 事件执行的操作中可以看出，此处使用了这两个属性。首先，Cell 属

性把单元格的垂直对齐方式设置为 Top。如果不进行这样的设置,当一个单元格中有内容时,表可能看起来就很古怪。然后检查所显示的日期(使用 Day 属性检查)是否是该月的 18 号。如果是,就运行 If Then 语句,使用 Cell 属性改变该单元格的样式。新样式增加了一个控件,还改变了单元格的边框和颜色。

可以看出,处理日历中的单个日期是相当容易的。可以很轻易地给日期指定需要的内容和外观。

Day 属性的一项有用功能是,通过把 Day 属性的 IsSelectable 设置为 False,可以关闭用于选择某个日期或日期范围的选项:

```
public void Calendar1_DayRender(Object sender, DayRenderEventArgs e)
{
    if(e.Day.Date < DateTime.Now)
    {
        e.Day.IsSelectable = false;
    }
}
```

5.19 AdRotator 服务器控件

虽然很多 Web 用户都觉得广告很烦人,但广告在 Web 上无处不在。有了 AdRotator 控件,就可以配置应用程序,使其向终端用户显示一系列广告。在这个控件中,可以使用源代码中的广告数据来代替该控件先前版本所使用的标准 XML 文件。

如果使用 XML 源获取广告信息,那么首先要创建 XML 广告文件。XML 广告文件可以添加一些新元素,以便更多地控制广告的外观和操作方式。程序清单 5-23 是 XML 广告文件的一个例子。

程序清单 5-23 XML 广告文件

```
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements
  xmlns="http://schemas.microsoft.com/AspNet/AdRotator-Schedule-File">
  <Ad>
    <ImageUrl>book1.jpg</ImageUrl>
    <NavigateUrl>http://www.wrox.com</NavigateUrl>
    <AlternateText>Beginning ASP.NET</AlternateText>
    <Impressions>50</Impressions>
    <Keyword>ASP.NET</Keyword>
  </Ad>
  <Ad>
    <ImageUrl>book2.jpg</ImageUrl>
    <NavigateUrl>http://www.wrox.com</NavigateUrl>
    <AlternateText>Beginning Visual C#</AlternateText>
    <Impressions>50</Impressions>
    <Keyword>C#</Keyword>
  </Ad>
</Advertisements>
```

这个 XML 文件用于存储出现在应用程序中的广告信息,其中包含一些元素(见表 5-1)。这些元素都是可选的。

表 5-1

元 素	描 述
ImageUrl	一个字符串值，表示要使用的图像的位置
NavigateUrl	一个字符串值，表示单击图像时要发送到的 URL
AlternateText	一个字符串值，如果图像在客户的浏览器上已关闭，或者没有找到图像，就显示这个字符串值
Impressions	一个数值，表示要显示的图像被选中的可能性
Keyword	一个字符串值，设置图像的类别，以确定是否允许过滤广告

有了 XML 广告文件之后，就可以使用 AdRotator 控件读取这个文件。程序清单 5-24 是使用该控件的一个例子。

程序清单 5-24 使用 AdRotator 控件作为横幅广告

```
<%@ Page Language="C#" %>

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>AdRotator Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:AdRotator ID="AdRotator1" runat="server"
                AdvertisementFile="Listing05-23.xml" />
            <p>
                Lorem ipsum dolor sit
                amet, consectetur adipiscing elit. Duis vel justo. Aliquam
                adipiscing. In mattis volutpat urna. Donec adipiscing, nisl eget
                dictum egestas, felis nulla ornare ligula, ut bibendum pede augue
                eu augue. Sed vel risus nec urna pharetra imperdiet. Aenean
                semper. Sed ullamcorper auctor sapien. Suspendisse luctus. Ut ac
                nibh. Nam lorem. Aliquam dictum aliquam purus.
            </p>
        </div>
    </form>
</body>
</html>
```

这个例子在页面的顶部把 XML 广告文件中指定的广告显示为横幅广告。

不一定要把所有的广告信息都放在 XML 广告文件中，而可以使用另一个数据源来绑定 AdRotator 控件。例如，把 AdRotator 控件绑定到 SqlDataSource 对象上，以如下方式从 SQL Server 中提取广告信息：

```
<asp:AdRotator ID="AdRotator1" runat="server"
    DataSourceId="SqlDataSource1" AlternateTextField="AlternateTF"
    ImageUrlField="Image" NavigateUrlField="NavigateUrl" />
```

AlternateTextField、ImageUrlField 和 NavigateUrlField 属性指向 SQL Server 中使用的这些项的列名。

5.20 Xml 服务器控件

Xml 服务器控件可以使用 XSL 样式表获取和转换 XML。Xml 控件可以使用两种不同的方式操作 XML。较为简单的方式是使用程序清单 5-25 中的构造代码。

程序清单 5-25 显示 XML 文档

```
<asp:Xml ID="Xml1" runat="server" DocumentSource="Food.xml"
  TransformSource="FoodTemplate.xslt"></asp:Xml>
```

这个方法只使用两个属性：一个是 DocumentSource，它指向 XML 文件的路径；另一个是 TransformSource，它提供了在转换 XML 文档时要使用的 XSLT 文件。

使用 Xml 服务器控件的另一种方式是把 XML 加载到一个对象中，再把该对象传送给 Xml 控件，如程序清单 5-26 所示。

程序清单 5-26 把 XML 文件加载到一个对象中，再把该对象提供给 Xml 控件

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Xml.Xsl" %>
<%@ Import Namespace="System.Xml.XPath" %>
<!DOCTYPE html>
<script runat="server">
    protected void Page_Load(Object sender, EventArgs e)
    {
        XPathDocument MyXmlDoc = new XPathDocument(Server.MapPath("Food.xml"));
        XslTransform MyXsltDoc = new XslTransform();
        MyXsltDoc.Load(Server.MapPath("FoodTemplate.xslt"));
        Xml1.XPathNavigator = MyXmlDoc.CreateNavigator();
        Xml1.Transform = MyXsltDoc;
    }
</script>
```



XslTransform 已被 XslCompiledTransform 替代。XML 控件没有更新为使用 XslCompiledTransform，可以继续使用 XslTransform，但在编译时会收到警告。

为了使控件正常工作，必须把 System.Xml 和 System.Xml.Xsl 名称空间导入页面。这个例子首先加载 XML 和 XSL 文件，然后再把这些文件赋给 Document 和 Transform 属性。

5.21 Panel 服务器控件

Panel 服务器控件可以封装一组操作或布局 ASP.NET 页面的控件。它基本上是所有其他控件的容器，可以把一组服务器控件及其他元素(如 HTML 和图像)转变为单个单元。

使用 Panel 控件封装一组其他元素的优点是，可以使用 Panel 控件的一个属性，把这些元素作为

单个单元来处理。例如,把 Font-Bold 属性设置为 True,会使 Panel 控件上的每一项都采用这个属性。

Panel 控件还有一个功能,就是滚动栏会根据 Panel 控件包含的信息量自动地滚动显示。甚至可以指定滚动栏的显示方式。

为了举例说明滚动栏的用法,可查看 Lorem Ipsum 文本的长版本(www.lipsum.com),并把文本放在 Panel 控件上,如程序清单 5-27 所示。

程序清单 5-27 使用 Panel 服务器控件的滚动栏功能

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Panel Server Control Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Panel ID="Panel1" runat="server" Height="300" Width="300"
            ScrollBars="auto">
            <p>Lorem ipsum dolor sit amet . . . </p>
        </asp:Panel>
    </form>
</body>
</html>
```

给 Panel 服务器控件的 Height 和 Width 属性赋值,并使用 ScrollBars 属性(本例中设置为 Auto),就可以在指定的区域内使用滚动栏显示该控件包含的信息,如图 5-28 所示。

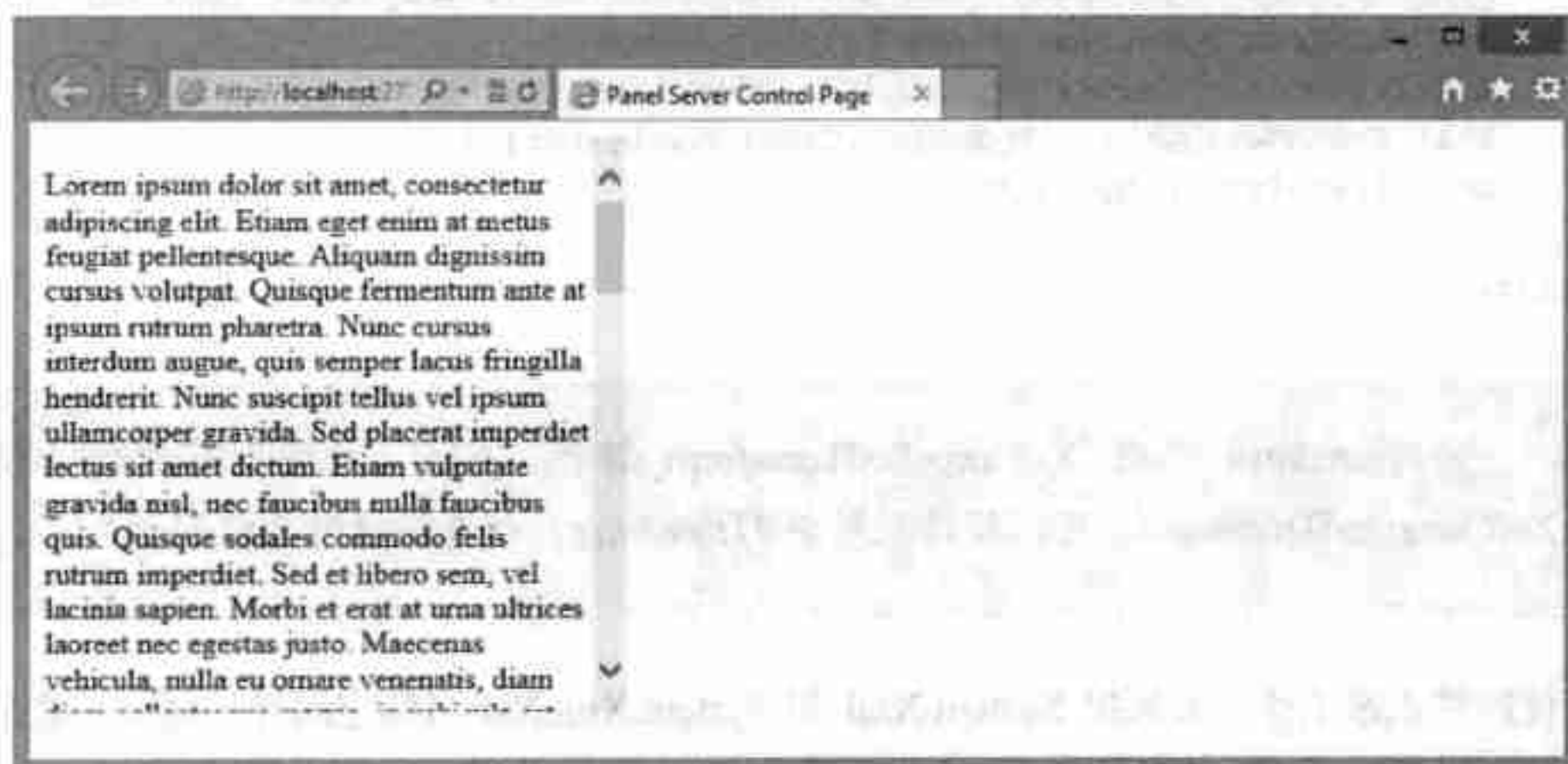


图 5-28

可以看出,在 300×300 像素的区域内添加了一个垂直滚动栏。Panel 控件默认根据需要使文本自动换行。要改变这种行为,可使用采用布尔值的 Wrap 属性:

```
<asp:Panel ID="Panel1" runat="server" Height="300" Width="300"
    ScrollBars="auto" Wrap="False" />
```

关闭自动换行功能可能会显示水平滚动栏(这取决于面板中包含的内容)。

如果不希望 ASP.NET 引擎选择要激活的滚动栏,可以使用 ScrollBars 属性进行判断。除了 Auto

值外, 该属性的值还可以是 None、Horizontal、Vertical 和 Both。

改变 Panel 控件的操作方式的另一个有趣属性是 HorizontalAlign, 它允许指定 Panel 控件中的内容如何水平对齐。该属性的值可以是 NotSet、Center、Justify、Left 和 Right。图 5-29 显示了 Panel 控件以不同水平对齐方式排列的结果。



图 5-29

还可以使用 Direction 属性, 把垂直滚动栏移动到 Panel 控件的左边。Direction 属性可以设置为 NotSet、LeftToRight 和 RightToLeft。在使用从右到左书写的语言(如一些亚洲语言)时, 使用 RightToLeft 设置是很理想的方式。但是, 该设置也会把滚动栏移动到 Panel 控件的左边。如果滚动栏在左边, 可将 HorizontalAlign 属性设置为 Left, 内容的显示如图 5-30 所示。

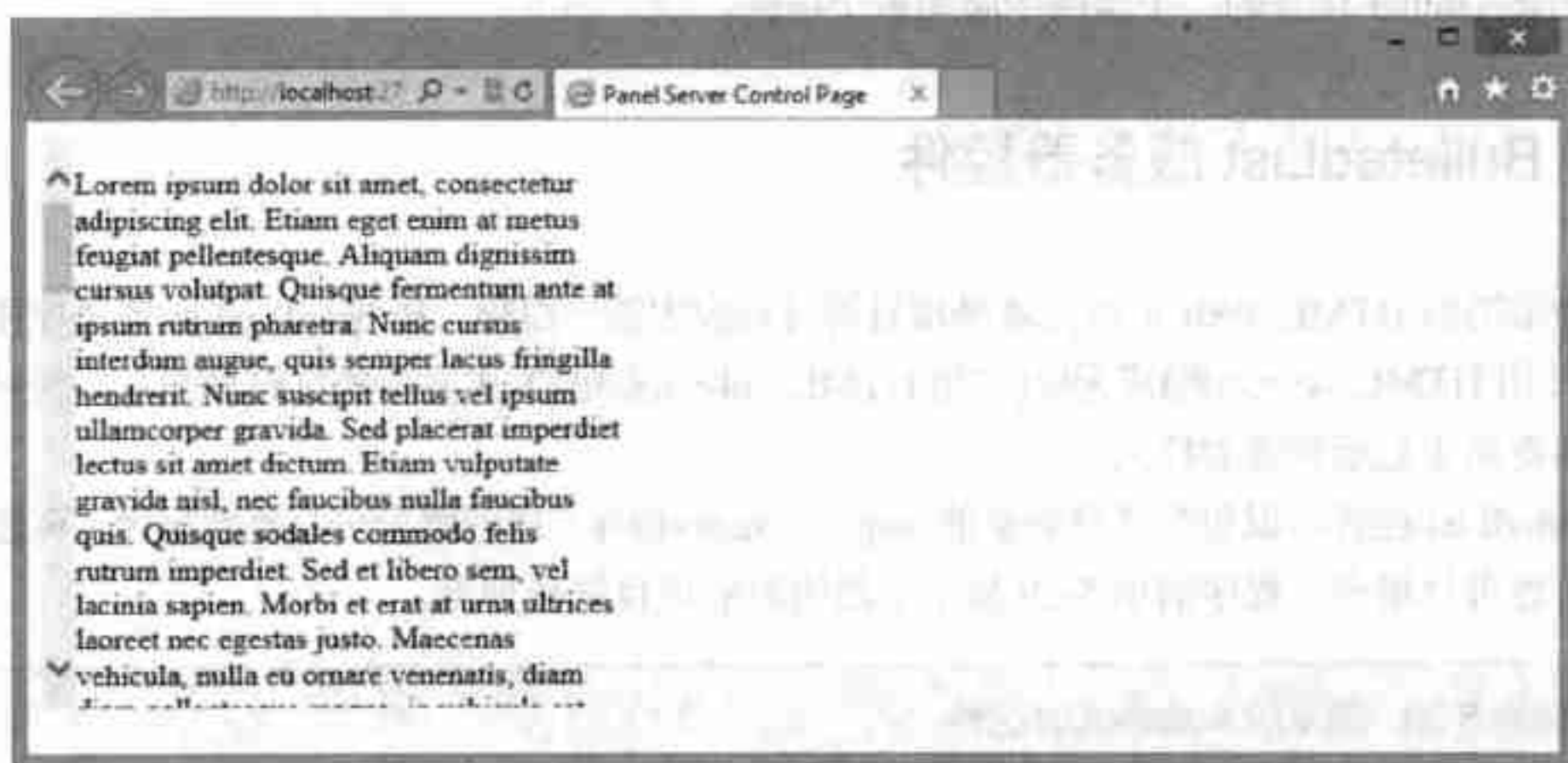


图 5-30

5.22 Placeholder 服务器控件

顾名思义, Placeholder 服务器控件就是占位符, 可以把对象动态插入页面。可以将该控件看作添加其他控件的标记。Panel 控件也有在页面的指定位置添加控件的功能。

为了说明该控件的用法, 下面在页面中插入一个 Placeholder 控件, 再以程序清单 5-28 所示的方式在服务器端代码中给该页面添加一些控件。

程序清单 5-28 使用 Placeholder 服务器控件给页面动态地添加一些控件

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
```



```

<script runat="server">
    protected void Page_Load(Object sender, EventArgs e)
    {
        Label MyNameLabel = new Label();
        MyNameLabel.Text = "Welcome, Jason!";
        Placeholder1.Controls.Add(MyNameLabel);
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Placeholder Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Placeholder ID="Placeholder1" runat="server" />
        </div>
    </form>
</body>
</html>

```

这个例子创建了 Label 控件的一个新实例，并给它赋值，之后把它添加到 Placeholder 控件中。可以在 Placeholder 控件的一个实例中添加多个控件。

5.23 BulletedList 服务器控件

一种常用的 HTML Web 页面元素是项目符号列表中的一组项。BulletedList 服务器控件可以按照有序(使用 HTML 元素)或无序(使用 HTML 元素)的方式显示项目符号列表。另外，该控件可以确定用于显示列表的样式。

BulletedList 控件可以包含任意数量的<asp:ListItem>控件，或者绑定到某类数据源，再使用检索出来的内容进行填充。程序清单 5-29 显示了最简单的项目符号列表。

程序清单 5-29 简单的 BulletedList 控件

```

<%@ Page Language="C#" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>BulletedList Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:BulletedList ID="Bulletedlist1" runat="server">
            <asp:ListItem>United States</asp:ListItem>
            <asp:ListItem>United Kingdom</asp:ListItem>
            <asp:ListItem>Finland</asp:ListItem>
            <asp:ListItem>Russia</asp:ListItem>
            <asp:ListItem>Sweden</asp:ListItem>
            <asp:ListItem>Estonia</asp:ListItem>
        </asp:BulletedList>
    </form>
</body>
</html>

```



```

    </form>
</body>
</html>

```

使用`<asp:BulletedList>`元素和`<asp:ListItem>`元素,可以生成一个简单的项目符号列表,如图 5-31 所示。

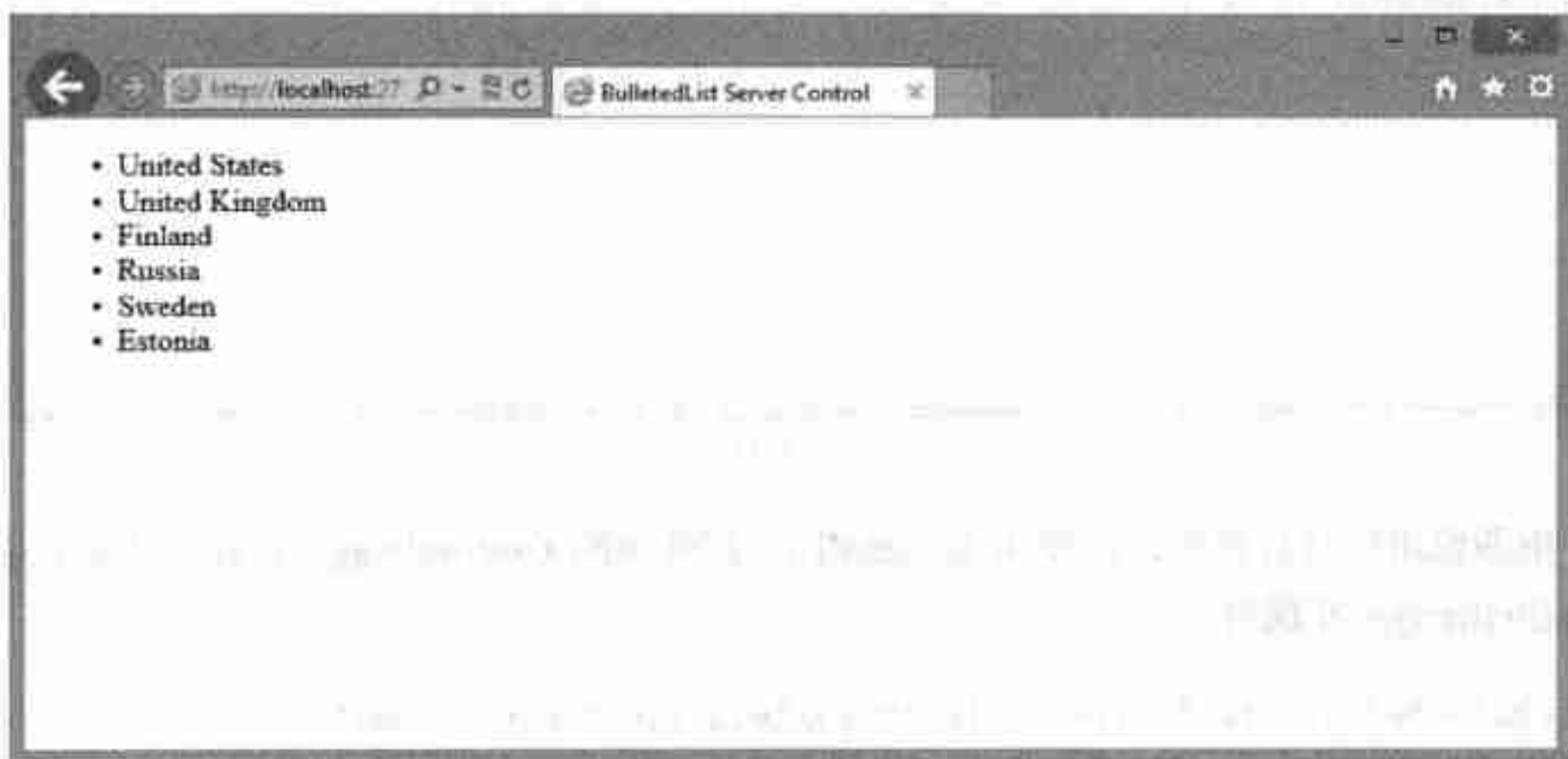


图 5-31

`BulletedList` 控件也支持通过一两个属性就可以轻松改变列表的样式。`BulletStyle` 属性可以改变列表中每一项前面的项目符号的样式,它的值可以是 `Numbered`、`LowerAlpha`、`UpperAlpha`、`LowerRoman`、`UpperRoman`、`Disc`、`Circle`、`Square`、`NotSet` 和 `CustomImage`。图 5-32 是这些样式的范例(其中不包括 `CustomImage` 设置的范例,它允许使用任意图像)。

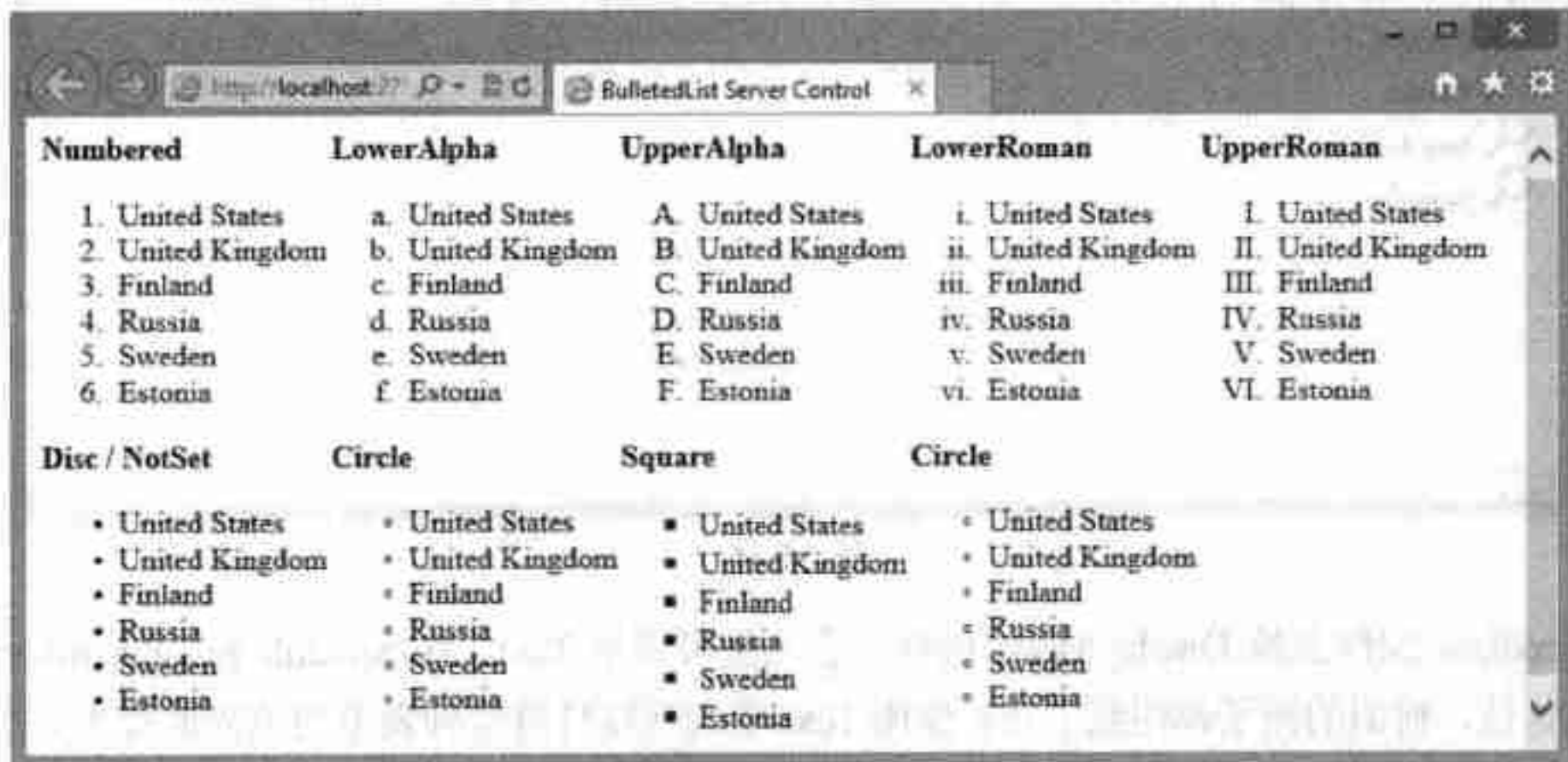


图 5-32

可以使用 `FirstBulletNumber` 属性改变编号样式(`Numbered`、`LowerAlpha`、`UpperAlpha`、`LowerRoman`、`UpperRoman`)中第一项的起始值。例如,使用 `UpperRoman` 设置时,该属性的值设置为 5,就会得到如图 5-33 所示的格式。

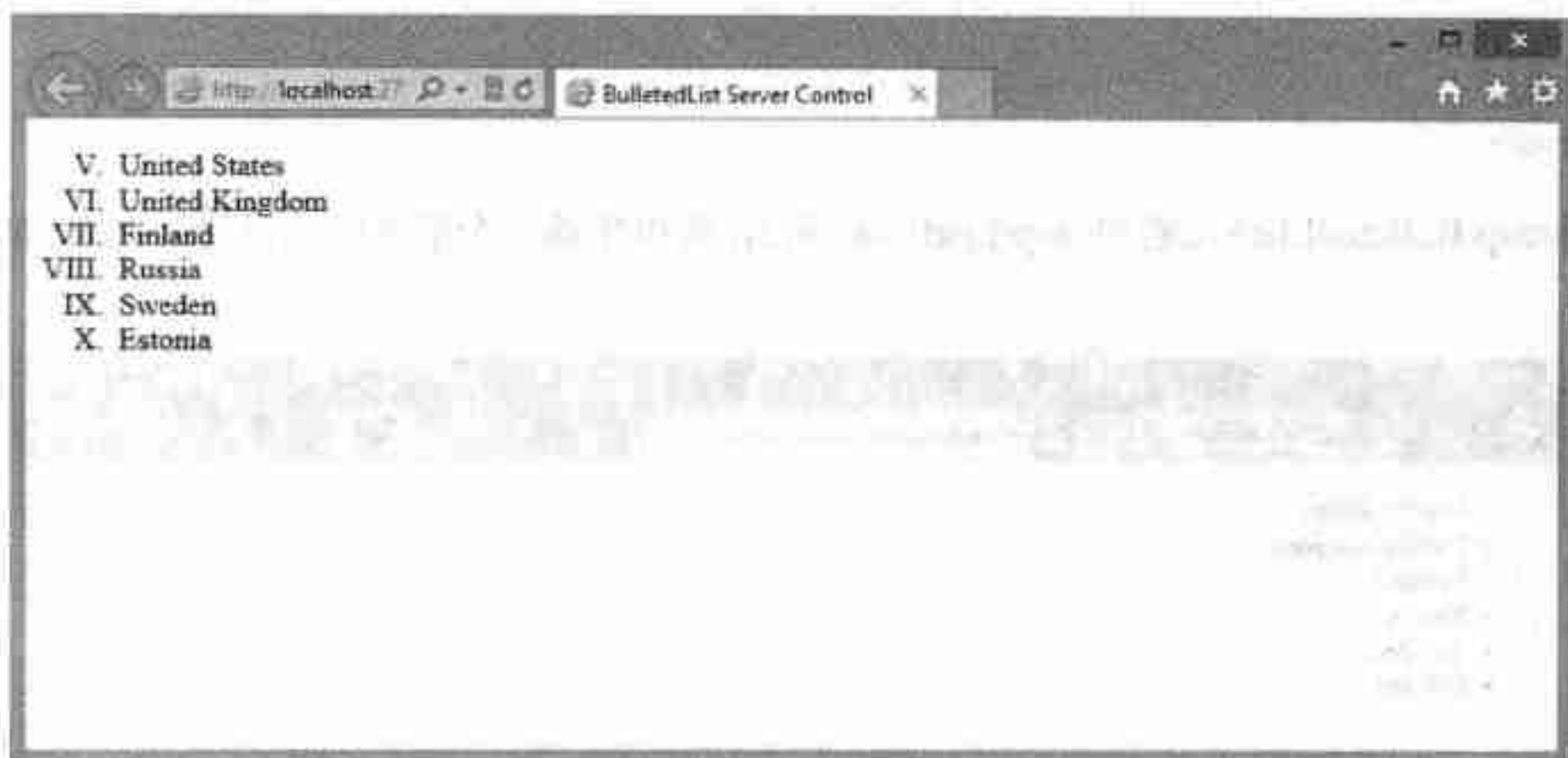


图 5-33

为了把图像用作项目符号，应使用 BulletedList 控件中的 CustomImage 设置。还必须以如下方式使用 BulletImageUrl 属性：

```
<asp:BulletedList ID="Bulletedlist1" BulletStyle="CustomImage"
    BulletImageUrl="~/search.jpg" runat="server">
```

图 5-34 是图像项目符号的一个例子。



图 5-34

BulletedList 控件支持 DisplayMode 属性，它的值可以是 Text、HyperLink 和 LinkButton。其中 Text 是默认值，前面的例子都用到了它。使用 Text 值表示项目符号列表中的项应设置为文本。使用 HyperLink 值表示每一项都会变成超链接——单击超链接的用户会被重定向到另一个页面，该页面由<asp:ListItem>控件的 Value 属性指定。使用 LinkButton 值会把项目符号列表中的每一项都变成回送至原来页面的超链接，利用它可以提取出终端用户选择的项，如程序清单 5-30 所示。

程序清单 5-30 使用 DisplayMode 属性的 LinkButton 值

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
```

```

<script runat="server">
    protected void BulletedList1_Click(object sender, BulletedListEventArgs e)
    {
        Label1.Text = "The index of item you selected: " + e.Index +
            "<br>The value of the item selected: " +
            BulletedList1.Items[e.Index].Text;
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>BulletedList Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:BulletedList ID="BulletedList1" runat="server"
            OnClick="BulletedList1_Click" DisplayMode="LinkButton">
            <asp:ListItem>United States</asp:ListItem>
            <asp:ListItem>United Kingdom</asp:ListItem>
            <asp:ListItem>Finland</asp:ListItem>
            <asp:ListItem>Russia</asp:ListItem>
            <asp:ListItem>Sweden</asp:ListItem>
            <asp:ListItem>Estonia</asp:ListItem>
        </asp:BulletedList>
        <asp:Label ID="Label1" runat="server">
        </asp:Label>
    </form>
</body>
</html>

```

在这个例子中,将 `DisplayMode` 属性的值设置为 `LinkButton`,并使用 `OnClick` 属性指向 `BulletedList1_Click` 事件。`BulletedList1_Click` 使用的是只显示 `Index` 属性的 `BulletedListEventArgs` 对象,从而可以确定所选项的索引号。

使用 `Items` 属性可以直接访问所选项的 `Text` 值,也可以使用该属性填充 `ListItem` 对象,具体方法如下:

```
ListItem blSelectedValue = BulletedList1.Items[e.Index];
```

前面介绍了如何创建项目符号列表,其中的各项以声明方式放在代码中。现在查看如何通过存储在数据存储中的项动态地创建项目符号列表。下面的例子说明了如何使用 `BulletedList` 控件绑定来自数据存储的结果,结果中的所有信息都来自于 XML 文件。

第一步是创建如程序清单 5-31 所示的 XML 文件。

程序清单 5-31 FilmChoices.xml

```

<?xml version="1.0" encoding="utf-8" ?>
<Books>
    <Book
        Title="Professional ASP.NET 4.5"
        Year="2013"
        Price="$59.99" />
    <Book
        Title="Beginning ASP.NET 4.5"

```



```

        Year="2012"
        Price="$59.99" />
    <Book
        Title="Beginning Visual C#"
        Year="2012"
        Director="$59.99" />
</Books>

```

为了使用 Title 属性将 Listing05-31 文件中的内容填充到 BulletedList 服务器控件中, 需要使用 XmlDataSource 控件访问该文件, 如程序清单 5-32 所示。

程序清单 5-32 动态地填充 BulletedList 服务器控件

```

<%@ Page Language="C#" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>BulletedList Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:BulletedList ID="BulletedList1" runat="server"
            DataSourceID="XmlDataSource1" DataTextField="Title">
        </asp:BulletedList>
        <asp:XmlDataSource ID="XmlDataSource1" runat="server"
            DataFile="~/Listing05-31.xml" XPath="Books/Book"></asp:XmlDataSource>
    </form>
</body>
</html>

```

在这个例子中, 使用 DataSourceID 属性指向 XmlDataSource 控件(就像使用可以绑定到数据源控件的其他任意控件一样)。连接到数据源控件之后, 就使用 DataTextField 属性指向 Title 属性。连接两个服务器控件并运行页面之后, 就会得到一个项目符号列表, 它完全是通过 XML 文件的内容生成的。图 5-35 显示了结果。

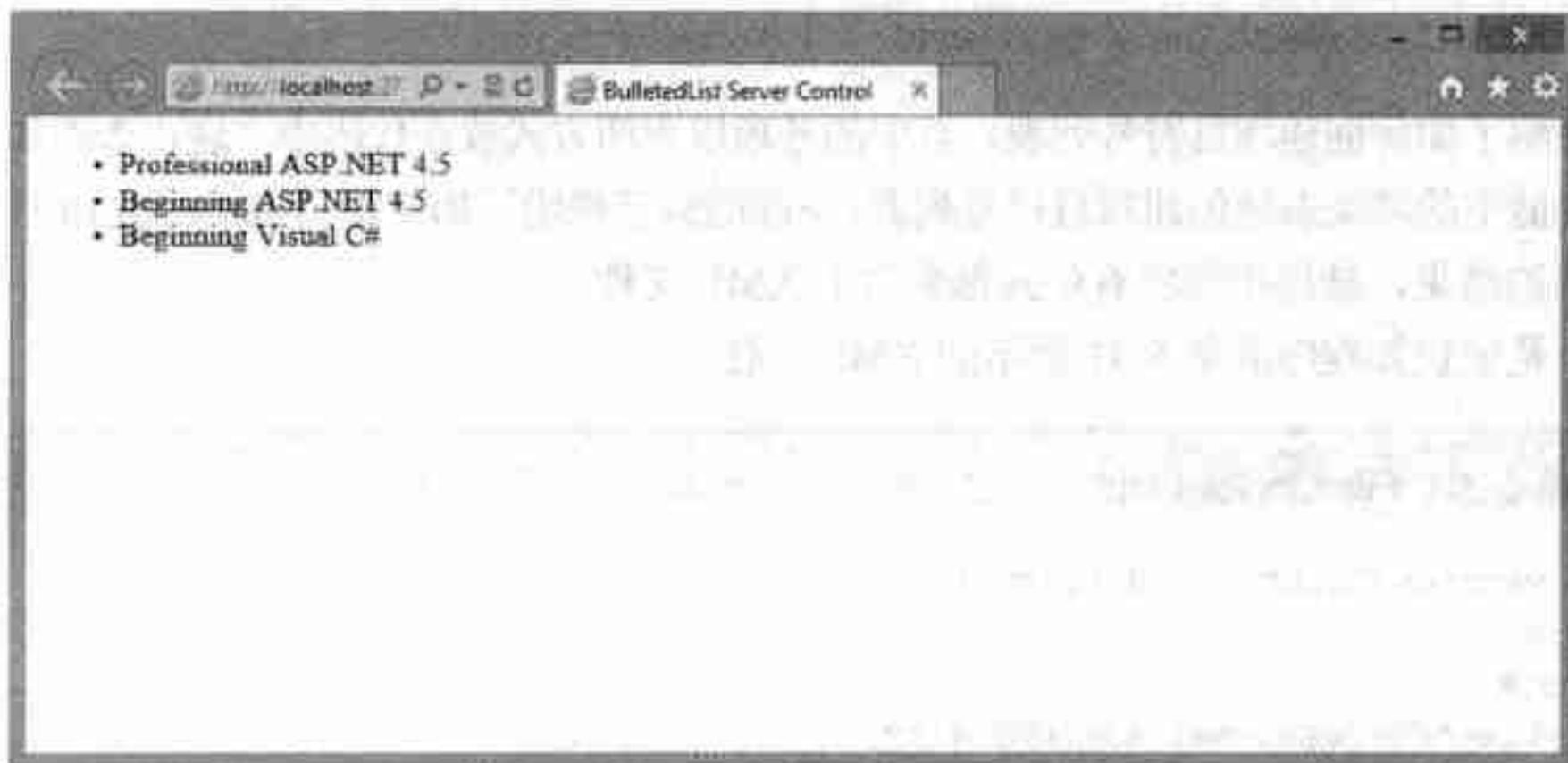


图 5-35



XmlDataSource 服务器控件实际上存在一些限制。在上面的例子中,该控件可以绑定到 BulletedList 服务器控件,这只是因为 Title 值是 XML 特性,而不是子元素。XmlDataSource 控件在进行数据绑定时只能把 XML 特性用作属性。如果要使用子元素,就必须使用 XmlDataSource 控件的 TransformFile 属性进行 XSLT 转换,将元素转换为特性。

5.24 HiddenField 服务器控件

多年以来,开发人员在 Web 页面上一直使用隐藏字段来进行状态管理。<input type="hidden">元素非常适合存储没有安全上下文的项。这些项只是数据点的占位符,我们要在页面上存储这些数据,而不是使用 Session 对象或把数据与页面的视图状态混合起来。视图状态(详见第7章)是在页面上存储信息的另一种有效方式,但许多开发人员都会禁用这个功能,以避免视图状态被破坏或降低页面性能。

隐藏字段只要放置在 Web 页面上,就不能以任何方式在浏览器上显示,但如果终端用户查看 HTML 页面的源代码,就可以看到隐藏字段。

程序清单 5-33 所示的例子使用 HiddenField 服务器控件保存 GUID,当终端用户在应用程序中导航时,只要附带该控件的值,GUID 就可以在各个页面上使用。

程序清单 5-33 使用 HiddenField 服务器控件

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Page_Load(Object sender, EventArgs e)
    {
        HiddenField1.Value = Guid.NewGuid().ToString();
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>HiddenField Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:HiddenField ID="HiddenField1" runat="Server" />
    </form>
</body>
</html>
```

在这个例子中,Page_Load 事件使用 GUID 来填充 HiddenField1 控件。查看所创建的空白 HTML 页面的源代码,就可以找到该隐藏字段及其值。结果如下所示(当然,GUID 的值会有所不同):

```
<input type="hidden" name="HiddenField1" id="HiddenField1"
value="d447f6b9-bfbe-4eaa-859a-fb4136e95cd2" />
```

在回送页面时, ASP.NET 可以检测出 HiddenField 服务器控件自上次回送页面以来是否改变了其值。这样就可以使用客户端脚本改变 HiddenField 的值, 并在页面事件中进行其他修改。

HiddenField 服务器控件有一个 ValueChanged 事件, 在值发生改变时就会触发这个事件:

```
protected void HiddenField1_ValueChanged(object sender, EventArgs e)
{
    // Handle event here
}
```

如果 HiddenField 服务器控件的值从上次绘制页面以来发生了改变, 那么在将 ASP.NET 页面回送给服务器时会触发 ValueChanged 事件。如果该控件的值没有变化, 就永远不会触发该事件。因此, 该事件方法可用于对 HiddenField 控件执行任意修改, 例如把值记录到数据库中, 或者修改用户配置文件中的值。

5.25 FileUpload 服务器控件

在早期版本的 ASP.NET 中, 可以使用 FileUpload 服务器控件上传文件。这个控件会把 `<input type="file">` 元素放在 Web 页面上, 让终端用户把文件上传到服务器。想要使用文件, 就必须对页面进行两处修改。例如, 必须给页面的 `<form>` 元素添加 `enctype="multipart/form-data"`。

自从 ASP.NET 2.0 以来, 就可以使用 FileUpload 服务器控件简化将文件上传给服务器的过程。在给页面添加上传文件的功能时, 只需包含新控件 `<asp:FileUpload>`, ASP.NET 就会完成其他工作, 包括将 `enctype` 属性添加到页面的 `<form>` 元素中。

5.25.1 使用 FileUpload 控件上传文件

把文件上传到服务器之后, 还可以提取上传文件的属性, 把它们显示给终端用户, 或者在页面的隐藏代码中使用这些值。程序清单 5-34 是一个使用 FileUpload 控件的例子。该页面包含一个 FileUpload 控件、一个 Button 控件和一个 Label 控件。

程序清单 5-34 使用 FileUpload 控件上传文件

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        if(FileUpload1.HasFile)
        {
            try
            {
                FileUpload1.SaveAs("C:\\Uploads\\" + FileUpload1.FileName);
                Label1.Text = "File name: " +
                    FileUpload1.PostedFile.FileName + "<br>" +
                    FileUpload1.PostedFile.ContentLength + " kb<br>" +
                    "Content type: " +
                    FileUpload1.PostedFile.ContentType;
            }
            catch(Exception ex)
            {
            }
        }
    }
}
```



```

        Label1.Text = "ERROR: " + ex.Message.ToString();
    }
    else
    {
        Label1.Text = "You have not specified a file.";
    }
}
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>FileUpload Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:FileUpload ID="FileUpload1" runat="server" />
        <p>
            <asp:Button ID="Button1" runat="server" Text="Upload"
                OnClick="Button1_Click" />
        </p>
        <p>
            <asp:Label ID="Label1" runat="server"></asp:Label>
        </p>
    </form>
</body>
</html>

```

从这个例子可以看出，整个过程非常简单。页面上的一个按钮启动上传过程。FileUpload 控件本身并不启动上传过程，而是必须通过另一个事件(如 Button_Click)来启动。

编译和运行这个页面时，在生成的页面源代码中有几个地方需要注意。生成的源代码如下所示：

```

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>
    FileUpload Server Control
</title></head>
<body>
    <form method="post" action="Listing05-34.aspx" id="form1"
        enctype="multipart/form-data">
    <div class="aspNetHidden">
    <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="YkgBQVHMwxdJnFcwbHY
G5i9hOlxcWnVeP+bjjdfpcRzkW30vLjxP75xY0wPaGlfa6YBcCAKm7gdi/7tonGDEXufgpgIGwHE4yb0t6
jWepmDlDxH9i0/G69Pmr76WXQjP2coa07Uxyh9sbhqERlzcqg==" />
    </div>
    <div class="aspNetHidden">
    <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
        value="gNlwDleVXBy2QmFixsfWZQGK7o0wSftXvY9Ff6ULdXYbqETya2cdZ5Ta2AmMcsvGIR
BNBJ3tCi8FgYsxjwfTlGGECrta3LiwFjipySbNhzhzhtwnY8sGfd9ygP96s9/aSE" />
    </div>
    <input type="file" name="FileUpload1" id="FileUpload1" />
    <p>
        <input type="submit" name="Button1" value="Upload" id="Button1" />
    </p>
    <p>
        <span id="Label1"></span>
    </p>

```

```

        </p>
    </form>
</body>
</html>

```

首先要注意，因为 FileUpload 控件在页面上，所以由 ASP.NET 4.5 修改了页面的 <form> 元素，添加了适当的 enctype 属性。另外，FileUpload 控件已转换为 HTML 元素 <input type="file">。

文件上传后，首先(在文件的 Button1_Click 事件处理程序中)要检查文件引用是否确实放在 <input type="file"> 元素中。如果已指定文件，就应尝试使用 FileUpload 控件的 SaveAs() 方法将所引用的文件上传给服务器。SaveAs() 方法带一个 String 参数，该参数包含保存文件的位置。在程序清单 5-34 使用的 String 参数中，可以看到该文件保存到 C 盘的 Uploads 文件夹中。

PostedFile.FileName 属性用于给已保存的文件指定与源文件相同的名称。如果要给文件指定其他名称，只需以如下方式使用 SaveAs() 方法：

```
FileUpload1.SaveAs("C:\\Uploads\\UploadedFile.txt")
```

还可以在文件的名称中指定上传的时间：

```
FileUpload1.SaveAs("C:\\Uploads\\" & System.DateTime.Now.ToFileTimeUtc() & ".txt")
```

成功完成文件的上传后，就使用上传文件的元数据填充页面上的 Label 控件。在上面的例子中，已获取文件的名称、大小和内容类型，并在页面上显示给终端用户。文件上传给服务器后，生成的页面如图 5-36 所示。

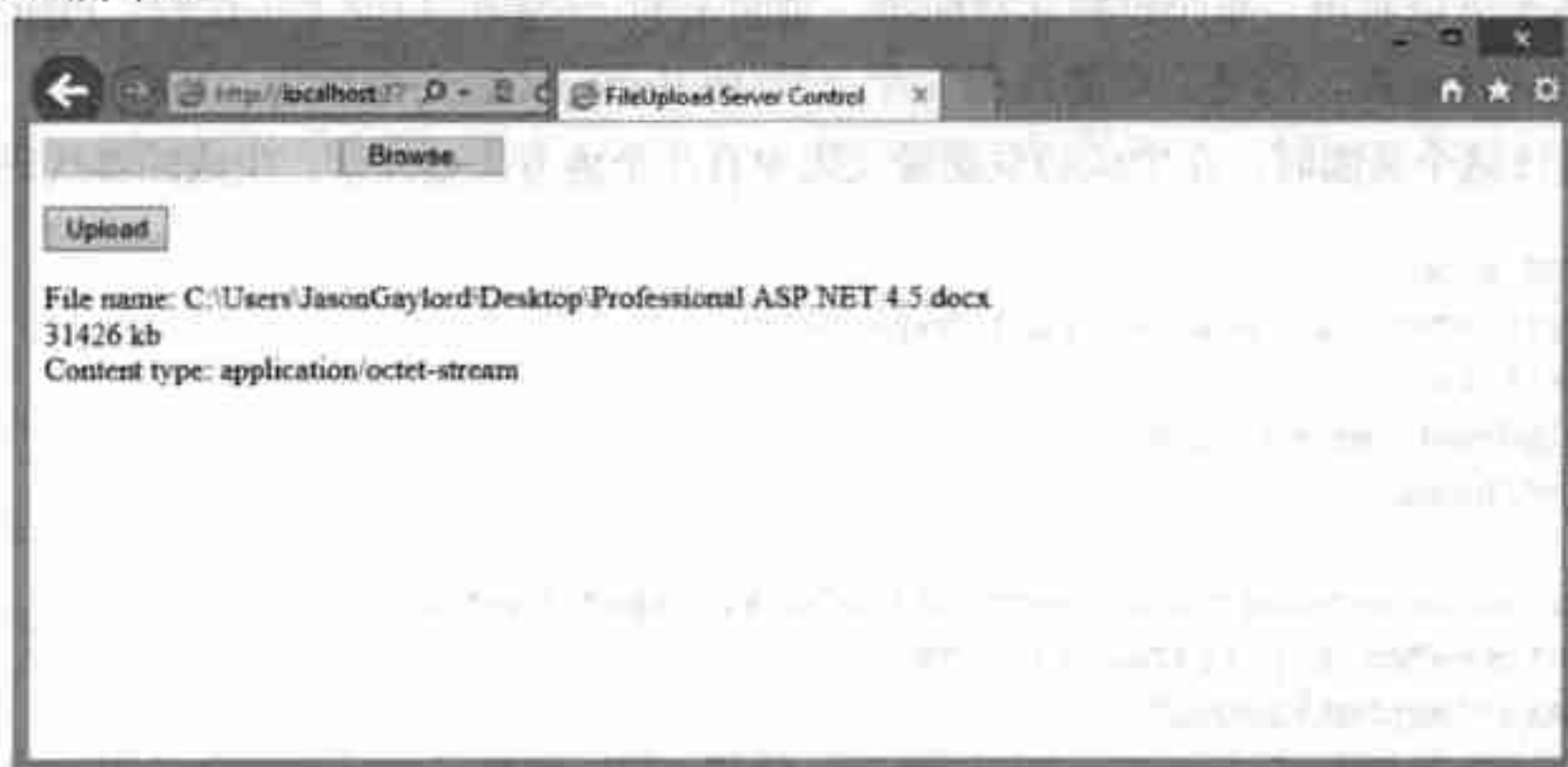


图 5-36

给另一台服务器上传文件是很容易出错的操作。在代码中使用正确的异常处理来上传文件是非常重要的，这就是例子中使用 Try Catch 语句来上传文件的原因。

5.25.2 给上传文件指定正确的 ASP.NET 权限

终端用户在应用程序中通过 FileUpload 控件给 Web 服务器上传文件时可能会出错，这可能是由于 ASP.NET 使用的账户不能写入服务器上的目标文件夹。如果 ASP.NET 不能写入指定的文件夹，可以使用该文件夹的属性启用该功能。

首先，右击接收上传的 ASP.NET 文件的文件夹并从提供的菜单中选择 Properties 命令，打开所选文件夹的 Properties 对话框。单击 Security 选项卡，确保列表中包含 IIS_IUSRS 选项，并且有写入

磁盘的适当权限。有了该权限后,就会得到如图 5-37 所示的结果。

如果在允许访问文件夹的用户列表中没有 IIS_IUSRS 选项,就应单击 Add 按钮,在所提供的文本区域中输入 IIS_IUSRS,如图 5-38 所示。

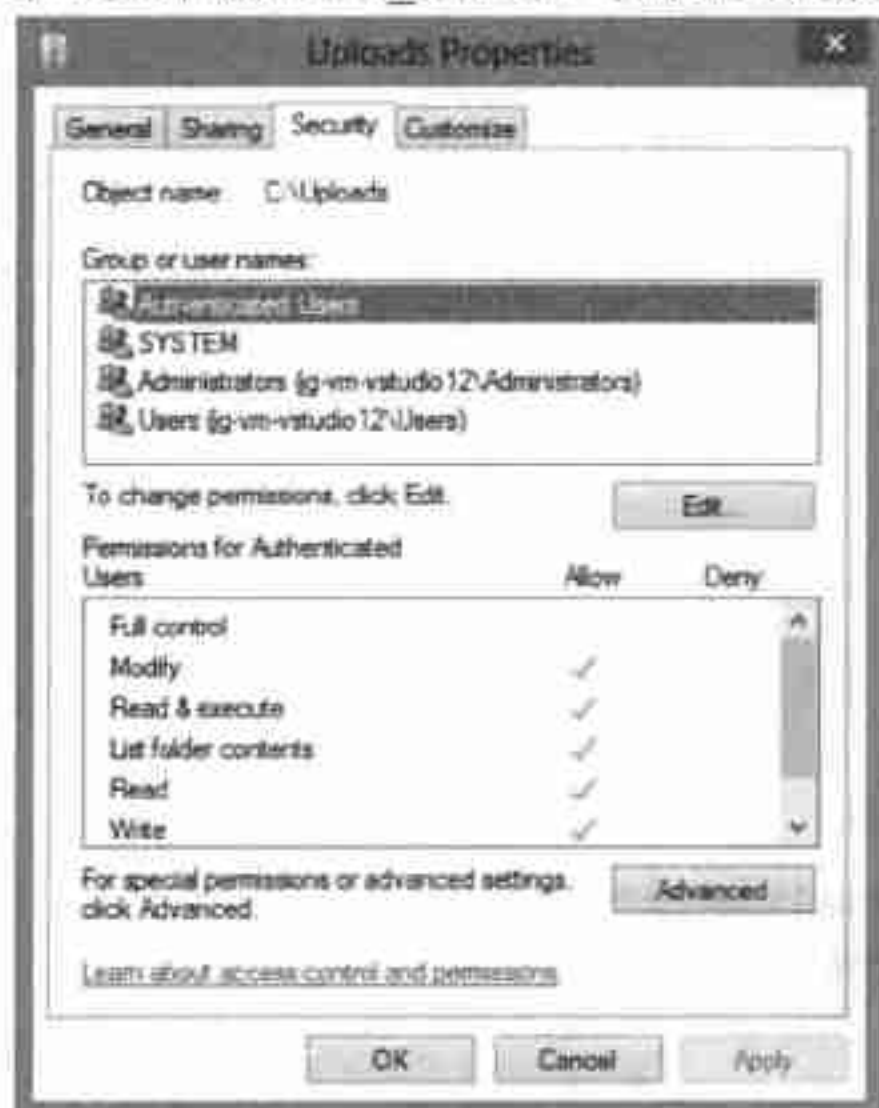


图 5-37



图 5-38

单击 OK 按钮,然后选中相应的复选框,为应用程序提供需要的权限。

5.25.3 理解文件的大小限制

终端用户可能从来都不会在应用程序的文件上传过程中遇到问题,但应注意其中的一些限制。在用户上传文件的过程中,文件的大小限制会发送给接收文件的服务器。默认的大小限制是 4MB(4096KB),如果用户试图上传大于 4096KB 的文件,传输就会失败。

文件大小限制可以保护应用程序。要禁止恶意的用户给 Web 服务器上传大量的大文件,试图占用服务器上的所有可用进程,这称为拒绝服务攻击。它会占用 Web 服务器的资源,使服务器拒绝向合法用户发送响应。

.NET 的一个优点是,它通常会提供一种绕过文件大小限制的方式。我们一般可以修改默认的文件大小设置。要修改允许的上传文件的大小,可以在 web.config 根文件(在 ASP.NET 4.5 配置文件夹 C:\WINDOWS\Microsoft.NET\Framework\v4.0.xxxxx\CONFIG 中)或应用程序的 web.config 文件中进行一些修改。

在 web.config 文件中创建节点<httpRuntime>。在这个文件中,默认允许的文件大小通过 Web 服务器允许的请求文件大小(4096KB)来指定。web.config 文件的<httpRuntime>部分如程序清单 5-35 所示。

程序清单 5-35 在 web.config 文件中改变文件大小的限制

```
<httpRuntime
  asyncPreloadMode="None"
  executionTimeout="110"
  maxRequestLength="4096"
```



```

requestLengthDiskThreshold="80"
useFullyQualifiedRedirectUrl="false"
minFreeThreads="8"
minLocalRequestFreeThreads="4"
appRequestQueueLimit="5000"
enableKernelOutputCache="true"
enableVersionHeader="true"
requireRootedSaveAsPath="true"
enable="true"
defaultRegexMatchTimeout="00:00:00"
shutdownTimeout="90"
delayNotificationTimeout="0"
waitChangeNotification="0"
maxWaitChangeNotification="0"
enableHeaderChecking="true"
sendCacheControlHeader="true"
apartmentThreading="false"
encoderType="System.Web.Util.HttpEncoder"
requestValidationMode="4.0"
requestValidationType="System.Web.Util.RequestValidator"
requestPathInvalidCharacters="&lt;;&gt;;*,%&amp;;,:,\,?"
maxUrlLength="260"
maxQueryStringLength="2048"
relaxedUrlToFileSystemMapping="false"
allowDynamicModuleRegistration="true"
fcgMode="NotSet" />

```

在 web.config 文件的<httpRuntime>部分可以进行许多操作,其中有两个属性 maxRequestLength 和 executionTimeout 非常值得注意。

maxRequestLength 属性指定发送给 Web 服务器的请求的大小。在上传文件时,文件包含在请求中。改变这个属性的值,就可以改变允许上传的文件大小。其值以 KB 为单位。要允许上传的文件大于默认的 4MB,应改变 maxRequestLength 属性,如下所示:

```
maxRequestLength="10240"
```

这个例子把 maxRequestLength 属性的值改为 10240KB(10MB)。有了这个设置后,终端用户就可以给服务器上传 10MB 的文件。改变 maxRequestLength 属性时要注意 executionTimeout 属性的设置。这个属性设置在 ASP.NET 停止请求之前(无论请求是否完成)试图给服务器发出请求的时间(以秒为单位)。默认设置为 90 秒。如果超出这个时间限制,终端用户就会在浏览器上收到超时错误通知。如果可以进行更大的请求,注意这些请求比较小请求的执行时间长。如果增大 maxRequestLength 属性,就应检查是否也增大 executionTimeout 属性。



如果处理的是较小的文件,那么最好减少 maxRequestLength 属性的值,以减少发送给 Web 服务器的请求的大小。这将有助于保护应用程序不受到拒绝服务攻击。

在 web.config 文件中进行的这些修改会把设置应用于服务器上的所有应用程序。如果只想把这些设置应用于当前处理的应用程序,应把<httpRuntime>节点添加到应用程序的 web.config 文件中,覆盖 web.config 根文件中的相应设置。<httpRuntime>节点一定要位于配置文件的<system.web>

节点之间。

5.25.4 从同一个页面上上传多个文件

前面介绍了一些给服务器上传文件的简单例子，下面查看如何从同一个页面给服务器上传多个文件。

Microsoft .NET Framework 没有内置从同一个 ASP.NET 页面上上传多个文件的功能。但是，只要多做少量工作，就可以像以前使用 .NET 1.x 一样轻松地完成这项任务。

其关键是在 ASP.NET 页面上导入 System.IO 类，再使用 `HttpFileCollection` 类和 `Request` 对象捕获要传送的所有文件。这样就可以从一个页面上上传任意多个文件。

也可以分别简单地处理页面上的每个 `FileUpload` 控件，如程序清单 5-36 所示。

程序清单 5-36 分别处理每个 FileUpload 控件

```
if(FileUpload1.HasFile) {
    /// Handle this upload
}
if(FileUpload2.HasFile) {
    /// Handle this upload
}
```

如果使用数量有限的文件上传框，那么这种方式是有效的；但在某些情况下，可能需要同时使用 `HttpFileCollection` 类来处理文件。例如，要在 ASP.NET 页面上处理动态生成的服务器控件列表。

为了说明这一点，建立一个 ASP.NET 页面，其中包含 3 个 `FileUpload` 控件和一个 `Submit` 按钮(使用 `Button` 控件)。在用户单击 `Submit` 按钮以将文件传送给服务器后，隐藏代码就会提取文件，把它们保存到服务器的指定位置。保存好文件后，所传送的文件信息就会显示在 ASP.NET 页面上，如程序清单 5-37 所示。

程序清单 5-37 给服务器上传多个文件

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.IO" %>
<!DOCTYPE html>
<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        string filepath = "C:\\\\Uploads";
        HttpFileCollection uploadedFiles = Request.Files;
        for(int i = 0; i < uploadedFiles.Count; i++)
        {
            HttpPostedFile userPostedFile = uploadedFiles[i];
            try
            {
                if(userPostedFile.ContentLength > 0)
                {
                    Label1.Text += "<u>File #" + (i + 1) +
                        "</u><br>";
                    Label1.Text += "File Content Type: " +
                        userPostedFile.ContentType + "<br>";
                }
            }
            catch { }
        }
    }
}
```

```

        Label1.Text += "File Size: " +
            userPostedFile.ContentLength + "kb<br>";
        Label1.Text += "File Name: " +
            userPostedFile.FileName + "<br>";
        userPostedFile.SaveAs(filepath + "\\" +
            Path.GetFileName(userPostedFile.FileName));
        Label1.Text += "Location where saved: " +
            filepath + "\\" +
            Path.GetFileName(userPostedFile.FileName) +
            "<p>";
    }
}
catch (Exception Ex)
{
    Label1.Text += "Error: <br>" + Ex.Message;
}
}
}
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>FileUpload Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <p>
            <asp:FileUpload ID="FileUpload1" runat="server" />
        </p>
        <p>
            <asp:FileUpload ID="FileUpload2" runat="server" />
        </p>
        <p>
            <asp:FileUpload ID="FileUpload3" runat="server" />
        </p>
        <p>
            <asp:Button ID="Button1" runat="server" Text="Upload"
                OnClick="Button1_Click" />
        </p>
        <p>
            <asp:Label ID="Label1" runat="server"></asp:Label>
        </p>
    </form>
</body>
</html>

```

这个 ASP.NET 页面允许终端用户至多选择 3 个文件,然后单击 Upload 按钮,触发 Button1_Click 事件。使用 HttpFileCollection 类和 Request.Files 属性可以控制从页面中上传的所有文件。当文件处于这种状态时,可以对它们进行任何处理,在本例中是查看文件的属性,并将它们写到屏幕上。最后,把文件保存到服务器根目录下的 Uploads 文件夹中。这个示例的运行结果如图 5-39 所示。

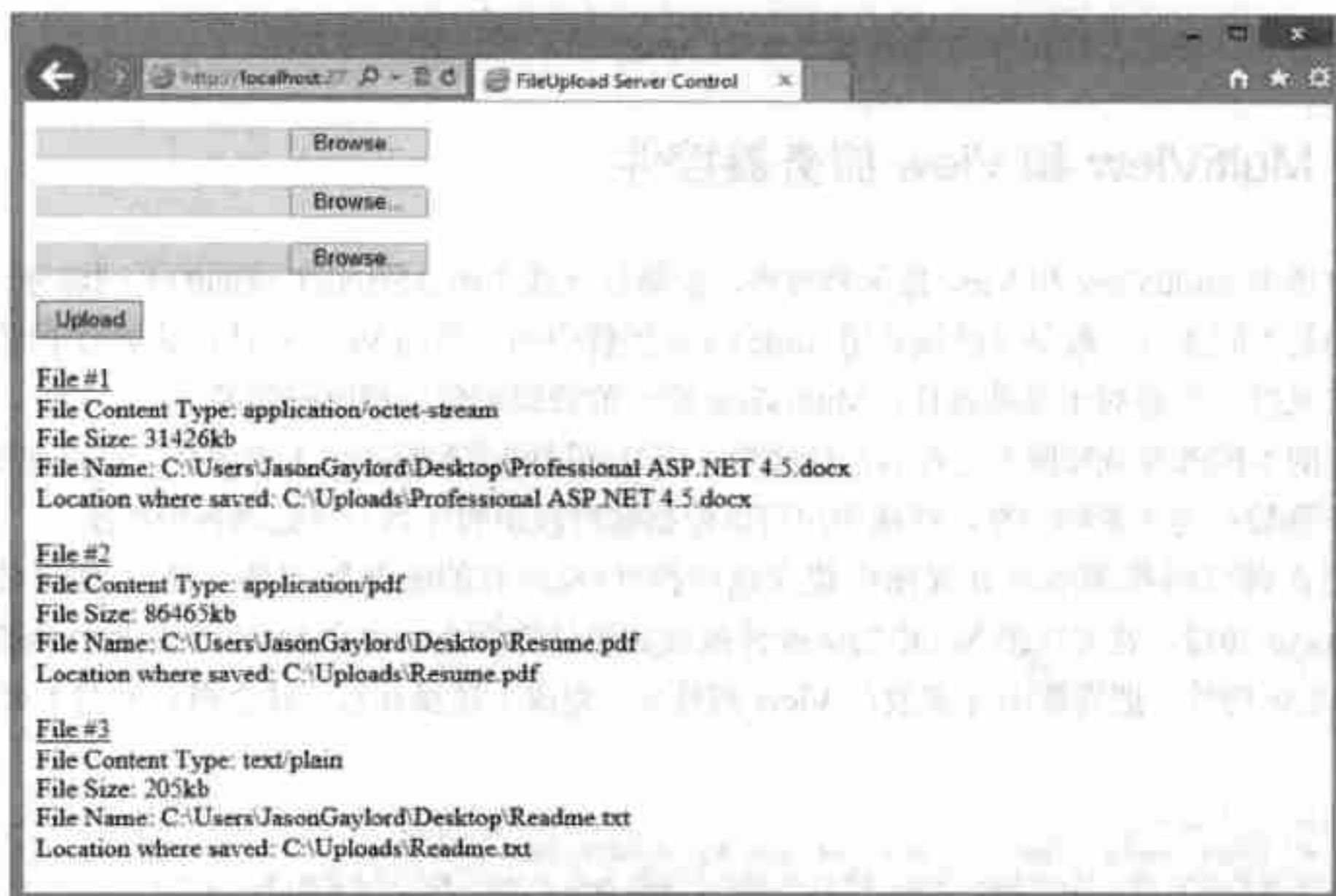


图 5-39

5.25.5 把上传的文件放在 Stream 对象中

FileUpload 控件的一个优点是,它不仅可以把文件保存到磁盘上,还可以把文件的内容放在 Stream 对象中。这需要使用 FileContent 属性,如程序清单 5-38 所示。

程序清单 5-38 把文件的内容上传到 Stream 对象中

```
Stream myStream;
myStream = FileUpload1.FileContent;
```

在这个简短示例中,创建了一个 Stream 对象。然后使用 FileUpload 控件的 FileContent 属性,把上传文件的内容放在该对象中。这是可以实现的操作,因为 FileContent 属性返回一个 Stream 对象。

5.25.6 把文件内容从 Stream 对象移到 Byte 数组中

由于能够把文件内容移动到 Stream 对象中,因此将文件内容移动到 Byte 数组中也是相当简单的(这对于把文件内容放在某种数据库中这样的操作很有用)。为此,首先把文件内容移动到 MemoryStream 对象中,再把该对象转换为需要的 Byte 数组对象。程序清单 5-39 展示了这个过程。

程序清单 5-39 把文件内容上传到 Byte 数组中

```
MemoryStream myStream;
myStream = (MemoryStream)FileUpload1.FileContent;
Byte[] myByteArray = new Byte[FileUpload1.PostedFile.ContentLength];
myByteArray = myStream.ToArray();
```

在这个例子中,创建了 Byte 数组的一个实例和一个 MemoryStream 对象。首先,与前面一样使用 FileUpload 控件的 FileContent 属性创建 MemoryStream 对象。然后可以很简单地使用 MemoryStream 对象的 ToArray()方法填充 myByteArray()实例。将文件放到 Byte 数组中之后,就可以根据需要处理

文件的内容。这是把文件保存为 SQL Server 的 Image 格式的一种常用方法。

5.26 MultiView 和 View 服务器控件

同时使用 MultiView 和 View 服务器控件,能够打开或关闭 ASP.NET 页面的不同部分。打开或关闭页面的不同部分,就是激活或禁用 MultiView 控件中的一系列 View 控件,这类似于改变 Panel 控件的可见性。但是对于某些操作,MultiView 控件的管理和使用都比较容易。

页面的不同部分或视图不会在客户端改变,而是通过回送到服务器上来改变。可以在每个视图中放置任意数量的元素和控件,终端用户可以根据赋给视图的序列号来处理各个视图。

可以在源代码视图或设计视图中建立这些控件(与所有的服务器控件一样)。如果使用的是 Visual Studio 2012,就可以把 MultiView 控件拖放到设计界面上,再在 MultiView 控件内拖放任意数量的 View 控件。把需要的元素放在 View 控件内。完成上述操作后,就会得到如图 5-40 所示的视图。



图 5-40

还可以直接在代码中创建自己的控件,如程序清单 5-40 所示。

程序清单 5-40 使用 MultiView 和 View 服务器控件

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            MultiView1.ActiveViewIndex = 0;
        }
    }
}
```

```

    }
    void NextView(object sender, EventArgs e)
    {
        MultiView1.ActiveViewIndex += 1;
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>MultiView Server Control</title>
</head>
<body>
    <form id="form2" runat="server">
        <asp:MultiView ID="MultiView1" runat="server">
            <asp:View ID="View1" runat="server">
                <em>Making a Great Book</em><br />
                <br />
                Surround yourself with talented authors.<br />
                <br />
                <asp:Button ID="Button1" runat="server" Text="Next Step"
                    OnClick="NextView" />
            </asp:View>
            <asp:View ID="View2" runat="server">
                <em>Making a Great Book</em><br />
                <br />
                Write content you are passionate about.<br />
                <br />
                <asp:Button ID="Button2" runat="server" Text="Next Step"
                    OnClick="NextView" />
            </asp:View>
            <asp:View ID="View3" runat="server">
                <em>Making a Great Book</em><br />
                <br />
                Have a bunch of smart technical editors review your work.<br />
                <br />
                <asp:Button ID="Button3" runat="server" Text="Next Step"
                    OnClick="NextView" />
            </asp:View>
            <asp:View ID="View4" runat="server">
                <em>Making a Great Book</em><br />
                <br />
                Release the book to publishing!
            </asp:View>
        </asp:MultiView>
    </form>
</body>
</html>

```

这个例子在 **MultiView** 控件中显示了 4 个视图。使用一个 **<asp:View>** 服务器控件构造每个视图，该服务器控件需要 **ID** 和 **runat** 属性。在 **MultiView** 控件的前 3 个视图(**View1**、**View2** 和 **View3**)中各添加一个按钮。这 3 个按钮指向一个服务器端事件，该事件会触发 **MultiView** 控件，进入视图系列中的下一个视图。

在单击这几个按钮中的任意一个之前，必须给 **MultiView** 控件的 **ActiveViewIndex** 属性指定一个值。**ActiveViewIndex** 属性表示应显示的视图，其值默认为 -1，这就说明在生成页面时不显示任何视

图。为了在绘制页面时显示第一个视图，把 `ActiveViewIndex` 属性设置为 0，这是第一个视图，因为其索引是基于 0 的。因此，程序清单 5-40 中的代码首先检查页面是不是从服务器回送过来的，如果不是，就将 `ActiveViewIndex` 属性分配给第一个 `View` 控件。

`MultiView` 控件中的每个按钮都会触发 `NextView` 方法。`NextView` 方法会给 `ActiveViewIndex` 值加 1，这样就会显示视图系列中的下一个视图，直到显示最后一个视图为止。视图系列如图 5-41 所示。



图 5-41

除了在第 1、第 2 和第 3 个视图添加 `Next Step` 按钮外，还可以在第 2、第 3 和第 4 个视图中各放置一个按钮，让用户反向导航视图。为此，在后 3 个视图中创建 3 个 `Previous Step` 按钮，在它们的 `OnClick` 事件中使其指向下面的方法：

```
void PreviousView(object sender, EventArgs e)
{
    MultiView1.ActiveViewIndex -= 1;
}
```

在上面的代码中，`PreviousView` 方法将 `ActiveViewIndex` 值减 1，从而显示视图系列中的前一个视图。

另一个选项可以给 `MultiView` 控件增加一点儿趣味性：通过给 `Label` 控件添加步骤计数器，从而显示终端用户当前在系列中执行的步骤。在 `Page_PreRender` 事件中添加如下代码：

```
Label1.Text = "Step " + (MultiView1.ActiveViewIndex + 1).ToString() +
    " of " + MultiView1.Views.Count.ToString();
```

在使用 MultiView 控件时, 终端用户会在第一个视图中看到 Step 1 of 3, 在下一个视图中则变成 Step 2 of 3, 依此类推。

5.27 Wizard 服务器控件

与 MultiView 控件一样, Wizard 服务器控件可以建立一系列显示给终端用户的步骤。Web 页面都是用于显示信息或收集信息的, 在许多情况下, 我们不希望一次显示所有的信息, 也不可能从终端用户处一次收集到所有的信息。

在构建多步骤的过程并且该过程在各个步骤中包含一定的逻辑时, 应使用 Wizard 控件来管理整个过程。第一次使用 Wizard 控件时, 注意它允许的定制程度比 MultiView 控件高。

Wizard 控件的最简单形式只是一个 <asp:Wizard> 元素, 其中可以包含任意数量的 <asp:WizardStep> 元素。程序清单 5-41 创建了一个包含 3 个步骤的 Wizard 控件。

程序清单 5-41 简单的 Wizard 控件

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Wizard server control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Wizard ID="Wizard1" runat="server" DisplaySideBar="True"
            ActiveStepIndex="0">
            <WizardSteps>
                <asp:WizardStep ID="WizardStep1" runat="server" Title="Step 1">
                    This is the first step.
                </asp:WizardStep>
                <asp:WizardStep ID="WizardStep2" runat="server" Title="Step 2">
                    This is the second step.
                </asp:WizardStep>
                <asp:WizardStep ID="WizardStep3" runat="server" Title="Step 3">
                    This is the third and final step.
                </asp:WizardStep>
            </WizardSteps>
        </asp:Wizard>
    </form>
</body>
</html>
```

在这个例子中, 使用 <asp:WizardSteps> 控件定义了 3 个步骤。每一步都包含内容——本例中是简单的文本, 但可以在这些步骤中放入任意内容, 例如其他 Web 服务器控件甚至用户控件。定义 WizardSteps 的顺序完全取决于它们在 <WizardSteps> 元素中的显示顺序。

<asp:Wizard> 元素本身包含两个重要的属性。第一个属性是 DisplaySideBar, 在这个例子中, 它

设置为默认值 True，表示所显示控件中的边栏导航系统允许终端用户快速导航到该过程的其他步骤。Wizard 控件的另一个属性 `ActiveStepIndex` 定义了向导的第一步，在本例中是步骤 0。

示例 Wizard 控件的 3 个步骤如图 5-42 所示。

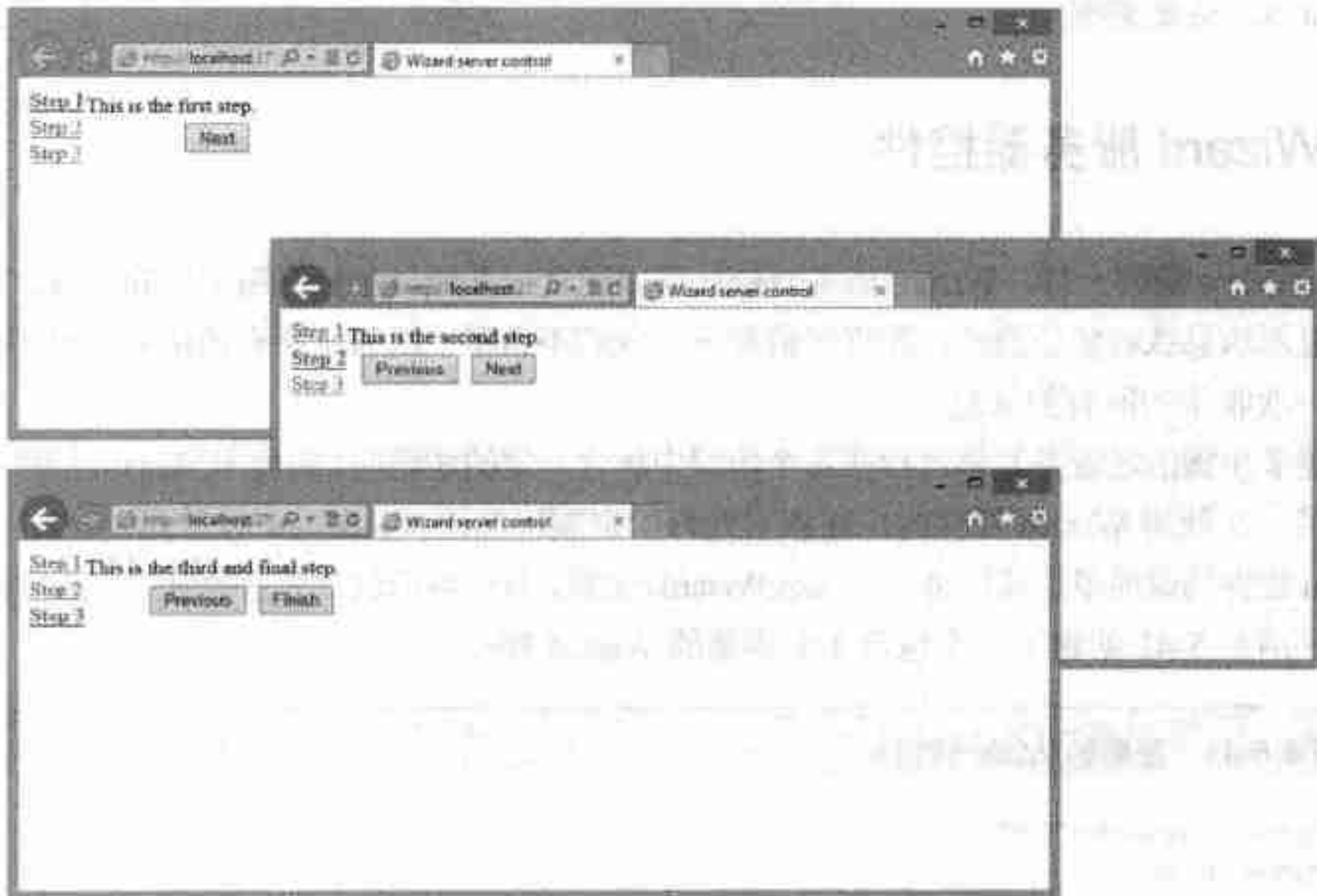


图 5-42

使用边栏导航可以轻松地访问已定义的步骤。Wizard 控件会在该过程的各个步骤中添加适当的按钮。第一步只添加 Next 按钮，中间的步骤会添加 Previous 和 Next 按钮，最后一步添加 Previous 和 Finish 按钮。用户可以使用边栏导航或每一步的按钮浏览这些步骤。还可以使用非常多的方式定制 Wizard 控件，从而提醒自己想起 ASP.NET 中的其他 Web 服务器控件，例如 Calendar 控件。由于有这么多种方式，因此本章只介绍几种基础方式，这些方式在我们构建的一些 Wizard 控件中比较常用。

5.27.1 定制边栏导航

图 5-43 所示示例中的步骤定义为 Step 1、Step 2 和 Step 3。根据给 Wizard 控件中每个 `<asp:WizardStep>` 元素指定的 Title 属性值创建链接：

```
<asp:WizardStep ID="WizardStep1" runat="server" Title="Step 1">
    This is the first step.
</asp:WizardStep>
```

在设计视图中创建的每个向导步骤都默认标记为 Step X(其中 X 是序列号)。很容易改变每个向导步骤的 Title 属性值，以定义最适合自己的步骤。在图 5-43 显示的 Wizard 控件中，边栏导航使用了重命名的标题。



图 5-43

5.27.2 AllowReturn 属性

定制 Wizard 控件的边栏导航时, 另一个有趣的属性是 AllowReturn。在向导步骤中把这个属性设置为 False, 就可以在终端用户查看该步骤后禁止他们再返回到这一步。终端用户不能向后导航到包含该属性的任何已查看过的步骤, 但可以返回到不包含该属性或该属性设置为 True 的任意步骤:

```
<asp:WizardStep ID="WizardStep1" runat="server" AllowReturn="False"
    Title="The very first step">This is the first step.</asp:WizardStep>
```

5.27.3 使用 StepType 属性

<asp:WizardStep>元素中另一个有用的属性是 StepType。StepType 属性定义了步骤中使用的按钮结构。默认情况下, Wizard 控件只在第一步中放置 Next 按钮, 它认为该步骤不需要 Previous 按钮。但是, 在中间步骤中使用 Next 和 Previous 按钮, 在最后一步中使用 Previous 和 Finish 按钮。Wizard 控件以这种方式绘制按钮, 因为在默认情况下, StepType 属性设置为 Auto, 即由 Wizard 控件确定按钮的放置情况。我们可以在<asp:WizardStep>元素中设置 StepType 属性, 从而确定在哪些步骤中使用什么按钮。

除了 Auto 值之外, StepType 值还可以是 Start、Step、Finish 或 Complete。Start 值表示所定义的步骤只有一个 Next 按钮, 它只允许用户前进到系列步骤中的下一步。Step 值表示向导步骤有 Next 和 Previous 按钮。Finish 值表示这个步骤包含 Previous 和 Finish 按钮。Complete 允许给浏览 Wizard 控件各步骤的终端用户显示一些最终消息。例如在程序清单 5-42 所示的 Wizard 控件中, 当终端用户进入最后一步并单击 Finish 按钮后, 什么都不会发生, 用户仍然位于最后一个页面。可以添加最后一步, 给出结束消息, 如程序清单 5-42 所示。

程序清单 5-42 在向导的步骤集合中包含完成步骤

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Wizard server control</title>
</head>
<body>
```

```

<form id="form1" runat="server">
  <asp:Wizard ID="Wizard1" runat="server" DisplaySideBar="True"
    ActiveStepIndex="0">
    <WizardSteps>
      <asp:WizardStep ID="WizardStep1" runat="server" Title="Step 1">
        This is the first step.
      </asp:WizardStep>
      <asp:WizardStep ID="WizardStep2" runat="server" Title="Step 2">
        This is the second step.
      </asp:WizardStep>
      <asp:WizardStep ID="WizardStep3" runat="server" Title="Step 3">
        This is the third and final step.
      </asp:WizardStep>
      <asp:WizardStep ID="WizardStep4" runat="server" Title="Final Step"
        StepType="Complete">
        Thanks for working through the steps.
      </asp:WizardStep>
    </WizardSteps>
  </asp:Wizard>
</form>
</body>
</html>

```

当终端用户单击 Step 3 中的 Finish 按钮时, 就会显示最后一步(Final Step), 并且其中没有按钮。

5.27.4 给 Wizard 控件添加标题

Wizard 控件可以通过主元素<asp:Wizard>的 HeaderText 属性在该控件的顶部添加标题。程序清单 5-43 提供了一个示例。

程序清单 5-43 使用 HeaderText 属性

```

<asp:Wizard ID="Wizard1" runat="server" DisplaySideBar="True"
  ActiveStepIndex="0" HeaderText="Step by Step with the Wizard Control"
  HeaderStyle-BackColor="DarkGray" HeaderStyle-Font-Bold="true"
  HeaderStyle-Font-Size="20">
  . . .
</asp:Wizard>

```

这段代码创建了一个标题, 它显示在向导的每一步中, 结果如图 5-44 所示。



图 5-44

5.27.5 使用向导的导航系统

如前所述,可以对 Wizard 控件进行非常高程度的定制——尤其是样式的定制。可以定制该过程的各个方面,以及每个元素显示给终端用户的方式。

特别要注意可用于定制导航按钮的选项。向导中的所有步骤都默认使用 Next、Previous 和 Finish 按钮。在主元素<asp:Wizard>中,可以修改这些按钮的所有方面,包括它们的操作方式。

首先,如果查看该元素的可用属性列表,就会注意到有一个可用按钮在默认情况下是不显示的,即 Cancel 按钮。把 DisplayCancelButton 属性的值设置为 True, Cancel 按钮就会出现在为每一步创建的导航中,包括最后一步。图 5-45 显示了其中一个步骤中的 Cancel 按钮。



图 5-45

确定要在 Wizard 导航中使用的按钮后,就可以选择它们的样式。默认使用常规的按钮,但是可以使用 CancelButtonType、FinishStepButtonType、FinishStepPreviousButtonType、NextStepButtonType、PreviousStepButtonType 和 StartStepNextButtonType 属性改变按钮的样式。如果使用这些按钮类型,并希望所有的按钮样式都保持一致,就必须把每个属性都改为相同的值。这些按钮特定的元素值有 Button、Image 和 Link。Button 是默认值,表示导航系统使用的是按钮。Image 值允许使用图像按钮,Link 值可以把导航系统中的选中项改为超链接。

除了<asp:Wizard>元素的这些按钮特定的属性之外,还可以指定一个 URL,在用户单击 Cancel 或 Finish 按钮时就会被重定向到这个 URL。要使用这两个按钮中的一个重定向用户,应使用 CancelDestinationPageUrl 或 FinishDestinationPageUrl 属性,把合适的 URL 设置为目标。

最后,不一定要使用导航系统的按钮上包含的默认文本。可以使用 CancelButtonText、FinishStepButtonText、FinishStepPreviousButtonText、NextStepButtonText、PreviousStepButtonText 和 StartStepNextButtonText 属性改变每个按钮上的文本。

5.27.6 使用 Wizard 控件的事件

Wizard 控件最方便的一项功能是可以把大型窗体分解为富有逻辑的多个部分。然后,终端用户可以一步步地处理窗体的每个部分。开发人员在处理窗体的输入值时有几个选项,因为 Wizard 控件有许多不同的事件。

Wizard 控件为每个可能的步骤都定义了事件,终端用户可以在使用该控件时触发这些事件。表 5-2 描述了每个可用事件。

表 5-2

事 件	说 明
ActiveStepChanged	终端用户从某一步移动到下一步时触发。该步骤是中间一步或最后一步并不重要。这个事件只是包含一般性的步骤切换
CancelButtonClick	终端用户单击导航系统中的 Cancel 按钮时触发
FinishButtonClick	终端用户单击导航系统中的 Finish 按钮时触发
NextButtonClick	终端用户单击导航系统中的 Next 按钮时触发
PreviousButtonClick	终端用户单击导航系统中的 Previous 按钮时触发
SideBarButtonClick	终端用户单击 Wizard 控件的边栏导航中的链接时触发

使用这些事件可以创建多步骤窗体，该窗体在终端用户从某一步进入下一步时保存他们的输入信息。也可以使用 FinishButtonClick 事件在过程的最后保存每个步骤中存储的所有信息。Wizard 控件将通过页面中的视图状态保存终端用户在每一步中输入的所有信息，以便在最后一步中处理所有这些值。另外还允许终端用户返回到前面的步骤，在把这些值保存到数据存储中之前修改它们。

采用隐藏代码或内联编码的 FinishButtonClick 事件如程序清单 5-44 所示。

程序清单 5-44 FinishButtonClick 事件

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Wizard1_FinishButtonClick(object sender,
        WizardNavigationEventArgs e)
    {
        . . .
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Wizard server control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Wizard ID="Wizard1" runat="server" DisplaySideBar="True"
            ActiveStepIndex="0" OnFinishButtonClick="Wizard1_FinishButtonClick">
            <WizardSteps>
                <asp:WizardStep ID="WizardStep1" runat="server" Title="Step 1">
                    This is the first step.
                </asp:WizardStep>
                <asp:WizardStep ID="WizardStep2" runat="server" Title="Step 2">
                    This is the second step.
                </asp:WizardStep>
                <asp:WizardStep ID="WizardStep3" runat="server" Title="Step 3">
                    This is the third and final step.
                </asp:WizardStep>
            </WizardSteps>
        </asp:Wizard>
    </form>
</body>
</html>
```

```

        </asp:Wizard>
    </form>
</body>
</html>

```

应该将 `OnFinishButtonClick` 属性添加到 `<asp:Wizard>` 主元素中, 以指向 `Wizard1_FinishButtonClick` 事件。程序清单 5-45 说明了具体方法。

程序清单 5-45 <asp:Wizard>元素的改变

```

<asp:Wizard ID="Wizard1" runat="server" DisplaySideBar="True"
    ActiveStepIndex="0" OnFinishButtonClick="Wizard1_FinishButtonClick">

```

Wizard 控件是功能最强大的控件之一, 它可以为终端用户把很长的工作流分解为更容易管理的多个部分。将很长的 Web 窗体分解为各个向导步骤后, 就可以使窗体易于理解, 终端用户也会少一些畏惧。

5.27.7 使用 Wizard 控件显示窗体元素

到目前为止, 我们已学习了如何处理 Wizard 控件的每个步骤, 包括如何给过程添加步骤, 如何处理控件的样式。下面看看如何把窗体元素放到 Wizard 控件上, 从而在多步骤过程中收集终端用户的信息。该例与 Wizard 控件的前几个例子一样简单, 在每个步骤中都只使用文本。

把窗体元素放在 Wizard 步骤过程中的一个优点是, Wizard 控件会在各个步骤中保存窗体元素的所有输入内容, 以便在最后一步中保存整个窗体的结果。这也说明, 在终端用户按下 `Previous` 按钮时, 他以前输入到窗体中的数据还在窗体中, 并且可以修改。

可以建立一个注册过程, 通过一个多步骤过程来输入窗体信息。该过程的最后一步会把结果存储在选择的数据库中, 但在这个例子中, 只把结果放在页面的一个 `Label` 控件上。

程序清单 5-46 在 Wizard 控件中建立窗体

```

<asp:Wizard ID="Wizard1" runat="Server">
    <WizardSteps>
        <asp:WizardStep ID="WizardStep1" runat="server"
            Title="Provide Personal Info">
            First name:<br />
            <asp:TextBox ID="fnameTextBox" runat="server"></asp:TextBox><br />
            Last name:<br />
            <asp:TextBox ID="lnameTextBox" runat="server"></asp:TextBox><br />
            Email:<br />
            <asp:TextBox ID="emailTextBox" runat="server"></asp:TextBox><br />
        </asp:WizardStep>
        <asp:WizardStep ID="WizardStep2" runat="server"
            Title="Membership Information">
            Are you already a member of our group?<br />
            <asp:RadioButton ID="RadioButton1" runat="server" Text="Yes"
                GroupName="Member" />
            <asp:RadioButton ID="RadioButton2" runat="server" Text="No"
                GroupName="Member" />
        </asp:WizardStep>
        <asp:WizardStep ID="WizardStep3" runat="server"
            Title="Provided Information"
            StepType="Complete" OnActivate="WizardStep3_Activate">

```

```

        <asp:Label ID="Label1" runat="server" />
    </asp:WizardStep>
</WizardSteps>
</asp:Wizard>

```

这个 Wizard 控件有 3 个步骤。第 1 步要求用户输入个人信息，第 2 步要求用户输入成员信息。第 3 步包含一个 Label 控件，它会显示所有的输入信息。通过第 3 个 WizardStep 控件上的 WizardStep 对象的特定 Activate 事件来完成该操作。WizardStep3_Activate 事件的代码如程序清单 5-47 所示。

程序清单 5-47 给 WizardStep 对象添加 Activate 事件

```

<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void WizardStep3_Activate(object sender, EventArgs e)
    {
        Label1.Text = "First name: " + fnameTextBox.Text.ToString() + "<br>" +
            "Last name: " + lnameTextBox.Text.ToString() + "<br>" +
            "Email: " + emailTextBox.Text.ToString();
    }
</script>

```

终端用户进入所显示的第 3 步时，会调用程序清单 5-47 中的 WizardStep3_Activate 方法。使用第 3 个 WizardStep 控件的 OnActivate 属性，终端用户在前面几步中提供的内容将用于填充 Label 控件。这 3 个步骤如图 5-46 所示。

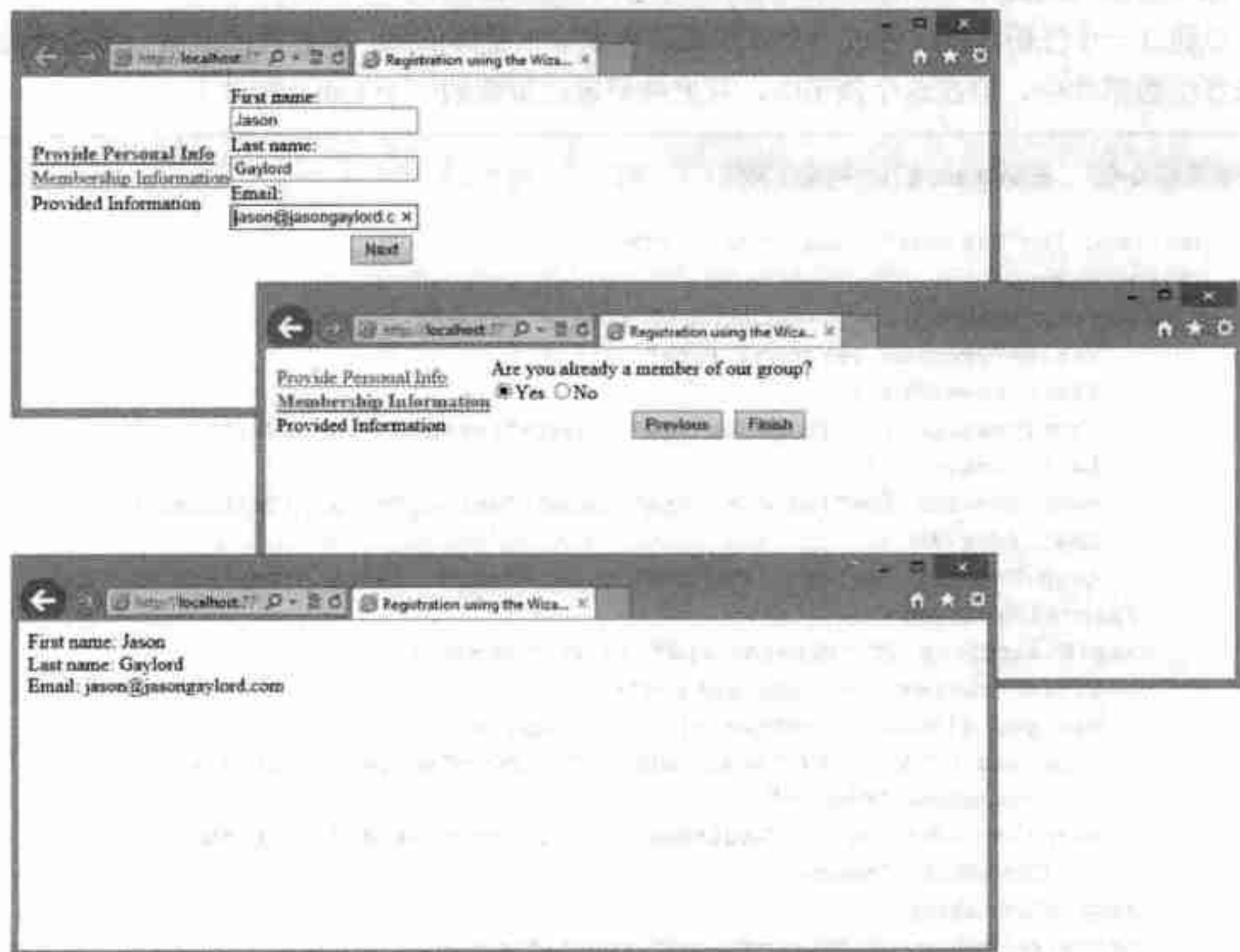


图 5-46

这个例子简单明了,但是我们可以使它复杂一些。假定要给该过程添加另一个 WizardStep 控件,而且希望只有在用户在 WizardStep2 中声明他是成员后才显示该控件。如果用户通过单选按钮回答他不是成员,就跳过新增的一步,直接进入最后一步,该步骤在 Label 控件中显示结果。首先给 Wizard 控件添加另一个 WizardStep 控件,如程序清单 5-48 所示。

程序清单 5-48 添加另一个 WizardStep 控件

```
<asp:Wizard ID="Wizard1" runat="server">
  <WizardSteps>
    <asp:WizardStep ID="WizardStep1" runat="server"
      Title="Provide Personal Info">
      First name:<br />
      <asp:TextBox ID="fnameTextBox" runat="server"></asp:TextBox><br />
      Last name:<br />
      <asp:TextBox ID="lnameTextBox" runat="server"></asp:TextBox><br />
      Email:<br />
      <asp:TextBox ID="emailTextBox" runat="server"></asp:TextBox><br />
    </asp:WizardStep>
    <asp:WizardStep ID="WizardStep2" runat="server"
      Title="Membership Information">
      Are you already a member of our group?<br />
      <asp:RadioButton ID="RadioButton1" runat="server" Text="Yes"
        GroupName="Member" />
      <asp:RadioButton ID="RadioButton2" runat="server" Text="No"
        GroupName="Member" />
    </asp:WizardStep>
    <asp:WizardStep ID="MemberStep" runat="server"
      Title="Provide Membership Number">
      Membership Number:<br />
      <asp:TextBox ID="mNumberTextBox" runat="server"></asp:TextBox>
    </asp:WizardStep>
    <asp:WizardStep ID="WizardStep3" runat="server"
      Title="Provided Information"
      StepType="Complete" OnActivate="WizardStep3_Activate">
      <asp:Label ID="Label1" runat="server" />
    </asp:WizardStep>
  </WizardSteps>
</asp:Wizard>
```

在工作流中添加了一步,要求用户输入成员号。只有在终端用户在 WizardStep2 中声明他是成员后才会显示这一步。因此需要添加事件(如程序清单 5-49 所示)来对成员资格进行检查。

程序清单 5-49 对是否显示某一步进行逻辑检查

```
void Wizard1_NextButtonClick(object sender, WizardNavigationEventArgs e)
{
  if(e.NextStepIndex == 2)
  {
    if(RadioButton1.Checked == true)
    {
      Wizard1.ActiveStepIndex = 2;
    }
    else
  }
```

```

        Wizard1.ActiveStepIndex = 3;
    }
}

```

要确定是否显示过程中的某一步, 可以使用 Wizard 控件的 NextButtonClick 事件。该事件使用 WizardNavigationEventArgs 类来代替一般的 EventArgs 类, 以获取 NextStepIndex 号和 CurrentStepIndex 号。

在程序清单 5-49 所示的例子中, 检查过程中显示的下一步是否为 2(因为索引是基于 0 的(0、1、2 等), 所以要查看索引值是否为 2)。如果是索引中的第二步, 就检查在前一个 WizardStep 中选中了哪个单选按钮。如果选择了 RadioButton1 控件(表示用户是成员), 就将过程中的下一步指定为索引 2。如果选择了 RadioButton2 控件(表示用户不是成员), 就把索引指定为 3(最后一步), 从而绕过过程中输入成员号的那一步。

还可以修改这个例子, 只有当用户包含在某个角色(如 Admin)中时才显示 WizardStep 控件。



有关角色管理的内容详见第 19 章。

程序清单 5-50 中列出的代码只有在用户包含在某个角色(如 Admin)中时才显示 WizardStep 控件。

程序清单 5-50 根据角色对是否显示某一步进行逻辑检查

```

void Wizard1_NextButtonClick(object sender, WizardNavigationEventArgs e)
{
    if (e.NextStepIndex == 2)
    {
        if (Roles.IsUserInRole("ManagerAccess"))
        {
            Wizard1.ActiveStepIndex = 2;
        }
        else
        {
            Wizard1.ActiveStepIndex = 3;
        }
    }
}

```

5.28 ImageMap 服务器控件

ImageMap 服务器控件可以把图像转变为导航菜单。过去, 许多开发人员会把图像分解为多个部分, 再在一个表中把它们放在一起, 将这些部分重新组合为图像。在终端用户单击整个图像的某个特定部分时, 应用程序就会选择该部分图像, 并根据该选择进行相应的操作。

利用 ImageMap 控件, 就可以使用一幅图像, 通过坐标指定图像上的特定热区。程序清单 5-51 演示了这样一个例子。

程序清单 5-51 指定可单击的图像部分

```

<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void ImageMap1_Click(object sender, ImageMapEventArgs e)
    {
        Response.Write("You selected: " + e.PostBackValue);
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>ImageMap Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:ImageMap ID="ImageMap1" runat="server" HotSpotMode="PostBack"
            ImageUrl="~/Images/kids.jpg" OnClick="ImageMap1_Click">
            <asp:CircleHotSpot PostBackValue="Addison" Radius="26" X="145"
                Y="372" />
            <asp:CircleHotSpot PostBackValue="Brayden" Radius="20" X="181"
                Y="314" />
            <asp:CircleHotSpot PostBackValue="Arianna" Radius="28" X="245"
                Y="344" />
        </asp:ImageMap>
    </form>
</body>
</html>

```

这个页面会显示一幅我和我孩子在水族馆的照片。如果单击孩子的脸，就把孩子的名字写入 Response 流，如图 5-47 所示。

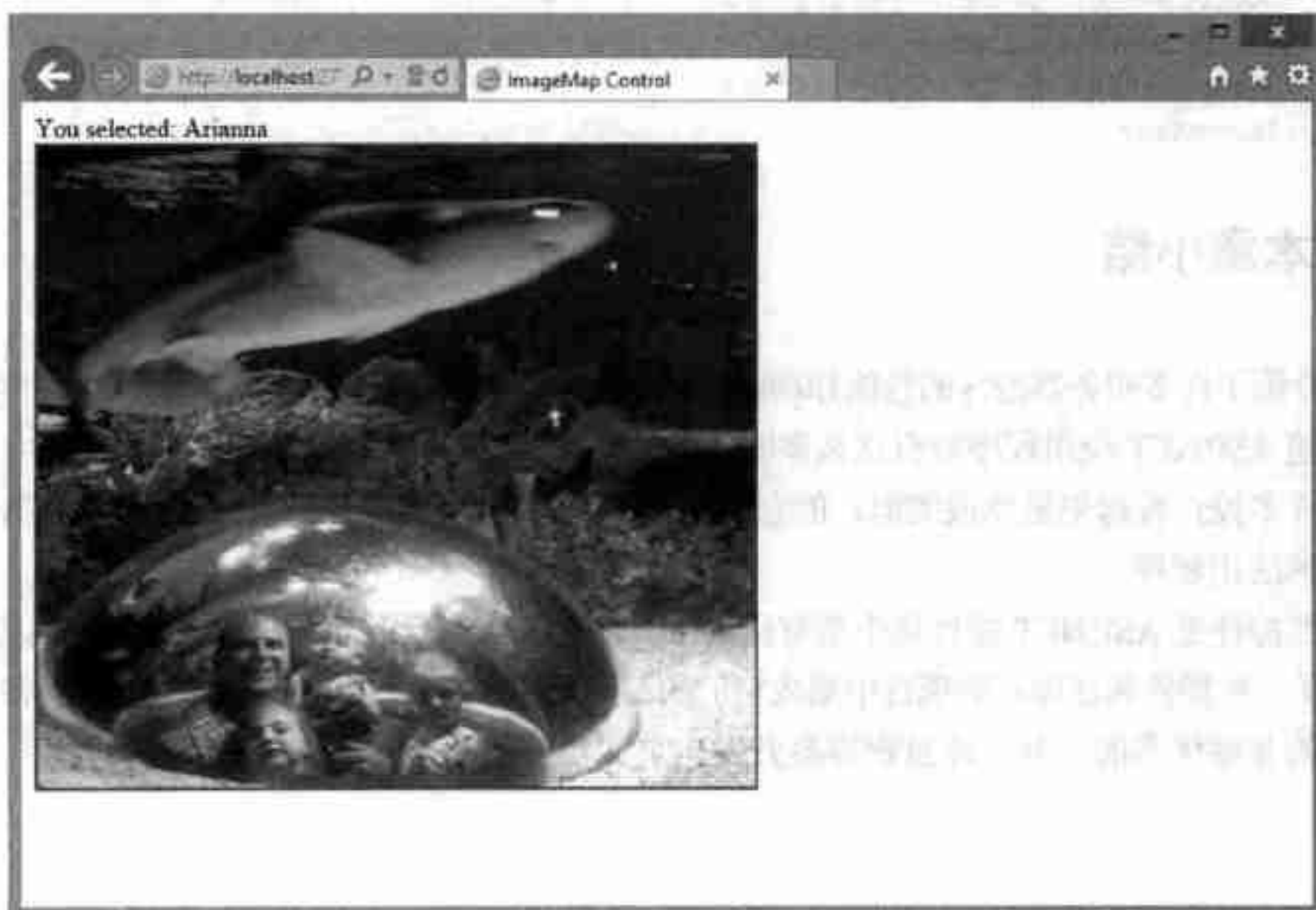


图 5-47

ImageMap 控件可以采用两种不同的方式指定热区。从程序清单 5-51 的例子可以看出, 可以使用<asp:CircleHotSpot>元素以圆形的方式指定热区。该控件使用热区矩形的 Top、Bottom、Left 和 Right 坐标。除了<asp:CircleHotSpot>控件之外, 还可以使用<asp:RectangleHotSpot>和<asp:PolygonHotSpot>控件, 每个控件都使用对应于其形状的坐标。

在定义图像上的热区之后, 就可以以几种方式响应终端用户对热区的单击。首先在根元素<asp:ImageMap>中使用 HotSpotMode 属性指定如何处理热区的单击。

HotSpotMode 属性的值可以是PostBack、Navigate 或 InActive。在前面的例子中, 将 HotSpotMode 属性的值设置为PostBack, 表示在终端用户单击热区之后, 应把该单击事件回送给服务器, 并在服务器上处理该单击事件。

由于 HotSpotMode 属性的值设置为PostBack, 而且创建了几个热区, 因此必须先确定选择了哪个热区。要确定这一点, 可以使用PostBackValue 属性给每个热区(<asp:RectangleHotSpot>)指定回送值。前面的例子使用 Arianna、Addison 和 BraydenSofia 作为三个热区的值。

PostBackValue 属性也会显示出现在浏览器中的帮助文本(显示在黄色框中), 当终端用户把鼠标停留在热区上时, 该文本就显示在鼠标指针的下面。

用户单击了一个热区后, 事件过程就在 Response.Write 语句中显示选中的值。

除了回送给服务器之外, 还可以在选择了某个热区时导航到一个完全不同的 URL。为此, 把主元素<asp:ImageMap>的 HotSpotMode 属性值改为 Navigate。接着在<asp:RectangleHotSpot>元素中使用 NavigateUrl 属性指定一个 URL, 如果终端用户单击了特定的热区, 就把终端用户定向到该 URL:

```
<asp:ImageMap ID="ImageMap1" runat="server" ImageUrl="~/Images/kids.jpg"
    HotSpotMode="Navigate">
    <asp:CircleHotSpot AlternateText="Addison" NavigateUrl="Addison.aspx"
        Radius="26" X="145" Y="372" />
    <asp:CircleHotSpot AlternateText="Brayden" NavigateUrl="Brayden.aspx"
        Radius="20" X="181" Y="314" />
    <asp:CircleHotSpot AlternateText="Arianna" NavigateUrl="Arianna.aspx"
        Radius="28" X="245" Y="344" />
</asp:ImageMap>
```

5.29 本章小结

本章介绍了许多服务器控件的性能和功能。在 ASP.NET 4.5 中, 有超过 50 个服务器控件可供使用。

在创建 ASP.NET 应用程序时有这么多服务器控件可供使用, 因此必须仔细考虑最适合完成任务的控件。许多控件看起来虽然很类似, 但它们提供不同的功能。使用这些控件, 可以为所有浏览器创建最佳的应用程序。

服务器控件是 ASP.NET 控件集中最有用的工具。它们非常有用, 可以节省开发人员大量的时间。本章介绍了一些控件和在以后的项目中集成它们的不同方式。所有这些控件都是可以在 ASP.NET 页面上使用的非常优秀的工具, 而且很容易开发出页面需要的功能。

验证服务器控件

本章要点

- 了解提供的验证服务器控件
- 掌握客户端和服务端验证
- 启用隐含验证

查看 Visual Studio 2012 中的 Toolbox 窗口时,尤其是在阅读了第 4 章和第 5 章后(这两章介绍了各种服务器控件),用户可能会惊讶于 ASP.NET 4.5 提供的服务器控件的数量。本章将介绍 Toolbox 窗口中一种特殊的服务器控件:验证服务器控件。

验证服务器控件可以处理终端用户在应用程序的窗体元素中输入的信息。这些控件可确保放在窗体上的数据的有效性。

在学习如何使用这些控件之前,首先介绍一下验证过程。

6.1 有效性验证

多年以来,人们一直在构建 Web 应用程序,目的通常是提供或收集信息。本章将主要讨论 Web 应用程序的信息收集和验证方面。如果使用应用程序收集数据,那么收集有效的数据是非常重要的。如果信息无效,收集它们就没有意义。

验证就是给所收集的数据应用的一系列规则。规则可多可少,或严格或宽松,这完全取决于业务规则和需求。不存在十全十美的验证过程,因为无论建立什么样的规则,总有一些用户能找出某种绕过这些规则的捷径。难点在于使规则的数量和严格程度保持平衡,并且不会牺牲应用程序的可用性。

为进行有效性验证而收集的数据来自于应用程序中提供的 Web 窗体。Web 窗体由不同类型的 HTML 元素组成,使用原始的 HTML 窗体元素、ASP.NET HTML 服务器控件或 ASP.NET Web Form 服务器控件构造这些元素。最终,窗体由许多不同类型的 HTML 元素构成,如文本框、单选按钮、复选框、下拉列表等。

在阅读本章时你会看到，可以给窗体元素应用不同类型的验证规则。但是，不能验证所收集信息的真实性，只能对如下问题的响应应用规则：

- 文本框中输入数据了吗？
- 文本框中输入的数据采用了电子邮件地址的形式吗？

从这些问题可以看出，可以对一个 HTML 窗体元素应用多个验证规则(本章后面有这方面的例子)。实际上，可以对一个元素应用任意多个规则。对元素应用的规则越多，应用到数据上的有效性验证就越严格。

6.2 客户端和服务端端的验证

如果对 Web 应用程序的开发比较陌生，就可能注意不到客户端和服务端验证之间的区别。假定终端用户在填充了一些信息后单击窗体上的 Submit 按钮。在 ASP.NET 中，这个窗体会被打包到一个请求中，并发送给应用程序所在的服务器。在请求/响应循环的这一刻，就可以对所提交的信息进行有效性验证。这就称为服务器端验证，因为该验证是在服务器上进行的。

另一方面，也可以在给终端用户的浏览器传送的页面上提供一个客户端脚本(通常采用 JavaScript 形式)，从而在窗体回送给服务器之前对输入到该窗体上的数据进行验证。这种验证称为客户端验证。

客户端验证能很快响应终端用户，这正是终端用户希望的情况。如果窗体有错误，使用客户端验证能确保终端用户很快就知道该错误。其原因是客户端验证如果正确调用，就会在窗体传送给服务器之前执行。客户端验证还会把验证需要的处理功能赋予客户端，这样就不需要进入服务器上的 CPU 循环来处理这些信息，因为客户端可以完成这一工作。

本章前面已经介绍过，客户端验证不太安全。在终端用户的浏览器上生成一个页面时，这个终端用户很容易查看页面的代码(只需右击浏览器并选择 View Code 命令)。此时，该用户除了可以查看页面的 HTML 代码外，还可以看到与该页面相关的所有 JavaScript。在客户端中验证窗体时，诡计多端的黑客也很容易把窗体(包含该黑客希望的值)作为有效窗体，重新发送给服务器。为此，客户端有时会简单地禁用浏览器上的客户端脚本编写功能，使验证无效。因此，客户端验证对于终端用户来说非常方便，但绝不能作为一种安全机制。但是，尽管有这些风险，客户端验证仍很普遍，因为它提供了更好的用户体验。

比较安全的验证形式是服务器端验证。这种验证在服务器上进行，而不是在客户端上进行。服务器端验证比较安全，因为这些验证不容易被绕过。在服务器上使用服务器代码(C#或 VB)检查窗体数据值。如果窗体无效，就将页面回送到客户端。尽管服务器端验证比较安全，但速度较慢。这是因为页面必须传送到一个远程位置进行检查。如果终端用户等待 20 秒仍未收到传送来的窗体，而是被告知其电子邮件地址的格式不正确，他们就会不高兴。

那么选择哪种验证是正确的呢？实际上，选择这两种验证都是正确的。比较好的方法是先进行客户端验证，在窗体验证通过并传送给服务器后，再使用服务器端验证进行检查。这种方法可以综合这两种验证的优点。它很安全，因为黑客不能绕过验证。他们可以绕过客户端验证，但很快会发现其窗体数据在传送给服务器后一定会被再次检查。这种验证技术的效率很高——具有客户端验证的快速特性。

6.3 ASP.NET 验证服务器控件

在 Web 开发过程中,对显示在 Web 页面上的窗体进行验证是很常见的操作。因此,在最初发布 ASP.NET 时,ASP.NET 小组引入了一系列验证服务器控件,以快速实现窗体的有效验证。

ASP.NET 不仅把窗体的验证作为服务器控件引入,还使这些控件智能化。使用其他一些 Web 技术进行开发时,开发人员必须确定在哪里进行窗体验证:在客户端还是服务器上。ASP.NET 验证服务器控件不需要进行这种进退两难的选择,因为在生成 ASP.NET 页面时,ASP.NET 会进行浏览器检测,并根据所收集的信息做出决策。

也就是说,如果浏览器支持 JavaScript,而 ASP.NET 可以发送 JavaScript,验证就在客户端进行。如果客户端不支持用于客户端验证的 JavaScript,JavaScript 就会被忽略,验证在服务器上进行。

此时,最好的情况是,即使在页面上启动了客户端验证,ASP.NET 也会在接收到提交的页面时进行服务器端验证,从而确保不牺牲安全性。验证服务器控件的这个决定性的特征意味着,要建立性能最佳的 ASP.NET Web 页面,而不能为了满足客户端和服务器的最低要求就降低 Web 应用程序的性能。



如前所述,服务器端验证比客户端验证慢。但是,因为服务器端验证仅在提交页面时进行,而且可以禁用控件的回送,所以对性能的影响不大严重。

现在,ASP.NET 4.5 中有 7 个验证控件。自 ASP.NET 第 1 版推出以来,没有增加任何新的验证服务器控件,但 ASP.NET 4.5 引入了一个新的验证技术——使用隐含的 JavaScript 功能,参见本章后面的内容。


可用的验证服务器控件如下:

- RequiredFieldValidator
- CompareValidator
- RangeValidator
- RegularExpressionValidator
- CustomValidator
- DynamicValidator
- ValidationSummary



所有的验证服务器控件都在 Validation 工具箱中,但 DynamicValidator 除外,它在 Dynamic Data 工具箱中。

使用 ASP.NET 验证服务器控件与使用其他 ASP.NET 服务器控件没有什么区别。这些控件都可以拖放到设计界面上,或者直接在 ASP.NET 页面的代码中操作。还可以修改这些控件,使其外观符合我们的要求——确保应用程序所需的外观唯一性。本章将介绍这些方面的内容。



如果 ASP.NET 验证控件无法满足要求，那么可以编写自定义的验证控件。但是，有许多第三方控件，例如，<http://www.peterblum.com/DES> 上的 Peter Blum's Validation and More (VAM)就包含 50 多个 ASP.NET 验证控件。

表 6-1 列出了每个验证服务器控件的功能。

表 6-1

验证服务器控件	说 明
RequiredFieldValidator	确保用户不跳过某个窗体输入字段
CompareValidator	允许使用比较运算符(等于、大于、小于等)比较用户的输入和另一项
RangeValidator	根据数字范围或字母范围检查用户的输入
RegularExpressionValidator	检查用户的输入是否匹配正则表达式定义的模式。这是一种很有用的控件，可用于检查电子邮件地址和电话号码
CustomValidator	使用自定义编码的验证逻辑检查用户的输入
DynamicValidator	在处理实体数据模型和扩展方法抛出的异常时使用。该控件是 ASP.NET Dynamic Data Framework 的一部分。有关该验证控件的更多内容，可以在 Internet 中搜索 DynamicValidator
ValidationSummary	在页面的某个特定位置显示验证控件的所有错误消息

6.3.1 验证原因

验证不是随时进行的，而是为响应事件而启动的，在大多数情况下是为了响应按钮单击事件。Button、LinkButton 和 ImageButton 服务器控件都可以启动页面的窗体验证。这是默认行为。把 Button 服务器控件拖放到窗体上，就会得到如下结果：

```
<asp:Button ID="Button1" runat="server" Text="Button" />
```

查看 Button 控件的属性，可以发现 CausesValidation 属性被设置为 true。如前所述，这是默认行为——页面上的所有按钮(无论有多少)都会启动窗体验证。

如果 ASP.NET 页面上有多个按钮，并且不希望每个按钮都启动窗体验证过程，就可以把要忽略验证过程的按钮(例如窗体的 Cancel 按钮，或者在另一个窗体元素中按下回车键，例如搜索框)的 CausesValidation 属性设置为 false：

```
<asp:Button ID="Button1" runat="server" Text="Cancel" CausesValidation="false" />
```

6.3.2 ASP.NET 4.5 中的隐含验证

传统上，验证是使用内置控件在服务器端处理，或者编写方法来检查窗体上的值。在客户端，会给浏览器省略这些 JavaScript。在这种情况下，终端用户可能被验证消息淹没。例如，如果 JavaScript 本身包含错误，页面就可能无法正确加载。为了避免这些问题，jQuery 验证插件更新为支持隐含验证。

隐含验证有助于减少用户与 Web 应用程序交互操作的侵犯。JavaScript 不是在线显示，而是给

合适的窗体元素添加 HTML5 data-* 特性。在 ASP.NET 4.5 中, 使用了 jQuery 验证插件, 默认支持隐含验证功能。

与 Visual Studio 的以前版本不同, Visual Studio 2012 包含一个真正空白的项目模板。这表示, 在解决方案中仅添加了一个项目和一个 web.config 文件。如果选择创建新网站, web.config 就只包含如下内容:

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.5"/>
    <httpRuntime targetFramework="4.5"/>
  </system.web>
</configuration>
```

可以在 ASP.NET 中使用的一个验证控件是 RequiredFieldValidator。程序清单 6-1 演示了 RequiredFieldValidator 控件的简单用法。

程序清单 6-1 RequiredFieldValidator 服务器控件的简单用法

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
  protected void Button1_Click(Object sender, EventArgs e)
  {
    if (Page.IsValid)
    {
      Label1.Text = "Page is valid!";
    }
  }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title></title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
      <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
        runat="server" Text="Required!" ControlToValidate="TextBox1"
        EnableClientScript="true">
      </asp:RequiredFieldValidator>
      <br />
      <asp:Button ID="Button1" runat="server" Text="Submit"
        OnClick="Button1_Click" />
      <br />
      <asp:Label ID="Label1" runat="server"></asp:Label>
    </div>
  </form>
</body>
</html>
```


如果运行这个页面，让文本框为空，提交窗体，就会得到异常，如图 6-1 所示。

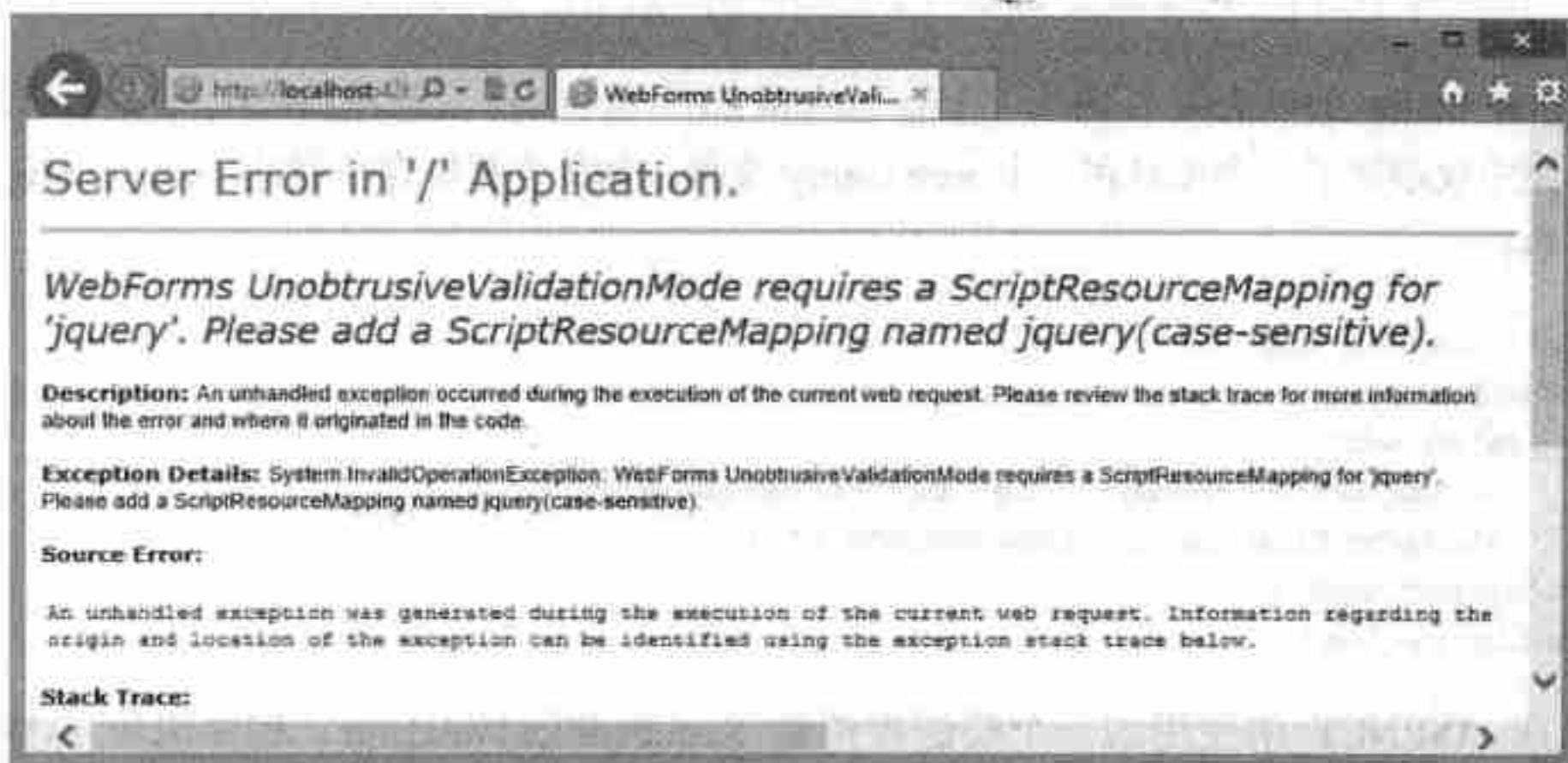


图 6-1

发生异常是因为 ASP.NET 希望用户定义验证的显示方式。在 web.config 文件中添加如下代码，可以启用隐含验证：

```
<appSettings>
  <add key="ValidationSettings:UnobtrusiveValidationMode" value="WebForms" />
</appSettings>
```

ValidationSettings:UnobtrusiveValidationMode 的值可以是 None 或 WebForms。

- WebForms 表示应使用 Web Forms 隐含验证模式。
- None 表示不使用隐含验证。

如果不在 web.config 文件中设置这个值，也可以在 global.asax 文件的 Application_Start 方法中设置，如下所示：

```
ValidationSettings.UnobtrusiveValidationMode = UnobtrusiveValidationMode.WebForms;
```

如果希望给每个页面设置这个值，就可以在 Page_Load 事件中添加如下代码：

```
Page.UnobtrusiveValidationMode = UnobtrusiveValidationMode.WebForms;
```

如果把 UnobtrusiveValidationMode 设置为使用 WebForms，就还没有完成，还必须用 ASP.NET Ajax ScriptManager 注册 jQuery JavaScript 库。



jQuery 详见第 25 章。

安装 NuGet 包，启用隐含验证

如果还没有在 Visual Studio 中安装 jQuery，下面是安装步骤。使用 Visual Studio 2012 中的 Package Manager 安装两个不同的包。要访问 Package Manager，可以进入 View | Other Windows | Package Manager Console。

(1) 需要安装的第一个包是 jQuery，它允许 ASP.NET 使用 jQuery JavaScript 库。安装该包后，脚本就放在 scripts 文件夹中。要使用 Package Manager 安装 jQuery，应输入 `install-package jQuery`，执行这个命令后，就安装了 jQuery 的最新版本。

(2) 需要安装的第二个包会把 jQuery 添加到 `PreApplicationStartupMethod` 的 `ScriptManager` 对象中。要使用 Package Manager 安装 jQuery，应输入 `install-package AspNet.ScriptManager.jQuery`。

6.3.3 RequiredFieldValidator 服务器控件

`RequiredFieldValidator` 控件只检查在 HTML 窗体元素中是否输入了信息。它是一个简单的验证控件，也是最常用的验证控件。必须为每个需要实施输入值规则的窗体元素指定 `RequiredFieldValidator` 控件。



所有的验证服务器控件，除了不执行验证操作的验证汇总控件之外，都只能一次验证一个输入控件。因此，如果窗体中的每个输入控件都需要验证，那么每个输入控件都需要一个 `RequiredFieldValidator` 控件。

6.3.1 节在介绍隐含验证时，如果试图使用 `RequiredFieldValidator` 控件，就会抛出异常。但从现在开始，添加了 jQuery，告诉 ASP.NET 用 `ScriptManager` 注册脚本来处理这个异常。页面上会显示一个简单的文本框和一个按钮。不要在文本框中输入任何值，单击 `Submit` 按钮，结果如图 6-2 所示。

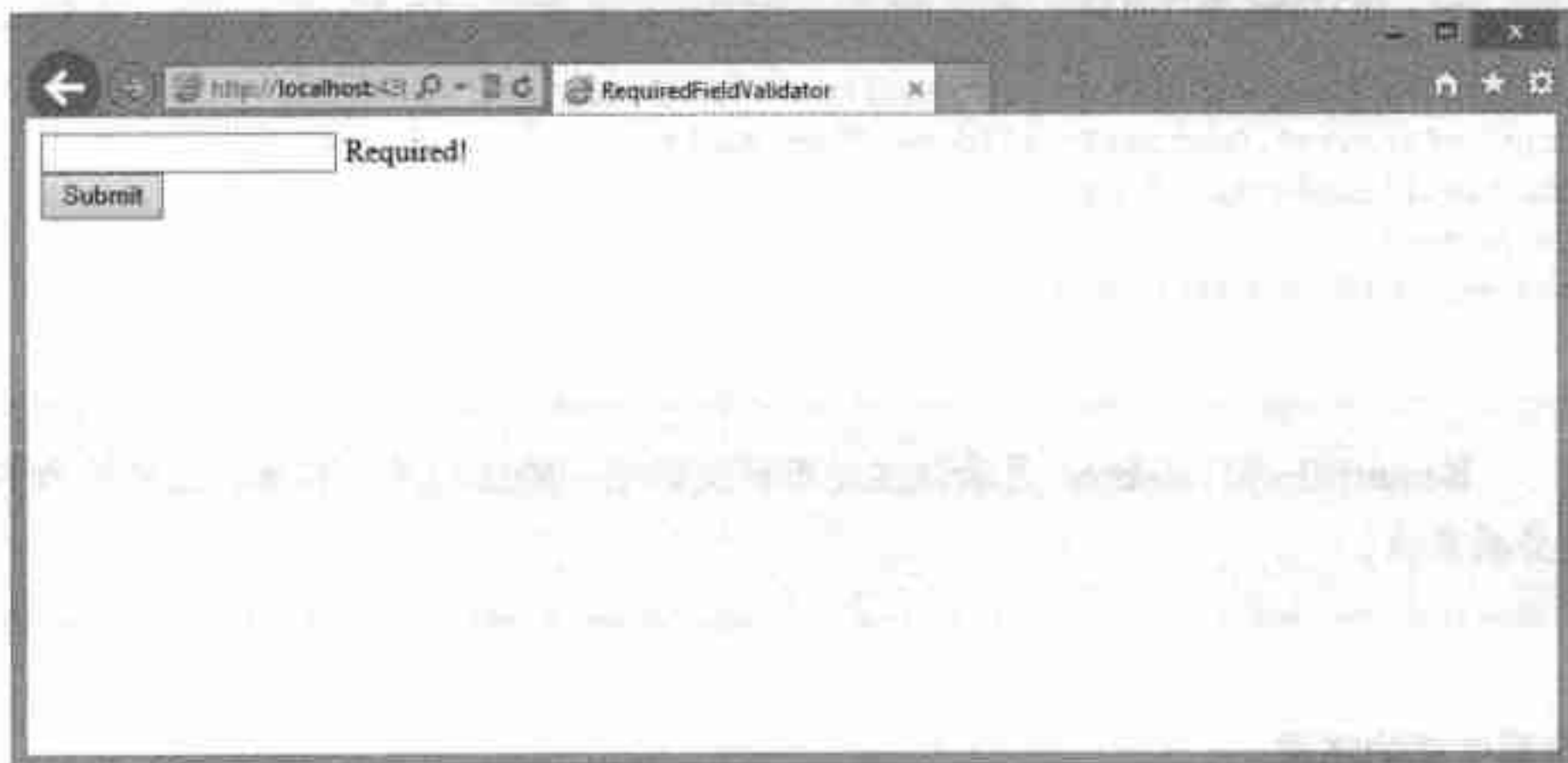


图 6-2

现在查看程序清单 6-1 中的代码。首先，`TextBox`、`Button` 和 `Label` 控件没有什么区别。它们的构建与不使用任何窗体验证时一样。但是，这个页面包含一个简单的 `RequiredFieldValidator` 控件。这个控件的几个属性值得特别注意，因为在我们创建的大多数验证服务器控件中都要使用它们。

第一个属性是 `Text`。如果验证失败，这个属性的值就会通过 Web 页面显示给终端用户。在本例中，该属性的值是简单的 `Required!` 字符串。第二个属性是 `ControlToValidate`，用于在该验证服务器控件和需要验证的 ASP.NET 窗体元素之间建立关联。在本例中，其值指定了窗体中的唯一元素，即文本框。



可以使用 `ControlToValidate` 属性验证的 Web 控件有:

- DropDownList
- FileUpload
- ListBox
- RadioButtonList
- TextBox

可以使用 `ControlToValidate` 属性验证的 HTML 控件有:

- HtmlInputFile
- HtmlInputPassword
- HtmlInputText
- HtmlSelect
- HtmlTextArea

从这个例子中可以看出, 可以根据 `<asp:RequiredFieldValidator>` 控件的属性构造错误消息。

还可以在 `<asp:RequiredFieldValidator>` 起始和结束节点之间表示这个错误消息, 如程序清单 6-3 所示。

程序清单 6-2 在节点之间放置值

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
    runat="server" ControlToValidate="TextBox1"
    EnableClientScript="true">
    Required!
</asp:RequiredFieldValidator>
```



`RequiredFieldValidator` 是验证值是否提交的唯一验证控件。其他验证控件都不需要提交值。

1. 查看生成的结果

如果浏览器允许进行客户端验证, `RequiredFieldValidator` 控件就使用客户端验证。右击页面, 从弹出的菜单中选择 `View Source` 命令, 就可以查看客户端验证代码。在页面代码中, 包含了程序清单 6-3 中所示的标记。

程序清单 6-3 生成的部分标记

```
<!-- Abbreviated for clarity -->
<input name="TextBox1" type="text" id="TextBox1" />
<span data-val-controltovalidate="TextBox1" id="RequiredFieldValidator1" data-val="true"
```



```

    data-val-evaluationfunction="RequiredFieldValidatorEvaluateIsValid"
    data-val-initialvalue="" style="visibility:hidden;">
        Required!
    </span>
    <br />
    <input type="submit" name="Button1" value="Submit"
onclick="javascript:WebForm_DoPostBackWithOptions(new
WebForm_PostBackOptions(&quot;Button1&quot;,, &quot;&quot;,, true, &quot;&quot;,, &quot;&quot;,,
false, false))" id="Button1" />

```

JavaScript 函数 `WebForm_DoPostBackWithOptions` 启动了客户端验证。注意 `span` 元素包含 4 个以 `data-val` 开头的特性。这些特性帮助 JavaScript 函数确定哪些元素用于验证，以及如何处理验证。



只有打开隐含验证功能，`data-*` 特性才会显示出来。如果关闭隐含验证功能，JavaScript 就会传递给浏览器来处理验证，如下所示：

```

<script type="text/javascript">
//
var RequiredFieldValidator1 = document.all ? document.all
["RequiredFieldValidator1"] :
document.getElementById("RequiredFieldValidator1");
RequiredFieldValidator1.controltovalidate = "TextBox1";
RequiredFieldValidator1.evaluationfunction =
    "RequiredFieldValidatorEvaluateIsValid";
RequiredFieldValidator1.initialvalue = "";
//]]&gt;
&lt;/script&gt;
</pre>
</div>
<div data-bbox="128 568 341 585" data-label="Section-Header">
<h2>2. 使用 InitialValue 属性</h2>
</div>
<div data-bbox="87 602 918 660" data-label="Text">
<p><code>RequiredFieldValidator</code> 控件的一个重要属性是 <code>InitialValue</code>。在程序清单 6-3 中，初始值显示为 <code>span</code> 元素的 <code>data-val-initialvalue</code> 特性。有时使用一些默认的属性填充窗体元素(如数据存储中的窗体元素)，这些窗体元素会给终端用户显示在窗体提交给服务器之前需要改变的值。</p>
</div>
<div data-bbox="87 666 918 704" data-label="Text">
<p>在使用 <code>InitialValue</code> 属性时，为 <code>RequiredFieldValidator</code> 控件指定元素的初始文本。终端用户需要先改变该文本值，再提交窗体。程序清单 6-4 是使用这个属性的一个例子。</p>
</div>
<div data-bbox="120 727 405 742" data-label="Caption">
<p>程序清单 6-4 使用 <code>InitialValue</code> 属性</p>
</div>
<div data-bbox="129 757 845 815" data-label="Text">
<pre>
&lt;asp:TextBox ID="TextBox1" runat="server" Text="Wrox"&gt;&lt;/asp:TextBox&gt;
&lt;asp:RequiredFieldValidator ID="RequiredFieldValidator1"
    runat="server" ControlToValidate="TextBox1" InitialValue="Wrox"
    EnableClientScript="true" Text="Please change the value of the textbox!" /&gt;
</pre>
</div>
<div data-bbox="87 827 907 866" data-label="Text">
<p>在这个例子中，<code>InitialValue</code> 属性包含 <code>Wrox</code> 值。在编译和运行页面时，文本框也包含这个值。<code>RequiredFieldValidator</code> 控件要求改变这个值，这样页面才是有效的。</p>
</div>
<div data-bbox="128 882 431 898" data-label="Section-Header">
<h2>3. 禁止为空，同时要求改变初始值</h2>
</div>
<div data-bbox="125 913 918 930" data-label="Text">
<p>在前面使用 <code>InitialValue</code> 属性的例子中，存在一个有趣的问题。首先，如果运行这个例子，在进</p>
</div>
<div data-bbox="913 952 948 965" data-label="Page-Footer">197</div>
```

行窗体验证时,终端用户可以在提交窗体时不给这个文本框输入任何值。空白的文本框不会引发验证错误,因为 RequiredFieldValidator 控件现在重新构建为要求终端用户只改变文本框的默认值(删除旧值即可)。以这种方式重新构建 RequiredFieldValidator 控件,验证规则将不要求在文本框中输入信息,只要求改变初始值。这样用户只要删除这个文本框中的内容,就可以绕过窗体验证过程。

该问题有一个解决方法。如前所述,一个窗体可以包含多条验证规则,其中一些规则可以应用于同一个窗体元素。为了修改文本框的初始值,同时禁止文本框为空(从而标记该元素是必需的元素),必须在页面上放置另一个 RequiredFieldValidator 控件。第二个 RequiredFieldValidator 控件与第一个 RequiredFieldValidator 控件都与同一个文本框相关联,如程序清单 6-5 所示。

程序清单 6-5 给同一个窗体元素使用两个 RequiredFieldValidator 控件

```
<asp:TextBox ID="TextBox1" runat="server" Text="Wrox"></asp:TextBox>
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
    runat="server" ControlToValidate="TextBox1" InitialValue="Wrox"
    EnableClientScript="true" Text="Please change the value of the textbox!" />
<asp:RequiredFieldValidator ID="RequiredFieldValidator2"
    runat="server" ControlToValidate="TextBox1"
    EnableClientScript="true" Text="Please do not leave this field blank!" />
```

在这个例子中,文本框关联了两个 RequiredFieldValidator 控件。第一个控件 RequiredFieldValidator1 通过使用 InitialValue 属性要求改变文本框的默认值。第二个控件 RequiredFieldValidator2 只是把 TextBox1 控件变成一个需要值的窗体元素。给一个窗体元素应用两条验证规则,就可以得到我们需要的行为。

4. 使用 RequiredFieldValidator 控件验证下拉列表

前面已介绍了许多使用 RequiredFieldValidator 控件和一系列文本框的例子,这个验证控件还可以用于其他窗体元素。

例如,可以把 RequiredFieldValidator 控件用于<asp:DropDownList>服务器控件。假定有一个下拉列表,需要终端用户从项列表中选择其职业。下拉列表的第一行包含终端用户选择什么内容的指令,并且我们希望下拉列表也变成一个必需的窗体元素。其代码如程序清单 6-6 所示。

程序清单 6-6 对下拉列表的验证

```
<asp:DropDownList ID="DropDownList1" runat="server">
    <asp:ListItem Selected="True">Select a profession</asp:ListItem>
    <asp:ListItem>Programmer</asp:ListItem>
    <asp:ListItem>Lawyer</asp:ListItem>
    <asp:ListItem>Doctor</asp:ListItem>
    <asp:ListItem>Artist</asp:ListItem>
</asp:DropDownList>
  

<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
    runat="server" Text="Please make a selection"
    ControlToValidate="DropDownList1"
    InitialValue="Select a profession">
</asp:RequiredFieldValidator>
```


与处理文本框一样,这个例子中的 RequiredFieldValidator 控件通过 ControlToValidate 属性关联 DropDownList 控件。验证控件绑定的下拉列表有初始值 Select a profession。显然,终端用户在把窗体传送给服务器时不应保留这个值,因此要使用 RequiredField Validator 控件的 InitialValue 属性。这个属性的值被赋给下拉列表最初选中的值,这将迫使终端用户在下拉列表中选择给定的职业,之后才能传送窗体。

6.3.4 CompareValidator 服务器控件

CompareValidator 控件可以比较两个窗体元素,也可以比较窗体元素包含的值与指定的常量。例如,可以指定窗体元素的值必须是整数,并且大于指定的数字。也可以要求值必须是字符串、日期或其他数据类型。

1. 根据其他控件来验证

使用 CompareValidator 控件的一种常用方式是比较两个窗体元素。例如,假定有一个应用程序,要求用户在访问站点时必须输入密码。创建一个文本框,要求用户输入密码;再创建第二个文本框,要求用户确认密码。因为文本框采用密码模式,所以终端用户看不到输入的内容,只能看出输入了多少个字符。为了减少终端用户的密码输入错误以及把不正确的密码输入系统中的机会,要求用户确认其密码。在窗体传送给系统后,只需比较两个文本框,检查它们是否匹配。如果匹配,就说明终端用户正确地输入了密码,从而可以把密码传送给系统。如果两个文本框不匹配,就表示窗体无效。程序清单 6-7 中的例子说明了这种情况。

程序清单 6-7 使用 CompareValidator 根据其他控件的值来检测指定的值

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Button1_Click(Object sender, EventArgs e) {
        if(Page.IsValid)
            Label1.Text = "Passwords match";
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>CompareFieldValidator</title>
</head>
<body>
    <form id="form1" runat="server">
        <p>
            Password<br />
            <asp:TextBox ID="TextBox1" runat="server"
                TextMode="Password"></asp:TextBox>
            &nbsp;
            <asp:CompareValidator ID="CompareValidator1"
                runat="server" Text="Passwords do not match!"
                ControlToValidate="TextBox2"
                ControlToCompare="TextBox1"></asp:CompareValidator>
        </p>
        <p>
```



```

        Confirm Password<br />
        <asp:TextBox ID="TextBox2" runat="server"
            TextMode="Password"></asp:TextBox>
    </p>
    <p>
        <asp:Button ID="Button1" OnClick="Button1_Click"
            runat="server" Text="Login"></asp:Button>
    </p>
    <p>
        <asp:Label ID="Label1" runat="server"></asp:Label>
    </p>
</form>
</body>
</html>

```

窗体上的 CompareValidator 控件类似于 RequiredFieldValidator 控件。CompareValidator 控件有一个 ControlToValidate 属性，它把该控件与页面上的一个窗体元素关联起来。在本例中，页面上只需要一个 CompareValidator 控件，因为只进行一次比较。在这个例子中，比较的是 TextBox2 和 TextBox1 的值。因此，使用了 ControlToCompare 属性，它指定要与 TextBox2 比较的值，在此是指 TextBox1 的值。



开发人员使用 CompareValidator 控件时的常见错误是弄错 ControlToValidate 和 ControlToCompare 属性的值，确保再次检查这些值。

这是非常简单的比较操作。如果在终端用户发送页面后两个文本框不匹配，就在浏览器上显示 CompareValidator 控件的 Text 属性值，如图 6-3 所示。



图 6-3

2. 根据常量来验证

除了根据其他控件中的值进行验证之外，还可以使用 CompareValidator 控件根据特定数据类型的常量执行验证。例如，假定注册窗体上有一个文本框，要求输入用户的年龄。在大多数情况下，应输入数字，而不是输入像 aa 或 bb 这样的值。程序清单 6-8 说明了如何确保输入的是数字。

程序清单 6-8 使用 CompareValidator 根据常量来验证

```

Age:
<asp:TextBox ID="TextBox1" runat="server" MaxLength="3">
</asp:TextBox>
  

<asp:CompareValidator ID="CompareValidator1" runat="server"
    Text="You must enter a number"
    ControlToValidate="TextBox1" Type="Integer"
    Operator="DataTypeCheck"></asp:CompareValidator>

```

在这个例子中，要求终端用户在文本框中输入数字。如果用户试图输入不包含数字的值来绕过验证过程，页面就是无效的，CompareValidator 控件就会显示 Text 属性的值。

为了指定在这些比较中使用的数据类型，只需使用 Type 属性，该属性的值可以是：

- Currency
- Date
- Double
- Integer
- String

不仅可以确保输入的数据是特定的类型，还可以在特定的常量比较时确保输入的数据是有效的。例如，可以确保在窗体元素中输入的数据大于、小于、等于、大于等于、小于等于某个值。程序清单 6-9 演示了这样一个例子。

程序清单 6-9 使用 CompareValidator 控件进行比较

```

Age:
<asp:TextBox ID="TextBox1" runat="server" MaxLength="3">
</asp:TextBox>
  

<asp:CompareValidator ID="CompareValidator1" runat="server"
    Text="You must be at least 18 to join"
    ControlToValidate="TextBox1" Type="Integer"
    Operator="GreaterThanOrEqual" ValueToCompare="18"></asp:CompareValidator>

```

在这个例子中，CompareValidator 控件关联到 TextBox1 控件，并要求其值必须是整数，另外，还需要使用 Operator 和 ValueToCompare 属性来确保输入的数字大于 18。因此，如果终端用户输入了 18 或更小的数字，验证就会失败，页面就是无效的。

Operator 属性的值可以是：

- Equal
- NotEqual
- GreaterThan
- GreaterThanOrEqual
- LessThan
- LessThanOrEqual
- DataTypeCheck

ValueToCompare 属性可以包含比较时使用的常量值。在前面的例子中，该值为 18。

6.3.5 RangeValidator 服务器控件

RangeValidator 控件类似于 CompareValidator 控件,但前者可确保终端用户提供的值或选项在指定的范围内,而不是大于或小于指定的常量。例如,返回要求输入终端用户年龄的文本框元素,对所提供的值进行验证,如程序清单 6-10 所示。

程序清单 6-10 使用 RangeValidator 控件检测整数值

```
Age:
<asp:TextBox ID="TextBox1" runat="server" MaxLength="3">
</asp:TextBox>
 
<asp:RangeValidator ID="RangeValidator1" runat="server"
    ControlToValidate="TextBox1" Type="Integer"
    Text="You must be between 30 and 40"
    MaximumValue="40" MinimumValue="30"></asp:RangeValidator>
```

在这个例子中,页面包含一个文本框,要求终端用户在其中输入年龄。RangeValidator 控件会分析所提供的值,并确保该值在 30 到 40 之间。可以使用 MaximumValue 和 MinimumValue 属性来设置该范围。RangeValidator 控件还能使用 Type 属性确保所输入的数据是整数类型,只要将 Type 属性设置为 Integer 即可。RangeValidator 控件可以处理的其他数据类型与 CompareValidator 控件相同。图 6-4 显示了这个例子的屏幕截图。

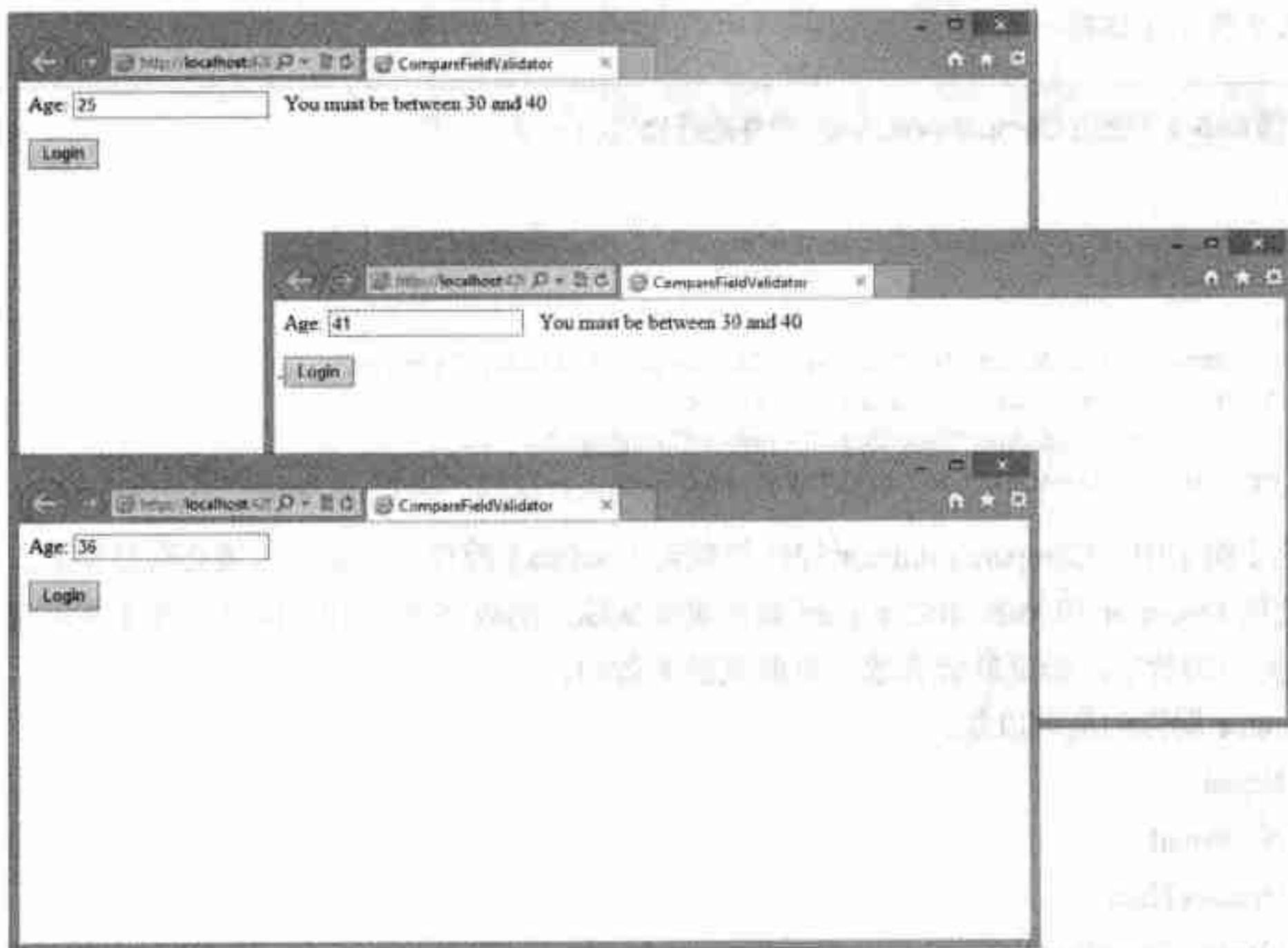


图 6-4

从图 6-4 中可以看出,小于 30 的值和大于 40 的值都会触发 RangeValidator 控件。30 到 40 之间的值(这里是 36)遵循该控件的验证规则。

RangeValidator 控件不仅可以验证数字(尽管它常常用于验证数字),还可以验证某个范围的字

字符串和其他项,包括日历中的日期。所有验证控件的 `Type` 属性都默认设置为 `String`, 以确保所提交的字符串的 ASCII 值在指定的范围内。可以使用 `RangeValidator` 控件确保在另一个服务器控件(如日历控件)中输入的数据在特定的日期范围内。

例如,假定建立一个 Web 窗体,要求输入顾客的到达日期,并且到达日期必须在当前日期后的两个星期之内。此时就可以使用 `RangeValidator` 控件检测该日期。

要检查的日期范围是动态生成的,因此在 `Page_Load` 事件中通过编程给 `MaximumValue` 和 `MinimumValue` 属性赋值。在设计视图中,该例的页面如图 6-5 所示。



图 6-5

终端用户会从 `Calendar` 控件中选择一个日期,再将该日期填充到 `TextBox` 控件中。接着,当终端用户单击窗体上的按钮时,系统就会通知他所选的日期是否有效。如果所选的日期是有效的,该日期就通过页面上的 `Label` 控件显示出来。这个例子的代码如程序清单 6-11 所示。

程序清单 6-11 使用 `RangeValidator` 控件检测字符串日期值

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        RangeValidator1.MinimumValue = DateTime.Now.ToShortDateString();
        RangeValidator1.MaximumValue =
            DateTime.Now.AddDays(14).ToShortDateString();
    }
    protected void Calendar1_SelectionChanged(object sender, EventArgs e)
    {
        TextBox1.Text = Calendar1.SelectedDate.ToShortDateString();
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        if (Page.IsValid)
        {
            Label1.Text = "You are set to arrive on: " + TextBox1.Text.ToString();
        }
    }
}
```

```

    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Date Validation Check</title>
</head>
<body>
    <form id="form1" runat="server">
        Arrival Date:
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>&nbsp;
        <asp:RangeValidator ID="RangeValidator1" runat="server"
            Text="You must only select a date within the next two weeks."
            ControlToValidate="TextBox1" Type="Date"></asp:RangeValidator>
        <br />
        <br />
        Select your arrival date:<br />
        <asp:Calendar ID="Calendar1" runat="server"
            OnSelectionChanged="Calendar1_SelectionChanged"></asp:Calendar>
        &nbsp;
        <br />
        <asp:Button ID="Button1" runat="server" Text="Button"
            OnClick="Button1_Click" />
        <br />
        <br />
        <asp:Label ID="Label1" runat="server"></asp:Label>
    </form>
</body>
</html>

```

从这段代码中可以看出,在加载页面时给 `MinimumValue` 和 `MaximumValue` 属性指定了动态值。在本例中, `MinimumValue` 属性得到的是 `DateTime.Now.ToShortDateString()` 值, `MaximumValue` 得到的是 14 天后的日期。

终端用户选择了一个日期后,就会使用 `Calendar1_SelectionChanged` 事件将选择的日期填充到 `TextBox1` 控件中。在用户选择日期并单击页面上的按钮后,就会触发 `Button1_Click` 事件,并使用 `Page.IsValid` 属性检查窗体的有效性。由于使用了 `Page.IsValid` 属性,因此会检查每个验证控件,确保验证通过。如果通过了验证,返回的值就是 `true`, 否则返回 `false`。无效的页面如图 6-6 所示。

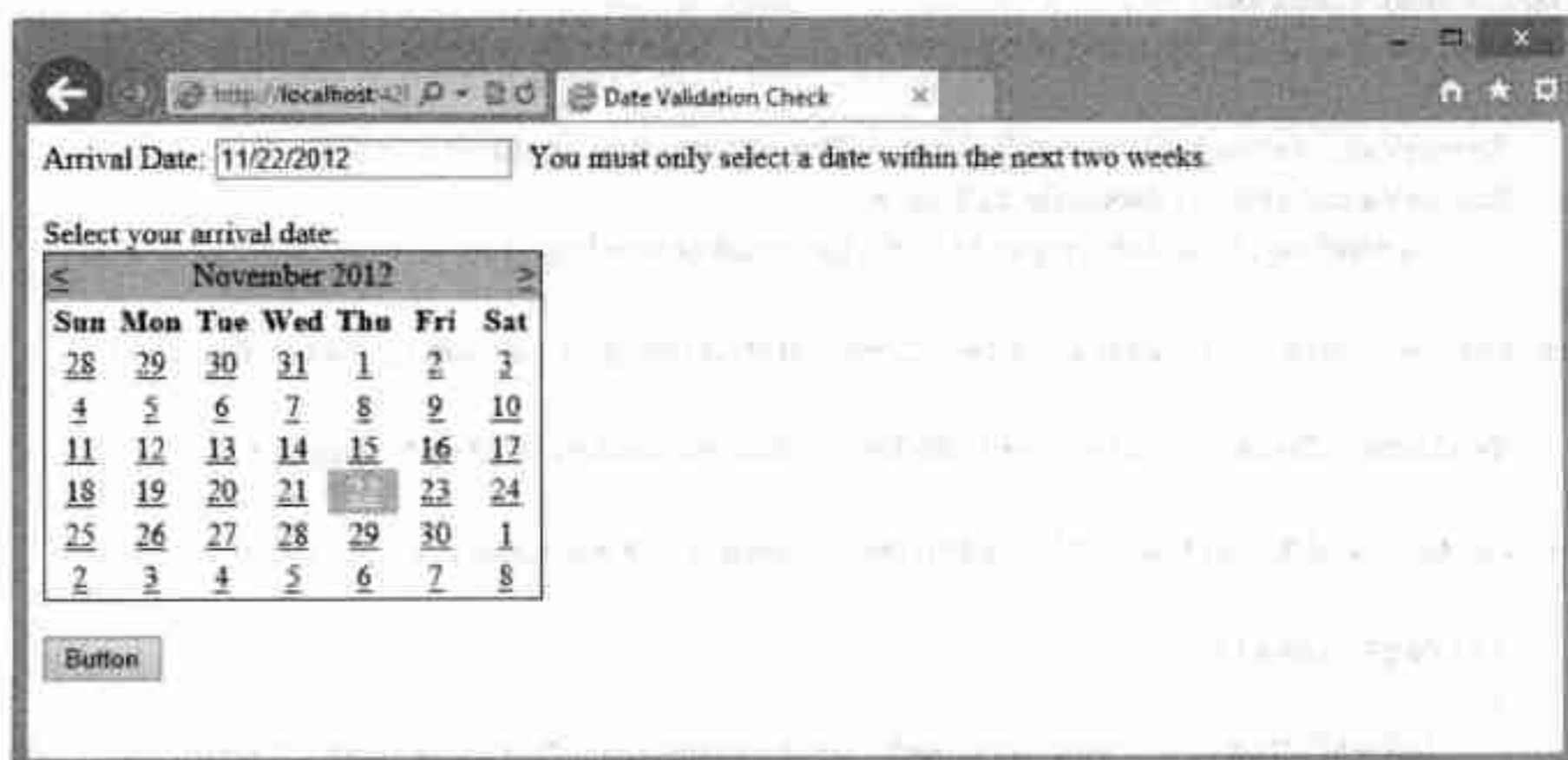


图 6-6

6.3.6 RegularExpressionValidator 服务器控件

开发人员喜欢使用的一个控件是 RegularExpressionValidator。在把验证规则应用于 Web 窗体时,这个控件提供了很大的灵活性。使用 RegularExpressionValidator 控件,可以根据正则表达式定义的模式检查用户的输入。

这就是说,可以定义一个应用于用户输入的结构,该结构可以检查用户输入的结构是否匹配我们定义的结构。例如,可以定义用户输入的结构必须采用电子邮件地址或 Internet URL 的形式;如果不匹配这个定义,页面就是无效的。程序清单 6-12 说明了如何验证输入到文本框中的内容,从而确保采用的是电子邮件地址的形式。

程序清单 6-12 确保文本框中的值是电子邮件地址

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat="server"
    Text="You must enter an email address"
    ValidationExpression="\w+([-+.']\w+)*@\w+([-.\w+)*\.\w+([-.\w+)*"
    ControlToValidate="TextBox1"></asp:RegularExpressionValidator>
```

与其他验证服务器控件一样,RegularExpressionValidator 控件使用 ControlToValidate 属性来将自己绑定到 TextBox 控件。如果验证测试失败,该控件的 Text 属性就把错误消息显示到屏幕上。该验证控件的独特属性是 ValidationExpression,它是一个字符串值,该字符串值是可以应用于输入值的正则表达式。

Visual Studio 2012 使用正则表达式编辑器,使正则表达式的使用更加容易。这个编辑器提供了几个能够应用于 RegularExpressionValidator 的常用正则表达式。要打开这个编辑器,应在设计视图中操作页面。确保在设计视图中突出显示 RegularExpressionValidator1 服务器控件,以查看该控件的属性。在 Visual Studio 的 Property 窗口中,单击 ValidationExpression 属性旁边的按钮,启动正则表达式编辑器,该编辑器如图 6-7 所示。

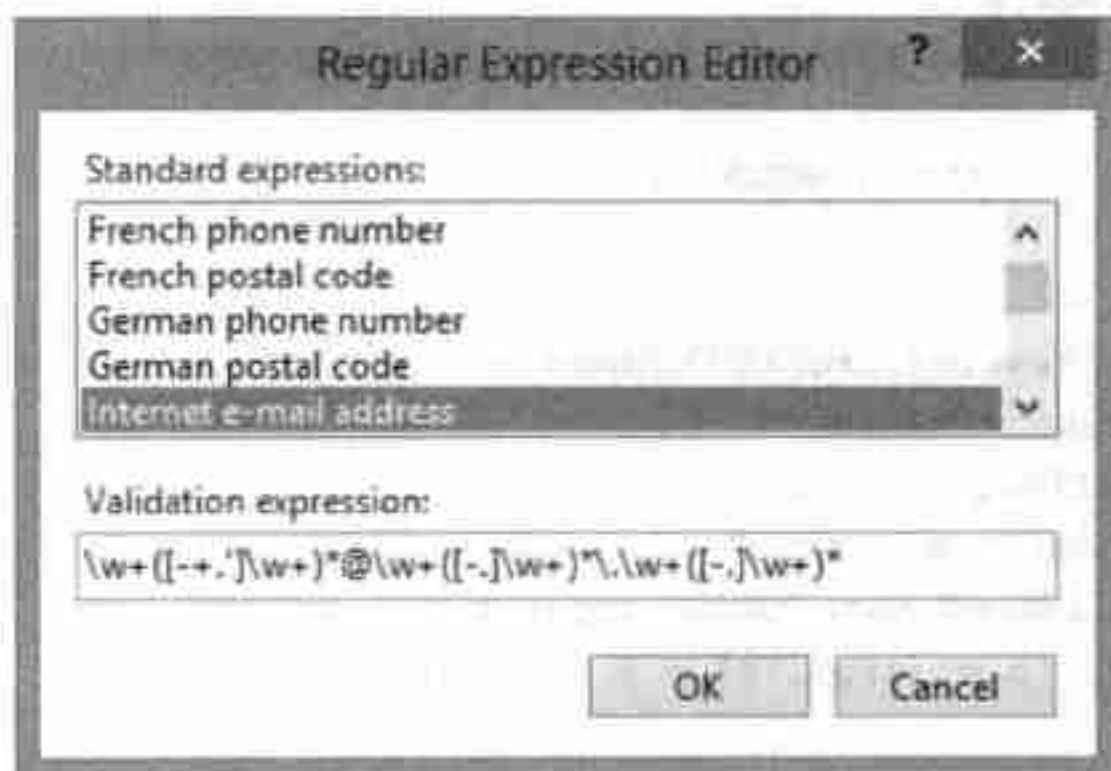


图 6-7

在这个编辑器中,可以看到用于电子邮件地址、Internet URL、邮政编码、电话号码和社保账号的正则表达式。



除了使用正则表达式编辑器生成这些复杂的正则表达式字符串之外，还可以在 Internet 站点 [RegExLib\(www.regexlib.com\)](http://www.regexlib.com) 上找到许多其他正则表达式。

6.3.7 CustomValidator 服务器控件

前面介绍了许多可用的验证控件。在许多情况下，这些验证控件包含许多可应用于 Web 窗体的验证规则。但是，有时这些控件并不合适，它们并没有提供我们需要的功能。此时就应使用 CustomValidator 控件。

CustomValidator 控件允许建立自己的客户端或服务器端验证，然后把它们应用于 Web 窗体。这样操作就可以对值或在数据层(如数据库)执行的计算进行验证，或者根据某些算术有效性规则来验证用户的输入(例如，确定数字是奇数还是偶数)。使用 CustomValidator 控件可以做许多工作。

1. 使用客户端验证

CustomValidator 控件的一个优秀功能是可以方便地提供定制的客户验证。许多开发人员都在应用程序中使用自己的 JavaScript 函数集合，而使用 CustomValidator 控件是实现这些功能的简便方式，且对函数的修改最少。例如，在大多数情况下，JavaScript 函数签名必须更新，以遵循特定的签名。

例如，有一个简单的窗体，其中要求终端用户输入一个数字。这个窗体使用 CustomValidator 控件对用户的输入执行定制的客户验证，以确保所提供的数字可以被 5 整除，如程序清单 6-13 所示。

程序清单 6-13 使用 CustomValidator 控件执行客户端验证

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Button1_Click(Object sender, EventArgs e)
    {
        Label1.Text = "VALID NUMBER!";
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>CustomValidator</title>
    <script type="text/javascript">
        function validateNumber(oSrc, args) {
            args.IsValid = (args.Value % 5 == 0);
        }
    </script>
</head>
<body>
    <form id="form1" runat="server">
        <p>
            Number:
            <asp:TextBox ID="TextBox1"
                runat="server"></asp:TextBox>
```

```

        &nbsp;
        <asp:CustomValidator ID="CustomValidator1"
            runat="server" ControlToValidate="TextBox1"
            Text="Number must be divisible by 5"
            ClientValidationFunction="validateNumber">
        </asp:CustomValidator>
    </p>
    <p>
        <asp:Button ID="Button1" OnClick="Button1_Click"
            runat="server" Text="Button"></asp:Button>
    </p>
    <p>
        <asp:Label ID="Label1" runat="server"></asp:Label>
    </p>
</form>
</body>
</html>

```

在这个 Web 窗体中执行了两个操作。首先，这个简单的窗体只有一个文本框要求用户输入数据。用户单击按钮时就会触发 `Button1_Click` 事件，然后该事件会填充页面上的 `Label1` 控件。只有执行了所有的验证，而且用户的输入通过了这些测试之后，才会执行这个简单的操作。

在这个页面上有一个不同的地方，即在 `<head>` 部分包含第二个 `<script>` 块。这是定制的 JavaScript。注意，现在 Visual Studio 2012 对这类构造代码非常友好，即使在页面的设计和源代码视图之间切换也是如此——Visual Studio 2008 之前的版本在这方面处理得就不太好。JavaScript 函数 `validateNumber` 如下所示：

```

<script type="text/javascript">
    function validateNumber(oSrc, args) {
        args.IsValid = (args.Value % 5 == 0);
    }
</script>

```

第二个 `<script>` 部分是客户端 JavaScript，在对文本框中输入的信息进行验证检查时，`CustomValidator` 控件会使用这些 JavaScript。与服务器端的验证方法一样，客户端验证方法需要两个参数：`source` 和 `arguments`。`source` 参数是对验证控件的引用。`arguments` 参数是一个对象，它包含两个属性 `args.IsValid` 和 `args.Value`。这里的 JavaScript 函数将使用 `args.IsValid` 属性，并根据验证检查的结果把它设置为 `true` 或 `false`。在本例中，检查用户输入的数字(`args.Value`)是否能被 5 整除。返回的 Boolean 值会被赋给 `args.IsValid` 属性，然后由 `CustomValidator` 控件使用。

与前面的其他控件一样，`CustomValidator` 控件也使用 `ControlToValidate` 属性把自己关联到页面的某个元素上。这里令人感兴趣的属性是 `ClientValidationFunction`，给这个属性提供的字符串值是触发 `CustomValidator` 控件时验证检查所要使用的客户端函数名。这里是 `validateNumber` 函数：

```
ClientValidationFunction="validateNumber"
```



验证检查函数可以在代码中设置，或者使用属性窗口设置。

运行这个页面，输入一个无效的数字，结果如图 6-8 所示。

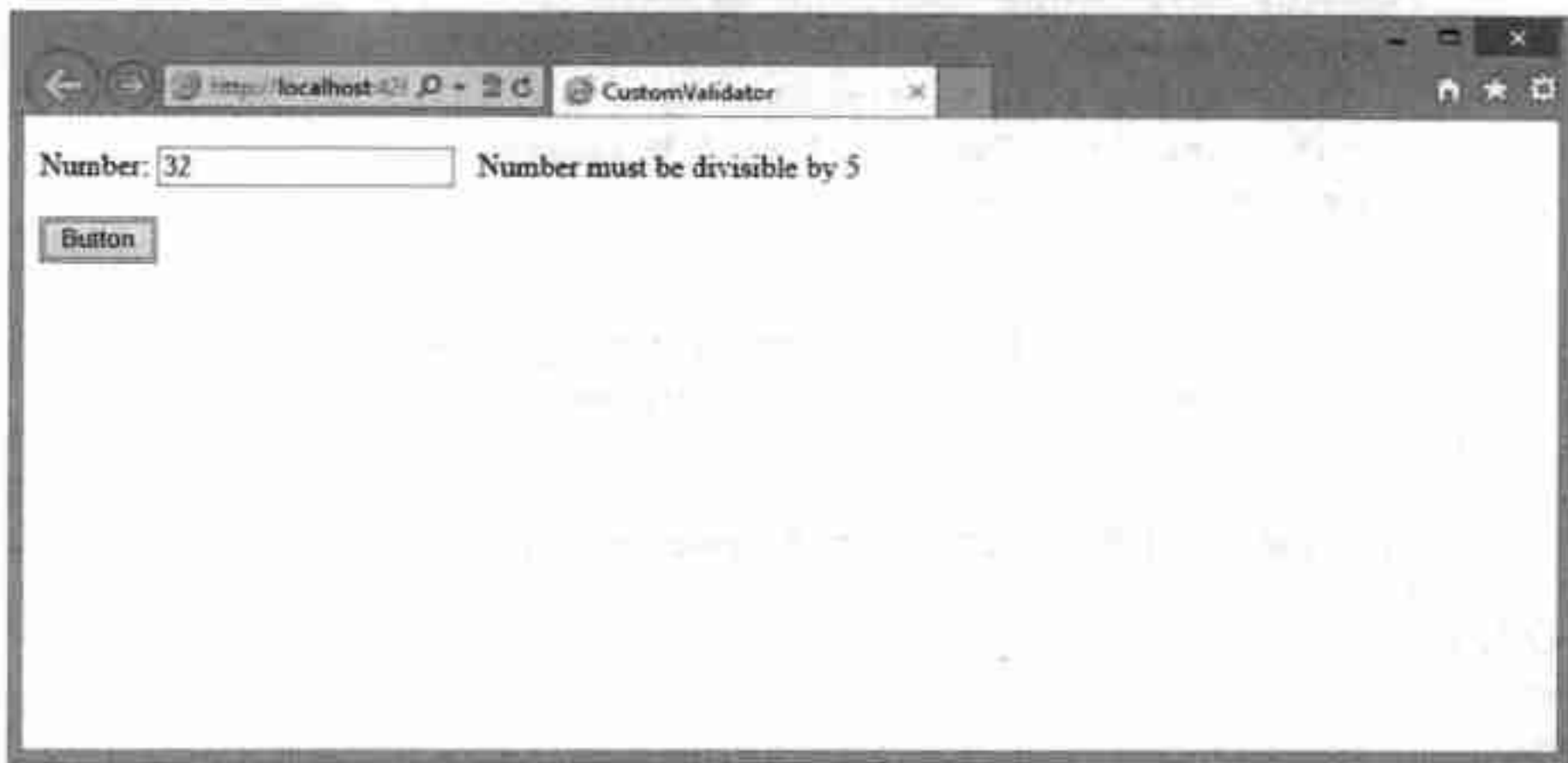


图 6-8

2. 使用服务器端验证

下面把相同的验证检查从客户端移到服务器上。CustomValidator 控件也允许进行定制的服务器端验证。创建服务器端验证与创建客户端验证一样简单。

如果创建自己的服务器端验证，就可以使它们达到应用程序需要的复杂程度。例如，如果要根据 XML 文件、数据库或其他文件中的动态值来检查用户的输入，就应使用 CustomValidator 进行服务器端验证。

要使用 CustomValidator 控件进行定制的服务器端验证，可以使用创建客户端验证时的相同例子。现在创建服务器端验证，确保用户输入的数字可以被 5 整除，如程序清单 6-14 所示。

程序清单 6-14 使用 CustomValidator 控件进行服务器端验证

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Button1_Click(Object sender, EventArgs e)
    {
        if(Page.IsValid)
        {
            Label1.Text = "VALID ENTRY!";
        }
    }
    void ValidateNumber(object source, ServerValidateEventArgs args)
    {
        try
        {
            int num = int.Parse(args.Value);
            args.IsValid = ((num % 5) == 0);
        }
        catch(Exception ex)
        {
            args.IsValid = false;
        }
    }
}
```



```

    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>CustomValidator</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <p>
                Number:
                <asp:TextBox ID="TextBox1"
                    runat="server"></asp:TextBox>
                &nbsp;
                <asp:CustomValidator ID="CustomValidator1"
                    runat="server" ControlToValidate="TextBox1"
                    Text="Number must be divisible by 5"
                    OnServerValidate="ValidateNumber"></asp:CustomValidator>
            </p>
            <p>
                <asp:Button ID="Button1" OnClick="Button1_Click"
                    runat="server" Text="Button"></asp:Button>
            </p>
            <p>
                <asp:Label ID="Label1" runat="server"></asp:Label>
            </p>
        </div>
    </form>
</body>
</html>

```

在上面的代码中没有使用客户端 JavaScript 函数，而使用了服务器端函数 `ValidateNumber`。服务器端函数的两个参数与客户端函数相同。与用于处理 `CustomValidator` 控件的其他函数一样，`ValidateNumber` 函数也必须使用 `ServerValidateEventArgs` 对象作为参数，以获得传递给函数的数据进行验证检查。`ValidateNumber` 函数本身没有什么特殊之处，它只是检查所提供的数字是否能被 5 整除。

在用于处理 `CustomValidator` 控件的定制函数中，通过 `args.Value` 对象获得了窗体元素中的值。然后，根据验证检查的结果把 `args.IsValid` 属性设置为 `True` 或 `False`。从前面的例子中可以看出，如果数字不能被 5 整除，那么 `args.IsValid` 就设置为 `False`，并抛出异常(如果在窗体元素中输入了字符串值，就会抛出异常)。构建好定制函数后，下一步就是把它应用于 `CustomValidator` 控件，如下所示：

```

<asp:CustomValidator ID="CustomValidator1"
    runat="server" ControlToValidate="TextBox1"
    Text="Number must be divisible by 5"
    OnServerValidate="ValidateNumber"></asp:CustomValidator>

```

为了把 `CustomValidator` 控件和服务端代码中的函数关联起来，只需使用 `OnServerValidate` 属性。设置这个属性的值为函数的名称，这里是 `ValidateNumber`。

运行这个例子会把页面回送给服务器，并且根据 `ValidateNumber` 函数进行验证检查。在服务器

上节将重新加载页面，并调用 Page_Load 事件。在程序清单 6-14 中会检查页面是否有效，使用 Page.IsValid 属性完成该操作：

```
if (Page.IsValid)
{
    Label1.Text = "VALID ENTRY!";
}
```

3. 结合使用客户端和服务端验证

如本章前面所述，必须考虑窗体的安全性，确保从窗体中收集的数据是有效的。因此，在决定使用客户端验证时(如程序清单 6-13 所示)，还应采取措施，重新把客户端函数构建为服务器端函数。之后，把 CustomValidator 控件与客户端和服务端函数关联起来。在程序清单 6-13 和 6-14 所示的数字检查验证中，可以在页面中使用这两个验证函数，把 CustomValidator 控件改为指向这两个函数，如程序清单 6-15 所示。

程序清单 6-15 带有客户端和服务端验证的 CustomValidator 控件

```
<asp:CustomValidator ID="CustomValidator1"
    runat="server" ControlToValidate="TextBox1"
    Text="Number must be divisible by 5"
    OnServerValidate="ValidateNumber"
    ClientValidationFunction="validateNumber"></asp:CustomValidator>
```

从这个例子中可以看出，同时使用 ClientValidationFunction 和 OnServerValidate 属性是很简单的。

6.3.8 ValidationSummary 服务器控件

ValidationSummary 控件并不对 Web 窗体中输入的内容进行验证。它是一个报告控件，供页面上的其他验证控件使用。ValidationSummary 验证控件可以用于汇总页面上所有的验证错误，而不是把这些验证错误留给每个验证控件。

可以把这个功能用于有完整验证过程的大型窗体。以易于辨识的方式向终端用户报告所有的验证错误是非常友好的。这些错误消息可以显示在列表、项目列表或段落中。

默认情况下，ValidationSummary 控件把验证错误列表显示为项目符号列表，如程序清单 6-16 所示。

程序清单 6-16 ValidationSummary 控件的部分页面示例

```
<p>
    First name
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    &nbsp;
    <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
        runat="server" ErrorMessage="You must enter your first name"
        ControlToValidate="TextBox1"></asp:RequiredFieldValidator>
</p>
<p>
    Last name
    <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
    &nbsp;
```

```

<asp:RequiredFieldValidator ID="RequiredFieldValidator2"
    runat="server" ErrorMessage="You must enter your last name"
    ControlToValidate="TextBox2"></asp:RequiredFieldValidator>
</p>
<p>
    <asp:Button ID="Button1" OnClick="Button1_Click" runat="server"
        Text="Submit"></asp:Button>
</p>
<p>
    <asp:ValidationSummary ID="ValidationSummary1" runat="server"
        HeaderText="You received the following errors:"></asp:ValidationSummary>
</p>
<p>
    <asp:Label ID="Label1" runat="server"></asp:Label>
</p>

```

这个例子要求终端用户输入名和姓。窗体上的每个文本框都有一个关联的 `RequiredFieldValidator` 控件。在编译并运行页面时，如果用户单击 `Submit` 按钮时未在文本框中输入值，就会引发两个验证错误。结果如图 6-9 所示。

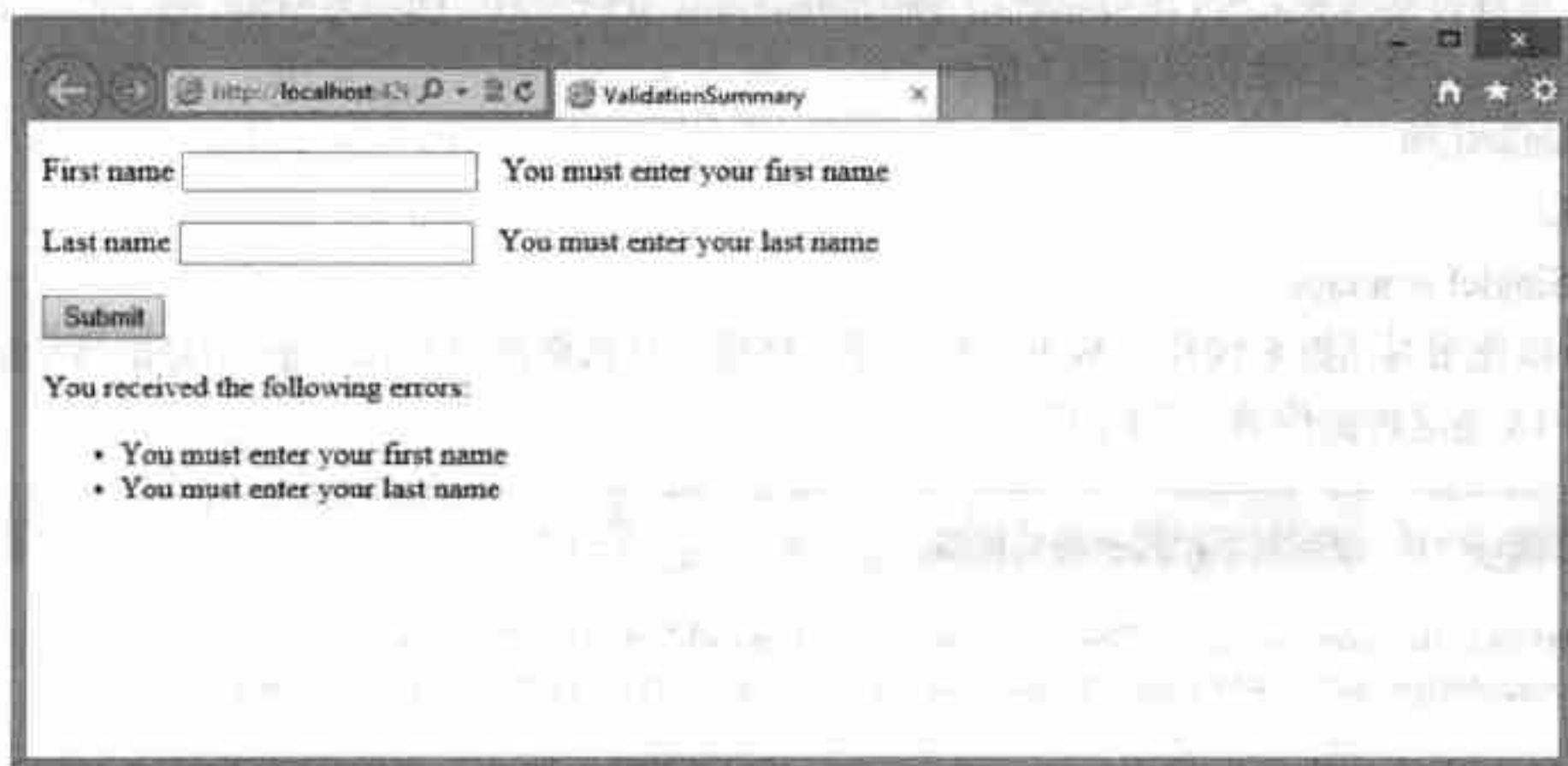


图 6-9

与前面窗体上验证控件的例子一样，这些验证错误也显示在各个文本框的旁边。但是，`ValidationSummary` 控件还会在 Web 窗体的控件位置把这些验证错误显示为红色的项目符号列表。在大多数情况下，我们都不希望这些错误在页面上向终端用户显示两次。而使用验证控件的 `Text` 属性和本章一直在使用的 `ErrorMessage` 属性可以改变这种情况。`Text` 属性是显示在验证控件旁边的文本值，`ErrorMessage` 属性是显示在项目符号列表中的文本值。该方法详见程序清单 6-17。

程序清单 6-17 使用验证控件的 `Text` 属性

```

<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
    runat="server" ErrorMessage="You must enter your first name"
    Text="" ControlToValidate="TextBox1"></asp:RequiredFieldValidator>

```

对验证控件进行这种修改会生成如图 6-10 所示的结果。

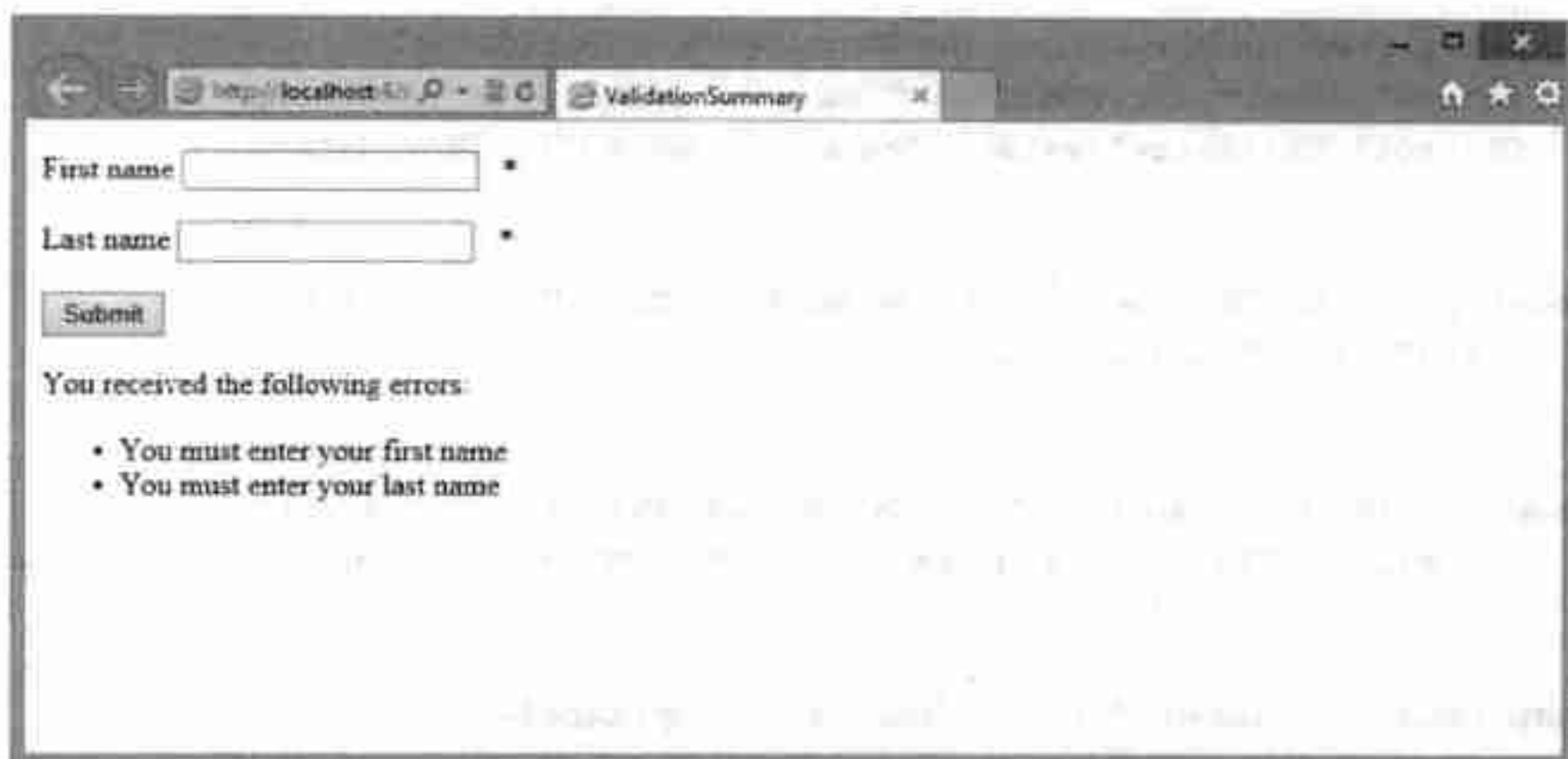


图 6-10

要获得这个结果，`ValidationSummary` 控件只需使用验证控件的 `ErrorMessage` 属性来显示验证错误即可。`Text` 属性由验证控件使用，而不是由 `ValidationSummary` 控件使用。

除了项目符号列表之外，还可以使用 `ValidationSummary` 控件的 `DisplayMode` 属性把显示的结果改为其他格式。这个属性可以有如下值：

- `BulletList`
- `List`
- `SingleParagraph`

还可以使用对话框来代替在 Web 页面上显示结果。对话框使用 JavaScript 消息框显示结果。程序清单 6-18 是这种操作的一个例子。

程序清单 6-18 使用对话框报告验证错误

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
    ShowMessageBox="true" ShowSummary="false"></asp:ValidationSummary>
```

从这个代码示例中可以看出，`ShowSummary` 属性设置为 `false`，表示验证错误的项目符号列表并没有显示在实际的 Web 页面上。但是，因为 `ShowMessageBox` 属性设置为 `true`，所以这些错误显示在消息框中，如图 6-11 所示。



图 6-11

6.4 关闭客户端验证功能

验证服务器控件会自动地为客户端提供客户端验证功能(假定请求容器可以正确处理所生成的 JavaScript), 但有时需要通过某种方式来控制这种行为。

可以关闭这些控件的客户端验证功能, 使它们不再独立地把客户端验证功能发送给请求者。例如, 无论请求容器提供了什么功能, 都把所有的验证放在服务器上。可以采用两种方式关闭这个功能。

第一种方式是控件级别的方式。每个验证服务器控件都有 `EnableClientScript` 属性, 它在默认时设置为 `true`, 但把它设置为 `false` 将阻止控件发出用于在客户端上执行验证的 JavaScript 函数, 以使在服务器上进行验证检查。这个属性的用法可参见程序清单 6-19。

程序清单 6-19 禁用验证控件中的客户端验证功能

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
    runat="server" ErrorMessage="You must enter your first name"
    Text="*" ControlToValidate="TextBox1" EnableClientScript="false">
</asp:RequiredFieldValidator>
```

还可以采用编程方式删除验证控件的客户端验证功能, 如程序清单 6-21 所示。

程序清单 6-20 采用编程方式删除客户端验证功能

```
protected void Page_Load(Object sender, EventArgs e)
{
    RequiredFieldValidator1.EnableClientScript = false;
    RequiredFieldValidator2.EnableClientScript = false;
}
```

另一种方式是在 `Page_Load` 事件中关闭页面上的所有验证控件的客户端脚本功能。如果要动态地确定不允许进行客户端验证, 使用这种方式会比较好。这种方式的用法如程序清单 6-21 所示。

程序清单 6-21 在 Page_Load 事件中禁用所有的客户端验证功能

```
protected void Page_Load(Object sender, EventArgs e)
{
    foreach(BaseValidator bv in Page.Validators)
    {
        bv.EnableClientScript = false;
    }
}
```

这个 `for each` 循环首先找到 ASP.NET 页面包含的验证器中的每个 `BaseValidator` 对象, 然后关闭页面上每个验证控件的客户端验证功能。

6.5 为错误通知使用图像和声音

前面为验证服务器控件发出的错误通知显示了简单的文本消息。在大多数情况下，这样操作已经足够，即显示一些简单的文本消息，以通知终端用户他们在窗体中输入的数据没有通过验证检查。

验证控件的一个有趣之处在于，可以显示的不仅仅是文本，还可以为错误通知使用图像和声音。

要为错误通知使用图像，需要使用验证控件的 `Text` 属性。可以把相应的 HTML 作为这个属性的值，如程序清单 6-22 所示。

程序清单 6-22 为错误通知使用图像

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
    runat="server" Text="<img src='error.jpg' alt='Error!' />"
    ControlToValidate="TextBox1" EnableClientScript="true">
</asp:RequiredFieldValidator>
```

从这个例子中可以看出，并没有在 Web 页面上输出文本，因为 `Text` 属性的值是一个 HTML 字符串。这个 HTML 字符串用于显示一幅图像。注意单引号和双引号的用法，这样在浏览器中生成页面时才不会出错。这个例子生成的结果如图 6-12 所示。



图 6-12

另一个有趣之处是在终端用户出错时添加带声音的通知。方法与为错误通知显示图像一样。程序清单 6-23 是带声音的错误通知的一个例子。

程序清单 6-23 为错误通知使用声音

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
    runat="server" ControlToValidate="TextBox1" EnableClientScript="false"
    Text="<audio controls='' src='C:\Windows\Media\tada.wav'
    autoplay='autoplay'></audio>"
</asp:RequiredFieldValidator>
```

在 `C:\Windows\Media` 目录下有许多 Windows 系统声音文件。在这个例子中，`Text` 属性使用 `<audio>` 元素把声音文件放在 Web 窗体上(只能用于兼容 HTML5 的浏览器)。只有当终端用户触发验

证控件时, 该声音文件才会播放。



引用声音文件(例如上述声音文件)时, 必须确保 Web 应用程序有监听声音的适当许可。建议把声音文件放在应用程序中或内容发现网络(Content Delivery Network, CDN)中, 还要注意 audio 元素不支持 IE 中的 .wav 文件。但是其他几个浏览器支持 .wav 文件。在 Internet 上搜索“HTML5 支持的音频文件格式”, 确定浏览器支持什么格式。

在为错误通知使用声音时, 必须禁用控件的客户端脚本功能。如果不这么做, 在浏览器中加载页面时, 无论是否触发验证错误, 都会播放声音。

6.6 使用验证组

在许多情况下, 开发人员希望在一个页面上放置多个窗体。我们常常将不同的验证控件赋予页面上的两个不同窗体。但此时会出现预料不到的情形。例如, 当终端用户提交一个窗体时, 会触发另一个窗体上的验证控件(即使用户没有操作该窗体), 从而阻止第一个窗体提交。有时, 开发人员希望用不同的方式处理一个窗体上的不同验证控件。

例如, 图 6-13 显示了 .NET Valley User Group 的一个基本页面, 它包含两个窗体。

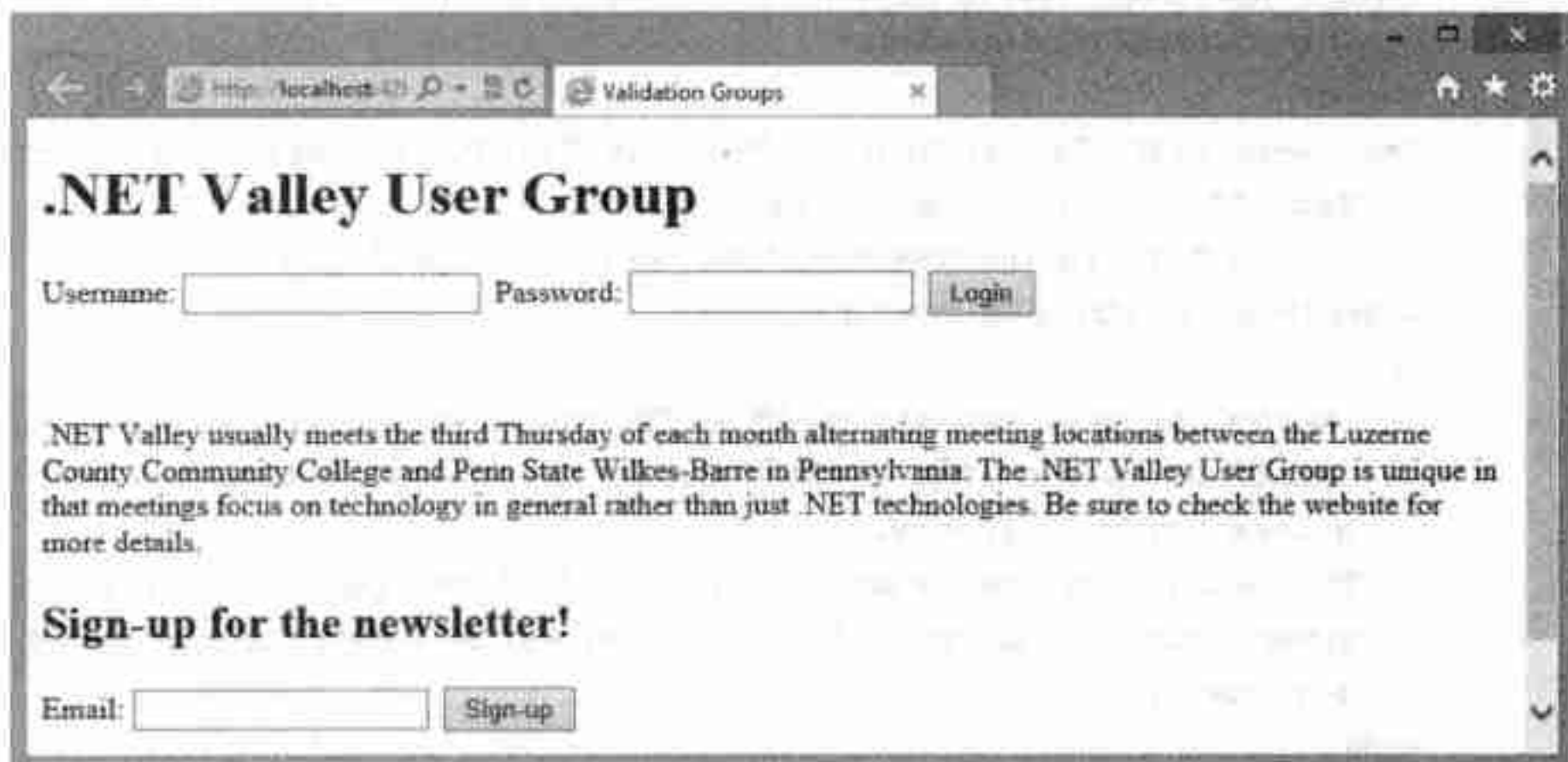


图 6-13

其中一个窗体用于站点的成员提供用户名和密码, 以登录站点的 Members Only 部分。页面上的第二个窗体用于用户签署用户组的新闻通讯。每个窗体都有自己的按钮和与之关联的一些验证控件。当有人提交其中一个窗体的信息时, 就会出问题。例如, 如果组中有一个成员提供了用户名和密码, 并单击了 Login 按钮。新闻通讯窗体上的验证控件就会触发, 因为在该窗体上没有提供电子邮件地址。如果对新闻通讯感兴趣的人在最后一个文本框中输入了电子邮件地址, 并单击了 Sign-up 按钮, 第一个窗体上的验证控件就会触发, 因为没有在该窗体中输入用户名和密码。

ASP.NET Web 控件提供了 ValidationGroup 属性, 可以把验证控件放在不同的组中。在终端用户单击页面上的一个按钮时, 该属性可以只激活需要的验证控件。程序清单 6-24 中的例子把用户组页

面上的验证控件放在不同的桶(bucket)中。

程序清单 6-24 使用 ValidationGroup 属性

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Validation Groups</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <h1>.NET Valley User Group</h1>
            Username:
                <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>&nbsp; Password:
                <asp:TextBox ID="TextBox2" runat="server"
                    TextMode="Password"></asp:TextBox>&nbsp;
                <asp:Button ID="Button1" runat="server" Text="Login"
                    ValidationGroup="Login" />
            <br />
            <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
                Text="* You must submit a username!"
                ControlToValidate="TextBox1" ValidationGroup="Login">
            </asp:RequiredFieldValidator>
            <br />
            <asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server"
                Text="* You must submit a password!"
                ControlToValidate="TextBox2" ValidationGroup="Login">
            </asp:RequiredFieldValidator>
            <p>
                .NET Valley usually meets the third Thursday of each month alternating meeting
                locations between the Luzerne County Community College and Penn State
                Wilkes-Barre in Pennsylvania.
                The .NET Valley User Group is unique in that meetings focus on technology in
                general rather than just .NET technologies. Be sure to check the website for
                more details.<br />
            </p>
            <h2>Sign-up for the newsletter!</h2>
            Email:
                <asp:TextBox ID="TextBox3" runat="server"></asp:TextBox>&nbsp;
                <asp:Button ID="Button2" runat="server" Text="Sign-up"
                    ValidationGroup="Newsletter" />&nbsp;
            <br />
            <asp:RegularExpressionValidator ID="RegularExpressionValidator1"
                runat="server"
                Text="* You must submit a correctly formatted email address!"
                ControlToValidate="TextBox3" ValidationGroup="Newsletter"
                ValidationExpression="\w+([-+.] \w+)*@ \w+([-+.] \w+)*\.\w+([-+.] \w+)*">
            </asp:RegularExpressionValidator>
```

```

<br />
<asp:RequiredFieldValidator ID="RequiredFieldValidator3" runat="server"
    Text="* You forgot your email address!"
    ControlToValidate="TextBox3" ValidationGroup="Newsletter">
</asp:RequiredFieldValidator>
</div>
</form>
</body>
</html>

```

ValidationGroup 属性为 String 值。核心服务器控件也有这个属性，因为像按钮单击这样的事件必须与特定的验证组相关联。

在这个例子中，每个按钮都有各自不同的验证组值。窗体上的第一个按钮把 Login 用作值，第二个按钮把 Newsletter 用作值。每个验证控件都与一个验证组相关联。因此，在终端用户单击页面上的 Login 按钮时，ASP.NET 就知道应只处理有相同验证组名的验证服务器控件。ASP.NET 会忽略与其他验证组关联的验证控件。

进行了此项改进后，就可以包含多组验证规则，根据需要使用相应的验证规则，如图 6-14 所示。



图 6-14

验证控件新增的另一个强大功能是 SetFocusOnError 属性。该属性为 Boolean 值，如果提交窗体时抛出验证错误，该属性就使页面上接收错误的窗体元素获得焦点。SetFocusOnError 属性的用法如下所示：

```

<asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
    Text="* You must submit a username!" SetFocusOnError="true"
    ControlToValidate="TextBox1" ValidationGroup="Login">
</asp:RequiredFieldValidator>

```

如果因为终端用户没有能在 TextBox1 中输入值而使得 RequiredFieldValidator1 抛出错误，页面就会重新绘制，使 TextBox1 获得焦点，如图 6-15 所示。



图 6-15

注意，如果页面上有多个验证控件的 `SetFocusOnError` 属性设置为 `true`，而且出现了多个验证错误，那么存在验证错误的最上面的窗体元素就会获得焦点。在前面的例子中，如果 Username 文本框(`TextBox1`)和 Password 文本框(`TextBox2`)都有与之关联的验证错误，页面就使 Username 文本框获得焦点，因为它是窗体上有错误的第一个控件。

6.7 本章小结

在处理窗体时，验证控件是非常有用的工具。它们被放在一个易于使用的软件包中，与.NET中的大多数对象一样，我们可以很轻易地根据自己的需求来定制它们的外观和行为。

在应用程序中，窗体的作用是收集数据，但如果数据是无效的，数据的收集就没有意义。这就是说，必须通过一系列不同的控件(即验证服务器控件)，建立可以在窗体上实现的验证规则。

本章详细介绍了各种验证控件，包括：

- `RequiredFieldValidator`
- `CompareValidator`
- `RangeValidator`
- `RegularExpressionValidator`
- `CustomValidator`
- `DynamicValidator`
- `ValidationSummary`

除了介绍这些基本验证控件之外，本章还讨论了如何应用客户端和服务端验证，以及ASP.NET 4.5 引入的隐含验证功能及其对客户端验证的影响。

第7章

用户控件和服务器控件

本章要点

- 创建、交互和加载用户控件
- 使用服务器控件
- 为开发人员优化服务器控件

在类似.NET这样的面向对象环境中,把代码封装到作用单一的小型可重用对象中是开发健壮系统的关键操作之一。例如,如果应用程序面向的是顾客,就要考虑创建 **Customer** 对象来表示顾客。这个对象封装了顾客可以在系统中使用的所有属性和行为。其优点是,可以建立一个代码块,其他对象可以与之交互,而且只需对一段代码进行创建、调试、部署和维护。**Customer** 对象一般称为业务对象,因为它封装了系统中特定实体所需要的所有业务逻辑。

.NET 作为面向对象的架构,其中还有其他类型的可重用对象。本章将主要论述和演示如何使用 .NET Framework 为 ASP.NET 应用程序创建两种不同类型的可重用的可视化组件:用户控件和服务器控件。

- 用户控件把已有的 ASP.NET 控件封装到容器控件中,而容器控件可以很方便地在 Web 项目中重用。
- 服务器控件将原始 HTML、客户端及服务器逻辑封装到单个对象中。可以编程服务器控件,它的内容最终会显示给客户端,以便用户在 Web 页面上与其进行交互。

因为用户控件和服务器控件的主题都很庞大,每个主题都需要整本书的篇幅来介绍,所以本章不可能研究所有的可用选项,只能概述用户控件、服务器控件的建立和使用。我们将为每种控件演示一些常见的场景。阅读完本章后,读者就可以根据所学的内容开始建立这些类型的基本控件,并可以自学更深入的内容。

7.1 用户控件

用户控件是 ASP.NET 控件可视化封装最简单的形式。由于它们最简单,因此创建和使用它们也

是最简单的。用户控件实际上是把已有的服务器控件分组到容器控件中，这样就可以创建能在整个 Web 项目中使用的功能强大的对象。

7.1.1 创建用户控件

在 Visual Studio 2012 中创建用户控件是非常简单的。要创建新的用户控件，首先应在 Web 站点中添加一个新的 User Control 文件。在 Website 菜单中，选择 Add New Item 选项。打开 Add New File 对话框后，在列表中选择 Web User Control File 模板，单击 Add 按钮。注意，在将文件添加到项目中后，该文件的扩展名是 .ascx。这个扩展名告诉 ASP.NET，该文件是用户控件。如果试图把用户控件直接加载到浏览器中，ASP.NET 就会返回错误，告知这种类型的文件不能用于客户端。

如果查看用户控件的 HTML 源代码(如程序清单 7-1 所示)，就会看到它们与标准 ASP.NET Web 页面的一些有趣区别。

程序清单 7-1 Web 用户控件的文件模板

```
<%@ Control Language="C#" ClassName="Listing07_01" %>

<script runat="server">

</script>
```

首先注意，源代码使用 @Control 指令来代替标准 Web 页面使用的 @Page 指令。其次，注意与标准的 ASP.NET Web 页面不同，控件中除了 <script> 标记之外没有其他 HTML 标记。包含用户控件的 Web 页面提供基本的 HTML 标记，如 <body> 和 <form> 标记。实际上，如果试图给用户控件添加服务器端的窗体标记，在把页面传送给客户端时，ASP.NET 就会返回错误消息。该错误消息说明，Web 页面中只能有一个服务器端的窗体标记。

要把控件添加到窗体上，只需把它们从工具箱拖放到用户控件上。程序清单 7-2 显示了添加标签和按钮后的用户控件。

程序清单 7-2 给 Web 用户控件添加控件

```
<%@ Control Language="C#" ClassName="Listing07_02" %>

<script runat="server">

</script>
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
<asp:Button ID="Button1" runat="server" Text="Button" />
```

给用户控件添加了控件后，就把用户控件放在标准的 ASP.NET Web 页面上。为此，可以把文件从 Solution Explorer 拖放到 Web 页面上。

图 7-1 显示了将用户控件拖放到 Web 页面上之后的情况。

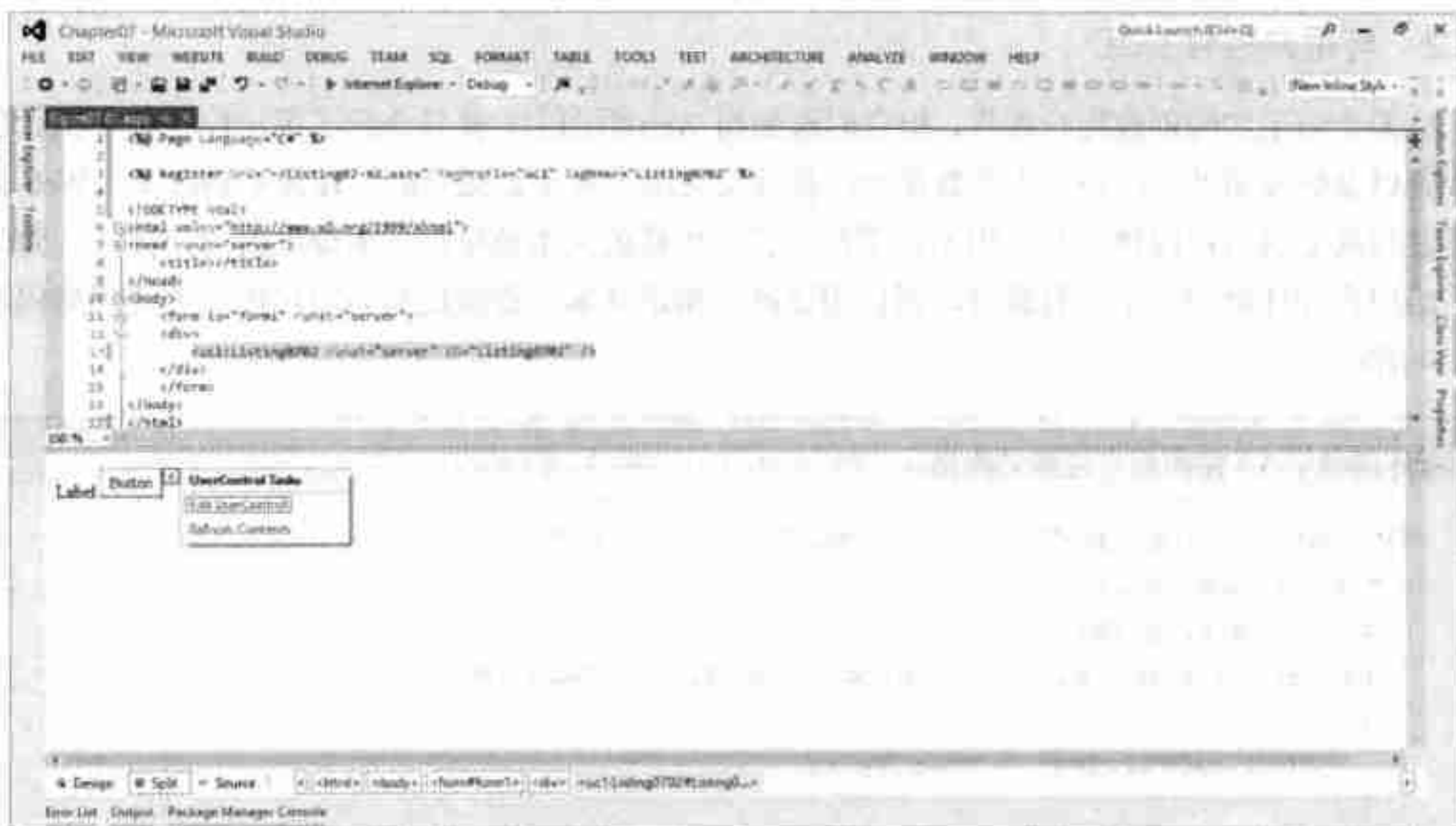


图 7-1

把用户控件放在 Web 页面上后, 在浏览器中打开该页面, 就会看到完全显示出来的 Web 页面。

用户控件完全参与到页面显示的整个过程中, 包含在用户控件中的控件与放在标准 ASP.NET Web 页面上的控件的操作完全一致。也就是说, 用户控件有自己的页面事件(如 Init、Load 和 PreRender), 它们在处理页面时执行。另外, 用户控件中的控件也会像通常那样触发事件。程序清单 7-3 演示了如何使用用户控件的 Page_Load 事件填充标签以及处理按钮的 Click 事件。

程序清单 7-3 在用户控件中创建控件事件

```
<%@ Control Language="C#" ClassName="Listing07_03" %>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        this.Label1.Text = "The quick brown fox jumped over the lazy dog";
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        this.Label1.Text = "The quick brown fox clicked the button on the page";
    }
</script>
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
<asp:Button ID="Button1" runat="server" Text="Button" />
```

在显示 Web 页面时, 标签的文本在用户控件加载时就会改变; 在单击页面的底部时, 标签的文本也会改变。实际上, 在这两个事件中放置断点, 就可以看到在执行页面时, ASP.NET 在用户控件的代码内部中断。

7.1.2 与用户控件交互

前面介绍了如何创建用户控件，把它们添加到 Web 页面上，并且介绍了用户控件如何执行自己的代码以及触发事件。但是，大多数用户控件在父页面上并不是孤立的，在许多情况下，Web 页面都需要与其上的用户控件交互。例如，要在标签上加载的文本必须由页面提供给用户控件。为此，只需给用户控件添加一个公有属性，再使用该属性指定文本。修改过的该用户控件的代码如程序清单 7-4 所示。

程序清单 7-4 使用用户控件的属性

```
<%@ Control Language="C#" ClassName="Listing07_04" %>
<script runat="server">
    public string Text { get; set; }
    protected void Page_Load(object sender, EventArgs e)
    {
        this.Label1.Text = this.Text;
    }
</script>
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
```

修改了用户控件后，只需在 Web 页面上填充该属性即可。程序清单 7-5 演示了如何在代码中设置 Text 属性。

程序清单 7-5 在 Web 主页上填充用户控件的属性

```
protected void Page_Load(Object sender, EventArgs e)
{
    Listing0704.Text = "The quick brown fox jumped over the lazy dog";
}
```

注意，用户控件的公有属性还可以由 Property Browser 提供，因此也可以使用 Property Browser 设置用户控件的属性。

用户控件是在 ASP.NET 中创建强大的、可重用组件的简单方式，很容易使用内置模板创建它们。由于用户控件参与页面处理的整个过程，因此可以创建与页面甚至页面上其他控件交互的控件。

7.1.3 动态加载用户控件

还可以在运行时动态创建用户控件，并将其添加到 Web 窗体上。ASP.NET 的 Page 对象有 LoadControl 方法，为该方法提供要加载的用户控件的虚拟路径，就可以在运行期间加载用户控件。这个方法返回一个一般的 Control 对象，然后就可以把它添加到页面的 Controls 集合中。程序清单 7-6 演示了如何使用 LoadControl 方法将用户控件动态添加到 Web 页面上。

程序清单 7-6 动态添加用户控件

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
```



```

    {
        Control myForm = Page.FindControl("Form1");
        Control cl = LoadControl("Listing07-04.ascx");
        myForm.Controls.Add(cl);
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Dynamically adding a user control</title>
</head>
<body>
    <form id="form1" runat="server">
        </form>
</body>
</html>

```

要把用户控件添加到页面上, 首先需要使用 `FindControl` 方法定位页面的窗体控件。如果用户控件包含的 ASP.NET 控件显示窗体元素, 如按钮或文本框, 这个用户控件就必须添加到该窗体元素的 `Controls` 集合中。



可以把包含某些 ASP.NET 元素(如标签、超链接或图像)的用户控件直接添加到 `Page` 对象的 `Controls` 集合中, 但把它们添加到窗体对象中更容易保持一致。在 `Page` 对象的 `Controls` 集合中添加必须包含在窗体对象中的控件(如 `Button` 控件), 会产生运行时分析器错误。

找到窗体后, 本例使用 `Page` 对象的 `LoadControl` 方法加载用户控件。该方法的参数是要加载的用户控件的虚拟路径, 并把加载的用户控件作为泛型 `Control` 对象返回。

最后, 将控件添加到窗体对象的 `Controls` 集合中。还可以把用户控件添加到 Web 页面上的其他容器控件中, 如 `Panel` 或 `Placeholder` 控件。



页面每次执行回送操作时, 都必须重新把控件添加到 ASP.NET 页面上。

加载了用户控件后, 还可以使用其对象模型, 这一点与处理其他控件一样。要访问用户控件的属性和方法, 应把控件从 `Control` 类型强制转换为实际的类型。为此, 还需要给页面添加 `@Reference` 指令。这会让 ASP.NET 编译用户控件, 并把它关联到 ASP.NET 页面上, 使页面知道在哪里查找用户控件的类型。程序清单 7-7 演示了如何在加载用户控件后强制转换其类型, 以访问用户控件的定制属性。这个例子加载了一个修改过的用户控件, 它包含一个 ASP.NET `TextBox` 控件和一个公有属性, 该属性可用于访问 `TextBox` 控件的 `Text` 属性。

程序清单 7-7 将用户控件强制转换回原来的类型

```

<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">

```



```

protected void Page_Load(object sender, EventArgs e)
{
    Control myForm = Page.FindControl("Form1");
    Listing07_04 cl = (Listing07_04)LoadControl("Listing07-04.ascx");
    myForm.Controls.Add(cl);
    cl.ID = "Listing07_04";
    cl.Text = "Text about our custom user control.";
}
</script>

```

注意，这个例子是把控件添加到窗体对象的 Controls 集合中后再设置 Text 属性。这个顺序非常重要，因为在页面回送后，在将控件添加到 Controls 集合中后才计算该控件的 ViewState。如果在计算控件的 ViewState 之前设置 Text 属性值(或用户控件的其他属性值)，该值将不会保存在 ViewState 中。

动态添加用户控件的另一个问题发生在使用输出高速缓存功能以高速缓存用户控件时。在这种情况下，在高速缓存控件后，LoadControl 方法不会返回控件的新实例，而会返回控件的高速缓存副本。在试图把控件强制转换为原始类型时，这就会出现异常，因为在高速缓存控件后，LoadControl 方法把控件返回为 PartialCachingControl 对象而不是其原始类型。因此，上面例子中的类型转换会抛出异常。

为了解决这个问题，只需在强制转换之前测试对象的类型即可，如程序清单 7-8 所示。

程序清单 7-8 检测高速缓存的用户控件

```

<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        Control myForm = Page.FindControl("Form1");
        Control cl = LoadControl("Listing07-04.ascx");
        myForm.Controls.Add(cl);

        if(cl is Listing07_04)
        {
            ((Listing07_04)cl).ID = "Listing07_04";
            ((Listing07_04)cl).Text = "Text about our custom user control (not cached)";
        }
        else if((cl is PartialCachingControl) &&
            ((PartialCachingControl)cl).CachedControl != null)
        {
            Listing07_04 listingControl =
                ((Listing07_04)((PartialCachingControl)cl).CachedControl);
            listingControl.ID = "Listing07_04";
            listingControl.Text = "Text about our custom user control (partially cached)";
        }
    }
</script>

```

这个例子演示了如何测试 LoadControl 方法返回的对象类型，并根据该类型设置属性。有关高速缓存的更多内容，请参阅第 22 章。

在前面演示如何动态添加用户控件的所有例子中，用户控件都是在 Page_Load 事件中添加的。但是，有时我们希望在其他事件中添加用户控件，例如在按钮的 Click 事件或 DropDownList 控件的 SelectedIndexChanged 事件中。使用这些事件动态添加用户控件有一定的困难，因为在页面每次回送时，并不总是触发这些事件。因此，需要采用一种方式跟踪添加用户控件的时间，从而可以在发生额外的页面回送时将该用户控件重新添加到 Web 页面上。

为此，一种简单的方式是使用 ASP.NET 会话跟踪将用户控件添加到 Web 页面上的时间，如程序清单 7-9 所示。

程序清单 7-9 在页面回送期间跟踪添加的用户控件

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        if((Session["Listing07-04"] == null) ||
            (!(bool)Session["Listing07-04"]))
        {
            Control myForm = Page.FindControl("Form1");
            Control c1 = LoadControl("Listing07-04.ascx");
            ((Listing07_04)c1).Text = "Loaded after first page load";
            myForm.Controls.Add(c1);
            Session["Listing07-04"] = true;
        }
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        if((Session["Listing07-04"] != null) &&
            ((bool)Session["Listing07-04"]))
        {
            Control myForm = Page.FindControl("Form1");
            Control c1 = LoadControl("Listing07-04.ascx");
            ((Listing07_04)c1).Text = "Loaded after a postback";
            myForm.Controls.Add(c1);
        }
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Button ID="Button1" runat="server" Text="Load Control"
                OnClick="Button1_Click" />
        </div>
    </form>
</body>
</html>
```



```

    </form>
</body>
</html>

```

这个例子使用一个简单的 Session 变量来跟踪用户控件是否已添加到页面上。在触发 Button1_Click 事件时, Session 变量设置为 true, 表示已添加用户控件。以后每次回送页面时, Page_Load 事件都检查 Session 变量是否设置为 true。如果是, 就把用户控件重新添加到页面上。

7.2 服务器控件

在 ASP.NET 中创建服务器控件是 ASP.NET 开发人员可以使用的最强大的工具之一。创建定制的服务器控件和扩展已有的控件都非常容易。ASP.NET 中的所有控件都派生于两个基类: System.Web.UI.WebControls.WebControl 和 System.Web.UI.ScriptControl。派生于 WebControl 类的类拥有参与页面架构所需的基本功能, 这些类具备创建显示可视化 HTML 控件所需的大多数常见功能, 并支持许多基本样式化元素, 例如 Font、Height 和 Width。由于 WebControl 类派生于 Control 类, 因此它的派生类拥有成为可设计组件的基本功能, 这意味着可以将它们添加到 Visual Studio 工具箱中, 拖放到页面设计器上, 以及在 Property Browser 中显示其属性和事件。

派生于 ScriptControl 类的控件建立在 WebControl 类提供的功能的基础之上, 并且添加了许多其他功能, 以便于使用客户端脚本库。这些类要求在控件的 PreRender 阶段把 ScriptManager 控件放在页面上, 并且派生的控件需要在 Render 事件中调用相应的 ScriptManager 方法。

7.2.1 Server Control 项目

为了方便地创建定制的服务器控件, Visual Studio 提供了两个不同的项目模板, 可创建基本的项目结构, 项目结构包含开始创建服务器控件所需的文件。图 7-2 显示了 Visual Studio 的 New Project 对话框中的 ASP.NET Server Control 和 ASP.NET AJAX Server Control 项目。

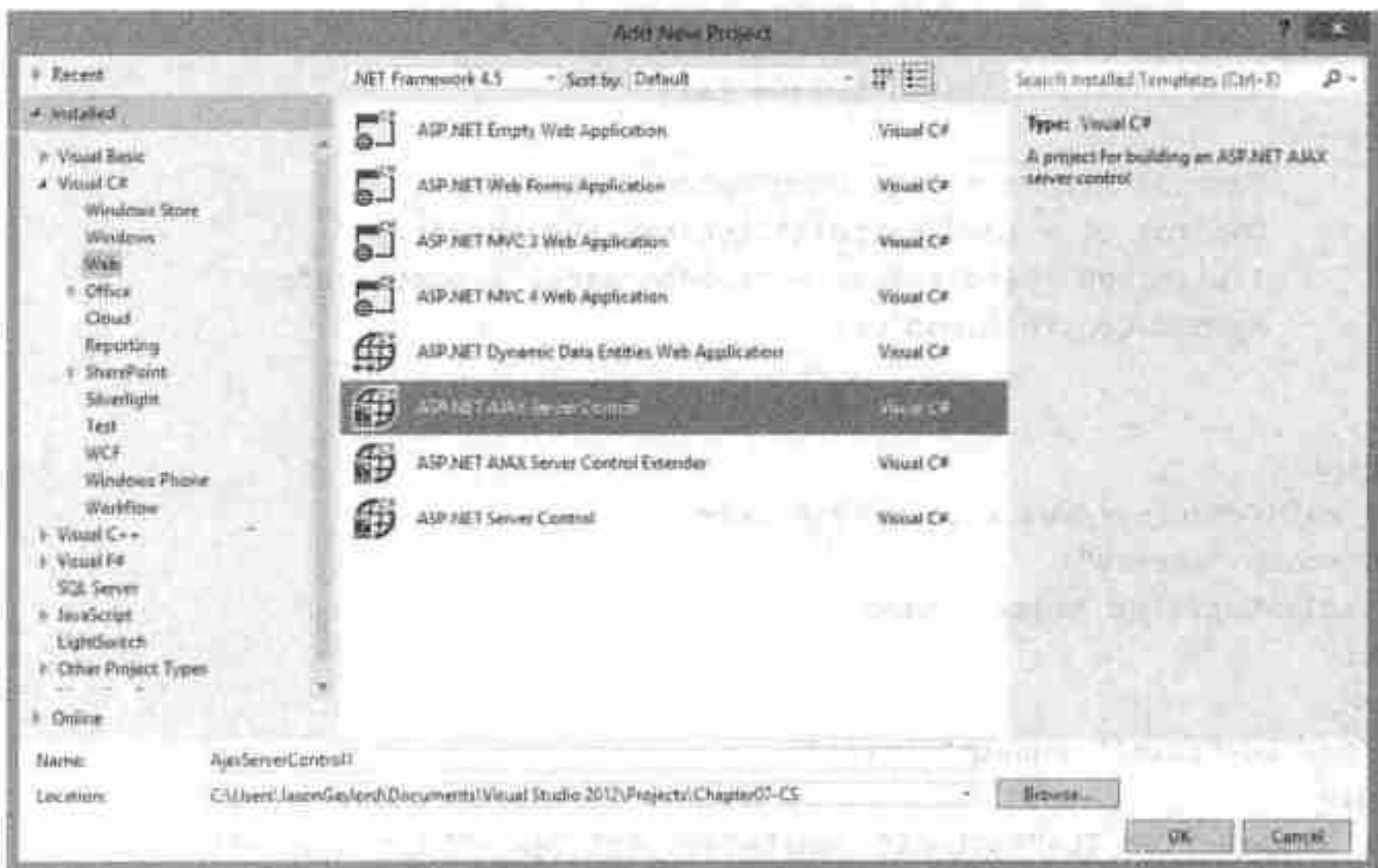


图 7-2

在默认情况下, ASP.NET Server Control 项目使用派生于 WebControl 的服务器控件类创建基本的类库项目。ASP.NET AJAX Server Control 项目也会创建基本的类库项目, 并且包含一个派生于 ScriptControl 的服务器控件类、一个 Resource 文件和一个 JavaScript 文件。创建这两种项目都会产生一个可运行的, 但基本上没有什么功能的服务器控件。

可以在 Add New Item 对话框中选择 ASP.NET Server Control 文件模板, 然后基于这个项目模板在其中添加其他的服务器控件类。注意, 这个模板与服务器控件项目中的默认模板略有不同。它使用不同的文件名方案, 并且在默认的控件模板中包含略有不同的代码。

创建新项目后, 可以为已有解决方案添加新的 Web 项目, 重建整个解决方案并打开默认的 Web 页面, 以此来测试这个控件。Visual Studio 会自动将服务器控件添加到工具箱中, 如图 7-3 所示。

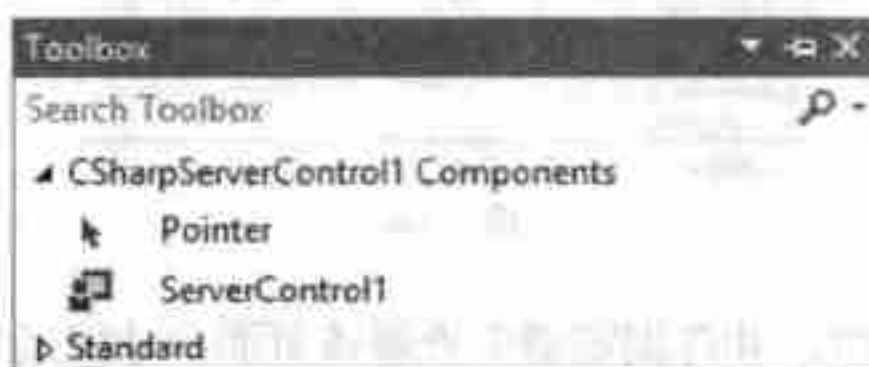


图 7-3

Visual Studio 可以把所打开解决方案中的项目包含的所有控件自动添加到工具箱中。

现在, 只需将控件拖放到 Web 窗体上, 对这个控件的引用就会添加到这个项目中, 并且这个控件也会添加到 Web 页面上。程序清单 7-10 列出了添加控件后的 Web 页面的源代码。

程序清单 7-10 在 Web 页面上添加 Web 控件库

```
<%@ Page Language="C#" %>
<%@ Register Assembly="CSharpServerControl1" Namespace="CSharpServerControl1"
    TagPrefix="ccl" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Adding a Custom Web Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <ccl:ServerControl1 ID="ServerControl1" runat="server" />
        </div>
    </form>
</body>
</html>
```

将控件拖放到 Web 窗体上之后, 在 Properties 窗口中查看它的属性。图 7-4 显示了该定制控件的属性。

此时可以看到控件已经包含了可视化控件需要的所有基本属性, 包括各种样式化和行为属性, 带有这些属性是因为该控件派生于 WebControl 类。它还继承了 WebControl 类提供的基本事件。



图 7-4

必须给 Text 属性输入一个值,并在浏览器中查看该页面,才能确保控件正常工作。图 7-5 显示了把 Text 属性设置为“Hello World!”时的页面。

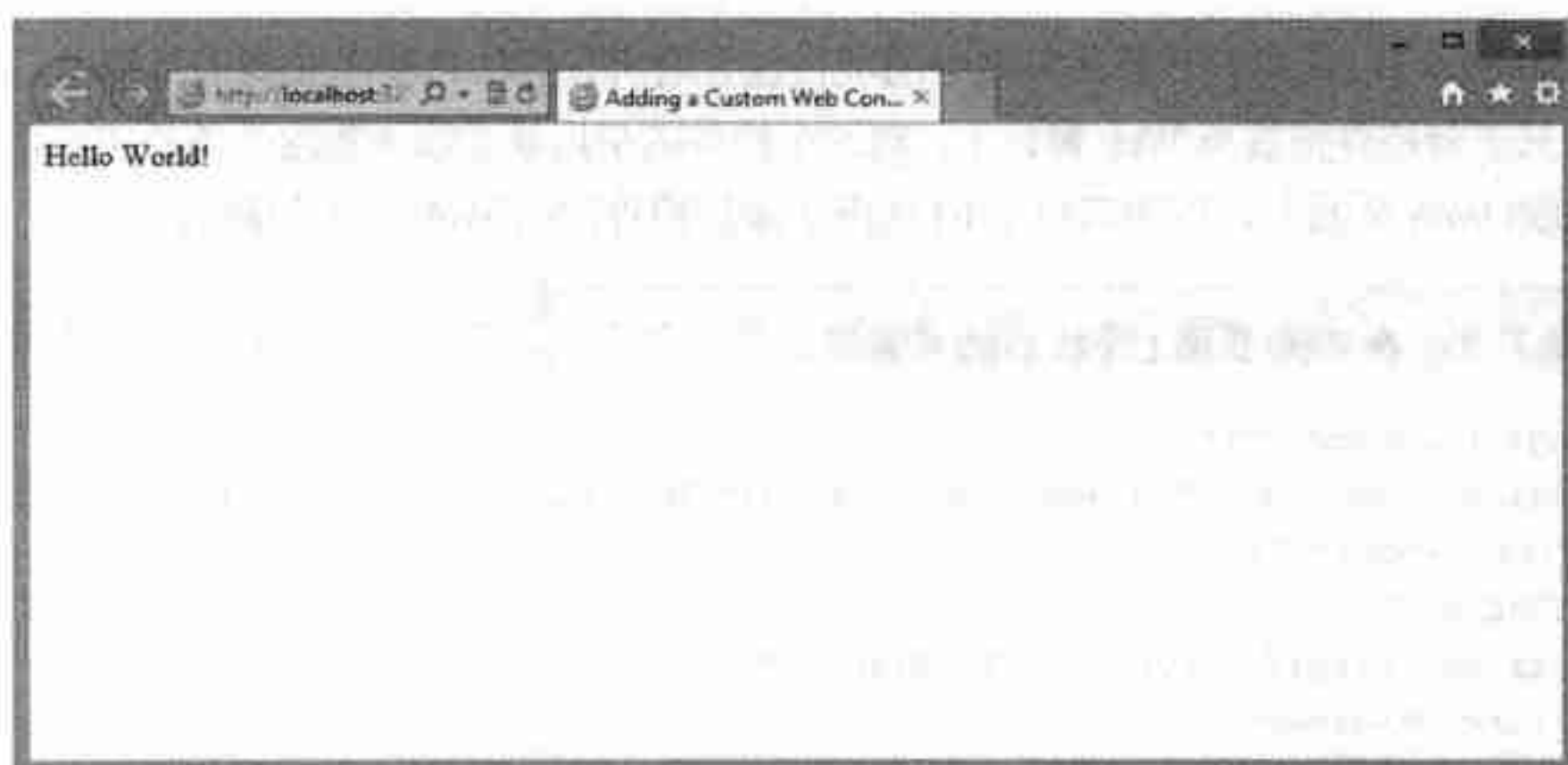


图 7-5

与预期的情况相同,控件把 Text 属性的值显示在 Web 页面上。

现在就有了一个基本的服务器控件项目,运行该项目,可以返回并查看由 ASP.NET Server Control 项目创建的模板类。该默认模板的代码如程序清单 7-11 所示(本章下载代码的 ServerControl1.cs)。

程序清单 7-11 Visual Studio ASP.NET Server Control 类模板

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Web;
using System.Web.UI;
```

```

using System.Web.UI.WebControls;

namespace CSharpServerControl1
{
    [DefaultProperty("Text")]
    [ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1>")]
    public class ServerControl1 : WebControl
    {
        [Bindable(true)]
        [Category("Appearance")]
        [DefaultValue("")]
        [Localizable(true)]
        public string Text
        {
            get
            {
                String s = (String)ViewState["Text"];
                return ((s == null)? "[" + this.ID + "]" : s);
            }

            set
            {
                ViewState["Text"] = value;
            }
        }

        protected override void RenderContents(HtmlTextWriter output)
        {
            output.Write(Text);
        }
    }
}

```

你会发现有关运行该项目所生成的默认服务器控件模板的一些有趣事情。首先请注意，使用特性修饰类的声明和 `Text` 属性。ASP.NET 服务器控件使用许多特性来指示不同类型的运行期间和设计期间行为。在接下来的章节中，你会了解到更多能够应用于服务器控件类和属性的特性。

其次，在默认情况下，该模板有 `Text` 属性和重写的方法 `RenderContents`，这个方法可以将 `Text` 属性的值显示在屏幕上。`RenderContents` 方法是输出来自服务器控件的内容的主要方法。

如果查看这个示例的 HTML 源代码，就会发现 ASP.NET 不仅在 HTML 标记中添加了 `Text` 属性的值，还把文本包含在 `` 块中。查看 `WebControl` 类的 `Render` 方法，可以看出 `Render` 方法除了调用 `RenderContents` 方法之外，还包含起始标记和结束标记，并且在控件内容的周围添加了 `Span` 标记。

```

protected internal override void Render(HtmlTextWriter writer)
{
    this.RenderBeginTag(writer);
    this.RenderContents(writer);
    this.RenderEndTag(writer);
}

```

如果为控件提供了 ID 值，那么 `span` 标记也会默认显示 ID 特性。包含 `span` 标记有时会出现问

题，因此如果希望禁止在 ASP.NET 中自动添加该标记，可以在控件中重写 `Render` 方法，并直接调用 `RenderContents` 方法。

```
protected override void Render(HtmlTextWriter writer)
{
    this.RenderContents(writer);
}
```

这个默认服务器控件模板示例演示了创建简单的服务器控件是多么容易。当然，这个控件没有什么功能，也缺乏服务器控件的许多功能。下一节将探讨如何使用 .NET Framework 的不同特性为服务器控件添加新的运行期间和设计期间的功能。

7.2.2 控件的特性

可以通过如下方式配置服务器控件提供给开发人员的设计期间体验：把特性添加到控件的类和属性。例如，查看上一节中的默认控件模板(见程序清单 7-11)，可以看出特性已经应用于类和 `Text` 属性。本节将研究可应用于服务器控件的特性，以及它们对控件行为的影响。

1. 类特性

服务器控件的类特性可以分为 3 类：全局控制运行期间行为的特性，在 Visual Studio 工具箱中如何显示控件的特性，以及将控件放到设计界面后控件行为的特性。表 7-1 列出了这些特性。

表 7-1

特 性	说 明
Designer	告诉设计器类，这个控件用于在 Visual Studio 的设计界面上显示控件在设计期间的视图
TypeConverter	为对象指定用作转换器的类型
DefaultEvent	表示用户在 Visual Studio 设计界面上双击控件时创建的默认事件
DefaultProperty	表示控件的默认属性
ControlBuilder	指定 <code>ControlBuilder</code> 类，用于在 ASP.NET 控件分析器中建立定制控件
ParseChildren	表示在服务器控件标记中嵌套的 XML 元素是用作属性还是子控件
TagPrefix	表示在 Web 页面的 HTML 中给控件添加的文本前缀

2. 属性/事件特性

属性特性用于控制服务器控件的许多不同方面。可以使用特性在 Visual Studio Property Browser 中控制属性和事件的行为方式；还可以使用特性控制属性和事件在设计期间的序列化。表 7-2 列出了一些可用的属性特性和事件特性。

表 7-2

特 性	说 明
Bindable	表示属性可以绑定到数据源
Browsable	表示在 Property Browser 中属性是否应在设计期间显示

(续表)

特 性	说 明
Category	表示在 Property Browser 中属性应显示在哪个类别中
Description	在 Property Browser 的底部显示一个文本字符串，描述该属性的作用
EditorBrowsable	表示属性显示在 Property Browser 中时是否可以编辑
DefaultValue	表示属性显示在 Property Browser 时的默认值
DesignerSerializationVisibility	指定属性在设计期间的序列化器中是否可见
NotifyParentProperty	表示在修改属性的值时通知其父属性
PersistChildren	表示在设计期间服务器控件的子控件是否保存为嵌套的内部控件
PersistenceMode	指定属性或事件在 ASP.NET 页面上如何保存
TemplateContainer	指定在创建后包含模板的 InamingContainer 容器的类型
Editor	表示控件用于编辑其值的 UI Type Editor 类
Localizable	表示属性包含可本地化的文本
Themable	表示属性是否有可应用于自身的主题

显然，表 7-1 和表 7-2 提供了许多信息。程序清单 7-11 中已经演示了一些特性，而本章的后面将使用这些表中列出的其他特性。

7.2.3 控件的显示

本章的前面介绍了如何使用 Visual Studio 项目模板启动和运行基本服务器控件，以及如何在服务器控件中应用特性来影响一些基本控件的行为。下面将重点介绍如何使用 ASP.NET 的特性给服务器控件添加更多高级的控件功能。

1. 页面事件的生命周期

在深入讨论服务器控件之前，首先了解普通 ASP.NET 页面的生命周期(包括服务器控件操作在内)将非常有用。控件的开发人员应负责重写在生命周期内执行的方法，实现自己的定制显示逻辑。

注意，在 Web 浏览器给服务器发出请求时会使用 HTTP，这是一个无状态协议。ASP.NET 提供了一个页面执行架构，有助于在 Web 应用程序中创建状态的幻象。这个架构基本上是每次在处理 ASP.NET 页面时执行的一系列方法和事件。图 7-6 显示了在控件的生命周期内调用的事件和方法。

在阅读本章后面的内容之后，你将会发现服务器控件会用到这里列出的许多事件和方法。理解它们以及它们的执行顺序是非常有用的，从而在为服务器控件添加新功能后，就可以构建自己的逻辑来遵循页面生命周期。



图 7-6

2. 显示 HTML 标记

服务器控件的主要任务是给 HTTP 输出流显示某种类型的标记语言，HTML 输出流由客户端返回并显示出来。如果客户端是标准的浏览器，控件就应显示 HTML；如果客户端是移动设备，控件就需要显示另一种类型的标记。如前所述，控件的开发人员应负责告诉服务器控件要显示什么标记。在控件的生命周期内调用重写的 `RenderContents` 方法，这主要是告诉控件要向客户端显示什么内容。在程序清单 7-12 中，注意 `RenderContents` 方法用于告诉控件输出 `Text` 属性的值(本章下载代码中的 `ServerControl1.cs`)。

程序清单 7-12 重写 `RenderContents` 方法

```
protected override void RenderContents(HtmlTextWriter output)
{
    output.Write(Text);
}
```

还要注意，`RenderContents` 方法有一个参数 `output`。这个参数是一个 `HtmlTextWriter` 类，控件通过它把 HTML 显示到客户端。这个特殊的写入器类专门用于在浏览器上显示兼容 HTML 4.0 的 HTML 代码。



即使 `HtmlTextWriter` 类显示兼容 HTML 4.0 的 HTML 代码，也仍可以完全控制所编写的元素和特性。因此，`HtmlTextWriter` 类也可以显示有效的 HTML5 标记。

`HtmlTextWriter` 类有许多用于显示 HTML 标记的方法，包括 `RenderBeginTag` 和 `WriteBeginTag`。程序清单 7-13 演示了如何修改控件的 `Render` 方法来显示 `<input>` 标记。

程序清单 7-13 使用 `HtmlTextWriter` 类显示 HTML 标记

```
protected override void RenderContents(HtmlTextWriter output)
{
    output.RenderBeginTag(HtmlTextWriterTag.Input);
    output.RenderEndTag();
}
```

首先，`RenderBeginTag` 方法用于显示 HTML 标记。使用这个方法显示 HTML 标记的优点在于，需要从 `HtmlTextWriterTag` 枚举中选择标记。使用 `RenderBeginTag` 方法和 `HtmlTextWriterTag` 枚举可以使控件自动支持不理解 HTML 4.0 语法的低级浏览器。

其次，注意还使用了 `RenderEndTag` 方法。顾名思义，这个方法显示结束标记。但是，不必在这个方法中指定要结束的标记。`RenderEndTag` 方法会自动结束 `RenderBeginTag` 方法显示的最后一个起始标记，这里是 `<input>` 标记。如果要显示多个 HTML 标记，应正确安排 `Begin` 和 `End` 显示方法的顺序。例如在程序清单 7-14 中，给控件添加了一个 `<div>` 标记，该标记在页面上显示时应位于 `<input>` 标记的外部。

程序清单 7-14 使用 `HtmlTextWriter` 类显示多个 HTML 标记

```
protected override void RenderContents(HtmlTextWriter output)
{
    output.RenderBeginTag(HtmlTextWriterTag.Div);
    output.RenderBeginTag(HtmlTextWriterTag.Input);
    output.RenderEndTag();
    output.RenderEndTag();
}
```

在基本理解如何显示简单的 HTML 标记之后，下面查看控件的输出。图 7-7 显示了页面的源代码。

可以看出，该控件显示了一些非常简单的 HTML 标记。另外要注意，该控件足够智能，可以识别出输入控件是否包含任何子控件，因此控件不需要显示完整的结束标记。相反，它会自动显示缩写方式 `</>`，而不是 `</input>`。



图 7-7

7.2.4 添加标记特性

显示 HTML 标记是建立控件的良好开端,但是该操作稍微有点简单。一般情况下,在显示 HTML 时,除了标记之外,还要在客户端上显示一些标记特性(如 ID 或 Name)。程序清单 7-15 演示了如何添加标记特性。

程序清单 7-15 显示 HTML 标记特性

```
protected override void RenderContents(HtmlTextWriter output)
{
    output.RenderBeginTag(HtmlTextWriterTag.Div);
    output.AddAttribute(HtmlTextWriterAttribute.Type, "text");
    output.AddAttribute(HtmlTextWriterAttribute.Id,
        this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Name,
        this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Value, this.Text);
    output.RenderBeginTag(HtmlTextWriterTag.Input);
    output.RenderEndTag();
    output.RenderEndTag();
}
```

从上面的示例中可以看出,使用 `AddAttribute` 方法给 `<input>` 标记添加了 3 个特性。另外,这里使用枚举 `HtmlTextWriterAttribute` 来选择要添加到标记中的特性。该枚举的作用与使用 `HtmlTextWriterTag` 枚举相同,也允许控件将其输出发送到低级浏览器。

与 `Render` 方法一样, `AddAttribute` 方法的顺序是非常重要的。在代码中, `AddAttribute` 方法应放在 `RenderBeginTag` 方法的前面。`AddAttribute` 方法将特性与 `RenderBeginTag` 方法显示的下一个 HTML 标记(这里是 `<input>` 标记)关联起来。

现在打开测试页面,查看添加了标记特性后的 HTML 源代码。图 7-8 显示了控件的 HTML 源代码。

在服务器控件中添加的标记特性现在是控件显示的 HTML 标记的一部分。



图 7-8

7.2.5 关于控件 ID

注意，在程序清单 7-15 中，使用控件的 ClientID 属性作为 Id 和 Name 特性的值是非常重要的。派生于 WebControl 类的控件会自动包含 3 个不同类型的 Id 属性：ID、UniqueID 和 ClientID。这些属性分别包含控件 ID 的不同版本，用于不同的场合。

ID 属性最显而易见。开发人员使用它获取和设置控件的 ID。在设计期间，ID 属性在页面上必须是唯一的。

UniqueID 属性是运行期间生成的只读属性，返回一个使用包含控件的 ID 预先定义的 ID。这样，ASP.NET 就可以唯一标识页面控件树中的每个控件，即使控件由容器控件(如 Repeater 或 GridView)多次使用，也可以唯一标识。例如，把这个定制控件添加到 Repeater 控件中，那么在执行页面时，Repeater 控件为每个定制控件显示的 UniqueID 就会改为包含 Repeater 的 ID：

```
MyRepeater:Ctrl0:MyCustomControl
```

从 ASP.NET 4 开始，根据 ClientIDMode 属性的值可以生成不同的 ClientID 属性。ClientIDMode 属性允许从 ASP.NET 用于生成 ClientID 的 4 个机制中选择一个：

- **AutoID**: 与 ASP.NET 早期版本中使用的行为一样。
- **Static**: 指定 ClientID 的值与 ID 值相同，不需要连接父容器的 ID。
- **Predictable**: 主要用于数据控件，连接控件的命名容器的 ID，但是生成的客户端 ID 值不包含像 ctlxxx 这样的字符串。相反，可以设置 ClientIDRowSuffix 属性，为生成的每个控件提供唯一值，这是 ClientIDMode 的默认值。
- **Inherit**: 指定控件生成的 ID 与其父容器相同。

另外，为了确保控件生成唯一的 ID，控件必须实现 INamingContainer 接口。这只是一个标记接口，不需要实现任何其他方法；但是，ASP.NET 运行时可以确保，无论控件的容器是什么，它在页面的树状层次结构中总是有唯一的名称。

7.2.6 给 HTML 设置样式

前面讨论了如何建立简单的服务器控件，显示适当的 HTML(包括特性)。但是，现代 Web 开发技术通常将 HTML 的使用限制为基本的内容描述机制，依靠 CSS 在 Web 页面中定位 HTML 元素和设置 HTML 元素的样式。本节将探讨如何使控件显示样式信息。

如 7.2 节开始部分所述，我们创建的控件继承了 WebControl 类。因此，这些控件都有用于显示大多数标准 CSS 样式特性的基础结构。在这个控件的 Property Browser 中，列出了许多样式属性，例如背景色、边框宽度和字体。还可以启动样式生成器来创建复杂的 CSS 样式。这些基本属性由 WebControl 类提供，但我们需要告诉控件显示在设计期间设置的值。为此，只需执行 AddAttributesToRender 方法。程序清单 7-16 列出了执行该操作的代码。

程序清单 7-16 显示样式属性

```
protected override void RenderContents(HtmlTextWriter output)
{
    output.RenderBeginTag(HtmlTextWriterTag.Div);
    output.AddAttribute(HtmlTextWriterAttribute.Type, "text");
    output.AddAttribute(HtmlTextWriterAttribute.Id,
        this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Name,
        this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Value, this.Text);
    this.AddAttributesToRender(output);
    output.RenderBeginTag(HtmlTextWriterTag.Input);
    output.RenderEndTag();
    output.RenderEndTag();
}
```

执行这个方法会告诉控件显示已设置的样式信息。这不仅可以显示与样式相关的属性，还会显示其他几个属性，如 ID、tabindex 和 tooltip。如果在控件前面的代码中手动显示这些属性，就会重复显示它们。

另外，应注意执行 AddAttributesToRender 方法的位置。在程序清单 7-16 中，该方法在显示 input 标记之前执行，这表示在 input 元素和包围 input 元素的 span 元素上显示属性。把方法调用放在 div 起始标记之前，可以确保属性会应用于 div 元素及其包围的 span 元素；如果将方法调用放在 div 结束标记之后，那么属性仅应用于 span 元素。

使用 Property Browser 可以把控件的背景色设置为 Silver，把字体设置为 Bold。在设置这些属性时，它们会自动添加到 ASP.NET 页面的控件标记中。在添加了样式后，控件标记如下所示：

```
<ccl:ServerControl1 BackColor="Red" Font-Bold=true
    ID="ServerControl11" runat="server" />
```

样式的改变会作为特性保存到控件中。在浏览器中执行这个页面时，样式信息应显示为 HTML，并且使文本框的背景色显示为银色，字体显示为粗体。图 7-9 显示了浏览器中的页面。



图 7-9

再次查看页面的源代码，可以看到样式信息已显示为 HTML 样式标记。图 7-10 显示了控件的 HTML 代码。



图 7-10

如果要更多地控制控件中样式的显示，可以使用 `HtmlTextWriter.AddStyleAttribute` 方法。与 `AddAttribute` 方法一样，`AddStyleAttribute` 方法也可以使用 `HtmlTextWriterStyle` 枚举指定要添加到控件的 CSS 特性。但与 `AddAttribute` 方法不同的是，使用 `AddStyleAttribute` 方法添加的特性在控件的样式特性中定义。程序清单 7-17 演示了 `AddStyleAttribute` 方法的使用。

程序清单 7-17 使用 `AddStyleAttribute` 方法添加控件样式

```
protected override void RenderContents(HtmlTextWriter output)
{
    output.RenderBeginTag(HtmlTextWriterTag.Div);
    output.AddAttribute(HtmlTextWriterAttribute.Type, "text");
    output.AddAttribute(HtmlTextWriterAttribute.Id,
        this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Name,
        this.ClientID + "_i");
```

```

        output.AddAttribute(HtmlTextWriterAttribute.Value, this.Text);
        output.AddStyleAttribute(HtmlTextWriterStyle.BackgroundColor, "Silver");
        output.RenderBeginTag(HtmlTextWriterTag.Input);
        output.RenderEndTag();
        output.RenderEndTag();
    }

```

运行这个示例，会把银色背景应用于控件。

7.2.7 添加客户端功能

显示 HTML 和给 HTML 设置样式的功能非常强大，还可以把其他资源传送给客户端，例如客户端脚本、图像和资源字符串。ASP.NET 提供了一些强大的新工具，可以在服务器控件上使用客户端脚本，以及把控件显示的 HTML 和其他资源放到客户端上。另外，ASP.NET 还有一个整体模型，可以在 Web 页面和服务器之间进行异步回调。

1. 使用客户端脚本

让控件利用客户端脚本，如 JavaScript，可以给控件添加强大的客户端功能。客户端脚本语言会利用客户端的浏览器创建更灵活的、更易用的控件。ASP.NET 提供了利用客户端脚本的许多方法，这些方法可用于控制脚本的显示位置和显示方式。

大多数需要显示客户端脚本的属性和方法可以通过 `ClientScriptManager` 类获得，可以使用 `Page.ClientScript` 方法访问该类。程序清单 7-18 演示了如何使用 `RegisterStartupScript` 方法向客户端显示 JavaScript。这个程序清单把代码添加到 `OnPreRender` 方法，而不是添加到前面示例使用的 `Render` 方法。这个方法允许每个控件通知页面它需要使用的客户端脚本。在调用 `Render` 方法后，页面就可以显示它在 `OnPreRender` 方法中收集的所有客户端脚本。如果在 `Render` 方法中调用了客户端脚本注册方法，就会在客户端脚本可以显示出来之前显示一部分页面。

程序清单 7-18 向浏览器显示客户端脚本

```

protected override void OnPreRender(EventArgs e)
{
    Page.ClientScript.RegisterStartupScript(typeof(Page),
        "ControlFocus", "document.getElementById('" +
        this.ClientID + "_i" + "').focus();", true);
}

```

在这个程序清单中，代码利用客户端脚本在加载 Web 页面时把控件的焦点自动转动到 `TextBox` 控件。在使用 `RegisterStartupScript` 方法时，注意它现在包含一个重载版本，该重载版本可以指定方法是否显示周围的脚本标记。如果给页面显示多个脚本，那么这个重载版本就很有用。

还要注意该方法需要一个键参数。这个参数用于唯一标识脚本块；如果在 Web 页面中注册了多个脚本块，就必须为每个块提供唯一键。可以使用 `IsStartupScriptRegistered` 方法和该键来确定以前是否已使用 `RegisterStartupScript` 方法将某个脚本块注册到客户端。

在浏览器中执行页面时，注意焦点现在自动位于文本框中。如果查看 Web 页面的源代码，就可以看到已将 JavaScript 写入页面的底部，如图 7-11 所示。



图 7-11

如果希望脚本显示在页面的顶部，可以使用 `RegisterStartupScriptBlock` 方法把脚本块显示在起始元素 `<form>` 的后面。

浏览器是从头至尾地分析 Web 页面的，因此如果在页面的顶部使用不包含在函数中的客户端脚本，那么在页面后面的代码中对 HTML 元素的任何引用都会失败。浏览器不会分析这部分页面。

能显示在页面加载时自动执行的脚本当然很好，但我们常常希望代码根据页面上某个 HTML 元素触发的事件来执行，如 `Click`、`Focus` 或 `Blur` 事件。为此，给要触发事件的 HTML 元素添加特性。程序清单 7-19 演示了如何修改控件的 `Render` 和 `PreRender` 方法来添加这个特性。

程序清单 7-19 使用客户端脚本和事件特性验证数据

```
protected override void RenderContents(HtmlTextWriter output)
{
    output.RenderBeginTag(HtmlTextWriterTag.Div);
    output.AddAttribute(HtmlTextWriterAttribute.Type, "text");
    output.AddAttribute(HtmlTextWriterAttribute.Id,
        this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Name,
        this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Value, this.Text);
    output.AddAttribute("OnBlur", "ValidateText(this)");
    output.RenderBeginTag(HtmlTextWriterTag.Input);
    output.RenderEndTag();
    output.RenderEndTag();
}

protected override void OnPreRender(EventArgs e)
{
    Page.ClientScript.RegisterStartupScript(
        typeof(Page),
        "ControlFocus", "document.getElementById('" +
```

```

        this.ClientID + "_i" + "').focus();", true);
Page.ClientScript.RegisterClientScriptBlock(
    typeof(Page),
    "ValidateControl",
    "function ValidateText(ctl) {" +
        "if (ctl.value=='') {" +
            "alert('Please enter a value.');" +
            "ctl.focus(); }" +
        "}", true);
}

```

可以看出，TextBox 控件被修改为检查空字符串。它还包含一个特性，该特性可将 JavaScriptOnBlur 事件添加到文本框中。OnBlur 事件在控件失去焦点时引发。此时，执行客户端的 ValidateText 函数，该函数是通过 RegisterClientScriptBlock 传送给客户端的。

返回的 HTML 如图 7-12 所示。



图 7-12

在页面中嵌入 JavaScript 的功能非常强大，但如果编写大量的客户端代码，就要考虑把这些 JavaScript 存储在一个外部文件中。使用 RegisterClientScriptInclude 方法可以把这个文件包含在 HTML 中。这个方法使用 URL 显示脚本标记，该 URL 是其 src 元素的值：

```
<script src="[url]" type="text/javascript"></script>
```

程序清单 7-20 演示了如何修改添加到程序清单 7-18 中的输入元素的验证代码，以便把 JavaScript 验证函数存储在外部的文件中。

程序清单 7-20 给 Web 页面添加客户端脚本包含文件

```

protected override void OnPreRender(EventArgs e)
{
    Page.ClientScript.RegisterClientScriptInclude(
        "UtilityFunctions", "Listing07-21.js");
    Page.ClientScript.RegisterStartupScript(

```

```

        typeof(Page),
        "ControlFocus", "document.getElementById('" +
        this.ClientID + "_i" + "').focus();",
        true);
    }

```

这段代码修改了 `OnPreRender` 事件，注册了一个客户端脚本包含文件，该文件包含 `ValidateText` 函数。我们需要在项目中添加一个 `JScript` 文件，并创建 `ValidateText` 函数，如程序清单 7-21 所示。

程序清单 7-21 包含在 JScript 文件中的 JavaScript 验证函数

```

function ValidateText(ctl)
{
    if(ctl.value=='') {
        alert('Please enter a value.');
```

```
        ctl.focus();
    }
}
```

`ClientScriptManager` 还提供可以注册隐藏的 HTML 字段并在 `OnSubmit` 事件中添加脚本函数的方法。

2. 访问嵌入的资源

要分布应用程序资源，如 JavaScript 文件、图像或资源文件，一种很好的方式是把它们直接嵌入编译好的程序集中。ASP.NET 通过使用 `RegisterClientScriptResource` 方法简化了该操作，这个方法是 `ClientScriptManager` 的一部分。

这个方法可以让 Web 页面在运行期间从编译好的程序集中检索存储的资源，如 JavaScript 文件。具体方式是使用 `HttpHandler` 从程序集中检索请求的资源，再把它们返回给客户端。`RegisterClientScriptResource` 方法会显示一个 `<script>` 块，其 `src` 值指向这个 `HttpHandler`（注意下载代码中的 `MiscEmbeddedScript.cs`、`AssemblyInfo.cs` 和 `MiscEmbeddedScript.aspx` 用于生成它）：

```

<script
src="/WebResource.axd?d=ktqObNC8c8uwwm_pAVTdak1ofzyXH33vO2kC2Pqa7gWUvT7XCNmAeT5Jig-
FTYk2SaMtNk3sR42AEFFZgrf7vQ_knZp3JtuIRq9xHCIDZBhHD0zyFRsRo-3AU0VeOyQ0rJl5SnnA-9ScC-
bwU0-PQ2&amp;t=634880142370628393" type="text/javascript"></script>

```

可以看出，`WebResource.axd` 处理程序用于返回资源，在这里，资源是 JavaScript 文件。使用这个方法可以检索存储在程序集中的任意资源，如图像或资源文件中的本地化内容字符串。

3. 异步回调

最后，ASP.NET 还在服务器控件中包含一个方便的机制，以激活基本的 AJAX 操作或客户端回调。客户端回调可以利用最现代的浏览器中的 `XmlHttp` 组件与服务器通信，而不实际地执行完整的回送过程。图 7-13 显示了 ASP.NET 架构中客户端回调的工作原理。

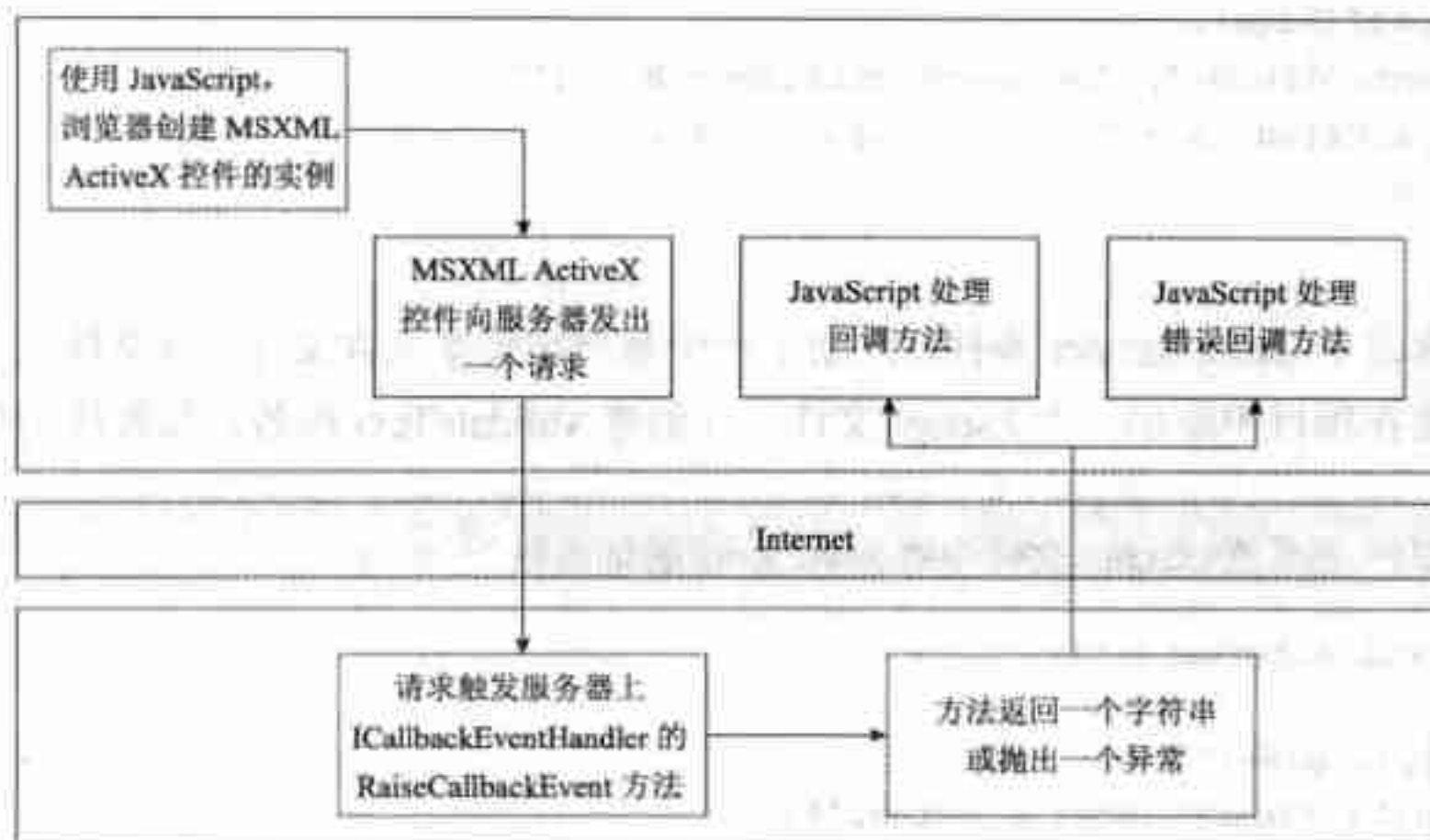


图 7-13

为了激活服务器控件上的回调功能，需要实现 `System.Web.UI.ICallbackEventHandler` 接口。这个接口需要实现两个方法：`RaiseCallbackEvent` 和 `GetCallbackResult`。它们是服务器端事件，在客户端执行回调时引发。在实现了该接口后，就要把客户端事件关联回服务器。为此，使用 `Page.ClientScript.GetCallbackEventReference` 方法，它可以指定两个客户端函数，一个用作回调处理程序，另一个用作错误处理程序。程序清单 7-22 演示了如何修改 `TextBox` 控件的 `Render` 方法，并添加 `RaiseCallbackEvent` 方法，以使用回调进行验证。

程序清单 7-22 添加异步回调以验证数据

```

protected override void RenderContents(HtmlTextWriter output)
{
    output.RenderBeginTag(HtmlTextWriterTag.Div);
    output.AddAttribute(HtmlTextWriterAttribute.Type, "text");
    output.AddAttribute(HtmlTextWriterAttribute.Id, this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Name, this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Value, this.Text);
    output.AddAttribute("OnBlur", "ClientCallback();");
    output.RenderBeginTag(HtmlTextWriterTag.Input);
    output.RenderEndTag();
    output.RenderEndTag();
}

protected override void OnPreRender(EventArgs e)
{
    Page.ClientScript.RegisterClientScriptInclude(
        "UtilityFunctions", "Listing07-23.js");

    Page.ClientScript.RegisterStartupScript(
        typeof(Page),
        "ControlFocus", "document.getElementById('" + this.ClientID + "_i" + "').focus();", true);

    Page.ClientScript.RegisterStartupScript(
        typeof(Page), "ClientCallback",
        "function ClientCallback() {" +

```

```
        "args=document.getElementById('" + this.ClientID + "_i" + "').value;" +
        Page.ClientScript.GetCallbackEventReference(this, "args",
            "CallbackHandler", null, "ErrorHandler", true) + "}", true);
    }

    public void RaiseCallbackEvent(string eventArgument)
    {
        int result;
        if(!Int32.TryParse(eventArgument, out result))
            throw new Exception("The method or operation is not implemented.");
    }

    public string GetCallbackResult()
    {
        return "Valid Data";
    }
}
```

可以看出，在此再次修改了 OnBlur 属性，这一次是调用 ClientCallback 方法。这个方法在 PreRender 事件中创建和显示。该事件的主要作用是填充客户端的 args 变量，调用客户端回调方法。

我们使用 GetCallbackEventReference 方法生成启动回调的客户端脚本。传送给该方法的参数指定了启动回调的控件、客户端回调方法的名称以及回调方法参数的名称。表 7-3 详细说明了 GetCallbackEventReference 方法的参数。

表 7-3

参 数	说 明
Control	启动回调的服务器控件
Argument	客户端变量，用于把参数传送给服务器端的事件处理程序
ClientCallback	用作回调方法的客户端函数。回调方法在服务器端处理完全成功时执行
Context	客户端变量，用于直接传送给接收它的客户端函数。该变量不传送给服务器
ClientErrorCallback	用作回调错误处理方法的客户端函数。回调错误处理方法在服务器端处理遇到错误时执行

在代码中，调用了两个客户端方法：CallbackHandler 和 ErrorHandler。这两个方法的参数是 args 和 ctx。

除了修改服务器控件的代码之外，还在 JavaScript 文件中添加了两个客户端回调函数。程序清单 7-23 显示了这两个新函数。

程序清单 7-23 客户端回调 JavaScript 函数

```
var args;
var ctx;

function ValidateText(ctl)
{
    if(ctl.value=='') {
        alert('Please enter a value.');
```



```
function CallbackHandler(args,ctx)
{
    alert("The data is valid");
}

function ErrorHandler(args,ctx)
{
    alert("Please enter a number");
}
```

现在,在浏览器中查看 Web 页面时,只要文本框失去焦点,就执行客户端回调来验证数据。回调会引发服务器上的 RaiseCallbackEvent 方法,该方法验证在 eventArguments 中传送给文本框的值。如果该值有效,就返回一个字符串,并引发客户端的 CallbackHandler 函数。如果该值无效,就抛出异常,执行客户端的 ErrorHandler 函数。

7.2.8 浏览器功能

本章前面论述了许多强大的功能,例如给客户端脚本添加样式和发送客户端脚本等,在编写自己的定制控件时可以使用这些功能。但是,如果使用这些功能,就必须考虑处理某些浏览器的方式,这些浏览器通常称为低级浏览器,它们可能不支持这些高级功能或不能启用它们。检测和响应低级浏览器的功能是创建控件时必须考虑的一个方面。ASP.NET 有一些强大的工具,可用于检测发出页面请求的浏览器的类型和版本,以及浏览器支持的功能。

1. .browser 文件

ASP.NET 引入了一种非常灵活的新方法,可配置、存储和搜索浏览器功能。所有的浏览器标识和功能信息都存储在.browser 文件中。ASP.NET 把这些文件放在 C:\Windows\Microsoft.NET\Framework\v4.0.xxxxx\CONFIG\Browsers 目录下。如果打开这个文件夹,就会看到 ASP.NET 提供的许多.browser 文件,它们描述了目前大多数常见桌面浏览器的功能,以及 PDA 和手机等设备上的浏览器信息。打开其中一个.browser 文件,会看到该文件包含浏览器的所有标识和功能信息。程序清单 7-24 列出了 iPhone 功能文件的内容,它通常位于机器的 C:\Windows\Microsoft.NET\Framework\v4.0.30319\Config\Browsers\iphone.browser 下。



如果使用 .NET Framework 的 x64 版本,也可以浏览 C:\Windows\Microsoft.NET\Framework64\v4.0.xxxxx\CONFIG\Browsers。

程序清单 7-24 浏览器功能文件示例

```
<browsers>
  <!-- Mozilla/5.0 (iPhone; U; CPU like Mac OS X; en) AppleWebKit/420+ (KHTML, like Gecko)
    Version/3.0 Mobile/1A543a Safari/419.3 -->
  <gateway id="IPhone" parentID="Safari">
    <identification>
      <userAgent match="iPhone" />
```



```

</identification>
<capabilities>
  <capability name="isMobileDevice" value="true" />
  <capability name="mobileDeviceManufacturer" value="Apple" />
  <capability name="mobileDeviceModel" value="iPhone" />
  <capability name="canInitiateVoiceCall" value="true" />
</capabilities>
</gateway>

<!-- Mozilla/5.0 (iPod; U; CPU like Mac OS X; en) AppleWebKit/420.1 (KHTML, like Gecko)
Version/3.0 Mobile/4A93 Safari/419.3 -->
<gateway id="iPod" parentID="Safari">
  <identification>
    <userAgent match="iPod" />
  </identification>
  <capabilities>
    <capability name="isMobileDevice" value="true" />
    <capability name="mobileDeviceManufacturer" value="Apple" />
    <capability name="mobileDeviceModel" value="iPod" />
  </capabilities>
</gateway>

<!-- Mozilla/5.0 (iPad; U; CPU OS 4_3 like Mac OS X; en-us) AppleWebKit/533.17.9
(KHTML, like Gecko) Version/5.0.2 Mobile/8F191 Safari/6533.18.5 -->
<gateway id="iPad" parentID="Safari">
  <identification>
    <userAgent match="iPad" />
  </identification>
  <capabilities>
    <capability name="isMobileDevice" value="true" />
    <capability name="mobileDeviceManufacturer" value="Apple" />
    <capability name="mobileDeviceModel" value="iPad" />
  </capabilities>
</gateway>
</browsers>

```

存储浏览器功能信息的这种方法的优点是，在创建新浏览器或发布新版本时，开发人员只需创建或更新**browser**文件以描述该浏览器的功能。

2. 访问浏览器的功能信息

理解了 ASP.NET 如何存储浏览器的功能信息后，就可以研究如何在运行期间访问这些信息，这样在编写控件时就可以根据浏览器来改变控件的显示内容。要访问发出请求的浏览器的功能信息，可以使用 `Page.Request.Browser` 属性。这个属性允许访问 `System.Web.HttpBrowserCapabilities` 类，该类提供了当前发出请求的浏览器的功能信息。还提供了许多属性，这些属性描述了浏览器可以支持什么功能，可以显示什么内容，以及需要什么功能。列表就是使用这些信息给 `TextBox` 控件添加功能。程序清单 7-25 演示了如何检测浏览器的功能，以确保浏览器支持 JavaScript。

程序清单 7-25 在服务器端代码中检测浏览器的功能

```
protected override void OnPreRender(EventArgs e)
```

```

{
    if (Page.Request.Browser.EcmaScriptVersion.Major > 0)
    {
        Page.ClientScript.RegisterClientScriptInclude(
            "UtilityFunctions", "Listing07-23.js");

        Page.ClientScript.RegisterStartupScript(
            typeof(Page), "ControlFocus", "document.getElementById('" +
                this.ClientID + "_i" + "').focus();", true);

        Page.ClientScript.RegisterStartupScript(
            typeof(Page), "ClientCallback",
            "function ClientCallback() {" +
                "args=document.getElementById('" + this.ClientID + "_i" +
                "').value;" +
                Page.ClientScript.GetCallbackEventReference(this, "args",
                    "CallbackHandler", null, "ErrorHandler", true) + "}", true);
    }
}

```

这是一个非常简单的例子，却说明了 `HttpBrowserCapabilities` 类的用法。

7.2.9 使用 ViewState

注意，在开发 Web 应用程序时，它们建立在无状态的 HTTP 协议基础之上。ASP.NET 提供了许多方式，使用户产生他们在使用有状态的应用程序的幻象，最普遍使用的方式是 `ViewState`。`ViewState` 可以在 Web 页面的整个生命周期中，把控件的状态存储在隐藏的表单字段(显示为 HTML 的一部分)内，以维护 Web 页面上控件和对象的状态。包含在表单字段中的状态可以由应用程序用于在进行回送时重新构建页面的状态。图 7-14 显示了 ASP.NET 如何在隐藏的表单字段中存储 `ViewState` 信息。



图 7-14

注意，页面包含隐藏的表单字段 `_ViewState`，这个表单字段的值是 Web 页面的 `ViewState`。在默认情况下，`ViewState` 在 ASP.NET 内置的所有服务器控件中都是激活的。但如果编写定制的服务器控件，就要确保该控件参与页面对 `ViewState` 的使用过程。

ASP.NET ViewState 实际上是一种存储格式, 这种格式可以把对象保存为键/值对, 并提取它们。如图 7-14 所示, 这些对象都由 ASP.NET 序列化, 并保存为加密字符串, 再作为隐藏的 HTML 表单字段传送给客户端。在页面回送给服务器时, ASP.NET 就可以使用这个隐藏的表单字段重新构建 StateBag, 之后就可以在服务器处理页面时访问它。



ViewState 有时可能非常庞大, 会影响整个页面大小, 因此应考虑采用另一种方法来存储 ViewState 信息。可以从 System.Web.UI.PageStatePersister 类派生一个类, 重写它的 Load 和 Save 方法, 以此创建自己的保存机制。

如程序清单 7-26 所示, 在默认情况下, ASP.NET Server Control 模板中的 Text 属性被设置为将其值存储在 ViewState 中。

程序清单 7-26 对 Text 属性使用 ViewState

```
public string Text
{
    get
    {
        String s = (String)ViewState["Text"];
        return ((s == null)? "[" + this.ID + "]" : s);
    }

    set
    {
        ViewState["Text"] = value;
    }
}
```

在 ASP.NET 服务器控件中创建新属性时应使用这个技术, 以确保终端用户在控件中设置的值会在页面回送期间保存下来。



在页面引发 OnInit 事件后, 就会加载 ViewState。如果在引发该事件之前, 控件修改了自身或另一个服务器控件, 这些修改就不会保存到 ViewState 中。

1. 类型和 ViewState

如前所述, ViewState 基本上是对对象的泛型集合, 但并不是所有的对象都可以添加到 ViewState 中。只有可以安全保存的类型才能在 ViewState 中使用, 因此像数据库连接或文件句柄这样的对象就不应添加到 ViewState 中。

另外, 对某些数据类型进行了优化, 以便在 ViewState 中使用它们。在给 ViewState 添加数据时, 应把数据封装到如下类型中:

- 基本类型(Int32、Boolean 等)
- 基本类型的数组

- ArrayList、HashTable
- Pair、Triplet
- Color、DateTime
- String、IndexedString
- 这些类型的 HybridDictionary
- 包含 TypeConverter 的对象, 但要注意如果使用这些类型, 性能就会降低
- 可序列化的对象(使用 Serializable 属性标记)

2. ControlState

有时控件必须在回送过程中存储通常是私有的少量重要信息。为了存储这种类型的信息, 即使开发人员禁用 ViewState, 也可以使用 ASP.NET 中一种不同类型的 ViewState, 称为 ControlState。ControlState 基本上只是用于定制控件的私有 ViewState。



为了确保数据在回送过程中可用, 应使用 ControlState 存储信息, 而不是 ViewState。ViewState 可以在 Web 应用程序或站点上禁用, 使依赖 ViewState 的控件失败。

有两个方法 SaveControlState 和 LoadControlState 可用于访问 ControlState, 但需要自己编写这些方法的实现代码。程序清单 7-27 演示了如何使用 LoadControlState 和 SaveControlState 方法。

程序清单 7-27 在服务器控件中使用 ControlState

```
[DefaultProperty("Text")]
[ToolboxData("<(0):Listing0727 runat=server></(0):Listing0727>")]
public class Listing0727 : WebControl
{
    string state;
    protected override void OnInit(EventArgs e)
    {
        Page.RegisterRequiresControlState(this);

        base.OnInit(e);
    }

    protected override void LoadControlState(object savedState)
    {
        state = (string)savedState;
    }

    protected override object SaveControlState()
    {
        return (object)"ControlSpecificData";
    }

    protected override void RenderContents(HtmlTextWriter output)
    {
    }
}
```

```

        output.Write("Control State: " + state);
    }
}

```

计划使用 `ControlState` 的控件必须在保存控件的状态数据之前调用 `Page.RegisterRequiresControlState` 方法。另外，必须在加载每个页面时调用 `RegisterRequiresControlState` 方法，因为其值在页面回送的过程中不会保存。

7.2.10 引发回送事件

如本章前面所述，ASP.NET 提供了一组非常强大的工具，可以用于开发服务器控件，并把它们传送给客户端浏览器。但这仍然是单向通信，因为数据只能由服务器传送给客户端。更好的情况是服务器控件可以把数据传送回服务器。把数据传送回服务器的过程一般称为页面回送。只要单击表单按钮或链接，使页面给 Web 服务器发出新请求，就是在进行页面回送。

ASP.NET 为处理 ASP.NET Web 页面的回送提供了一个功能丰富的架构。另外，ASP.NET 还有一个开发模型用来模拟标准的 Windows Forms 事件模型。该开发模型允许使用控件，即使这些控件显示在客户端浏览器上，也会引发服务器端代码中的事件。它还提供了一种简单的机制，可以把服务器控件植入该架构，允许创建可以启动页面回送的控件。图 7-15 显示了 ASP.NET 回送架构。



图 7-15

为了启动页面回送, ASP.NET 默认使用了客户端脚本。如果能让定制控件也可以进行回送, 就必须在控件显示方法期间使用 `GetPostBackEventReference` 方法把回送启动脚本添加到 HTML 元素事件中。程序清单 7-28 演示了如何把回送客户端脚本添加到标准 HTML Button 元素的 `onClick` 事件。

程序清单 7-28 给服务器控件添加回送功能

```
protected override void RenderContents(HtmlTextWriter output)
{
    PostBackOptions p = new PostBackOptions(this);
    output.AddAttribute(HtmlTextWriterAttribute.Onclick,
        Page.ClientScript.GetPostBackEventReference(p));
    output.AddAttribute(HtmlTextWriterAttribute.Id,
        this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Name,
        this.ClientID + "_i");
    output.RenderBeginTag(HtmlTextWriterTag.Button);
    output.Write("My Button");
    output.RenderEndTag();
}
```

当调用 `GetPostBackEventReference` 方法时, 必须给它传送一个 `PostBackOptions` 对象。该对象可用于指定一些配置选项, 这些选项会影响 ASP.NET 启动回送的方式。

可以把回送的 JavaScript 添加到任意客户端事件。可以在控件中包含一些预回送逻辑, 甚至可以把代码添加到客户端函数。

控件可以启动回送后, 就要把页面回送过程中执行的事件添加到控件。为了在客户端对象上引发服务器端事件, 需要实现 `System.Web.IPostBackEventHandler` 接口。程序清单 7-29 演示了如何给程序清单 7-28 中的 Button 控件实现该接口。

程序清单 7-29 在服务器控件中处理回送事件

```
[DefaultProperty("Text")]
[ToolboxData("<{0}:Listing0729 runat=server></{0}:Listing0729>")]
public class Listing0729 : WebControl, IPostBackEventHandler
{
    public event EventHandler Click;

    public virtual void OnClick(EventArgs e)
    {
        if(Click != null)
        {
            Click(this, e);
        }
    }

    public void RaisePostBackEvent(string eventArgument)
    {
        OnClick(EventArgs.Empty);
    }

    protected override void RenderContents(HtmlTextWriter output)
```



```

    {
        PostBackOptions p = new PostBackOptions(this);
        output.AddAttribute(HtmlTextWriterAttribute.Onclick,
            Page.ClientScript.GetPostBackEventReference(p));
        output.AddAttribute(HtmlTextWriterAttribute.Id,
            this.ClientID + "_i");
        output.AddAttribute(HtmlTextWriterAttribute.Name,
            this.ClientID + "_i");
        output.RenderBeginTag(HtmlTextWriterTag.Button);
        output.Write("My Button");
        output.RenderEndTag();
    }
}

```

当用户单击按钮时,就执行页面回送,并且 ASP.NET 会在控件中调用 `RaisePostBackEvent` 方法,这会引发一个服务器端事件。如果控件中的许多不同的客户端事件都能启动回送,就可以使用 `RaisePostBackEvent` 方法的 `eventArgument` 参数改变控件的行为,以确定哪个元素引发了回送。可以使用前面提到过的 `PostBackOptions` 对象来设置 `eventArgument` 参数。

7.2.11 处理回送数据

学习了如何在 `ViewState` 中存储控件的数据以及如何在控件中添加回送功能之后,现在就可以使控件在页面上处理用户输入到表单字段中的数据。当 ASP.NET 控件启动页面回送时,页面上所有的表单数据都会传送给服务器。服务器控件可以访问页面上传送的数据并与其交互,可以把这些信息存储到 `ViewState` 中,并能产生有状态的应用程序的幻象。

要访问这些回送数据,控件必须实现 `System.Web.IPostBackDataHandler` 接口。这个接口允许 ASP.NET 把在回送过程中传回服务器的表单数据传递给控件。

`IPostBackDataHandler` 接口要求实现两个方法: `LoadPostData` 和 `RaisePostBackDataChangedEvent`。程序清单 7-30 演示了如何在文本输入控件上实现 `IPostBackDataHandler` 接口的方法。

程序清单 7-30 访问服务器控件中的回送数据

```

[DefaultProperty("Text")]
[ToolboxData("<{0}:Listing0730 runat=server></{0}:Listing0730>")]
public class Listing0730 : WebControl, IPostBackEventHandler, IPostBackDataHandler
{
    [Bindable(true)]
    [Category("Appearance")]
    [DefaultValue("")]
    [Localizable(true)]
    public string Text
    {
        get
        {
            String s = (String)ViewState["Text"];
            return((s == null) ? "[" + this.ID + "]" : s);
        }
        set
        {
            ViewState["Text"] = value;
        }
    }
}

```

```

    }

    protected override void RenderContents(HtmlTextWriter output)
    {
        PostBackOptions p = new PostBackOptions(this);
        output.AddAttribute(HtmlTextWriterAttribute.Id, this.ClientID);
        output.AddAttribute(HtmlTextWriterAttribute.Name, this.ClientID);
        output.AddAttribute(HtmlTextWriterAttribute.Value, this.Text);
        output.RenderBeginTag(HtmlTextWriterTag.Input);
        output.RenderEndTag();
    }

    public bool LoadPostData(string postDataKey,
        System.Collections.Specialized.NameValueCollection postCollection)
    {
        this.Text = postCollection[postDataKey];
        return false;
    }

    public void RaisePostBackDataChangedEvent()
    {
    }

    public event EventHandler Click;

    public virtual void OnClick(EventArgs e)
    {
        if(Click != null)
        {
            Click(this, e);
        }
    }

    public void RaisePostBackEvent(string eventArgument)
    {
        OnClick(EventArgs.Empty);
    }
}

```

在整个回送期间，ASP.NET 会为该控件调用 `LoadPostData` 方法，并把表单提交的所有数据以 `NameValueCollection` 的形式传送给它。`postDataKey` 方法的参数允许控件访问 `NameValueCollection` 为它指定的回送数据。

使用这些方法参数可以把输入元素的文本保存到服务器控件的 `Text` 属性中。在前面的 `ViewState` 例子中，`Text` 属性把值保存到 `ViewState` 中。当另一次页面回送发生时，会自动使用输入元素的值重新填充该控件。

`LoadPostData` 方法需要返回一个布尔值，这个值表示在 `LoadPostData` 方法执行完毕后，ASP.NET 是否应该调用 `RaisePostBackDataChangedEvent` 方法。例如，如果在控件中创建一个 `TextChanged` 事件，通知控件的文本已经改变，就需要从 `RaisePostBackDataChangedEvent` 方法返回 `true`，从而可以随后在 `RaisePostBackDataChangedEvent` 方法中引发这个事件。

7.2.12 复合控件

前面在介绍服务器控件时，主要探讨了单个 HTML 控件的显示，但这是非常有限的功能。创建非常强大的控件时，常常需要把几个 HTML 元素组合在一起。通常可以使用 `RenderContents` 方法来使用多个 HTML 元素，但 ASP.NET 也允许在定制的服务器控件中使用已有的 ASP.NET 控件。这类控件称为复合控件。

为了演示复合控件的创建，可以尝试把程序清单 7-30 中所示的控件改为复合控件。程序清单 7-31 显示了这个创建过程。

程序清单 7-31 创建复合控件

```
[DefaultProperty("Text")]
[ToolboxData("<{0}:Listing0731 runat=server></{0}:Listing0731>")]
public class Listing0731 : CompositeControl
{
    protected TextBox textbox = new TextBox();

    protected override void CreateChildControls()
    {
        this.Controls.Add(textbox);
    }
}
```

这个程序清单中的许多地方都很重要。首先，注意该控件类继承了 `CompositeControl`，而不是 `WebControl`。派生于 `CompositeControl` 可以为这类控件提供几个额外、专用的功能。

其次，代码中没有 `Render` 方法，而是在 `CreateChildControls` 方法中创建了另一类服务器控件的一个实例，并把它添加到 `Controls` 集合中。在运行这个例子时，它会显示一个与前面控件相同的文本框。实际上，它显示的 HTML 完全相同。

把复合控件(如上一个例子中的文本框)拖放到设计界面上时，可以注意到即使在复合控件中使用强大的 ASP.NET `TextBox` 控件，该控件的属性也不会显示在 `Properties Explorer` 中显示出来。为了通过父容器显示子控件的属性，必须在父控件中创建相应的属性。例如，如果要通过父控件显示 ASP.NET 文本框的 `Text` 属性，就要在父控件中创建 `Text` 属性。程序清单 7-32 演示了该过程。

程序清单 7-32 显示复合控件中的控件属性

```
[DefaultProperty("Text")]
[ToolboxData("<{0}:Listing0732 runat=server></{0}:Listing0732>")]
public class Listing0732 : CompositeControl
{
    protected TextBox textbox = new TextBox();

    public string Text
    {
        get
        {
            EnsureChildControls();
            return textbox.Text;
        }
        set
    }
}
```



```

        {
            EnsureChildControls();
            textbox.Text = value;
        }
    }

    protected override void CreateChildControls()
    {
        this.Controls.Add(textbox);
        this.ChildControlsCreated = true;
    }
}

```

注意，这个属性用于填充底层控件的属性。另外，在访问底层控件的属性之前，必须调用 `EnsureChildControls` 方法。这个方法可确保在访问容器控件的子控件之前已初始化它们。

7.2.13 模板化控件

除了复合控件之外，还可以创建模板化控件。模板化控件允许开发人员指定用于显示控件的一部分 HTML，在容器控件中嵌套其他控件。我们应很熟悉 `Repeater` 或 `DataList` 控件，它们都是模板化控件，可以指定在显示页面时绑定的数据如何显示。

为了演示模板化控件，下面的代码给出了一个简单的例子，在 Web 页面上显示一条文本消息。由于该控件是模板化控件，因此开发人员完全控制了消息的显示方式。

首先，创建 `Message` 服务器控件，用作容器控件中的模板。程序清单 7-33 中的类扩展了已有的 `Panel` 控件，添加了两个属性——`Name` 与 `Text`，以及一个新的构造函数。

程序清单 7-33 创建模板化控件的内部控件类

```

public class Message : Panel, INamingContainer
{
    public string Name { get; internal set; }
    public string Text { get; internal set; }
}

```

稍后将说明可以访问 `Message` 类的公有属性，以便把动态内容插入模板中。还将说明如何将 `Name` 和 `Text` 属性的值作为显示的模板化控件的一部分显示。

接着，创建一个新的服务器控件，它是 `Message` 控件的容器。这个服务器控件负责显示嵌套在其中的模板化控件，如程序清单 7-34 所示。

程序清单 7-34 创建模板化控件的容器类

```

[DefaultProperty("Text")]
[ToolboxData("<{0}:Listing0734 runat=server></{0}:Listing0734>")]
public class Listing0734 : WebControl
{
    [Browsable(false)]
    public Message TemplateMessage { get; internal set; }

    [PersistenceMode(PersistenceMode.InnerProperty)]
    [TemplateContainer(typeof(Message))]
}

```

```

public virtual ITemplate MessageTemplate{get; set;}

[Bindable(true)]
[DefaultValue("")]
public string Name{get; set;}

[Bindable(true)]
[DefaultValue("")]
public string Text{get; set;}

public override void DataBind()
{
    EnsureChildControls();
    ChildControlsCreated = true;

    base.DataBind();
}

protected override void CreateChildControls()
{
    this.Controls.Clear();
    this.TemplateMessage = new Message() { Name = Name, Text = Text };

    if(this.MessageTemplate == null)
    {
        this.MessageTemplate = new DefaultMessageTemplate();
    }

    this.MessageTemplate.InstantiateIn(this.TemplateMessage);
    Controls.Add(this.TemplateMessage);
}

protected override void RenderContents(HtmlTextWriter writer)
{
    EnsureChildControls();
    ChildControlsCreated = true;

    base.RenderContents(writer);
}
}

```

现在开始仔细研究这个例子。首先要注意的是 `MessageTemplate` 属性，这个属性使 Visual Studio 知道该控件包含一个模板，并允许它为该模板显示 IntelliSense。使用 `PersistenceMode` 特性标记该属性，表示模板化控件应在 ASPX 页面中保存为控件标记中的内部属性。另外，还使用 `TemplateContainer` 特性标记该属性，以告诉 ASP.NET 这个属性表示哪种类型的模板化控件。在本例中，该属性表示前面创建的 `Message` 模板化控件。

这个容器控件有两个公有属性：`Name` 和 `Text`。这些属性用于填充 `Message` 控件的 `Name` 和 `Text` 属性，因为该类不允许开发人员直接设置这些属性。

最后，由 `DataBind` 方法调用的 `CreateChildControls` 方法完成了这个控件中的大部分工作。它创建了一个新的 `Message` 对象，把 `Name` 和 `Text` 属性的值传送给构造函数作为其值。`CreateChildControls` 方法执行完毕后，就继续执行基本的 `DataBind` 操作。这一点很重要，因为 `Name` 和 `Text` 属性的值

在 `CreateChildControls` 方法中计算，并可以把这些属性值插入模板化控件。

在创建模板化控件时要注意，如果开发人员没有为控件指定模板，会发生什么情况？在上面的例子中，如果从 `TemplateContainer` 中删除了 `MessageTemplate` 控件，那么在运行 Web 页面时就会生成 `NullReferenceException` 异常，这是因为容器控件的 `MessageTemplate` 属性会返回一个空值。为了避免这种情况，可以在容器控件中包含默认模板类。默认模板的例子如程序清单 7-35 所示。

程序清单 7-35 创建模板化控件的默认模板类

```
internal sealed class DefaultMessageTemplate : ITemplate
{
    public void InstantiateIn(Control container)
    {
        Literal l = new Literal();
        l.Text = "No MessageTemplate was included.";
        container.Controls.Add(l);
    }
}
```

注意，`DefaultMessageTemplate` 实现了 `ITemplate` 接口。这个接口要求实现 `InstantiateIn` 方法，该方法用于提供默认模板内容。

要包含默认模板，只需在 `TemplatedControl` 类中添加这个类。还需要修改 `CreateChildControls` 方法，以检测空的 `MessageTemplate`，然后创建默认模板的一个实例并使用它。

```
if(this.MessageTemplate == null)
{
    this.MessageTemplate = new DefaultMessageTemplate();
}
```

创建好控件和模板后，就可以把它们拖放到测试 Web 页面上。程序清单 7-36 演示了该控件如何用于定制数据的显示。

程序清单 7-36 给 Web 页面添加模板化控件

```
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        this.Listing07341.DataBind();
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Templated Web Controls</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <ccl:Listing0734 ID="Listing07341" runat="server" Name="John Doe"
                Text="Hello World!">
                <MessageTemplate>The user '<%# Container.Name %>' has a
                    message for you: <br /><%# Container.Text %>
                </MessageTemplate>
            </ccl:Listing07341>
        </div>
    </form>
</body>
</html>
```



```

        </cc1:Listing0734>
    </div>
</form>
</body>
</html>

```

从这个程序清单中可以看出，`<cc1:TemplatedControl>`控件包含一个 `MessageTemplate`，此处已对其进行定制以显示 `Name` 和 `Text` 属性的值。图 7-16 显示了 `MessageTemplate` 显示在浏览器上之后的页面。

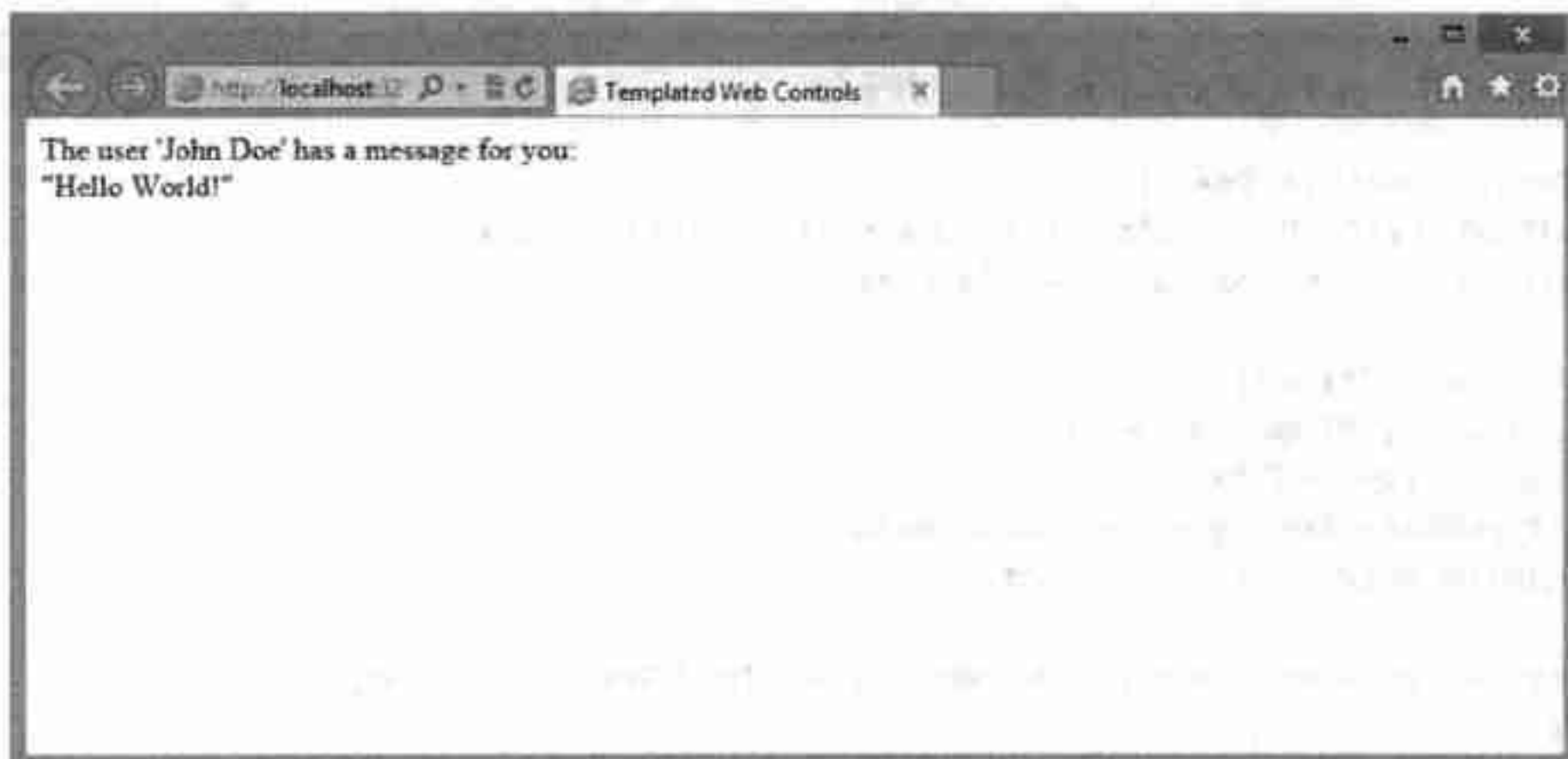


图 7-16

7.2.14 在设计期间创建控件

本章前面主要讨论了在客户端浏览器上显示的内容，但浏览器并不是服务器控件的唯一使用者。Visual Studio 和使用服务器控件的开发人员也是它的使用者，因此在使用控件时还需要考虑这些情况。

ASP.NET 对开发人员在设计期间使用控件方面进行了大量的改进，其中一些改进不需要额外的编码，例如用户控件和基本服务器控件的“所见即所得”功能。但对于比较复杂的情况，在使用控件时，ASP.NET 提供了一些选项，让开发人员在设计期间更好地完成任务。

在编写服务器控件时，开发人员在设计期间的操作要优先于运行期间的操作。也就是说，在设计界面上改变控件的外观以响应控件属性的改变，以及在设计界面上引入其他服务器控件。创建服务器控件在设计期间的操作涉及 3 个主要组件：

- 类型转换器
- 设计器
- UI 类型编辑器

这些主题中的每个都需要整章的篇幅来介绍，因此本节只概述它们，简介它们如何关联到控件在设计期间的操作，再举一些简单的例子以说明它们的用法。

1. 类型转换器

`TypeConverter` 类可以在类型之间执行转换操作。Visual Studio 在设计期间使用类型转换器，把对象的属性值转换为 `String` 类型，以便把它们显示在 `Property Browser` 上。在开发人员修改属性时，

类型转换器再把属性值转换回原来的类型。

ASP.NET 包含许多不同的类型转换器, 在创建控件的设计期间行为时可以使用它们。这些转换器可以转换大多数数字类型, 还可以转换 Font、Color、DateTime 和 Guid 类型。为了查看 .NET Framework 中可用的类型转换器, 最简单的方式是使用 MSDN Library 帮助, 在 .NET Framework 中搜索派生于 `TypeConverter` 类的类。

找到要在控件属性上使用的类型转换器后, 就使用 `TypeConverter` 特性标记该属性, 如程序清单 7-37 所示。

程序清单 7-37 将 `TypeConverter` 特性应用于属性

```
[DefaultProperty("Text")]
[ToolboxData("<{0}:Listing0737 runat=server></{0}:Listing0737>")]
public class Listing0737 : WebControl
{
    [Bindable(true)]
    [Category("Appearance")]
    [DefaultValue("")]
    [TypeConverter(typeof(GuidConverter))]
    public Guid BookId { get; set; }

    protected override void RenderContents(HtmlTextWriter output)
    {
        output.Write(BookId.ToString());
    }
}
```

在这个例子中, 提供一个属性接受并返回 `Guid` 类型的对象。Property Browser 不能显示 `Guid` 对象, 因此要将其值转换为字符串, 这样它才能正确显示在 Property Browser 中。使用 `TypeConverter` 特性标记这个属性, 并在本例中把 `GuidConverter` 指定为要使用的类型转换器, 这样就可以将复杂的对象(如 `Guid`)正确显示在 Property Browser 中。

2. 定制的类型转换器

如果内置的转换器都不适合, 那么可以创建自己的定制类型转换器。类型转换器派生于 `System.ComponentModel.TypeConverter` 类。程序清单 7-38 中的定制类型转换器可以把定制对象 `Name` 转换为字符串, 也可以把字符串转换为 `Name` 对象。

程序清单 7-38 创建定制的类型转换器

```
using System;
using System.ComponentModel;
using System.Globalization;

public class Name
{
    public Name(string first, string last)
    {
        First = first;
        Last = last;
    }
}
```

```

    }

    public string First { get; set; }
    public string Last { get; set; }
}

public class NameConverter : TypeConverter
{
    public override bool CanConvertFrom(ITypeDescriptorContext context,
        Type sourceType) {
        if (sourceType == typeof(string)) {
            return true;
        }
        return base.CanConvertFrom(context, sourceType);
    }

    public override object ConvertFrom(ITypeDescriptorContext context,
        CultureInfo culture, object value) {
        if (value is string) {
            string[] v = ((string)value).Split(new char[] { ' ' });
            return new Name(v[0], v[1]);
        }
        return base.ConvertFrom(context, culture, value);
    }

    public override object ConvertTo(ITypeDescriptorContext context,
        CultureInfo culture, object value, Type destinationType) {
        if (destinationType == typeof(string)) {
            return ((Name)value).First + " " + ((Name)value).Last;
        }
        return base.ConvertTo(context, culture, value, destinationType);
    }
}

```

NameConverter 类重写了 3 个方法：**CanConvertFrom**、**ConvertFrom** 和 **ConvertTo**。**CanConvertFrom** 方法可以控制转换器能转换的类型；**ConvertFrom** 方法把字符串表示转换回 **Name** 对象；**ConvertTo** 方法把 **Name** 对象转换为字符串表示。

建立好类型转换器后，就可以使用它给控件中的属性添加 **TypeConverter** 特性，如程序清单 7-37 所示。

3. 控件设计器

Visual Studio 设计界面上的控件依赖控件设计器为终端用户创建的设计期间操作。WinForms 和 ASP.NET 的控件设计器都是派生于 **System.ComponentModel.Design.ComponentDesigner** 的类。.NET 提供了抽象基类 **System.Web.UI.Design.ControlDesigner**，专门用于创建 ASP.NET 控件设计器。为了访问这些类，需要在项目中添加对 **System.Design.dll** 程序集的引用。

.NET 有许多内置的控件设计器类，在创建定制控件时可以使用它们。但在开发服务器控件时，.NET 会自动使用默认的设计器。使用的设计器取决于要创建的控件类型。例如，在创建第一个 **TextBox**

控件时, Visual Studio 会使用 `ControlDesigner` 类来获得文本框的所见即所得的设计期间显示。如果是开发派生于 `ControlContainer` 类的服务器控件, .NET 就会自动使用 `ControlContainerDesigner` 类作为设计器。

还可以在控件的类上使用 `Designer` 特性, 明确指定要用于在设计期间显示控件的设计器, 如程序清单 7-39 所示。

程序清单 7-39 给控件类添加 Designer 特性

```
[DefaultProperty("Text")]
[ToolboxData("<{0}:Listing0739 runat=server></{0}:Listing0739>")]
[Designer(typeof(System.Web.UI.Design.ControlDesigner))]
public class Listing0739 : WebControl
```



在程序清单 7-39 所示的代码中, 使用了 `Listing0739` 控件, 在创建该控件的过程中修改了它的名称。如果使用默认名称, 可能得到 `WebCustomControl1` 类。

注意, `Designer` 特性已添加到 `WebCustomControl1` 类。我们指定控件应使用 `ControlDesigner` 类(在 `System.Design` 程序集中)作为其设计器。可以指定的其他内置设计器有:

- `CompositeControlDesigner`
- `TemplatedControlDesigner`
- `DataSourceDesigner`

每个设计器都为控件提供了设计期间的特定操作, 我们可以选择一个适合于所创建的控件类型的设计器。

设计期间的区域

如前所述, ASP.NET 允许创建由其他服务器控件和文本组成的服务器控件。ASP.NET 允许使用设计器区域(`designer regions`)技术来创建在设计期间具有可编辑部分的服务器控件。设计器区域允许创建在一个控件中定义的多个独立区域, 并允许响应由设计区域引发的事件。这个事件可以是设计器在设计界面上绘制一个控件, 也可以是用户单击控件的某个区域, 或是进入或退出模板编辑模式。

为了演示如何使用设计器区域, 下面创建一个可应用于定制控件设计器的容器控件, 如程序清单 7-40 所示。

程序清单 7-40 创建带设计器区域的复合控件

```
[Designer(typeof(MultiRegionControlDesigner))]
[ToolboxData("<{0}:Listing0740 runat=\"server\" width=\"100%\">\" +
    "</{0}:Listing0740>")]
public class Listing0740 : CompositeControl
{
    // Define the templates that represent 2 views on the control
    private ITemplate _view1;
    private ITemplate _view2;
```

```

// These properties are inner properties
[PersistenceMode(PersistenceMode.InnerProperty), DefaultValue(null)]
public virtual ITemplate View1
{
    get{return _view1;}
    set{_view1 = value;}
}

[PersistenceMode(PersistenceMode.InnerProperty), DefaultValue(null)]
public virtual ITemplate View2
{
    get{return _view2;}
    set{_view2 = value;}
}

// The current view on the control; 0= view1, 1=view2, 2=all views
private int _currentView = 0;
public int CurrentView
{
    get{return _currentView;}
    set{_currentView = value;}
}

protected override void CreateChildControls()
{
    Controls.Clear();

    ITemplate template = View1;
    if(_currentView == 1)
        template = View2;

    Panel p = new Panel();
    Controls.Add(p);

    if(template != null)
        template.InstantiateIn(p);
}
}

```

容器控件创建了两个 ITemplate 对象作为要显示的控件。ITemplate 对象是这个服务器控件的控件容器，允许把其他服务器控件或文本拖放到这个控件上。该控件还使用 Designer 特性指示 Visual Studio，在设计界面上显示这个控件时应使用 MultiRegionControlDesigner 类。

下面创建控件设计器，为控件定义区域。程序清单 7-41 列出了设计器类。

程序清单 7-41 用于定义设计器区域的定制设计器类

```

public class MultiRegionControlDesigner :
    System.Web.UI.Design.WebControls.CompositeControlDesigner {

    protected int _currentView = 0;

    private Listing0740 myControl;
}

```

```
public override void Initialize(IComponent component)
{
    base.Initialize(component);
    myControl = (Listing0740)component;
}

public override bool AllowResize {get{return true;}}

protected override void OnClick(DesignerRegionMouseEventArgs e)
{
    if(e.Region == null)
        return;

    if(e.Region.Name == "Header0" && _currentView != 0) {
        _currentView = 0;
        UpdateDesignTimeHtml();
    }

    if(e.Region.Name == "Header1" && _currentView != 1) {
        _currentView = 1;
        UpdateDesignTimeHtml();
    }
}

public override String GetDesignTimeHtml(DesignerRegionCollection regions)
{
    BuildRegions(regions);
    return BuildDesignTimeHtml();
}

protected virtual void BuildRegions(DesignerRegionCollection regions)
{
    regions.Add(new DesignerRegion(this, "Header0"));
    regions.Add(new DesignerRegion(this, "Header1"));

    // If the current view is for all, we need another editable region
    EditableDesignerRegion edr0 = new
        EditableDesignerRegion(this, "Content" + _currentView, false);
    edr0.Description = "Add stuff in here if you dare:";
    regions.Add(edr0);

    // Set the highlight, depending upon the selected region
    if(_currentView == 0 || _currentView == 1)
        regions[_currentView].Highlight = true;
}

protected virtual string BuildDesignTimeHtml()
{
    StringBuilder sb = new StringBuilder();
    sb.Append(BuildBeginDesignTimeHtml());
    sb.Append(BuildContentDesignTimeHtml());
    sb.Append(BuildEndDesignTimeHtml());

    return sb.ToString();
}
```



```

protected virtual String BuildBeginDesignTimeHtml()
{
    // Create the table layout
    StringBuilder sb = new StringBuilder();
    sb.Append("<table ");

    // Styles that we'll use to render for the design-surface
    sb.Append("height='" + myControl.Height.ToString() + "' width='" +
        myControl.Width.ToString() + "'>");

    // Generate the title or caption bar
    sb.Append("<tr height='25px' align='center' " +
        "style='font-family:tahoma;font-size:10pt;font-weight:bold;'>" +
        "<td style='width:50%' " + DesignerRegion.DesignerRegionAttributeName +
        "='0'>");
    sb.Append("Page-View 1</td>");
    sb.Append("<td style='width:50%' " +
        DesignerRegion.DesignerRegionAttributeName + "='1'>");
    sb.Append("Page-View 2</td></tr>");

    return sb.ToString();
}

protected virtual String BuildEndDesignTimeHtml()
{
    return("</table>");
}

protected virtual String BuildContentDesignTimeHtml()
{
    StringBuilder sb = new StringBuilder();
    sb.Append("<td colspan='2' style='");
    sb.Append("background-color:" + myControl.BackColor.Name.ToString() +
        ";<' ");

    sb.Append(DesignerRegion.DesignerRegionAttributeName + "='2'>");

    return sb.ToString();
}

public override string GetEditableDesignerRegionContent
    (EditableDesignerRegion region)
{
    IDesignerHost host =
        (IDesignerHost)Component.Site.GetService(typeof(IDesignerHost));

    if(host != null) {
        ITemplate template = myControl.View1;

        if(region.Name == "Content1")
            template = myControl.View2;

        if(template != null)
            return ControlPersister.PersistTemplate(template, host);
    }
}

```

```

    }

    return String.Empty;
}

public override void SetEditableDesignerRegionContent
    (EditableDesignerRegion region, string content)
{
    int regionIndex = Int32.Parse(region.Name.Substring(7));

    if(content == null)
    {
        if(regionIndex == 0)
            myControl.View1 = null;
        else if(regionIndex == 1)
            myControl.View2 = null;
        return;
    }

    IDesignerHost host =
        (IDesignerHost)Component.Site.GetService(typeof(IDesignerHost));

    if(host != null)
    {
        ITemplate template = ControlParser.ParseTemplate(host, content);

        if(template != null)
        {
            if(regionIndex == 0)
                myControl.View1 = template;
            else if(regionIndex == 1)
                myControl.View2 = template;
        }
    }
}
}

```

设计器重写了 `GetDesignTimeHtml` 方法，调用 `BuildRegions` 和 `BuildDesignTimeHtml` 方法来修改控件在 Visual Studio 设计界面上显示的 HTML。

`BuildRegions` 方法在控件中创建了 3 个设计区域，其中两个是标题区域，另一个是可编辑的内容区域。将这些区域添加到 `DesignerRegionCollection` 中。`BuildDesignTimeHtml` 方法调用 3 个方法来创建控件在设计期间生成的 HTML。

设计器类还包含两个重写的方法：`GetEditableDesignerRegionContent` 和 `SetEditableDesignerRegionContent`，以获得和设置可编辑的设计器区域内容。这两个方法根据当前活动的设计器区域模板，获取或设置相应的内容 HTML。

最后，设计器类包含 `OnClick` 方法，用于响应控件在设计期间引发的单击事件。这个控件使用 `OnClick` 事件切换控件在设计期间显示的当前区域。

在给 Web 窗体添加控件时，可以在两个可编辑的区域间切换，每个区域都会维护自己的内容。图 7-17 显示了控件在 Visual Studio 设计界面上的外观。

从图 7-17 中可以看出，控件包含 3 个独立的设计区域。单击设计区域 1 或 2 时，就会触发设计

器中的 OnClick 方法，并在设计界面上重新绘制控件，改变设计区域 3 中的模板区域。

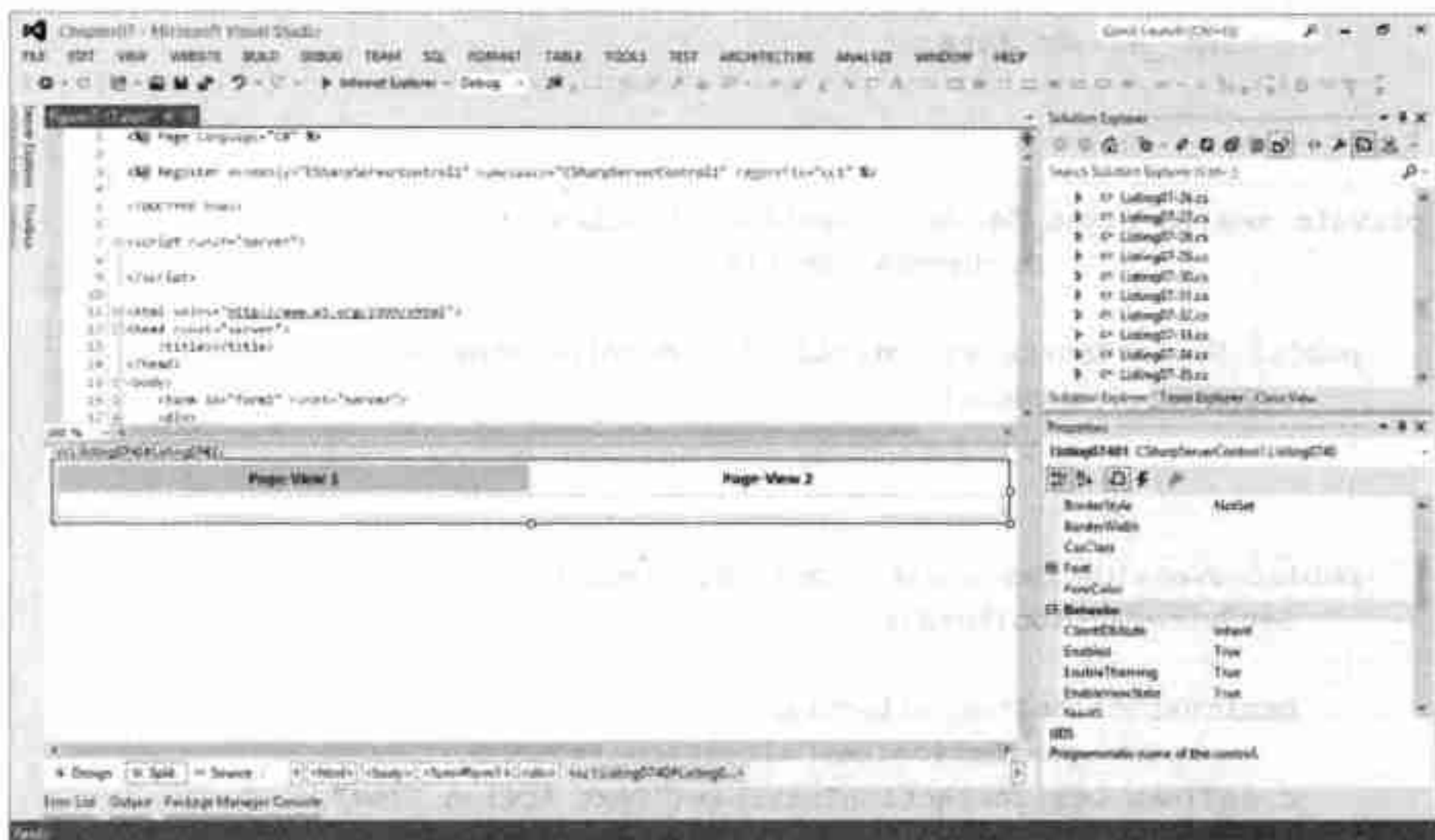


图 7-17

设计器操作

ASP.NET 对设计期间支持的另一个重大功能是控件智能标记。智能标记可以让使用控件的开发人员快速访问常见的控件属性。要在服务器控件的智能标记中添加菜单项，应创建一个继承于 `DesignerActionList` 的新类。`DesignerActionList` 包含服务器控件显示的设计器操作项的列表。派生于 `DesignerActionList` 的类可以重写 `GetSortedActionItems` 方法，创建自己的 `DesignerActionItemsCollection` 对象，并为这个对象添加设计器操作项。

可以在集合中添加几个不同类型的 `DesignerActionItem`：

- `DesignerActionTextItem`
- `DesignerActionHeaderItem`
- `DesignerActionMethodItem`
- `DesignerActionPropertyItem`

程序清单 7-42 中的控件设计器类包含一个派生于 `DesignerActionList` 的私有类。

程序清单 7-42 给控件设计器添加设计器操作

```
public class Listing0742Designer : ControlDesigner
{
    private DesignerActionListCollection _actionLists = null;

    public override DesignerActionListCollection ActionLists
    {
        get
        {
            if (_actionLists == null) {
                _actionLists = new DesignerActionListCollection();
                _actionLists.AddRange(base.ActionLists);
            }
        }
    }
}
```



```

        _actionLists.Add(new ServerControl44ControlList(this));
    }
    return _actionLists;
}

private sealed class ServerControl44ControlList :
    DesignerActionList
{
    public ServerControl44ControlList(ControlDesigner c)
        : base(c.Component)
    {
    }

    public override DesignerActionItemCollection
        GetSortedActionItems()
    {
        DesignerActionItemCollection c =
            new DesignerActionItemCollection();
        c.Add(new DesignerActionTextItem("Text Action Item",
            "Custom Category"));
        return c;
    }
}

```

控件设计器类重写了 `ActionsLists` 属性。该属性创建了 `TextControlList` 类的一个实例，而 `TextControlList` 类派生于 `DesignerActionList`，并重写了 `GetSortedActionItems` 方法。该方法创建了一个新的 `DesignerActionListCollection` 对象，并把 `DesignerActionTextItem` 添加到集合中，如图 7-18 所示。使用 `DesignerActionTextItem` 类可以给智能标记添加文本菜单项。



图 7-18

如图 7-18 所示，在给 Web 页面添加控件时，控件就会有智能标记，内容是 `DesignerActionTextItem` 类。

4. UI 类型编辑器

UI 类型编辑器为控件的用户提供了定制界面，以便直接在 Property Browser 中编辑属性。我们熟悉的一种 UI 类型编辑器是 Color Picker，在修改大多数 ASP.NET 控件的 ForeColor 属性时，就会打开 Color Picker。ASP.NET 拥有许多内置的 UI 类型编辑器，很容易编辑较复杂的属性类型。在 .NET Framework 中查找可用的 UI 类型编辑器时，最简单的方式是在 MSDN Library 帮助中搜索派生于 UITypeEditor 类的类型，或者搜索 Internet。

找到要为控件属性使用的类型编辑器后，就可以使用 Editor 特性把 UI 类型编辑器应用于该属性，如程序清单 7-43 所示。

程序清单 7-43 给控件属性添加 UI 类型编辑器

```
[ToolboxData("<{0}:Listing0743 runat=server></{0}:Listing0743>")]
public class Listing0743 : WebControl
{
    [Bindable(true)]
    [Category("Appearance")]
    [DefaultValue("")]
    [Editor(typeof(System.Web.UI.Design.UrlEditor),
        typeof(System.Drawing.Design.UITypeEditor))]
    public string Url { get; set; }

    protected override void RenderContents(HtmlTextWriter output)
    {
        output.Write(this.Url);
    }
}
```

在这个例子中，为控件创建了 Url 属性。这个属性是 URL，因此可以让控件用户在设计期间更容易完成任务。可以使用 UrlEditor 类型编辑器使用户更容易选择 URL。图 7-19 是用户编辑控件属性时显示的 URL 编辑器。

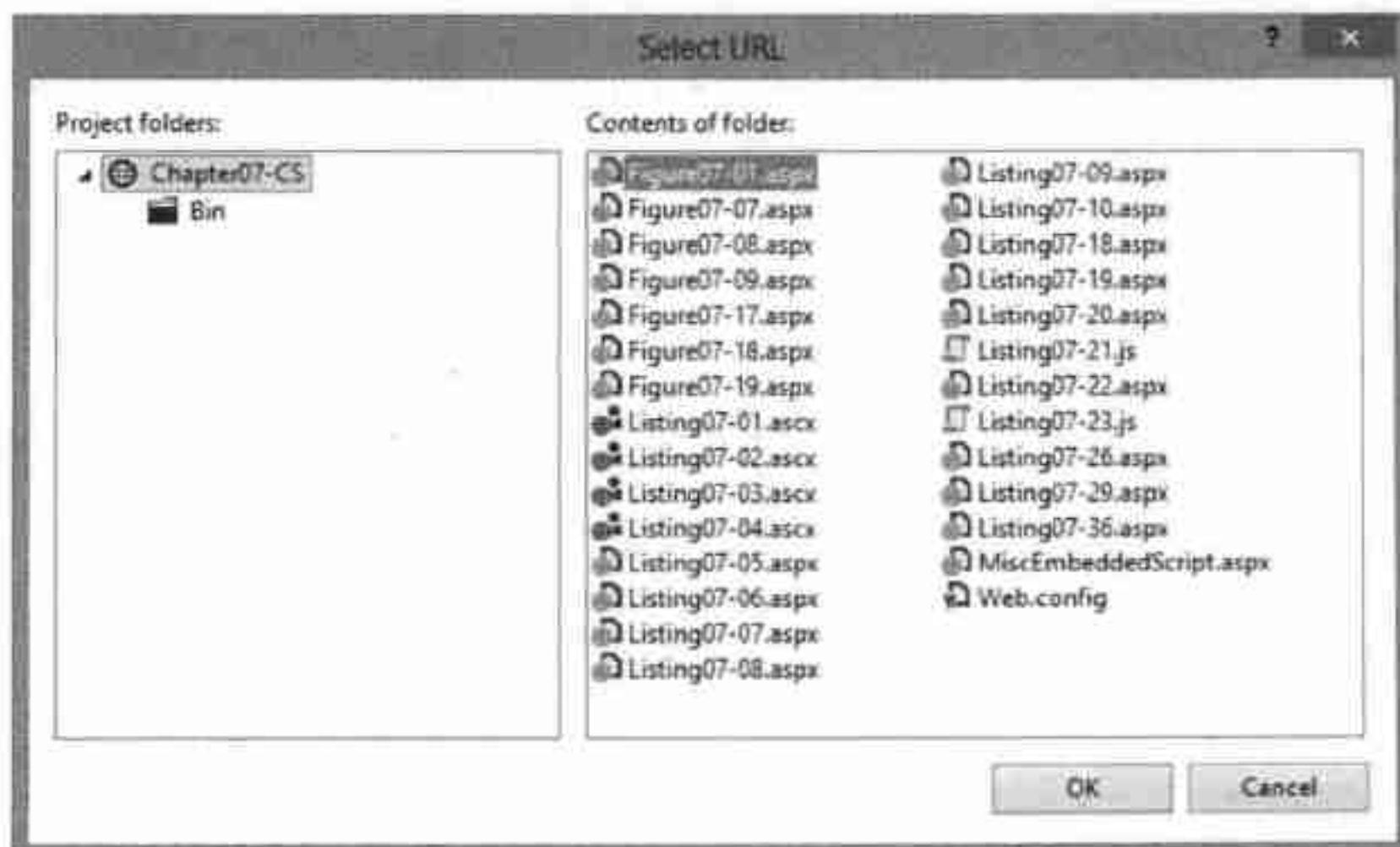


图 7-19

7.3 本章小结

本章介绍了创建可重用的封装代码块的许多方式。首先介绍了用户控件，这是创建控件的最简单形式。我们讨论了如何创建用户控件，如何使它们与 Web 页面交互。创建用户控件非常简单，但这种控件缺乏其他控件创建选项的可移植性。

本章接着论述了如何创建定制的服务器控件。我们学习了编写定制服务器控件的许多工具，包括用于传送 HTML、创建 CSS 样式等方面的工具。本章还讨论了可以创建的服务器控件类型，包括继承自 WebControl 类的服务器控件、模板化控件等，模板化控件可以给控件的用户提供定义服务器控件显示的强大功能。

最后，本章介绍了如何通过使用类型转换器、设计界面的交互操作和服务器控件中的定制属性编辑器，为服务器控件的用户提供设计期间的各种操作方式。

第Ⅲ部分

数 据 访 问

- 第 8 章 数据绑定
- 第 9 章 模型绑定
- 第 10 章 使用 LINQ 查询
- 第 11 章 Entity Framework
- 第 12 章 ASP.NET Dynamic Data
- 第 13 章 使用服务

第 8 章

数 据 绑 定

本章要点

- 使用数据源控件
- 运用内联的数据绑定语法
- 使用数据绑定表达式

ASP.NET 刚开始发布时,最令人兴奋的功能是在运行时把整个数据集合绑定到控件上,并且无须编写大量的代码。控件知道它们与数据绑定,并为数据集合中的每一项显示相应的 HTML。另外,还可以把控件绑定到任意类型的数据源上,从简单的数组到复杂的 Oracle 数据库查询结果。这比传统的 ASP 前进了一大步,在 ASP 中,开发人员需要编写所有的数据访问代码,遍历记录集,并为每条数据记录手动编写相应的 HTML 代码。

本章将探讨服务器端的数据控件,并说明如何使用数据源控件方便快捷地把数据绑定到数据绑定控件上。本章还将讨论 ASP.NET 中数据绑定列表控件的功能,最后介绍内联数据绑定语法和内联 XML 数据绑定。

8.1 数据源控件

在 ASP.NET 1.0/1.1 中,执行数据绑定操作时一般要编写一些数据访问代码,以检索 DataReader 或 DataSet 对象;然后把数据对象绑定到诸如 DataGrid、DropDownList 或 ListBox 这样的服务器控件上。如果要更新或删除绑定的数据,就要编写数据访问代码来实现该操作。程序清单 8-1 是 ASP.NET 1.0/1.1 中数据绑定操作的一个典型例子。

程序清单 8-1 ASP.NET 1.0/1.1 中典型的数据绑定操作

```
SqlConnection conn = new SqlConnection();  
SqlCommand cmd = new SqlCommand("SELECT * FROM Customers", conn);  
SqlDataAdapter da = new SqlDataAdapter(cmd);
```



```
DataSet ds = new DataSet();
da.Fill(ds);
DataGrid1.DataSource = ds;
DataGrid1.DataBind();
```

从 ASP.NET 1.0/1.1 开始, ASP.NET 就通过使用数据源控件引入了一个新的抽象层。如图 8-1 所示, 这些控件抽象了底层数据提供程序的使用, 例如 SQL 数据提供程序或 OLE DB 数据提供程序。也就是说, 不再需要考虑使用数据提供程序的方式和原因, 数据源控件完成了所有这些任务。我们只需知道数据在哪里, 如果必要的话, 还应知道如何为执行 CRUD(Create、Retrieve、Update、Delete, 创建、检索、更新、删除)操作构建查询。

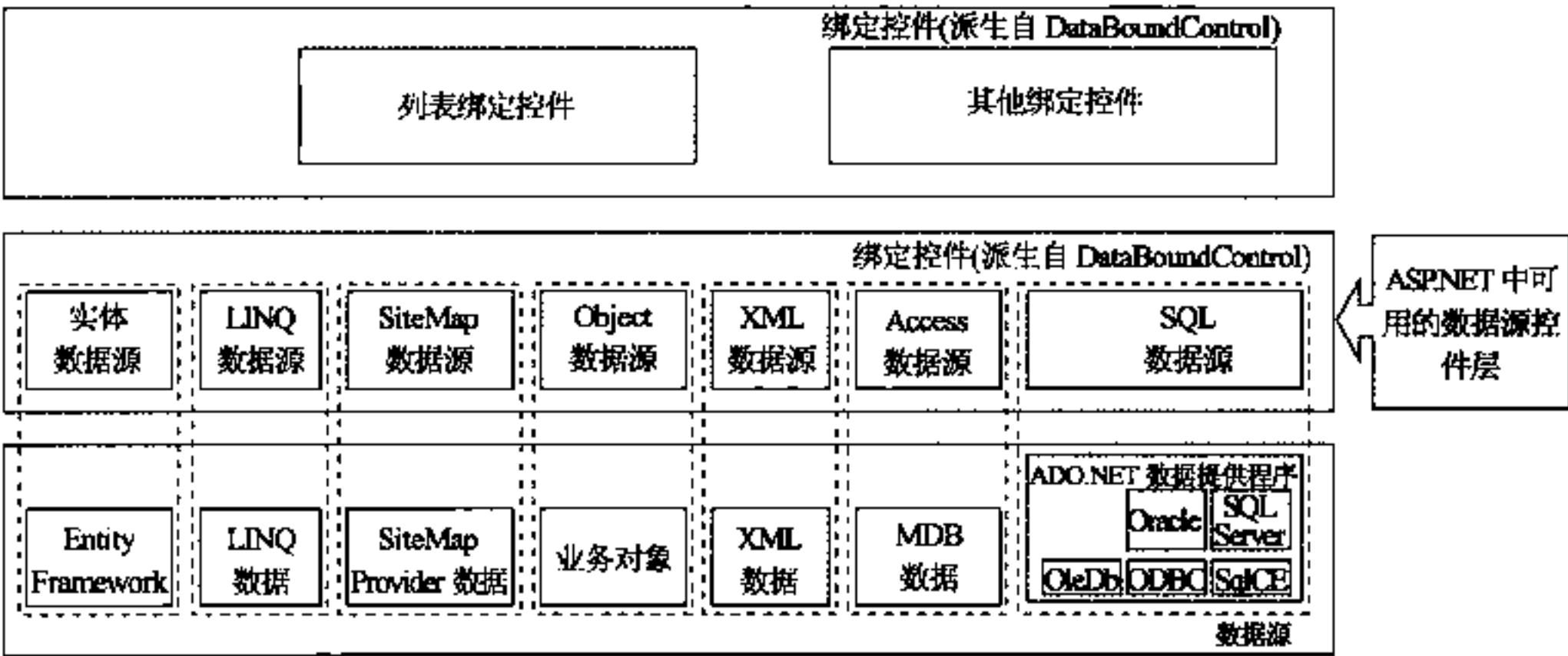


图 8-1

另外, 由于数据源控件都派生于 Control 类, 因此可以像使用其他 Web 服务器控件那样多次使用它们。例如, 可以使用声明性的标记明确定义和控制数据源控件的操作, 或者通过编程方式定义和控制它们。事实上, 虽然肯定可以在代码中控制数据源控件, 但是本章的大多数示例将说明如何只使用 Visual Studio 2012 的向导和声明性语法来执行强大的数据库查询。

ASP.NET 中 7 个内置的数据源控件分别用于特定类型的数据访问。表 8-1 描述了 ASP.NET 中的每个数据源控件。

表 8-1

控 件 名	说 明
SqlDataSource	允许访问支持 ADO.NET 数据提供程序的所有数据源。该控件默认可以访问 ODBC、OLE DB、SQL Server、Oracle 和 SQL Server Compact 提供程序
LinqDataSource	可以使用 LINQ 查询访问不同类型的 LinqToSql 对象
ObjectDataSource	可以对业务对象或其他返回数据的类执行特定的数据访问
XmlDataSource	可以对 XML 文档执行特定的数据访问, 包括物理访问和内存访问
SiteMapDataSource	可以对站点地图提供程序存储的 Web 站点的站点地图数据进行特定访问
AccessDataSource	可以对 Access 数据库执行特定的访问
EntityDataSource	可以对 Entity Framework 模型进行特定的访问

所有的数据源控件都派生于 `DataSourceControl` 类或 `HierarchicalDataSourceControl` 类, 这两个类都派生于 `Control` 类并实现了 `IDataSource` 和 `IListSource` 接口。也就是说, 虽然每个控件都要使用特定的数据源, 但所有的数据源控件共享一组基本的核心功能。这也说明, 很容易根据特定数据源的结构创建定制的数据源控件。

8.1.1 SqlDataSource 控件

如果数据存储在 SQL Server、SQL Server Express、SQL Server LocalDB、SQL Server Compact 或 Oracle 中, 就应使用 `SqlDataSource` 数据源控件。该控件提供了一个易于使用的向导, 引导用户完成整个配置过程; 也可以通过直接在源代码视图中修改控件属性, 手动修改控件。本节介绍如何创建并配置 `SqlDataSource` 控件, 以及如何过滤结果。本书的后面一些章节中将会介绍在与其他数据控件(如 `GridView`)联合使用的情况下, 允许用户使用 `SqlDataSource` 控件更新和删除数据。

打开一个 ASP.NET Web Form 页面, 把 `SqlDataSource` 控件从工具箱拖放到窗体上, 就可以创建该控件。在 Visual Studio 工具箱的 Data 部分中可以找到所有与数据相关的控件。

1. 配置数据连接

把控件拖放到 Web 页面上后, 就要告诉它应使用什么数据库连接。最简单的方式是使用 `Configure Data Source` 向导, 如图 8-2 所示。从数据源控件的智能标记菜单中选择 `Configure Data Source` 命令, 即可启动这个向导。

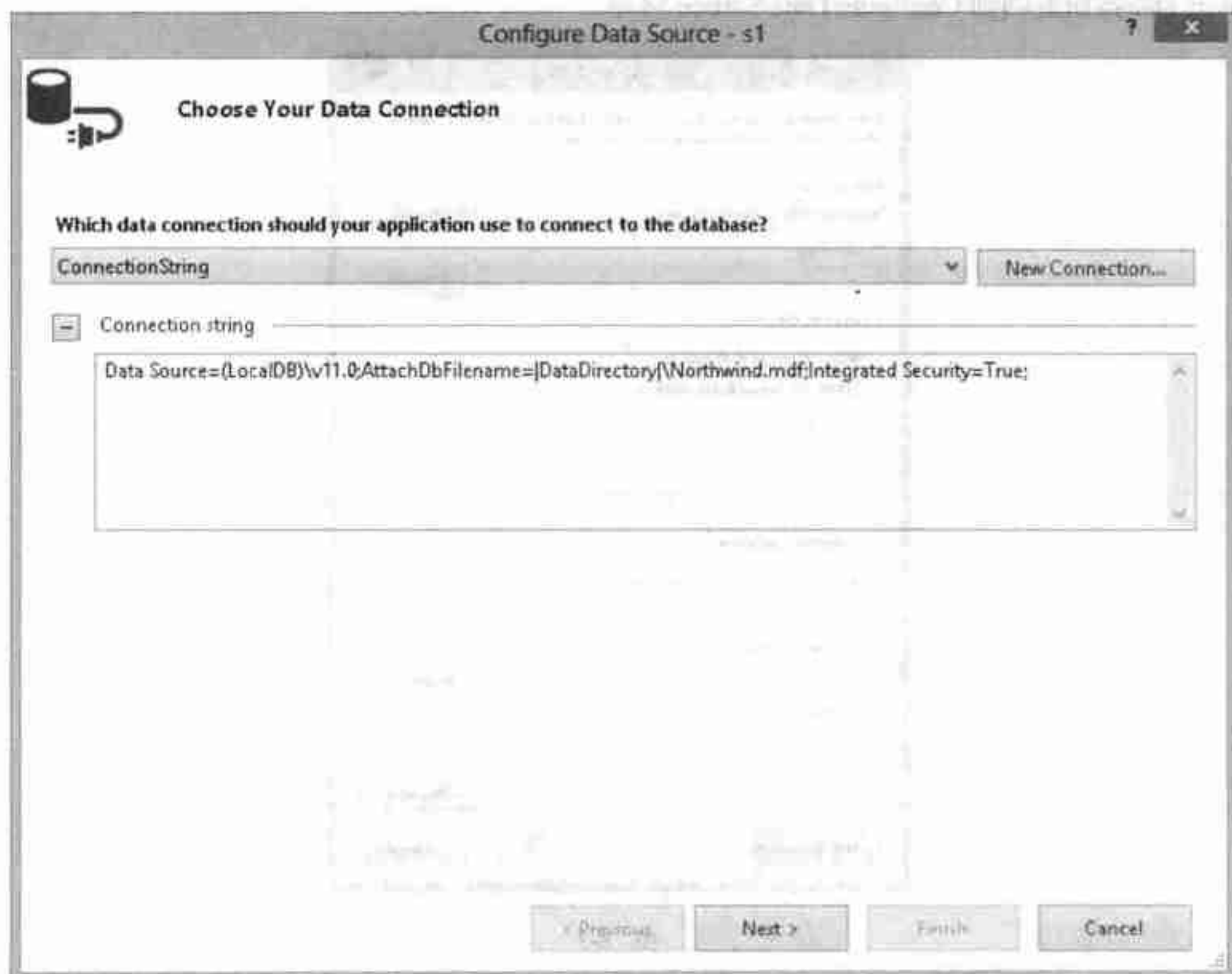


图 8-2

打开向导后, 可以从下拉列表中选择一个已经存在的数据库连接, 也可以创建一个新的连接。本章中的大多数例子将 Northwind 数据库作为数据源, 因此如果相应的连接不存在, 就应创建到 Northwind 数据库的连接。



Northwind 数据库包含为本章下载代码的一部分。

如果要创建新连接, 可单击 New Connection 按钮, 打开 Choose Data Source 对话框。该对话框允许用户为这个连接选择特定的数据源, 以及为选定的数据源选择数据提供程序。



提供程序列表是由 machine.config 文件的 DbProviderFactory 节点中包含的数据生成的。如果还有新的提供程序要在向导中显示, 可以修改该文件, 使其包含该提供程序的信息。

选择好数据源和提供程序以后, 单击 Continue 按钮, 打开 Add Connection 对话框, 该对话框允许用户为新的数据库连接设置所有属性。图 8-3 显示的对话框用于配置 SQL Server 数据库连接。

接着, 给数据库连接填充相关的信息。单击 Test Connection 按钮, 验证连接信息是否正确, 然后单击 OK 按钮返回到 Configure Data Source 向导。



图 8-3

单击 Next 按钮, 继续执行向导的下一步。这一步可以使用户将数据库连接信息保存在 web.config

文件中，从而便于维护和部署应用程序。另外还允许指定连接信息在配置文件中存储时使用的键。如果不在 web.config 文件中存储连接信息，它们就在实际的.aspx 页面上存储为 SqlDataSource 控件的属性，名为 `ConnectionString`。如果所选的提供程序不是 SQL Data Provider，就使用 `ProviderName` 属性存储该设置。

向导的下一步允许选择要从数据库中检索的数据。如图 8-4 所示，这一步提供了一个下拉列表，其中包含数据库中的所有可用表和视图。选择了一个表或视图后，就可以从列表框中选择要包含在查询中的列。可以使用星号(*)选择所有的列，也可以选中每个列名旁边的复选框来选择某些列。

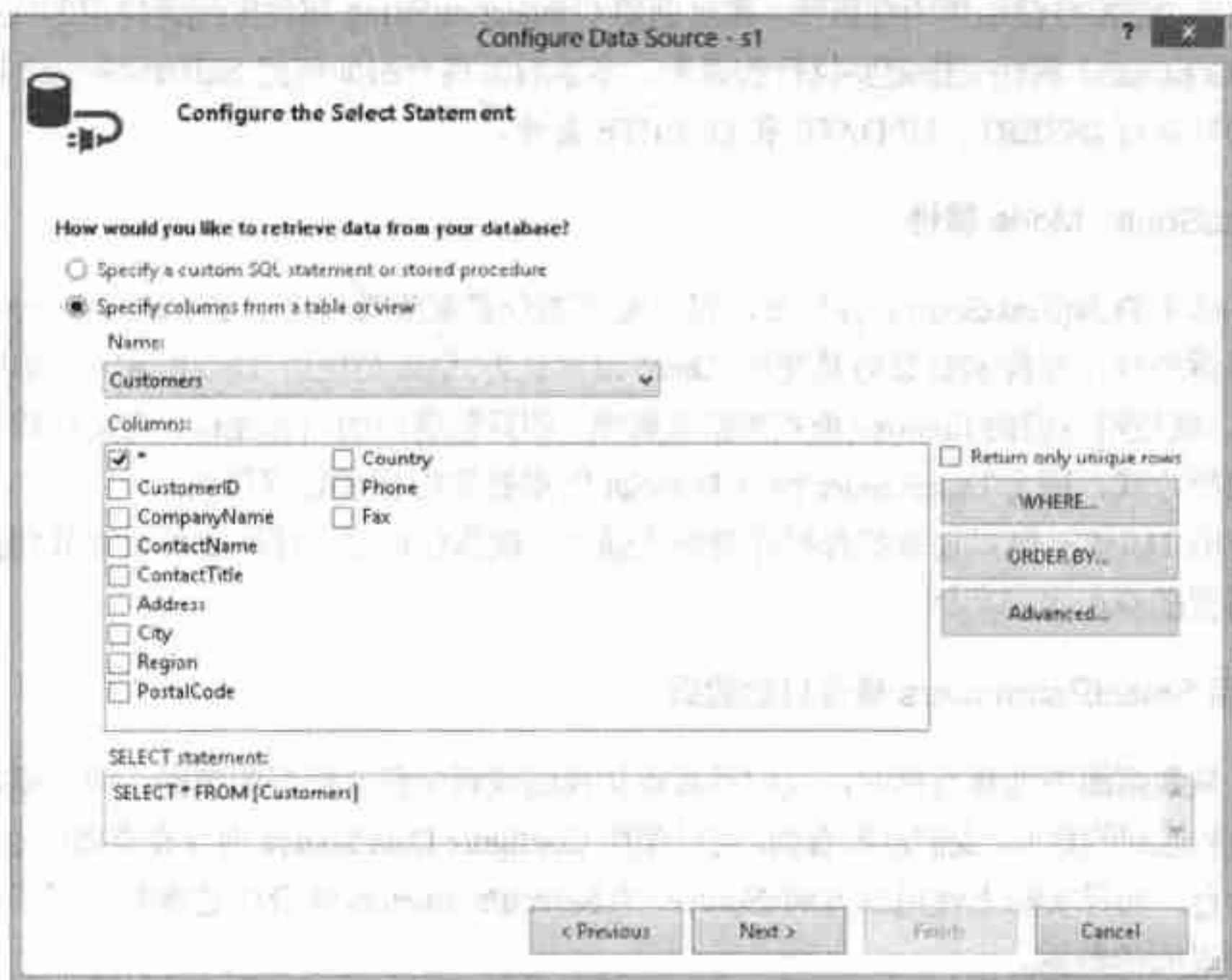


图 8-4

单击 WHERE 或 ORDER BY 按钮，还可以为查询指定用于过滤的 WHERE 子句和用于为查询结果排序的 ORDER BY 子句。目前不需要输入额外的 WHERE 或 ORDER BY 设置。

Advanced 按钮包含两个高级选项。我们可以让向导根据所创建的 SELECT 语句，为数据生成 INSERT、UPDATE 和 DELETE 语句。还可以把数据源控件配置为使用 Optimistic Concurrency。



Optimistic Concurrency 是一种数据库技术，有助于防止对数据的意外覆盖。启用 Optimistic Concurrency 时，SqlDataSource 控件使用的 Update 和 Delete SQL 语句就会改为包含初始值和更新值。执行查询时，会将目标记录中的数据与 SqlDataSource 控件的初始值比较，如果它们有区别，就说明自从 SqlDataSource 控件提取出数据以来，这些数据已经改变，因此不执行 Update 或 Delete 操作。

该向导的最后一步允许用户预览数据源控件所选择的数据，以验证查询是否按照希望的方式执行。单击 Finish 按钮结束该向导。

在配置完数据连接后，切换到 Visual Studio 的源代码视图，查看向导如何为控件生成相应的属性，如程序清单 8-2 所示。

程序清单 8-2 Visual Studio 生成的典型 SqlDataSource 控件

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    SelectCommand="SELECT * FROM [Customers]"
    ConnectionString="<%$ ConnectionStrings:ConnectionString %>" />
```

可以看出，该控件使用声明性语法，通过创建 `ConnectionString` 属性来配置使用的连接，并通过创建 `SelectCommand` 属性来指定要执行的查询。本章后面将介绍如何把 `SqlDataSource` 控件配置为在数据改变时执行 `INSERT`、`UPDATE` 和 `DELETE` 命令。

2. DataSourceMode 属性

建立好基本的 `SqlDataSource` 控件后，可以配置的众多重要属性之一是 `DataSourceMode`。这个属性可以告诉控件，在检索数据时是使用 `DataSet`(默认方式)还是使用 `DataReader`。如果选择使用 `DataReader`，就使用所谓的 `firehose` 模式来检索数据，即只能前向的只读光标。这是从数据源中读取数据的最快捷方式，因为 `DataReader` 没有 `DataSet` 所需要的内存和处理开销。

选择使用 `DataSet` 可以使数据源控件变得更强大，能执行过滤、排序或分页等其他操作。它还支持控件内置的高速缓存功能。

3. 使用 SelectParameters 集合过滤数据

当然，从数据源中选择数据时，我们不希望从视图或表中获取所有的数据，而希望在查询中指定参数以限制返回的数据。我们已经看到，可以使用 `Configure Data Source` 向导在查询中添加 `WHERE` 子句。在后台，向导实际上使用 `SqlDataSource` 的 `SelectParameters` 集合创建参数，用于在运行时过滤从查询中返回的数据。

`SelectParameters` 集合由派生于 `Parameters` 类的类型组成。可以在该集合中合并任意多个参数。数据源控件将使用这些参数创建动态的 SQL 查询。表 8-2 列举并描述了一些可用的参数类型。

表 8-2

参 数	说 明
ControlParameter	使用指定控件的属性值
CookieParameter	使用 cookie 的键值
FormParameter	使用 Forms 集合中的键值
QueryStringParameter	使用 QueryString 集合中的键值
ProfileParameter	使用用户配置文件中的键值
RouteParameter	使用所请求 URL 的路由部分的键值
SessionParameter	使用当前用户的会话中的键值

所有的参数控件都派生于 `Parameters` 类，因此它们都包含几个很有用的共有属性，其中一些属性如表 8-3 所示。

表 8-3

属 性	说 明
Type	允许强类型化参数的值
ConvertEmptyStringToNull	如果赋予控件的值是 System.String.Empty, 就应把它转换为 Null
DefaultValue	如果参数的值是 Null, 就指定其默认值

程序清单8-3中的代码把参数QueryStringParameter添加到SqlDataSource控件的SelectParameters集合中。可以看出, SelectCommand 查询改为包含一条 WHERE 子句。运行这段代码时, 查询字符串字段 ID 的值被绑定到 SelectCommand 中的@CustomerID 占位符, 从而允许只选择其 CustomerID 字段匹配查询字符串字段值的顾客。

程序清单 8-3 使用 SelectParameter 控件过滤选择的数据

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    SelectCommand="SELECT * FROM [Customers]
        WHERE ([CustomerID] = @CustomerID)"
    ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
    DataSourceMode="DataSet">
    <SelectParameters>
        <asp:QueryStringParameter Name="CustomerID"
            QueryStringField="ID" Type="String">
        </asp:QueryStringParameter>
    </SelectParameters>
</asp:SqlDataSource>
```

除了可以使用 Configure Data Source 向导创建 SelectParameters 集合之外, 还可以手动在标记中定义它们, 或者使用 Command and Parameter Editor 对话框创建参数。在设计模式下查看 Web 页面时, 修改 SqlDataSource 控件的 SelectQuery 属性, 就可以打开 Command and Parameter Editor 对话框。

SqlDataSource 控件包含过滤查询结果的另一种方法, 即 FilterParameters。FilterParameters 具有与 SelectParameters 相同的基本过滤功能, 但它使用的是一种不同的技术, 本章后面部分将讨论 SelectParameters 和 FilterParameters 之间的具体区别。

4. ConflictDetection 属性

ConflictDetection 属性可以告诉 SqlDataSource 控件, 正在更新的值和数据库中的值是否冲突。它可以避免用户无意中覆盖彼此更新的数据。当该属性的值设置为 OverwriteChanges 时, 就使用用户最后更新的值。

如果该属性的值设置为 CompareAllValues, 数据源控件就会比较原始数据值(检索出来的值)和数据库中的当前数据值。如果控件检测到两者不同, 就不允许更新; 否则, 就更新数据库。

确定更新是否遇到并发错误的一种方式是在 SqlDataSource 的 Updated 事件中测试 AffectedRows 属性。程序清单 8-4 演示了这样一种方式。

程序清单 8-4 在更新数据后检测并发错误

```
protected void SqlDataSource1_Updated(object sender,
```



```

        SqlDataSourceStatusEventArgs e)
    {
        if (e.AffectedRows > 0)
            this.lblMessage.Text = "The record has been updated";
        else
            this.lblMessage.Text = "Possible concurrency violation";
    }

```

5. 处理数据库错误

数据源控件的事件非常有助于捕获和处理在对数据库执行 SQL 命令时发生的错误。例如，程序清单 8-5 演示了如何使用 `SqlDataSource` 控件的 `Updated` 事件处理作为异常返回给应用程序的数据库错误。

程序清单 8-5 使用 `SqlDataSource` 控件的 `Updated` 事件处理数据库错误

```

protected void SqlDataSource1_Updated(object sender,
    System.Web.UI.WebControls.SqlDataSourceStatusEventArgs e)
{
    if (e.Exception != null)
    {
        this.lblMessage.Text = e.Exception.Message;
        e.ExceptionHandled = true;
    }
}

```

这个示例的一个要点是设置 `ExceptionHandled` 属性的代码。这个属性默认返回 `false`。因此，即使检测到 `Exception` 属性不为 `null` 并尝试处理错误，该异常仍然会返回给应用程序。把 `ExceptionHandled` 属性设置为 `true`，就告诉 .NET 已成功地处理了异常，应用程序可以安全地继续执行。

8.1.2 AccessDataSource 控件

该控件通过使用 Jet Data 提供程序提供了一种访问 Access 数据库的特殊方式，但是由于派生自 `SqlDataSource`，该控件仍然使用 SQL 命令来执行数据检索操作。

8.1.3 LinqDataSource 控件

如同 `SqlDataSource` 控件能够将自己的属性设置转换为 SQL 查询语句，从而为 SQL 数据库生成查询一样，`LinqDataSource` 控件也可以为应用程序中的 `LinqToSQL` 数据对象生成 LINQ 查询。

8.1.4 EntityDataSource 控件

`EntityDataSource` 控件是个特殊的数据源控件，是专门为使用 ADO.NET Entity Framework 的应用程序而设计的。



有关 Entity Framework 的深入讨论参见第 11 章。

`EntityDataSource` 控件可以在 Web 页面上为数据控件处理选择、更新、插入、删除数据的操作，同时还可以自动地对数据进行排序和分页，从而更容易地绑定数据，以及方便地对 Entity Framework

模型中的数据导航。与所有其他的数据源控件一样,在使用 EntityDataSource 控件时,首先要将该控件从工具箱拖放到设计界面上,然后从控件的智能标记中选择 **Configure** 选项来启动控件的配置向导。打开配置向导后,为控件选择或提供一个连接字符串,然后选择默认的容器名称并单击 **Next** 按钮。图 8-5 显示了选择连接到 Northwind 数据库的数据模型的例子。



图 8-5

下一个界面允许选择希望在数据源控件中显示的特定 EntitySetName。如果想要确保查询只返回某个特定类型,还可以指定 EntityTypeFilter。选择 EntitySetName 后,可以通过从列表中选择属性来为它们创建定制的投影。图 8-6 展示了从 Northwind 数据模型中选择 Customers EntitySetName。



图 8-6

最后，在该屏幕上可以配置控件以允许自动插入、更新和删除数据。注意，如果已经通过选择特定的列创建了一个定制的投影，就不能使用这些选项。

完成对数据源控件的配置之后，就可以像平常一样绑定任何数据绑定控件。程序清单 8-6 显示了配置向导生成的标记。

程序清单 8-6 EntityDataSource 配置向导生成的标记

```
<asp:EntityDataSource ID="EntityDataSource1" runat="server"
  ConnectionString="name=NorthwindEntities"
  DefaultContainerName="NorthwindEntities" EnableDelete="True"
  EnableFlattening="False" EnableInsert="True" EnableUpdate="True"
  EntitySetName="Customers" EntityTypeFilter="Customer">
</asp:EntityDataSource>
```

EntityDataSource 控件包括各种各样的其他属性和事件，可以用来进一步定制控件的行为。

- 使用 CommandText 属性可以完全定制控件用于选择数据的查询操作。该属性接受 Entity SQL 命令作为输入。如果同时设置 CommandText 和 EntitySetName，控件就会抛出 InvalidOperationException 异常。
- EntityDataSource 控件还允许通过 GroupBy 和 Where 属性对数据进行分组和过滤。这两个属性接受 Entity SQL 表达式，分别用于指定分组操作和过滤操作。
- 使用 CommandText 或 GroupBy 属性可以使数据源控件生成定制的投影，根据本章前面的讨论，这意味着控件将会忽略插入、更新和删除操作，即使已经明确地启用这些操作。

与其他任何数据源控件一样，可以指定用于该控件的所有查询中的参数。

8.1.5 给复杂的过滤使用 QueryExtender

尽管 EntityDataSource 和 LinqDataSource 控件包含过滤数据的内置功能，但它们不具备 LINQ 为创建包含过滤的查询所需的所有功能。此时可以使用 QueryExtender，它允许定义复杂的搜索、数据范围过滤、复杂的多列 OrderBy 子句，甚至完全自定义的表达式。表 8-4 列出了可用的过滤表达式类型。

表 8-4

表达式类型	说 明
SearchExpression	在字段中搜索字符串值，把它们与指定的字符串相比较。该表达式可以执行 StartsWith、EndsWith 或 Contains 搜索
RangeExpression	与 SearchExpression 类似，但使用一对值来定义最小和最大范围
PropertyExpression	比较列的属性值和指定值
OrderByExpression	可以用指定的列和排序方向对数据排序
CustomExpression	可以提供自定义的 LINQ 表达式
MethodExpression	可以调用包含自定义 LINQ 查询的方法
OfTypeExpression	可以按类型过滤数据源中的元素

QueryExtender 使用实现了 IQueryableDataSource 接口的任意数据源控件，默认包含

LinqDataSource 和 EntityDataSource 控件。

为了说明如何使用 QueryExtender, 首先在工具箱中把一个 QueryExtender 控件拖放到设计界面上。在页面的标记中, 将 EntityDataSource 控件的 ID 指定为 QueryExtender 的 TargetControlID 属性值, 把 QueryExtender 连接到 EntityDataSource。现在只需在 QueryExtender 中定义要使用的过滤表达式。程序清单 8-7 演示了如何使用 SearchExpression。

程序清单 8-7 使用 QueryExtender 控件过滤查询结果

```
<asp:EntityDataSource ID="EntityDataSource1" runat="server"
    ConnectionString="name=NorthwindEntities"
    DefaultContainerName="NorthwindEntities" EnableDelete="True"
    EnableFlattening="False" EnableInsert="True" EnableUpdate="True"
    EntitySetName="Customers" EntityTypeFilter="Customer">
</asp:EntityDataSource>
<asp:QueryExtender ID="QueryExtender2"
    runat="server" TargetControlID="EntityDataSource1">
    <asp:SearchExpression SearchType="StartsWith"
        DataFields="CustomerID">
        <asp:QueryStringParameter DefaultValue="A"
            QueryStringField="search" />
    </asp:SearchExpression>
</asp:QueryExtender>
```

可以看出, 表达式使用 QueryStringParameter, 把查询结果过滤为只包含以查询字符串字段 search 指定的值开头的 CustomerID。

8.1.6 XmlDataSource 控件

XmlDataSource 控件提供了绑定内存中或物理磁盘上的 XML 文档的一种简单方式。该控件有许多属性, 便于指定包含数据的 XML 文件以及把源 XML 转换为更合适格式的 XSLT 转换文件。还可以提供 XPath 查询, 以选择某个数据子集。

程序清单 8-8 显示了如何使用 MSDN Web 站点的 RSS 源, 选择其中所有的项节点, 绑定到一个列表控件(如 GridView)上。

程序清单 8-8 通过 XmlDataSource 控件使用 RSS 源

```
<asp:XmlDataSource ID="XmlDataSource1" runat="server"
    DataFile="http://msdn.microsoft.com/rss.xml"
    XPath="rss/channel/item" />
```

除了可以与 XmlDataSource 一起使用的声明性属性之外, 还可以使用其他有帮助的属性和事件。

有时 XML 没有存储在物理文件中, 而是作为字符串存储在应用程序内存或数据库中。该控件提供的 Data 属性可以包含控件能够绑定的简单 XML 字符串。注意, 如果同时设置 Data 属性和 DataFile 属性, 那么 DataFile 属性将优先于 Data 属性。

另外, 在某些情况下, 要把绑定的 XML 从 XmlDataSource 控件导出到其他对象, 如果将 XML 绑定到 GridView 等控件, 就要保存对底层 XML 的修改。XmlDataSource 控件提供了两个方法来完成这项任务。第一个方法是 GetXmlDocument, 它可以通过返回一个基本的 System.Xml.XmlDocument 对象来导出 XML, 该 XmlDocument 对象包含数据源控件中加载的 XML。第二个方法是控件的 Save 方

法,可以把对 XmlDataSource 控件中 XML 的修改持久保存到磁盘上。执行这个方法需要在 DataFile 属性中提供文件路径。

XmlDataSource 控件还提供了许多专用的事件。Transforming 事件在应用 Transform 或 TransformFile 属性中指定的 XSLT 转换之前触发,可以为转换过程提供定制参数。

8.1.7 ObjectDataSource 控件

ObjectDataSource 控件可以把数据控件直接绑定到中间层业务对象,这些业务对象可以采用硬编码方式,也可以自动从程序(如 Object Relational (O/R)映射程序)中生成。

为了演示如何使用 ObjectDataSource 控件,在项目中创建两个类,一个是 Customer 类,另一个类包含可以从集合中选择、插入、更新和删除 Customer 对象的方法。程序清单 8-9 显示了用于演示的类。

程序清单 8-9 创建 Customer 类和 CustomerRepository 类

```
public class Customer
{
    public string CustomerID{get; set;}
    public string CompanyName{get; set;}
    public string ContactName{get; set;}
    public string ContactTitle{get; set;}
}
public class CustomerRepository
{
    public CustomerRepository()
    {
    }
    public List<Customer> Select(string customerId)
    {
        // Implement logic here to retrieve the Customer
        // data based on the methods customerId parameter
        List<Customer> _customers = new List<Customer>();
        _customers.Add(new Customer() {
            CompanyName = "Acme", ContactName = "Wiley Cyote",
            ContactTitle = "President", CustomerID = "ACMEC" });
        return _customers;
    }
    public void Insert(Customer c)
    {
        // Implement Insert logic
    }
    public void Update(Customer c)
    {
        // Implement Update logic
    }
    public void Delete(Customer c)
    {
        // Implement Delete logic
    }
}
```

要开始使用 ObjectDataSource,应首先把该控件拖放到设计界面上。利用该控件的智能标记

打开配置向导。向导打开后，从下拉列表中选择要用作数据源的业务对象。下拉列表显示了 Web 站点的 App_Code 文件夹中可以成功编译的所有类。这里想要使用程序清单 8-9 中创建的 CustomerRepository 类。

ObjectDataSource 控件用于执行 CRUD 操作的方法必须遵循一些规则，这样控件才能理解。例如，控件的 SELECT 方法必须返回 DataSet、DataReader 或强类型化的集合。控件的每个操作选项卡都说明了控件希望指定的方法用于执行什么操作。另外，如果方法不遵循特定操作要求的规则，就不会显示在该选项卡的下拉列表中。

最后，如果 SELECT 方法包含参数，向导就要创建 SelectParameters，用来向方法提供参数数据。配置完 ObjectDataSource 后，页面中就包含了程序清单 8-10 所示的源代码。

程序清单 8-10 配置向导生成的 ObjectDataSource 代码

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server"
    DeleteMethod="Delete" InsertMethod="Insert"
    SelectMethod="Select" TypeName="CustomerRepository"
    UpdateMethod="Update" DataObjectTypeName="Customer">
    <SelectParameters>
        <asp:QueryStringParameter Name="customerId"
            QueryStringField="ID" Type="string" />
    </SelectParameters>
</asp:ObjectDataSource>
```

可以看出，向导为指定的 SELECT、UPDATE、INSERT 和 DELETE 方法生成了属性。另外，该向导还添加了 Select 参数。根据应用程序，可以把该参数改为前面讨论的任意 Parameter 对象，如 ControlParameter 或 QueryStringParameter 对象。

8.1.8 SiteMapDataSource 控件

SiteMapDataSource 控件可以处理存储在 Web 站点的 SiteMap 配置文件中的数据(如果站点中有该文件的话)。如果需要在运行时根据用户的权限或状态改变站点地图数据，该控件会很有帮助。

关于 SiteMapDataSource 控件有两个地方值得注意：

- 首先，SiteMapDataSource 控件不支持其他数据源控件都有的任何数据高速缓存选项(8.2 节将介绍)，因此不能自动地高速缓存站点地图数据。
- 其次，SiteMapDataSource 控件没有像其他数据源控件那样的配置向导，这是因为该 SiteMap 控件只能绑定到 Web 站点的 SiteMap 配置数据文件，所以不会有其他配置。



SiteMapDataSource 控件的 SiteMapProvider 属性允许指定要使用的站点地图提供程序。作为规则，使用默认的站点地图提供程序，但也可以指定其他的站点地图提供程序。更多信息参见 <http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.sitemapdatasource.sitemapprovider>。

8.2 数据源控件的高速缓存

ASP.NET 包含强大的高速缓存基础结构, 允许在固定的时间段内高速缓存服务器中的任意对象。第 22 章将会介绍有关 ASP.NET 高速缓存的更多知识。

数据源控件可以利用这种内置的高速缓存基础结构, 轻松地高速缓存查询结果。所有控件都使用相同的基本语法来配置高速缓存, 允许创建简单的高速缓存策略, 包括高速缓存方向、过期策略和键依赖。



只有把 `DataSourceMode` 属性设置为 `DataSet`, 才能使用 `SqlDataSource` 控件的高速缓存功能。如果设置为 `DataReader`, 控件就会抛出 `NotSupportedException` 异常。

高速缓存的持续时间可以设置为特定的时间长度, 也可以设置为 `Infinite`, 使高速缓存的数据永远不过期。程序清单 8-11 说明了如何给数据源控件添加高速缓存功能。

程序清单 8-11 启用 `SqlDataSource` 控件的高速缓存功能

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
  SelectCommand="SELECT * FROM [Customers]"
  ConnectionString="<%%$ ConnectionStrings:ConnectionString %>"
  DataSourceMode="DataSet" ConflictDetection="CompareAllValues"
  EnableCaching="True" CacheKeyDependency="SomeKey"
  CacheDuration="Infinite" />
```

一些控件还添加了专门用于某个数据源的其他高速缓存功能, 扩展了这个核心高速缓存功能集。例如, 如果使用 `SqlDataSource` 控件, 就可以使用该控件的 `SqlCacheDependency` 属性创建 SQL 依赖。

8.3 数据绑定控件

本节介绍一些数据绑定控件, 它们可以绑定到数据源控件上, 并可以显示、创建、编辑和删除数据。

8.3.1 GridView 控件

`GridView` 控件是功能非常强大的数据网格控件, 可以显示整个数据集合, 添加排序和分页功能, 执行内联编辑。

开始使用 `GridView` 时, 先把该控件拖放到 ASP.NET Web 页面的设计界面上, 此时 ASP.NET 会提示选择要绑定到网格上的数据源控件。可以使用本章前面创建的 `SqlDataSource` 控件。

程序清单 8-12 显示了 `GridView` 在 `SqlDataSource` 控件中的简单使用。这个示例移除了明确的字段定义, 从而允许控件基于数据源的结构自动生成列。

程序清单 8-12 在 ASP.NET Web 页面中使用 GridView 控件

```

<html>
<head runat="server">
  <title>Using the GridView Control</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:GridView ID="GridView1" runat="server"
        DataSourceID="SqlDataSource1">
      </asp:GridView>
      <asp:SqlDataSource ID="SqlDataSource1" runat="server"
        SelectCommand="SELECT * FROM [Customers]"
        ConnectionString="<%%$ ConnectionStrings:ConnectionString %>"
        DataSourceMode="DataSet"
        ConflictDetection="CompareAllValues" EnableCaching="True"
        CacheKeyDependency="MyKey" CacheDuration="Infinite">
      </asp:SqlDataSource>
    </div>
  </form>
</body>
</html>

```

运行该页面时, ASP.NET 使用 `SqlDataSource` 控件执行数据库查询, 并将结果绑定到 `GridView` 控件。`GridView` 控件生成包含查询返回的所有数据的表布局。

给 `GridView` 指定了数据源后, 正常情况下网格会自行更新以保持与数据源中的数据相匹配。将 `AutoGenerateFields` 属性设置为 `false`, 并在 `GridView` 的列集合中为数据源中的每个公有属性或数据库表列生成一个字段。程序清单 8-13 显示了明确定义的列集合。

程序清单 8-13 明确定义的 GridView 列

```

<asp:GridView ID="GridView1" runat="server" DataSourceID="SqlDataSource1"
  AutoGenerateColumns="False" DataKeyNames="CustomerID">
  <Columns>
    <asp:BoundField DataField="CustomerID"
      HeaderText="CustomerID" ReadOnly="True"
      SortExpression="CustomerID" />
    <asp:BoundField DataField="CompanyName"
      HeaderText="CompanyName" SortExpression="CompanyName" />
    <asp:BoundField DataField="ContactName"
      HeaderText="ContactName" SortExpression="ContactName" />
    <asp:BoundField DataField="ContactTitle"
      HeaderText="ContactTitle" SortExpression="ContactTitle" />
    <asp:BoundField DataField="Address" HeaderText="Address"
      SortExpression="Address" />
    <asp:BoundField DataField="City" HeaderText="City"
      SortExpression="City" />
    <asp:BoundField DataField="Region" HeaderText="Region"
      SortExpression="Region" />
    <asp:BoundField DataField="PostalCode" HeaderText="PostalCode"
      SortExpression="PostalCode" />
    <asp:BoundField DataField="Country" HeaderText="Country"

```

```
SortExpression="Country" />
<asp:BoundField DataField="Phone" HeaderText="Phone"
SortExpression="Phone" />
<asp:BoundField DataField="Fax" HeaderText="Fax"
SortExpression="Fax" />
</Columns>
</asp:GridView>
```

注意，在创建列定义时，控件默认为每一列都使用 BoundField 类型。每个 BoundField 都有 DataField 属性(该属性将字段和数据源的某个属性连接起来)和定义好的 SortExpression 属性。该控件还检测数据源中的只读属性并设置字段的 ReadOnly 属性。

GridView 控件在显示时会触发一些事件，可以使用这些事件改变该控件的输出，或者向应用程序添加额外的定制逻辑。表 8-5 是对这些事件的描述。

表 8-5

事件名	说明
DataBinding	在计算 GridView 的数据绑定表达式时触发
RowCreated	每次在网格中创建新行时触发。在显示网格前，必须为控件中的每一行创建 GridViewRow 对象。RowCreated 事件可以在创建行时为行插入定制的内容
RowDataBound	把 GridViewRow 绑定到数据源中的相应数据时触发该事件。这个事件可以计算绑定到当前行的数据，并根据需要改变输出
DataBound	绑定完成并准备显示 GridView 时触发该事件

RowDataBound 事件特别有用，可以在绑定过程中为绑定到 GridView 的每一个数据源项插入相应的逻辑。程序清单 8-14 说明了如何使用 RowDataBound 事件检查绑定到当前网格行的数据，并插入专门的逻辑，在此例中是检查 Region 项的值是否为 null。如果是，该逻辑就修改 GridView 中该行的 ForeColor 和 BackColor 属性。

程序清单 8-14 使用 RowDataBound 插入定制的显示逻辑

```
<script runat="server">
protected void GridView1_RowDataBound(object sender,
GridViewRowEventArgs e)
{
if(e.Row.DataItem != null)
{
System.Data.DataRowView drv =
(System.Data.DataRowView)e.Row.DataItem;
if(drv["Region"] == System.DBNull.Value)
{
e.Row.BackColor = System.Drawing.Color.Red;
e.Row.ForeColor = System.Drawing.Color.White;
}
}
}
</script>
```

GridView 还包含对应于选择、插入、更新和删除数据的事件，这些事件详见本章后面的内容。

1. 处理没有相应数据的状况

有时绑定到 GridView 的数据源不包含控件要绑定的数据。在这种情况下, 需要为终端用户提供一些反馈, 通知他们没有控件要绑定的数据。GridView 为解决此问题提供了两种机制。

第一种机制是使用 EmptyDataText 属性。这个属性可以指定一个文本字符串, 当没有 GridView 控件要绑定的数据时, 就将该文本字符串显示给用户。加载 ASP.NET 页面并且 GridView 控件确定在其绑定的数据源中没有数据时, 就会创建包含 EmptyDataText 属性值的特殊 DataRow, 并显示给用户。程序清单 8-15 说明了如何给 GridView 添加这个属性。

程序清单 8-15 给 GridView 添加 EmptyDataText 属性

```
<asp:GridView ID="GridView1" runat="server"
    DataSourceID="SqlDataSource1" DataKeyNames="CustomerID"
    AutoGenerateColumns="True"
    EmptyDataText="No data was found using your query"></asp:GridView>
```

另一种机制是使用 EmptyDataTemplate 控件模板, 在不存在控件要绑定的数据时, 该模板可以完全定制用户看到的特殊数据行。



控件模板只是容器, 可用于添加其他内容, 例如文本、HTML 控件, 甚至 ASP.NET 控件。GridView 控件为不同的情形提供了各种模板, 包括 EmptyDataTemplate 模板。本章将介绍这些模板。

在 Visual Studio 设计界面中可以通过两种方式访问该模板。第一种方式是右击 GridView 控件, 在上下文菜单中展开 Edit Template 选项, 从中选择 EmptyDataTemplate 命令。访问该模板的第二种方式是从 GridView 控件的智能标记中选择 Edit Templates 选项。这会使 GridView 进入模板编辑模式, 并显示一个对话框, 从中可以选择要编辑的模板。在这里只需要从下拉列表中选择 EmptyDataTemplate 选项即可, 如图 8-7 所示。

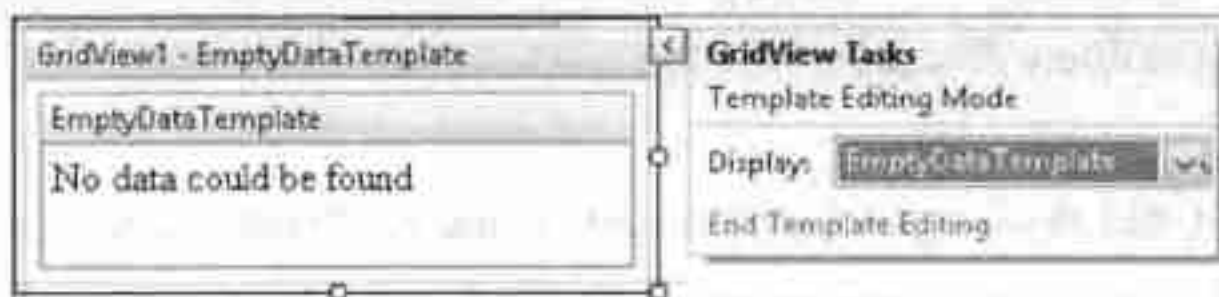


图 8-7

进入模板编辑模式后, 就可以在设计界面的模板编辑器上添加定制的文本或控件。编辑完模板后, 右击 GridView 控件或打开 GridView 控件的智能标记, 选择 End Template Editing 选项。

切换到源代码视图, 会看到 <EmptyDataTemplate> 元素已添加到 GridView 控件中。该元素应包含在编辑模板时添加的所有内容。程序清单 8-16 是一个 EmptyDataTemplate 示例。

程序清单 8-16 使用 EmptyDataTemplate

```
<EmptyDataTemplate>
    No data could be found based on your query parameters.
    Please enter a new query.
```

```
</EmptyDataTemplate>
```

当然，也可以在源代码视图中添加模板及其内容。

如果在绑定到数据源的过程中遇到 Null 值，GridView 控件还允许配置一个值来显示。例如，使用 <asp:BoundField> 控件添加一列，如程序清单 8-17 所示。

程序清单 8-17 使用 Null 值

```
<asp:BoundField DataField="Region" HeaderText="Region"
  NullDisplayText="N/A" SortExpression="Region" />
```

这里通过配置 <asp:BoundField> 控件来显示 Customers 表的 Region 列。在查看 Region 列中的数据时，注意并不是每一行都有值。如果不想显示空白单元格来表示空值，就可以使用 NullDisplayText 属性来配置 GridView 控件，从而显示文本以取代列中的空项。

2. 列的排序

对数据排序的功能是用户导航数据时的最基本的工具之一。在 GridView 控件中，只需把 AllowSorting 属性设置为 True，即可启用列的排序功能。控件会在内部完成所有的排序逻辑。程序清单 8-18 说明了如何在网格中添加这个属性。

程序清单 8-18 给 GridView 控件添加排序功能

```
<asp:GridView ID="GridView1" runat="server"
  DataSourceID="SqlDataSource1" DataKeyNames="CustomerID"
  AutoGenerateColumns="True" AllowSorting="True"></asp:GridView>
```

启用排序功能后，所有网格列的标题就都变成了超链接。

GridView 控件还可以按升序或降序来排序。如果重复单击列标题，排序顺序就会在升序和降序之间来回切换。GridView 的 Sort 方法还可以接受多个 SortExpression，从而可以进行多列排序。程序清单 8-19 说明了如何使用 GridView 的 Sorting 事件实现多列排序。

程序清单 8-19 向 GridView 添加多列排序功能

```
<script runat="server">
  protected void GridView1_Sorting(object sender, GridViewSortEventArgs e)
  {
    string oldExpression = GridView1.SortExpression;
    string newExpression = e.SortExpression;
    if(oldExpression.IndexOf(newExpression) < 0)
    {
      if(oldExpression.Length > 0)
        e.SortExpression = newExpression + "," + oldExpression;
      else
        e.SortExpression = newExpression;
    }
    else
    {
      e.SortExpression = oldExpression;
    }
  }
}
```



```

    }
</script>

```

上面的程序清单使用 `Sorting` 事件控制 `GridView` 控件的 `SortExpression` 属性值。`Sorting` 事件的参数允许检查当前的排序表达式, 确定排序方向, 甚至可以取消排序操作。`GridView` 还提供了 `Sorted` 事件, 它在排序完成后触发。

3. 启用 `GridView` 的数据分页功能

`GridView` 控件还允许方便地添加另一个常用的网格功能——分页。将 `AllowPaging` 属性的值设置为 `True`, 或者在 `GridView` 的智能标记中选中 `Enable Paging` 复选框, 就可以启用 `GridView` 控件的分页功能。`GridView` 控件默认每一页显示 10 条记录, 并在网格的底部添加分页功能。程序清单 8-20 是一个修改网格以启用其排序和分页功能的例子。

程序清单 8-20 启用 `GridView` 控件的排序和分页功能

```

<asp:GridView ID="GridView1" runat="server"
    DataSourceID="SqlDataSource1" DataKeyNames="CustomerID"
    AutoGenerateColumns="True" AllowSorting="True"
    AllowPaging="True"></asp:GridView>

```

`GridView` 包括各种各样的属性来定制分页功能。例如, `GridView` 的 `PageSize` 属性可以控制页面上显示的记录数; `PagerSettings` 属性可以使用各种分页模式指定网格的分页程序如何显示, 分页模式包括 `NextPrevious`、`NextPreviousFirstLast`、`Numeric`(默认值)和 `NumericFirstLast`。另外, 指定 `GridView` 中的 `PagerStyle` 元素, 可以定制网格显示分页文本的方式, 包括字体颜色、字号、类型以及文本对齐方式和各种其他样式选项。程序清单 8-21 说明了如何定制 `GridView` 控件以使用 `NextPrevious` 模式, 并通过 `PagerStyle` 元素指定分页程序文本的样式。

程序清单 8-21 在 `GridView` 控件中使用 `PagerStyle` 元素和 `PagerSettings` 属性

```

<asp:GridView ID="GridView1" runat="server" DataSourceID="SqlDataSource1"
    DataKeyNames="CustomerID" AutoGenerateColumns="True"
    AllowSorting="True" AllowPaging="True" PageSize="10">
    <PagerStyle HorizontalAlign="Center" />
    <PagerSettings Position="TopAndBottom"
        FirstPageText="Go to the first page"
        LastPageText="Go to the last page"
        Mode="NextPreviousFirstLast">
    </PagerSettings>
</asp:GridView>

```

`PagerSettings` 属性和 `PagerStyle` 元素的选项列表非常长, 因此这里没有列出所有的选项。在 `Visual Studio` 的帮助文档中可以找到完整的选项列表。

另外, `GridView` 控件还提供了 `PageIndexChanging` 和 `PageIndexChanged` 事件, 它们可用于定制网格的标准分页操作, 它们分别在 `GridView` 的当前页面索引改变的前后触发。用户单击网格中的分页链接时, 页面的索引就会改变。`PageIndexChanging` 事件的参数可以在页面索引实际改变之前检查新页面索引的值, 或者取消分页事件。

GridView 控件的 `EnableSortingAndPagingCallbacks` 属性可以指示该控件是否使用客户端回调来执行排序和分页操作。启用这个属性会将 GridView 的排序或分页操作改为使用客户端的 AJAX 回调来检索数据，而不是执行完整的页面回送操作。



如果希望深入学习把 AJAX 集成到 ASP.NET 应用程序中的方式,可参阅第 23 章,该章介绍了 ASP.NET AJAX Framework,说明了如何在应用程序中利用它的功能。

4. 定制 GridView 中的列

网格中要显示的数据常常不是简单的文本数据,而是要使用其他类型的控件显示的数据,或者是根本不需要显示的数据。

如果已经将网格配置成根据绑定的数据源自动生成网格列,那么网格就会为数据源显示的每一个公有属性创建字段。另外,对于所有属性(除了返回 Boolean 类型值的属性外),网格都默认使用标准的 `BoundField` 类型。该类型将其他的所有类型都当作字符串对待。而对于 Boolean 类型的属性,网格会默认使用 `CheckBoxField` 类型来代替。

这些默认行为可能不是应用程序的最优行为,例如为所有的顾客存储 Web 站点的地址,并希望 `CustomerName` 列显示为超链接,从而允许终端用户直接链接到顾客的 Web 站点。GridView 控件包含许多特殊的 Field 类型,从而便于在网格列中显示像超链接、复选框或按钮之类的页面元素。

可以使用两种方式配置 GridView 列:通过 GridView 控件的智能标记中的选项,或者通过在源代码视图中直接编辑列标记。

在 GridView 智能标记中单击 `Edit Columns` 链接,将会打开 Fields 对话框,如图 8-8 所示。在这个对话框中,可以改变已有列的可见性、标题文本、常见的样式选项和列的许多其他属性。

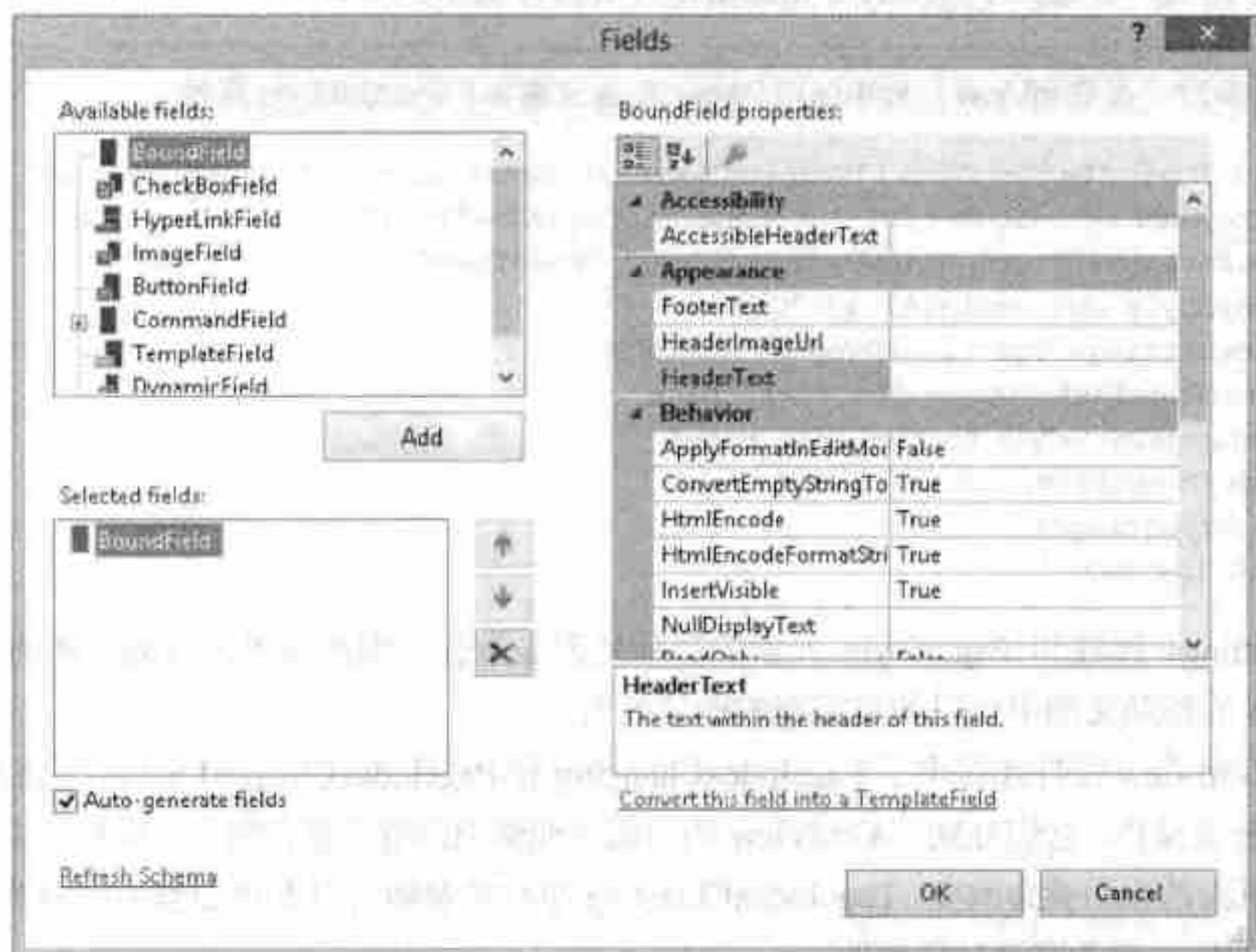


图 8-8

从 GridView 控件的智能标记中选择 Add New Column 链接, 将会显示 Add Field 对话框, 如图 8-9 所示。该对话框中的选项允许给网格添加全新的列。根据从下拉列表中选择列的字段类型, 该对话框会显示该列类型的相应选项。

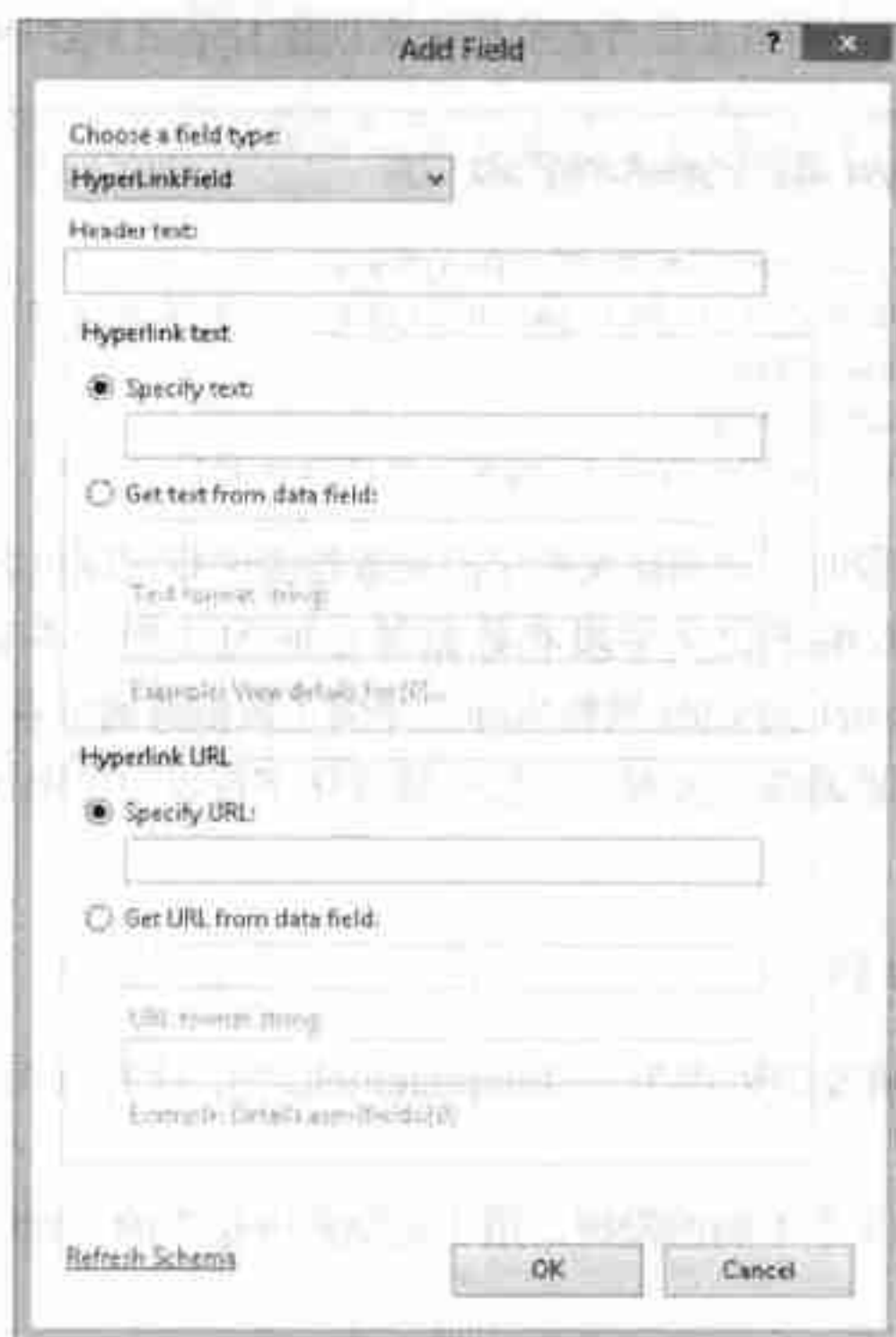


图 8-9

在 Add Field 对话框中可以选择的字段类型如表 8-6 所示。

表 8-6

字段控件	说 明
BoundField	显示数据源中字段的值。这是 GridView 控件的默认列类型
CheckBoxField	为 GridView 控件中的每一项显示一个复选框, 这种列字段类型一般用于显示带布尔值的字段
HyperLinkField	把数据源中字段的值显示为超链接 URL。这种列字段类型可以把第二个字段绑定到超链接文本
ButtonField	为 GridView 控件中的每一项显示一个命令按钮或命令链接。它允许创建一系列定制按钮控件或链接控件, 如 Add 或 Remove 按钮
CommandField	表示一个特殊的字段, 它显示命令按钮或连接, 以在数据绑定控件上执行选择、编辑、插入或删除操作
ImageField	当字段中的数据表示一幅图像时, 或显示所提供的 URL 中的图像时, 自动显示该图像
TemplateField	根据特定的模板, 为 GridView 控件中的每一项显示用户定义的内容。通过这种列字段类型可以创建定制的列字段

前面描述的示例是希望允许终端用户链接到顾客的 Web 站点, 因此要从下拉列表中选择

HyperLinkField 选项。这样，Add Field 对话框就会改变，允许输入超链接信息，包括 URL、数据字段和列的格式化字符串。

也可以在源代码视图中修改网格的列，手动添加或编辑先前表中列出的任意字段类型。程序清单 8-22 说明了如何在源代码视图中添加合适的标记来创建 HyperLinkField 字段。

程序清单 8-22 给 GridView 添加 HyperLinkField 控件

```
<asp:HyperLinkField DataTextField="CompanyName"
    HeaderText="CompanyName" DataNavigateUrlFields="CustomerID,Country"
    SortExpression="CompanyName"
    DataNavigateUrlFormatString=
        http://www.example.com/Customer.aspx?id={0}&country={1} />
```

在源代码视图中添加字段时，必须确保在字段的数据源中指定属性名。每种字段类型都有一个(或一组)不同的属性，可以用来定义字段和数据源之间的连接。在前一个例子中可以看出，HyperLinkField 的 DataNavigateUrlFields 属性实际上提供了数据源属性名的逗号分隔列表，因此可以指定要绑定到该列的多个数据源值。接着，可以在格式化字符串中使用这些字段传递两个查询字符串参数。

5. 使用 TemplateField 列

GridView 控件中一种重要的列类型是 TemplateField 列，这种列可以使用模板完全定制列的内容。

TemplateField 列提供了 6 个不同的模板，用于定制列中指定的不同区域或状态，如编辑状态。表 8-7 列出了这些模板。

表 8-7

模 板 名	说 明
ItemTemplate	用于显示数据绑定控件的 TemplateField 中的一项
AlternatingItemTemplate	用于显示 TemplateField 的替换项
EditItemTemplate	用于显示编辑状态下的 TemplateField 项
InsertItemTemplate	用于显示插入状态下的 TemplateField 项
HeaderTemplate	用于显示 TemplateField 的标题部分
FooterTemplate	用于显示 TemplateField 的脚注部分

要在 GridView 中使用 TemplateField，只需要使用前面介绍的 Add Field 对话框给网格添加列类型即可。<asp: TemplateField>标记充当了列可以包含的各种模板的容器。可以使用 Visual Studio 2012 设计界面的模板编辑功能向模板添加内容，也可以在源代码视图中手动为 TemplateField 元素直接添加定制内容。

ItemTemplate 控制列中每个单元格的默认内容，程序清单 8-23 演示了如何使用 ItemTemplate 定制列的内容。

程序清单 8-23 使用 ItemTemplate

```

<asp:TemplateField HeaderText="CurrentStatus">
  <ItemTemplate>
    <table>
      <tr>
        <td>
          <asp:Button ID="Button2"
            runat="server" Text="Enable" /></td>
        <td>
          <asp:Button ID="Button3"
            runat="server" Text="Disable" /></td>
        </tr>
      </table>
    </ItemTemplate>
  </asp:TemplateField>

```

在这个示例中，ItemTemplate 包含 HTML 表和 ASP.NET Button 控件的组合。

因为 GridView 控件绑定了数据，所以还可以使用数据绑定表达式访问绑定到控件的数据，如 Eval、XPath 或 Bind 表达式。程序清单 8-24 说明了如何使用 Eval 方法添加数据绑定表达式，以设置 Button 控件的文本字段。本章后面将介绍有关数据绑定表达式的更多内容。

程序清单 8-24 添加数据绑定表达式

```

<asp:TemplateField HeaderText="CurrentStatus">
  <ItemTemplate>
    <table>
      <tr>
        <td>
          <asp:Button ID="Button2" runat="server"
            Text='<%# "Enable " + Eval("CustomerID") %>' />
        </td>
        <td>
          <asp:Button ID="Button3" runat="server"
            Text='<%# "Disable " + Eval("CustomerID") %>' />
        </td>
      </tr>
    </table>
  </ItemTemplate>
</asp:TemplateField>

```

TemplateField 中其他常用的模板包括 InsertTemplate 和 EditTemplate。在网格中的一行进入插入或编辑模式时，网格就会使用这些模板。8.4.2 节将使用 InsertItemTemplate 和 EditItemTemplate 在 GridView 控件中插入和编辑数据。

8.3.2 编辑 GridView 中的数据行

用户不但希望在浏览器中查看数据，还希望编辑数据，并把对数据进行的修改保存回数据源。当将 GridView 控件与数据源控件组合到一起时，GridView 控件会简化编辑绑定到网格中的数据。为了说明该控件到底能够使编辑工作变得多么容易，可以修改上一个示例中使用的 SqlDataSource 和 GridView 控件，从而允许用户编辑 Customers 表中的数据。

注意，虽然本章主要关注对绑定到 `SqlDataSource` 控件的 `GridView` 中的数据进行更新，但是当 `GridView` 与其他数据源控件连接时，也可以更新数据。只要 `GridView` 是相同的，不管绑定哪个数据源控件，都允许终端用户将网格行置于编辑模式中，并且插入或删除数据所需的配置是相同的。使数据源控件将对数据的修改持久保存到底层的数据存储中所需的配置，对于每个数据源控件都是不同的。如果使用的是 `SqlDataSource` 之外的其他数据源控件，可以参考本章有关持久保存插入、更新和删除控件数据的详细讨论。

1. 为数据更新配置 `SqlDataSource`

首先修改 `SqlDataSource` 控件，方法是使用 `Configure Data Source` 向导或手动在源代码视图添加标记来添加 `UpdateCommand` 属性。该属性告诉数据源控件，在需要执行更新时应执行什么 SQL 命令。程序清单 8-25 显示了添加 `UpdateCommand` 属性所需的代码。

程序清单 8-25 给 `SqlDataSource` 控件添加 `UpdateCommand` 属性

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
  SelectCommand="SELECT * FROM [Customers]"
  ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
  DataSourceMode="DataSet"
  UpdateCommand="UPDATE [Customers] SET [CompanyName] = @CompanyName,
    [ContactName] = @ContactName, [ContactTitle] = @ContactTitle,
    [Address] = @Address, [City] = @City, [Region] = @Region,
    [PostalCode] = @PostalCode, [Country] = @Country,
    [Phone] = @Phone, [Fax] = @Fax
    WHERE [CustomerID] = @ CustomerID">
</asp:SqlDataSource>
```

注意，`UpdateCommand` 属性包含许多占位符，如 `@CompanyName`、`@Country`、`@Region` 和 `@CustomerID`，它们代表在更新一行时来自于 `GridView` 的相应信息。每一个占位符都分别与 `SqlDataSource` 控件的 `UpdateParameters` 集合中定义的某个 `Parameter` 元素相对应。`UpdateParameters` 集合如程序列表 8-26 所示，其工作方式与本章前面讨论的 `SelectParameters` 元素非常类似。

程序清单 8-26 给 `SqlDataSource` 控件添加 `UpdateParameters` 元素

```
<UpdateParameters>
  <asp:Parameter Type="String" Name="CompanyName"></asp:Parameter>
  <asp:Parameter Type="String" Name="ContactName"></asp:Parameter>
  <asp:Parameter Type="String" Name="ContactTitle"></asp:Parameter>
  <asp:Parameter Type="String" Name="Address"></asp:Parameter>
  <asp:Parameter Type="String" Name="City"></asp:Parameter>
  <asp:Parameter Type="String" Name="Region"></asp:Parameter>
  <asp:Parameter Type="String" Name="PostalCode"></asp:Parameter>
  <asp:Parameter Type="String" Name="Country"></asp:Parameter>
  <asp:Parameter Type="String" Name="Phone"></asp:Parameter>
  <asp:Parameter Type="String" Name="Fax"></asp:Parameter>
  <asp:Parameter Type="String" Name="CustomerID"></asp:Parameter>
</UpdateParameters>
```

每个 `Parameter` 元素都使用两个属性来创建到底层数据源的连接，其中 `Name` 是数据库列名，`Type`

是数据库列的数据类型。在本例中，所有的参数都是 String 类型。

记住，也可以使用本章前面提到的任意参数类型，如 UpdateParameters 元素中的 ControlParameter 和 QueryStringParameter。

2. 为数据更新配置 GridView

在为更新配置了 SqlDataSource 后，需要创建一种方式，以使 GridView 中的行能进入编辑模式，还需要创建一种方式以告诉 GridView 控件，数据库表的主键是什么。

GridView 包含两个内置的属性：AutoGenerateEditButton 和 CommandField，它们可以使行进入编辑模式。

当 AutoGenerateEditButton 属性设置为 True 时，这将告诉网格要为每一行添加一个带有编辑按钮的 CommandField 列。单击其中的一个按钮，就可以将相关的行置于编辑模式。程序清单 8-27 显示了如何在 GridView 控件中添加 AutoGenerateEditButton 属性。

程序清单 8-27 在 GridView 控件中添加 AutoGenerateEditButton 属性

```
<asp:GridView ID="GridView1" runat="server"
    DataSourceID="SqlDataSource1" DataKeyNames="CustomerID"
    AutoGenerateColumns="True" AllowSorting="True" AllowPaging="True"
    AutoGenerateEditButton="true" />
```

GridView 控件还包含 AutoGenerateSelectButton 和 AutoGenerateDeleteButton 属性。使用这两个属性可以轻松地为网格添加行选择和行删除功能。

CommandField 列是一种特殊的字段类型，可以使终端用户对 GridView 中的行执行不同命令。程序清单 8-28 显示了如何配置 CommandField 以使用户将一行置于编辑状态。

程序清单 8-28 使用 CommandField 添加编辑功能

```
<asp:CommandField ShowHeader="True" HeaderText="Command"
    ShowEditButton="True" />
```

可以将命令显示为链接、按钮，甚至图像。

通过浏览器查看 Web 页面时可以发现，新的编辑列已经添加进来。单击 Edit 链接，就可以编辑该数据行的内容。

CommandField 元素还有一些属性，可以准确控制列中显示的内容。可以指示列是否显示 Cancel、Delete、Edit、Insert 或 Select 命令。

为了完成对网格的配置以使其能够编辑，需要确保网格知道数据库表中的哪些列配置为主键。可以使用 GridView 的 DataKeyNames 属性来指定，参见程序清单 8-29。

程序清单 8-29 向 GridView 控件添加 DataKeyNames 属性

```
<asp:GridView ID="GridView1" runat="server"
    DataSourceID="SqlDataSource1" DataKeyNames="CustomerID"
    AutoGenerateColumns="False" AllowSorting="True" AllowPaging="True">
```

如果表的主键不止一列，可以使用逗号隔开的列表来指定多个列。

给不希望用户编辑的列添加 `ReadOnly` 属性, 就可以控制允许用户编辑的网格列。程序清单 8-30 演示了如何给 ID 列添加 `ReadOnly` 属性。

程序清单 8-30 给 `BoundField` 列添加 `ReadOnly` 属性

```
<asp:BoundField DataField="CustomerID" HeaderText="CustomerID"
SortExpression="CustomerID" ReadOnly="True" />
```

再次浏览 Web 页面, 单击 Edit 按钮, 可以看出 ID 列是不能编辑的。

3. 更新数据时处理错误

为了通过 `GridView` 在更新数据时检查出错误, 可以使用 `RowUpdated` 事件。程序清单 8-31 演示了如何在用户试图更新数据后检查错误。

程序清单 8-31 使用 `RowUpdated` 事件检查更新错误

```
<script runat="server">
    protected void GridView1_RowUpdated(object sender,
        GridViewUpdatedEventArgs e)
    {
        if(e.Exception != null)
        {
            this.lblErrorMessage.Text = e.Exception.Message;
        }
    }
</script>
```

`RowUpdated` 事件参数包含 `Exception` 属性。程序清单 8-31 检查这个属性的值是否是 `null`。如果不是, 就说明出现了错误, 并将相应的消息显示给用户。

4. 使用 `TemplateField` 的 `EditItemTemplate` 模板

本章前面介绍了 `TemplateField` 及其包含的一些模板, 其中一个模板是 `EditItemTemplate`, 网格使用它为进入编辑模式的行显示 `TemplateField` 列。使用 `EditItemTemplate` 可以完全定制用户的数据编辑方式。例如, 对 `Region` 列来说, 一种较好的编辑方式是将所有可能的值显示为 `DropDownList`, 而不是简单的文本框(`BoundField` 默认的编辑方式)。

为此, 只需把 `Region` 列从 `BoundField` 改为 `TemplateField`, 添加 `ItemTemplate` 和 `EditItemTemplate` 模板。在 `EditItemTemplate` 模板中, 可以添加一个 `DropDownList` 控件, 提供相应的数据绑定信息, 使该控件绑定到唯一的 `Region` 列表。程序清单 8-32 说明了如何给 `GridView` 添加 `ItemTemplate` 和 `EditItemTemplate` 模板。

程序清单 8-32 给 `GridView` 添加 `ItemTemplate` 和 `EditItemTemplate` 模板

```
<asp:TemplateField HeaderText="Country">
    <ItemTemplate><%# Eval("Country") %></ItemTemplate>
    <EditItemTemplate>
        <asp:DropDownList ID="DropDownList1" runat="server"
            DataSourceID="SqlDataSource2"
```

```

        DataTextField="Country" DataValueField="Country">
</asp:DropDownList>
<asp:SqlDataSource ID="SqlDataSource2" runat="server"
    ConnectionString=
        "<%$ ConnectionStrings:ConnectionString %>"
    SelectCommand="SELECT DISTINCT [Country] FROM [Customers]">
</asp:SqlDataSource>
</EditItemTemplate>
</asp:TemplateField>

```

在 `ItemTemplate` 模板中使用了一个简单的 `Eval` 数据绑定表达式, 在数据行的默认显示模式下显示该列的值。在 `EditItemTemplate` 模板中, 包含了一个绑定到 `SqlDataSource` 控件的 `DropDownList` 控件。

要从 `DropDownList` 控件中显示当前选择的 `Country` 值, 可使用 `RowDataBound` 事件。程序清单 8-33 演示了该过程。

程序清单 8-33 使用 `RowDataBound` 事件选择一个 `DropDownList` 项

```

<script runat="server">
    protected void GridView1_RowDataBound(object sender,
        GridViewRowEventArgs e)
    {
        // Check for a row in edit mode.
        if( (e.Row.RowState == DataControlRowState.Edit) ||
            (e.Row.RowState == (DataControlRowState.Alternate |
                DataControlRowState.Edit)) )
        {
            System.Data.DataRowView drv =
                (System.Data.DataRowView)e.Row.DataItem;
            DropDownList ddl =
                (DropDownList)e.Row.Cells[8].FindControl("DropDownList1");
            ListItem li =
                ddl.Items.FindByValue(drv["Country"].ToString());
            li.Selected = true;
        }
    }
</script>

```

如上面的程序清单所示, 要设置相应的 `DropDownList` 值, 首先使用 `RowState` 属性检查当前绑定的 `GridViewRow` 是否处于编辑状态。`RowState` 属性是 `DataControlRowState` 值的按位组合。表 8-8 列出了 `GridViewRow` 的所有可能状态。

表 8-8

行 状 态	说 明
替换	表示该行是替换行
编辑	表示该行当前处于编辑模式
插入	表示该行是新行, 当前处于插入模式
正常	表示该行当前处于正常状态
已选中	表示该行是 <code>GridView</code> 中的当前选中行

为了正确确定当前的行状态，必须与 RowState 属性进行多次比较。RowState 可以瞬间处于多种状态，例如替换和编辑状态，因此需要使用按位比较来正确确定 GridViewRow 是否处于编辑状态。

确定了行处于编辑状态之后，使用 FindControl 方法定位相应 GridViewRow 单元格中的 DropDownList 控件。这个方法可以按名称定位服务器控件。找到 DropDownList 控件后，定位 DropDownList 中相应的 ListItem，把它的 Selected 属性设置为 True。

用户更新行后，还需要使用 GridView 事件把 DropDownList 控件的值添加回 GridView。为此，可以使用 RowUpdating 事件，如程序清单 8-34 所示。

程序清单 8-34 使用 RowUpdating 事件

```
<script runat="server">
    protected void GridView1_RowUpdating(object sender,
        GridViewUpdateEventArgs e)
    {
        GridViewRow gvr =
            this.GridView1.Rows[this.GridView1.EditIndex];
        DropDownList ddl =
            (DropDownList)gvr.Cells[8].FindControl("DropDownList1");
        e.NewValues["Country"] = ddl.SelectedValue;
    }
</script>
```

在这个事件中，使用 EditIndex 属性确定当前正在编辑的 GridViewRow。这个属性包含当前处于编辑状态的 GridViewRow 的索引。找到这一行后，使用 FindControl 方法定位相应行单元格中的 DropDownList 控件，如程序清单 8-34 所示。找到 DropDownList 控件后，把该控件的 SelectedValue 属性添加到 GridView 控件的 NewValues 集合中。

8.3.3 删除 GridView 数据

从 GridView 生成的表中删除数据比编辑数据更容易。只要在 SqlDataSource 控件和 GridView 控件中添加一些代码，就可以删除表中的一整行数据。

与前面添加的 Edit 按钮非常类似，只要把 AutoGenerateDeleteButton 属性设置为 True 或者使用 CommandField，就可以给网格添加 Delete 按钮。使用 AutoGenerateDeleteButton 属性添加 Delete 按钮的过程如程序清单 8-35 所示。

程序清单 8-35 给 GridView 添加 Delete 按钮

```
<asp:GridView ID="GridView2" runat="server"
    DataSourceID="SqlDataSource1" DataKeyNames="CustomerID"
    AutoGenerateColumns="True" AllowSorting="True" AllowPaging="True"
    AutoGenerateEditButton="true" AutoGenerateDeleteButton="true"/>
```

修改 SqlDataSource 控件也很简单，可以通过使用 Configure Data Source 向导或者在标记中手动操作来完成该操作。程序清单 8-36 显示了如何向 SqlDataSource 控件添加 DeleteCommand 属性。

程序清单 8-36 给 SqlDataSource 控件添加删除功能

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
```



```

SelectCommand="SELECT * FROM [Customers]"
ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
DataSourceMode="DataSet"
DeleteCommand="DELETE From Customers
    WHERE (CustomerID = @CustomerID)"
UpdateCommand="UPDATE [Customers]
    SET [CompanyName] = @CompanyName,
    [ContactName] = @ContactName, [ContactTitle] = @ContactTitle,
    [Address] = @Address, [City] = @City, [Region] = @Region,
    [PostalCode] = @PostalCode, [Country] = @Country,
    [Phone] = @Phone, [Fax] = @Fax
    WHERE [CustomerID] = @original_CustomerID">
<!-- Update parameters removed for clarity -->
</asp:SqlDataSource>

```

与 UpdateCommand 属性一样, DeleteCommand 属性也使用指定的参数来决定应该要删除的行。因此,在 SqlDataSource 控件中定义这个属性。为此,在 SqlDataSource 控件中添加<DeleteParameters>部分,如程序清单 8-37 所示。

程序清单 8-37 在 SqlDataSource 控件中添加<DeleteParameters>部分

```

<DeleteParameters>
  <asp:Parameter Name="CustomerID" Type="String">
  </asp:Parameter>
</DeleteParameters>

```

这是<DeleteParameters>集合需要的唯一参数,因为这个用于删除的 SQL 命令只需要行中的 CustomerID,就可以删除整行。在运行包含这些代码的示例时,网格中会显示 Delete 链接,单击 Delete 链接会完全删除选中的行。

与更新数据时一样,在执行删除操作时检查数据库的错误是个好方法。程序清单 8-38 演示了如何使用 GridView 的 RowDeleted 事件和 SqlDataSource 的 Deleted 事件检查在删除过程中可能发生的错误。

程序清单 8-38 使用 RowDeleted 事件捕获 SQL 错误

```

<script runat="server">
  protected void GridView1_RowDeleted(object sender,
    GridViewDeletedEventArgs e)
  {
    if(e.Exception != null)
    {
      this.lblErrorMessage.Text = e.Exception.Message;
      e.ExceptionHandled = true;
    }
  }

  protected void SqlDataSource1_Deleted(object sender,
    SqlDataSourceStatusEventArgs e)
  {
    if(e.Exception != null)
    {
      this.lblErrorMessage.Text = e.Exception.Message;
      e.ExceptionHandled = true;
    }
  }

```

```

    }
}
</script>

```

注意，这两个事件都把 `Exception` 属性提供为事件参数的一部分。如果该属性不为空，就表示发生了可以处理的异常。如果选择处理该异常，就应该把 `ExceptionHandled` 属性设置为 `true`；否则，异常会显示给终端用户。

8.3.4 DetailsView 控件

`DetailsView` 控件是数据绑定控件，可以一次查看一条数据记录。`GridView` 控件适合于查看一组数据，但在许多情况下，我们可能只需要显示一条记录而不是整个数据集，此时 `DetailsView` 控件就可以发挥作用。它提供了与 `GridView` 相同的许多数据操作和显示功能，可以对数据进行分页、更新、插入和删除。

要开始使用 `DetailsView`，首先把该控件拖放到设计界面上。与 `GridView` 一样，可以使用 `DetailsView` 的智能标记创建和设置该控件的数据源。在本节的示例中，只需使用前面介绍的用于 `GridView` 的 `SqlDataSource` 控件即可。将 `SqlDataSource` 设置为 `DetailsView` 控件的数据源并运行该页面。程序清单 8-39 显示了将 `DetailsView` 控件绑定到 `SqlDataSource` 控件的标记。

程序清单 8-39 绑定到 `SqlDataSource` 控件的 `DetailsView` 控件

```

<asp:DetailsView ID="DetailsView1" runat="server"
    DataSourceID="SqlDataSource1" DataKeyNames="CustomerID"
    AutoGenerateRows="True"></asp:DetailsView>
    <asp:SqlDataSource ID="SqlDataSource1" runat="server"
        SelectCommand="SELECT * FROM [Customers]"
        ConnectionString=
            "<%$ ConnectionStrings:ConnectionString %>"
        DataSourceMode="DataSet">
    </asp:SqlDataSource>

```

如果只是想显示一条记录，就要改变 `SqlDataSource` 控件的 `SelectCommand`，使其只返回一个 `Customer` 对象，而不是像查询一样返回所有的 `Customer` 对象。然而，如果从数据库中返回多个对象，就可以把 `DetailsView` 的 `AllowPaging` 属性设置为 `True`，启用分页功能，如程序清单 8-40 所示。

程序清单 8-40 在 `DetailsView` 控件上启用分页功能

```

<asp:DetailsView ID="DetailsView1" runat="server"
    DataSourceID="SqlDataSource1" DataKeyNames="CustomerID"
    AutoGenerateRows="True" AllowPaging="true"></asp:DetailsView>

```

可以在 `DetailsView` 的智能标记中选中 `Enable Paging` 复选框，也可以在源代码视图中给控件添加 `AllowPaging` 属性。与 `GridView` 一样，`DetailsView` 控件可以使用 `PagerSettings` 和 `Pager` 样式定制控件的分页程序。

1. 定制 `DetailsView` 的显示

通过选择 `DetailsView` 控件显示的字段，可以定制该控件的外观。该控件默认显示其绑定数据源

的每一个公有属性。但是,使用与 GridView 控件相同的基本语法,DetailsView 控件也允许指定要显示的属性,如程序清单 8-41 所示。

程序清单 8-41 定制 DetailsView 控件的显示

```
<asp:DetailsView ID="DetailsView1" runat="server"
    DataSourceID="SqlDataSource1" DataKeyNames="CustomerID"
    AutoGenerateRows="False">
    <Fields>
        <asp:BoundField ReadOnly="True" HeaderText="CustomerID"
            DataField="CustomerID" SortExpression="CustomerID"
            Visible="False" />
        <asp:BoundField ReadOnly="True" HeaderText="CompanyName"
            DataField="CompanyName" SortExpression="CompanyName" />
        <asp:BoundField HeaderText="ContactName"
            DataField="ContactName" SortExpression="ContactName" />
        <asp:BoundField HeaderText="ContactTitle"
            DataField="ContactTitle"
            SortExpression="ContactTitle" />
    </Fields>
</asp:DetailsView>
```

在这个例子中,使用 BoundField 对象,在 DetailsView 控件的 Fields 集合中只定义了 Customers 表的 4 个字段。

2. 结合使用 DetailsView 和 GridView

下面将介绍使用 DetailsView 和 GridView 常见的显示情形。在这个例子中,使用 GridView 显示数据的主视图,使用 DetailsView 显示 GridView 中选中行的细节。Customers 表是数据源。程序清单 8-42 列出了所需的代码。

程序清单 8-42 结合使用 DetailsView 和 GridView

```
<html>
<head id="Head1" runat="server">
    <title>GridView & DetailsView Controls</title>
</head>
<body>
    <form id="form1" runat="server">
        <p>
            <asp:GridView ID="GridView1" runat="server"
                DataSourceID="SqlDataSource1"
                DataKeyNames="CustomerID"
                AutoGenerateSelectButton="True" AllowPaging="True"
                AutoGenerateColumns="True" PageSize="5">
                <SelectedRowStyle ForeColor="White" BackColor="#738A9C"
                    Font-Bold="True" />
            </asp:GridView>
        </p>
        <p><b>Customer Details:</b></p>
        <asp:DetailsView ID="DetailsView1" runat="server"
            DataSourceID="SqlDataSource2">
```



```

        AutoGenerateRows="True" DataKeyNames="CustomerID">
    </asp:DetailsView>
    <asp:SqlDataSource ID="SqlDataSource1" runat="server"
        SelectCommand="SELECT * FROM [Customers]"
        ConnectionString=
            "<%%$ ConnectionStrings:ConnectionString %>" />
    <asp:SqlDataSource ID="SqlDataSource2" runat="server"
        SelectCommand="SELECT * FROM [Customers]"
        FilterExpression="CustomerID='{0}'"
        ConnectionString=
            "<%%$ ConnectionStrings:ConnectionString %>">
    <FilterParameters>
        <asp:ControlParameter Name="CustomerID"
            ControlId="GridView1"
            PropertyName="SelectedValue" />
    </FilterParameters>
    </asp:SqlDataSource>
</form>
</body>
</html>

```

为了说明其工作原理, 首先查看对第二个 `SqlDataSource` 控件(`SqlDataSource2`)的修改: 添加了 `FilterExpression` 属性, 用于过滤由 `SelectCommand` 属性检索的数据。在这个例子中, `FilterExpression` 的值设置为 `CustomerID='{0}'`, 表示 `SqlDataSource` 控件按照传送给它的 `CustomerID` 值来过滤返回的数据。

`FilterExpression` 属性中指定的参数 `CustomerID` 在 `SqlDataSource` 控件的 `<FilterParameters>` 集合中定义。在这个例子中, 使用 `<asp:ControlParameter>` 指定 `GridView` 控件的 `SelectedValue` 属性, 从而填充参数的值。

3. SelectParameters 与 FilterParameters

在上面的例子中, `FilterParameters` 似乎提供了与 `SelectParameters`(8.1.1 节讨论了该集合)相同的功能。虽然有相同的效果, 但它们使用完全不同的方法。如前所述, 使用 `SelectParameters` 可以把值插入 `SelectCommand` 指定的 `WHERE` 子句中, 这会限制从 `SQL Server` 返回的数据行, 这些数据行由数据源控件保存到内存中。其优点是限制从 `SQL` 返回的数据量, 可以使应用程序运行得更快, 减少消耗的内存量。其缺点是只能操作 `SQL` 查询返回的有限的数据子集。

另一方面, `FilterParameters` 不使用 `WHERE` 子句, 而是要求从服务器返回所有的数据, 然后将过滤器应用于数据源控件的内存数据。这种过滤方法的缺点是需要从数据存储返回更多的数据。但是, 有时这这也是一个优点, 如果要对一大块数据进行许多过滤操作(例如, 启用 `DetailView` 的分页功能), 在每次需要下一条记录时, 就不需要从数据存储中再次调用数据存储。所有的数据都由数据源控件存储在缓存中。

8.3.5 使用 DetailsView 插入、更新和删除数据

使用 `DetailsView` 插入数据类似于通过 `GridView` 控件插入数据。要使用 `DetailsView` 插入数据, 只需在 `DetailsView` 控件中添加 `AutoGenerateInsertButton` 属性, 如程序清单 8-43 所示。

程序清单 8-43 在 DetailsView 控件中添加 AutoGenerateInsertButton 属性

```
<asp:DetailsView ID="DetailsView1" runat="server"
    DataSourceID="SqlDataSource1" DataKeyNames="CustomerID"
    AutoGenerateRows="True" AutoGenerateInsertButton="True" />
```

接着, 给 SqlDataSource 控件添加 InsertCommand 和相应的 InsertParameters 元素, 如程序清单 8-44 所示。

程序清单 8-44 给 SqlDataSource 控件添加 InsertCommand 属性

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    SelectCommand="SELECT * FROM [Customers]"
    InsertCommand="INSERT INTO [Customers] ([CustomerID],
        [CompanyName], [ContactName], [ContactTitle], [Address],
        [City], [Region], [PostalCode], [Country], [Phone], [Fax])
        VALUES (@CustomerID, @CompanyName, @ContactName, @ContactTitle,
        @Address, @City, @Region, @PostalCode, @Country, @Phone, @Fax)"
    DeleteCommand="DELETE FROM [Customers]
        WHERE [CustomerID] = @original_CustomerID"
    ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
    <InsertParameters>
        <asp:Parameter Type="String" Name="CustomerID"></asp:Parameter>
        <asp:Parameter Type="String" Name="CompanyName"></asp:Parameter>
        <asp:Parameter Type="String" Name="ContactName"></asp:Parameter>
        <asp:Parameter Type="String" Name="ContactTitle"></asp:Parameter>
        <asp:Parameter Type="String" Name="Address"></asp:Parameter>
        <asp:Parameter Type="String" Name="City"></asp:Parameter>
        <asp:Parameter Type="String" Name="Region"></asp:Parameter>
        <asp:Parameter Type="String" Name="PostalCode"></asp:Parameter>
        <asp:Parameter Type="String" Name="Country"></asp:Parameter>
        <asp:Parameter Type="String" Name="Phone"></asp:Parameter>
        <asp:Parameter Type="String" Name="Fax"></asp:Parameter>
    </InsertParameters>
</asp:SqlDataSource>
```

使用 DetailsView 控件更新和删除数据类似于从 GridView 中更新和删除数据, 只需在 DetailView 控件中指定 UpdateCommand 或 DeleteCommand 属性, 然后提供相应的 UpdateParameters 和 DeleteParameters 元素。

8.3.6 ListView 控件

ASP.NET 还包含另外一个列表样式的控件——ListView, 该控件填补了高度结构化的 GridView 控件和未结构化的 DataList 和 Repeater 控件之间的空白。

过去, 许多需要网格样式的数据控件的开发人员都选择使用 GridView, 这是因为 GridView 便于使用, 而且提供了诸如数据编辑、分页和排序等强大功能。但遗憾的是, 开发人员越钻研这个控件, 就越发现难以控制它显示 HTML 输出的方式。如果要减少控件生成的标记量, 或者要单独使用 CSS 控制控件的布局 and 样式, 这就很成问题。

另一方面, 许多开发人员选择使用 DataList 或 Repeater 控件, 因为它们可以通过增强的控件控制显示。这两个控件不包含布局功能, 允许开发人员完全自由地布置数据。但是, 这两个控件缺乏

GridView 的一些基本功能，例如分页和排序，并且 Repeater 控件缺乏数据编辑功能。

ListView 试图填补这些控件之间的空白。这个控件本身不在运行期间生成 HTML 标记，而是依赖 11 个不同的控件模板，这些模板表示控件的不同区域和这些区域的可能状态。在这些模板中，可以在设计期间放置控件自动生成的标记或开发人员创建的标记，但在这两种情况下，开发人员仍然可以全面控制控件中各个数据项的标记以及整个控件的布局标记。另外，由于该控件能理解并处理数据的编辑和分页，因此可以让控件完成许多数据管理工作，开发人员可以把主要精力集中在数据的显示上。

1. 开始使用 ListView 控件

要开始使用 ListView，只需把该控件拖放到设计界面上，给它指定数据源，这与处理其他数据绑定的列表控件相同。但在指定了数据源后，并没有我们期望的设计期间的布局预览。这是因为 ListView 默认没有定义布局，完全由开发人员定义该控件的布局。事实上，该控件在设计期间的显示甚至说明，至少需要定义 ItemTemplate 和 LayoutTemplate 才能使用该控件。LayoutTemplate 用作控件的根模板，ItemTemplate 用作控件中各个数据项的模板。

定义 ListView 需要的模板时有两个选项。可以修改 ListView 智能标记中的 Current View 选项，直接编辑模板，也可以从该控件的智能标记中选择预定义的布局。改变 Current View 可以查看每个可用模板在运行期间的视图，直接编辑这些模板的内容，如同通常编辑其他控件的模板一样。图 8-10 显示了 ListView 智能标记中的 Current View 下拉列表。

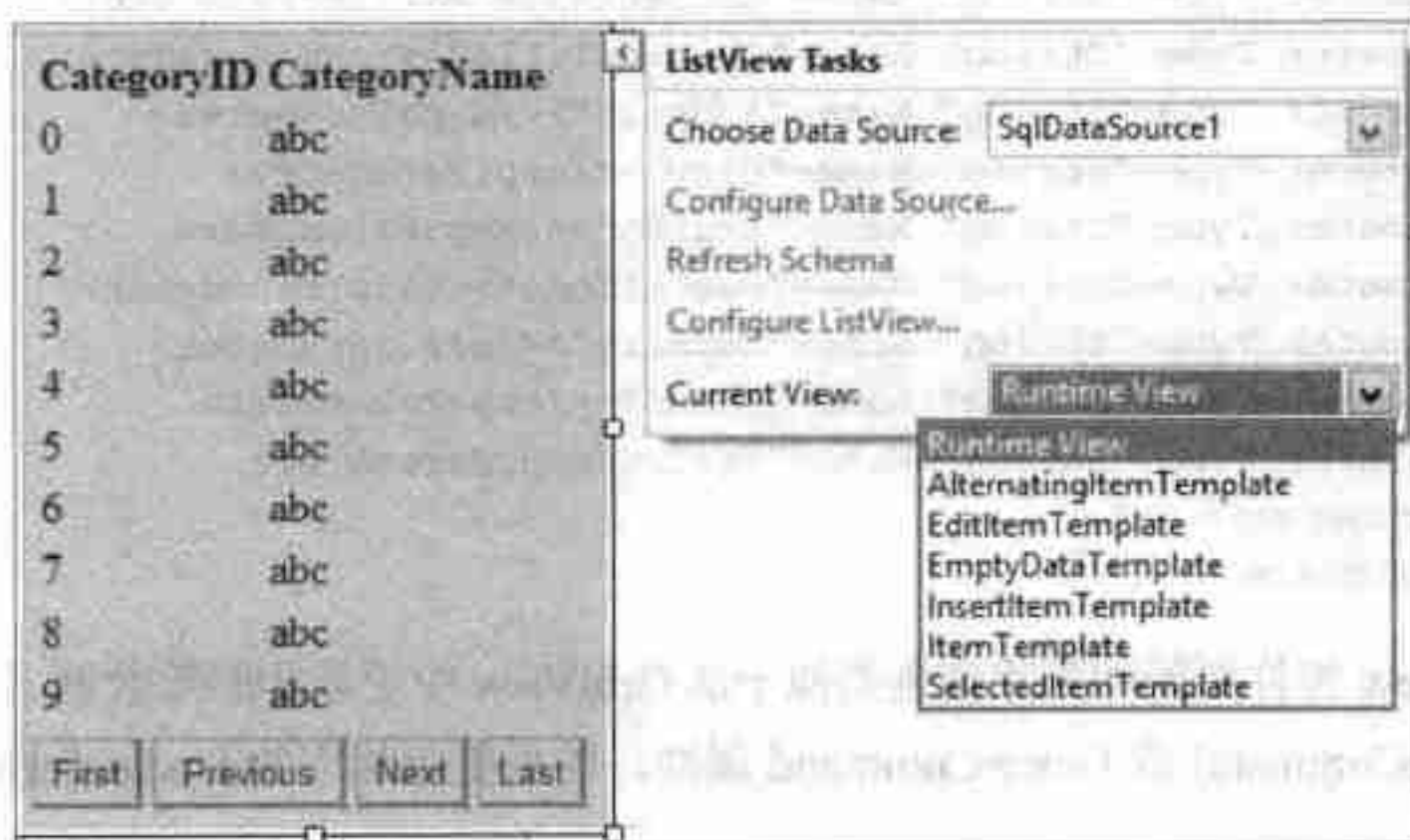


图 8-10

第二个选项更简单，即从 Configure ListView 对话框中选择预定义的布局模板。要打开这个对话框，只需单击智能标记中的 ConfigureListView 选项。在该对话框中，可以在几个不同的预定义布局之间进行选择，可以选择不同的样式选项，甚至可以配置基本的操作，如编辑和分页。

这个控件包含 5 个不同的布局类型：Grid、Tiled、Bulleted List、Flow 和 Single Row。它还包含 4 个不同的样式选项。每种类型的预览都显示在对话框中，改变当前选择的布局 and 样式，预览就会更新。

ListView 模板

给 ListView 应用布局模板后，如果查看 Visual Studio 的源代码视图，就会发现为了提供布局，

该控件生成了大量的标记。这些标记是根据从 Configure ListView 对话框中选择的布局生成的。

仔细观察为 Grid 布局生成的标记，会发现默认情况下该控件创建了 7 个控件模板的标记：ItemTemplate、AlternatingItemTemplate、SelectedItemTemplate、InsertItemTemplate、EditItemTemplate、EmptyDataTemplate 和 LayoutTemplate。这些仅是该控件所带的 11 个不同模板中的 7 个，也可以使用它们为控件的不同状态提供标记。选择不同的预定义布局选项，会使控件生成不同的模板集合。当然，也可以手动添加或删除任意模板。表 8-9 列出了所有 11 个模板。

表 8-9

模板名称	说 明
ItemTemplate	为控件中的每个数据项提供用户界面
AlternatingItemTemplate	为控件中的隔行数据项提供唯一的 UI
SelectedItemTemplate	为当前选中的数据项提供唯一的 UI
InsertItemTemplate	为在控件中插入新数据项提供 UI
EditItemTemplate	为在控件中编辑已有的数据项提供 UI
EmptyItemTemplate	为在当前页面的最后一组中没有要显示的数据时创建的行提供唯一的 UI
EmptyDataTemplate	当绑定的数据对象不包含数据项时显示的模板
LayoutTemplate	该模板用作 ListView 控件的根容器，用于控制数据项的整体布局
GroupSeparatorTemplate	用于提供组之间的分隔符 UI
GroupTemplate	用于为分组的内容提供唯一的 UI
ItemSeparatorTemplate	用于在每个数据项之间提供分隔符 UI

使用模板允许 ListView 控件保留组成 ListView 的标记部分和状态的基本信息，并且仍然可以几乎全面控制 ListView 的 UI。

ListView 数据项的显示

总体来说，ListView 非常灵活，几乎可以完全控制其绑定数据的显示方式，但它的一些基本结构定义了刚才描述的模板之间的关系。如前所述，该控件至少需要定义 LayoutTemplate 和 ItemTemplate 这两个模板。LayoutTemplate 是根控件模板，因此应在该模板中定义 ListView 的数据项集合的整体布局。

例如，如果查看由 Grid 布局生成的模板标记，就会看到 LayoutTemplate 包含<table>元素定义、表行<tr>定义和为每个列标题定义的<td>元素。

另一方面，应在 ItemTemplate 中定义各个数据项的布局。如果再次查看为 Grid 布局生成的模板标记，就会看到 ItemTemplate 是表行<tr>元素，后跟一系列包含实际数据的表单元格<td>元素。

在显示 ListView 时，ItemTemplate 应显示在 LayoutTemplate 中，但需要一种机制告诉控件，ItemTemplate 应放置在 LayoutTemplate 的什么地方。ListView 控件通过在 LayoutTemplate 中查找数据项容器来完成此项工作。数据项容器是 HTML 容器元素，该元素设置了 runat="server" 属性，并且包含值为 itemContainer 的 id 属性。该元素可以是任意有效的 HTML 容器元素，但如果查看默认的 LayoutTemplate，会看到它使用<tbody>元素。

```
<tbody id="itemContainer">
```

```
</tbody>
```

为了增加控件的整体灵活性,甚至 ListView 查找的数据项容器元素 id 也是可以配置的。该控件默认尝试定位 id 属性设置为 itemContainer 的元素,但可以通过改变控件的 ItemContainerID 属性来修改控件查找的 id 值。

如果控件没有找到适合用作数据项容器的 HTML 元素,就会抛出异常。

ListView 使用标识为 itemContainer 的元素定位 ItemTemplate 和数据项级别的任意模板,例如 AlternatingItemTemplate、EditItemTemplate、EmptyItemTemplate、InsertItemTemplate、ItemSeperatorTemplate 和 SelectedItemTemplate。在显示过程中,控件仅根据所绑定的数据项的状态(选择、编辑或替换),把相应的项模板放在数据项容器中。

ListView 组的显示

除了数据项容器之外,ListView 还支持另一种容器类型:分组容器。结合使用分组容器与 GroupTemplate,就可以把较大的数据项组分成较小的组。每一组中的项数由控件的 GroupItemCount 属性设置。在显示一定数量的项模板后,如果要输出一些额外的 HTML,就可以使用 GroupTemplate。使用 GroupTemplate 时,存在与前面讨论的相同问题。但是,在这里不是把两个模板关联起来,引入 GroupTemplate 就意味着要关联 3 个模板:ItemTemplate 关联到 GroupTemplate,而 GroupTemplate 关联到 LayoutTemplate。

ListView 在显示时会查看是否已经定义了 GroupTemplate。如果控件找到了 GroupTemplate,就会查看 LayoutTemplate 中是否提供了分组容器。如果已经定义了 GroupTemplate,控件就要求定义分组容器,否则就抛出异常。分组容器的工作方式与前面描述的数据项容器相同,但分组容器元素的 id 值应是 groupContainer,而不是 itemContainer。与数据项容器一样,可以通过改变控件的 GroupContainerID 属性来修改控件查找的特定 id 值。

查看 ListView 的 Tiled 布局生成的标记时,可以看到一个使用分组容器的例子。这个布局的 LayoutTemplate 显示了一个用作分组容器的表,如下所示:

```
<table id="groupContainer" runat="server" border="0" style="">
</table>
```

定义了分组容器后,还需要定义数据项容器,但不是在 LayoutTemplate 中定义,而需要在 GroupTemplate 中定义。再次查看 Tiled 布局,会发现在其 GroupTemplate 中定义了一个用作数据项容器的表行。

```
<tr id="itemContainer" runat="server">
</tr>
```

在显示时,ListView 首先输出其 LayoutTemplate,再输出 GroupTemplate。之后 ItemTemplate 输出 GroupItemCount 属性定义的次数。达到组的项数后,ListView 就输出 GroupTemplate,再输出 ItemTemplate,并且为它绑定的每个数据项都重复这个过程。

使用 EmptyItemTemplate

使用 GroupTemplate 时,还要注意绑定到 ListView 控件的数据项数可能不能被 GroupItemCount 值整除。如果创建了一个 ListView 布局,该布局又依赖 HTML 表排列其数据项,就尤其要注意这个问题,因为最后一行定义的表单元格数有可能少于前面的表行,从而使控件输出的 HTML 无效,

并可能导致显示问题。为了解决这个问题, ListView 控件包含了 EmptyItemTemplate。如果使用 GroupTemplate, 并且没有足够的数据项达到 GroupItemCount 值, 就显示这个模板。

例如, 如果绑定到 ListView 控件的数据源包含 4 个数据项, 但控件的 GroupItemCount 设置为 3, 表示在每个组中显示 3 个 ItemTemplate。在显示的第二个组中只剩下一个数据项要显示, 因此该控件使用 EmptyItemTemplate(假定定义了该模板)填充剩余的项。

2. ListView 数据绑定和命令

因为 ListView 不在运行期间生成任何布局标记, 也不像 GridView 那样包含任何自动生成字段的逻辑, 所以每个模板都使用标准的 ASP.NET 内联数据绑定语法来定位每个数据项在已定义布局中的值。有关 ASP.NET 的内联数据绑定语法, 详见本章后面的内容。

查看控件创建的默认 Grid 布局的 ItemTemplate, 就可以看到一个内联绑定的例子。在这个模板中, 使用 ASP.NET 标签显示绑定数据源的每一列, 该标签的文本属性设置为一个数据绑定计算表达式:

```
<asp:Label ID="ProductNameLabel" runat="server"
    Text='<%# Eval("ProductName") %>' />
```

因为该控件使用这个灵活的模型显示绑定的数据, 所以可以利用这一点把数据放在模板的任意位置上, 甚至可以使用 ASP.NET 的数据绑定功能在显示绑定的数据之前处理它们。

每个显示绑定数据的 ListView 模板都使用相同的 ASP.NET 绑定语法, 只是为数据提供一个不同的模板。例如, 如果可以在 Grid 布局中编辑, 就会发现 EditItemTemplate 根据底层的数据类型, 使用文本框或复选框替代了 ItemTemplate 使用的 ASP.NET 标签。

```
<asp:TextBox ID="ProductNameTextBox" runat="server"
    Text='<%# Bind("ProductName") %>' />
```

另外, 这种灵活性还允许精确地选择让终端用户如何编辑数据(假定数据可以编辑)。这里没有使用标准的 ASP.NET 文本框, 而是可以轻易地使用下拉列表甚至第三方编辑控件替代它。

要让 ListView 为数据项显示 EditItemTemplate, 该控件需要使用与 GridView 控件相同的命令概念。ItemTemplate 提供了 3 个命令, 可用于改变数据项的状态, 如表 8-10 所示。

表 8-10

命令名	说明
Edit	使指定的数据项进入编辑模式并为该数据项显示 EditTemplate
Delete	从底层数据源中删除指定的数据项
Select	把 ListView 控件的 Selected 索引设置为指定数据项的索引

这些命令与 ASP.NET Button 控件的 CommandName 属性结合使用。使用 ListView 配置对话框激活 Editing 和 Deleting 功能, 就可以看到在 ListView 默认 Grid 布局的 ItemTemplate 中使用的这些命令。这样做会生成一个包含 Edit 和 Delete 按钮的新列, 这两个按钮把 CommandName 属性分别设置为 Edit 和 Delete。

```
<asp:Button ID="DeleteButton" runat="server"
```



```
CommandName="Delete" Text="Delete" />
<asp:Button ID="EditButton" runat="server"
CommandName="Edit" Text="Edit" />
```

ListView 的其他模板提供了其他命令，如表 8-11 所示。

表 8-11

模 板	命 令 名	说 明
EditItemTemplate	Update	更新 ListView 数据源中的数据，并把数据项返回给 ItemTemplate 进行显示
EditItemTemplate	Cancel	取消编辑，并把数据项返回给 ItemTemplate
InsertItemTemplate	Insert	把数据插入 ListView 数据源
InsertItemTemplate	Cancel	取消插入，重置 InsertTemplate 控件绑定值

3. ListView 分页和 Pager 控件

ASP.NET 包含另一个控件 DataPager，ListView 使用它提供分页功能。DataPager 控件用于向终端用户显示分页的导航功能，并与实现了 IPagableItemContainer 接口的数据绑定控件(在 ASP.NET 中就是 ListView 控件)一起完成数据分页任务。事实上，如果在 ListView 配置对话框中选中 Paging 复选框，激活 ListView 控件的分页功能，该控件就会在其 LayoutTemplate 中插入一个新的 DataPager 控件。ListView 为 Grid 布局生成的默认分页标记如下：

```
<asp:DataPager ID="DataPager1" runat="server">
  <Fields>
    <asp:NextPreviousPagerField ButtonType="Button" FirstPageText="First"
      LastPageText="Last" NextPageText="Next" PreviousPageText="Previous"
      ShowFirstPageButton="True" ShowLastPageButton="True" />
  </Fields>
</asp:DataPager>
```

该控件的标记显示，在 DataPager 中创建了一个 Fields 集合，它包含一个 NextPreviousPagerField 对象。顾名思义，使用 NextPreviousPagerField 对象会使 DataPager 显示 Next 和 Previous 按钮作为其用户界面。DataPager 控件包含 3 类 Field 对象：NextPreviousPagerField、NumericPagerField 和 TemplatePagerField 对象。NumericPagerField 对象会生成一个简单的页面编号列表，而 TemplatePagerField 可以指定定制的分页用户界面。这些不同的 Field 类型都包含许多属性，可用于精确控制 DataPager 显示用户界面的方式。另外，DataPager 包含一个 Fields 集合，而不是一个简单的 Field 属性，因此可以在一个 DataPager 控件中显示几种不同的 Field 对象。

TemplatePagerField 是一种独特的 Field 对象，它不包含用户界面本身，而只包含一个模板，可使用该模板完全定制分页程序的用户界面。程序清单 8-45 演示了 TemplatePagerField 的使用。

程序清单 8-45 创建定制的 DataPager 用户界面

```
<asp:DataPager ID="DataPager1" runat="server">
  <Fields>
    <asp:TemplatePagerField>
      <PagerTemplate>
        Page
```

```

        <asp:Label ID="Label1" runat="server"
            Text=
                "<%# (Container.StartRowIndex/Container.PageSize)+1%" />
        <asp:Label ID="Label2" runat="server"
            Text=
                "<%# Container.TotalRowCount/Container.PageSize%" />
    </PagerTemplate>
</asp:TemplatePagerField>
</Fields>
</asp:DataPager>

```

注意, 这个例子使用 ASP.NET 数据绑定提供总页数、页面尺寸和页面开始的行, 这些值都是通过 DataPager 控件显示的。

如果要在 PagerTemplate 中使用定制的导航控件, 例如使用 Button 控件改变当前显示的页面, 就应为该控件创建标准的 Click 事件处理程序。在这个事件处理程序中, 可以访问 DataPager 的 StartRowIndex、TotalRowCount 和 PageSize 属性, 计算 ListView 在显示时应使用的新 StartRowIndex。

与 GridView 提供的分页功能不同, DataPager 是一个独立的控件, 可以把它自由地放在 Web 页面上。前面的所有例子都是把 DataPager 控件直接放在 ListView 中, 但这个控件可以放在 Web 窗体的任何地方。在程序清单 8-46 中, 一个重要的变化是使用了 PagedControlID 属性。

程序清单 8-46 把 DataPager 控件放在 ListView 的外部

```

<asp:DataPager ID="DataPager1" runat="server"
    PagedControlID="ListView1">
    <Fields>
        <asp:NumericPagerField />
    </Fields>
</asp:DataPager>

```

PagedControlID 属性可以指定这个分页程序应使用哪个控件。

8.3.7 FormView 控件

FormView 控件的工作方式类似于 DetailsView 控件, 因为它也是显示绑定数据源控件中的一个数据项, 并可以添加、编辑和删除数据。它的独特之处在于是在定制模板中显示数据, 可以更多地控制数据的显示和编辑方式。FormView 控件还包含 EditItemTemplate 和 InsertItemTemplate, 它可以确定控件在进入编辑或插入模式下的显示方式。程序清单 8-47 是在设计 FormView 控件的定制 ItemTemplate 时 Visual Studio 生成的代码。

程序清单 8-47 使用 FormView 控件显示和编辑数据

```

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Using the FormView control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:FormView ID="FormView1" runat="server"
            DataSourceID="SqlDataSource1"

```

```
DataKeyNames="CustomerID" AllowPaging="True">
<EditItemTemplate>
    CustomerID:
    <asp:Label Text='<%= Eval("CustomerID") %>'
        runat="server" ID="CustomerIDLabel1">
    </asp:Label><br />
    CompanyName:
    <asp:TextBox Text='<%= Bind("CompanyName") %>'
        runat="server"
        ID="CompanyNameTextBox"></asp:TextBox><br />
    ContactName:
    <asp:TextBox Text='<%= Bind("ContactName") %>'
        runat="server"
        ID="ContactNameTextBox"></asp:TextBox><br />
    ContactTitle:
    <asp:TextBox Text='<%= Bind("ContactTitle") %>'
        runat="server"
        ID="ContactTitleTextBox"></asp:TextBox><br />
    Address:
    <asp:TextBox Text='<%= Bind("Address") %>'
        runat="server"
        ID="AddressTextBox"></asp:TextBox><br />
    City:
    <asp:TextBox Text='<%= Bind("City") %>' runat="server"
        ID="CityTextBox"></asp:TextBox><br />
    Region:
    <asp:TextBox Text='<%= Bind("Region") %>'
        runat="server"
        ID="RegionTextBox"></asp:TextBox><br />
    PostalCode:
    <asp:TextBox Text='<%= Bind("PostalCode") %>'
        runat="server"
        ID="PostalCodeTextBox"></asp:TextBox><br />
    Country:
    <asp:TextBox Text='<%= Bind("Country") %>'
        runat="server"
        ID="CountryTextBox"></asp:TextBox><br />
    Phone:
    <asp:TextBox Text='<%= Bind("Phone") %>' runat="server"
        ID="PhoneTextBox"></asp:TextBox><br />
    Fax:
    <asp:TextBox Text='<%= Bind("Fax") %>' runat="server"
        ID="FaxTextBox"></asp:TextBox><br />
    <br />
    <asp:Button ID="Button2" runat="server" Text="Button"
        CommandName="update" />
    <asp:Button ID="Button3" runat="server" Text="Button"
        CommandName="cancel" />
</EditItemTemplate>
<ItemTemplate>
    <table width="100%">
        <tr>
            <td style="width: 439px">
                <b>
                    <span style="font-size: 14pt">
```



```

        Customer Information</span>
</b>
</td>
<td style="width: 439px" align="right">
    CustomerID:
    <asp:Label ID="CustomerIDLabel"
        runat="server"
        Text='<%# Bind("CustomerID") %>''>
    </asp:Label></td>
</tr>
<tr>
<td colspan="2">
    CompanyName:
    <asp:Label ID="CompanyNameLabel"
        runat="server"
        Text='<%# Bind("CompanyName") %>''>
    </asp:Label><br />
    ContactName:
    <asp:Label ID="ContactNameLabel"
        runat="server"
        Text='<%# Bind("ContactName") %>''>
    </asp:Label><br />
    ContactTitle:
    <asp:Label ID="ContactTitleLabel"
        runat="server"
        Text='<%# Bind("ContactTitle") %>''>
    </asp:Label><br />
<br />
<table width="100%"><tr>
<td colspan="3">
    <asp:Label ID="AddressLabel"
        runat="server"
        Text='<%# Bind("Address") %>''>
    </asp:Label></td>
</tr>
<tr>
<td style="width: 100px">
    <asp:Label ID="CityLabel"
        runat="server"
        Text='<%# Bind("City") %>''>
    </asp:Label></td>
<td style="width: 100px">
    <asp:Label ID="RegionLabel"
        runat="server"
        Text='<%# Bind("Region") %>''>
    </asp:Label></td>
<td style="width: 100px">
    <asp:Label ID="PostalCodeLabel"
        runat="server"
        Text='<%# Bind("PostalCode") %>''>
    </asp:Label>
</td>
</tr>
<tr>
<td style="width: 100px" valign="top">

```

```

        <asp:Label ID="CountryLabel"
            runat="server"
            Text='<%= Bind("Country") %>'>
        </asp:Label></td>
    <td style="width: 100px"></td>
    <td style="width: 100px">Phone:
        <asp:Label ID="PhoneLabel"
            runat="server"
            Text='<%= Bind("Phone") %>'>
        </asp:Label><br />
        Fax:
        <asp:Label ID="FaxLabel"
            runat="server"
            Text='<%= Bind("Fax") %>'>
        </asp:Label><br />
    </td>
</tr></table>
<asp:Button ID="Button1" runat="server"
    Text="Button" CommandName="edit" />
</td>
</tr></table>
</ItemTemplate>
</asp:FormView>
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    SelectCommand="SELECT * FROM [Customers]"
    ConnectionString=
        "<%= ConnectionStrings:ConnectionString %>"
    </asp:SqlDataSource>
</div>
</form>
</body>
</html>

```

8.4 其他数据绑定控件

ASP.NET 还包含许多其他可以绑定到数据源的简单控件。本节将介绍一些这样的控件，以及如何在 Web 应用程序中将它们连接到数据。

8.4.1 TreeView 控件

TreeView 只能显示层次结构的数据，因此只能绑定到 XmlDataSource 和 SiteMapDataSource 控件，这两个控件是专门为绑定到层次结构数据源(如 SiteMap 文件)而设计的控件。程序清单 8-48 是一个可用于 SiteMapDataSource 控件的示例 SiteMap 文件。

程序清单 8-48 示例 SiteMap 文件

```

<siteMap>
    <siteMapNode url="page3.aspx" title="Home" description="" roles="">
        <siteMapNode url="page2.aspx"
            title="Content" description="" roles="" />
        <siteMapNode url="page4.aspx"

```

```

        title="Links" description="" roles="" />
    <siteMapNode url="page1.aspx"
        title="Comments" description="" roles="" />
    </siteMapNode>
</siteMap>

```

程序清单 8-49 演示了如何把 TreeView 控件绑定到 SiteMapDataSource 控件，以生成 Web 站点的导航。

程序清单 8-49 在 SiteMapDataSource 控件中使用 TreeView

```

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Using the TreeView control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:TreeView ID="TreeView1" runat="server" DataSourceID="SiteMapDataSource1" />
            <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
        </div>
    </form>
</body>
</html>

```

8.4.2 Menu 控件

与 TreeView 控件一样，Menu 控件也可以在垂直的弹出式菜单中显示层次结构数据。Menu 控件也只能绑定到 XmlDataSource 和 SiteMapDataSource 控件。程序清单 8-50 演示了如何使用之前在 TreeView 控件示例中利用的 SiteMap 数据，把它改为使用 Menu 控件显示。

程序清单 8-50 在 SiteMapDataSource 控件中使用 Menu 控件

```

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Using the Menu control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1" />
            <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
        </div>
    </form>
</body>
</html>

```

8.4.3 Chart 控件

这个控件非常适合于设计并运行漂亮的图表。Chart 服务器控件支持许多图表类型，包括：

- Point
- FastPoint
- Bubble
- Line
- Spline
- StepLine
- FastLine
- Bar
- StackedBar
- StackedBar100
- Column
- StackedColumn
- StackedColumn100
- Area
- SplineArea
- StackedArea
- StackedArea100
- Pie
- Doughnut
- Stock
- CandleStick
- Range
- SplineRange
- RangeBar
- RangeColumn
- Radar
- Polar
- ErrorBar
- BoxPlot
- Renko
- ThreeLineBreak
- Kagi
- PointAndFigure
- Funnel
- Pyramid

有这么多不同的图表样式！Chart 服务器控件在 Visual Studio 2012 工具箱的 Data 选项卡下，是 System.Web.DataVisualization 名称空间的一部分。

把 Chart 控件从工具箱拖放到页面的设计界面上，就会显示可用图表类型的可视化表示，图 8-11 是一个例子。

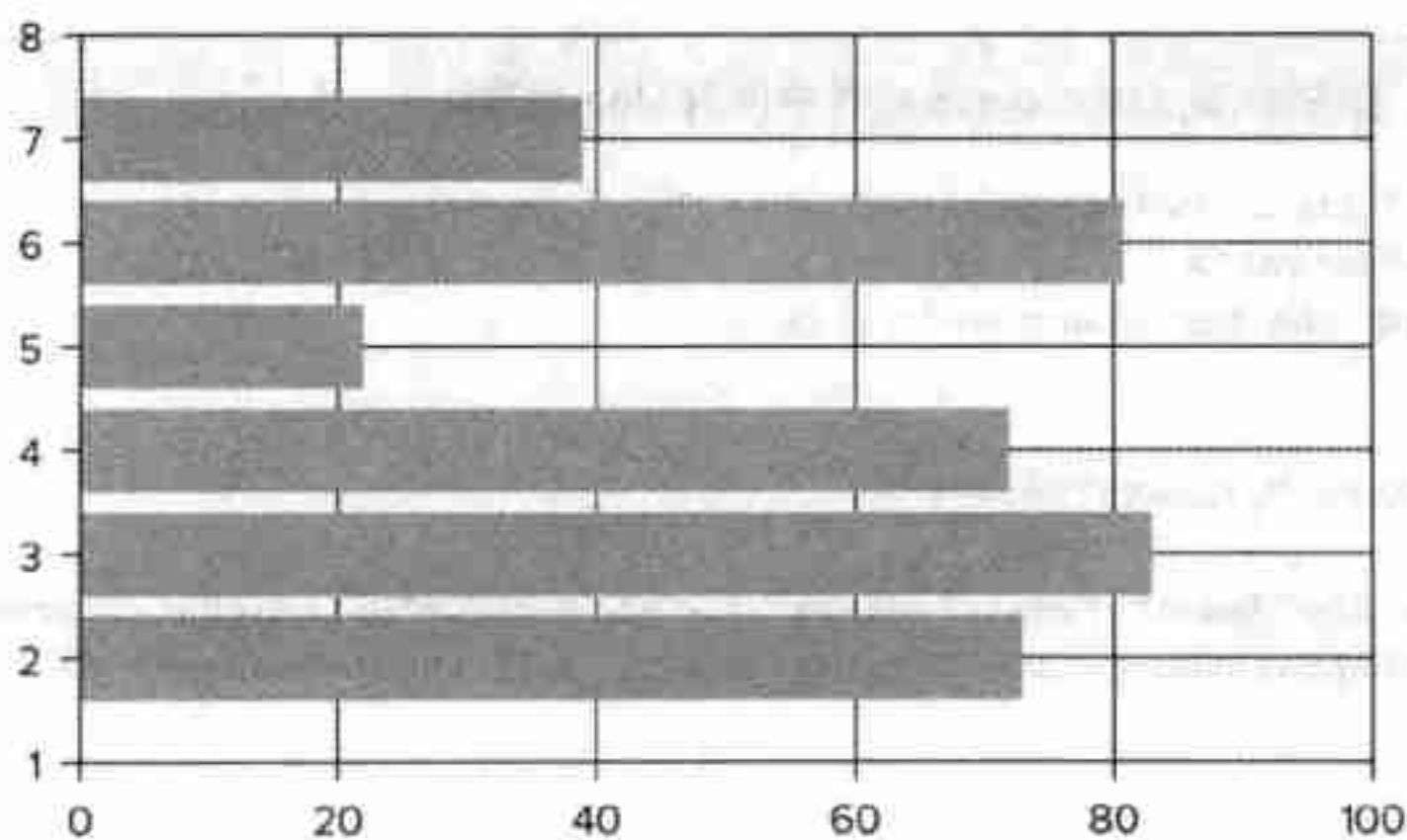


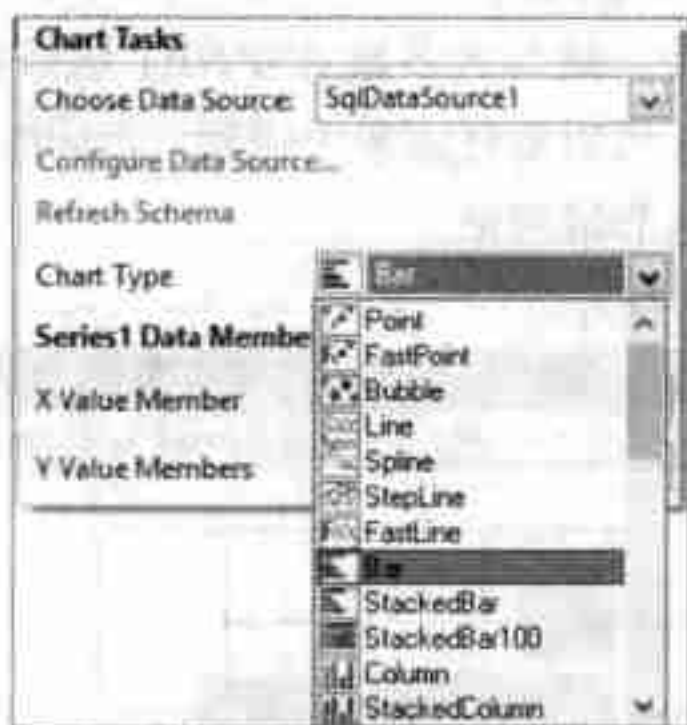
图 8-11

打开该控件的智能标记，就可以给图表分配数据提供程序，选择要使用的图表类型。改变图表类型，会在 IDE 的设计视图中显示该图表的外观(即使还没有使用任何底层的数据)。与所有其他的控件都不同，这个控件的内容很多，差不多需要一整本书的篇幅来介绍。为了开始使用 Chart 服务器控件，试试下面的简单例子。

创建一个新的 Web 应用程序，在应用程序的 App_Data 文件夹中添加 Northwind 数据库。之后，把 Chart 服务器控件拖放到页面的设计界面上。选择数据源时，在该控件的智能标记中，从下拉菜单中选择<New Data Source>。按照这个向导的指令进行，确保选择 SQL 数据源选项。在这个过程中，所选择的选项应允许选择定制的 SQL 语句，给这个操作使用如下 SQL：

```
SELECT TOP(5) [ProductName], [UnitsInStock] FROM [Products]
ORDER BY ProductName DESC
```

完成该操作,把 Chart 服务器控件绑定到这个数据源控件后,在 Chart 服务器控件的智能标记中就会有更多的选项,如图 8-12 所示。



8-12

现在可以选择一系列数据成员，选择 x 轴和 y 轴的数据。这里把产品名赋予 x 轴，把预订的数量作为 y 轴。加大图表的宽度后，代码应如程序清单 8-51 所示。

程序清单 8-51 用新的 Chart 服务器控件建立的图表

```
<@ Register Assembly="System.Web.DataVisualization, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35"
Namespace="System.Web.UI.DataVisualization.Charting" TagPrefix="asp" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
<title>MultiView Server Control</title>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:Chart ID="Chart1" runat="server"
DataSourceID="SqlDataSource1"
Width="500px">
<Series>
<asp:Series ChartType="Bar" Name="Series1"
XValueMember="ProductName"
YValueMembers="UnitsInStock" YValuesPerPoint="2">
</asp:Series>
</Series>
<ChartAreas>
<asp:ChartArea Name="ChartArea1">
</asp:ChartArea>

```

```

        </ChartAreas>
    </asp:Chart>
    <asp:SqlDataSource ID="SqlDataSource1" runat="server"
        ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
        SelectCommand="SELECT TOP (5) [ProductName], [UnitsInStock] FROM [Products]
        ORDER BY ProductName DESC"></asp:SqlDataSource>
</div>
</form>
</body>
</html>

```

从中可以看出,完成这个任务需要的代码不多。最明显的是,把 **Chart** 服务器控件放在页面上,会在页面的顶部添加 **@Register** 指令,这与大多数 ASP.NET 服务器控件都不同。

在这个控件的 **<Series>** 元素中,可以有任意多个序列,在同时为多个数据项建立图表(例如两只或多只股票价格的时间序列)时,这非常常见。

运行这些代码,结果如图 8-13 所示。

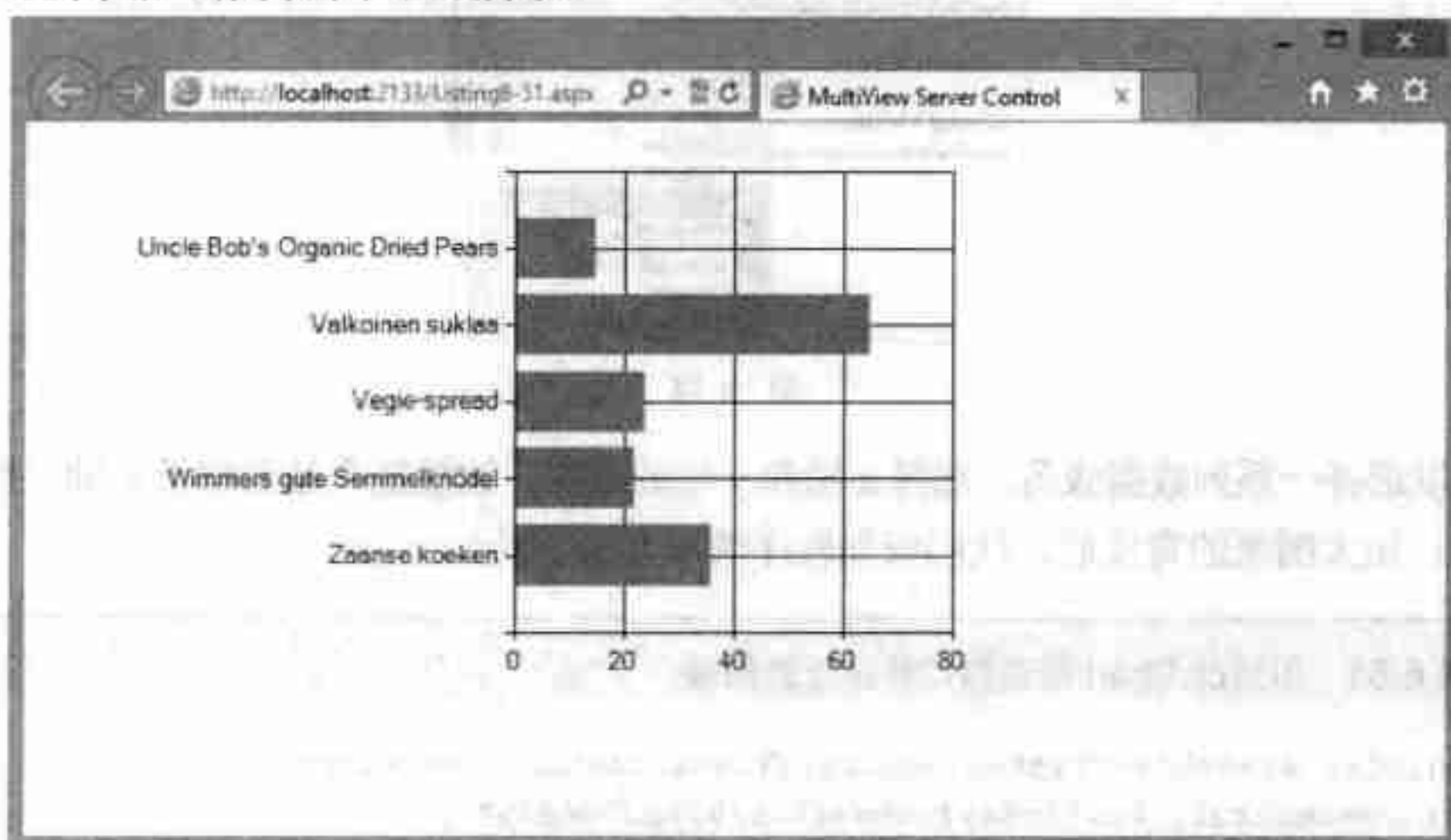


图 8-13

8.5 内联数据绑定语法

ASP.NET 中的另一个数据绑定功能是内联数据绑定语法。ASP.NET 1.0/1.1 中的内联语法主要委托给模板化控件,如 **DataList** 和 **Repeater** 控件,但即使在当时,有时也很难使它们按照希望的方式工作。在 ASP.NET 1.0/1.1 中,如果需要使用内联数据绑定,就要创建如程序清单 8-52 所示的过程。

程序清单 8-52 在 ASP.NET 1.0 中使用内联数据绑定

```

<asp:Repeater ID="Repeater1" runat="server"
    DataSourceID="SqlDataSource1">
    <HeaderTemplate>
        <table>
    </HeaderTemplate>
    <ItemTemplate>
        <tr>

```



```

        <td>
            <%# Container.DataItem("ProductID")%><br/>
            <%# Container.DataItem("ProductName")%><br/>
            <%# DataBinder.Eval(
                Container.DataItem, "UnitPrice", "{0:c}")%><br/>
        </td>
    </tr>
</ItemTemplate>
<FooterTemplate>
</table>
</FooterTemplate>
</asp:Repeater>

```

在这个例子中，使用 Repeater 控件显示了一系列雇员。由于 Repeater 控件是模板控件，因此使用数据绑定在模板的相应位置输出与雇员相关的数据。使用 Eval 方法还可以提供格式化信息，例如显示 Date 或 Currency 格式的信息。

在 ASP.NET 的后续版本中，内联数据绑定的概念基本不变，但是简化了语法，并且用户可以使用几个强大的绑定工具。

8.5.1 数据绑定语法

ASP.NET 提供了 3 种不同方式来进行数据绑定。首先，可以继续使用 Container.DataItem 语法，利用已有的绑定方法：

```
<%# Container.DataItem("Name") %>
```

这种方式很不错，因为如果要从 ASP.NET 的以前版本中迁移，就不需要修改已有的 Web 页面。但如果创建新的 Web 页面，就应采用最简单的绑定形式，即直接使用 Eval 方法：

```
<%# Eval("Name") %>
```

还可以继续使用 Eval 方法的格式化重载版本来格式化数据：

```
<%# Eval("HireDate", "{0:mm dd yyyy}" ) %>
```

除了这些变化之外，ASP.NET 还包括一种新形式的数据绑定，称为双向数据绑定。双向数据绑定支持绑定数据的读取和写入操作。这是使用 Bind 方法实现的，除了方法名不同之外，该方法的工作方式与 Eval 方法类似：

```
<%# Bind("Name") %>
```

Bind 方法应在控件 GridView、DetailsView 或 FormView 中使用，这些控件能够自动更新数据源。

在使用数据绑定语句时，<%#与%>定界符之间的所有内容都作为表达式来处理。这一点很重要，因为在执行数据绑定时，它提供了一些额外的功能。例如，可以附加额外的数据：

```
<%# "Foo " + Eval("Name") %>
```

或者给方法传送计算出来的值：

```
<%# DoSomeProcess( Eval("Name") )%>
```

8.5.2 XML 数据绑定

XML 在应用程序中越来越普遍, 因此 ASP.NET 也有几种专门用于绑定 XML 数据源的方式。这些数据绑定表达式可以处理 XML 的层次格式。另外, 除了方法名不同之外, 这些绑定方法的工作方式与前面讨论的 Eval 和 Bind 方法完全相同。这些绑定方法应在使用 XmlDataSource 控件时应用。使用 XPathBinder 类的第一种绑定格式如下所示:

```
<% XPathBinder.Eval(Container.DataItem, "employees/employee/Name") %>
```

注意, XPathBinder 没有像 Eval 方法那样指定列名, 而是绑定 XPath 查询的结果。与标准的 Eval 表达式一样, XPath 数据绑定表达式也有一种简写格式:

```
<% XPath("employees/employee/Name") %>
```

另外, 与 Eval 方法一样, XPath 数据绑定表达式也可以把格式应用于数据:

```
<% XPath("employees/employee/HireDate", "{0:mm dd yyyy}") %>
```

XPathBinder 类使用提供的 XPath 查询返回一个节点。如果要从 XmlDataSource 控件中返回多个节点, 可以使用 XPathBinder 类的 Select 方法。这个方法返回匹配 XPath 查询的节点列表。

```
<% XPathBinder.Select(Container.DataItem, "employees/employee") %>
```

或者使用简写格式:

```
<% XpathSelect("employees/employee") %>
```

8.6 表达式和表达式生成器

表达式是由 ASP.NET 在运行期间分析以返回数据值的语句。ASP.NET 在分析 SqlDataSource 控件时, 会自动使用表达式完成检索数据库连接字符串等任务, 因此我们在页面上可能看到过这些语句。程序清单 8-53 显示了一个 ConnectionString 表达式。

程序清单 8-53 ConnectionString 表达式

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
  SelectCommand="SELECT * FROM [Customers]"
  ConnectionString="<%%$ ConnectionStrings:ConnectionString %>" />
```

当 ASP.NET 尝试分析 ASP.NET Web 页面时, 会查找包含在 <%%\$ 与 %> 定界符之间的表达式。<%%\$ 与 %> 定界符告诉 ASP.NET, 这是一个要分析的表达式。如程序清单 8-53 所示, ASP.NET 尝试在 web.config 文件中定位 NorthwindConnectionString 值。ASP.NET 知道要查找该值是因为 ConnectionString 表达式的前缀, 它告诉 ASP.NET 使用 ConnectionStringsExpressionBuilder 类来分析表达式。

ASP.NET 有几个表达式生成器, 包括从 web.config 文件的 AppSettings 部分中检索值的表达式生成器、检索 ConnectionString 值的表达式生成器(如程序清单 8-53 所示), 以及检索本地资源文件值的表达式生成器。程序清单 8-54 和 8-55 分别演示了如何使用 AppSettingsExpressionBuilder 和

ResourceExpressionBuilder。

程序清单 8-54 使用 AppSettingsExpressionBuilder

```
<asp:Label runat="server" ID="Label1"
    Text="<%$ AppSettings: LabelText %>" />
```

程序清单 8-55 使用 ResourceExpressionBuilder

```
<asp:Label runat="server" ID="Label1"
    Text="<%$ Resources: MyAppResources, Label1Text %>" />
```

除了使用表达式生成器类之外,还可以通过从 System.Web.Compilation.ExpressionBuilder 基类派生类来创建自己的表达式。这个基类提供了几个可重写的方法,如果希望 ASP.NET 正确分析表达式,就需要实现这些可重写的方法。程序清单 8-56 是一个简单的定制表达式生成器。

程序清单 8-56 使用简单的定制表达式生成器

```
[ExpressionPrefix("MyFirstCustomExpression")]
[ExpressionEditor("MyFirstCustomExpressionEditor")]
public class MyFirstCustomExpression : ExpressionBuilder
{
    public override System.CodeDom.CodeExpression
        GetCodeExpression(BoundPropertyEntry entry, object parsedData,
            ExpressionBuilderContext context)
    {
        return new CodeCastExpression("Int64",
            new CodePrimitiveExpression(1000));
    }
}
```

在这个例子中有几个地方值得注意。首先,如前所述,从 ExpressionBuilder 类派生了 MyCustomExpression 类。其次,重写了 GetCodeExpression 方法。这个方法提供了帮助执行该方法的几个参数,并且给 ASP.NET 返回一个 CodeExpression 对象,以便 ASP.NET 在运行期间检索数据值。



CodeExpression 类是 .NET 的 CodeDom 基础架构中的一个基类。派生于 CodeExpression 的类提供了生成 .NET 代码的抽象方式(VB 和 C#),这个 CodeDom 基础架构可帮助在运行期间动态地创建和运行代码。

BoundPropertyEntry 参数项指定表达式绑定到哪个属性。例如,在程序清单 8-54 和 8-55 中,Label 的 Text 属性绑定到 AppSettings 和 Resources 表达式。对象参数 parsedData 包含由 ParseExpression 方法(详见本章后面的内容)分析和返回的数据。最后,ExpressionBuilderContext 参数上下文允许引用与表达式相关的虚拟路径或模板化的控件。

在 GetCodeExpression 方法的主体中,创建一个新的 CodeCastExpression 对象,CodeCastExpression 是一个派生于 CodeExpression 基类的类。CodeCastExpression 告诉 .NET 要生成相应代码,将一种数据类型强制转换为另一种数据类型。在这个例子中,是将值 1000 强制转换为 Int64 数据类型。在 .NET

执行 `CodeCastExpression` 时, 会输出 C# 代码 `(long)(1000)` 或 VB 代码 `CType(1000, long)`。注意派生于 `CodeExpression` 的许多类都可用于生成最终的代码表达式。

要注意的最后一点是添加到类中的两个属性: `ExpressionPrefix` 和 `ExpressionEditor`, 它们会帮助 .NET 确定把这个类用作表达式。在分析表达式时, 这两个属性还会帮助 .NET 定位正确的表达式生成器类。

创建了表达式生成器类后, 就要把这个情况告诉 .NET, 为此, 可以在 `web.config` 文件的 `compilation` 节点中添加 `expressionBuilders` 节点。注意将 `ExpressionPrefix` 的值添加到 `expressionBuilder` 中, 可帮助 ASP.NET 在运行期间定位正确的表达式生成器类。

```
<compilation debug="true" strict="false" explicit="true">
  <expressionBuilders>
    <add expressionPrefix="MyCustomExpression" type="MyCustomExpression"/>
  </expressionBuilders>
</compilation>
```

`GetCodeExpression` 方法不是 `ExpressionBuilder` 类中可重写的唯一成员, 还有其他几个有用的成员, 包括 `ParseExpression` 方法、`SupportsEvaluate` 方法和 `EvaluateExpression` 方法。

`ParseExpression` 方法可以把分析过的表达式数据传送给 `GetCodeExpression` 方法。例如, 在程序清单 8-56 中, `CodeCastExpression` 值 1000 是硬编码的。但是, 如果要想让开发人员把这个值作为表达式的一部分传送进来, 就可以使用 `ParseExpression` 方法, 如程序清单 8-57 所示。

程序清单 8-57 使用 `ParseExpression` 方法

```
[ExpressionPrefix("MySecondCustomExpression")]
[ExpressionEditor("MySecondCustomExpressionEditor")]
public class MySecondCustomExpression : ExpressionBuilder
{
    public override System.CodeDom.CodeExpression
        GetCodeExpression(BoundPropertyEntry entry, object parsedData,
            ExpressionBuilderContext context)
    {
        return new CodeCastExpression("Int64",
            new CodePrimitiveExpression(parsedData));
    }
    public override object ParseExpression
        (string expression, Type propertyType,
            ExpressionBuilderContext context)
    {
        return expression;
    }
}
```

最后介绍的两个 `ExpressionBuilder` 重写成员是 `SupportsEvaluate` 方法和 `EvaluateExpression` 方法。如果要在非编译的情形下运行 Web 站点(在 `@Page` 指令中指定 `compilationMode="never"`), 就需要重写这两个方法。在页面以非编译模式执行时, `SupportsEvaluate` 属性返回一个布尔值, 告诉 ASP.NET 这个表达式是否能计算。如果返回 `True`, 并且页面在非编译模式下执行, 就使用 `EvaluateExpression` 方法返回数据值, 而不是使用 `GetCodeExpression` 方法。`EvaluateExpression` 方法返回一个表示数据值的对象, 如程序清单 8-58 所示。

程序清单 8-58 重写 SupportsEvaluate 和 EvaluateExpression 方法

```

[ExpressionPrefix("MyThirdCustomExpression")]
[ExpressionEditor("MyThirdCustomExpressionEditor")]
public class MyThirdCustomExpression : ExpressionBuilder
{
    public override System.CodeDom.CodeExpression
        GetCodeExpression(BoundPropertyEntry entry, object parsedData,
            ExpressionBuilderContext context)
    {
        return new CodeCastExpression("Int64",
            new CodePrimitiveExpression(parsedData));
    }
    public override object ParseExpression
        (string expression, Type propertyType,
            ExpressionBuilderContext context)
    {
        return expression;
    }
    public override bool SupportsEvaluate
    {
        get
        {
            return true;
        }
    }
    public override object EvaluateExpression(object target,
        BoundPropertyEntry entry, object parsedData,
        ExpressionBuilderContext context)
    {
        return parsedData;
    }
}

```

如程序清单 8-58 所示, 如果要重写 EvaluateExpression 方法, 只需从 SupportsEvaluate 属性中返回 True, 然后从 EvaluateExpression 方法中返回一个对象即可。

8.7 本章小结

本章介绍了 ASP.NET 中的数据绑定。数据源控件(如 LinqDataSource、SqlDataSource 或 XmlDataSource 控件)的引入大大简化了任意数据源中的数据查询和显示。使用数据源控件的向导, 可以轻松地生成强大的数据访问功能, 而几乎不需要编写任何代码。

你还了解到, 甚至开发新手都可以轻松地将数据源控件与 GridView、ListView 和 DetailsView 控件结合起来使用, 从而创建强大且编码量最少的数据操作应用程序。

本章还介绍了 ASP.NET 包含的大量可以绑定数据的控件, 论述了 ASP.NET 工具箱中包含的数据绑定控件的功能, 如 GridView、TreeView、ListView、FormView 和 Menu 控件。

本章最后介绍了改进的内联数据绑定语法, 以及增强的 XML 专用数据绑定表达式。

第9章

模型绑定

本章要点

- 理解模型绑定
- 使用强类型化的控件
- 扩展模型绑定

模型绑定在 ASP.NET MVC 中引入，便于将客户端提交的数据转换为适合在服务器上使用和验证的格式。模型绑定在 ASP.NET MVC 1.0 架构中引入，ASP.NET MVC 2.0 发布时，又引入了一个新系统——可扩展的模型绑定，使开发人员在把客户端数据绑定到服务器方面有更强的功能和更大的灵活性。

在 ASP.NET 4.5 推出之前，这个系统只能用于 ASP.NET MVC，但在 ASP.NET 4.5 中，模型绑定也可以用于建立 ASP.NET Web 窗体应用程序。ASP.NET 4.5 Web 窗体中的模型绑定系统基于可扩展的模型绑定系统。

上一章学习了如何使用数据控件进行数据绑定。本章将学习如何利用模型绑定进行数据绑定，了解使用模型绑定的优点。

9.1 模型绑定

模型绑定用于两个目的：提供了一种从客户端提取数据，再把它们绑定到模型的方式；还提供了一种验证机制——一旦绑定模型，就可以运行验证规则，在保存模型之前，确定模型是否有效。ASP.NET Web 窗体中的模型绑定把模型绑定体系结构的功能引入了控件，提供了一个系统，允许开发人员更方便地集成模式，例如 PRG(Post-Redirect-Get，传递-重定向-获得)、Repository(资源库)等。它还使应用程序代码更简洁、应用程序的单元测试更方便。

模型绑定将关联到现有的数据绑定控件。使用模型绑定可以从控件中提取值，这样客户端的值就可以用于服务器上的检查，控件也可以绑定到从模型绑定系统中返回的值。模型绑定与已有的数据绑定控件一起使用，因此可以使用上一章学习的概念。

例如,假定有一个 Web 页面,用户可以在其中使用 DetailsView 控件输入个人信息。用户输入个人信息,并将该页面提交给服务器时,模型绑定系统就会从 DetailsView 控件中提取值,以便在服务器上进行某些验证操作。如果验证成功,就可以把这个人的信息保存到数据库中。此时,这个人就是模型绑定系统与之交互的模型。模型可以是 ADO.NET Entity Framework 模型或 Entity Framework Code First 模型。



这些模型详见第 11 章。

下面介绍使用模型绑定的一些常见数据操作。

9.1.1 选择数据

首先创建一个新的 Web 窗体页面,使用 GridView 控件选择并过滤顾客数据。程序清单 9-1 说明了如何配置 GridView 控件,以使用模型绑定提取数据。这个控件使用两个新属性: ItemType 和 SelectMethod。

- ItemType 告诉模型绑定系统,控件应绑定到什么类型的模型。ItemType 属性是可选的,包含它,就可以为数据绑定表达式获得强类型化的 IntelliSense。本章后面将详细介绍这些强类型化的控件。
- SelectMethod 属性确定在页面上调用什么方法来获得记录。程序清单 9-2 显示了选择的方法 (SelectCustomers)。

程序清单 9-1 使用了模型绑定的 GridView

```
<asp:GridView ID="GridView1" runat="server" ItemType="Customer" AllowPaging="true"
AllowSorting="true" PageSize="2" SelectMethod="SelectCustomers"></asp:GridView>
```

程序清单 9-2 模型绑定 Select 方法的示例

```
CustomerContext _context = new CustomerContext();
public IEnumerable<Customer> SelectCustomers()
{
    return _context.Customer.AsEnumerable();
}
```

在程序清单 9-2 中, SelectCustomers 方法返回一系列 Customer 对象。如这个程序清单所示,我们首先实例化 CustomerContext,你在使用 Entity Framework Code First 时会用到它。实例化 CustomerContext 时,可以访问所有的 Customer 对象,执行任意 LINQ 查询来过滤结果。

9.1.2 分页

GridView 控件有内置的分页和排序功能。使用模型绑定,可以从 SelectCustomers 方法中返回一个 IQueryable<T>以利用这些功能。程序清单 9-3 说明了如何启用分页和排序功能。为了获得分页和排序功能,需要在 GridView 控件上启用分页和排序功能,如程序清单 9-1 所示。

程序清单 9-3 使用模型绑定进行分页和排序

```

CustomerContext _context = new CustomerContext();
public IQueryable<Customer> SelectCustomers()
{
    return _context.Customer.AsQueryable();
}

```

在这个程序清单中，注意 `SelectCustomers` 方法返回 `IQueryable`，而不是 `IEnumerable`。从 `SelectCustomers` 方法中返回 `IQueryable<T>`，并启用 `GridView` 控件的排序/分页功能时，`GridView` 可以自动启用分页和排序功能。

9.1.3 过滤

我们常常需要过滤给用户显示的结果，而不希望返回表中的所有记录。上一章学习了如何使用 `SqlDataSource` 控件中的 `Select` 参数来过滤数据。在模型绑定中，使用 `ValueProviders` 和 `ValueProvider` 特性也可以获得过滤功能。程序清单 9-4 说明了如何使用 `QueryString` 中的值来过滤数据。在这个示例中，添加一个查询字符串，如 `?ID=5`，就只返回 ID 为 5 的记录。

程序清单 9-4 使用查询字符串来过滤数据

```

public IEnumerable<Customer> SelectCustomers([System.Web.ModelBinding.QueryString] int? id)
{
    if(id.HasValue)
        return _context.Customer.Where(c => c.ID == id).AsEnumerable();
    else
        return _context.Customer.AsEnumerable();
}

```

在 `QueryString` 中传递 `ID=5` 时，模型绑定系统就会提取这个值，作为参数提供给 `SelectCustomers` 方法。因为本例使用了 `Entity Framework Code First`，所以可以编写一个 LINQ 查询，过滤出 ID 为 5 的顾客列表，并把结果返回给 `GridView` 控件。

9.1.4 使用值提供程序

程序清单 9-4 说明了如何使用 `QueryString` 值提供程序特性来过滤结果。这个特性告诉模型绑定系统，在运行期间把查询字符串中的一个值绑定到 `id` 参数。模型绑定系统会根据需要进行类型转换。

值提供程序特性最终从值提供程序中获取值，并告诉模型绑定系统使用哪个值提供程序。表 9-1 列出了架构中可用的值提供程序特性。

表 9-1

值提供程序特性	说 明
Form	值从 Form 集合中检索
Control	值从指定的控件中检索
QueryString	值从 QueryString 集合中检索
Cookie	值从 Cookie 集合中检索
Profile	值从 Profile 集合中检索

(续表)

值提供程序特性	说 明
RouteData	值从 RouteData 集合中检索
Session	值从 Session 集合中检索

使用控件进行过滤

下面介绍一个高级示例，这个示例将根据另一个服务器控件中的值过滤结果。我们常常需要根据(例如)下拉列表来过滤值。程序清单 9-5 说明了如何使用 Control 值提供程序特性从下拉列表控件中检索值，接着使用该值过滤结果，在 GridView 控件中显示它们。

程序清单 9-5 使用控件进行过滤

```
<asp:DropDownList ID="DropDown1" runat="server" ItemType="Customer"
    SelectMethod="SelectCustomersForDropDownList" AppendDataBoundItems="true"
    AutoPostBack="true"
    DataTextField="ID" DataValueField="ID">
</asp:DropDownList>
<asp:GridView ID="GridView1" runat="server" ItemType="Customer"
    SelectMethod="SelectCustomers">
</asp:GridView>

public IEnumerable<Customer> SelectCustomers([System.Web.ModelBinding.Control] int?
DropDown1)
{
    if (DropDown1.HasValue)
        return _context.Customer.Where(c => c.ID == DropDown1).AsEnumerable();
    else
        return _context.Customer.AsEnumerable();
}

public IEnumerable<Customer> SelectCustomersForDropDownList()
{
    return _context.Customer.AsEnumerable();
}
```

在这个程序清单中，当从 DropDownList 控件中选择值时，GridView 就会绑定到从 DropDownList 控件中选择的值。模型绑定系统会提取从 DropDownList 控件中选择的值(本例是 ID)，使用 Control 值提供程序特性把它作为参数传递给 SelectCustomers 方法。

9.1.5 插入数据

在 ASP.NET 4.5 中，数据绑定控件更新为使用模型绑定。现在要学习如何使用模型绑定插入记录。使用 DetailsView 控件就可以插入记录。在这个控件上，必须设置新属性 InsertMethod，它可以在页面上调用。程序清单 9-6 列出了 InsertCustomers 方法。

程序清单 9-6 使用 InsertCustomers 方法和模型绑定

```

<asp:DetailsView runat="server" ItemType="Customer" SelectMethod="SelectCustomers"
    InsertMethod="InsertCustomer" AutoGenerateInsertButton="true" AllowPaging="true">
</asp:DetailsView>

public void InsertCustomer(Customer customer)
{
    _context = new CustomerContext();
    if (ModelState.IsValid)
    {
        _context.Customer.Add(customer);
    }
}

```

仔细查看 InsertCustomer, 注意这个方法的输入是 Customer 类型, 这是 DetailsView 控件要绑定的模型的类型(这是通过控件的 ItemType 属性指定的)。

插入值时, 模型绑定系统就从 DetailsView 控件中提取值, 并填充 Customer 类型的模型, 这样 Customer 模型就可以在服务器上用于任何类型的验证。调用 InsertCustomer 时, 可以进行检查, 确定在模型绑定系统中是否有验证错误。如果一切正常, 就可以把这个新的 Customer 添加到 Customer 集合中, 并保存到数据库中。

9.1.6 更新数据

插入记录时使用的绑定方法可以处理简单的情形。但是, 数据控件常常不能为模型的每个成员提供值, 要么因为这些成员是关系对象, 要么因为它们不显示在控件中。在这种情况下, 最好提取主键, 从数据存储中加载模型, 并告诉模型绑定系统, 把数据控件中的值绑定到模型。程序清单 9-7 说明了 UpdateMethod 如何更新记录。

程序清单 9-7 模型绑定以主键作为参数进行更新

```

<asp:DetailsView runat="server" ItemType="Customer" SelectMethod="SelectCustomers"
    UpdateMethod="UpdateCustomer" AutoGenerateEditButton="true" DataKeyNames="ID"
    AllowPaging="true">
</asp:DetailsView>

CustomerContext _context = new CustomerContext();
public Customer SelectCustomers()
{
    return _context.Customer.First();
}

public void UpdateCustomer(int id)
{
    _context = new CustomerContext();
    var customer = _context.Customer.Where(c => c.ID == id).First();
    TryUpdateModel(customer);
    if (ModelState.IsValid)
    {
    }
}

```


程序清单 9-7 演示了如何更新一条记录。所有的数据绑定控件都有属性 `DataKeyNames`，它唯一地标识了一条记录。这个属性使用主键字段的名称。`DetailsView` 控件执行插入操作时，模型绑定系统就用 `DataKeyNames` 属性的值填充更新方法中的参数。参数 `id` 现在包含主键 ID 的值，可以用于检索用户正在更新的记录。调用 `TryUpdateModel` 时，模型绑定系统就用用户更新记录时指定的值，更新 `Customer` 模型的值。



执行更新操作时，要设置控件的 `UpdateMethod` 属性。同样，也可以设置 `DeleteMethod` 属性，对记录执行删除操作。

1. 在模型绑定系统中验证记录

在大多数情况下，都希望根据某个自定义的业务逻辑验证记录，再把记录保存到数据库中。模型绑定很容易实现这个功能，获得更简洁的实现方式。如果对 `ObjectDataSource` 使用业务验证规则，就无法方便地把业务层的异常传播回页面。最终会在业务层抛出异常，在控件中捕获，再显示一条自定义错误消息。这样代码就比较繁琐。

模型绑定系统的优点是可以清楚地把绑定和验证分隔开。这意味着可以非常方便地使用自定义验证功能。从模型绑定系统中抛出的验证错误通过验证汇总来显示，这样就很容易自定义 UI，帮助维护简洁的代码。

在 ASP.NET Web 窗体中启用模型绑定的一个优点是，可以插入不同的验证机制。例如，可以使用数据注释特性来添加特性，以表示模型的一些元数据。模型绑定系统可以在验证过程中使用该元数据。程序清单 9-8 演示了如何告诉模型绑定系统需要 `FirstName` 属性。

程序清单 9-8 添加数据注释

```
[Required()]
public string FirstName {get; set;}
```

在这个示例中，如果没有给 `FirstName` 属性输入值，对 `TryUpdateModel` 方法的调用就会返回 `false`。如果页面上有一个 `ValidationSummary` 控件，就会在验证汇总中显示错误消息。程序清单 9-9 说明了如何配置 `ValidationSummary` 控件，以显示模型绑定系统中的错误。检查验证错误的另一种方式是检查页面的 `ModelState` 属性，在模型绑定期间，如果出现错误，这个属性就由模型绑定系统填充。

程序清单 9-9 配置 `ValidationSummary` 控件以显示模型绑定系统中的错误

```
<asp:ValidationSummary runat="server" ShowModelStateErrors="true" />
```

2. 将业务逻辑与页面分开

前面一直在页面上调用模型绑定方法。尽管这很常见，但会使页面隐藏文件包含许多代码。页面必须处理 UI 逻辑和业务逻辑。上一章介绍 `ObjectDataSource` 时，业务逻辑放在单独的类中，页面代码只处理 UI。程序清单 9-10 演示了如何在模型绑定的过程中实现这个功能。必须告诉模型绑定

系统在何处加载模型绑定方法。为此，需要重写数据绑定控件的 `OnCallingDataMethods` 方法。接着模型绑定系统实例化 `repository` 类，查找该类中的方法。这种方式能使应用程序逻辑简洁得多，也更容易测试。

程序清单 9-10 将业务逻辑与页面分隔开

```
protected void GridView1_CallingDataMethods(object sender, CallingDataMethodsEventArgs e)
{
    e.DataMethodsObject = new CustomerRepository();
}
```

9.2 使用强类型化的控件

ASP.NET 2.0 Web 窗体引入了模板控件的概念。模板允许自定义服务器控件提交的标记，一般与数据绑定表达式一起使用。本节介绍 ASP.NET 4.5 中使数据绑定和 HTML 编码更容易的改进之处。

在 ASP.NET 2.0 中，单向数据绑定用 `Eval` 和 `Bind` 辅助方法来完成。这些辅助方法对数据进行后期绑定。程序清单 9-11 演示了 `Eval` 辅助方法的使用。

程序清单 9-11 数据绑定辅助方法

```
<asp:FormView ID="editCustomer" runat="server">
    <ItemTemplate>
        <div>
            First Name:<%# Eval("FirstName") %>
        </div>
    </ItemTemplate>
</asp:FormView>
```

这种方式的一个缺点是，因为这些表达式是后期绑定的，所以必须传递一个字符串来表示属性名。这说明，无法获得对成员名的 `IntelliSense`，以支持代码导航(例如 `Go To Definition`)或编译期间的检查支持。

控件是强类型的，就意味着通过新属性 `ItemType`，就可以声明数据要绑定到哪种类型的数据。设置这个属性时，控件有绑定表达式的两个新属性——`Item` 和 `BindItem`。

`Item` 等价于 `Eval` 方法，`BindItem` 等价于 `Bind` 方法。程序清单 9-12 是一个强类型化的控件的例子。

程序清单 9-12 强类型化的控件

```
<asp:FormView ID="editCustomer" runat="server" ItemType="Customer"
    SelectMethod="SelectCustomer" >
    <ItemTemplate>
        <div>
            First Name:<%# Item.FirstName %>
        </div>
    </ItemTemplate>
</asp:FormView>
```

如果在 Visual Studio 中输入 BindItem, 如程序清单 9-12 所示, 就会获得 Customer 模型中所有属性的 IntelliSense 支持, 如图 9-1 所示。如果在输入时出错, 就会在 Visual Studio 中显示错误。

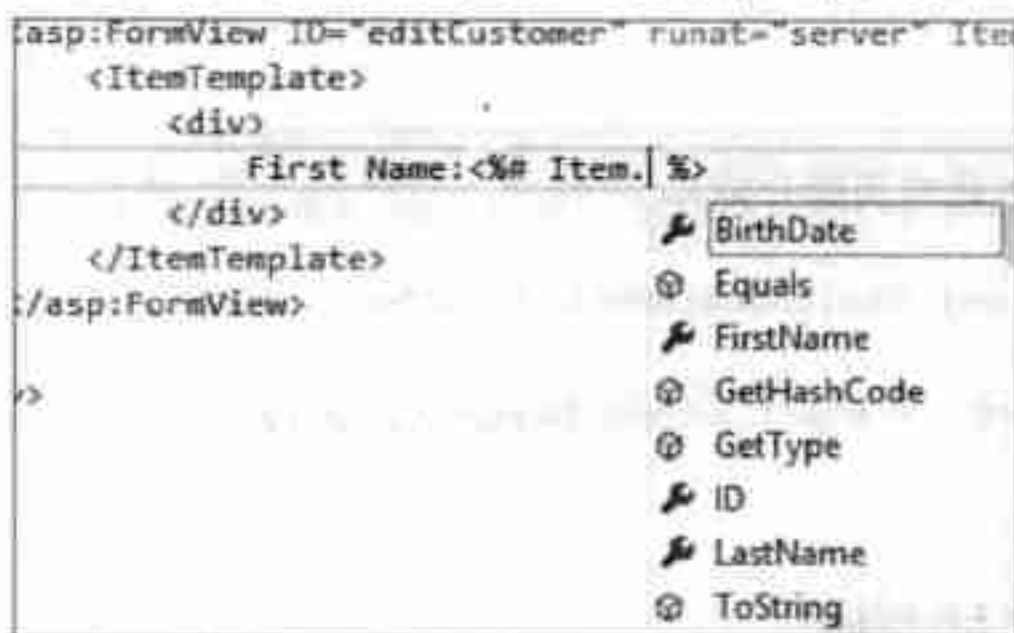


图 9-1

9.3 扩展模型绑定

ASP.NET Web Forms 是很强大的 Web 开发架构, 自定义和扩展架构是比较痛苦的。但是, 构建模型绑定系统的原则是能以更简便的方式自定义和扩展该系统, 以匹配不同的开发场合。本节将学习如何自定义值提供程序和模型绑定器。最后介绍如何扩展 ModelDataSource, 以控制模型绑定系统的工作过程。

9.3.1 定制的值提供程序

前面介绍了值提供程序特性。它们对指定模型绑定系统从何处提取值提供了基本支持。但是, 有时并不知道值来自哪里, 或者希望编写应用程序, 利用反馈机制——允许值来自 Form 集合或 QueryString 集合。此时, 就可以编写自己的值提供程序特性和值提供程序。程序清单 9-13 演示了这种值提供程序特性和值提供程序。

程序清单 9-13 实现自定义的值提供程序和特性

```
public class AggregateValueProvider : IValueProvider, IUnvalidatedValueProvider
{
    private readonly List<IUnvalidatedValueProvider> _valueProviders = new
        List<IUnvalidatedValueProvider>();
    public AggregateValueProvider(ModelBindingExecutionContext
        modelBindingExecutionContext)
    {
        _valueProviders.Add(new FormValueProvider(modelBindingExecutionContext))
        _valueProviders.Add(new QueryStringValueProvider(modelBindingExecutionContext));
    }
    public bool ContainsPrefix(string prefix)
    {
        return _valueProviders.Any(vp => vp.ContainsPrefix(prefix));
    }
    public ValueProviderResult GetValue(string key)
    {
        return GetValue(key, false);
    }
}
```

```

    public ValueProviderResult GetValue(string key, bool skipValidation)
    {
        return _valueProviders.Select(vp => vp.GetValue(key, skipValidation))
            .LastOrDefault();
    }
}
public class AggregateValueAttribute : ValueProviderSourceAttribute
{
    public override IValueProvider GetValueProvider(ModelBindingExecutionContext
        modelBindingExecutionContext)
    {
        return new AggregateValueProvider(modelBindingExecutionContext);
    }
}

```

这个程序清单显示了值提供程序和使用值提供程序的值提供程序特性。值提供程序把 Form 和 QueryString 值提供程序作为源。这表示模型绑定系统调用这个自定义的值提供程序时，自定义的值提供程序会首先检查 Form 集合来提取值，如果没有找到值，就检查 QueryString 集合。程序清单 9-14 演示了如何在应用程序中使用这个自定义的值提供程序。

程序清单 9-14 使用自定义的值提供程序

```

CustomerContext _context = new CustomerContext();
public IEnumerable<Customer> SelectCustomers([AggregateValue] int? id)
{
    if(id.HasValue)
        return _context.Customer.Where(c => c.ID == id).AsEnumerable();
    else
        return _context.Customer.AsEnumerable();
}

```

如这个程序清单所示，可以指定为值提供程序定义的定制值提供程序特性，以使用定制的值提供程序。AggregateValue 特性的工作方式与其他值提供程序特性(如 QueryString)相同。模型绑定系统调用 AggregateValue 特性以绑定 id 参数的值。AggregateValue 特性调用 AggregateValueProvider，以获得 id 参数的值。

9.3.2 定制模型绑定器

在 ASP.NET 4.5 中，模型绑定的实现提供了对不同数据类型的支持。尽管基本的实现方式考虑了常用数据类型的绑定，但有时模型绑定系统不能绑定特定的数据类型。此时，可以编写自己的模型绑定器，再插入模型绑定系统。只要模型绑定尝试绑定这种类型的模型，就会调用定制的模型绑定器，提取并保存值，填充模型。

例如，如果需要修改定制的模型绑定器把数据绑定到模型(例如 DateTime 类型)的行为方式，就应以另一种本地格式分别存储日期和时间。为此，需要实现定制的模型绑定器。因为 Web Forms 模型绑定系统的实现基于可扩展的模型绑定，所以此修改很容易完成。程序清单 9-15 演示了模型绑定器的定制实现。

要实现定制的模型绑定器，必须实现提供程序和绑定器。

- 提供程序：由模型绑定系统调用，检查提供程序是否能处理特定的类型。在程序清单 9-15 和 9-16 中，处理了 DateTime 类型。调用提供程序时，如果提供程序可以处理 DateTime 类型，提供程序就调用绑定器。
- 绑定器：负责解析来自模型绑定系统的值，并填充模型。

程序清单 9-15 实现定制模型绑定器

```
public class MyDateTimeBinder : IModelBinder
{
    public bool BindModel(ModelBindingExecutionContext modelBindingExecutionContext,
        ModelBindingContext bindingContext)
    {
        var valueProviderResult = bindingContext.ValueProvider.GetValue
            (bindingContext.ModelName);
        var inputdate = valueProviderResult != null ? valueProviderResult.AttemptedValue
            : null;
        DateTime dt = new DateTime();
        bool success = DateTime.TryParse(inputdate, CultureInfo.GetCultureInfo("en-GB"),
            DateTimeStyles.None, out dt);
        bindingContext.Model = dt;
        return bindingContext.Model != null;
    }
}

public class MyDateTimeProvider : System.Web.ModelBinding.ModelBinderProvider
{
    public override IModelBinder GetBinder(ModelBindingExecutionContext
        modelBindingExecutionContext, ModelBindingContext bindingContext)
    {
        if(bindingContext.ModelType == typeof(DateTime))
            return new MyDateTimeBinder();
        return null;
    }
}
```

模型绑定系统尝试把数据绑定到模型时，会检查已注册的提供程序列表，确定哪个提供程序可以找到绑定器，以绑定特定类型的值。注册的提供程序会按顺序调用。如果没有实现通用的解决方案，也不关心找不到特定类型的绑定器时出现的故障，就可以把自己的提供程序注册为第一个。程序清单 9-16 演示了如何在应用程序中注册模型绑定器。

程序清单 9-16 注册定制模型绑定器

```
System.Web.ModelBinding.ModelBinderProviders.Providers.Insert(0, new MyDateTimeProvider());
```

9.3.3 定制的 ModelDataSource

这种实现方式的核心是，Web Forms 中的模型绑定系统基于数据控件。该实现方式使用可扩展的模型绑定和控件体系结构。模型绑定实现为数据源 ModelDataSource，它实现了 IDataSource。这与其他 ObjectDataSource 对象的模式相同。例如，还有一个 ModelDataSourceView 对象，它包含的逻辑可以执行选择、插入、更新和删除数据控件中的调用等操作。

这意味着这种实现是完全可扩展的。如果希望重写选择调用的方式，就可以编写自己的

ModelDataSource 和 ModelDataSourceView 对象，把它们插入模型绑定系统。如果希望扩展模型绑定系统，以便实现主从绑定，就可以采用这种方式。

程序清单 9-17 说明了如何实现定制的 ModelDataSource 和 ModelDataSourceView 对象。这种定制的实现方式只返回表中的前三行，但很容易为更复杂的情形扩展该方式。

程序清单 9-17 实现定制的 ModelDataSource

```
public class MyModelView : ModelDataSourceView
{
    private readonly MyModelDataSource _owner;
    public MyModelView(MyModelDataSource owner)
        : base(owner)
    {
        _owner = owner;
    }
    protected override IEnumerable ExecuteSelect(DataSourceSelectArguments arguments)
    {
        CustomerContext _context = new CustomerContext();
        return _context.Customer.Take(3).AsEnumerable();
        //return _context.Customer.Distinct<Customer>().AsEnumerable();
    }
}

public class MyModelDataSource : ModelDataSource
{
    private MyModelView _view;
    public MyModelDataSource(Control dataControl)
        : base(dataControl)
    {
    }
    public override ModelDataSourceView View
    {
        get
        {
            if(_view == null)
            {
                _view = new MyModelView(this);
            }
            return _view;
        }
    }
}
```

程序清单 9-18 说明了如何在应用程序中使用这个定制的 ModelDataSource。在数据控件中，可以重写事件 OnCreatingModelDataSource，以重写该控件应使用的 ModelDataSource。

程序清单 9-18 调用定制的 ModelDataSource

```
protected void GridView1_CreatingModelDataSource(object sender,
    CreatingModelDataSourceEventArgs e)
{
    e.ModelDataSource = new CS.MyModelDataSource((GridView)sender);
}
```

9.4 本章小结

模型绑定系统是 Web Forms 数据绑定的下一步。它使用可扩展模型绑定系统的强大功能和灵活性，把它与数据控件的功能结合起来，使应用程序的开发更简单。它还使实现代码更简洁，支持单元测试和 IOC 容器等范例。

数据绑定使数据访问更关注代码，并允许在 ASP.NET 中重用数据注释特性。使用值提供程序更容易完成过滤任务。本章最后介绍了如何定制和扩展模型绑定系统，完成定制的绑定和选择任务，这使得该架构非常容易采用。

第 10 章

使用 LINQ 查询

本章要点

- 探讨 LINQ 查询的各种类型
- 了解传统查询方法的局限性
- 使用 LINQ 简化查询操作

.NET 3.5 引入了一种新技术,称为 LINQ(Language Integrated Query)。之后 LINQ 就成为.NET 开发的一个重要技术。在.NET 4.0 和.NET 4.5 中,Microsoft 继续更新 LINQ,提高其性能。LINQ 填补了传统.NET 语言和查询语言(如 SQL)之间的空白,传统.NET 语言提供了强类型化和完整的面向对象开发功能,而查询语言的语法是专门为查询操作设计的。将 LINQ 引入.NET 后,查询在.NET 中就成为极为有用的概念,无论正在讨论的是对象、XML 还是数据库查询。

LINQ 包含 3 种基本类型的查询(这可以扩展到其他数据源):LINQ to Objects、LINQ to XML(或 XLINQ)和用于数据库环境的 LINQ(如 LINQ to SQL 或 LINQ to Entities)。每种查询都提供了特殊的功能,专门用于查询某种数据源。

本章将介绍这 3 种 LINQ,包括 LINQ to SQL 和 LINQ to Entities,以及它们如何简化查询操作。本章还将讨论.NET CLR 中用于创建 LINQ 查询的一些语言功能,以及 Visual Studio 提供的用于支持 LINQ 的工具。



本章主要介绍.NET Framework 包含的 LINQ 功能,但 LINQ 是高度可扩展的,可以用于在任何数据源上创建查询架构。虽然有关如何实现自己的 LINQ 提供程序的讨论超出了本章的范围,但 LINQ 有许多实现方式,可以查询各种数据存储,例如 LDAP、SharePoint 甚至 Amazon.com。来自 Oakleaf Systems 的 Roger Jennings 在他的博客上维护着一个第三方 LINQ 提供程序的列表,网址是 <http://oakleafblog.blogspot.com/2007/03/third-party-linq-providers.html>。

10.1 LINQ to Objects

LINQ 的第一种类型(也是最基本的一种类型)是 LINQ to Objects。它允许对任意可枚举的对象(实现了 `IEnumerable` 接口的对象)执行复杂的查询操作。创建可以查询或排序的可枚举对象并不是 .NET 的新增功能,但在 .NET 3.5 版本之前执行该操作通常需要编写大量的代码。这些代码常常比较复杂,其他开发人员很难阅读和理解,也很难维护。

10.1.1 传统的查询方法

为了真正理解 LINQ 如何提高查询集合的能力,必须先理解没有 LINQ 时如何查询集合。为此,首先查看如何创建一个简单的查询,它包含一个组,并且不使用 LINQ 进行排序。程序清单 10-1 是简单的 `Movie` 类,该类用作这些示例的基础。

程序清单 10-1 基本的 `Movie` 类

```
using System;

public class Movie
{
    public string Title{get; set;}
    public string Director{get; set;}
    public int Genre{get; set;}
    public int RunTime{get; set;}
    public DateTime ReleaseDate{get; set;}
}
```

有了这样一个可使用的基类后,下面查看如何使用它。程序清单 10-2 演示了如何在 ASP.NET 页面上创建 `Movie` 对象的一个简单分类列表,再把该列表绑定到 `GridView` 控件。`GridView` 控件显示了 `Movie` 类所有公有属性的值。请把日期的格式改为本地格式,以使这个例子可以运行。

程序清单 10-2 生成 `Movie` 对象列表并绑定到 `GridView` 控件

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Collections.Generic" %>

<script runat="server">

    protected void Page_Load(object sender, EventArgs e)
    {
        var movies = GetMovies();

        this.GridView1.DataSource = movies;
        this.GridView1.DataBind();
    }

    public List<Movie> GetMovies()
    {
        return new List<Movie> {
            new Movie { Title="Shrek", Director="Andrew Adamson", Genre=0,
```

```

        ReleaseDate=DateTime.Parse("5/16/2001"), RunTime=89 },
        new Movie { Title="Fletch", Director="Michael Ritchie", Genre=0,
        ReleaseDate=DateTime.Parse("5/31/1985"), RunTime=96 },
        new Movie { Title="Casablanca", Director="Michael Curtiz", Genre=1,
        ReleaseDate=DateTime.Parse("1/1/1942"), RunTime=102 },
        new Movie { Title="Batman", Director="Tim Burton", Genre=1,
        ReleaseDate=DateTime.Parse("6/23/1989"), RunTime=126 }
    };
}
</script>
<html>
<head runat="server">
    <title>My Favorite Movies</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:GridView ID="GridView1" runat="server">
            </asp:GridView>
        </div>
    </form>
</body>

</html>

```

运行这个示例，就会生成一个典型的 ASP.NET Web 页面，其中包含一个简单的网格，该网格显示了所有的 Movie 数据。

现在，如果要开始在该电影列表上执行查询，会出现什么情况呢？例如，要过滤这些数据，仅显示某种类型的电影。程序清单 10-3 是实现此过滤的一般方式。

程序清单 10-3 过滤 Movie 对象列表

```

protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();

    var query = new List<Movie>();
    foreach(var m in movies)
    {
        if(m.Genre == 0) query.Add(m);
    }

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}

```

如这个例子所示，要过滤数据以使页面仅显示某种类型的电影，只需创建一个新的临时集合，并且使用 foreach 循环遍历数据即可。

这种技术似乎很简单，但仍然需要定义指定的操作(查找出某种类型的所有电影)，并明确定义如何完成该操作(使用临时集合和 foreach 循环)。另外，当需要执行更复杂的查询(如涉及分组和排序的查询)时，又该如何实现呢？代码的复杂度将急剧增加，如程序清单 10-4 所示。

程序清单 10-4 对 Movie 对象列表进行分组和排序

```

public class Grouping
{
    public int Genre{get; set;}
    public int MovieCount{get; set;}
}

protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();

    Dictionary<int, Grouping> groups = new Dictionary<int, Grouping>();
    foreach(Movie m in movies)
    {
        if(!groups.ContainsKey(m.Genre))
        {
            groups[m.Genre] = new Grouping { Genre = m.Genre, MovieCount = 0 };
        }
        groups[m.Genre].MovieCount++;
    }

    List<Grouping> results = new List<Grouping>(groups.Values);
    results.Sort(delegate(Grouping x, Grouping y)
    {
        return
            x.MovieCount > y.MovieCount ? -1 :
            x.MovieCount < y.MovieCount ? 1 :
            0;
    });
    this.GridView1.DataSource = results;
    this.GridView1.DataBind();
}

```

要把 Movie 数据分组到各类型中，计算出每种类型的电影有多少部，这需要添加一个新类，创建一个 Dictionary，并实现一个委托。这个看似简单的任务需要执行上述相当复杂的操作，不仅要明确定义指定的操作，还要明确指定如何实现这些操作。

另外，因为代码的复杂度急剧增加，所以很难确定代码正在做什么。例如，如果要在已有的应用程序中修改自己不熟悉的代码，该怎么办？需要多长时间确定代码在做什么？

10.1.2 使用 LINQ 替代传统的查询

创建 LINQ 的目的是克服前面讨论的查询对象集合的许多缺点。LINQ 不需要明确定义如何执行查询，而允许在比较抽象的级别上定义查询。只要定义了查询要返回的内容，.NET 及其编译器就会确定查询如何运行的细节。

前面介绍了不使用 LINQ 如何查询对象集合。本节则讨论 LINQ，查看如何使用它大大简化这些查询，并论述其他类型的查询。本节中的例子首先简单修改 10.1.1 节的例子，说明 LINQ 如何使同一项任务变得更容易完成。

在开始学习本节内容之前，需要理解 LINQ 是 .NET Framework 的扩展，因此它自己的一组程序集是独立的。LINQ 的基本功能位于 System.Core.dll 程序集中。这个程序集没有替代已有的 .NET

Framework 功能，而是增加了其功能。另外，在默认情况下，Visual Studio 中的项目包含对这个程序集的引用，因此在开始新的 ASP.NET Web 项目时就可以使用 LINQ。

1. 基本 LINQ 查询和投射

程序清单 10-2 中的基本示例只是生成了电影的分类列表，并把该列表绑定到 GridView 控件。程序清单 10-5 说明了如何修改代码，以使用 LINQ 查询电影列表，并把结果集绑定到 GridView 控件。

程序清单 10-5 使用 LINQ 创建查询

```
protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();

    var query = from m in movies
                select m;

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

如果分解这个代码示例的结构，就会发现其中有 3 个基本操作。首先，代码使用 GetMovies 方法获得泛型集合 List<Movie>。

之后，代码使用一个非常简单的 LINQ 查询，从电影集合中选择出所有的 Movie 对象。注意，这个 LINQ 查询在查询语句中利用了语言关键字 from 和 select。这些新增的语法是 .NET 语言中极为重要的成员，因此 Visual Studio 可以提供强类型化检查和 IntelliSense 等开发辅助功能，更便于查找和修改代码中的问题。

该查询还定义了新变量 m。在查询中以两种方式使用这个变量。首先，通过在变量 m 的 from 语句中定义它，告诉 LINQ 让 m 表示单个集合项，在本例中是 Movie 对象。告诉 LINQ 可以使它理解我们要查询的对象结构，如本章后面所述，这样做还可以利用 IntelliSense 帮助创建查询。

在查询中第二次使用 m 是在 select 语句中，这告诉 LINQ 输出匹配 m 的结构的投射，投射是把对象转换为一种新形式的操作，这种新形式常常只包含达到特定目的所需的属性。在本例中表示 LINQ 创建了匹配 Movie 对象结构的投射。

只要使用新关键字和 select 运算符明确定义要从查询中返回的字段，就可以方便地创建自己的定制投射，如程序清单 10-6 所示。

程序清单 10-6 使用 LINQ 创建定制投射

```
protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();

    var query = from m in movies
                select new { m.Title, m.Genre };
}
```

```

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}

```

注意，这里不是简单地选择 `m`，而是定义了一个新的投射，其中只包含 `Title` 和 `Genre` 值。也可以明确定义字段名称，指定要在结果集中出现的对象。例如，要给 `Title` 和 `Genre` 字段指定更明确的名称，以便更完整地描述它们的内容，就可以使用 LINQ 来轻易地完成这项任务，如程序清单 10-7 所示。

程序清单 10-7 创建定制投射的字段名称

```

protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();

    var query = from m in movies
                select new { MovieTitle = m.Title, MovieGenre = m.Genre };

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}

```

这个示例把结果集中出现的字段明确定义为 `MovieTitle` 和 `MovieGenre`。如图 10-1 所示，由于执行了这项改动，`GridView` 中的列标题变得更加匹配。最后，代码把 `GridView` 控件绑定到 LINQ 查询返回的 `Movie` 对象的可枚举列表。

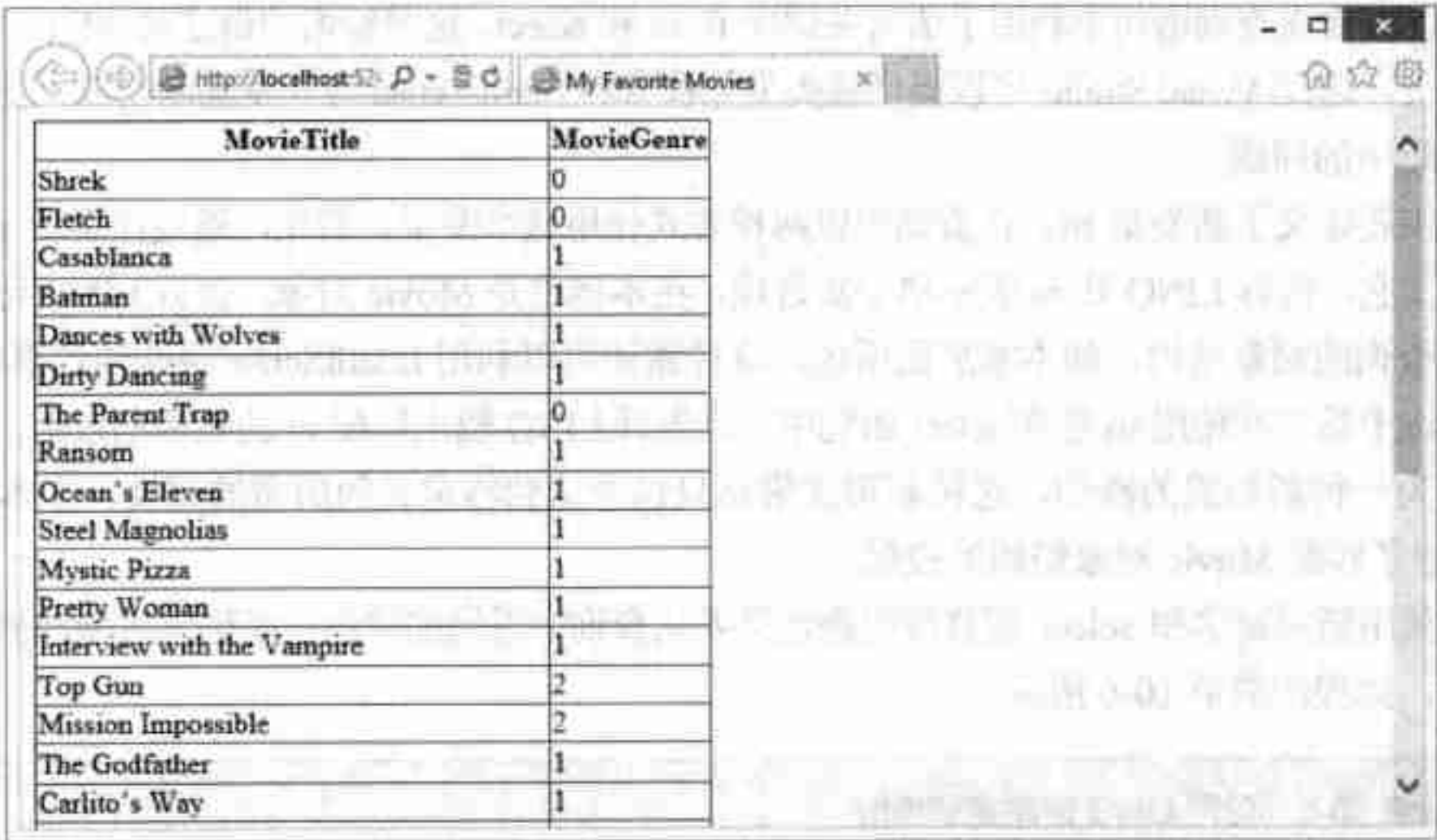


图 10-1

如图 10-2 所示，运行程序清单 10-5 中的代码，会得到与程序清单 10-2 生成的相同的 Web 页面。



Title	Director	Genre	RunTime	ReleaseDate
Shrek	Andrew Adamson	0	89	5/16/2001 12:00:00 AM
Fletch	Michael Ritchie	0	96	5/31/1985 12:00:00 AM
Casablanca	Michael Curtiz	1	102	1/1/1942 12:00:00 AM
Batman	Tim Burton	1	126	6/23/1989 12:00:00 AM
Dances with Wolves	Kevin Costner	1	180	11/21/1990 12:00:00 AM
Dirty Dancing	Emile Ardolino	1	100	8/21/1987 12:00:00 AM
The Parent Trap	Nancy Meyers	0	127	7/29/1998 12:00:00 AM
Ransom	Ron Howard	1	121	11/8/1996 12:00:00 AM
Ocean's Eleven	Steven Soderbergh	1	116	12/7/2001 12:00:00 AM
Steel Magnolias	Herbert Ross	1	117	11/15/1989 12:00:00 AM
Mystic Pizza	Donald Petrie	1	104	10/21/1988 12:00:00 AM
Pretty Woman	Garry Marshall	1	119	3/23/1990 12:00:00 AM
Interview with the Vampire	Neil Jordan	1	123	11/11/1994 12:00:00 AM
Top Gun	Tony Scott	2	110	5/16/1986 12:00:00 AM
Mission Impossible	Brian De Palma	2	110	5/22/1996 12:00:00 AM
The Godfather	Francis Ford Coppola	1	175	3/24/1972 12:00:00 AM
Carlito's Way	Brian De Palma	1	144	11/10/1993 12:00:00 AM

图 10-2

LINQ 还可以使用 `orderby` 语句对结果排序。与 SQL 一样，也可以选择按升序或降序对结果排序，如程序清单 10-8 所示。

程序清单 10-8 使用 LINQ 控制对象的排序

```
protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();

    var query = from m in movies
                orderby m.Title descending
                select new { MovieTitle = m.Title, MovieGenre = m.Genre };

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

LINQ 语法的另一个优秀功能是极大地提高了代码的可读性和可理解性。LINQ 允许用户简单地表示出查询的意图，告诉编译器希望代码做什么，这样编译器就会很好地去执行。



利用这些关键字，可以使用简单的、类似 SQL 的简洁语法来构建 LINQ 查询，但不能保证会有奇迹发生。这些关键字实际上映射到 `movies` 集合的扩展方法。使用这些扩展方法可以直接编写出相同的 LINQ 查询，如下所示：

```
var query = movies.Select(m => m);
```

编译器会在编译过程中转换关键字语法。查看 `List<T>` 的对象结构，会发现其中没有 `Select` 方法，那么 `Select` 方法是如何添加到泛型集合 `List<Movies>` 中的呢？LINQ 使用扩展方法添加了 `Select` 方法，以及用于基类 `Enumerable` 的许多其他方法。因此，LINQ 利用这些方法扩展实现了 `IEnumerable`

的任意类。在 Visual Studio 中右击 Select 方法，从上下文菜单中选择 View Definition 选项，就可以看到 LINQ 添加的所有方法。Visual Studio 会显示 LINQ 的 Enumerable 类的元数据。如果滚动这个类，不仅会看到 Select 方法，还会看到其他方法，如 Where、Count、Min、Max，以及 LINQ 自动添加到实现了 IEnumerable 接口的对象中的许多其他方法。

2. 使用定制的比较器对结果排序

使用 orderby 语句或 OrderBy 和 OrderByDescending 扩展方法，要排序类型的默认比较器就会比较值。有时需要更多地控制结果的顺序，这超出了默认比较器能提供的功能。此时，可以创建定制的比较器，使用 OrderBy 或 OrderByDescending 扩展方法的重载版本，把它传递给排序操作。为了创建定制的比较器，需要创建一个继承了 IComparer<> 的类。使用程序清单 10-2 中的电影数据，如果按导演排序，结果就按导演的名排序，因为字符串包含名和姓。如果结果要按姓排序，就需要创建如程序清单 10-9 所示的定制比较器。

程序清单 10-9 按导演的名排序的定制比较器

```
using System.Collections.Generic;
public class LastNameComparer : IComparer<string>
{
    public int Compare(string x, string y)
    {
        var director1LastName = x.Substring(x.LastIndexOf(' '));
        var director2LastName = y.Substring(y.LastIndexOf(' '));
        return director1LastName.CompareTo(director2LastName);
    }
}
```

要在 OrderBy 或 OrderByDescending 扩展方法中使用这个新的比较器，可以把程序清单 10-8 改为程序清单 10-10。

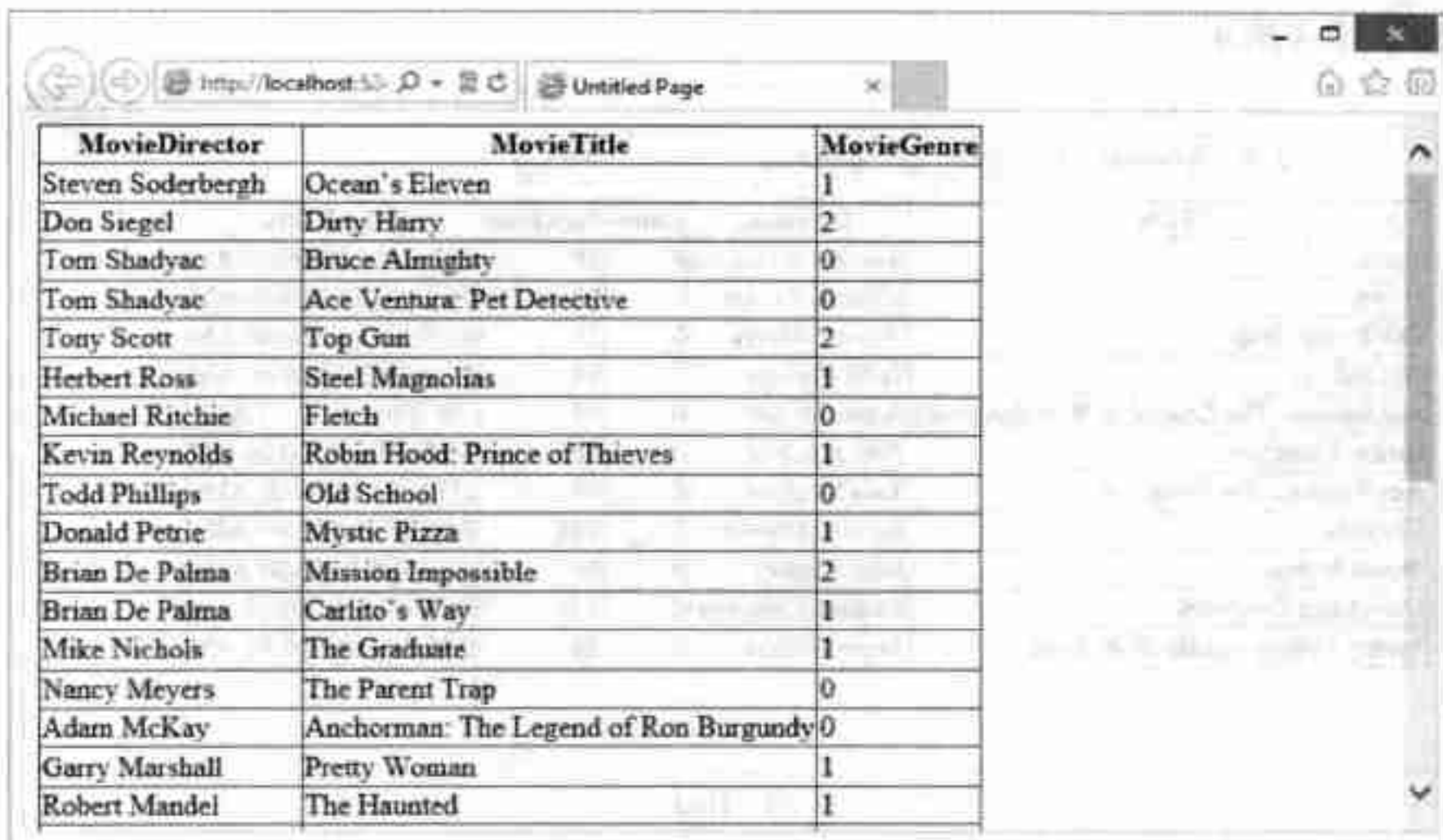
程序清单 10-10 使用定制比较器来排序

```
protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();

    var query = movies.OrderByDescending(m => m.Director, new LastNameComparer())
        .Select(m => new { MovieDirector = m.Director, MovieTitle = m.Title,
            MovieGenre = m.Genre });

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

图 10-3 显示了运行程序清单 10-10 的结果。导演已被添加到结果中，并按导演的姓降序排序。



MovieDirector	MovieTitle	MovieGenre
Steven Soderbergh	Ocean's Eleven	1
Don Siegel	Duty Harry	2
Tom Shadyac	Bruce Almighty	0
Tom Shadyac	Ace Ventura: Pet Detective	0
Tony Scott	Top Gun	2
Herbert Ross	Steel Magnolias	1
Michael Ritchie	Fletch	0
Kevin Reynolds	Robin Hood: Prince of Thieves	1
Todd Phillips	Old School	0
Donald Petrie	Mystic Pizza	1
Brian De Palma	Mission Impossible	2
Brian De Palma	Carlito's Way	1
Mike Nichols	The Graduate	1
Nancy Meyers	The Parent Trap	0
Adam McKay	Anchorman: The Legend of Ron Burgundy	0
Garry Marshall	Pretty Woman	1
Robert Mandel	The Haunted	1

图 10-3

延迟执行

LINQ 的一项有趣特性是其延迟执行的行为。这表示, 尽管可以在代码的某个位置执行查询语句, 但 LINQ 非常智能, 可以延迟执行查询直到遇到该查询。例如, 在前面的示例中, 尽管 LINQ 查询放在 GridView 控件的绑定之前, 但 LINQ 不会执行我们定义的查询, 直到 GridView 控件开始遍历查询结果时才执行该查询。

延迟查询的重要优点是可以在 LINQ 查询执行前修改。LINQ 查询返回一个 IQueryable 对象, IQueryable 继承了 IEnumerable, 但数据要在实际迭代时才可用。尝试迭代结果会让 LINQ 执行查询。也有方法可以使 LINQ 查询立即执行, 例如以 To 开头的方法(如 ToList)和返回结果的方法(如 Count)。

因为查询要在迭代结果时才执行, 所以可以在执行查询之前, 给它添加额外的过滤器。

3. 使用 LINQ 过滤数据

LINQ 还可以使用熟悉的、类似 SQL 的 where 语法来添加查询过滤器。修改程序清单 10-3 中的 LINQ 查询, 通过在查询中添加一条 where 子句来添加过滤器, 如程序清单 10-11 所示。

程序清单 10-11 给 LINQ 查询添加过滤器

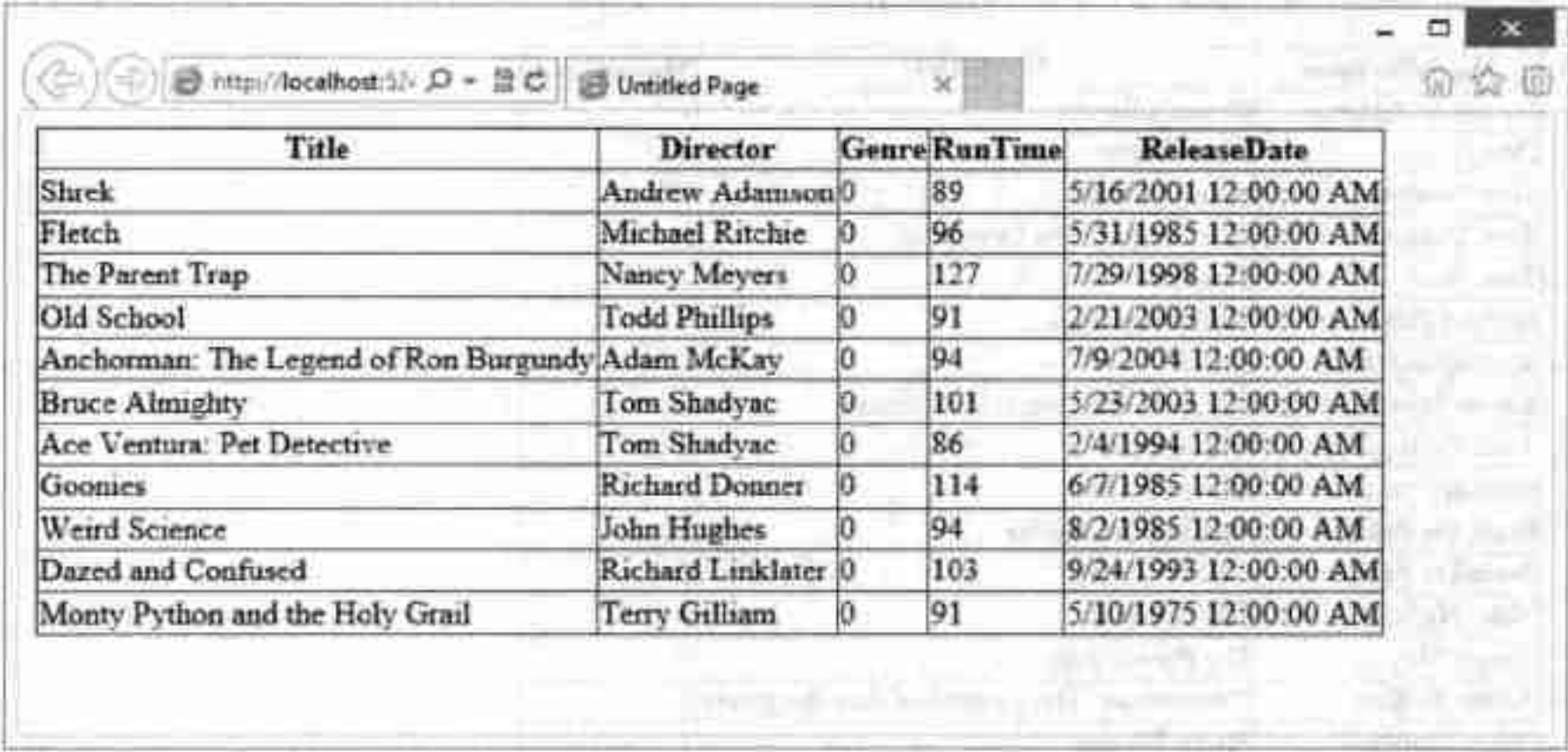
```
protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();

    var query = from m in movies
                where m.Genre==0
                select m;

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

把这个简单的 where 子句添加到 LINQ 查询中, 该查询返回的结果就会被过滤, 仅显示 0 类型

的电影，如图 10-4 所示。



The screenshot shows a web browser window with the address bar at 'http://localhost:52...' and the title 'Untitled Page'. The main content is a table with the following data:

Title	Director	Genre	RunTime	ReleaseDate
Shrek	Andrew Adamson	0	89	5/16/2001 12:00:00 AM
Fletch	Michael Ritchie	0	96	5/31/1985 12:00:00 AM
The Parent Trap	Nancy Meyers	0	127	7/29/1998 12:00:00 AM
Old School	Todd Phillips	0	91	2/21/2003 12:00:00 AM
Anchorman: The Legend of Ron Burgundy	Adam McKay	0	94	7/9/2004 12:00:00 AM
Bruce Almighty	Tom Shadyac	0	101	5/23/2003 12:00:00 AM
Ace Ventura: Pet Detective	Tom Shadyac	0	86	2/4/1994 12:00:00 AM
Goonies	Richard Donner	0	114	6/7/1985 12:00:00 AM
Weird Science	John Hughes	0	94	8/2/1985 12:00:00 AM
Dazed and Confused	Richard Linklater	0	103	9/24/1993 12:00:00 AM
Monty Python and the Holy Grail	Terry Gilliam	0	91	5/10/1975 12:00:00 AM

图 10-4

还要注意，LINQ 是.NET 的重要成员，因此在构建 LINQ 查询时，Visual Studio 可以提供非常优秀的编码方式。在这个例子中，输入 where 子句时，Visual Studio 会通过 IntelliSense 提供 m 的可能参数(Movie 对象)，如图 10-5 所示。

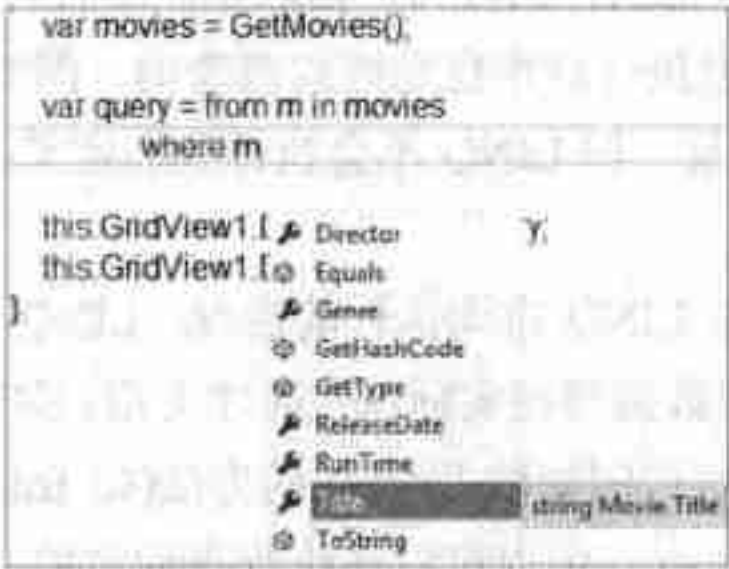


图 10-5

LINQ 中的 where 子句与 SQL 的 where 子句类似，也可以包含子查询和多条 where 子句，如程序清单 10-12 所示。

程序清单 10-12 给 LINQ 查询添加 where 子句

```
protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();

    var query = from m in movies
                where m.Genre == 0 && m.RunTime > 92
                select m;

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

在这个例子中，where 子句包含两个参数：一个限制电影类型，另一个限制电影的播放时间。

10.1.3 使用 LINQ 分组数据

LINQ 使用与 SQL 的 `group` 类似的语法，极大地简化了数据的分组。为了说明 LINQ 是如何简化该过程的，可以修改前面的程序清单 10-4 以使用 LINQ 查询，修改后的代码如程序清单 10-13 所示。

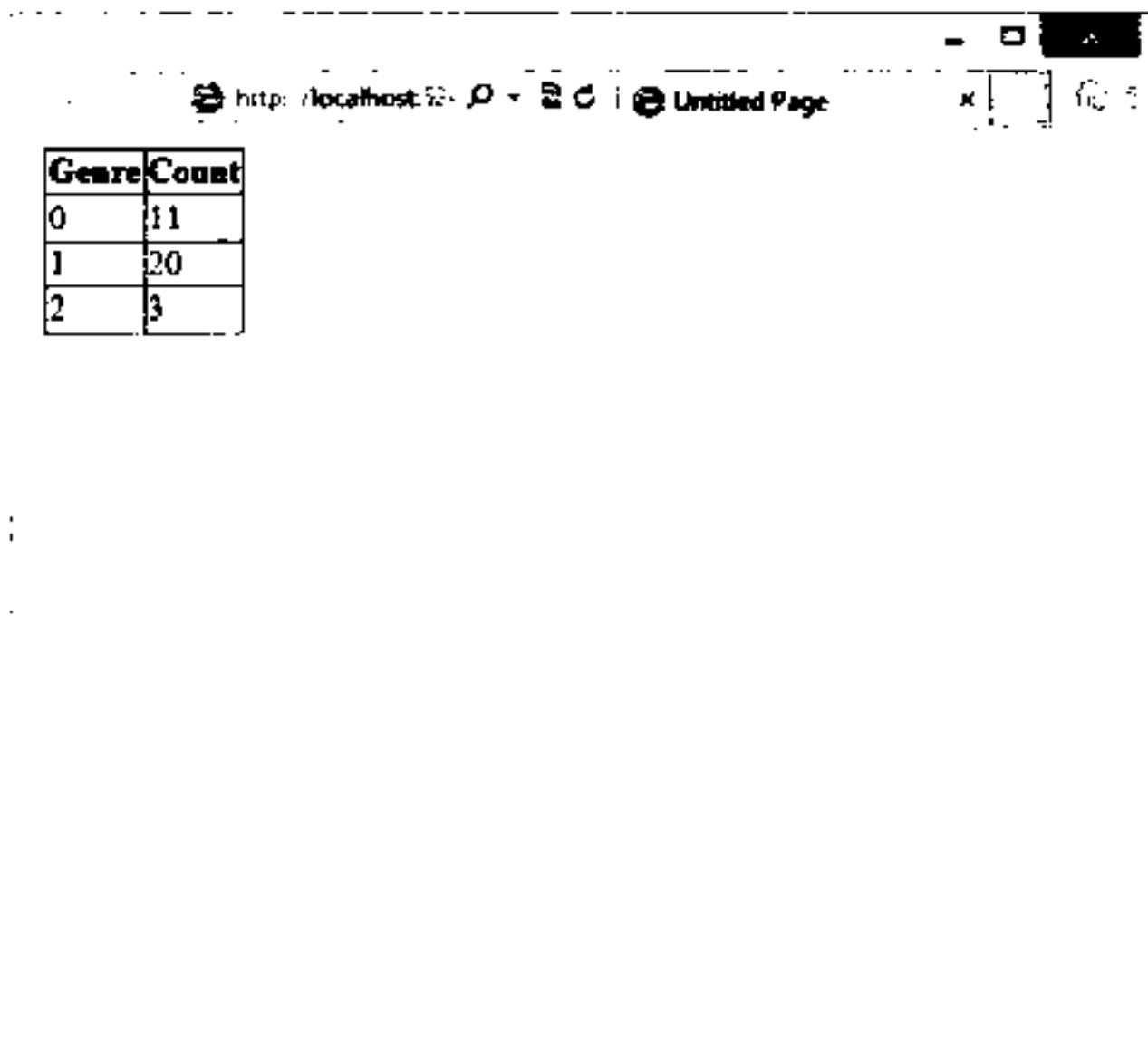
程序清单 10-13 使用 LINQ 查询对数据进行分组

```
protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();

    var query = from m in movies
                group m by m.Genre into g
                select new { Genre = g.Key, Count = g.Count() };

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

这个 LINQ 查询使用 `group` 关键字或 VB.NET 中的 `Group By` 按类型分组电影数据。另外，因为分组操作不会导致任何结果，所以查询仍然使用前面讨论的技术创建定制查询投射。这个查询的结果如图 10-6 所示。



Genre	Count
0	11
1	20
2	3

图 10-6

使用 LINQ 分组可以极大地减少需要的代码量。如果把程序清单 10-4 中执行分组操作所需的代码量与先前使用 LINQ 的程序清单相比，就可以看出代码行的数量从 18 行减少到 3 行，代码的可读性和简洁性也有所提高。

10.1.4 使用其他 LINQ 运算符

除了基本的选择、过滤和分组之外，LINQ 还包含可以在集合上执行的许多运算符。其中大多

数运算符都可以像 SQL 中的运算符那样在任意集合上使用(如 Count、Min、Max、Average 和 Sum), 如程序清单 10-14 所示。

程序清单 10-14 使用 LINQ 查询的运算符

```
protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();

    this.TotalMovies.Text = movies.Count.ToString();
    this.LongestRuntime.Text = movies.Max(m => m.RunTime).ToString();
    this.ShortestRuntime.Text = movies.Min(m => m.RunTime).ToString();
    this.AverageRuntime.Text = movies.Average(m => m.RunTime).ToString();
}
```

这个程序清单演示了 Count、Min、Max 和 Average 运算符用于 movies 集合的情况。注意, 除了 Count 运算符之外, 其他运算符都需要为方法提供要在操作中处理的特定字段, 使用 Lambda 表达式来完成该操作。

10.1.5 建立 LINQ 连接

LINQ 还允许使用熟悉的、类似 SQL 的连接语法来合并不同集合中的数据。例如, 在前面的示例数据中, 只能把类型显示为数字 ID。更好的方式是显示每个类型的名称。为此, 只需创建 Genre 类, 定义该类的属性即可, 如程序清单 10-15 所示。

程序清单 10-15 简单的 Genre 类

```
public class Genre
{
    public int ID { get; set; }
    public string Name { get; set; }
}
```

接着就可以在 Web 页面上添加 GetGenres 方法, 返回 Genre 对象列表, 如程序清单 10-16 所示。

程序清单 10-16 填充 Genres 集合

```
public List<Genre> GetGenres()
{
    return new List<Genre> {
        new Genre { ID=0, Name="Comedy" },
        new Genre { ID=1, Name="Drama" },
        new Genre { ID=2, Name="Action" }
    };
}
```

最后, 修改 Page_Load 事件, 包括 LINQ 查询, 以检索出 Genres 列表, 再使用 LINQ 将之连接到 Movies 列表, 如程序清单 10-17 所示。

程序清单 10-17 使用 LINQ 查询连接 Genre 数据和 Movie 数据

```
protected void Page_Load(object sender, EventArgs e)
```



```

{
    var movies = GetMovies();
    var genres = GetGenres();

    var query = from m in movies
                join g in genres on m.Genre equals g.ID
                select new { m.Title, Genre = g.Name };

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}

```

从这个例子中可以看出，连接语法非常简单。只要告诉 LINQ 包含 Genres 对象，再告诉 LINQ 应关联哪些字段即可。

10.1.6 使用 LINQ 分页数据

使用 LINQ 提供的 Skip 和 Take 方法，可以方便地在 Web 应用程序中包含分页逻辑。Skip 方法可以在结果集中跳过指定数量的记录。Take 方法可以指定要从结果集中返回的记录数。先调用 Skip 方法，再调用 Take 方法，就可以从结果集的指定位置返回指定数量的记录，如程序清单 10-18 所示。

程序清单 10-18 使用 LINQ 方法进行简单的分页

```

protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();
    var genres = GetGenres();

    var query = (from m in movies
                 join g in genres on m.Genre equals g.ID
                 select new { m.Title, g.Name }).Skip(10).Take(10);

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}

```

运行这些代码，结果将从列表的第 10 条记录开始，并且仅显示 10 条记录。

10.2 LINQ to XML

LINQ 的第二种类型是 LINQ to XML。顾名思义，LINQ to XML 允许使用基本的 LINQ 语法查询 XML 文档。与基本 LINQ 功能一样，.NET 的 LINQ to XML 功能也作为基本 .NET Framework 的扩展，没有改变任何已有的功能。另外，与核心 LINQ 功能一样，LINQ to XML 功能也包含在独立的程序集 System.Xml.Linq 中。

为了介绍如何使用 LINQ 查询 XML，本节也使用 10.1 节的 Movie 数据，但需要将这些数据转换为 XML。程序清单 10-19 演示了转换为简单 XML 文档的部分 Movie 数据。包含全部转换数据的 XML 文件位于本章的下载代码中。

程序清单 10-19 XML 数据的示例文件

```
<?xml version="1.0" encoding="utf-8" ?>
<Movies>
  <Movie>
    <Title>Shrek</Title>
    <Director>Andrew Adamson</Director>
    <Genre>0</Genre>
    <ReleaseDate>5/16/2001</ReleaseDate>
    <RunTime>89</RunTime>
  </Movie>
  <Movie>
    <Title>Fletch</Title>
    <Director>Michael Ritchie</Director>
    <Genre>0</Genre>
    <ReleaseDate>5/31/1985</ReleaseDate>
    <RunTime>96</RunTime>
  </Movie>
  <Movie>
    <Title>Casablanca</Title>
    <Director>Michael Curtiz</Director>
    <Genre>1</Genre>
    <ReleaseDate>1/1/1942</ReleaseDate>
    <RunTime>102</RunTime>
  </Movie>
</Movies>
```

下面开始考虑如何使用 LINQ to XML 查询 XML 文档，首先从基本查询开始。

10.2.1 一个简单的 LINQ to XML 示例

程序清单 10-20 演示了使用 LINQ to XML 的一个简单的选择查询。

程序清单 10-20 使用 LINQ 查询 XML 数据文件

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Linq" %>
<%@ Import Namespace="System.Xml.Linq" %>
<script runat="server">
  protected void Page_Load(object sender, EventArgs e)
  {
    var query = from m in
      XElement.Load(MapPath("Movies.xml")).Elements("Movie")
    select m;

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
  }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>My Favorite Movies</title>
</head>
<body>
```



```

<form id="form1" runat="server">
<div>
    <asp:GridView ID="GridView1" runat="server">
    </asp:GridView>
</div>
</form>
</body>
</html>

```

注意，在这个查询中，我们直接告诉 LINQ 从哪里加载 XML 数据，应从该文档中检索出数据的哪些元素，在本例中是指所有的 Movie 元素。除了这些微小的改动外，该 LINQ 查询与前面的查询完全相同。

执行这段代码，会得到如图 10-7 所示的页面。

Value	Xml	HasAttributes	HasElements	IsEmpty	Value
ShrekAndrew Adams05/16/200188	<Movie><Title>Shrek</Title><Director>Andrew Adams</Director><Genre>0</Genre><ReleaseDate>5/16/2001</ReleaseDate><RunTime>88</RunTime></Movie>	10	10	10	ShrekAndrew Adams05/16/200188
FletchMichael Ritchie07/31/198396	<Movie><Title>Fletch</Title><Director>Michael Ritchie</Director><Genre>0</Genre><ReleaseDate>7/31/1983</ReleaseDate><RunTime>96</RunTime></Movie>	10	10	10	FletchMichael Ritchie07/31/198396
CasablancaMichael Curtiz11/1/1942102	<Movie><Title>Casablanca</Title><Director>Michael Curtiz</Director><Genre>1</Genre><ReleaseDate>11/1/1942</ReleaseDate><RunTime>102</RunTime></Movie>	10	10	10	CasablancaMichael Curtiz11/1/1942102
BatmanTim Burton16/23/1989126	<Movie><Title>Batman</Title><Director>Tim Burton</Director><Genre>1</Genre><ReleaseDate>6/23/1989</ReleaseDate><RunTime>126</RunTime></Movie>	10	10	10	BatmanTim Burton16/23/1989126
Dances with WolvesKevin Costner11/1/1990140	<Movie><Title>Dances with Wolves</Title><Director>Kevin Costner</Director><Genre>1</Genre><ReleaseDate>11/1/1990</ReleaseDate><RunTime>140</RunTime></Movie>	10	10	10	Dances with WolvesKevin Costner11/1/1990140
Dirty DancingEmile Ardeleanu18/21/1987100	<Movie><Title>Dirty Dancing</Title><Director>Emile Ardeleanu</Director><Genre>1</Genre><ReleaseDate>8/21/1987</ReleaseDate><RunTime>100</RunTime></Movie>	10	10	10	Dirty DancingEmile Ardeleanu18/21/1987100
The Parent TrapNancy Meyers07/29/1998127	<Movie><Title>The Parent Trap</Title><Director>Nancy Meyers</Director><Genre>0</Genre><ReleaseDate>7/29/1998</ReleaseDate><RunTime>127</RunTime></Movie>	10	10	10	The Parent TrapNancy Meyers07/29/1998127
RansomRon Howard11/13/1996123	<Movie><Title>Ransom</Title><Director>Ron Howard</Director><Genre>1</Genre><ReleaseDate>11/13/1996</ReleaseDate><RunTime>123</RunTime></Movie>	10	10	10	RansomRon Howard11/13/1996123
Ocean ElevenSteven Soderbergh11/27/2001118	<Movie><Title>Ocean Eleven</Title><Director>Steven Soderbergh</Director><Genre>1</Genre><ReleaseDate>11/27/2001</ReleaseDate><RunTime>118</RunTime></Movie>	10	10	10	Ocean ElevenSteven Soderbergh11/27/2001118
SteelMagnusJensen Rasmussen11/15/1999117	<Movie><Title>Steel</Title><Director>Magnus Rasmussen</Director><Genre>1</Genre><ReleaseDate>11/15/1999</ReleaseDate><RunTime>117</RunTime></Movie>	10	10	10	SteelMagnusJensen Rasmussen11/15/1999117
Mythic PizzaDonald Petrus10/21/1988104	<Movie><Title>Mythic Pizza</Title><Director>Donald Petrus</Director><Genre>1</Genre><ReleaseDate>10/21/1988</ReleaseDate><RunTime>104</RunTime></Movie>	10	10	10	Mythic PizzaDonald Petrus10/21/1988104
Pretty WomanGarry Marshall12/23/1990119	<Movie><Title>Pretty Woman</Title><Director>Garry Marshall</Director><Genre>1</Genre><ReleaseDate>12/23/1990</ReleaseDate><RunTime>119</RunTime></Movie>	10	10	10	Pretty WomanGarry Marshall12/23/1990119
Interview with the VampireNeil Jordan11/11/1994123	<Movie><Title>Interview with the Vampire</Title><Director>Neil Jordan</Director><Genre>1</Genre><ReleaseDate>11/11/1994</ReleaseDate><RunTime>123</RunTime></Movie>	10	10	10	Interview with the VampireNeil Jordan11/11/1994123
Top GunTony Scott02/16/1986110	<Movie><Title>Top Gun</Title><Director>Tony Scott</Director><Genre>2</Genre><ReleaseDate>2/16/1986</ReleaseDate><RunTime>110</RunTime></Movie>	10	10	10	Top GunTony Scott02/16/1986110
Mission ImpossibleBrian De Palma23/22/1996110	<Movie><Title>Mission Impossible</Title><Director>Brian De Palma</Director><Genre>2</Genre><ReleaseDate>5/23/1996</ReleaseDate><RunTime>110</RunTime></Movie>	10	10	10	Mission ImpossibleBrian De Palma23/22/1996110
The GodfatherFrancis Ford	<Movie><Title>The Godfather</Title><Director>Francis Ford				The GodfatherFrancis Ford

图 10-7

注意，包含在查询结果集中的字段没有显示我们期望的节点数据，每个子节点也没有作为 GridView 中的独立字段显示。这是因为该程序清单中使用的查询返回了 XElement 对象的集合，而不是我们期望的 Movie 对象。LINQ 无法指定每个节点应映射为什么对象类型。但是，可以在查询中添加一些映射逻辑，指定把每个节点映射为 Movie 对象，以及节点的子元素如何映射为 Movie 对象的属性，如程序清单 10-21 所示。

程序清单 10-21 使用 LINQ 映射 XML 元素

```

protected void Page_Load(object sender, EventArgs e)
{
    var query = from m in XElement.Load(MapPath("Movies.xml")).Elements("Movie")
                select new Movie {
                    Title = (string)m.Element("Title"),
                    Director = (string)m.Element("Director"),
                    Genre = (int)m.Element("Genre"),
                    ReleaseDate = (DateTime)m.Element("ReleaseDate"),
                    RunTime = (int)m.Element("RunTime")
                };
}

```

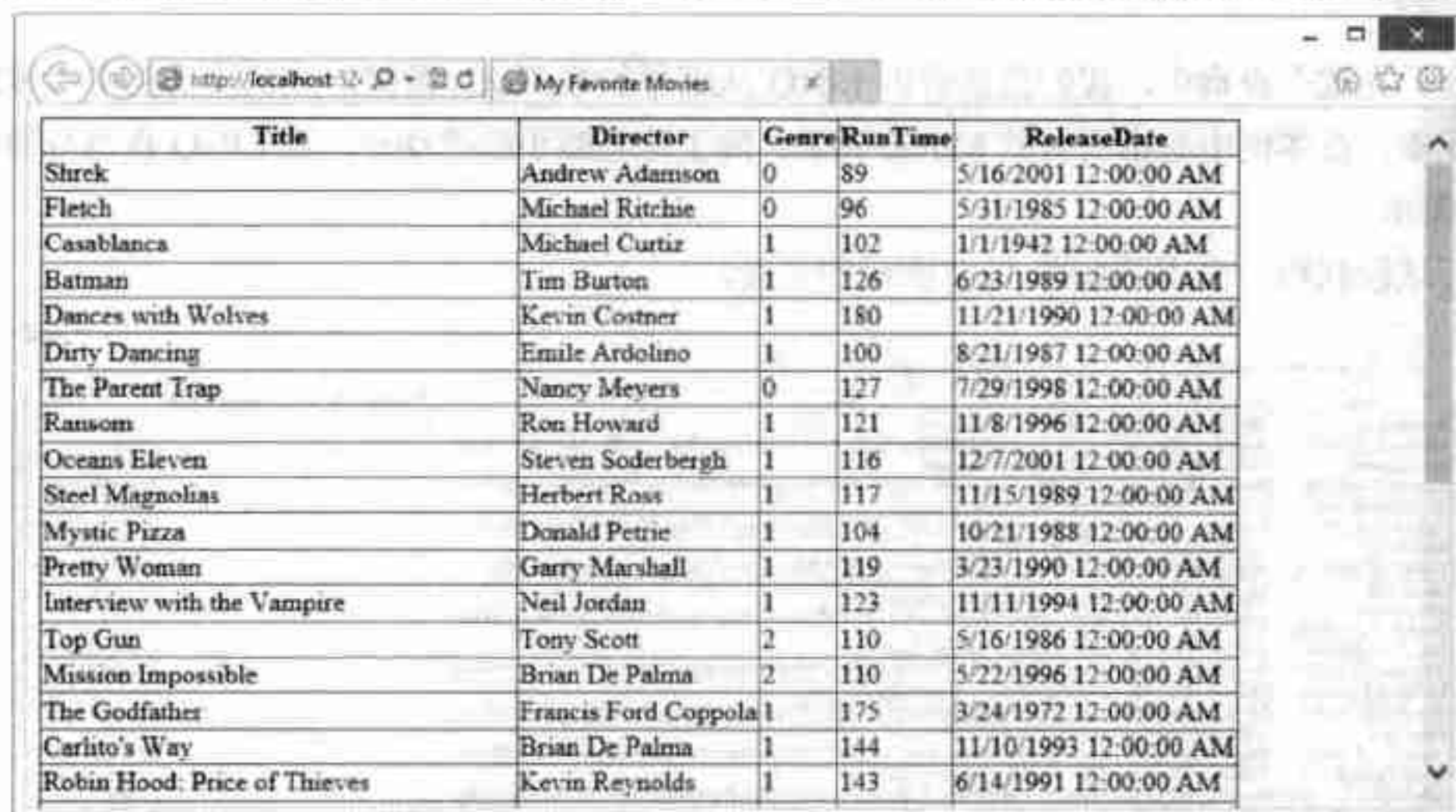


```

this.GridView1.DataSource = query;
this.GridView1.DataBind();
}

```

可以看出, 查询现在包含了映射逻辑, 因此 LINQ 知道我们的真正意图是创建结果集, 其中包含 Movie 元素内部节点的值。运行这段代码会使 GridView 包含我们期望的内容, 如图 10-8 所示。



Title	Director	Genre	RunTime	ReleaseDate
Shrek	Andrew Adamson	0	89	5/16/2001 12:00:00 AM
Fletch	Michael Ritchie	0	96	5/31/1985 12:00:00 AM
Casablanca	Michael Curtiz	1	102	1/1/1942 12:00:00 AM
Batman	Tim Burton	1	126	6/23/1989 12:00:00 AM
Dances with Wolves	Kevin Costner	1	180	11/21/1990 12:00:00 AM
Dirty Dancing	Emile Ardolino	1	100	8/21/1987 12:00:00 AM
The Parent Trap	Nancy Meyers	0	127	7/29/1998 12:00:00 AM
Ransom	Ron Howard	1	121	11/8/1996 12:00:00 AM
Oceans Eleven	Steven Soderbergh	1	116	12/7/2001 12:00:00 AM
Steel Magnolias	Herbert Ross	1	117	11/15/1989 12:00:00 AM
Mystic Pizza	Donald Petrie	1	104	10/21/1988 12:00:00 AM
Pretty Woman	Garry Marshall	1	119	3/23/1990 12:00:00 AM
Interview with the Vampire	Neil Jordan	1	123	11/11/1994 12:00:00 AM
Top Gun	Tony Scott	2	110	5/16/1986 12:00:00 AM
Mission Impossible	Brian De Palma	2	110	5/22/1996 12:00:00 AM
The Godfather	Francis Ford Coppola	1	175	3/24/1972 12:00:00 AM
Carlito's Way	Brian De Palma	1	144	11/10/1993 12:00:00 AM
Robin Hood: Price of Thieves	Kevin Reynolds	1	143	6/14/1991 12:00:00 AM

图 10-8



XElement 的 Load 方法试图加载整个 XML 文档, 因此, 最好不要使用这个方法加载非常大的 XML 文件。

10.2.2 连接 XML 数据

LINQ to XML 支持与 LINQ to Objects 相同的查询过滤和分组操作。还支持数据的连接, 可以把两个不同的 XML 文档中的数据合并起来, 这个任务以前是很难完成的。下面使用与 LINQ to Objects 相同的基本连接条件。基本的 XML 数据仍然只包含 Genre 的 ID 值。但最好在结果集中显示实际的 Genre 名称。

对于 XML 数据, Genre 并不存储在独立的列表中, 而存储在完全独立的 XML 文件中, 如程序清单 10-22 所示。

程序清单 10-22 XML 数据

```

<?xml version="1.0" encoding="utf-8" ?>
<Genres>
  <Genre>
    <ID>0</ID>
    <Name>Comedy</Name>
  </Genre>

```

```

</Genre>
<Genre>
  <ID>1</ID>
  <Name>Drama</Name>
</Genre>
<Genre>
  <ID>2</ID>
  <Name>Action</Name>
</Genre>
</Genres>

```

要连接数据，可以使用与程序清单 10-17 非常类似的连接查询，如程序清单 10-23 所示。

程序清单 10-23 使用 LINQ 连接 XML 数据

```

protected void Page_Load(object sender, EventArgs e)
{
    var query = from m in XElement.Load(MapPath("Movies.xml")).Elements("Movie")
                join g in XElement.Load(MapPath("Genres.xml")).Elements("Genre")
                on (int)m.Element("Genre") equals (int)g.Element("ID")
                select new {
                    Title = (string)m.Element("Title"),
                    Director = (string)m.Element("Director"),
                    Genre = (string)g.Element("Name"),
                    ReleaseDate = (DateTime)m.Element("ReleaseDate"),
                    RunTime = (int)m.Element("RunTime")
                };

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}

```

在这个例子中，使用 `XElement.Load` 方法作为 LINQ 连接语句的一部分，告诉 LINQ 从哪里加载 Genre 数据。连接完数据后，就可以像访问 Movie 数据的元素那样，访问 Genre 数据的元素。

10.3 LINQ to SQL

顾名思义，LINQ to SQL 可以快捷地查询基于 SQL 的数据源，如 SQL Server 2005 及其以后版本。与之前介绍的 LINQ 类型一样，LINQ to SQL 也是核心 .NET Framework 的扩展，其功能位于 `System.Data.Linq` 程序集中。

10.3.1 使用 O/R 映射器

除了每种 LINQ 都提供的一般 IntelliSense 和强类型化检查之外，LINQ to SQL 还直接在 Visual Studio 中包含基本的 Object/Relation(O/R)映射器。O/R 映射器可以快速将基于 SQL 的数据源映射为 CLR 对象，之后就可以使用 LINQ 进行查询。

使用 O/R 映射器，会把新的 LINQ to SQL Classes 文件添加到 Web 站点项目中。使用 LINQ to SQL File 文档类型可以方便地、可视化地创建数据上下文，之后就可以访问和执行 LINQ 查询。图 10-9 显示了 Add New Item 对话框中的 LINQ to SQL Classes 文件类型。

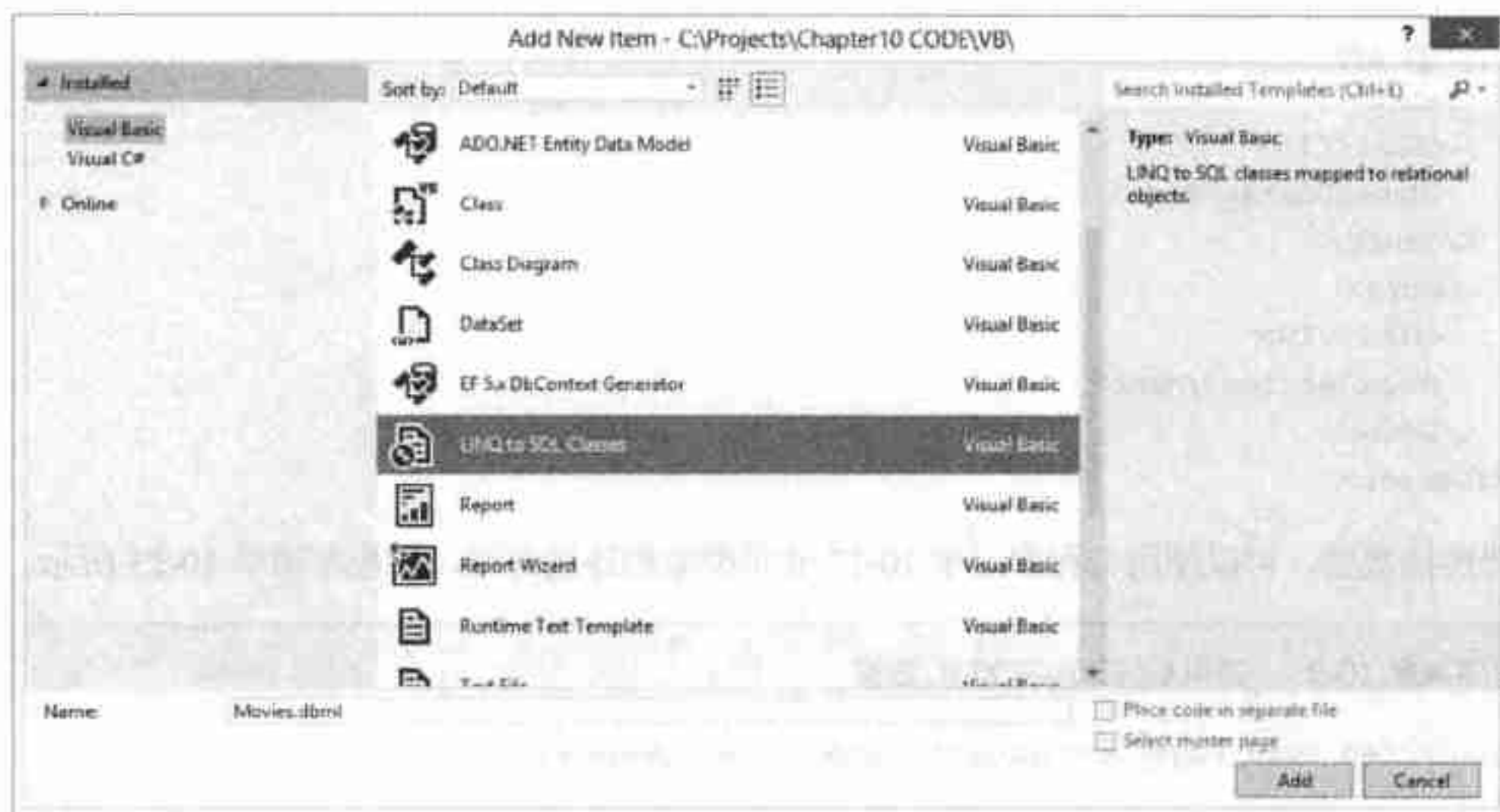


图 10-9

单击 Add New Item 对话框中的 OK 按钮，将文件添加到项目中后，Visual Studio 就会说明，要把 LINQ to SQL 文件添加到 Web 站点的 App_Code 目录下。在该目录下找到该文件后，LINQ to SQL Classes 文件创建的数据上下文就可以在 Web 站点的任意位置访问。

添加完文件后，Visual Studio 会自动在 LINQ to SQL 设计界面中打开文件。这是一个简单的 O/R 映射器设计工具，可以添加、创建、删除和关联数据对象。在设计界面上修改对象时，LINQ to SQL 会生成镜像了这些对象的结构的对象类。以后在准备为数据对象编写 LINQ 查询时，这些类允许 Visual Studio 提供设计期间的 IntelliSense 支持、强类型化以及编译期间的类型检查。因为 O/R 映射器主要用于 LINQ to SQL，所以也便于创建 SQL 对象的 CLR 对象表示，如表、视图和存储过程。

为了演示 LINQ to SQL 的用法，下面使用本章前面使用的示例 Movie 数据。对于本节，这些数据存储在 SQL Server Express 数据库中。



这个数据库(名为 Movies.mdf)的副本可以从 Wrox 网站(www.wrox.com)下载。

打开设计界面后，再打开 Visual Studio Server Explorer 工具，找到 Movies 数据库，展开该数据库的 Tables 文件夹。把 Movies 表从 Server Explorer 拖放到设计界面上。注意，只要把数据库表拖放到设计界面上，就会自动显示其结构。设计器会创建对应的实体类，并显示在设计界面上。

把表对象拖放到 LINQ to SQL 设计界面上时，Visual Studio 会检查表名称，如果需要，还会自动把生成的类名显示为单数形式。这样做有助于遵循 .NET Framework 的类命名规则。例如，如果把 Products 表从数据库拖放到设计界面上，Visual Studio 就会自动选择单数形式的名称 Product 作为所生成的类的名称。

虽然设计器一般会为类名指定正确的单复数形式，但这不是 100% 正确。例如，把 Movies 表拖放到设计界面上时，设计器会把 Movies 表变成不正确的单数形式 Movie。设计器还允许在设计界面上修改实体名。为此，只需在设计界面上选择实体，单击实体名即可。

添加了 Movie 实体后, 把 Genres 表拖放到设计界面上。Visual Studio 会创建这个表的类(注意给它指定了单数形式的名称 Genre)。另外, Visual Studio 还在 Movie 和 Genre 之间发现了一个已有的外键关系, 因此在两个表之间添加了一条虚线。该虚线的箭头表示两个表之间存在的外键关系的方向。添加了 Movies 和 Genres 表的 LINQ to SQL 设计界面如图 10-10 所示。

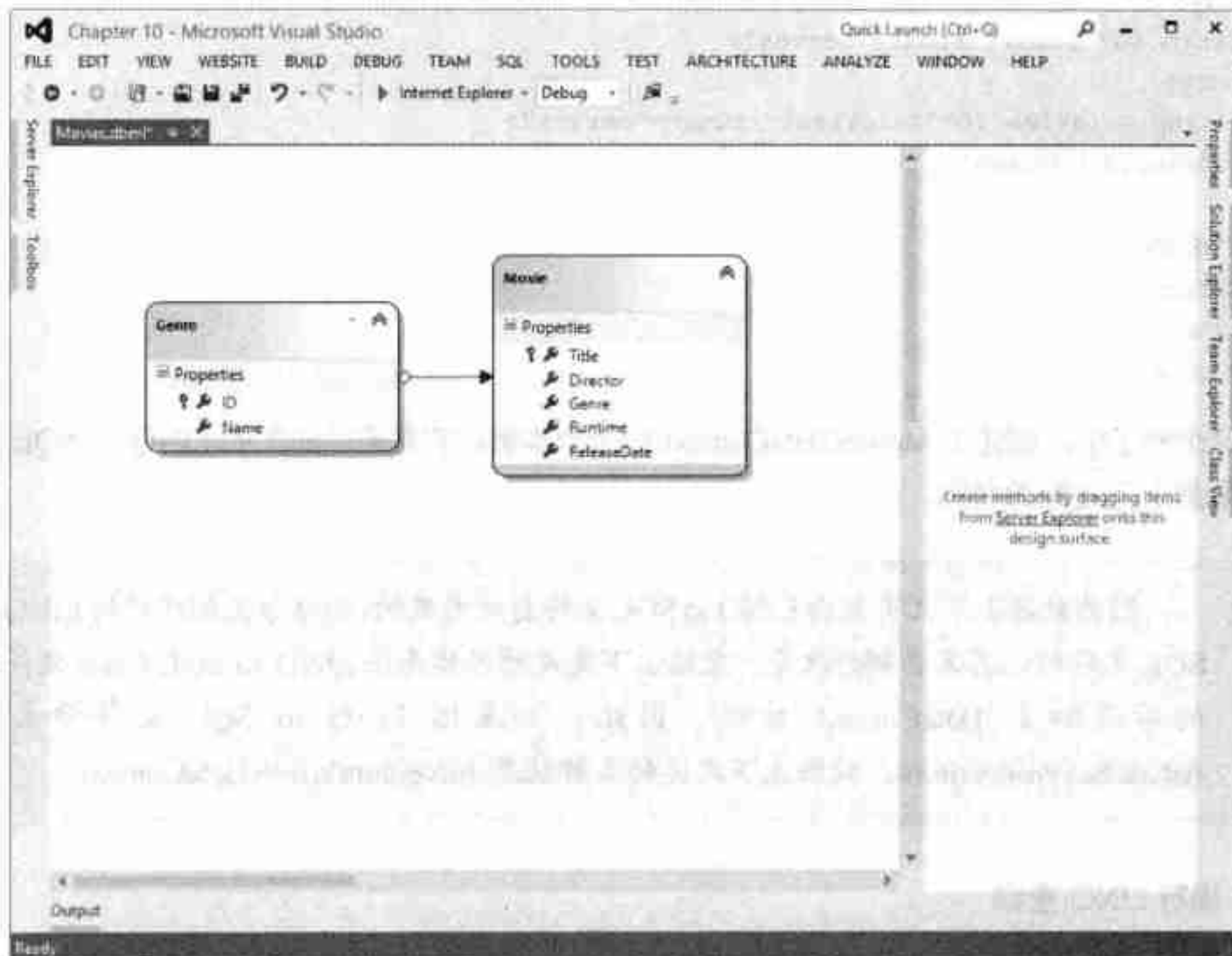


图 10-10

10.3.2 访问和查询数据

建立了 LINQ to SQL 文件后, 访问其数据上下文以及查询其数据就会非常简单。

1. 访问数据

首先, 在要访问数据的 Web 页面上创建数据上下文类的实例, 如程序清单 10-24 所示。

程序清单 10-24 创建新的数据上下文

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        MoviesDataContext dc = new MoviesDataContext();

    }
</script>
```

```

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title> My Favorite Movies </title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:GridView ID="GridView1" runat="server">
        </asp:GridView>

    </div>
  </form>
</body>
</html>

```

在这个例子中，创建了 MoviesDataContext 的一个实例，它是前面添加的 LINQ to SQL 文件所生成的数据上下文类的名称。



因为数据上下文类是由 LINQ to SQL 文件自动生成的，所以每次创建新的 LINQ to SQL 文件时，其名称都会改变。数据上下文类的名称是在 LINQ to SQL Class 文件名的后面加上 DataContext 后缀，因此，如果把 LINQ to SQL 文件命名为 AdventureWorks.dbml，数据上下文类的名称就是 AdventureWorksDataContext。

2. 编写 LINQ 查询

在页面上添加数据上下文后，就可以为之编写 LINQ 查询。如前所述，因为 LINQ to SQL 生成的对象类镜像了数据库表的结构，所以在编写 LINQ 查询时可以获得 IntelliSense 支持。程序清单 10-25 列出了与前面几节中相同的基本 Movie 列表查询。

程序清单 10-25 在 LINQ to SQL 中查询 Movie 数据

```

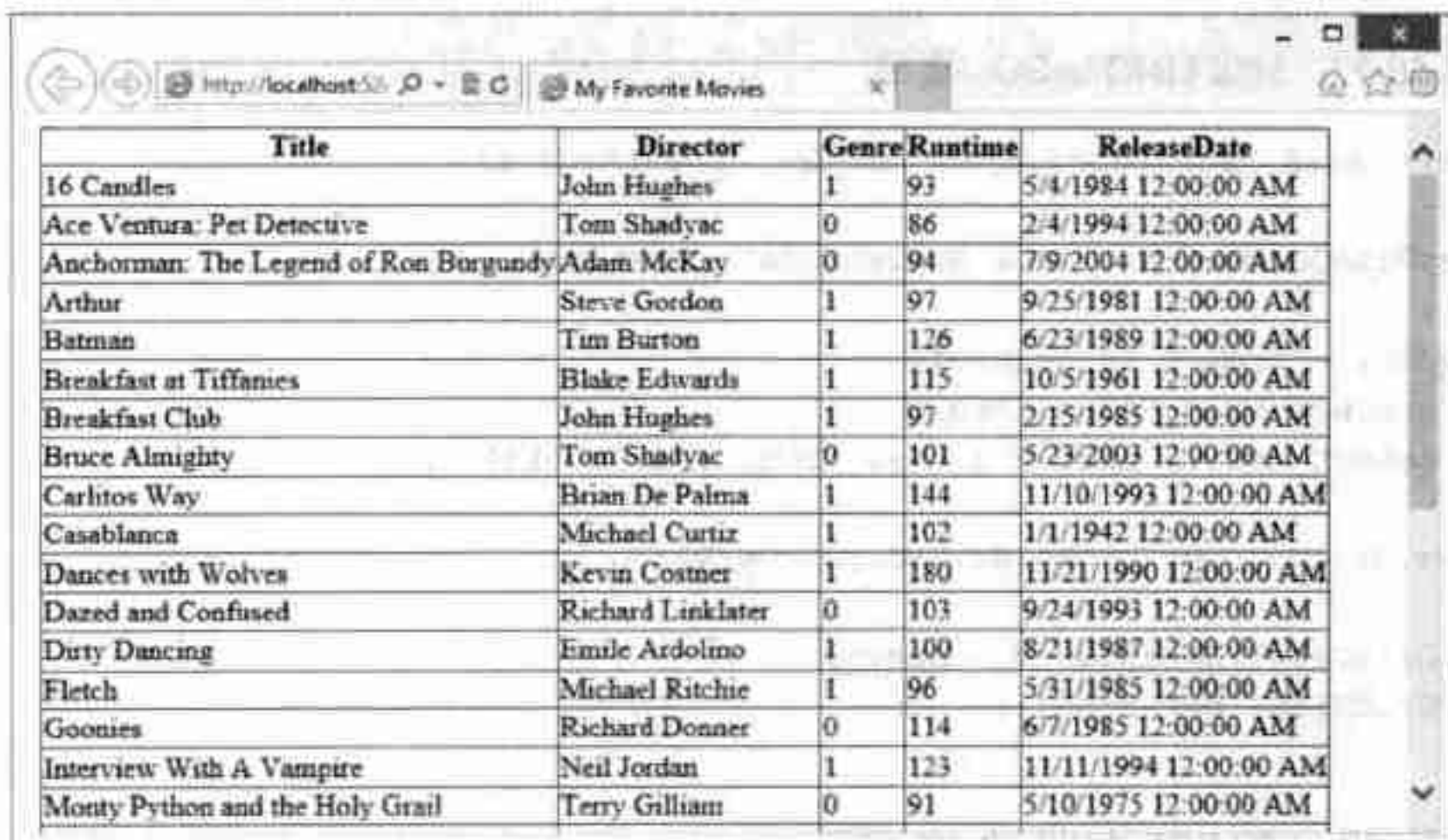
protected void Page_Load(object sender, EventArgs e)
{
  MoviesDataContext dc = new MoviesDataContext();

  var query = from m in dc.Movies
               select m;

  this.GridView1.DataSource = query;
  this.GridView1.DataBind();
}

```

如图 10-11 所示，运行这段代码会在数据库中生成 Movie 对象的原始列表。



Title	Director	Genre	Runtime	ReleaseDate
16 Candles	John Hughes	1	93	5/4/1984 12:00:00 AM
Ace Ventura: Pet Detective	Tom Shadyac	0	86	2/4/1994 12:00:00 AM
Anchorman: The Legend of Ron Burgundy	Adam McKay	0	94	7/9/2004 12:00:00 AM
Arthur	Steve Gordon	1	97	9/25/1981 12:00:00 AM
Batman	Tim Burton	1	126	6/23/1989 12:00:00 AM
Breakfast at Tiffanies	Blake Edwards	1	115	10/5/1961 12:00:00 AM
Breakfast Club	John Hughes	1	97	2/15/1985 12:00:00 AM
Bruce Almighty	Tom Shadyac	0	101	5/23/2003 12:00:00 AM
Carlitos Way	Brian De Palma	1	144	11/10/1993 12:00:00 AM
Casablanca	Michael Curtiz	1	102	1/1/1942 12:00:00 AM
Dances with Wolves	Kevin Costner	1	180	11/21/1990 12:00:00 AM
Dazed and Confused	Richard Linklater	0	103	9/24/1993 12:00:00 AM
Duty Dancing	Emile Ardolino	1	100	8/21/1987 12:00:00 AM
Fletch	Michael Ritchie	1	96	5/31/1985 12:00:00 AM
Goonies	Richard Donner	0	114	6/7/1985 12:00:00 AM
Interview With A Vampire	Neil Jordan	1	123	11/11/1994 12:00:00 AM
Monty Python and the Holy Grail	Terry Gilliam	0	91	5/10/1975 12:00:00 AM

图 10-11

注意，没有编写创建这个页面所需的任何数据库访问代码。LINQ 自动完成了这项工作，甚至根据 LINQ 语法生成了 SQL 查询。编写了查询后，就可以在 Visual Studio 输出窗口中看到 LINQ 为该查询生成的 SQL，如程序清单 10-26 所示。

程序清单 10-26 把 LINQ to SQL 查询写入输出窗口

```
protected void Page_Load(object sender, EventArgs e)
{
    MoviesDataContext dc = new MoviesDataContext();

    var query = from m in dc.Movies
                select m;

    System.Diagnostics.Debug.WriteLine(query);

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

现在，使用 Visual Studio 调试 Web 站点，就会看到 SQL 查询，如图 10-12 所示。

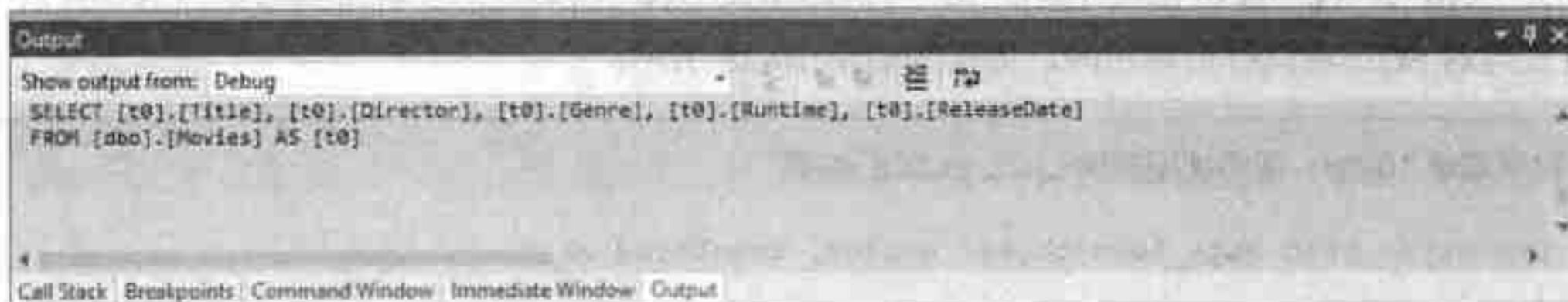


图 10-12

可以看出，生成的 SQL 是标准的 SQL 语法。LINQ 非常擅长于优化生成的 SQL，甚至是比较复杂的查询，例如程序清单 10-27 中的分组查询。

程序清单 10-27 分组 LINQ to SQL 数据

```
protected void Page_Load(object sender, EventArgs e)
{
    MoviesDataContext dc = new MoviesDataContext();

    var query = from m in dc.Movies
                group m by m.Genre into g
                select new { Genre = g.Key, Count = g.Count() };

    System.Diagnostics.Debug.WriteLine(query);

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

为这个查询生成的 SQL 如图 10-13 所示。

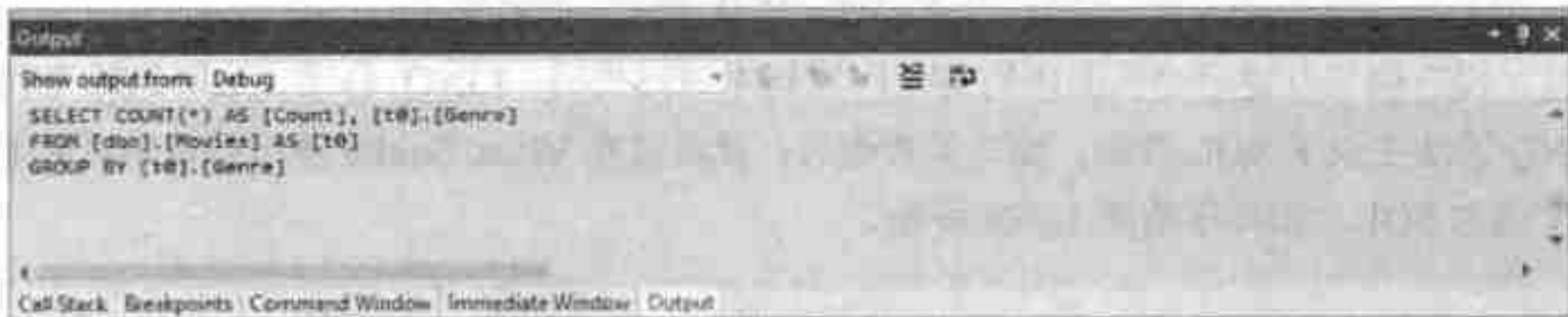


图 10-13

注意，LINQ to SQL 生成了为当前使用的 SQL Server 版本优化了的 SQL。

LINQ 还包含一个日志记录选项，通过设置数据上下文类的 Log 属性可以启用该选项。

10.3.3 使用其他 SQL 查询方法

LINQ to SQL 适合于生成 SQL 查询语法，但是有时使用其他 SQL 查询方法更适合，如存储过程或视图。

1. 使用 SQL 视图

要使用 SQL 视图和 LINQ to SQL，只需把视图拖放到 LINQ to SQL 设计界面上，这与操作标准的 SQL 表相同。视图显示在设计界面上，与前面添加的表一样。之后，就可以对视图执行查询，查询方式与对 SQL 表执行查询相同，如程序清单 10-28 所示。

程序清单 10-28 使用视图查询 LINQ to SQL 数据

```
protected void Page_Load(object sender, EventArgs e)
{
    MoviesDataContext dc = new MoviesDataContext();

    var query = from m in dc.AllMovies
                select m;

    System.Diagnostics.Debug.WriteLine(query);

    this.GridView1.DataSource = query;
```

```
this.GridView1.DataBind();
}
```

2. 使用存储过程

LINQ to SQL 把表和视图显示为属性，而存储过程可能需要参数。因此，LINQ to SQL 在数据上下文对象中把它们显示为方法调用，以便提供方法的参数值，再由 LINQ 把它们转换为存储过程参数。程序清单 10-29 显示了一个简单的存储过程，用于从数据库中检索指定的类型。

程序清单 10-29 简单的 SQL 存储过程

```
CREATE PROCEDURE dbo.GetGenre
(
    @id int
)
AS
SELECT * FROM Genre WHERE ID = @id
```

可以像处理表和视图那样，把存储过程从 Server Explorer 拖放到 LINQ to SQL Classes 设计界面上，以将该存储过程添加到 LINQ to SQL 设计器。如果希望存储过程从数据库的表中返回数据集，就应把存储过程拖放到 LINQ 类，LINQ 类表示查询返回的类型。程序清单 10-27 中的存储过程返回匹配所提供的 ID 的所有 Genre 记录。因此，应把 GetGenre 存储过程拖放到 Visual Studio 设计器的 Genres 表上。这就告诉设计器生成一个方法，该方法返回 Genre 对象的集合。将存储过程拖放到设计界面上后，该存储过程就显示在设计界面右边的列表中，这一点与表和视图不同。添加 GetGenre 存储过程后的设计界面如图 10-14 所示。

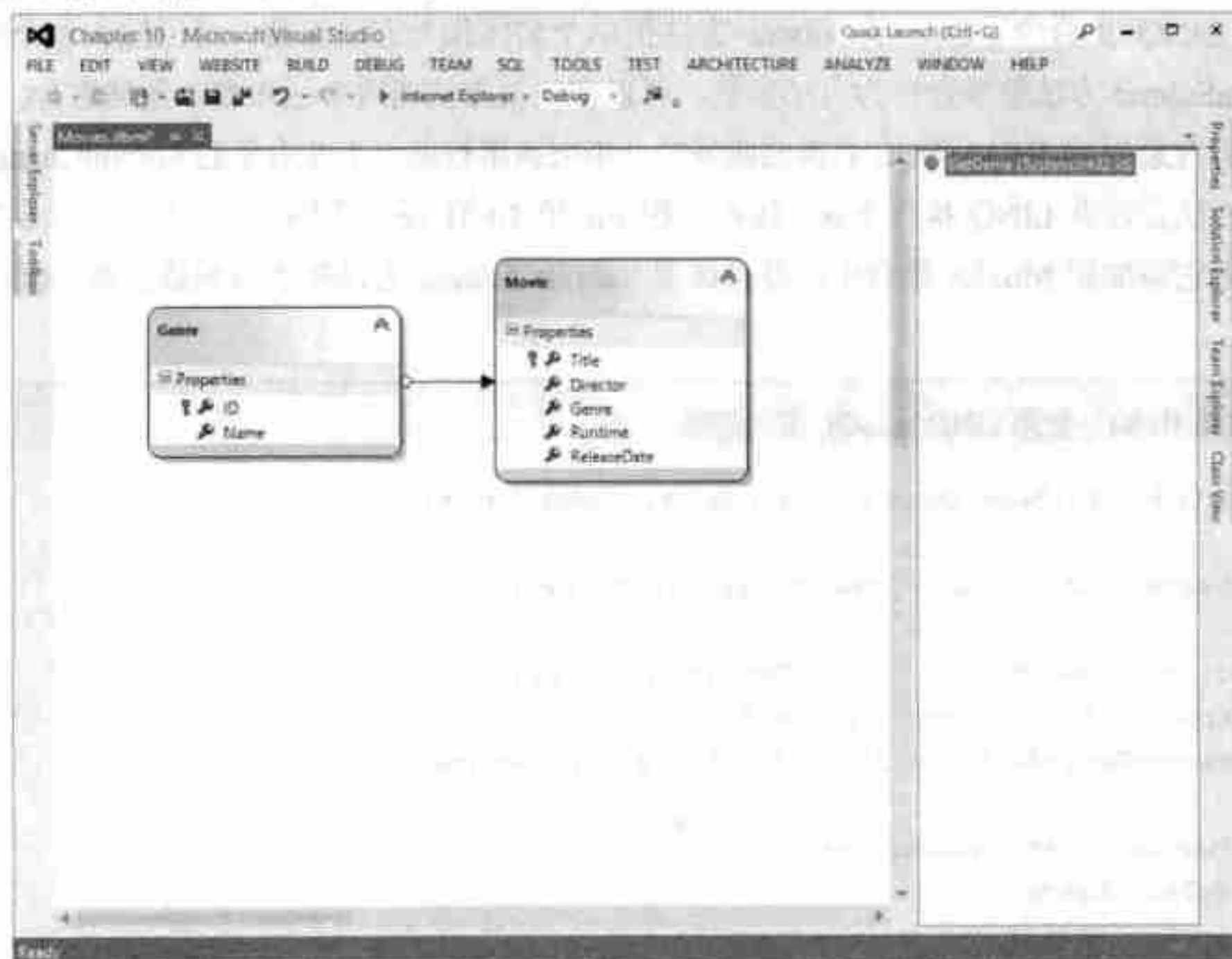


图 10-14

添加了存储过程后，就可以通过数据上下文访问它，这一点与访问表和视图一样。但是如前所述，LINQ to SQL 把存储过程显示为方法调用。因此，需要提供方法参数，如程序清单 10-30 所示。

程序清单 10-30 从存储过程中选择数据

```
protected void Page_Load(object sender, EventArgs e)
{
    MoviesDataContext dc = new MoviesDataContext();

    this.GridView1.DataSource = dc.GetGenre(1);
    this.GridView1.DataBind();
}
```

10.3.4 通过 LINQ 插入、更新和删除查询

LINQ to SQL 不仅可用于创建强大的查询，从数据源中选择数据，还可以管理插入、更新和删除操作。LINQ to SQL 完成该工作的方式默认与数据选择方式相同。LINQ to SQL 使用 SQL 结构的对象类表示方法动态地生成 SQL 命令 Insert、Update 和 Delete。与数据选择相同，也可以使用存储过程执行插入、更新和删除操作。

1. 使用 LINQ 插入数据

使用 LINQ to SQL 插入数据时只需创建要插入对象的新实例，并将之添加到对象集合中。LINQ 类提供了两个方法——InsertOnSubmit 和 InsertAllOnSubmit，使用它们可以方便地创建对象，并将对象添加到 LINQ 集合中。InsertOnSubmit 方法把单个实体作为方法参数，可以插入单个实体，而 InsertAllOnSubmit 方法把集合作为方法参数，可以在一次方法调用中插入整个数据集合。

添加了对象后，LINQ to SQL 就需要额外的一步来调用数据上下文对象的 SubmitChanges 方法。调用这个方法会告诉 LINQ 执行 Insert 操作。程序清单 10-31 是一个例子，它创建一个新的 Movie 对象，再把它添加到 Movies 集合中，并且调用 SubmitChanges 方法把所做的修改持久保存到 SQL 数据库中。

程序清单 10-31 使用 LINQ to SQL 插入数据

```
protected void Page_Load(object sender, EventArgs e)
{
    MoviesDataContext dc = new MoviesDataContext();

    Movie m = new Movie { Title="The Princess Bride",
        Director="Rob Reiner", Genre=0,
        ReleaseDate=DateTime.Parse("9/25/1987"), Runtime=98 };

    dc.Movies.InsertOnSubmit(m);
    dc.SubmitChanges();
}
```


2. 使用存储过程插入数据

当然，本章前面已经编写了一个复杂的存储过程，将数据插入数据库表中。LINQ 很容易使用已有的存储过程把数据插入表中。为此，在 LINQ to SQL 设计界面上，选择要在其中插入数据的实体，在本例中就是 Movie 实体。选择了该实体后，打开其属性窗口，找到 Default Methods 部分，如图 10-15 所示。

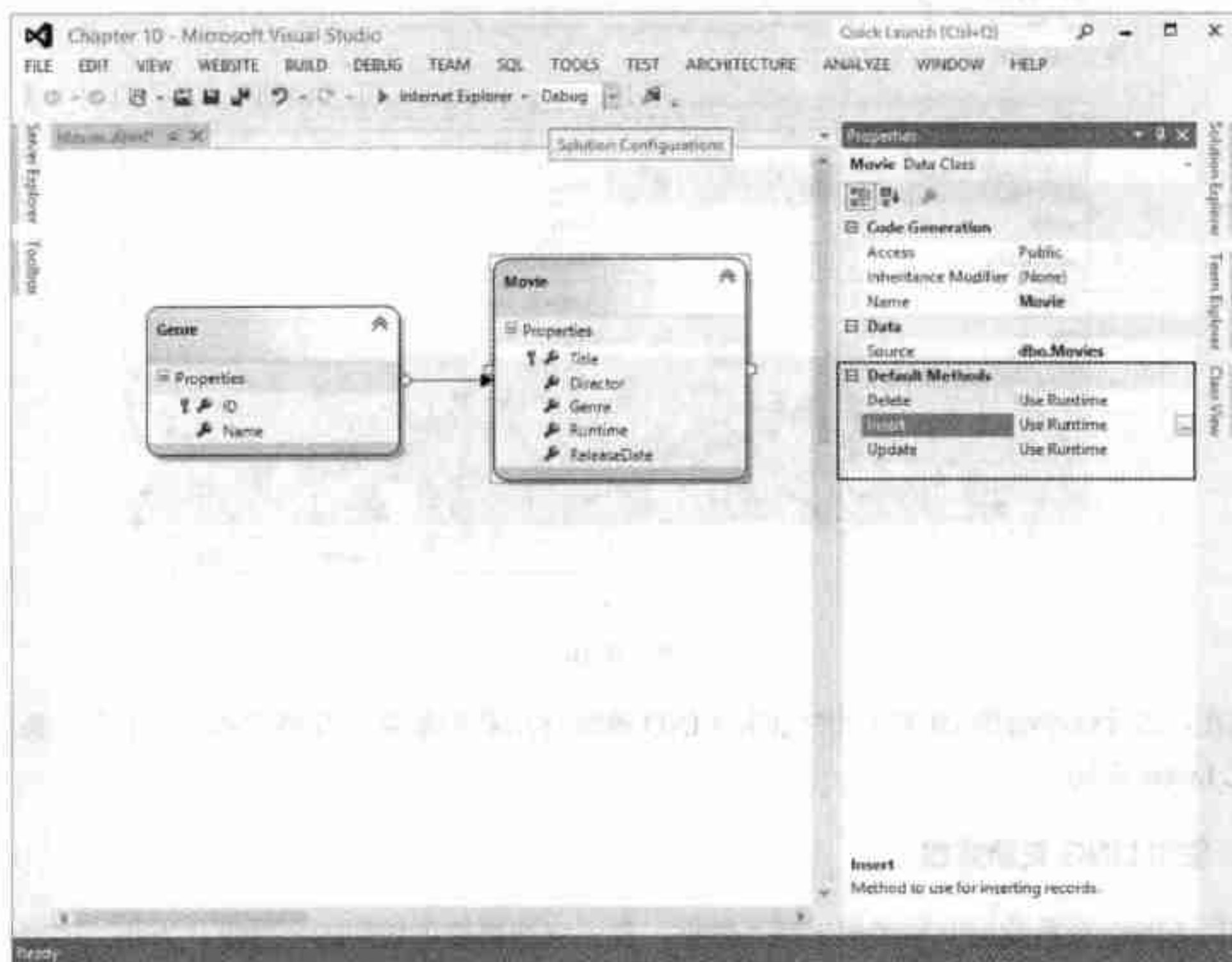


图 10-15

Default Methods 部分包含 3 个属性——Delete、Insert 和 Update，它们定义了 Movies 表上执行这些操作时 LINQ 应使用的行为。在默认情况下，每个属性的值都设置为 Use Runtime，它告诉 LINQ 在运行期间动态生成 SQL 语句。我们希望使用存储过程把数据插入表中，因此需要打开 Insert 属性的 Configure Behavior 对话框。

在这个对话框中，把 Behavior 单选按钮从 Use runtime 改为 Customize。接着，从这些单选按钮下面的下拉列表中选择对应的存储过程。选择完存储过程后，LINQ 会自动把表中的列匹配到存储过程的输入参数。但是，也可以在需要时手动修改这些匹配。

最终的 Configure Behavior 对话框如图 10-16 所示。

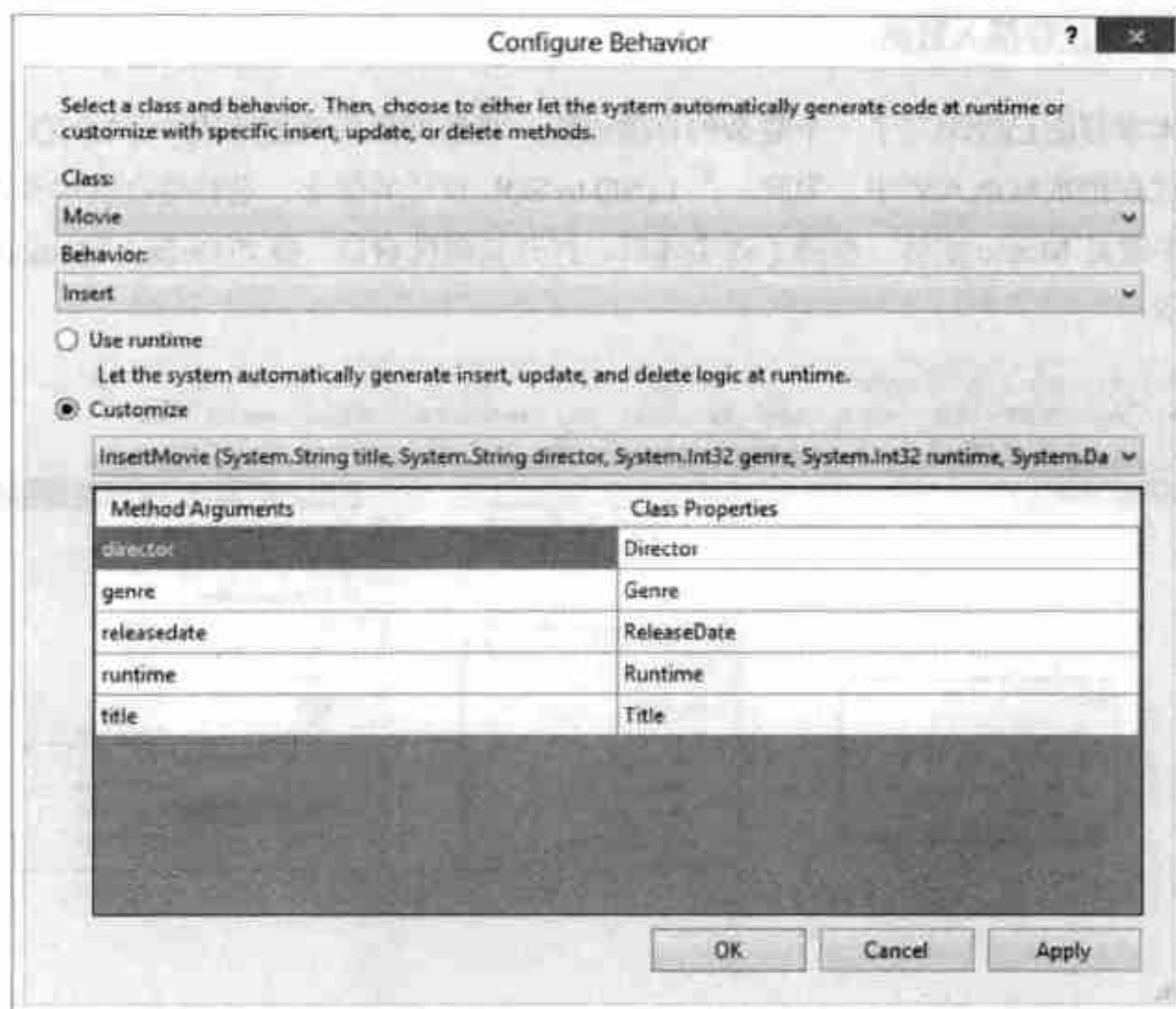


图 10-16

现在，运行程序清单 10-31 中的代码，LINQ 就会使用我们配置的存储过程，而不会动态生成一条 SQL Insert 语句。

3. 使用 LINQ 更新数据

使用 LINQ 更新数据非常类似于插入数据。首先应获得要更新的指定对象。为此，可以使用要更新的集合的 Single 方法。Single 标量方法根据其输入参数从集合中返回对象。如果有多条记录匹配该参数，Single 方法只返回第一条匹配的记录。

有了要更新的记录后，只需修改对象的属性值，再调用数据上下文类的 SubmitChanges 方法即可。程序清单 10-32 列出了更新特定电影所需的代码。

程序清单 10-32 使用 LINQ to SQL 更新数据

```
protected void Page_Load(object sender, EventArgs e)
{
    MoviesDataContext dc = new MoviesDataContext();

    var movie = dc.Movies.Single(m => m.Title == "Fletch");
    movie.Genre = 1;

    dc.SubmitChanges();
}
```


处理数据并发

LINQ to SQL 还包含和使用默认的优化并发功能。这意味着如果两个用户从数据库中检索出相同的记录并试图更新它们，那么第一个将其更新操作提交给服务器的用户就会胜出。如果第二个用户试图在第一个用户之后更新记录，LINQ to SQL 就会检测到原始记录已更新，从而引发 `ChangeConflictException` 异常。

4. 使用 LINQ 删除数据

最后，LINQ to SQL 还可以从 SQL 数据源中删除数据。LINQ to SQL 设计器生成的每个数据类对象都包含两个方法——`DeleteOnSubmit` 和 `DeleteAllOnSubmit`，用于从集合中删除对象。顾名思义，`DeleteOnSubmit` 方法从集合中删除单个对象，而 `DeleteAllOnSubmit` 方法从集合中删除所有记录。

程序清单 10-33 说明了如何使用 LINQ、`DeleteOnSubmit` 和 `DeleteAllOnSubmit` 方法从数据源中删除数据。

程序清单 10-33 使用 LINQ to SQL 删除数据

```
protected void Page_Load(object sender, EventArgs e)
{
    MoviesDataContext dc = new MoviesDataContext();

    //Select and remove all Action movies
    var query = from m in dc.Movies
                where m.Genre == 2
                select m;

    dc.Movies.DeleteAllOnSubmit(query);

    //Select a single movie and remove it
    var movie = dc.Movies.Single(m => m.Title == "Fletch");
    dc.Movies.DeleteOnSubmit(movie);

    dc.SubmitChanges();
}
```

与其他 SQL 命令一样，必须调用数据上下文类的 `SubmitChanges` 方法，把所做的修改保存到 SQL 数据源中。

10.4 LINQ to Entities

当需要快速构建数据访问代码时，LINQ to SQL 是非常适合的工具。在有设计优秀的数据库时，LINQ to SQL 也工作得很好。但是，LINQ to SQL 只支持实体类和数据库表之间的一对一映射。

Entity Framework (EF) 是对象-关系映射器，允许 .NET 开发人员使用域专用的对象处理关系数据，这些对象构成了所谓的概念模型。EF 还允许连接许多不同的数据提供程序。另外，也可以混合和匹配许多不同的数据库厂商、应用程序服务器或协议来设计一组混合的对象，构建各种表、源、服务等。

LINQ to Entities 允许开发人员使用 Visual Basic 或 Visual C# 给 Entity Framework 概念模型编写查询。LINQ to SQL 查询最终会创建在后台数据库中执行的 SQL，而 LINQ to Entities 会把 LINQ 查询转换为 Entity Framework 能理解的命令树查询，对 Entity Framework 执行这些查询，返回可由 Entity Framework 和 LINQ 使用的对象。尽管运行 LINQ to Entities 查询时，代码在后台执行，这与 LINQ to SQL 查询大不相同，但 LINQ 语法是相同的。



Entity Framework 详见第 11 章。

10.4.1 创建 Entity Framework 数据模型

为了使用 LINQ to Entities，需要创建 EF 数据模型。如果按照本章的示例执行，就已经有了一个带 App_Code 文件夹的项目。在 Visual Studio 的 Solution Explorer 中右击 App_Code 文件夹，选择 Add | New Item，打开 Add New Item 对话框。在这个对话框中，选择要使用的语言(在对话框的左侧面板中)。可以添加的可用选项包含 ADO.NET Entity Data Model。选择这一项，命名为 MoviesDM.edmx。在 Entity Data Model 向导中选择 Generate from database 选项。使用已有的 MoviesConnectionString，连接设置名使用默认的 MovieEntities。这也是以后使用的 EF 数据上下文的名称。在数据库对象列表中选择 Movies 表，其他设置采用默认值。单击 Finish 创建数据模型。打开实体数据模型设计器后，可能会显示安全警告，询问是否要运行文本模板。O/R 设计器使用 T4 模板自动生成 DbContext 文件，还为由数据库生成的每个实体自动生成一个文件。详见第 11 章。选中不再显示消息的选项，单击 OK。

如果现在构建解决方案，就会得到错误，指出 Movie 对象的属性已存在于另一个对象中。这是因为 LINQ to SQL 也生成了包含所有相同属性的 Movie 对象。为了避免这些冲突，应修改 LINQ to Entities 数据模型设计器生成的实体类名。在数据模型设计器中右击 Movie 对象，选择 Properties。把 Entity Set Name 改为 EFMovies，把 Name 改为 EFMovie。图 10-17 显示了修改后的属性面板。



图 10-17

修改后，保存 MovieDM.edmx 文件。右击 MovieDM.edmx 文件，选择 Run Custom Tool，重新生成实体类。

10.4.2 访问数据

创建一个新的 Web 窗体，使用实体数据模型显示数据。为了开始使用实体数据模型，在要访问数据的 Web 窗体中创建 EF 数据上下文类的实例，如程序清单 10-34 所示。

程序清单 10-34 创建新的 EF 数据上下文

```
<%@ Page Language="C#" %>

<!DOCTYPE html>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        MoviesEntities dc = new MoviesEntities();
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title> My Favorite Movies </title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:GridView ID="GridView1" runat="server">
            </asp:GridView>
        </div>
    </form>
</body>
</html>
```

在这个例子中，创建了 MoviesEntities 的一个实例，这是 LINQ to Entities 根据在 Entity Data Model 向导中指定的连接设置名生成的数据上下文类名。

10.4.3 编写 LINQ 查询

有了 EF 数据上下文类的一个实例后，就可以为之编写 LINQ 查询了。注意为 EF 数据上下文编写 LINQ 查询与使用 LINQ to SQL 编写查询是相同的。与其他 LINQ 查询一样，在编写 LINQ 查询时，LINQ to Entities 也提供了 IntelliSense 支持。程序清单 10-35 是基本的 Movie 列表查询，它在前面的章节中使用 LINQ 的其他技术编写。

程序清单 10-35 使用 LINQ to Entities 查询电影数据


```
protected void Page_Load(object sender, EventArgs e)
{
    MoviesEntities dc = new MoviesEntities();
```

```

var query = from m in dc.EFMovies select m;
GridView1.DataSource = query.ToList();
GridView1.DataBind();
}

```

图 10-18 显示了运行代码的结果。可以看出，与前面的例子一样，运行代码会生成数据库中电影的原始列表。



Title	Director	Genre	Runtime	ReleaseDate
16 Candles	John Hughes	1	93	5/4/1984 12:00:00 AM
Ace Ventura: Pet Detective	Tom Shadyac	0	86	2/4/1994 12:00:00 AM
Anchorman: The Legend of Ron Burgundy	Adam McKay	0	94	7/9/2004 12:00:00 AM
Arthur	Steve Gordon	1	97	9/25/1981 12:00:00 AM
Batman	Tim Burton	1	126	6/23/1989 12:00:00 AM
Breakfast at Tiffanies	Blake Edwards	1	115	10/5/1961 12:00:00 AM
Breakfast Club	John Hughes	1	97	2/13/1985 12:00:00 AM
Bruce Almighty	Tom Shadyac	0	101	5/23/2003 12:00:00 AM
Carlitos Way	Brian De Palma	1	144	11/10/1993 12:00:00 AM
Casablanca	Michael Curtiz	1	102	1/1/1942 12:00:00 AM
Dances with Wolves	Kevin Costner	1	180	11/21/1990 12:00:00 AM
Dazed and Confused	Richard Linklater	0	103	9/24/1993 12:00:00 AM
Dirty Dancing	Emile Ardolino	1	100	8/21/1987 12:00:00 AM
Fletch	Michael Ritchie	1	96	5/31/1985 12:00:00 AM
Goonies	Richard Donner	0	114	6/7/1985 12:00:00 AM
Interview With A Vampire	Neil Jordan	1	123	11/11/1994 12:00:00 AM
Monty Python and the Holy Grail	Terry Gilliam	0	91	5/10/1975 12:00:00 AM

图 10-18

LINQ 语法的类似性不仅仅体现在简单的查询中。可以添加过滤器和其他查询操作。例如，程序清单 10-36 修改了这个查询，使之包含过滤器和排序操作。

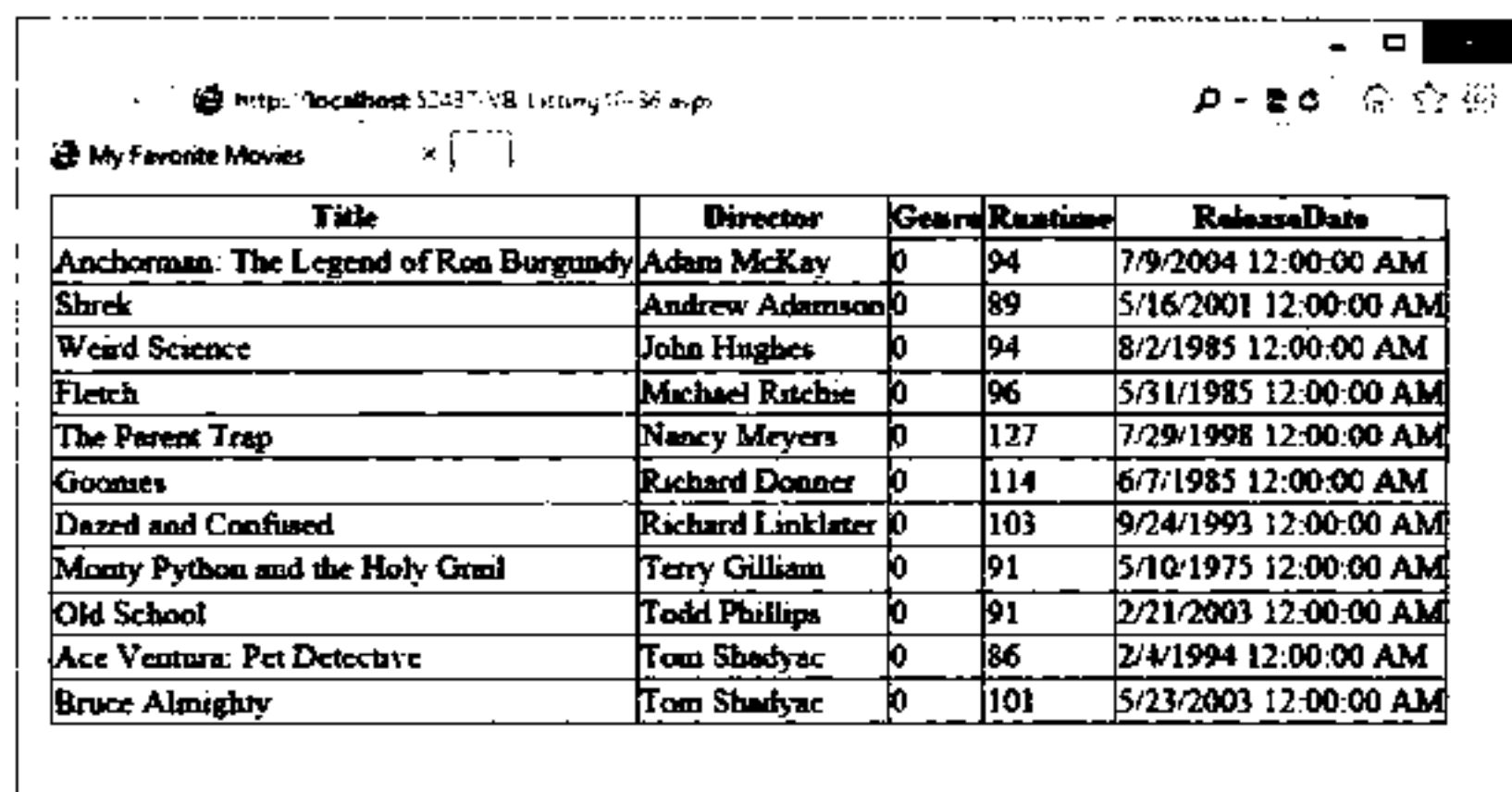
程序清单 10-36 给 LINQ to Entities 查询添加过滤和排序功能

```

protected void Page_Load(object sender, EventArgs e)
{
    MoviesEntities dc = new MoviesEntities();
    var query = from m in dc.EFMovies
        where m.Genre == 0
        orderby m.Director
        select m;
    GridView1.DataSource = query.ToList();
    GridView1.DataBind();
}

```

尽管程序清单 10-36 中的代码使用了 LINQ to Entities，但使用 LINQ 的其他技术查询时，LINQ 语法也是相同的。图 10-19 显示了运行程序清单 10-36 中的代码后的结果。



Title	Director	Genre	Runtime	ReleaseDate
Anchorman: The Legend of Ron Burgundy	Adam McKay	0	94	7/9/2004 12:00:00 AM
Shrek	Andrew Adamson	0	89	5/16/2001 12:00:00 AM
Weird Science	John Hughes	0	94	8/2/1985 12:00:00 AM
Fletch	Michael Ritchie	0	96	5/31/1985 12:00:00 AM
The Parent Trap	Nancy Meyers	0	127	7/29/1998 12:00:00 AM
Goonies	Richard Donner	0	114	6/7/1985 12:00:00 AM
Dazed and Confused	Richard Linklater	0	103	9/24/1993 12:00:00 AM
Monty Python and the Holy Grail	Terry Gilliam	0	91	5/10/1975 12:00:00 AM
Old School	Todd Phillips	0	91	2/21/2003 12:00:00 AM
Ace Ventura: Pet Detective	Tom Shadyac	0	86	2/4/1994 12:00:00 AM
Bruce Almighty	Tom Shadyac	0	101	5/23/2003 12:00:00 AM

图 10-19

10.5 本章小结

本章介绍了 .NET 4.5 中的 LINQ 查询大大简化了在 .NET 中查询数据的过程。LINQ 使查询变成重要的概念，直接内嵌在 .NET Framework 中。

本章首先讨论了执行对象查询的当前方法，包括基本的数据过滤、分组和排序操作。本章也论述了传统对象查询技术的缺点，包括需要开发人员定义查询应做什么，以及明确指定查询该如何做。另外，我们还看到甚至很简单的操作也需要非常复杂的代码，而这些代码很难阅读和维护。

接着，本章介绍了 LINQ 的 3 种基本类型：LINQ to Objects、LINQ to XML 和 LINQ to SQL。每种 LINQ 都使用相同的基本查询语法，大大简化了对象、XML 和 SQL 数据库的查询。本章讨论了如何使用类似于 SQL 的基本查询语法进行选择、过滤和分组。这种查询语法非常简洁，易于阅读，并且包含与 SQL 一样的许多运算符功能。

本章还探讨了 LINQ to SQL 中的基本 O/R 映射器。O/R 映射器便于创建表示 SQL 结构的 CLR 对象，例如表、视图和存储过程。创建好 CLR 对象后，就可以使用 LINQ 查询对象。

最后，本章介绍了如何使用 LINQ to SQL 修改数据库中的数据，如何使用生成的 SQL 语句，以及如何使用定制的存储过程。LINQ to Entities 可以合并使用更好的数据库访问技术——Entity Framework，但使用相同的我们已熟悉的 LINQ 语法。

第 11 章

Entity Framework

本章要点

- 了解映射和关系
- 使用 Database First 创建实体数据模型(Entity Data Model, EDM)
- 在 EDM 中使用 EntityDataSource 控件
- 使用 Code First 创建实体数据模型

访问数据几乎是每个应用程序都必须执行的一项操作。几乎所有的应用程序都以某种方式处理数据,这些数据来自于内存(内存中的数据)、数据库、XML 文件、文本文件或其他地方。许多开发人员都发现很难从 C#和 Visual Basic 的强类型化的面向对象环境迁移到对象是次要元素的数据层。在 Entity Framework 推出之前,要从使用强类型化的对象编写代码,迁移到使用从数据存储中返回的、弱类型化的数据集合很难实现,且常常出错。数据访问使应用程序的开发变得比较复杂,这有许多原因,其中之一就是,从设计和实现的角度来看,与数据库交互操作的代码和处理应用程序逻辑的代码是完全不同的。

Microsoft 一直在尝试简化程序员的常见任务,一般的做法是组合使用新的接口和 IDE 帮助,分离出这些任务中比较困难的部分。随着 Entity Framework 最新版本的推出,在数据库和应用程序之间导航会比以前容易一些。实际上,使用 Entity Framework Code First,跨越代码和数据库中的数据之间的鸿沟有了更多的选择。

使用 .NET 编程语言的一大优势是,代码中的对象是强类型化的。程序员很容易浏览名称空间,使用 Visual Studio IDE 中的调试器等。但在访问数据时,事情就有了很大的不同。这不是强类型化的环境,调试起来很痛苦,甚至根本没有调试功能,大多数时间都在把字符串作为查询传送给数据库。开发人员还需要了解底层数据以及它们的结构,理解所有的数据项之间是如何相关的。

Entity Framework 是一组 ADO.NET 技术,克服了上的许多困难。现在,使用 Entity Framework(和 LINQ),就有了一个轻量级的界面,它要为要处理的底层数据存储提供了一个强类型化的接口。使用这些技术,就可以一直处于熟悉的编码环境中,并能将底层数据作为对象访问,从而使用 IDE、IntelliSense 甚至调试功能。不再需要担心开发人员以前面对的、弱类型化的数据列表问题。Entity

Framework 允许使用已经熟悉的强类型对象处理数据。

现在不仅可以使⤿熟悉的方法访问和处理数据，还可以处理多个数据存储中的数据。ADO.NET Data Provider 模型在 .NET Framework 中提供了通用的托管界面来连接数据存储，并与之交互操作。Entity Framework 建立在 ADO.NET Data Provider 模型的基础上，允许使用 Entity Framework 处理某个支持的提供程序可用的数据源。 .NET Framework 包含 ADO.NET 提供程序，以直接访问 Microsoft SQL Server，用 ODBC 和 OLE DB 驱动程序间接访问其他数据库。但第三方提供程序允许直接访问其他流行的数据存储，例如 Oracle、MySQL、DB2 等。

本章将概述 Entity Framework 的最新版本以及如何在 ASP.NET 应用程序中使用这个架构。Entity Framework 的最新版本现在可以 EntityFramework NuGet 包的形式获取。NuGet Package Manager 已经安装在 Visual Studio 2012 中。

11.1 能否使用同一种语言

如前所述，建立与数据库通信的应用程序应该比较简单。其难点在于代码中的对象和数据库中的对象在本质上是不同的。数据库层中的对象与应用程序栈的其他层之间进行通信是复杂性增加的主要原因。

Entity Framework 可以把应用程序对象映射到关系数据库模式。例如，有一个 Orders 表，其中包含对 Customer 对象和 StoreItem 对象的引用，并且我们使用的是关系数据库模式，那么可以在各个表之间使用 JOIN 语句创建这些实体。但是，如果在 C# 或 Visual Basic 代码中使用该构造方法，就会创建一个 Order 对象，由它的一个属性引用 Customer 对象和 StoreItem 对象。通常，开发人员需要映射这些对象。

过去，需要使用 DataSet 对象解决这个映射问题。DataSet 对象是表的内存表示方法，其中包含多个表、表连接和约束。但是，在人们希望开发的纯面向对象的代码中，常常不推荐使用 DataSet 对象。

在数据库中表示数据时，要通过数据库的关系模式把数据表示为逻辑模型。但是，需要使用概念模型来完成应用程序的编写。为了把逻辑层和概念层联系起来，必须建立映射层。映射层允许把对象从代码使用的 .NET 类传送给在数据库中使用的关系数据库模式，如图 11-1 所示。

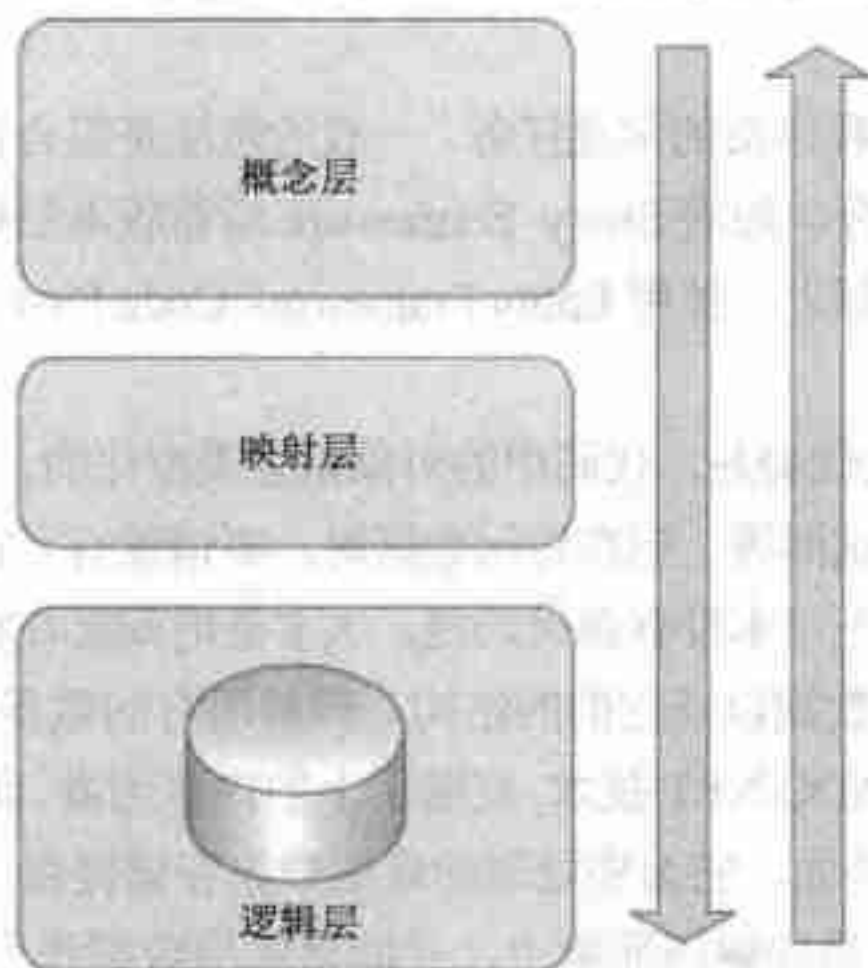


图 11-1

映射层有时被看作数据访问层。Microsoft 在过去几年提供了许多数据访问技术,许多第三方公司也想了不少方法,以使映射尽可能简单。一些第三方公司甚至为特定的数据库提供了映射或数据访问传输机制。

在使应用程序代码中的实体映射到数据库模式的过程尽可能简单这一方面,Microsoft 走在了前面。它推出了 Entity Framework,使应用程序与数据库通信所需编写的代码比以前少了一些。

注意,有时同时需要应用程序代码中的对象和在数据库关系模式中构建的实体。

因此,Entity Framework 包含:

- 在应用程序代码中表示的数据库模型
- 当前使用的数据存储的定义(例如,SQL Server 数据库中的数据表示)
- 上述两个元素之间的映射

11.1.1 开发 workflow 选项

开发人员喜欢选项。在 Entity Framework 中处理数据有 3 个选项:

- Database First
- Model First
- Code First

有了数据库后,就可以使用 Entity Framework 生成数据模型。Database First 和 Model First 开发 workflow 利用了 Visual Studio 2012 中获得支持的富设计器和工具。使用 Database First 会在设计界面上给已有数据库建立模型层。使用 Model First,则通过设计器定义模型层,再使用它生成数据库模式。数据库模式用于创建新数据库。设计器生成 DDL 语句以创建数据库。

另一个 workflow 选项最初在 Entity Framework 4 中引入,称为 Code First 开发选项。Code First 开发选项是以代码为中心的方法,不使用设计器。在 Code First 开发选项中,使用“旧的类”定义模型。数据库的创建和维护使用约定/配置方法来实现,所以不需要明确配置。

11.1.2 实体数据模型

在代码中表示实体数据模型(Entity Data Model, EDM)时,EDM 是抽象的数据概念模型,通常构建为 .NET 类,可以像处理代码中的其他对象那样处理 .NET 类。可以使用 Conceptual Schema Definition Language (CSDL)创建概念层,CSDL 是对象的 XML 定义。

使用 Store Schema Definition Language (SSDL)定义逻辑层,逻辑层详细描述了要在数据库中存储的关系模式,这包括表的定义、表之间的关系以及约束。

实体对象模型的最后一部分是映射层。映射层使用 Mapping Specification Language (MSL)映射 CSDL 和 SSDL 实例。

把这 3 层合并使用有以下几个原因。首先,在 EDM 中,这 3 层共享一个公共类型系统,因此可以指定应用程序的代码能理解的类型,当前使用的数据库也能理解这些类型。EDM 还具有使用继承和处理复杂对象的能力,以及在层之间进行适当映射的能力。

这 3 层——概念层、逻辑层和它们之间的映射层——存储在 .edmx 文件的 XML 中。Visual Studio 2012 的优点是可以使用图形化的设计器自动创建这些层和相应的 .edmx 文件。然后使用图形化设计器图形化地显示和编辑 .edmx 文件,这样就不必直接编辑 XML 了。

使用 Code First 开发选项,可以直接编写 C# 或 VB 类,定义概念模型。接着就可以使用概念模型生成数据库模式了。

11.2 创建第一个实体数据模型

为了举例说明 Entity Framework 的用法,下面首先从数据库中读取数据,并在 ASP.NET 应用程序中使用设计器工作流显示它们。

对于这个例子,需要安装 Visual Studio 2010 或 Visual Studio 2012。在这个 IDE 中,创建一个新的空白 ASP.NET Web 应用程序,名为 AspnetEntityFx,这是一个标准的 ASP.NET 应用程序。



接着,需要有一个数据库才能工作。可以从 www.wrox.com/go/SQLSever2012DataSets 上下载用于本书的 AdventureWorks 数据库。

要创建 App_Data 文件夹以添加数据库文件,应右击项目,选择 Add | Add ASP.NET Folder | App_Data。要添加数据库,可以在 Solution Explorer 中右击 App_Data 文件夹,从弹出的菜单中选择添加已有项的命令。

把数据库文件添加到项目中后,就需要把数据库升级到 SQL Express LocalDB。从 Visual Studio 菜单中选择 View,再选择 Server Explorer,打开 Server Explorer。展开 Data Connections,找到刚才添加的数据库文件。右击 AdventureWorks2012_Data.mdf 数据连接,选择 Modify Connection。单击 Advanced 按钮,打开 Advanced Properties 对话框。验证 Data Source 属性设置为(LocalDB)\v11.0, User Instance 属性设置为 False。图 11-2 显示的 Advanced Properties 对话框包含 Data Source 和 User Instance 属性的正确设置。

单击 OK 退出 Advanced Properties 对话框,再单击 OK 退出 Modify Connection 对话框,此时会显示一条消息,说明所连接的数据库与 SQL Server 的当前实例不兼容。图 11-3 显示了这个窗口。单击 Yes 升级数据库。



图 11-2



图 11-3

升级完数据库后,就准备创建 EDM。在 Visual Studio 的 Solution Explorer 中右击项目,选择 Add | New Item 命令,这会打开 Add New Item 对话框。在这个对话框中,选择当前采用的.NET 语言的 Data 选项(在对话框的左侧窗格中)。可以从 Data 选项添加的项包括 ADO.NET Entity Data Model,如图 11-4 所示。

对于这个例子,把实体数据模型命名为 EmployeeDM.edmx,如图 11-4 所示。

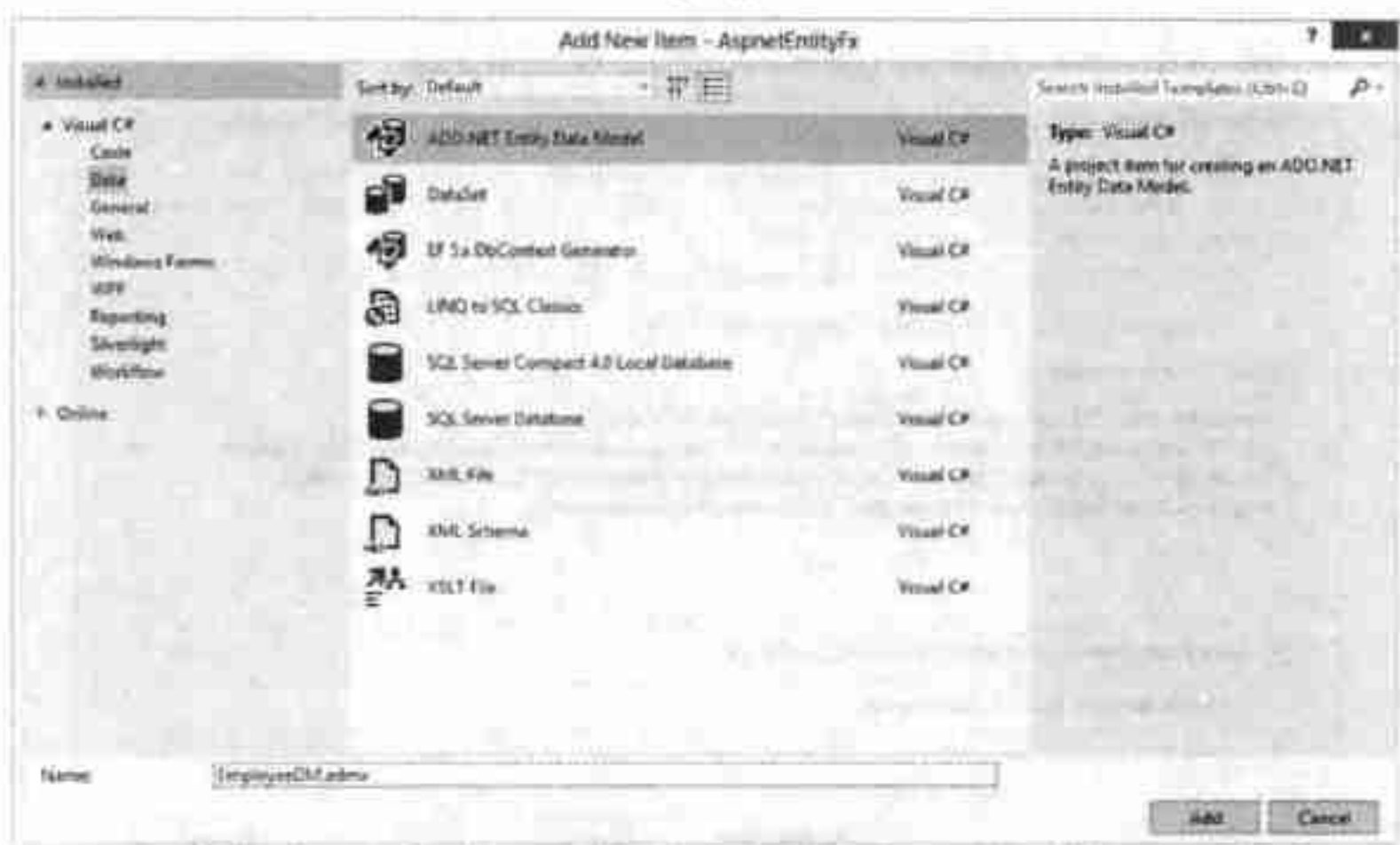


图 11-4

11.2.1 使用 EDM 向导

单击 Add 按钮添加这个文件时,并不会立即插入文件,而是启动向导。创建实体数据模型有两种方式:第一种是以已有的数据库为基础创建实体数据模型;第二种是从头开始创建实体数据模型。.NET Framework 4.5 使第二种方式更容易实现。如果选择使用第二种方式,那么可以先创建实体数据模型,然后利用向导来创建该模型的数据库表示。

该向导的第一个屏幕显示了这两种方式,如图 11-5 所示。

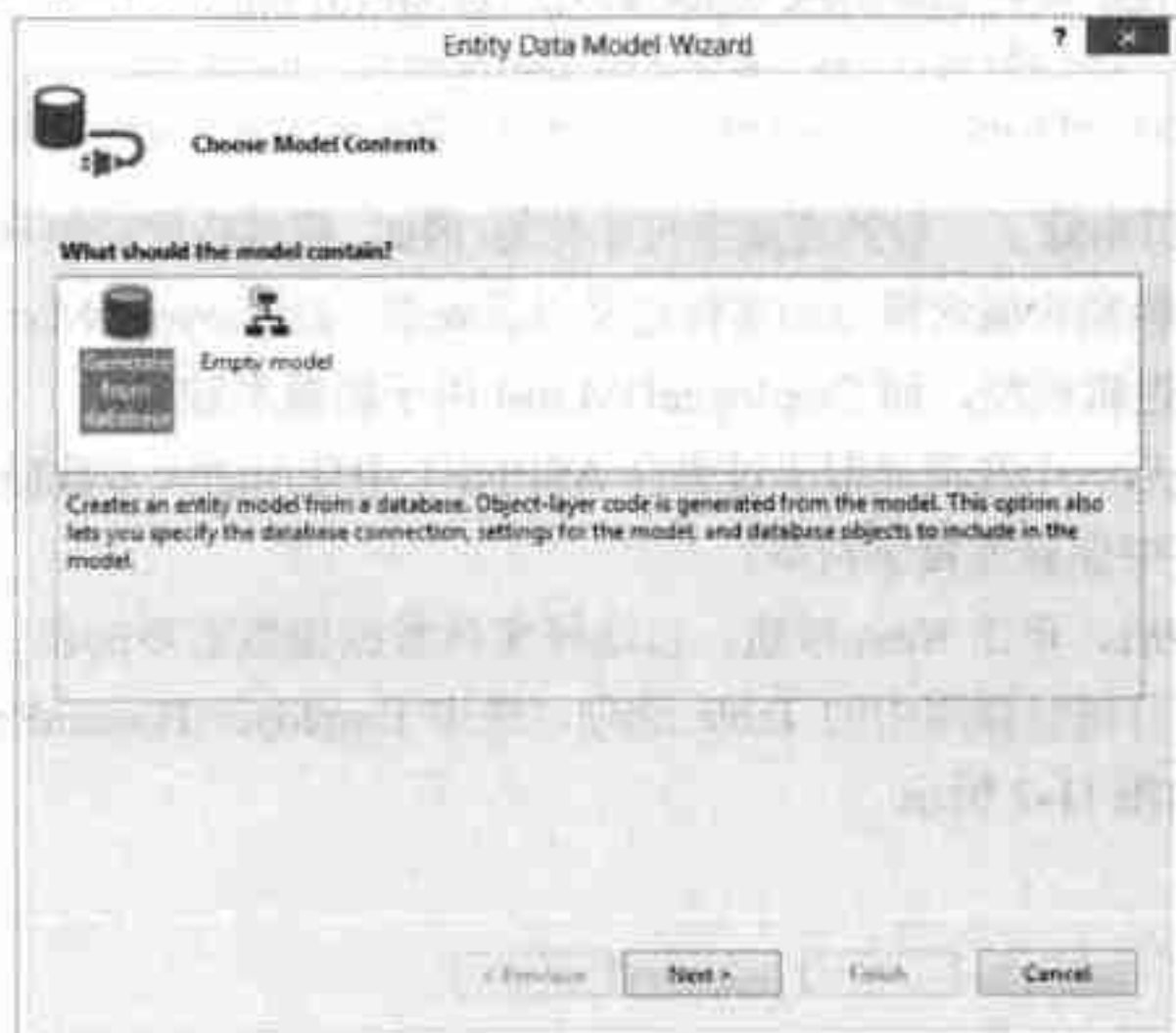


图 11-5

对于这个例子，选择 Generate from database 选项。单击向导中的 Next 按钮后，下一步是建立指向数据库的实体连接字符串，如图 11-6 所示。

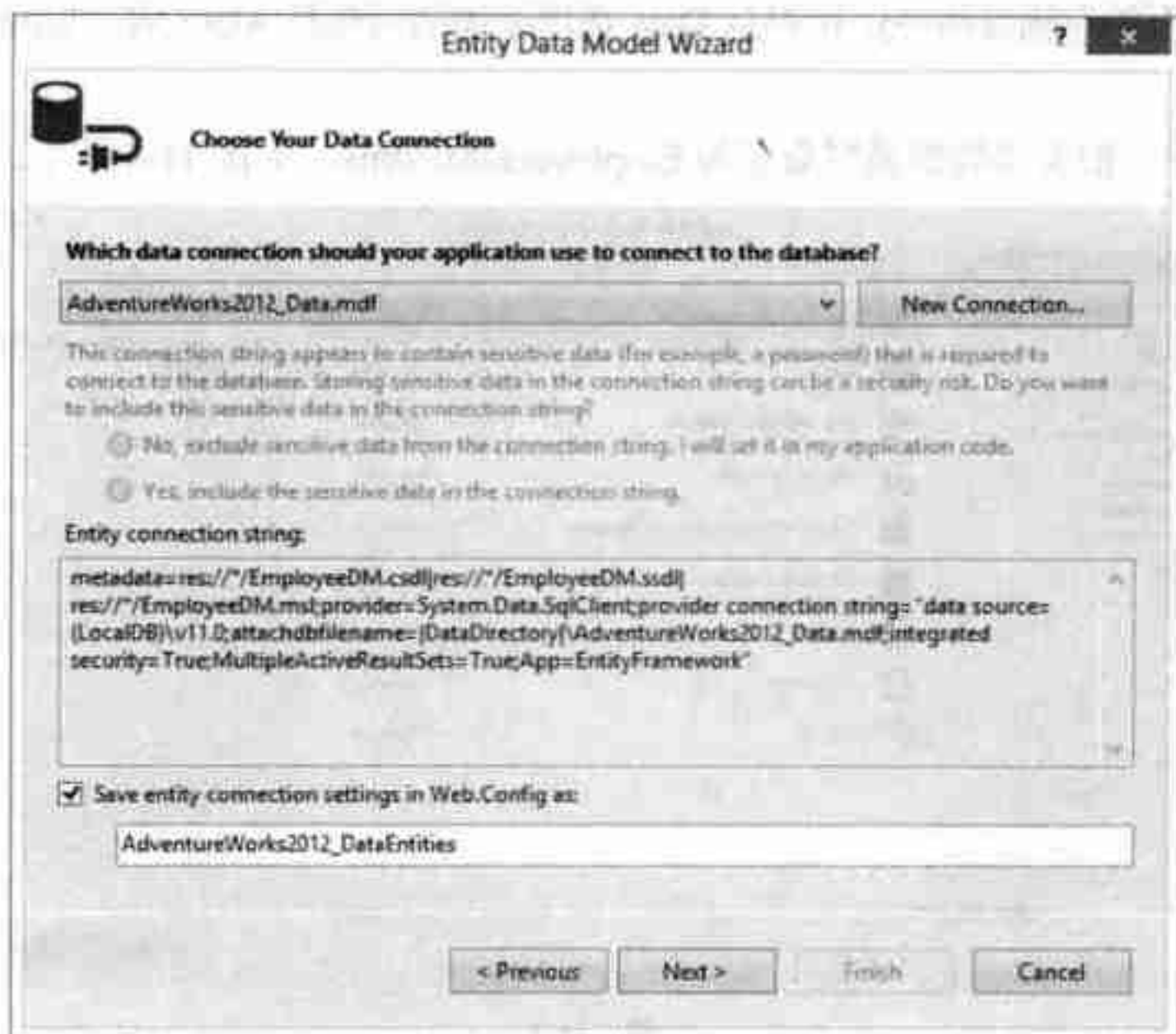


图 11-6

在这一步中，可以从第一个下拉列表框中选择要使用的数据库。如果按照前面的指示已把 AdventureWorks2012_Data.mdf 文件添加到项目中，那么该文件就显示为这个下拉列表框中的选项。在下拉列表框中选择这个数据库选项后，会显示实体连接字符串，非常类似于一般的连接字符串：

```
metadata=res://*/EmployeeDM.csdl|res://*/EmployeeDM.ssdl|res://*/EmployeeDM.msl;
provider=System.Data.SqlClient;
provider connection string="Data Source=(LocalDB)\v11.0;
attachdbfilename=|DataDirectory|\AdventureWorks2012_Data.mdf;
integrated security=True; MultipleActiveResultSets=True; App=EntityFramework "
```

注意，实体连接字符串除了一般的连接字符串信息(例如，提供程序特性和提供程序连接字符串)之外，还包含用于逻辑模型和概念模型的实体定义以及映射。EmployeeDM.csdl 文件用于概念模型，EmployeeDM.ssdl 用于逻辑模型，而 EmployeeDM.msl 用于需要的映射。

这个对话框中的最后一个选项类似于过去在 ASP.NET 中使用的大多数提供程序的配置，允许在项目的 web.config 文件中保存连接字符串。

完成向导的这一步后，单击 Next 按钮，以选择实体数据模型需要的表、视图和存储过程。

对于这个例子，展开树状视图中的 Table 选项，选中 Employee(HumanResources)表旁边的复选框，以选择这个表，如图 11-7 所示。



图 11-7

注意，向导的这部分可以定义在代码中访问模型时使用的名称空间，这个例子中使用的是默认选项 AdventureWorks2012_DataModel。

这一步是向导的最后一部分。单击 Finish 按钮，就会显示实体数据模型的设计器界面。打开设计器界面后，会显示一个安全警告，询问是否要运行文本模板。该设计器使用 T4 模板自动生成 DbContext 文件，并为选中的、由数据库生成的每个实体生成一个文件。单击不再显示消息的选项，再单击 OK。

11.2.2 使用 Entity Framework 设计器

内置于 Visual Studio 中且使用 Entity Framework 的 O/R 设计器非常强大，因为它允许可视化地配置概念层，控制如何把概念层映射到逻辑层。

如果按照前面的定义完成向导，在设计界面上就会显示一个 Employee 对象，如图 11-8 所示。

在设计器中突出显示 Employee 对象，Visual Studio 的 Properties 窗格就会显示一些基本属性，如图 11-9 所示。



图 11-8

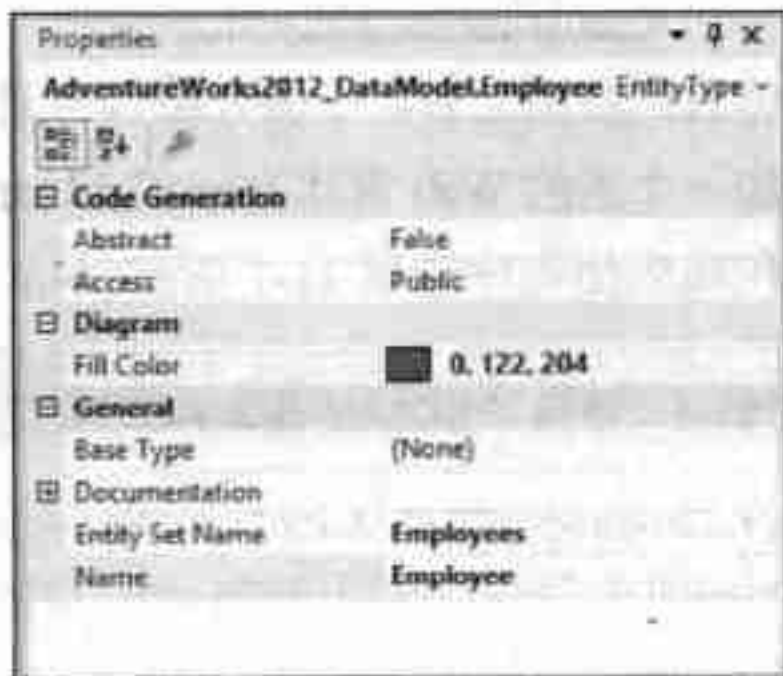


图 11-9

在这里可以修改对象的访问修饰符，为对象提供一些基本的文档。Visual Studio 还提供了一些视图来使用 Entity Framework。创建了.edmx 文件后，就会自动打开 Model Browser 窗口。

另一个重要的视图是 Entity Data Model Mapping Details 窗口。进入这个窗口有两种方式：可以从 Visual Studio 菜单中选择 View | Other Windows | Entity Data Mapping Details 命令，或者在设计器中右击 Employee 对象，从弹出的菜单中选择 Table Mapping 命令。Entity Data Model Browser 和 Entity Data Mapping Details 窗口如图 11-10 所示。

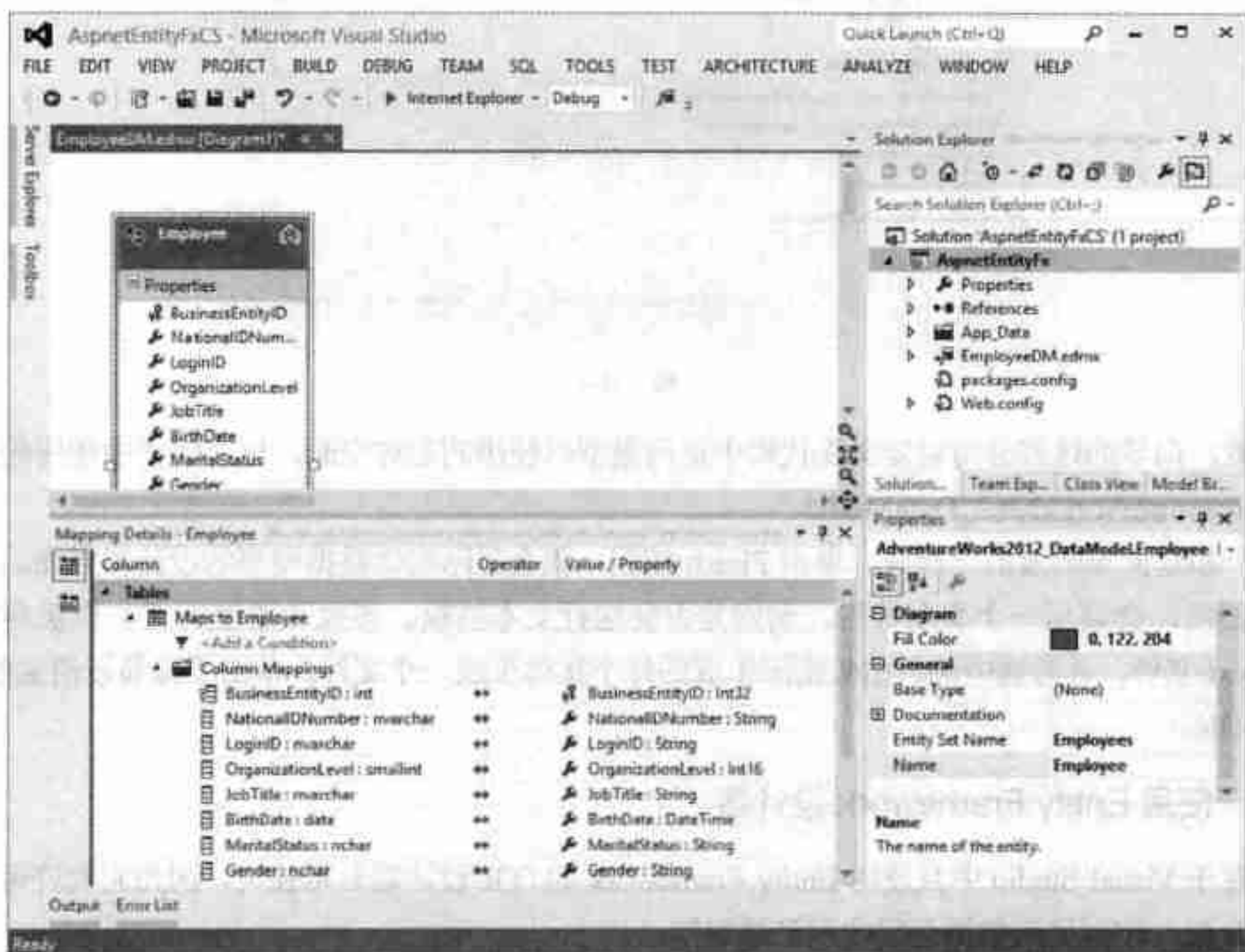


图 11-10

现在已有了这个简单的对象和.edmx 文件，下一步是建立一个小型 ASP.NET 页面来使用实体数据模型。

11.2.3 使用 EDM 建立 ASP.NET Web 页面

有了实体数据模型之后，下面介绍如何建立一个简单的 Web 窗体来使用该模型。第一步是在项目中添加一个新的 Web 窗体 BasicGrid.aspx，在该页面上添加一个 GridView 控件。ASP.NET Web 窗体的代码应如程序清单 11-1 所示(本章下载代码中的 BasicGrid.aspx)。

程序清单 11-1 使用了 EDM 的基本 ASP.NET Web 窗体

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeBehind="BasicGrid.aspx.cs" Inherits="AspnetEntityFx.BasicGrid" %>

<!DOCTYPE html>
```

```

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>My EDM</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:GridView ID="GridView1" runat="server">
            </asp:GridView>
        </div>
    </form>
</body>
</html>

```

这是该 Web 窗体的 C# 版本。注意页面上只有一个 GridView 控件。下面将使用这个控件填充来自刚创建的实体数据模型的结果。

要使用实体数据模型，程序清单 11-2 显示了程序清单 11-1 中的 ASP.NET Web 窗体的隐藏代码页面(本章下载代码中的 BasicGrid.aspx.cs)。

程序清单 11-2 ASP.NET 页面的隐藏代码页面

```

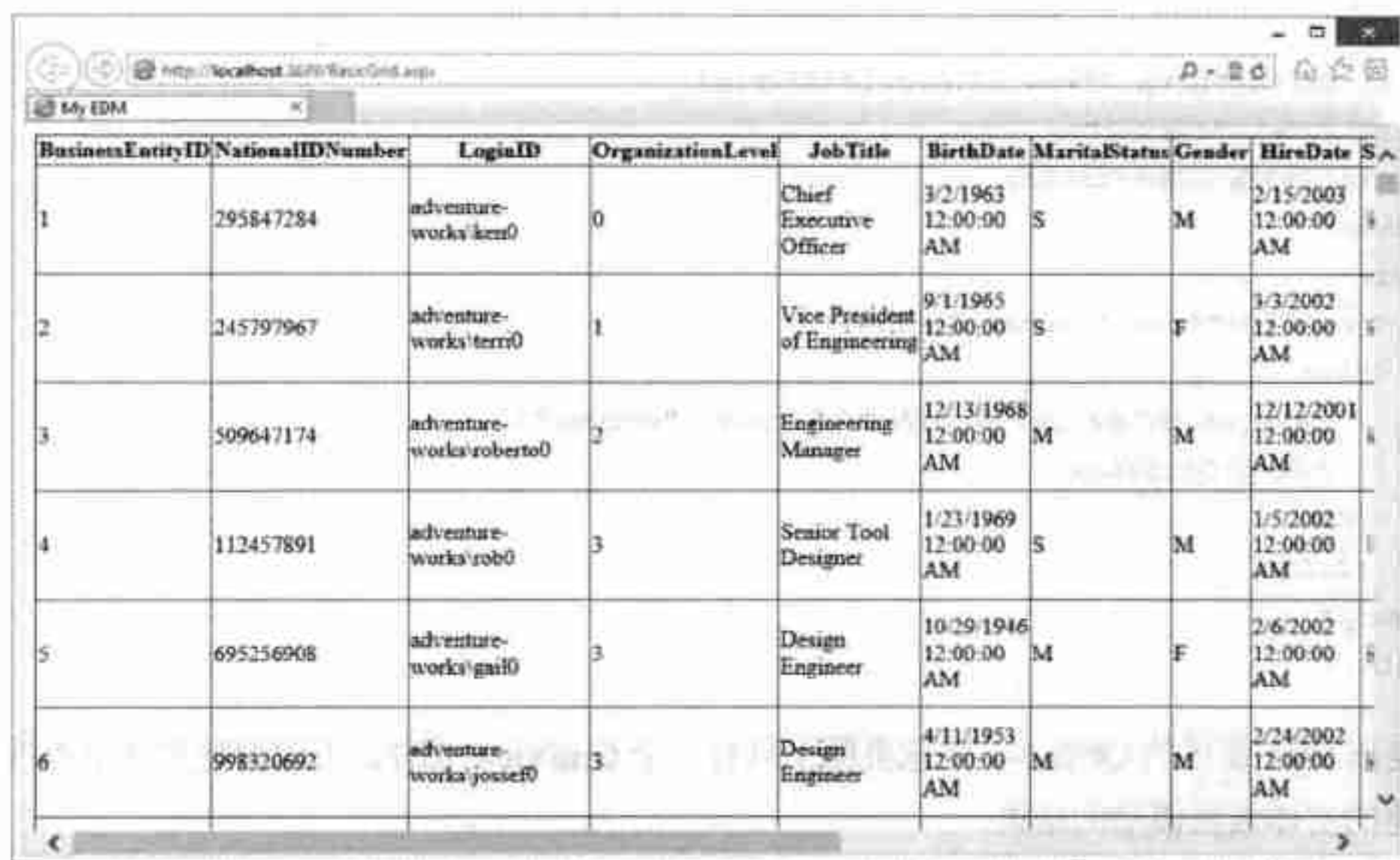
using System;
using System.Linq;

namespace AspNetEntityFx
{
    public partial class BasicGrid : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            AdventureWorks2012_DataEntities adventureWorks2012_DataEntities =
                new AdventureWorks2012_DataEntities();

            var query = from emp in
                adventureWorks2012_DataEntities.Employees
                select emp;
            GridView1.DataSource = query.ToList();
            GridView1.DataBind();
        }
    }
}

```

可以看出，这个页面的代码并不多。运行该 Web 窗体，会生成如图 11-11 所示的结果。



BusinessEntityID	NationalIDNumber	LoginID	OrganizationLevel	JobTitle	BirthDate	MaritalStatus	Gender	HireDate	S
1	295847284	adventure-works\kent0	0	Chief Executive Officer	3/2/1963 12:00:00 AM	S	M	2/15/2003 12:00:00 AM	
2	245797967	adventure-works\terri0	1	Vice President of Engineering	9/1/1965 12:00:00 AM	S	F	3/3/2002 12:00:00 AM	
3	509647174	adventure-works\roberto0	2	Engineering Manager	12/13/1968 12:00:00 AM	M	M	12/12/2001 12:00:00 AM	
4	112457891	adventure-works\rob0	3	Senior Tool Designer	1/23/1969 12:00:00 AM	S	M	1/5/2002 12:00:00 AM	
5	695256908	adventure-works\gail0	3	Design Engineer	10/29/1946 12:00:00 AM	M	F	2/6/2002 12:00:00 AM	
6	998320692	adventure-works\jossef0	3	Design Engineer	4/11/1953 12:00:00 AM	M	M	2/24/2002 12:00:00 AM	

图 11-11

要使用新的实体数据模型，应创建该模型的一个实例：

```
AdventureWorks2012_DataEntities adventureWorks2012_DataEntities = New
AdventureWorks2012_DataEntities();
```

这个实例管理对数据库的连接，并生成要在数据库上执行的查询。下面是一条 LINQ 语句：

```
var query = from emp in
    adventureWorks2012_DataEntities.Employees
    Select emp
```

这里使用了隐式类型化的变量 query。赋予 query 对象的值是 Employees 属性的值，该属性的类型是 IQueryable<Employee>。这个 LINQ 查询只是把 Employee 表中的所有内容提取出来，将其放在 query 对象中，以便在代码中使用。

完成了 LINQ 操作后，就把 query 对象作为数据源绑定到 GridView1 控件：

```
GridView1.DataSource = query.ToList()
GridView1.DataBind()
```

这是一对一映射的简单例子。下一个例子介绍如何使用多对多关系。

11.3 理解关系

前面的例子说明了一对一映射——映射到 Employee 表的 Employee 对象。下面介绍一对一和一对多关系，以及多对一和多对多关系。

11.3.1 一对一和一对多关系

一对一关系是指在实体数据模型中一个表映射到一个类型的结构，也称为 Table per

Type(TPT)模型。

为了更详细地说明一对一关系，我们将从上一个使用 Employee 表的例子开始介绍。

如果查看 AdventureWorks2012_Data.mdf 数据库文件的详细信息，就会发现其中有许多不同的表。就这个例子使用的数据库的 Employee 部分而言，可以看到如图 11-12 所示的数据库关系。

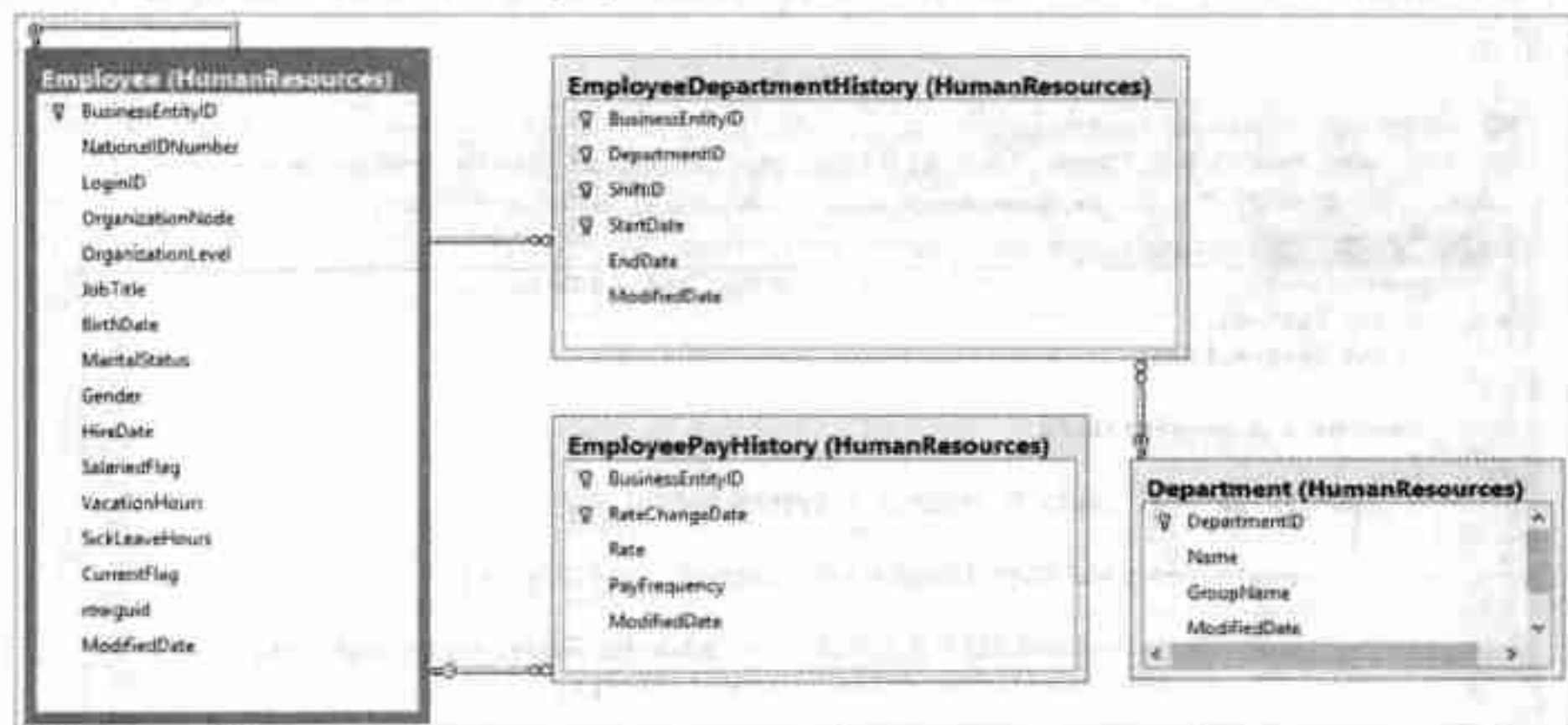


图 11-12

可以看出，除了前面使用的 Employee 表之外，还有其他表，如 EmployeeDepartmentHistory、Department、EmployeeAddress 和 EmployeePayHistory 表，它们有特定的映射。在图 11-12 中，这些表都通过 BusinessEntityID 外键关联起来。

为了与映射进行对比，可以选择在本章前面创建的 EmployeeDM.edmx 文件。在这个文件的设计界面上，右击并选择 Update Model from Database 命令。这会打开 Update Wizard 对话框，如图 11-13 所示。



图 11-13

展开 Tables 节点，添加对遗漏的表的引用，选中 EmployeeDepartmentHistory、Department 和 EmployeePayHistory 表旁边的复选框。从图 11-13 中可以看出，与其他类型的雇员数据建立的是一对多关系。

有了这个结构，通过 IntelliSense 还可以发现，现在有了到每个指定表的类型(或对象)映射，如图 11-14 所示。

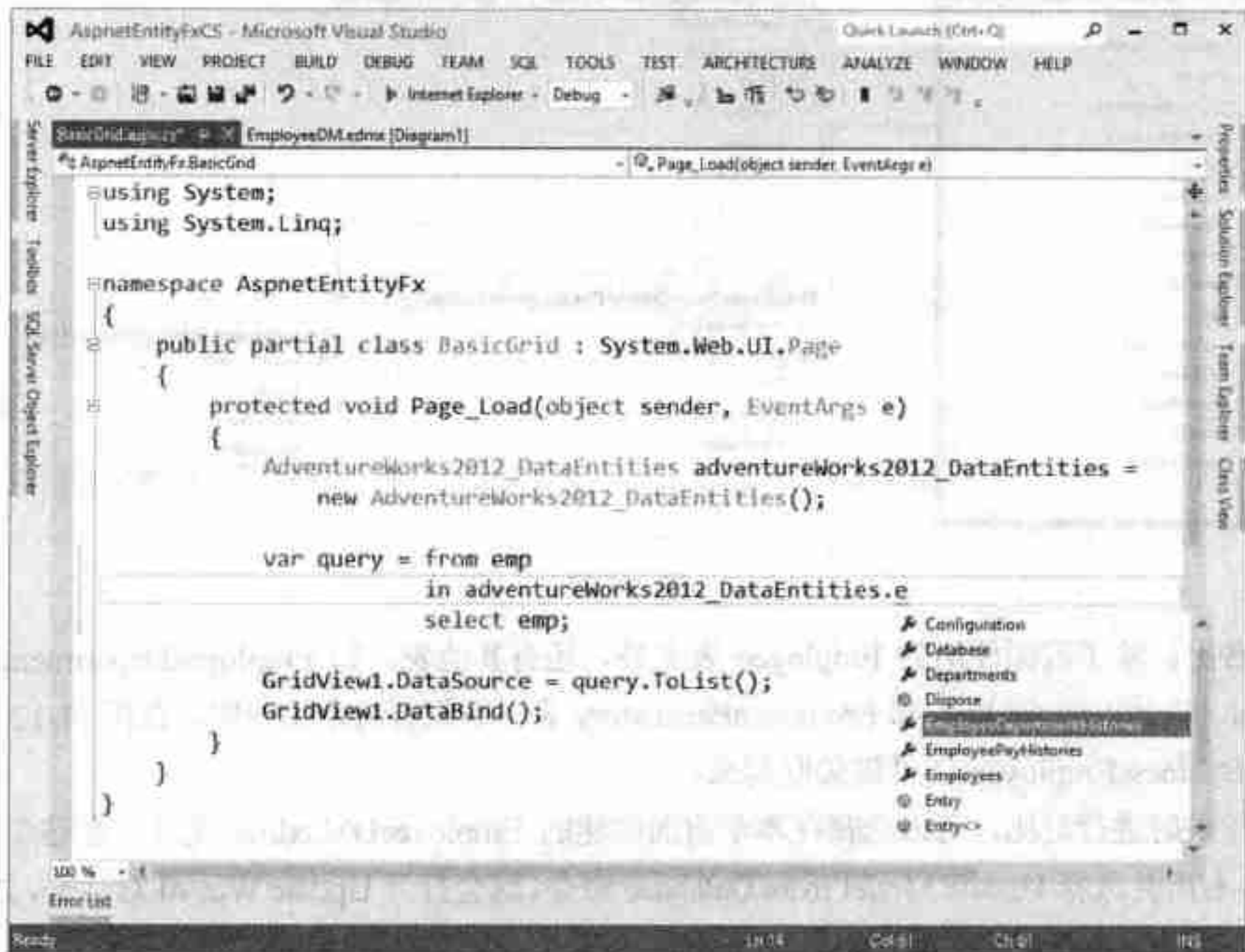


图 11-14

添加了对这些表的引用后，就可以使用所有的对象，如下所述。对于这个例子，创建一个简单的 ASP.NET Web 窗体，它只包含一个 BulletedList 控件，接着在 Web 窗体的隐藏代码中使用程序清单 11-3 中的代码(本章下载代码中的 OneToMany.aspx.cs)。

程序清单 11-3 使用一对多映射

```
using System;
using System.Web.UI.WebControls;

namespace AspNetEntityFx
{
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            AdventureWorks2012_DataEntities adventureWorks2012_DataEntities =
                new AdventureWorks2012_DataEntities();

            foreach (var employee in adventureWorks2012_DataEntities.Employees)
```

```

    {
        ListItem li = new ListItem();

        li.Text = employee.BusinessEntityID + " ";

        foreach (var pay in employee.EmployeePayHistories)
        {
            li.Text += "Pay Rate: " + pay.Rate + " ";
        }

        BulletedList1.Items.Add(li);
    }
}
}
}

```

首先访问 `Employee` 对象，此时还没有加载其他对象。第一次访问 `EmployeePayHistory` 对象时，如果还没有加载，就会自动加载。不再需要像架构的以前版本那样明确加载相关的对象。

运行这些代码，会得到如图 11-15 所示的结果。

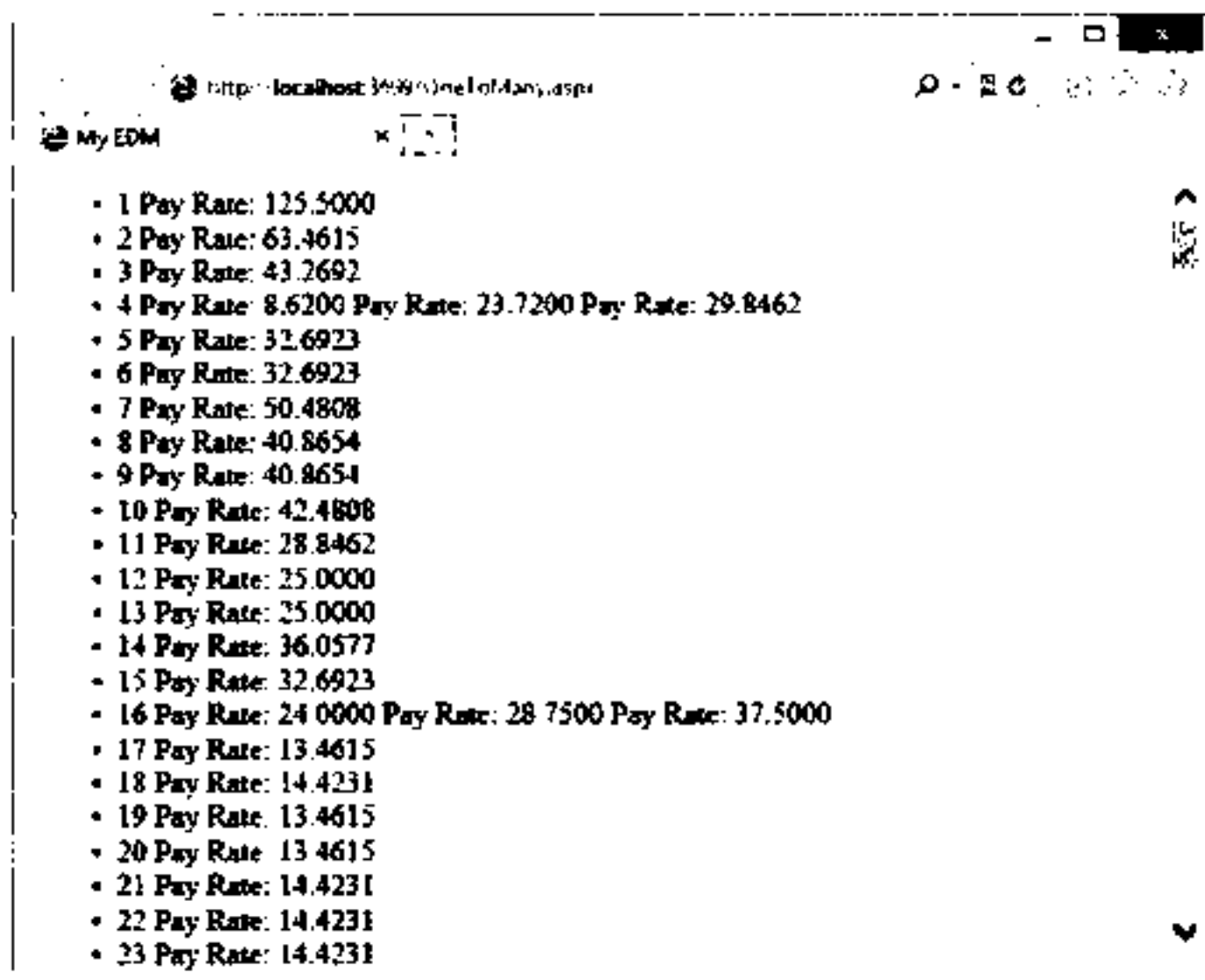


图 11-15

11.3.2 多对一和多对多关系

除了一对一和一对多关系之外，Entity Framework 还支持多对一和多对多关系。在这两种关系中，Entity Framework 会在查询数据库时执行相应的表连接操作。

创建一个新的实体数据模型(Sales.edmx 文件)，其中包含 `Customer`、`SalesTerritory`、`SalesOrderHeader` 和 `SalesOrderDetail` 表，该模型如图 11-16 所示。

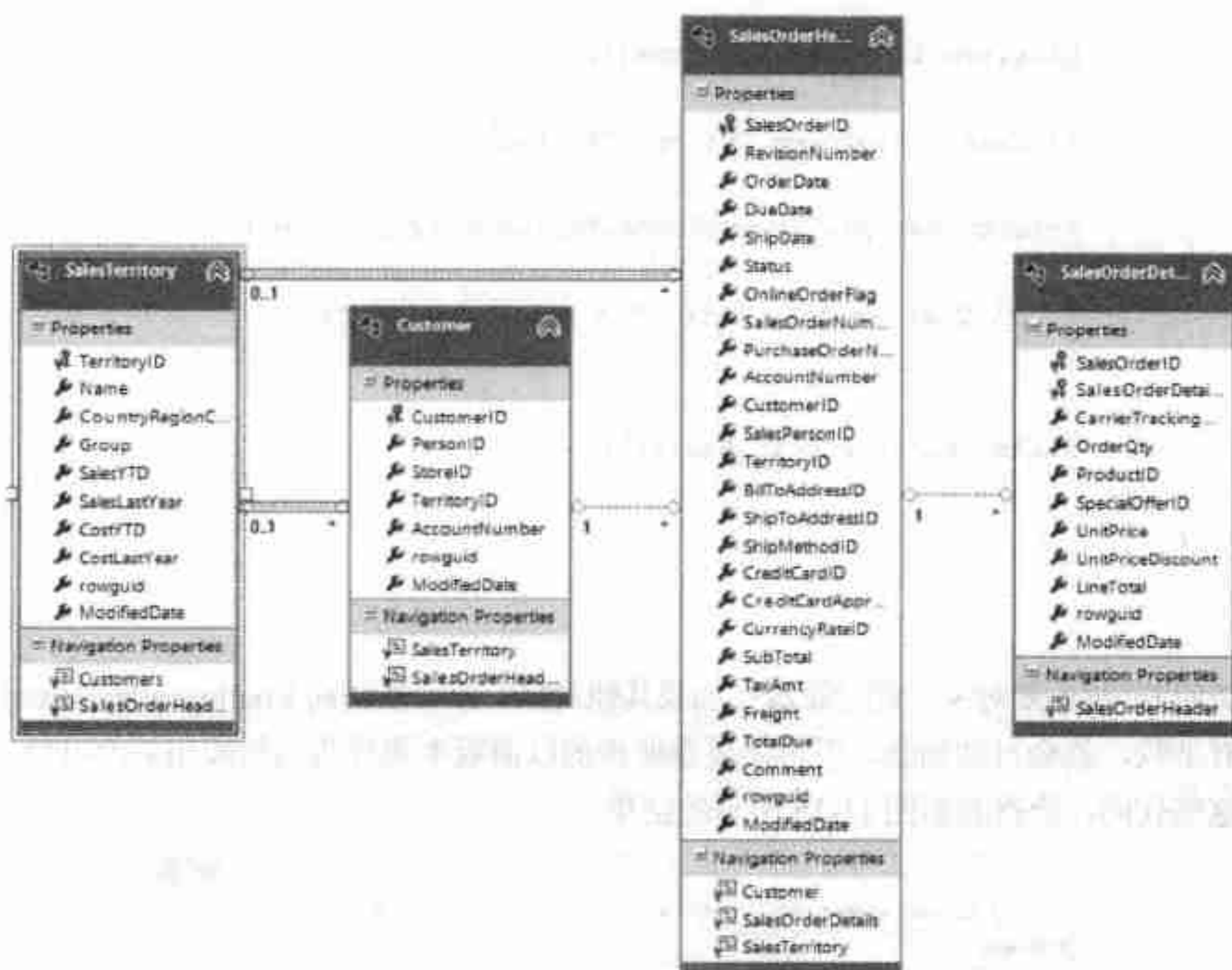


图 11-16

查看设计器中连接可视化对象的线，就可以了解对象之间的关系。如果线的一端有星号，就表示“多”。如果线的另一端有数字“1”，如 SalesOrderHeader 和 Customer 对象之间的线，就表示这是多对一关系。在设计器中突出显示关系线，就会在 Visual Studio 的 Properties 窗口中显示该关系的细节，通过窗口中的两个 End 属性值显示这些细节。

下面的页面执行一些跨表连接操作，这个操作如程序清单 11-4 所示(本章下载代码中的 ManyToMany.aspx.cs)。对于这个例子，只需保留仅包含 GridView 控件的简单 Web 窗体，并使用下面所示的隐藏代码。

程序清单 11-4 让 ADO.NET Entity Framework 在表之间执行连接操作

```
using System;
using System.Linq;

namespace AspNetEntityFx
{
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            AdventureWorks2012_DataEntities1 adventureWorks2012_DataEntities =
                new AdventureWorks2012_DataEntities1();

            var query = from o in
```

```

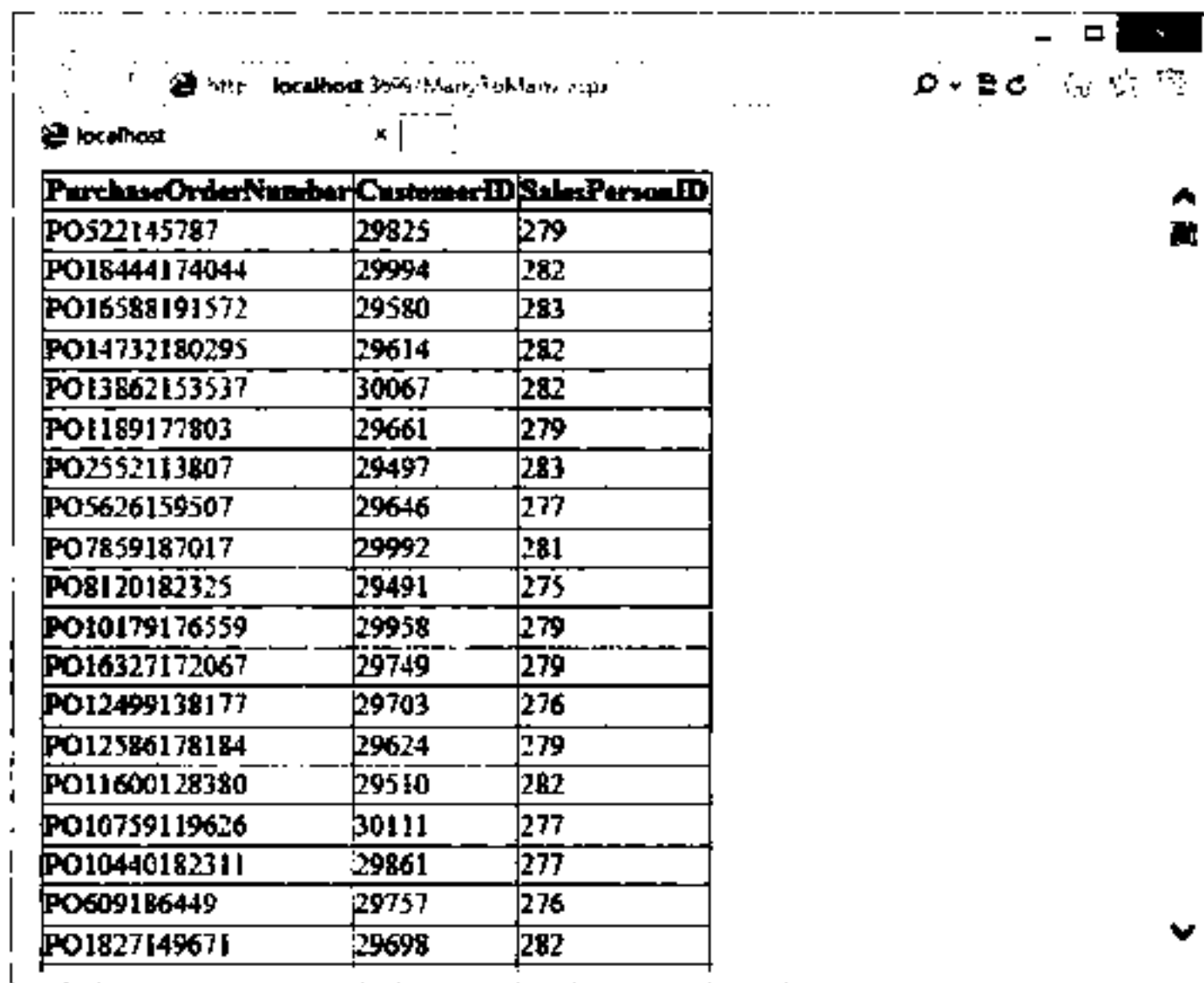
        adventureWorks2012_DataEntities.SalesOrderHeaders
        where o.SalesOrderDetails.Any(Quantity =>
            Quantity.OrderQty > 5)
        select new {o.PurchaseOrderNumber,
            o.Customer.CustomerID, o.SalesPersonID};

    GridView1.DataSource = query.ToList();
    GridView1.DataBind();
}
}
}

```

这个查询从 3 个不同的表中提取内容，Entity Framework 会自动在表之间建立相应的连接。

在这个例子中处理了 SalesOrderHeader 表中订单数量超过 5 的所有项。在选择出来的项中，从几个表中为动态对象提取字段。最后，再次把结果绑定到一个标准的 GridView 控件，最终结果如图 11-17 所示。



PurchaseOrderNumber	CustomerID	SalesPersonID
PO522145787	29825	279
PO18444174044	29994	282
PO16588191572	29580	283
PO14732180295	29614	282
PO13862153537	30067	282
PO1189177803	29661	279
PO2552113807	29497	283
PO5626159507	29646	277
PO7859187017	29992	281
PO8120182325	29491	275
PO10179176559	29958	279
PO16327172067	29749	279
PO12499138177	29703	276
PO12586178184	29624	279
PO11600128380	29510	282
PO10759119626	30111	277
PO10440182311	29861	277
PO609186449	29757	276
PO1827149671	29698	282

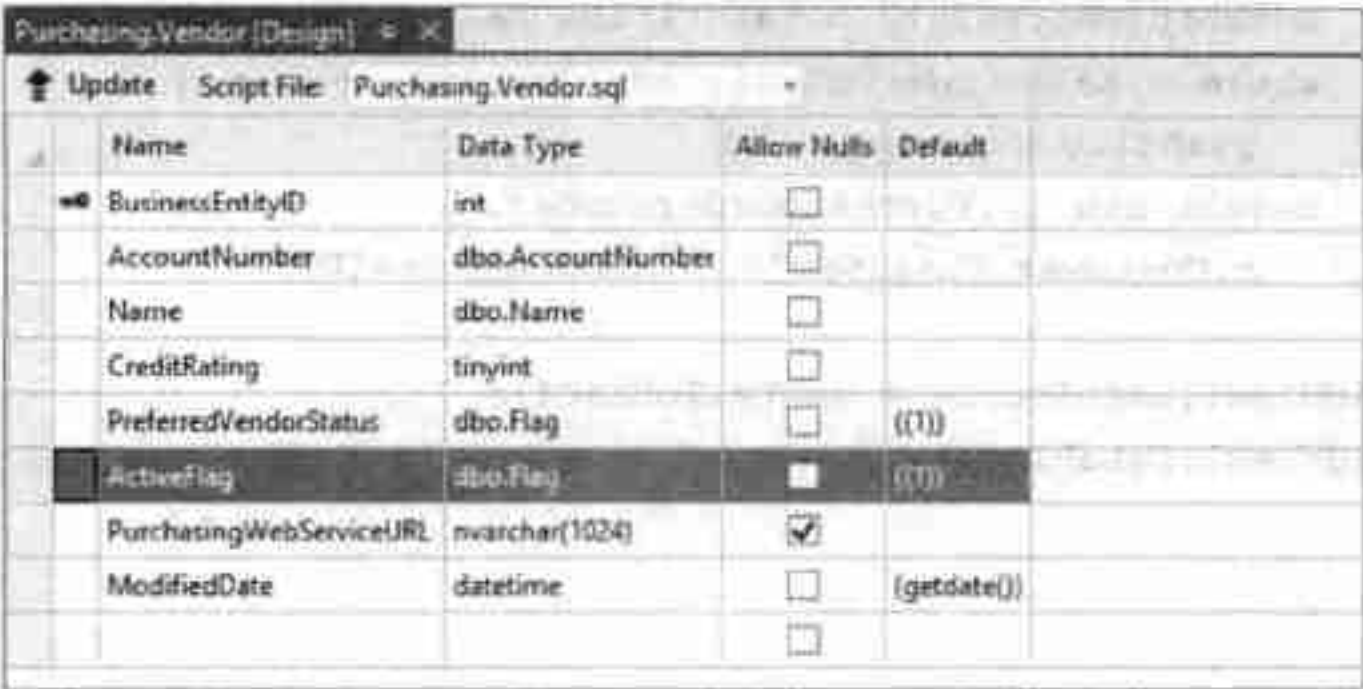
图 11-17

11.4 在 EDM 中使用继承功能

在构建实体数据模型时可以进行继承，继承方式与处理 CLR 中的对象时一样简单。

通过继承可以使用和查找专门的对象。例如，可以修改 Customer 对象，建立一个查询，按照对象引用来查找特定类型的顾客，而不是按值来查找顾客。

创建一个新的实体数据模型(Vendor.edmx 文件)，它只包含 Vendor 表(Purchasing 部分)。在 Visual Studio 中打开 Vendor 表的定义，如图 11-18 所示。



Name	Data Type	Allow Nulls	Default
BusinessEntityID	int	<input type="checkbox"/>	
AccountNumber	dbo.AccountNumber	<input type="checkbox"/>	
Name	dbo.Name	<input type="checkbox"/>	
CreditRating	tinyint	<input type="checkbox"/>	
PreferredVendorStatus	dbo.Flag	<input type="checkbox"/>	((1))
ActiveFlag	dbo.Flag	<input checked="" type="checkbox"/>	((1))
PurchasingWebServiceURL	nvarchar(1024)	<input checked="" type="checkbox"/>	
ModifiedDate	datetime	<input type="checkbox"/>	(getdate())

图 11-18

从图 11-18 中可以看出，ActiveFlag 属性的类型是 bit，这表示该属性的值只能是表示真或假的 1 或 0。对于这个例子，应建立专门的类型来引用不活跃的供应商，以便区分活跃的和 not 活跃的这两种类型。

有了 Vendor 表之后，右击设计器界面，从弹出的菜单中选择 Add | Entity 命令，打开 Add Entity 对话框。

这个对话框中提供了名为 InactiveVendor 的实体，并使其继承基类 Vendor。最终，Add Entity 对话框应如图 11-19 所示。

这一步会给映射添加可视化表示，如图 11-20 所示。



图 11-19



图 11-20

下一步是把两个对象进一步关联起来，并为它们的关系提供一些逻辑。为了完成这项任务，首先从 Vendor 实体对象中删除 ActiveFlag 标量属性，因为这个例子不需要它。

之后，在设计器中突出显示 Vendor 对象，在 Visual Studio 的 Mapping Details 窗口中查看这个对象的映射细节。在这个窗口中添加一个 ActiveFlag 属性值等于 1 的条件，如图 11-21 所示。

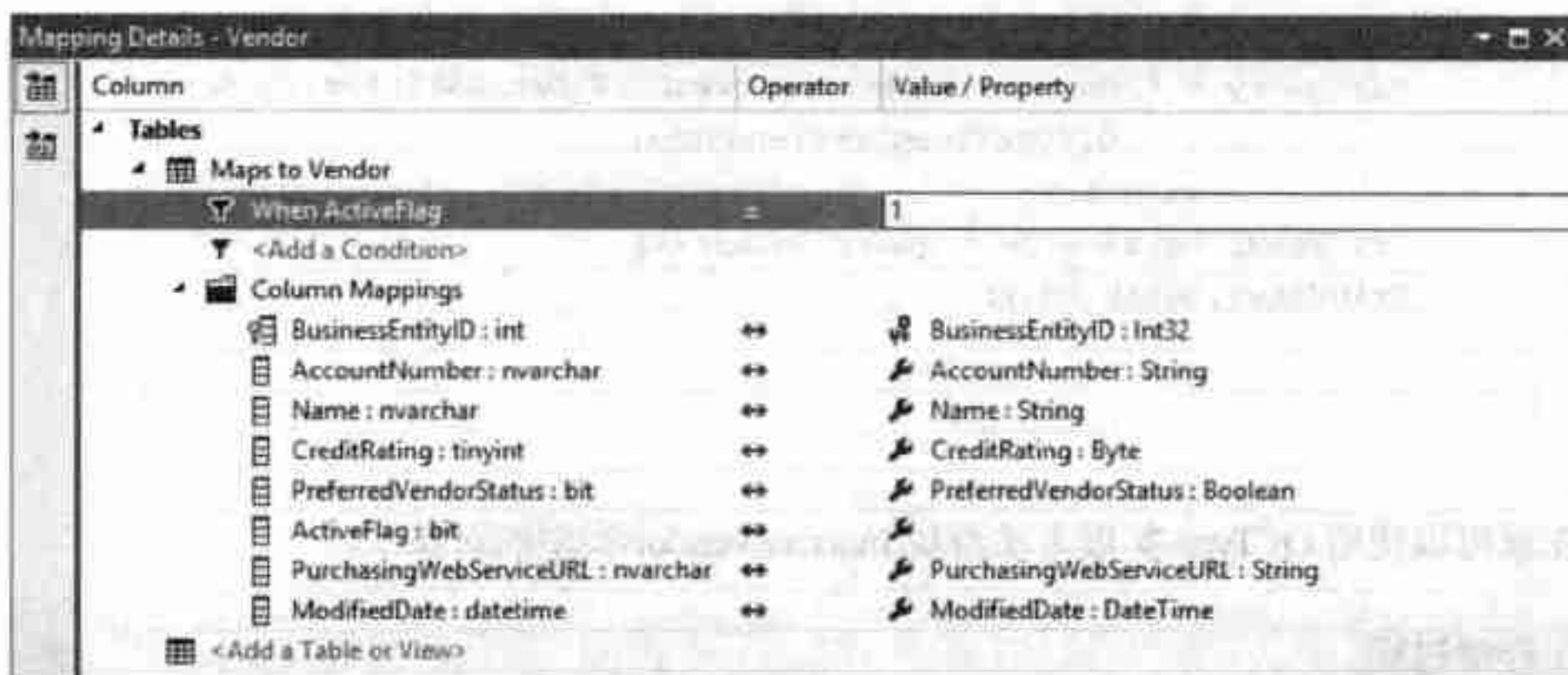


图 11-21

该设置表示，如果 `ActiveFlag` 属性的值是 1，那么对象的类型就是 `Vendor`。下一步是建立 `InactiveVendor` 对象。要在 Mapping Details 窗口中查看这个对象，首先应在 Mapping Details 窗口中添加 `Vendor` 表，之后创建一个 `ActiveFlag` 属性值等于 0 的条件，如图 11-22 所示。

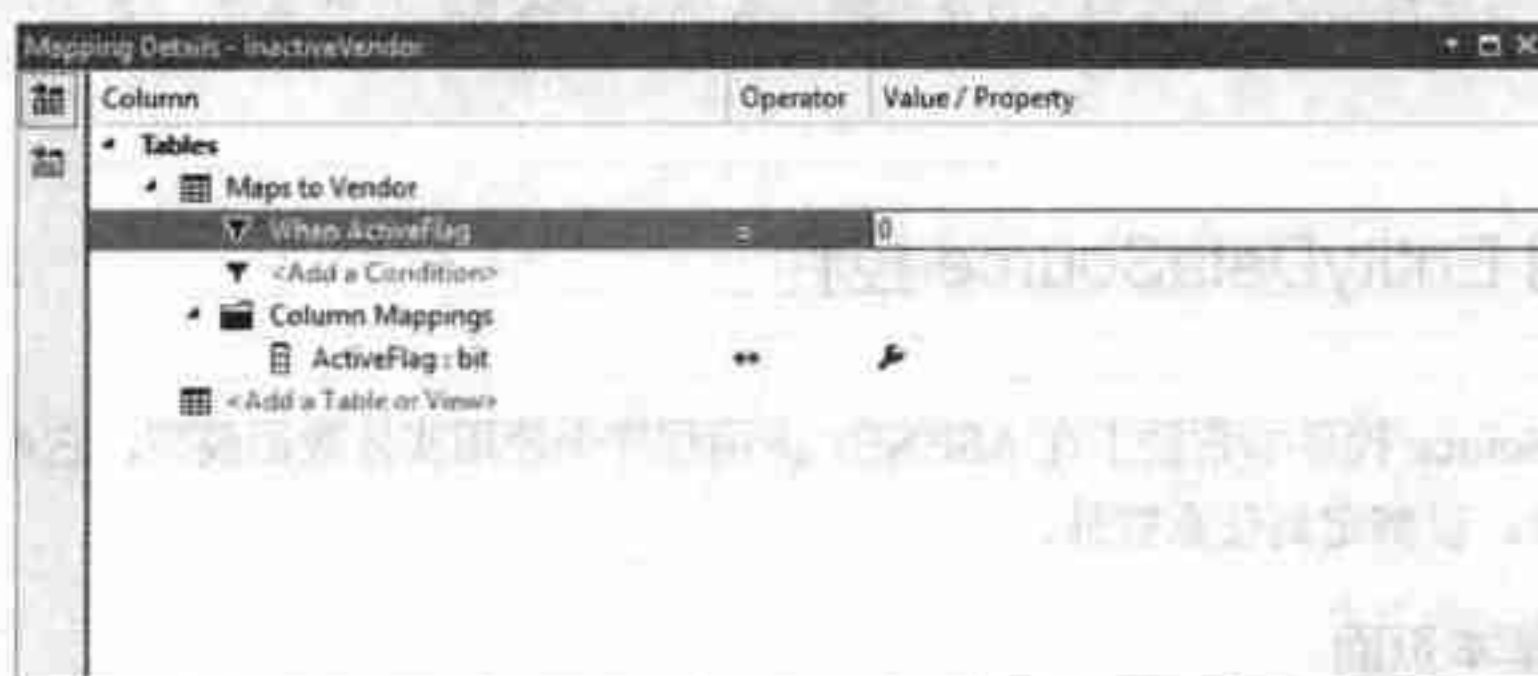


图 11-22

完成这些工作后就可以确定，如果 `ActiveFlag` 属性的值是 1，对象在实体数据模型中的类型就是 `Vendor`。但如果 `ActiveFlag` 属性的值是 0，对象的类型就是 `InactiveVendor`。可以在代码中使用该逻辑，如程序清单 11-5 所示(本章下载代码中的 `Inheritance.aspx.cs`)。

程序清单 11-5 在实体数据模型中使用继承功能

```
using System;
using System.Linq;

namespace AspNetEntityFx
{
    public partial class Inheritance : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            AdventureWorks2012_DataEntities3 adventureWorks2012_DataEntities =
                new AdventureWorks2012_DataEntities3();
        }
    }
}
```

```

        var query = from v in adventureWorks2012_DataEntities.Vendors
                     .OfType<InactiveVendor>()
                     select v;
        GridView1.DataSource = query.ToList();
        GridView1.DataBind();
    }
}

```

现在就可以使用 Of Type 扩展方法查找 InactiveVendor 类型的对象了。

使用存储过程

如果在使用 EF 或其他 ORM 技术之前就处理数据访问,就可能非常依赖存储过程。该技术要求舍弃以前建立的、准备用于应用程序的大量存储过程吗?并没有如此要求,因为存储过程的使用与本章前面介绍的其他对象一样简单。许多开发新应用程序的开发人员不再大量使用过去常见的存储过程,而是使用 EF 自动生成的命令。本章不介绍存储过程的使用,但要注意 EF 支持使用存储过程。如果有以前的存储过程,或者因为标准或兼容的缘故只能使用存储过程,EF 仍是数据访问的有效选项。

11.5 使用 EntityDataSource 控件

EntityDataSource 控件非常便于在 ASP.NET 应用程序中使用实体数据模型,它可以自动执行必要的 LINQ 工作,以绑定到任意控件。

11.5.1 创建基本页面

对于这个使用 EntityDataSource 控件的例子,应返回 Sales.edmx 文件,并使用该模型中的 Customer 对象。

首先创建一个使用该模型的 ASP.NET Web 窗体。在 ASP.NET Web Application 项目中创建这个 Web 窗体,并使用程序清单 11-6 所示的代码(本章下载代码中的 EntityDataSource.aspx)。

程序清单 11-6 使用 EntityDataSource 服务器控件

```

<%@ Page Language="C#" AutoEventWireup="true"
    CodeBehind="EntityDataSource.aspx.cs" Inherits="AspnetEntityFx.EntityDataSource" %>

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title> </title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:GridView ID="GridView1" runat="server">
        </asp:GridView>
    </form>
</body>

```

```

<br />
<asp:EntityDataSource ID="EntityDataSource1" runat="server">
</asp:EntityDataSource>
</form>
</body>
</html>

```

有了这个页面后，就可以配置页面上的 EntityDataSource 控件，以使用前面创建的实体数据模型，接着可以把 GridView 控件绑定到这个数据源控件。

11.5.2 配置数据源控件

接着，配置页面上的数据源控件，使其能为前面创建的实体数据模型工作。为此，可以直接在 ASP.NET Web 窗体的代码中编码 EntityDataSource 控件，也可以使用数据源配置向导。对于这个例子，我们使用的是向导。

在页面上突出显示数据源控件，从可用选项中找到 Configure Data Source 链接。单击该链接，显示的第一个界面(如图 11-23 所示)要求配置ObjectContext。

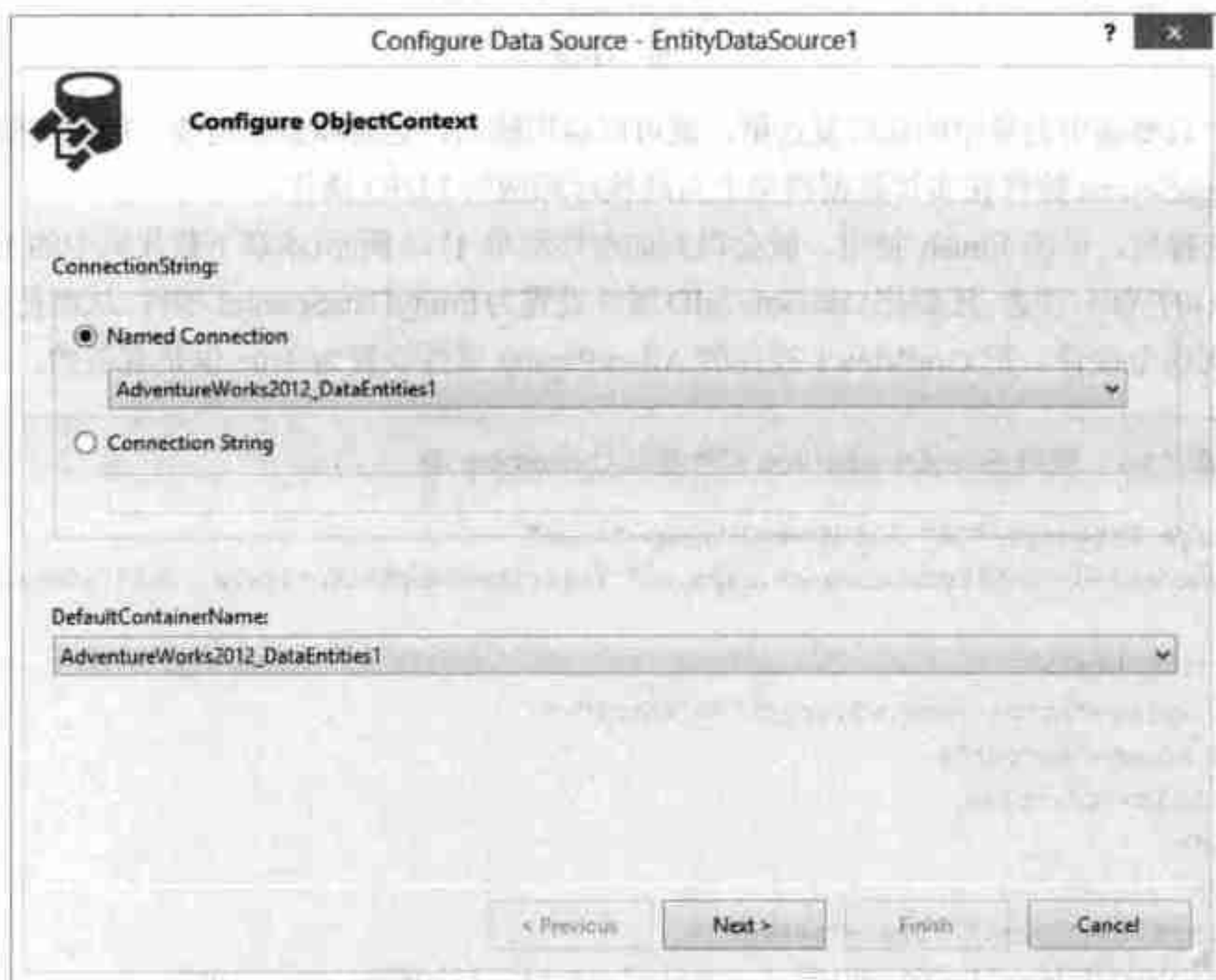


图 11-23

对于这个例子，使用前面创建的 AdventureWorks2012_DataEntities 对象。单击 Next 按钮，配置数据选择过程。在这个例子中，选择 Customers 表，如图 11-24 所示。

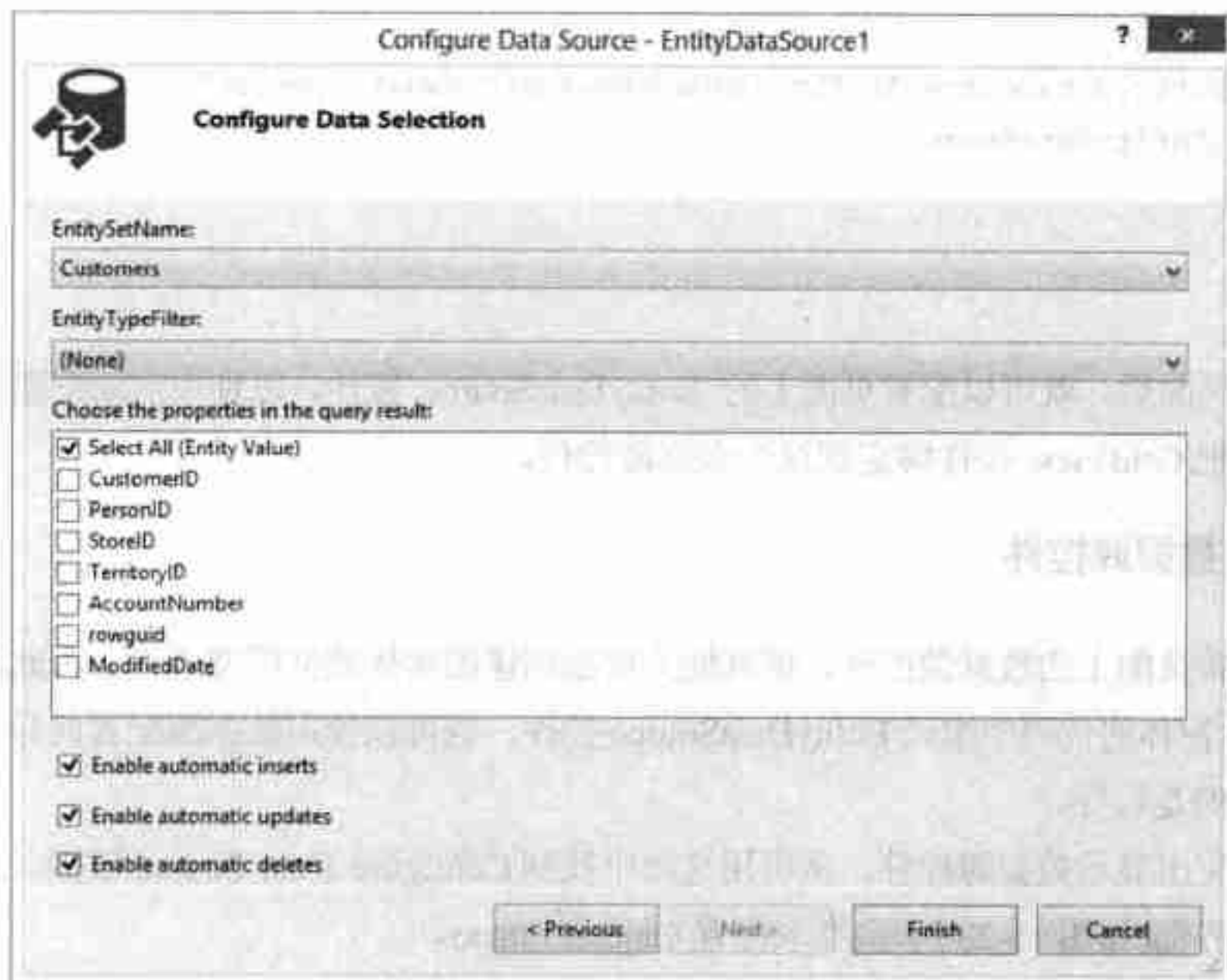


图 11-24

注意，只要选中向导中的相应复选框，就可以启用插入、更新和删除功能。启用这些功能就允许 EntityDataSource 控件在实体数据模型上自动执行相应的 LINQ 操作。

完成选择后，单击 Finish 按钮，就会得到如程序清单 11-7 所示(本章下载代码中的 EntityDataSource.aspx)的代码。注意，还要把 DataSourceID 属性设置为 EntityDataSource1 控件，从而把 GridView1 控件绑定到这个控件。把 GridView1 控件的 AllowPaging 属性设置为 True 也是有益的。

程序清单 11-7 使用 EntityDataSource 控件提取 Customers 表

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeBehind="EntityDataSource.aspx.cs" Inherits="AspNetEntityFx.EntityDataSource" %>

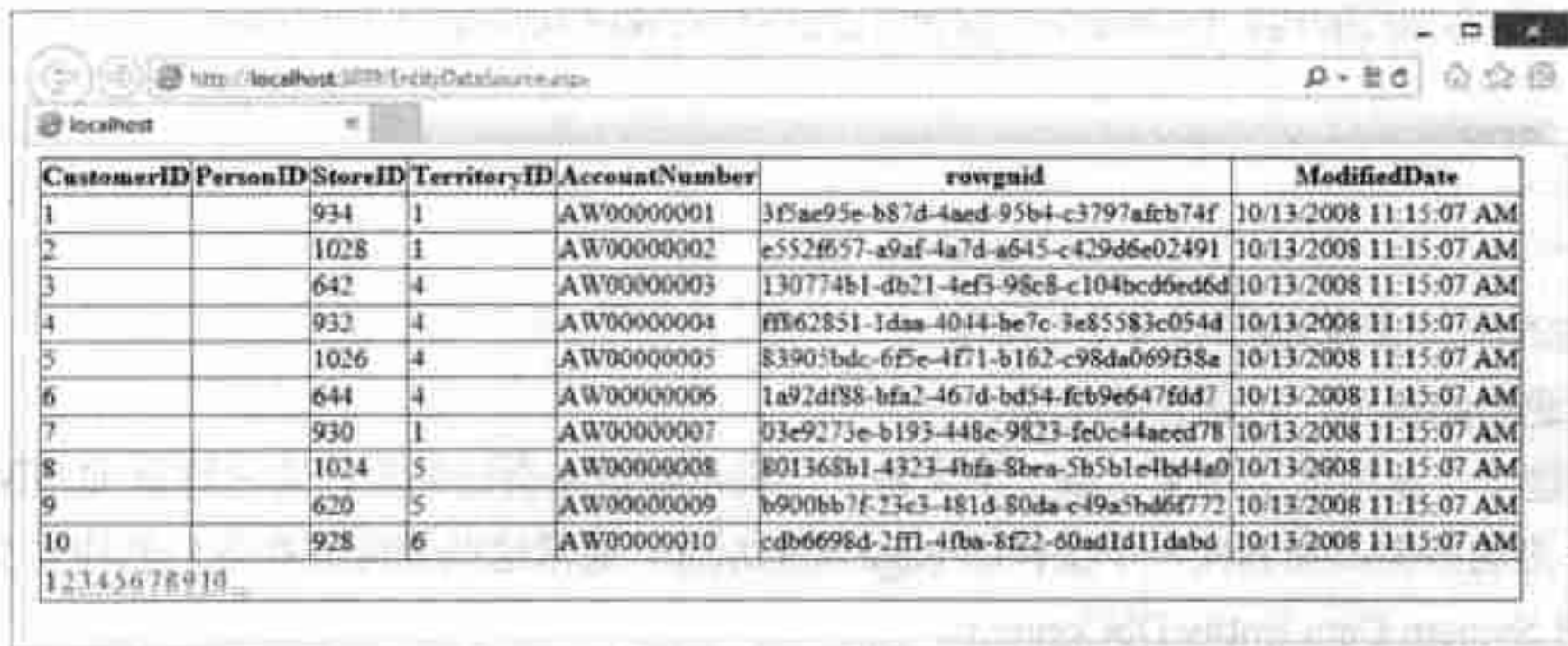
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title> </title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:GridView ID="GridView1" runat="server" AllowPaging="True"
            DataSourceID="EntityDataSource1">
        </asp:GridView>
        <asp:EntityDataSource ID="EntityDataSource1" runat="server"
            ConnectionString="name=AdventureWorks2012_DataEntities1NorthwindEntities"
            DefaultContainerName="AdventureWorks2012_DataEntities1NorthwindEntities"
            EnableDelete="True" EnableFlattening="False"
            EnableInsert="True"
            EnableUpdate="True" EntitySetName="Customers">
```

```

</asp:EntityDataSource>
</form>
</body>
</html>

```

运行这个页面，生成的结果如图 11-25 所示。在结果中，可以查看实体数据模型的各项。



CustomerID	PersonID	StoreID	TerritoryID	AccountNumber	rowguid	ModifiedDate
1		934	1	AW00000001	3f5ae95e-b87d-4aed-95b4-c3797afcb74f	10/13/2008 11:15:07 AM
2		1028	1	AW00000002	e552f657-a9af-4a7d-a645-c429d6e02491	10/13/2008 11:15:07 AM
3		642	4	AW00000003	130774b1-db21-4ef3-98c8-c104bcd6ed6d	10/13/2008 11:15:07 AM
4		932	4	AW00000004	ff862851-1daa-4044-be7c-3e85583c054d	10/13/2008 11:15:07 AM
5		1026	4	AW00000005	83905bdc-6f5e-4f71-b162-c98da069f38a	10/13/2008 11:15:07 AM
6		644	4	AW00000006	1a92df88-bfa2-467d-bd54-fcb9e647fdd7	10/13/2008 11:15:07 AM
7		930	1	AW00000007	03e9273e-b193-448c-9823-fe0c44aecd78	10/13/2008 11:15:07 AM
8		1024	5	AW00000008	801368b1-4323-4bfa-8bea-5b5b1e4bd4a0	10/13/2008 11:15:07 AM
9		620	5	AW00000009	b900bb7f-23c3-481d-80da-c49a5bd6f772	10/13/2008 11:15:07 AM
10		928	6	AW00000010	cdb6698d-2ff1-4fba-8f22-d0ad1d11dabd	10/13/2008 11:15:07 AM

图 11-25

11.6 Entity Framework Code First

Code First 最初在 Entity Framework 的 4.1 版本中引入。Code First 工作流是处理数据和模型的一种以代码为中心的方法。它使用设计器界面，为 Database First 和 Model First 开发工作流提供了一种替代方法。Code First 使用约定-配置方法，允许只考虑使用 C# 类或 VB 类定义模型。接着这些类可以映射到已有的数据库，或者用于生成一个模式，该模式可以用于生成新数据库。如果需要其他配置，例如更高级的映射，就可以使用数据注释或流畅的 API，提供额外的细节。

Code First Migrations 在 Entity Framework 的 4.3.1 版本中引入。在引入 Migrations 之前，修改 Code First 模型意味着删除数据库，再重建。在开发应用程序的过程中，这是可以接受的，但一旦把应用程序发布到产品中，每次修改模型，就要删除数据库，再重建就是无法接受的。Code First 允许在代码中表示模型，同样，Code First Migrations 也允许在代码中表示数据库模式迁移。每次修改模型，并希望把修改发布到数据库中时，就创建迁移。迁移都是类，表示自从上一次迁移以来对模型的修改。因此，可以在项目中直接维护数据库修改的历史。

11.6.1 创建 Code First 模型

在项目中添加新的 Web 窗体 CodeFirst.aspx。在页面上添加一个 GridView 控件。ASP.NET 页面的 C# 代码应如程序清单 11-8 所示(本章下载代码的 CodeFirst.aspx)。

程序清单 11-8 使用 Code First 模型的基本页面

```

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="CodeFirst.aspx.cs"
Inherits="AspNetEntityFX.CodeFirst" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

```



```

<head runat="server">
    <title>Code First Model</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:GridView ID="GridView1" runat="server"></asp:GridView>
        </div>
    </form>
</body>
</html>

```

在 ASP.NET 代码隐藏页面中, 需要:

(1) 创建初始模型, 它包含两个类。

(2) 创建一个派生的上下文。该上下文是一个表示数据库会话的类, 它允许查询和保存数据。

对于本例, 将这些类添加到同一个文件的 Page 类的后面。通常这些类应放在各自的独立文件中。上下文派生自 `System.Data.Entity.DbContext`。

(3) 对于模型中的每个类, 都创建一个 `DbSet`。这告诉 Entity Framework 哪些实体要映射到数据库。

(4) 接着, 给 `Page_Load` 方法添加一些代码, 给数据库添加一些数据, 再显示这些数据。

程序清单 11-9(本章下载代码中的 `CodeFirst.aspx.cs`)显示了添加模型类、上下文和使用它们的代码后的代码隐藏页面。

程序清单 11-9 Code First ASP.NET 页面的代码隐藏页面

```

using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;

namespace AspNetEntityFX
{
    public partial class CodeFirst : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            using (var context = new TeamContext())
            {
                var team = new Team { TeamName = "Team 1" };
                context.Teams.Add(team);
                team = new Team { TeamName = "Team 2" };
                context.Teams.Add(team);
                team = new Team { TeamName = "Team 3" };
                context.Teams.Add(team);
                context.SaveChanges();

                var query = from t in context.Teams
                           select t;

                GridView1.DataSource = query.ToList();
                GridView1.DataBind();
            }
        }
    }
}

```



```

    }

    }
}

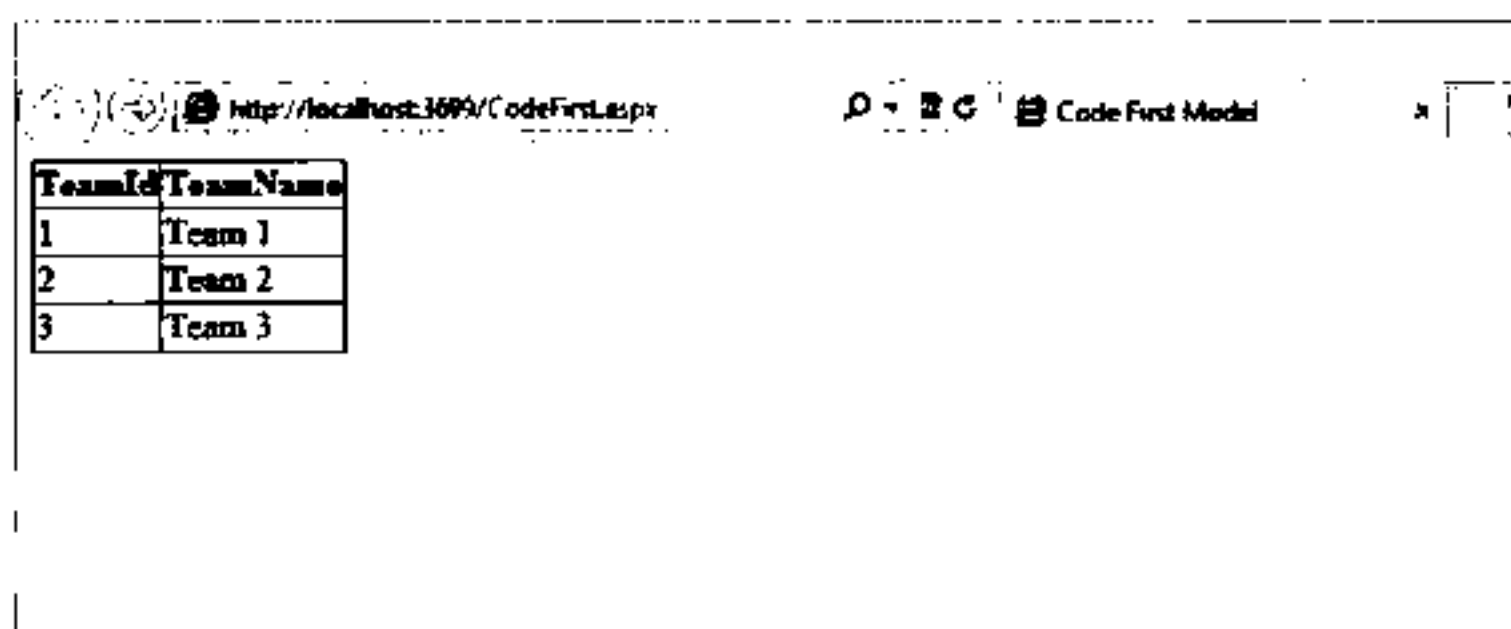
public class Team
{
    public int TeamId { get; set; }
    public string TeamName { get; set; }
}

public class Player
{
    public int PlayerId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
}

public class TeamContext : DbContext
{
    public DbSet<Team> Teams { get; set; }
    public DbSet<Player> Players { get; set; }
}
}

```

这些就是让 Entity Framework 根据模型创建数据库所需的全部代码。注意不必创建与数据库的连接，甚至不需要引用连接字符串，这是因为采用了 Code First 的约定-配置方法。执行页面，结果如图 11-26 所示。



TeamId	TeamName
1	Team 1
2	Team 2
3	Team 3

图 11-26

为了告诉 Entity Framework 把模型类映射到数据库，为每个模型类创建 `DbSet<EntityType>`：

```

public DbSet<Team> Teams { get; set; };
public DbSet<Player> Players { get; set; };

```

为了使用新模型，应在所创建的数据库中添加一些数据。只需创建类的一个新实例，把它添加到 Teams DbSet 中，告诉数据库上下文，将修改保存到数据库中：

```

var team = new Team { TeamName = "Test 3" };
context.Teams.Add(team);
context.SaveChanges();

```

现在可以使用数据库上下文查询数据了，就像使用 Database First 设计器 workflow 处理所创建的

EDM 一样:

```
var query = from t in context.Teams select t
```

从 LINQ 查询返回 IQueryable 后, 就把数据绑定到 GridView 控件:

```
GridView1.DataSource = query.ToList();
GridView1.DataBind();
```

11.6.2 约定-配置

在前面的例子中, DbContext 根据默认约定创建了一个数据库。该数据库在用户第一次建立到数据库的连接时创建。

确定在哪里创建数据库取决于在机器上安装了哪个 SQL Express 版本。如果在机器上运行的是本地 SQL Express 实例, Code First 就把数据库安装在本地。默认情况下, SQL Express 随 Visual Studio 2010 一起安装。而 Visual Studio 2012 默认安装 SQL Express 的新版本 SQL Express LocalDB。如果 Code First 找不到 SQL Express 的实例, 数据库就使用 LocalDB 安装。

无论使用什么数据库, 数据库都使用派生上下文的完整限定名称来命名。在这个例子中, 数据库的名称是 AspNetEntityFX.TeamContext。

使用 Visual Studio 中的 Server Explorer 可以查看创建的数据库。添加一个新的数据连接。对于服务器名, 如果数据库使用 LocalDB 创建, 就指定 (LocalDB)\v11.0; 如果数据库使用 SQL Express 创建, 就指定 \SQLEXPRESS。把数据库名指定为派生上下文的完整限定名称。图 11-27 显示了完成的 Add Connection 信息。

添加好到数据库的数据连接后, 展开数据库表, 查看 Code First 创建的模式。图 11-28 显示了 Code First 创建的数据库模式。

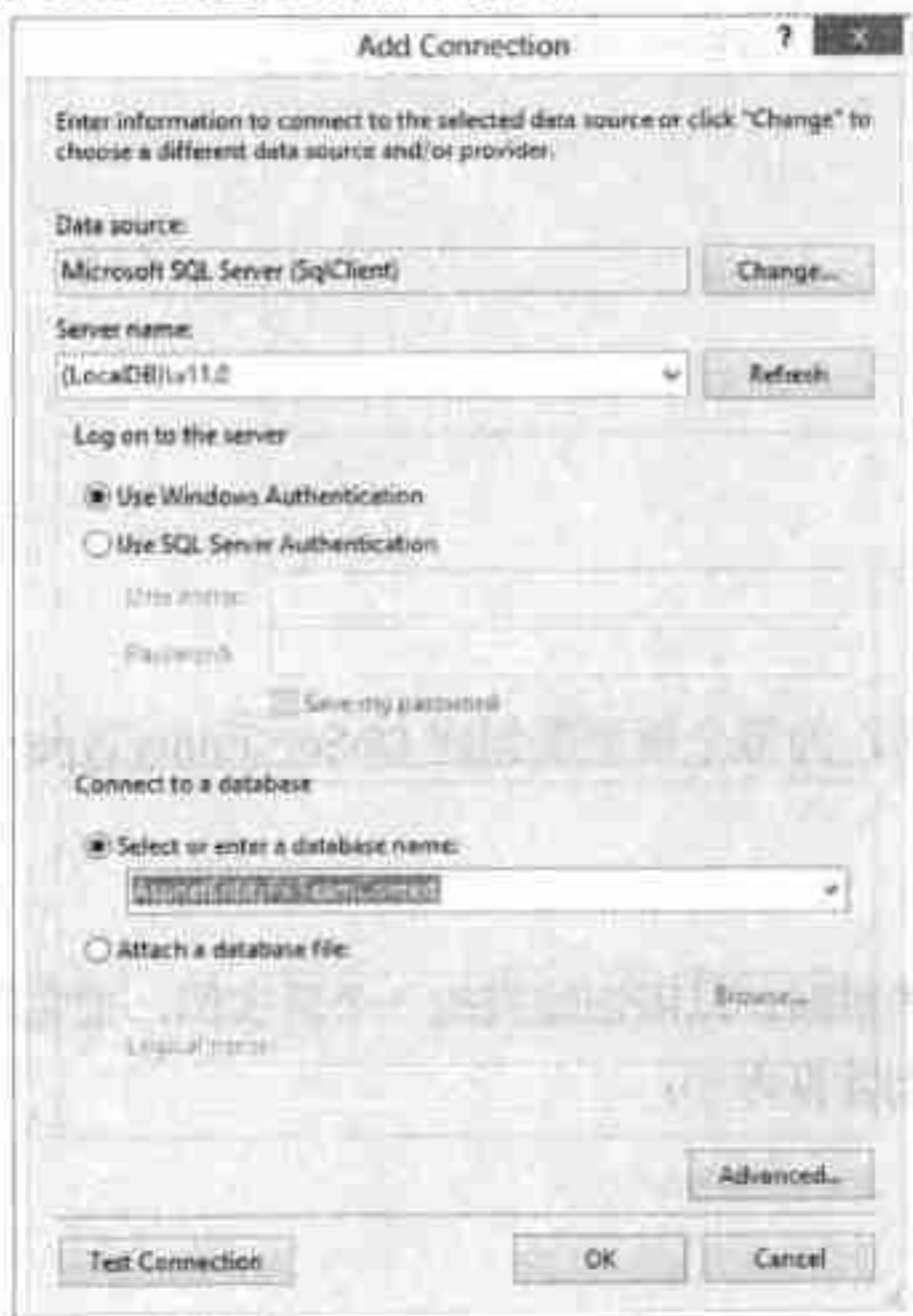


图 11-27



图 11-28

DbContext 根据定义的 DbSet 属性, 确定需要在模型中包含什么类, 接着 Code First 使用一组默认约定来确定表名、列名、主键、数据类型和数据库模式的其他部分。注意这些默认约定可以使用配置项修改。

11.6.3 Code First 中的关系

与使用设计器一样, 也可以在 Code First 模型的实体之间创建关系。为了演示这一点, 应在 Team 和 Player 实体之间创建一对多关系。

Entity Framework 使用导航属性表示两个实体类型之间的关系。导航属性可以管理两个方向的关系。如果关系是一对一, 就创建属性来引用相关的类型。如果关系是一对多或对多对一, 属性就是关系中涉及的类型的集合。属性还可以使用默认的 Code First 约定表示依赖对象上的外键。程序清单 11-10(本章下载代码中的 CodeFirst.aspx.cs)是修改类, 在 Team 和 Player 类之间创建关系后的代码文件。

程序清单 11-10 显示一对多 Code First 关系的代码

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;

namespace AspNetEntityFX
{
    public partial class CodeFirst : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            using (var context = new TeamContext())
            {
                var team = new Team { TeamName = "Team 1" };
                context.Teams.Add(team);
                team = new Team { TeamName = "Team 2" };
                context.Teams.Add(team);
                team = new Team { TeamName = "Team 3" };
                context.Teams.Add(team);
                context.SaveChanges();

                var query = from t in context.Teams
                            select t;

                GridView1.DataSource = query.ToList();
                GridView1.DataBind();
            }
        }
    }

    public class Team
    {
        public int TeamId { get; set; }
        public string TeamName { get; set; }
    }
}
```



```

        public virtual List<Player> Players { get; set; }
    }

    public class Player
    {
        public int PlayerId { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }

        public int TeamId { get; set; }
        public virtual Team Team { get; set; }
    }

    public class TeamContext : DbContext
    {
        public DbSet<Team> Teams { get; set; }
        public DbSet<Player> Players { get; set; }
    }
}

```

上面的代码在 **Team** 和一组相关的 **Player** 实体之间创建了如下关系：

```
public virtual List<Player> Players { get; set; }
```

注意，在数据库中查询特定的 **Team** 时，可能不需要填充 **Player** 对象集合。把 **Players** 属性设置为虚拟，就告诉 **DbContext**，懒惰加载 **Players** 集合。懒惰加载意味着，在试图访问集合时，自动从数据库中填充集合。

也可以创建从 **Player** 到 **Team** 的关系。添加一个导航属性，构建到 **Team** 对象的关系。这也是一个虚拟属性，以使用懒惰加载。添加另一个属性，告诉 **Code First** 在生成数据库模式时，创建到 **Team** 数据库表的外键：

```

    public int TeamId { get; set; }
    public virtual Team Team { get; set; }

```

如果现在尝试执行新代码，就会得到异常，因为数据库模式不再匹配模型。在新代码可以执行之前，必须更新数据库。下一节介绍如何使用 **Code First Migrations**，通过对模型的修改，更新数据库模式。

11.6.4 Code First Migrations

Code First Migrations 在 **Entity Framework 4.3.1** 中引入。在引入之前，修改 **Code First** 模型意味着删除数据库，再重建。显然，这不适合于产品阶段的应用程序。迁移是将模型的修改保存到数据库中的解决方案。与 **Code First** 模型一样，也可以使用代码建立迁移。但可以使用几个 **Package Manager Console** 命令建立迁移。

使用 **Code First Migrations** 的第一步是启用它：

(1) 在 **Visual Studio** 菜单中单击 **Tools | Library Package Manager | Package Manager Console**，打开 **Package Manager Console**。

(2) 在控制台的提示符中，运行命令：**Enable-Migrations**。

(3) 如果按照本章其他示例的指令执行, 项目中就会有几个派生的上下文。因此 `Enable-Migrations` 命令会回应一条消息, 指示无法确定给哪个上下文启用迁移。必须告诉该命令给哪个上下文启用迁移。为此, 运行命令:

```
Enable-Migrations -ContextTypeName AspnetEntityFX.TeamContext
```

命令执行完之后, 就会在项目中添加新文件夹 `Migrations`, 在该文件夹中放置以下两个新文件:

- 第一个文件 `Configuration.cs` 包含迁移用于派生上下文的配置设置。
- 第二个文件是初始的迁移文件, 其中包含已应用于数据库的修改。迁移文件名包含时间戳, 用于给迁移排序。

现在需要创建新的迁移, 其中包含上一节对模型进行的修改。为了添加新的迁移, 运行 `Add-Migration` 命令, 并给它传递迁移的名称。在 `Package Manager Console` 中, 运行命令 `Add-Migration TeamPlayerRelationship`。

运行命令, 就在 `Migrations` 文件夹中创建了新的迁移文件。程序清单 11-11(`X_TeamPlayerRelationship.cs`, 其中 `X` 是时间戳)显示了新迁移文件的内容。

程序清单 11-11 新迁移文件的内容

```
namespace AspnetEntityFX.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class TeamPlayerRelationship : DbMigration
    {
        public override void Up()
        {
            AddColumn("dbo.Players", "TeamId", c => c.Int(nullable: false));
            AddForeignKey("dbo.Players", "TeamId", "dbo.Teams", "TeamId",
                cascadeDelete: true);
            CreateIndex("dbo.Players", "TeamId");
        }

        public override void Down()
        {
            DropIndex("dbo.Players", new[] { "TeamId" });
            DropForeignKey("dbo.Players", "TeamId", "dbo.Teams");
            DropColumn("dbo.Players", "TeamId");
        }
    }
}
```

`Add-Migration` 命令仅创建了一个迁移文件, 其中包含对数据库模式的更新。不会自动应用这些更新。为了对数据库执行模式修改, 需要 `Update-Database` 命令。在 `Package Manager Console` 中运行 `Update-Database` 命令。命令执行完之后, 就可以在 `Server Explorer` 中查看对数据库的修改。图 11-29 显示了执行迁移后的数据库模式。

接着, 可以继续更新模型, 创建新迁移, 更新数据库。



图 11-29

11.7 本章小结

本章介绍了 Entity Framework(EF)的一些核心功能,开发人员可以选择如何使用 EF。EF 设计器允许使用更图形化的界面。可以选择建立模型并生成数据库,或者从已有的数据库生成模型。Entity Framework 添加了改进的功能,并会继续添加新功能,以简化数据的处理。Code First 和 Migrations 等功能提供了处理数据,以使用基于代码的工作流和基于设计器的工作流的选项。

ASP.NET 还包含一个使用 Entity Framework 的控件——EntityDataSource。

Entity Framework 允许使用许多不同的数据源,并且允许在面向对象的面向对象业务逻辑层和模式驱动的数据库层之间明确建立映射层,或者使用约定-配置方法把类映射到数据库层。

第 12 章

ASP.NET Dynamic Data

本章要点

- 理解 Dynamic Data 功能
- 理解模型绑定

ASP.NET 提供了一个功能，允许动态地创建数据驱动的 Web 应用程序。ASP.NET Dynamic Data 不仅是代码生成器，它还可以提供一个基本应用程序，而这个应用程序是可以全面修改的。因此，开发人员可以快速、方便地创建数据入口或应用程序，允许终端用户使用和查看后台数据库。

您还可以轻松地把 Dynamic Data 集成到应用程序中，建立一个丰富的 UI 层，它由模型绑定和数据注释特性驱动。

本章将介绍如何建立 ASP.NET Dynamic Data 应用程序，并联合使用 Dynamic Data 功能和模型绑定。

12.1 Dynamic Data 功能

ASP.NET Dynamic Data 的功能在 .NET Framework 3.5 SP1 中首次引入，并在 .NET Framework 4.5 中改进为使用 Entity Framework。首先，可以创建一个新的 Dynamic Data Entities Web Site 项目。

Visual Studio 2012 会创建一个基本应用程序，刚开始时它没有连接任何数据源或对象模型。这些连接工作需要由开发人员来完成。但是在此之前，我们首先查看 Visual Studio 已经创建什么内容。

12.1.1 默认应用程序中创建的核心文件

把预先生成的 Dynamic Data 应用程序赋予数据库之前，会通过前面提到的 Visual Studio 过程创建核心应用程序的许多文件。

自动生成的项和显示在 Visual Studio 的 Solution Explorer 中的内容一般称为基本框架 (scaffolding)。尽管生成了许多代码，但是不要担心自己进入了某个特定的数据驱动应用程序。生成的内容之所以称为“基本框架”，是因为它是一个架构，可以全部提取出来，也可以为了任何目的而

全面修改和扩展。它是提供自动生成应用程序所需的表示层和数据库层支持的架构。我们无法把它锁定为面向一组特定的模型、外观和操作方式，甚至无法锁定为某种方法，我们不能为了满足特定的需求而对其进行修改。

即使在 Solution Explorer 中发现有许多预先生成的代码，也不一定要通过这些代码来使用 ASP.NET Dynamic Data。实际上，甚至可以把 ASP.NET Dynamic Data 添加到已有的应用程序中，不一定要从 ASP.NET Dynamic Data 项目模板开始。

接着，本章介绍预先生成的应用程序，它允许使用后台数据库。

12.1.2 应用程序特性

与标准的 ASP.NET 应用程序相比，这种应用程序最大的一个亮点是添加了文件夹 DynamicData。这个文件夹包含预先生成的 ASP.NET 应用程序，它允许通过浏览器使用数据库。

这种应用程序的目标是允许在整个 CRUD(Create、Read、Update 和 Delete)过程中使用数据库，也可以限制在应用程序中提供的互操作量。

为了说明该应用程序如何使用数据库工作，需要进一步研究组成应用程序的控件和页面。展开 DynamicData 文件夹，其中包含如下子文件夹：

- Content
- CustomPages
- EntityTemplates
- FieldTemplates
- Filters
- PageTemplates

除了这些文件夹之外，还有专门用于该应用程序的 web.config 文件。

应用程序的 Content 文件夹包含两部分内容：用于页面模板的控件以及应用程序的样式表所使用的底层图像。

CustomPages 文件夹是独立的，允许放置要包含在数据驱动的 Web 应用程序中的任意定制页面。从头开始创建应用程序时，这个文件夹不包含任何文件，它是空文件夹。

EntityTemplates 文件夹可以非常轻松地获得想要的布局，因此不要求建立定制的页面。初始情况下，该文件夹中包含一个 Default.ascx 控件(用户控件)以及该控件的编辑和插入版本。

FieldTemplates 文件夹比较有趣，因为它反映了应用程序的一些比较详细的细节。整个应用程序设计为针对数据库工作，但它对所使用的数据库类型并不了解。FieldTemplates 文件夹是应用程序显示来自数据库的底层数据类型的一种方式。

Filters 文件夹用于给布尔值(true/false)、外键和枚举创建下拉菜单。这些菜单允许终端用户根据数据库中的键过滤表。

PageTemplates 文件夹包含用于建立应用程序的核心页面。注意，许多核心结构都使用这些页面来表示应用程序中的表。PageTemplates 文件夹包含如下页面：

- Details.aspx
- Edit.aspx
- Insert.aspx
- List.aspx

- ListDetails.aspx

List.aspx 页面用于已连接数据库中的表，Details.aspx 页面用于查看表中的一行，ListDetails.aspx 页面用于查看表的主/明细视图和行关系，Edit.aspx 和 Insert.aspx 页面分别用于执行编辑和插入操作。

Global.asax 文件定义了 Dynamic Data 应用程序使用的所有路由。这表示可以在不修改页面内容的情况下修改 URL。因为路由是通用的，所以它们可用于应用程序中的所有表。

使用 Data Annotations 可以定制要显示在 UI 上的列名，再使用它们为用户输入的列名添加验证功能。本章后面介绍模型绑定时将具体学习。

12.1.3 运行应用程序

创建了 DynamicDataWebSite 后，就需要连接一个可以使用的数据库。对于本例，需要包含 Northwind 数据库。之后，下一步是建立定义好的实体数据模型层来处理底层的数据库。

可以在整个解决方案中注册 NorthwindEntities 对象。NorthwindEntities 是使用 LINQ to Entities 建立的数据模型。这个 NorthwindEntities 上下文在 Global.asax 文件中注册，Global.asax 在项目中创建。程序清单 12-1 说明了如何在 Global.asax 中注册上下文。

程序清单 12-1 在 Global.asax 中注册上下文

```
DefaultModel.RegisterContext(typeof(NORTHWNDModel.NORTHWNDEntities),
    new ContextConfiguration() { ScaffoldAllTables = true });
```

这里把模型注册为 NorthwindDataContext 类型的 DataContext 对象，并把 ScaffoldAllTables 属性设置为 true(默认为 false)，表示要把模型中的所有表表示包含到生成的应用程序中。

12.1.4 运行应用程序的结果

运行应用程序时，注意第一个页面允许查看构成数据模型的所有表，如图 12-1 所示。



图 12-1

应用程序读取数据库的内容时，其工作相当简单。单击某个表的名称(这是应用程序中的超链接)，就会提供该表的内容，这个视图如图 12-2 所示。



图 12-2

表视图的样式很不错，它允许编辑、删除或查看数据库中每一行的细节。如果有一对多关系，还可以进一步查看子行的信息。该页面的另一个有趣之处是表的导航。在表的底部有一些分页控件，如图 12-2 所示。

除了使用表中数据行信息的编辑、删除和明细视图之外，还可以单击表底部的 Insert New Item 链接，在数据库中插入新行，生成如图 12-3 所示的视图。



图 12-3

编辑数据行会打开一个相同类型的页面，如图 12-4 所示。单击 Territories 表中某行旁边的 Edit 链接，就会进入这个页面。



图 12-4

这个应用程序的另一个有趣之处是，它可以使用数据模型中元素的一对多关系。例如，单击 Orders 表链接，会生成如图 12-5 所示的视图。



12-5

在这个视图中，除了表及其内容之外，还通过页面顶部的下拉列表框提供了过滤功能。在这个例子中，可以按顾客、雇员或承运商过滤订单。甚至可以组合使用这些元素来过滤数据项。在图 12-5

中查看表的内容时,要注意的另一个方面是:该图显示的不是 CustomerID、EmployeeID 或 ShipperID,而是这些项的名称。在进一步深入查看包含这些项的视图时,应用程序会引用这些项的标识符。

最后要理解的是,由于这是一个 ASP.NET 4.5 应用程序,因此它正确使用了 AJAX。例如,在过滤表中的项时,注意页面进行的是部分刷新,而不是全部刷新。

12.1.5 向已有页面中添加动态数据

当在 .NET Framework 3.5 SP1 中首次引入 ASP.NET Dynamic Data 时,为了使页面获得动态性,需要进行相应的设置。随着 ASP.NET 4.5 的发布,在 Web 页面中添加动态部分变得更加方便。

现在可以使用新的 DynamicDataManager 服务器控件或 Page_Init 调用中的 EnablingDynamicData。程序清单 12-2 是一个例子。

程序清单 12-2 在已有的 GridView 控件中添加动态数据

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Page_Init()
    {
        GridView1.EnableDynamicData(typeof(Customer));
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>DynamicDataManager Example</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:GridView ID="GridView1" runat="server" AllowPaging="True"
            DataSourceID="EntityDataSource1"
            AutoGenerateColumns="False" DataKeyNames="RegionID">
            <Columns>
                <asp:BoundField DataField="RegionID" HeaderText="RegionID"
                    ReadOnly="True" SortExpression="RegionID">
                </asp:BoundField><asp:BoundField DataField="RegionDescription"
                    HeaderText="RegionDescription"
                    SortExpression="RegionDescription">
                </asp:BoundField>
            </Columns>
        </asp:GridView>
        <asp:EntityDataSource ID="EntityDataSource1" runat="server"
            EntitySetName="Regions"
            ConnectionString="name=NORTHWNDEntities"
            DefaultContainerName="NORTHWNDEntities"
            EnableDelete="True" EnableFlattening="False" EnableInsert="True"
            EnableUpdate="True">
        </asp:EntityDataSource>
    </div>
    </form>
</body>
</html>
```


使用与前面例子中相同的 Northwind 数据上下文，将一个基本的 EntityDataSource 控件添加到页面中，以使用 GridView1 服务器控件。在这个例子中，也将 EntityDataSource 控件设置为处理 Regions 表。我们还需要一个显示顾客列表的网格。

12.2 理解模型绑定

在 .NET Framework 4.0 中，可以把 Dynamic Data 功能添加到已有的网站中，但仍很难充分利用 Dynamic Data 中的丰富支持。在 .NET Framework 4.5 中引入模型绑定后，就很容易把这些功能集成到应用程序中。这种集成通过模型绑定给开发人员提供类似于 ASP.NET MVC 的体验。本节介绍把 Dynamic Data 与模型绑定集成起来的一些方式。即使不从 Dynamic Data Web Application 项目模板开始，这种集成也是可行的。要使用模型绑定和 Dynamic Data，只需使用模型绑定和数据绑定控件。如果希望定制列和表的外观，只需在应用程序中添加 FieldTemplates 和 EntityTemplates 文件夹。



第 9 章详细介绍了模型绑定。

12.2.1 特性驱动的 UI

本节解释如何利用 Field 和 Entity 模板，以及 Data Annotation 特性来定制某列或表的外观。表 12-1 列出了可用于定制 UI 的常用特性。System.ComponentModel.DataAnnotations.dll 中包含所有这些特性。

表 12-1

特 性 名	说 明
DataType	指定应与这个列关联的类型
Display	指定 Dynamic Data 如何显示列，例如列名
DisplayFormat	指定 ASP.NET Dynamic Data 如何格式化数据字段
Enum	允许 .NET Framework 枚举映射到数据列
ScaffoldColumn	指定系统是否构建列
UIHint	指定 Dynamic Data 用于显示数据字段的字段模板

1. 字段模板

如前所述，Field Templates 文件夹包含映射到某种类型的列的用户控件。例如，Dynamic Data 系统使用 DateTime.ascx 显示 DateTime 类型的列。这些用户控件是可以重用的。这表示，如果两个表有 DateTime 列，就可以给这两个表使用相同的用户控件。

程序清单 12-3 说明了如何修改列名，它使用 Display 特性显示 LastName 列。运行页面，会显示 Last Part of Name。

程序清单 12-3 使用 Display 特性修改列的 UI

```
[Display(Name="Last Part of Name")]
public string LastName { get; set; }
```

现在假定要修改 LastName 列的编辑/插入视图的外观。为此，可以创建一个新的 Dynamic Data Field Template 用户控件，用于修改 UI。程序清单 12-4 说明了如何使用定制的字段模板 LastName。LastName 用户控件必须放在 Field Template 文件夹中。

要添加新的字段模板，可以右击网站，选择 Dynamic Data field 项模板。图 12-6 显示了该模板。

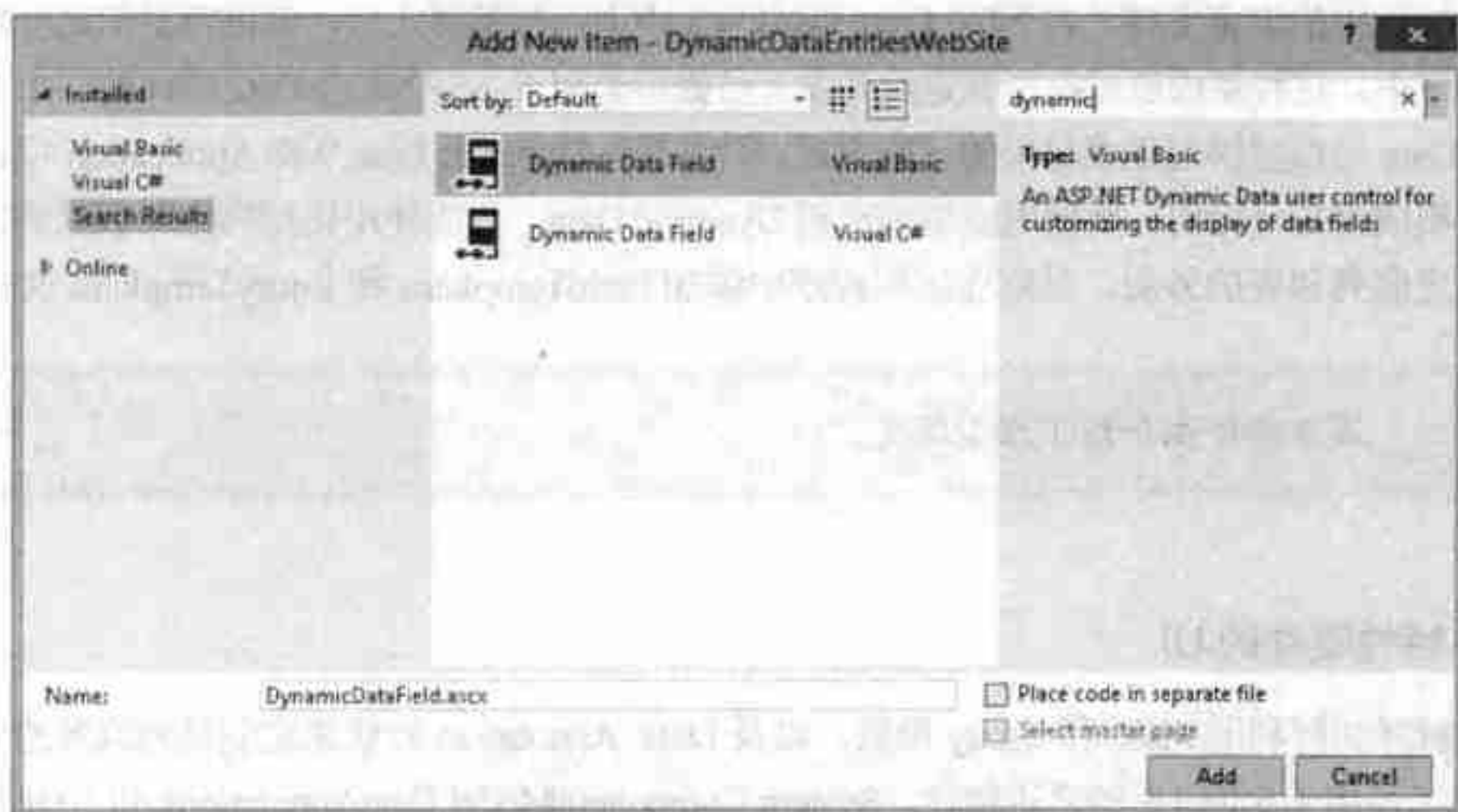


图 12-6

程序清单 12-4 使用 UIHint 特性修改使用的 FieldTemplate

```
[UIHint("LastName")]
public string LastName { get; set; }
```

2. 实体模板

前面学习了如何定制表中的列。现在假定要定制整个表的外观，同时使用 Dynamic Data 自动生成列。这可以使用实体模板实现。实体模板是 Dynamic Data 用于显示表中所有列的用户控件。

程序清单 12-5 显示了插入数据时实体模板的标记，它们由 FormView 控件使用，如程序清单 12-6 所示。

程序清单 12-5 用于插入表的典型实体模板

```
<asp:EntityTypeTemplate runat="server" ID="EntityTemplate1">
  <ItemTemplate>
    <tr class="td">
      <td class="DDLighthead">
        <asp:Label runat="server" OnInit="Label_Init"
          OnPreRender="Label_PreRender" />
      </td>
      <td>
```

```

        <asp:DynamicControl runat="server" ID="DynamicControl" Mode="Insert"
                        OnInit="DynamicControl_Init" />
    </td>
</tr>
</ItemTemplate>
</asp:EntityTypeTemplate>

```

程序清单 12-6 FormView 控件使用实体模板插入数据

```

<asp:FormView runat="server" ID="FormView1" DefaultMode="Insert"
    ItemType="Customer" InsertMethod="InsertCustomer">
    <InsertItemTemplate>
        <table>
            <asp:DynamicEntity ID="DynamicEntity1" runat="server"
                Mode="Insert" />

            <tr>
                <td>
                    <asp:LinkButton ID="Insert" runat="server"
                        CommandName="Insert" Text="Insert" />
                    <asp:LinkButton ID="Cancel" runat="server"
                        CommandName="Cancel" Text="Cancel"
                        CausesValidation="false" />
                </td>
            </tr>
        </table>
    </InsertItemTemplate>
</asp:FormView>

```

可以共享实体模板，为数据库中的所有表创建相同的外观。如果希望定制某个表的视图，就可以给这个表创建定制的实体模板。为此，可以在 `DynamicData\EntityTemplates` 下创建一个子文件夹，它应使用表名来命名，这里是 `Customer`。

12.2.2 特性驱动验证

特性驱动的验证是使用 `Data Annotations` 给应用程序添加验证逻辑的一种方式。前面介绍了一些可用于 UI 定制的特性，表 12-2 列出了一些验证特性。

表 12-2

特 性 名	说 明
Key	表示唯一标识某实体的一个或多个属性
Range	给数据字段的值指定数值范围
RegularExpression	指定在 ASP.NET Dynamic Data 中，数据字段的值必须匹配指定的正则表达式
Required	指定数据字段的值是必选的
StringLength	指定数据字段允许的最大和最小字符长度

1. 基本验证

开始使用数据绑定控件和模型绑定时，可以避免的一项操作是验证。模型绑定系统在绑定数据时，使用 `Dynamic Data` 处理表中列的验证。这表示，应用程序获得了列的客户端和服务端验证，

验证错误可以在 ValidationSummary 控件中查看。

2. 数据注释

还可以定制列上的验证类型。程序清单 12-4 说明了如何给 LastName 列添加一些正则表达式验证规则。如果输入一个不以 a 开头的姓，就会在验证汇总中获得如下验证错误：

The field LastName must match the regular expression 'a*'

注意，因为还设置了 DisplayName 特性来修改列名，所以在错误消息中会得到新的 DisplayName，如下所示：

The field Last Part of Name must match the regular expression 'a*'

这是 Dynamic Data 的一种非常引人注目的用法，因为它定制了应用程序的 UI 而没有修改模型。

程序清单 12-7 使用 RegularExpression 验证

```
[Display(Name = "Last Part of Name")]
[RegularExpression("a*")]
public string LastName { get; set; }
```

12.3 本章小结

本章探讨了 ASP.NET 中用于快速、方便地建立 Dynamic Data 驱动的应用程序的新功能。将这些新功能和 Visual Studio 一起使用，就可以在不到 5 分钟的时间内建立提供了完整 CRUD 功能的报表应用程序。同时，我们还可以使用所有这些功能和模型绑定，给应用程序提供丰富一致的体验。

第13章

使用服务

本章要点

- 建立和使用 Web 服务
- 理解 WCF
- 使用 Web API

本章将介绍 XML Web 服务的建立, 以及如何使用 XML Web 服务接口, 并把它们集成到 ASP.NET 应用程序中。本章首先介绍 .NET 中 XML Web 服务的基础知识, 探讨一些底层技术, 如 SOAP、WSDL 等。本章中间部分将重点介绍 Windows Communication Foundation, 简称 WCF。最后, 本章将介绍最新的通信架构 Web API, 这个架构相当新, 可用于快速、轻松地通过 API 提供核心功能。

13.1 不同系统之间的通信

我们生活在一个多样化的世界。在大型企业中, 整个组织及其数据存储库很少放在一家供应商的平台上。在大多数情况下, 组织由多个系统组成——一些系统基于 UNIX, 一些系统基于 Microsoft, 一些系统基于其他平台。永远都不可能把所有的信息都放在一个平台上, 同时保证所有数据从一台服务器无缝地移动到另一台服务器上。因此, 各种系统必须能够相互通信。如果不同的系统很容易相互通信, 那么在企业中移动特定的数据集就是一个简单的过程, 不需要复制系统和数据存储。

在引入 XML 语言时, 标记语言显然已成为在企业中进行必要集成的基础。XML 非常强大, 系统无论使用什么平台、语言或数据存储, 都可以使用它操作 DataSet。

XML 源于 1986 年创建的标准通用标记语言(Standard Generalized Markup Language, SGML)。由于 SGML 非常复杂, 需要简单一些的语言, 因此诞生了 XML。

XML 非常适合于数据表示, 因为它允许开发人员以他们认为合适的方式构建 XML 文档。因此, XML 也是混乱无序的。在不同的系统之间发送自己构建的 XML 文档并没有太大意义, 因为必须为每一对通信的系统定制显示和使用的模型。

供应商和业界不久就认识到, XML 需要一种特殊的结构, 要制定一些规则来规范通信。定义 XML 结构的规则大大简化了不同系统之间的通信。工具供应商现在可以自动处理通信过程, 以及使用通信协议自动创建应用程序的所有组件。

业界一直使用简单对象访问协议(Simple Object Access Protocol, SOAP)支持标准的 XML 结构。以前人们尝试使用组件技术, 如分布式组件对象模型(Distributed Component Object Model, DCOM)、远程方法调用(Remote Method Invocation, RMI)、公用对象请求代理程序体系结构(Common Object Request Broker Architecture, CORBA)和 Internet Inter-ORB 协议(IIOP)来解决通信问题, 但都以失败告终, 因为这些技术都由一家供应商驱动, 或是供应商特有的。因此, 不可能在整个业界实现它们。

SOAP 可以显示和使用复杂的数据结构, 包括 DataSet 或建立了所有关系的数据表。SOAP 相当简单, 很容易理解。与 ASP.NET 一样, XML Web 服务也主要在 HTTP 上工作。我们发送和使用的 DataSet 可以在同一条 Internet 线路(HTTP)上传输, 从而绕过了许多防火墙(因为它们通过端口 80 传输)。

那么, 在线路上传输的到底是什么? ASP.NET Web 服务一般通过 HTTP Post 协议, 在 HTTP 上使用 SOAP。SOAP 请求示例(从客户端发送给 Web 服务器的 Web 服务)使用了程序清单 13-1 中所示的结构。

程序清单 13-1 SOAP 请求

```
POST /WebService.asmx HTTP/1.1
Host: localhost
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 19
<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <HelloWorld xmlns="http://tempuri.org/" />
  </soap12:Body>
</soap12:Envelope>
```

该请求被发送给 Web 服务以调用 WebMethod HelloWorld(WebMethod 详见本章后面的内容)。Web 服务的 SOAP 响应如程序清单 13-2 所示。

程序清单 13-2 SOAP 响应

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 14
<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <HelloWorldResponse xmlns="http://tempuri.org/">
      <HelloWorldResult>Hello World </HelloWorldResult>
    </HelloWorldResponse>
```



```
</soap12:Body>
</soap12:Envelope>
```

从程序清单 13-1 和 13-2 的例子中可以看出, 这条消息包含的是一个 XML 文件。除了<xml>节点的 XML 声明之外, 还包含了作为 SOAP 消息的 XML 结构。SOAP 消息使用了根节点<soap:Envelope>, 该节点又包含<soap:Body>, 即 SOAP 消息主体。包含在 SOAP 消息中的其他元素有 SOAP 报头<soap:Header>和 SOAP 错误<soap:Fault>。



有关 SOAP 消息结构的更多信息, 可查阅位于 W3C 网站 <http://www.w3.org/tr/soap> 上的 SOAP 规范。

13.2 建立一个简单的 XML Web 服务

建立 XML Web 服务就是把一些信息或逻辑显示给企业中的另一个实体、商业伙伴或顾客。进一步说, 建立 Web 服务就是开发人员从支持 SOAP 通信的类中建立一个或多个方法。

使用 Visual Studio 2012 可以建立 XML Web 服务。首先是创建一个新的 Web 站点: 从 IDE 菜单中选择 File | New | Web Site 命令。打开 New Web Site 对话框, 从中选择 ASP.NET Empty Web Site 选项, 如图 13-1 所示。

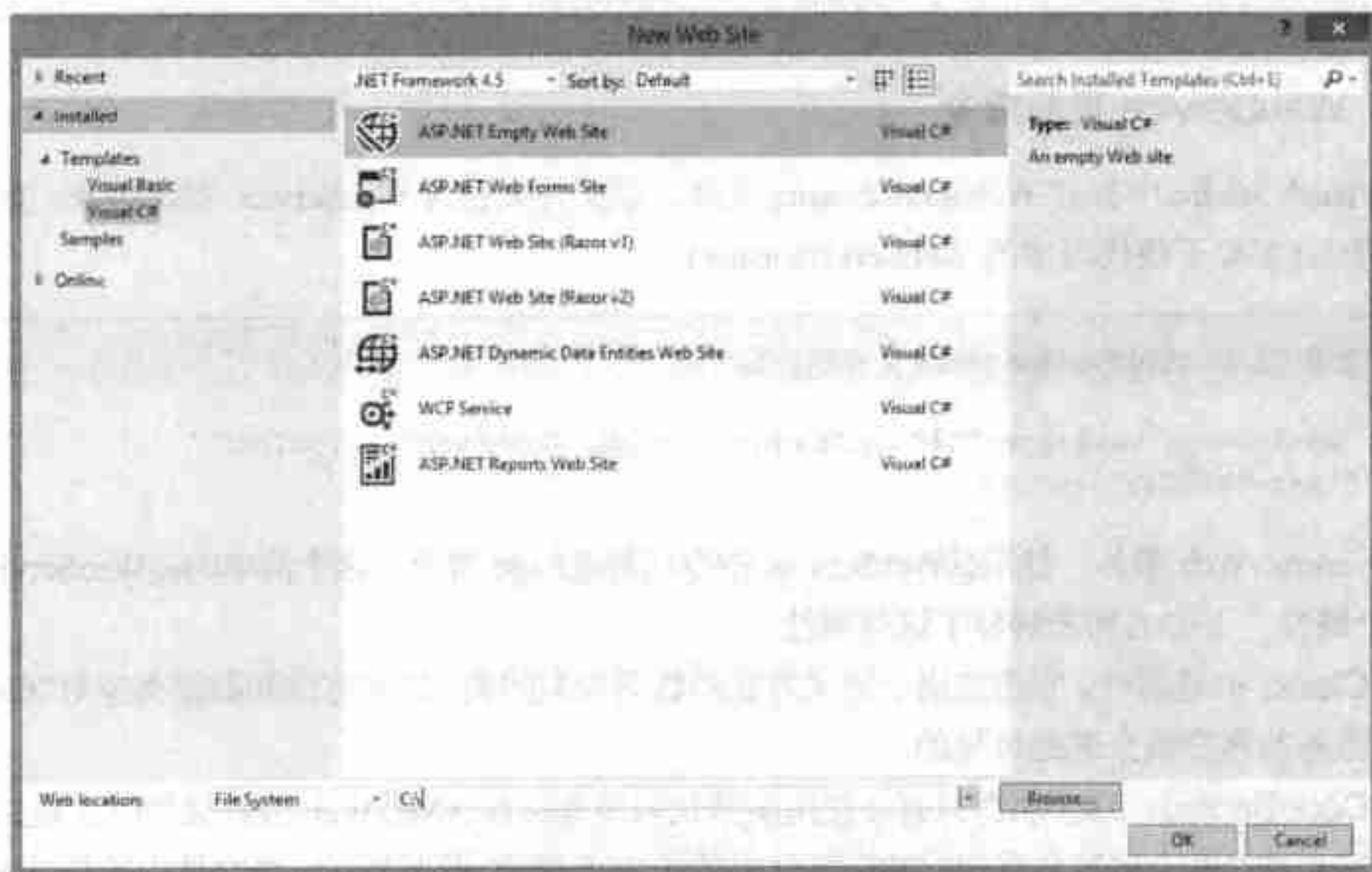


图 13-1

建立新项目后, 右击该项目, 为其添加一个新文件。为此, 从选项列表中选择 Web Service (WebService.asmx)。这样会创建一个名为 WebService.asmx 的 XML Web 服务, 其隐藏代码文件 WebService.cs 位于 App_Code 文件夹中, 如图 13-2 所示。

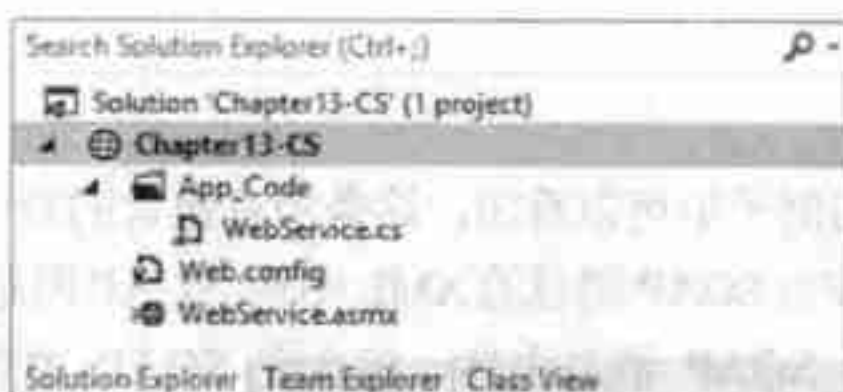


图 13-2



本章后面主要介绍 ASMX Web 服务。2009 年, ASMX Web 服务被标记为过时。因此, 其中的代码没有用未来的 ASP.NET 版本更新(但安全性更新了), 但是, 包含这些内容是为了帮助支持已有的 ASMX Web 服务。

检查 WebService.asmx 文件。所有的 ASP.NET Web 服务文件都使用扩展名 .asmx, 而不是 ASP.NET 页面使用的扩展名 .aspx。



ASP.NET Web 服务是使用 ASP.NET 创建 Web 服务的最初方法。在社区, 它们统称为 ASMX 服务, 另外两种创建 Web 服务的方法是 WCF 和 Web API, 参见本章后面的内容。

13.2.1 WebService 页面指令

在 Visual Studio 中打开 WebService.asmx 文件, 该文件只包含 WebService 页面指令, 如程序清单 13-3 所示(本章下载代码中的 WebService.asmx)。

程序清单 13-3 WebService.asmx 文件的内容

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/WebService.cs"
    Class="WebService" %>
```

对于 .asmx Web 服务, 使用 @WebService 指令代替 @Page 指令。这个简单的 @WebService 指令只有 4 个属性。下面的列表解释了这些属性:

- **Class:** 必选属性, 它指定用于定义方法和数据类型的类, 这些方法和数据类型对 XML Web 服务的客户端来说是可见的。
- **CodeBehind:** 这个属性只有在使用隐藏代码模型操作 XML Web 服务文件时才是必选的。它允许在两个更容易管理的独立部分中操作 Web 服务, 而不是在一个文件中操作 Web 服务。CodeBehind 属性带一个字符串值, 它表示 Web 服务中第二部分(即包含所有 Web 服务逻辑的类文件)的物理位置。在 ASP.NET 中, 最好把隐藏代码文件放在 App_Code 文件夹中, 从最初打开 Web 服务项目时 Visual Studio 创建的默认 Web 服务开始, 就放在该文件夹中。
- **Debug:** 可选属性, 可以设置为 True 或 False。如果 Debug 属性设置为 True, XML Web 服务在编译时就使用调试符号; 把该属性设置为 False, 在编译 Web 服务时就不带调试符号。
- **Language:** 必选属性, 指定用于 Web 服务的语言。

13.2.2 Web 服务的基类文件

下面看看 WebService.cs 文件, 这是 XML Web 服务的隐藏代码文件。在默认情况下, WebService.cs 文件中已包含了代码结构, 如程序清单 13-4 所示(本章下载代码中的/App_Code/WebService.cs)。

程序清单 13-4 Visual Studio 为 Web 服务提供的默认代码结构

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class WebService : System.Web.Services.WebService {
    [WebMethod]
    public string HelloWorld() {
        return "Hello World";
    }
}
```

这里要注意几个地方。首先, 现在在 C# 解决方案中包含 System.Linq 名称空间。另外, 还包含一个注释的 System.Web.Script.Services.ScriptService 对象(为了简短起见, 程序清单 13-4 删除了它), 它用于处理 ASP.NET AJAX 脚本。要使用这个对象, 只需取消对它的注释。

System.Web.Services.Protocols 名称空间默认包含在程序清单 13-4 的 VB 版本中。因此, 在使用该名称空间提供的 SOAP 报头和其他功能时, 不必再包含该名称空间。

最后, 可以看到<WebServiceBinding>特性。它会建立遵循 WS-I Basic Profile 1.0 版本(详见 <http://www.ws-i.org/Profiles/BasicProfile-1.0.html>)的 XML Web 服务响应。

13.2.3 把定制的 DataSet 显示为 SOAP

为了建立自己的 Web 服务示例, 应删除 WebService.asmx 文件, 创建新文件 Customers.asmx。这个 Web 服务会显示 SQL Server 中的 Customers 表, 然后执行如程序清单 13-5 所示的代码(本章下载代码中的/App_Code/Customers.cs)。



本书使用的 AdventureWorks 数据库可以从 www.wrox.com/go/SQLSever2012DataSets 上下载。

程序清单 13-5 使用 XML Web 服务显示 AdventureWorks 数据库中的 Customers 表

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Web;
```



```

using System.Web.Services;

[WebService(Namespace = "http://adventureworks/customers")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Customers : System.Web.Services.WebService {
    [WebMethod]
    public DataSet GetCustomers() {
        SqlConnection conn;
        SqlDataAdapter myDataAdapter;
        DataSet myDataSet;
        string cmdString = "Select * From SalesLT.Customer";

        conn = new SqlConnection("Server=(LocalDB)\\v11.0;integrated security=True;
            attachdbfilename=|DataDirectory|\\AdventureWorksLT2012_Data.mdf;");
        myDataAdapter = new SqlDataAdapter(cmdString, conn);

        myDataSet = new DataSet();
        myDataAdapter.Fill(myDataSet, "Customers");

        return myDataSet;
    }
}

```

1. WebService 特性

所有的 Web 服务都封装在一个类中，这个类在类声明之前通过 **WebService** 特性定义为 Web 服务。下面是一个例子：

```
[WebService(Namespace = "http://adventureworks/customers")]
```

WebService 特性有几个属性。在 Web 服务中，默认使用 **WebService** 特性的 **Namespace** 属性，其初始值为 `http://tempuri.org/`。这是一个临时的名称空间，可以使用更有意义的初始名称代替，例如驻留 XML Web 服务的 URL。该属性的值并不一定要是实际的 URL，而可以是任意字符串值，最好是唯一的值。

注意，在 **WebService** 特性中，VB 和 C# 两种语言以不同的方式定义其属性。Visual Basic 2012 使用冒号和等号来设置属性：

```
Namespace:="http://adventureworks/customers"
```

C# 仅使用等号给 **WebService** 特性中的属性赋值：

```
Namespace = "http://adventureworks/customers"
```

WebService 特性的其他属性有 **Name** 和 **Description**。**Name** 属性允许改变 Web 服务名通过 ASP.NET 测试页面(本章后面将讨论测试页面)显示给开发人员的方式。**Description** 属性可以提供 Web 服务的文本描述，该描述也显示在 ASP.NET Web 服务测试页面上。如果 **WebService** 特性包含多个属性，就使用逗号将它们分隔开，如下所示：

```
<WebService(Namespace:="http://adventureworks/customers", Name:="GetCustomers")>
```

2. WebMethod 特性

在程序清单 13-5 中, Customers 类只有一个 WebMethod。WebService 类可以包含任意多个 WebMethod, 也可以同时包含标准方法以及通过在方法声明之前使用 WebMethod 特性转变为 WebMethod 的方法。只有应用了 WebMethod 特性的方法才能通过 HTTP 线路访问。

与 WebService 特性一样, WebMethod 特性也可以包含一些属性, 如下所示:

- **BufferResponse:** BufferResponse 属性设置为 True 时, XML Web 服务的响应就保存在内存中, 并作为完整的包发送。如果该属性设置为默认值 False, 就在服务器上构造响应的同时将其发送给客户端。
- **CacheDuration:** 指定响应在系统的高速缓存中保存的时间(秒)。其默认设置为 0, 表示禁用高速缓存。把 XML Web 服务的响应放在高速缓存中, 会提高 Web 服务的性能。
- **Description:** 对在 XML Web 服务的.aspx 测试页面上显示的 WebMethod 应用文本描述。
- **EnableSession:** 把 EnableSession 属性设置为 True, 会激活某个 WebMethod 的会话状态, 其默认设置是 False。
- **MessageName:** 给 WebMethod 指定唯一的名称。如果要使用重载的 WebMethod(详见本章后面的讨论), 这就是必须执行的一步。
- **TransactionOption:** 为 WebMethod 指定事务处理支持, 其默认设置是 Disabled。如果 WebMethod 是启动事务的根对象, Web 服务就可以使用另一个需要事务处理的 WebMethod 参与事务处理。其值还可以是 NotSupported、Supported、Required 和 RequiresNew。

13.2.4 XML Web 服务接口

程序清单 13-5 中的 Customers Web 服务只有一个 WebMethod, 它返回一个 DataSet, 其中包含 SQL Server AdventureWorks 数据库中完整的 Customers 表。

在浏览器中运行 Customers.asmx, 就会执行 ASP.NET Web 服务测试页面。Web 服务的这个可见接口可用于测试, 也可以由使用该 Web 服务的开发人员用作引用页面。为 Customers Web 服务的 GetCustomers WebMethod 生成的页面如图 13-3 所示。

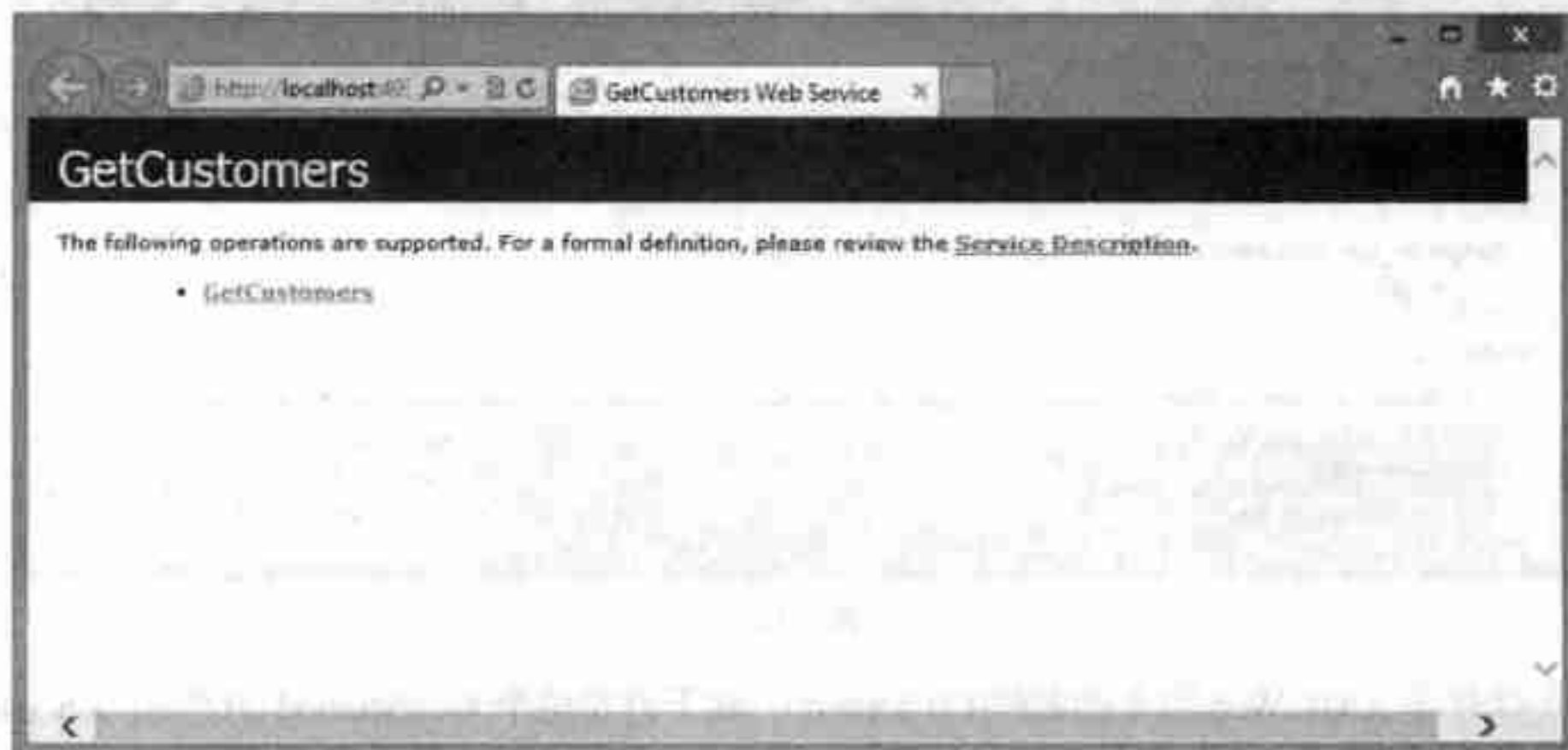


图 13-3

该接口在页面顶部的蓝色栏(在这幅黑白图中是黑色栏)中显示了 Web 服务的名称。在默认情况

下, Web 服务名使用类的名称, 除非通过 WebService 特性的 Description 属性改变了该名称, 如前面的定义所示。该页面还以项目列表的形式显示了指向该 Web 服务的所有 WebMethod 的链接。在这个例子中, 只有一个 WebMethod: GetCustomers。

该页面也显示了该 Web 服务的 WSDL(Web Services Description Language, Web 服务描述语言)文档的链接(该链接在图 13-3 中的标题是 Service Description)。WSDL 文件是 Customers Web 服务的实际接口。XML 文档(如图 13-4 所示)并不适合于我们使用, 而适合于 Visual Studio 等工具, 它会通知该工具需要什么数据才能使用 Web 服务。每个 Web 服务都要求, 请求必须带有某种类型的参数。在发出请求时, Web 服务的响应就会与一组使用特定数据类型定义的数据一起返回。发出请求需要执行的操作以及从响应中会返回什么内容(如果我们是使用者)都在 WSDL 文档中进行了描述。

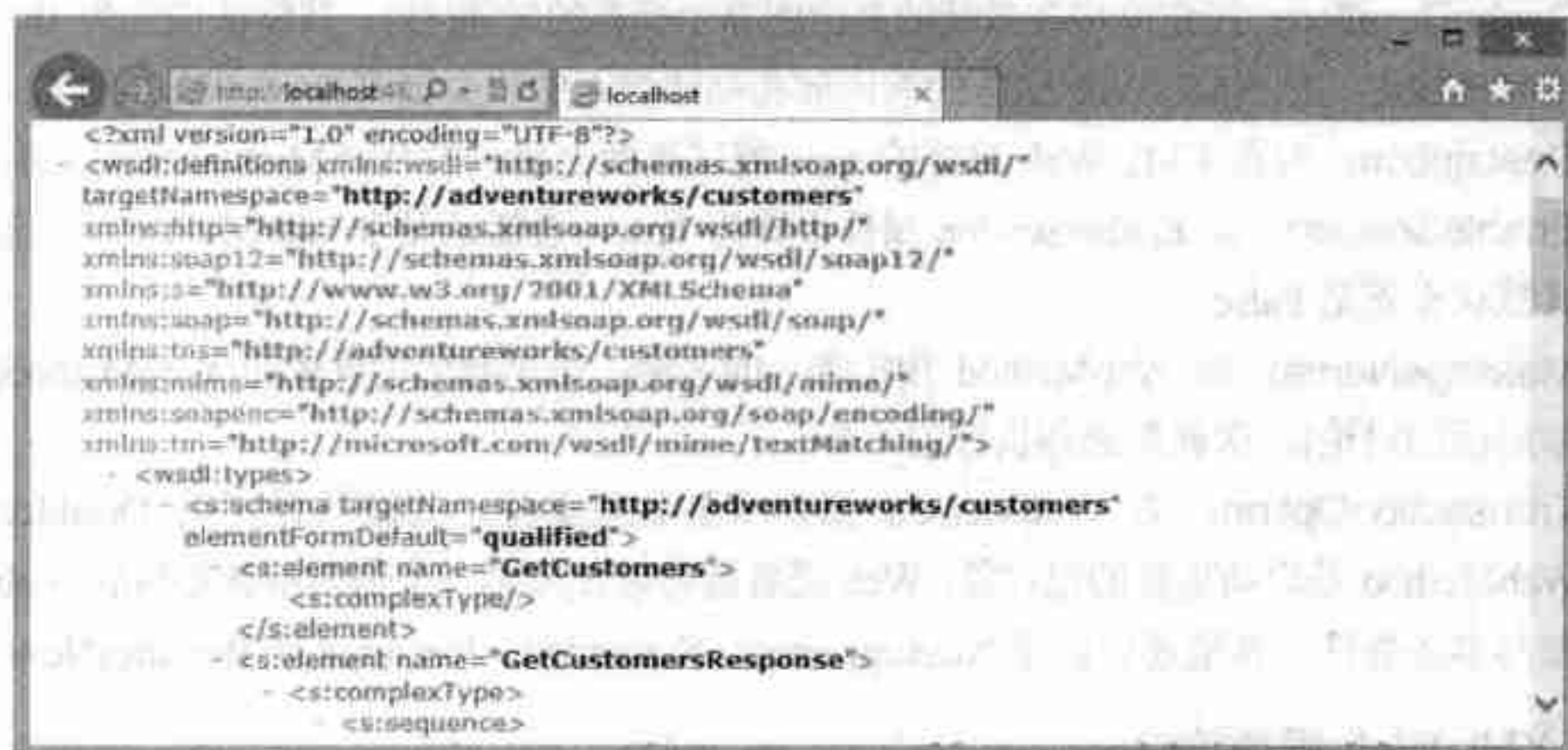


图 13-4

单击 GetCustomers 链接, 进入如图 13-5 所示的新页面, 该页面不仅详细描述了 WebMethod, 还可以直接在浏览器中测试 WebMethod。

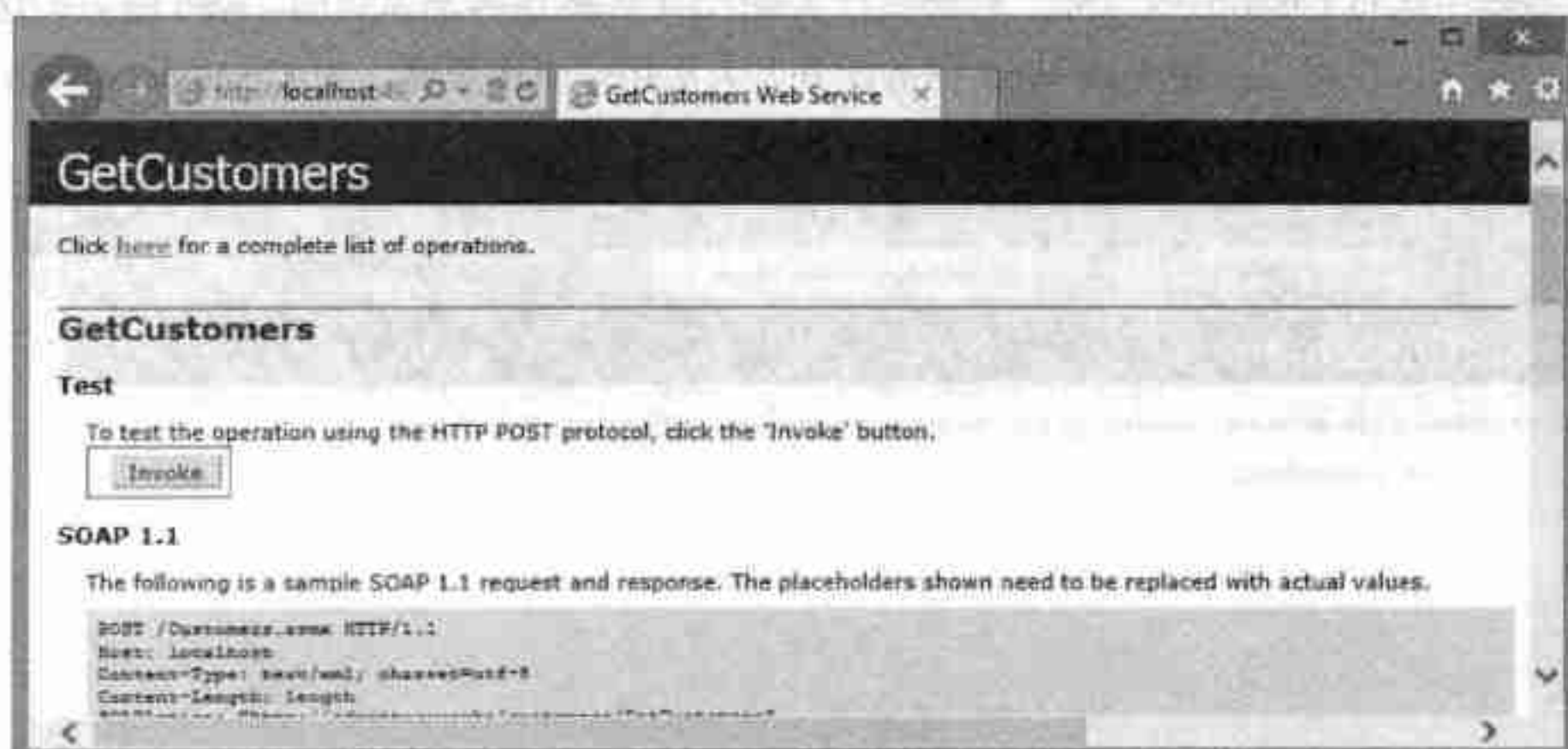


图 13-5

页面的顶部是 XML Web 服务的名称(Customers), 其下方是这个 WebMethod 的名称(GetCustomers)。页面显示了要使用该 WebMethod 的 SOAP 消息结构, 以及用于响应的 SOAP 消息结构。在 SOAP 例子的下方是通过 HTTP Post(带名/值对)使用 XML Web 服务的例子。可以使用 HTTP Post 来替代 SOAP。

可以在该页面上直接测试 WebMethod。在 Test 部分有一个窗体，如果所调用的 WebMethod 需要输入一些参数以获得响应，该窗体上就会包含一些文本框，这样就可以在单击 Invoke 按钮之前提供参数。如果所调用的 WebMethod 不需要参数，页面上就只有 Invoke 按钮。

单击 Invoke 按钮，就会把一个 SOAP 请求发送给 Web 服务，其结果是显示一个新的浏览器实例，如图 13-6 所示。



图 13-6

现在万事俱备，接下来就可以在 ASP.NET 应用程序中使用 XML Web 服务了。

13.3 使用简单的 XML Web 服务

前面只讨论了 XML Web 服务的一部分。使用 .NET，尤其是 ASP.NET，在企业不同的系统中或世界各地不同的系统中把数据和逻辑显示为 SOAP 是一项很简单的任务。下面就在 ASP.NET 应用程序中实际地使用 XML Web 服务。

XML Web 服务并不局限于在 ASP.NET 应用程序中使用，但因为本书介绍的是 ASP.NET，所以主要探讨这方面的使用。在其他类型的应用程序中使用 XML Web 服务并不困难，实际上与在 ASP.NET 中使用它们相当类似。注意，前面介绍的 Web 服务都可以在 Windows 窗体应用程序和移动应用程序中使用。甚至可以结合使用 XML Web 服务和其他 Web 服务，使 Web 服务由其他 Web 服务集合组成。

13.3.1 添加 Web 引用

要使用本章前面创建的 Customers Web 服务，应创建新的 ASP.NET Web 站点 CustomerConsumer。在 ASP.NET 应用程序中使用 XML Web 服务的第一步是引用远程对象，即引用 Web 服务。为此，应在 Visual Studio 的 Solution Explorer 中右击项目的根节点，选择 Add Web Reference 命令，打开

Web Reference 对话框，该对话框允许输入定义的 URL。但是，为了打开传统的 Web 服务引用对话框，可以单击 Advanced 按钮。接着在 Advanced 设置页面的底部选择 Compatibility 部分的 Add Web Reference，打开 Add Web Reference 对话框，如图 13-7 所示。

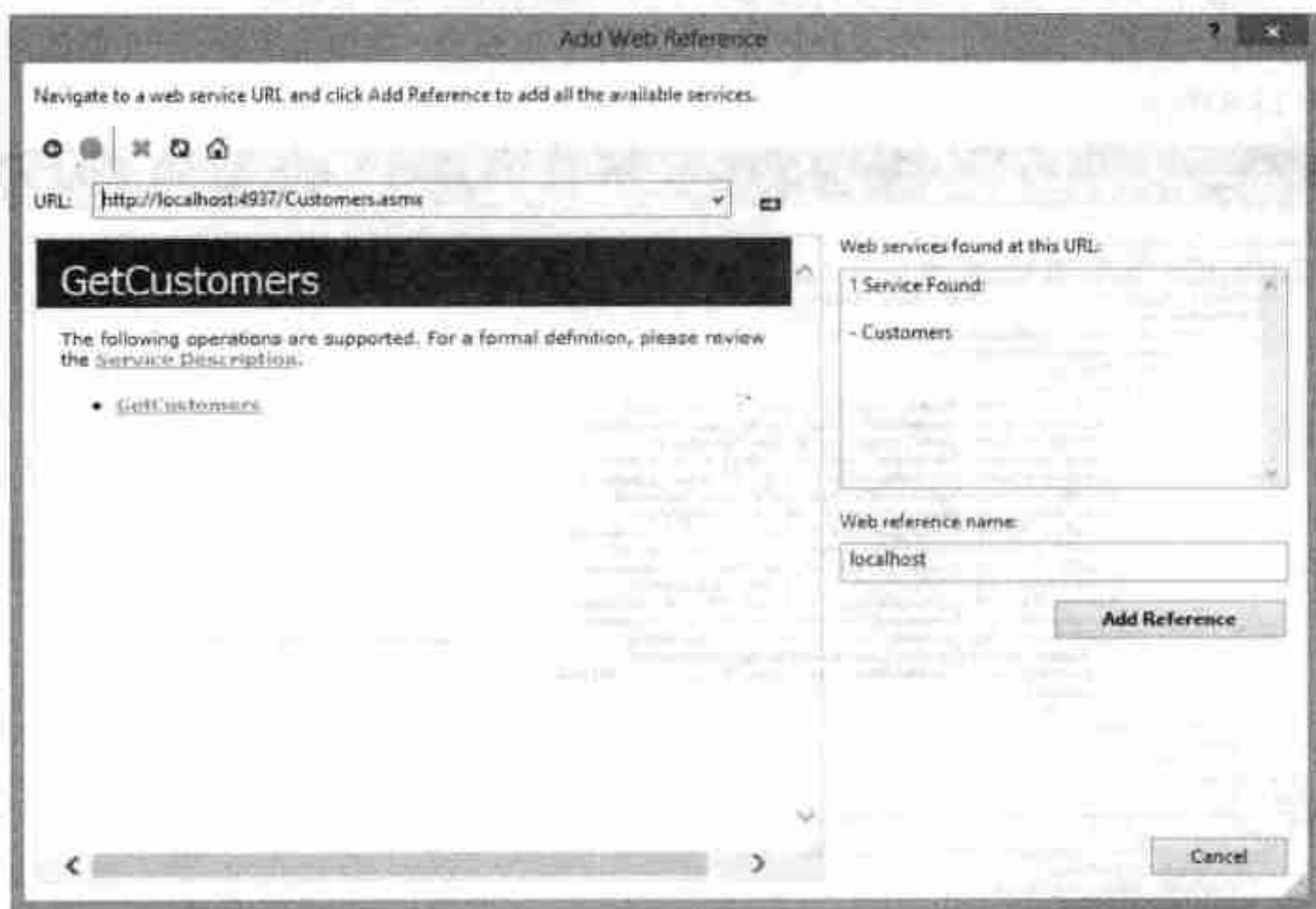


图 13-7



在 Visual Studio 2010 及其早期版本中，Add Web Reference 对话框有自己的菜单项 Add Web Reference。但在 WCF 和 Web API 出现后，就删除了这个菜单项。

Add Web Reference 对话框允许指定要引用的 .asmx 文件。该对话框查找的实际是 WSDL 文件。Microsoft 的 XML Web 服务会根据其 .asmx 文件自动生成 WSDL 文件。要在浏览器中打开 WSDL 文件，只需输入 Web 服务的 .asmx 文件的 URL，并在该字符串的最后添加 ?WSDL。例如，输入如下内容(这不是一个实际的 Web 服务，只是一个例子)：

```
http://www.wrox.com/MyWebService/Customers.asmx?WSDL
```

Add Web Reference 对话框会自动查找基于 Microsoft 的 XML Web 服务的 WSDL 文件，因此应给不是基于 Microsoft 的 XML Web 服务输入其 WSDL 文件的 URL。



如果使用 Visual Studio 与其内置的 Web 服务器，而不是 IIS，那么还需要把 Web 服务器使用的端口号插入 URL。在该例中，URL 应该类似于 `http://localhost:4937/Customers.asmx?WSDL`。

在 Add Web Reference 对话框中，可以把引用从默认名称改为更有意义的名称。如果在一台计算机上工作，Web 引用的名称就是 localhost；如果要使用远程 Web 服务，名称就是 URL 的逆序字符串，例如 com.wrox.www。在这两种情况下，最好重命名 Web 引用，使名称更有意义，在应用程序中更易于使用。在前面的例子中，将 Web 引用重命名为 AdventureWorksCustomers。

单击 Add Reference 按钮，让 Visual Studio 在应用程序的 web.config 文件中建立对 Web 服务的引用，如图 13-8 所示。在 App_WebReferences 文件夹下还包含一些文件，例如 Web 服务的 WSDL 文件的副本。

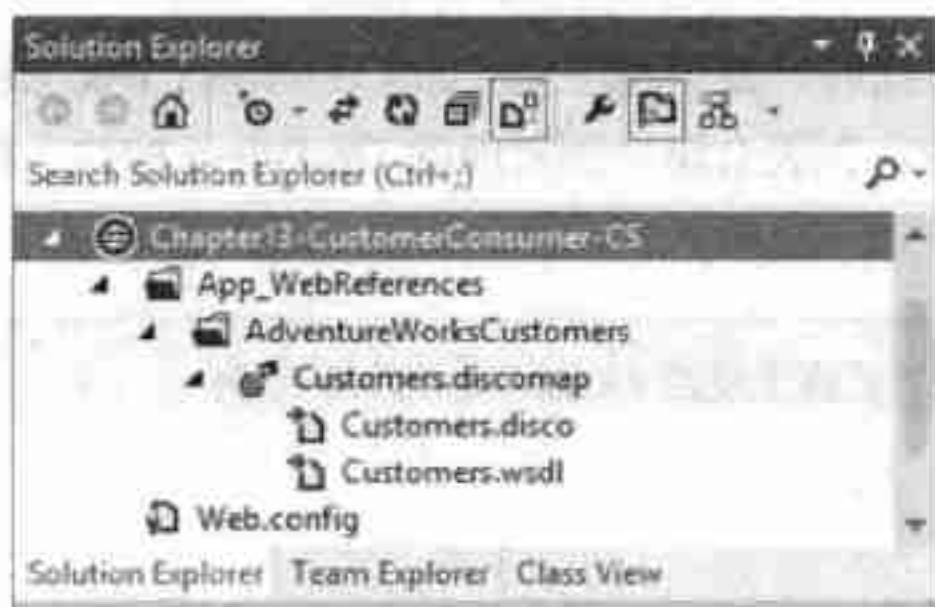


图 13-8

应用程序的 web.config 文件在 <appSettings> 部分中包含对 Web 服务的引用，如程序清单 13-6 所示。

程序清单 13-6 在建立对 Web 服务的引用后修改 web.config 文件

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="false" targetFramework="4.5"/>
    <httpRuntime targetFramework="4.5"/>
  </system.web>
  <appSettings>
    <add key="AdventureWorksCustomers.Customers"
        value="http://localhost:4473/Customers.asmx"/>
  </appSettings>
</configuration>
```

AdventureWorksCustomers 引用和 Web 服务名放在一起，提供了键值 AdventureWorksCustomers.Customers。value 特性的值是 Customers Web 服务的位置，该 Web 服务位于 Customers.asmx 页面中。

13.3.2 在客户端应用程序中调用 Web 服务

建立了对 XML Web 服务的引用后，就可以在 ASP.NET 应用程序中使用了。在项目中创建一个新的 Web 窗体。有了这个页面，就可以直接在应用程序中使用远程数据库 AdventureWorks 中的 Customers 表。数据放在 GridView 控件中。

在页面的设计部分，放置一个按钮和一个 GridView 控件，使页面如图 13-9 所示。

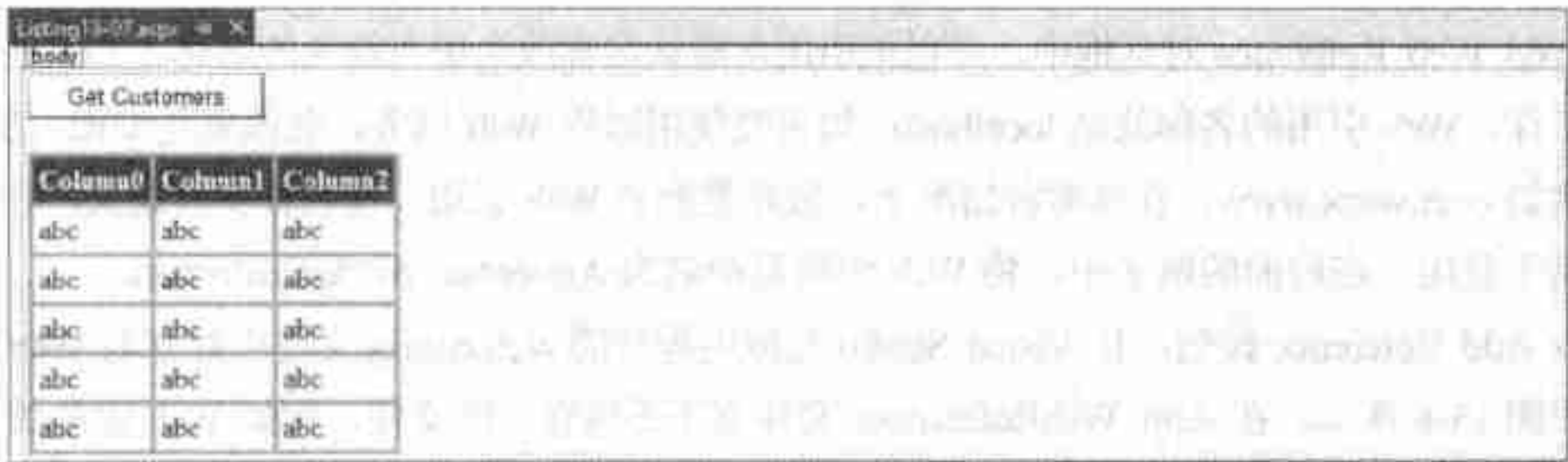


图 13-9

终端用户单击窗体中的按钮时，应用程序就会给 Customers Web 服务发送 SOAP 请求，获得包含 Customers 表的 SOAP 响应，该表会绑定到页面的 GridView 控件。程序清单 13-7 显示了这个简单应用程序的代码。

程序清单 13-7 在 ASP.NET 页面上使用 Customers Web 服务

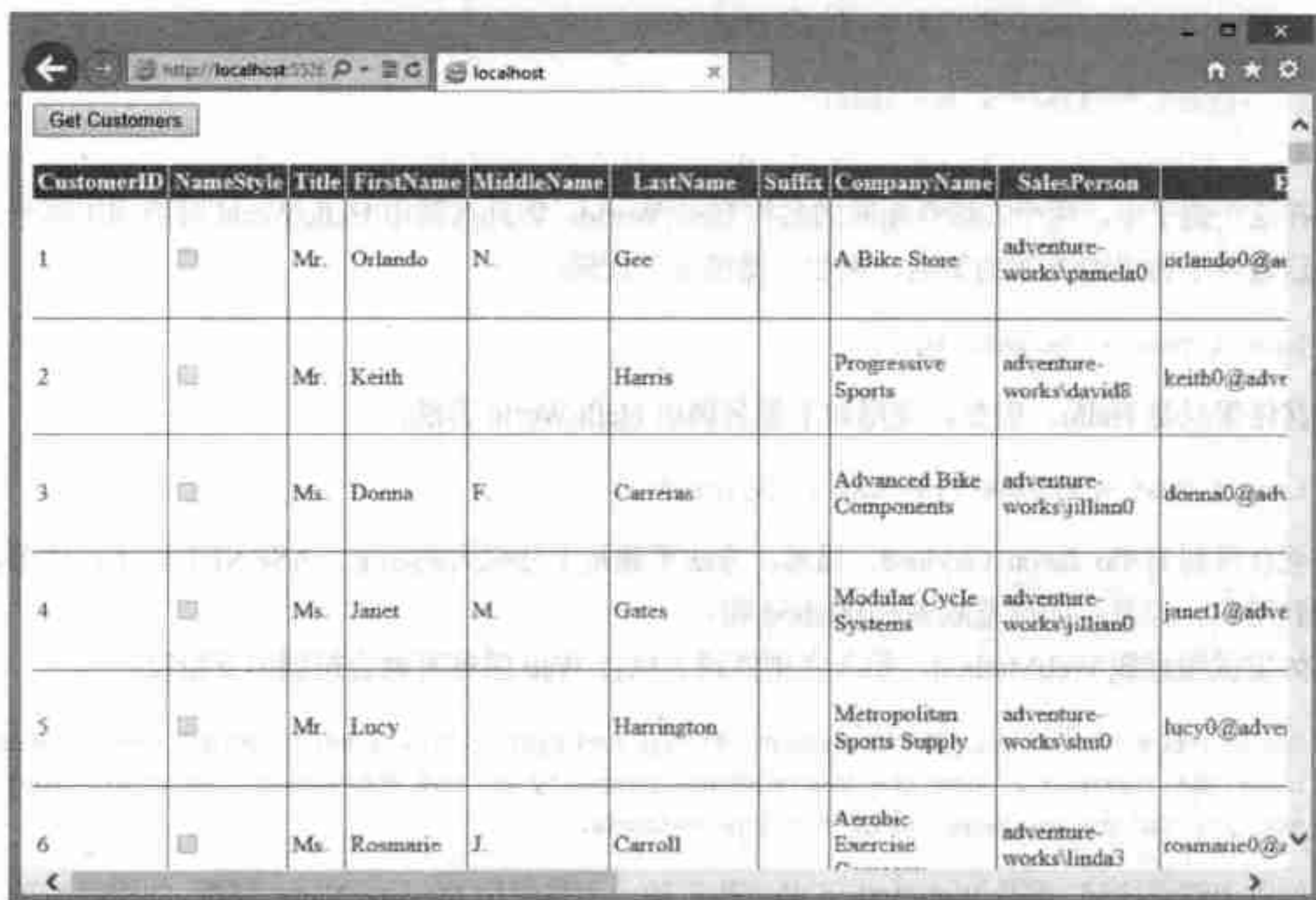
```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Button1_Click(Object sender, EventArgs e)
    {
        AdventureWorksCustomers.Customers ws = new AdventureWorksCustomers.Customers();
        GridView1.DataSource = ws.GetCustomers();
        GridView1.DataBind();
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Web Service Consumer Example</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Button ID="Button1" runat="server" Text="Get Customers"
                OnClick="Button1_Click" />
            <br />
            <br />
            <asp:GridView ID="GridView1" runat="server" BorderWidth="1px"
                BackColor="#DEBA84" CellPadding="3" CellSpacing="2" BorderStyle="None"
                BorderColor="#DEBA84">
                <FooterStyle ForeColor="#8C4510" BackColor="#F7DFB5"></FooterStyle>
                <PagerStyle ForeColor="#8C4510" HorizontalAlign="Center"></PagerStyle>
                <HeaderStyle ForeColor="White" Font-Bold="True"
                    BackColor="#A55129"></HeaderStyle>
                <SelectedRowStyle ForeColor="White" Font-Bold="True"
                    BackColor="#738A9C"></SelectedRowStyle>
                <RowStyle ForeColor="#8C4510" BackColor="#FFF7E7"></RowStyle>
            </asp:GridView>
        </div>
    </form>
</body>
</html>
```

```

    </form>
</body>
</html>

```

终端用户会看到一个简单的按钮。单击它，ASP.NET 应用程序就会给远程 XML Web 服务发送 SOAP 请求。返回的 DataSet 绑定到 GridView 控件，并重新绘制页面，如图 13-10 所示。



The screenshot shows a web browser window with the address bar set to 'http://localhost:5516'. Below the address bar is a button labeled 'Get Customers'. Below the button is a GridView table displaying customer information. The table has 10 columns: CustomerID, NameStyle, Title, FirstName, MiddleName, LastName, Suffix, CompanyName, SalesPerson, and Email. There are 6 rows of data.

CustomerID	NameStyle	Title	FirstName	MiddleName	LastName	Suffix	CompanyName	SalesPerson	Email
1		Mr.	Orlando	N.	Gee		A Bike Store	adventure-works/pamela0	orlando0@adventure-works.com
2		Mr.	Keith		Harris		Progressive Sports	adventure-works/david8	keith0@adventure-works.com
3		Ms.	Donna	F.	Carreno		Advanced Bike Components	adventure-works/jillian0	donna0@adventure-works.com
4		Ms.	Janet	M.	Gates		Modular Cycle Systems	adventure-works/jillian0	janet1@adventure-works.com
5		Mr.	Lucy		Harrington		Metropolitan Sports Supply	adventure-works/shu0	lucy0@adventure-works.com
6		Ms.	Rosmarie	J.	Carroll		Aerobic Exercise Equipment	adventure-works/linda3	rosmarie0@adventure-works.com

图 13-10

Customers Web 服务由 AdventureWorksCustomers.Customers 代理对象的实例调用：

```
AdventureWorksCustomers.Customers ws = new AdventureWorksCustomers.Customers();
```

然后，就可以像使用项目中的其他对象那样使用 ws 对象。在程序清单 13-7 的代码示例中，把 ws.GetCustomers() 方法调用的结果赋予 GridView 控件的 DataSource 属性：

```
GridView1.DataSource = ws.GetCustomers();
```

在应用程序中开发或使用更多的 Web 服务，就会看出它们的强大和效用。

13.4 重载 WebMethod

在 .NET 的面向对象模型中，可以在开发的代码中使用方法重载。面向对象的语言支持多态，而方法重载是多态的一部分。方法重载允许多个方法使用相同的名称，但其签名不同。通过方法重载，可以调用一个方法，但该调用会根据请求的完整签名路由到合适的方法。标准的方法重载如程序清单 13-8 所示。

程序清单 13-8 .NET 中的方法重载

```
public string HelloWorld()
{
    return "Hello";
}
public string HelloWorld(string FirstName)
{
    return "Hello " + FirstName;
}
```

在这个例子中，两个方法有相同的名称 HelloWorld，因此在调用 HelloWorld 时应调用哪个方法呢？这取决于传送给方法的签名。例如，提供如下代码：

```
Label1.Text = HelloWorld();
```

其结果只是 Hello。但是，使用如下签名调用 HelloWorld 方法：

```
Label1.Text = HelloWorld("Jason Gaylord");
```

就会得到 Hello Jason Gaylord。显然，方法重载是十分强大的功能，ASP.NET 应用程序可以有效地利用它。但是，如何重载 WebMethod 呢？

如果试图重载 WebMethod，那么在浏览器上执行 Web 服务时就会得到如下错误：

```
Both System.String HelloWorld(System.String) and System.String HelloWorld() use the message name 'HelloWorld'. Use the MessageName property of the WebMethod custom attribute to specify unique message names for the methods.
```

如以上错误所述，重载 WebMethod 必须执行的一步是使用 MessageName 属性，如程序清单 13-9 所示(本章下载代码中的 WebService.cs)。

程序清单 13-9 在.NET 中重载 WebMethod

```
[WebMethod(MessageName = "HelloWorld")]
public string HelloWorld() {
    return "Hello World";
}
[WebMethod(MessageName = "HelloWorldWithFirstName")]
public string HelloWorld(string FirstName)
{
    return "Hello " + FirstName;
}
```

除了添加 WebMethod 特性的 MessageName 属性之外，还必须禁止 Web 服务遵循 WS-I Basic Profile 规范：如果为 Web 服务重载 WebMethod，Web 服务就不遵循该规范。禁止遵循 WS-I Basic Profile 规范有两种方式。第一种方式是给代码添加<WebServiceBinding>特性，如程序清单 13-10 所示。

程序清单 13-10 修改 Web 服务以使其不遵循 WS-I Basic Profile 规范

```
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.None)]
```



```
public class WebService : System.Web.Services.WebService
{
    // Code here...
}
```

另一种方式是在 web.config 文件中关闭 WS-I Basic Profile 功能，如程序清单 13-11 所示。

程序清单 13-11 使用 web.config 文件关闭 WS-I Basic Profile 功能

```
<configuration>
  <system.web>
    <webServices>
      <conformanceWarnings>
        <remove name="BasicProfile1_1" />
      </conformanceWarnings>
    </webServices>
  </system.web>
</configuration>
```

允许 Web 服务重载 WebMethod 后，在浏览器中打开 Web 服务的接口测试页面时，就可以看到 MessageName 属性值定义的两个 WebMethod，如图 13-11 所示。

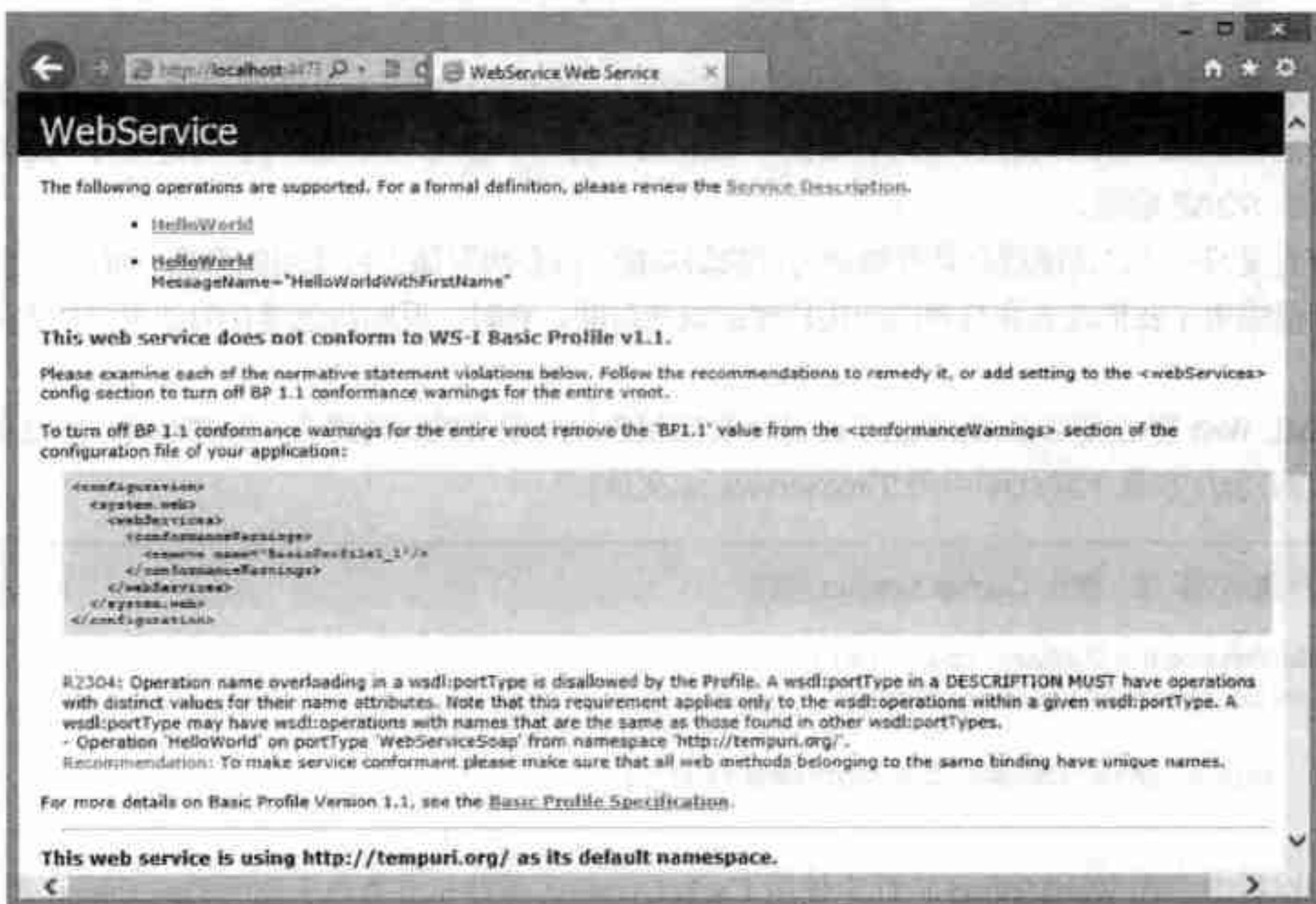


图 13-11

尽管两个 WebMethod 方法的名称是相同的，但 MessageName 属性显示它们是不同的方法。在使用该 Web 服务的开发人员建立对 Web 服务的 Web 引用时，将只看到一个方法名(在本例中是 HelloWorld)。在使用这些方法的应用程序中，该方法名显示在 Visual Studio 2012 的 IntelliSense 中，如图 13-12 所示。



图 13-12

在弹出的方框中为开发人员提供了方法的签名结构，其中包含两个选项：一个签名是空的，另一个签名需要一个字符串。

13.5 高速缓存 Web 服务的响应

高速缓存是每一个使用.NET 建立的应用程序的一项重要功能。ASP.NET 中的大多数高速缓存功能都已在第 22 章中讨论过，除了.NET 中 Web 服务的一个功能，该功能可以高速缓存发送给服务使用者的 SOAP 响应。

首先复习一下，高速缓存具有维护内存块的功能，内存块存储了可重用的数据、对象和各种项。这个功能缩短了我们建立和管理的应用程序的响应时间。有时，返回高速缓存的结果可以大大提高性能。

XML Web 服务使用 `CacheDuration` 属性来控制 SOAP 响应的高速缓存，程序清单 13-12 演示了该属性的用法(本章下载代码中的 `TimeService.cs` 文件)。

程序清单 13-12 使用 `CacheDuration` 属性

```

[WebMethod(CacheDuration = 60)]
public string GetServerTime()
{
    return DateTime.Now.ToLongTimeString();
}

```

可以看出，在 `WebMethod` 特性中使用 `CacheDuration` 属性非常类似于使用 `Description` 和 `Name` 属性。`CacheDuration` 属性带一个 `Integer` 值，它等于高速缓存 SOAP 响应的秒数。

在收到第一个请求时，SOAP 响应就由服务器高速缓存起来，用户在下一分钟会得到 SOAP 响应中相同的时间戳。经过这一分钟后，就抛弃该高速缓存，生成一个新的响应，并再次存储在高速缓存中，用于服务下一分钟的所有其他请求。

高速缓存 SOAP 响应后，如果每次重新建立的响应没有任何改变，应用程序的性能就会大大提高。

13.6 使用 SOAP 报头

要扩展 SOAP 消息的功能，一种常见形式是把请求的元数据添加到 SOAP 消息中。元数据通常添加到 SOAP 封套的名为“SOAP 报头”的部分中。图 13-13 演示了 SOAP 消息的结构。

整个 SOAP 消息称为 SOAP 封套。包含在 SOAP 消息中的是 SOAP 主体，这是前面每个例子中操作的部分。SOAP 主体是 SOAP 消息的必备元素。

SOAP 消息还有一个可选的组成部分，称为 SOAP 报头。在 SOAP 报头中可以放置与整个 SOAP 请求相关的元数据，而不是在 WebMethod 的签名中集成它们。使元数据与实际请求分开是很重要的。

在 SOAP 报头中放置哪些类型的信息？SOAP 报头可以包括许多信息，一种常见的信息是使用 Web 服务或获取某些逻辑或数据所需的身份验证和授权信息。例如，可以把用户名和密码放在消息的 SOAP 报头中。

13.6.1 使用 SOAP 报头建立 Web 服务

下面的例子基于 Web 服务 HelloWorld，在 Visual Studio 中第一次使用这个 Web 服务时，它会显示在默认的.asmx 页面中，如程序清单 13-4 所示。把新的.asmx 文件命名为 HelloSoapHeader.asmx。第一步是添加一个类，它是表示客户端要在 SOAP 报头中放置的内容的对象，如程序清单 13-13 所示。



图 13-13

程序清单 13-13 表示 SOAP 报头的类

```
public class HelloHeader : System.Web.Services.Protocols.SoapHeader
{
    public string Username;
    public string Password;
}
```

这个类表示 SOAP 报头对象，它必须继承自 System.Web.Services.Protocols.SoapHeader 中的 SoapHeader 类。SoapHeader 类把<soap:header>元素的有效负载序列化到 XML 中。在程序清单 13-13 中，这个 SOAP 报头需要两个元素——用户名和密码，它们都是 String 类型。在这个类中创建的名称用于 SOAP 报头构造的子元素，因此给它们指定描述性的名称非常重要。

程序清单 13-14 中的 Web 服务类创建了 HelloHeader 类的一个实例。

程序清单 13-14 使用 SOAP 报头的 Web 服务类

```
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1,
    EmitConformanceClaims = true)]
public class HelloSoapHeader : System.Web.Services.WebService {
    public HelloHeader myHeader;

    [WebMethod]
    [SoapHeader("myHeader")]
```



```

    public string HelloWorld()
    {
        if(myHeader == null)
        {
            return "Hello World";
        }
        else
        {
            return "Hello " + myHeader.Username + ". " +
                "<br>Your password is: " + myHeader.Password;
        }
    }
}

```

Web 服务 `HelloSoapHeader` 包含单个 `WebMethod: HelloWorld`。在 Web 服务类中(但在 `WebMethod` 的外部)创建 `SoapHeader` 类的一个实例。可使用如下代码实现该操作:

```
public myHeader As HelloHeader
```

创建了 `HelloHeader` 类的实例 `myHeader` 后,就可以在 `WebMethod` 中使用该实例。Web 服务可以包含任意多个 `WebMethod`,因此所有的 `WebMethod` 不一定要使用实例化的 SOAP 报头。把 `SoapHeader` 特性放在 `WebMethod` 声明的前面,就可以指定该 `WebMethod` 是否使用 SOAP 报头类的特定实例:

```

[WebMethod]
[SoapHeader("myHeader")]
public string HelloWorld()
{
    // Code here...
}

```

在这个例子中, `SoapHeader` 特性带一个字符串值,该值是 `SoapHeader` 类的实例名,这里是 `myHeader`。

之后, `WebMethod` 开始使用 `myHeader` 对象。如果没有找到 `myHeader` 对象(表示客户没有在构造的 SOAP 消息中发送 SOAP 报头),就返回简单的 `Hello World`。如果在 SOAP 请求的 SOAP 报头中提供了值,就在返回的字符串值中使用这些值。

13.6.2 通过 SOAP 报头使用 Web 服务

建立一个 ASP.NET 应用程序,使用 SOAP 报头给 Web 服务发送 SOAP 请求并不困难。与不包含 SOAP 报头的 Web 服务一样,需要在 Visual Studio 中建立对远程 Web 服务的 Web 引用。

对于 ASP.NET 页面,创建只包含一个 `Label` 控件的简单页面,将 Web 服务的输出放在该 `Label` 控件中。该 ASP.NET 页面的代码如程序清单 13-15 所示。

程序清单 13-15 通过 SOAP 报头使用 XML Web 服务的 ASP.NET 页面

```

<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e)

```

```

    {
        helloSoapHeader.HelloSoapHeader ws = new helloSoapHeader.HelloSoapHeader();
        helloSoapHeader.HelloHeader wsHeader = new helloSoapHeader.HelloHeader();
        wsHeader.Username = "Jason Gaylord ";
        wsHeader.Password = "Lights";
        ws.HelloHeaderValue = wsHeader;
        Label1.Text = ws.HelloWorld();
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Working with SOAP headers</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="Label1" Runat="server"></asp:Label>
        </div>
    </form>
</body>
</html>

```

上面的代码实例化了两个对象。第一个是 Web 服务对象 `HelloSoapHeader`。第二个是 `SoapHeader` 对象，它被实例化为 `wsHeader`。实例化这两个对象之后，在应用程序中发出 SOAP 请求之前构建 SOAP 报头。该操作很简单，只需给 `wsHeader` 对象的 `Username` 和 `Password` 属性赋值即可。之后，使用 `HelloHeaderValue` 属性把 `wsHeader` 对象关联到 `ws` 对象。在建立好的 SOAP 报头对象和 `WebMethod` 对象(`ws`)之间建立关联之后，就可以如往常一样发出 SOAP 请求：

```
Label1.Text = ws.HelloWorld();
```

运行页面，在浏览器中产生的结果如图 13-14 所示。

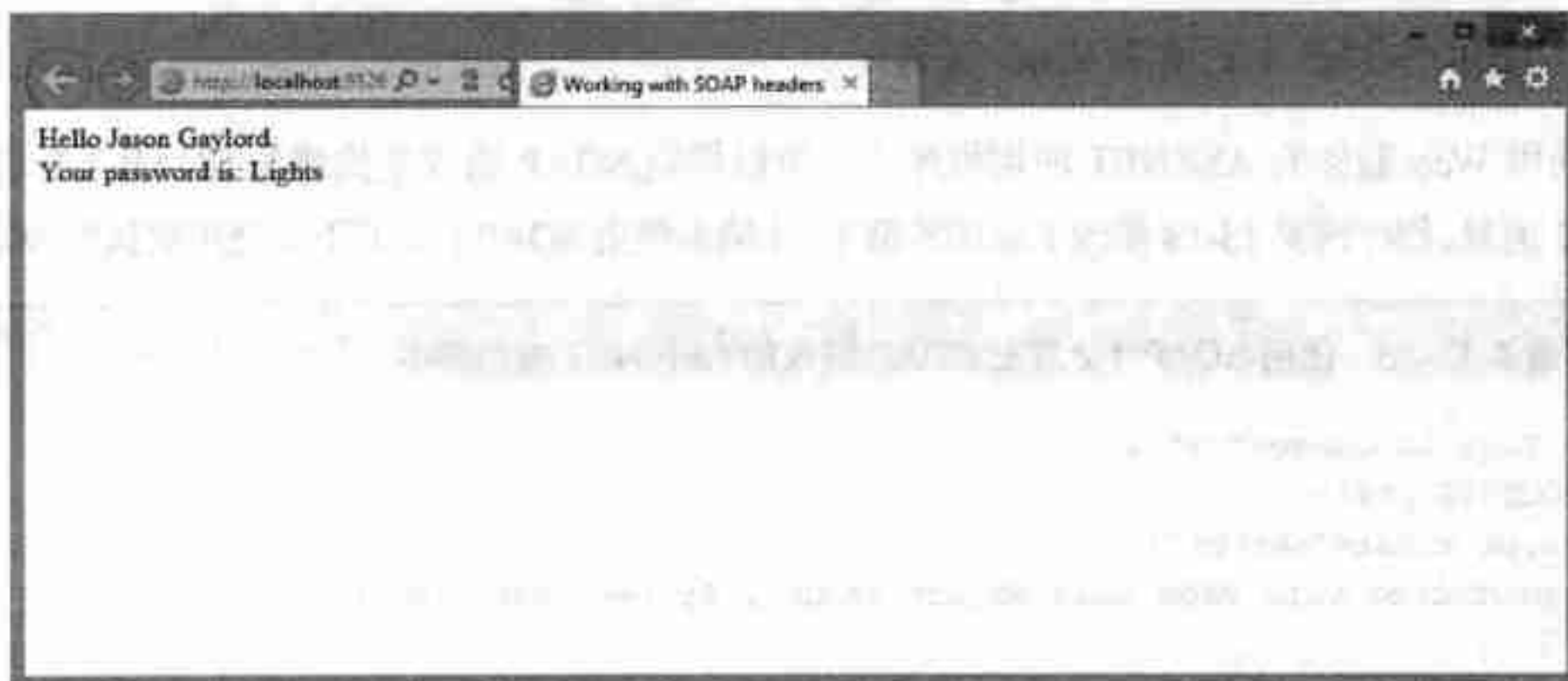


图 13-14

有趣的是，SOAP 请求会显示，SOAP 报头实际上构建在整个 SOAP 消息内部，如程序清单 13-16 所示。

程序清单 13-16 SOAP 请求

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Header>
    <HelloHeader xmlns="http://tempuri.org/">
      <Username>Jason Gaylord</Username>
      <Password>Lights</Password>
    </HelloHeader>
  </soap:Header>
  <soap:Body>
    <HelloWorld xmlns="http://tempuri.org/" />
  </soap:Body>
</soap:Envelope>
```

这段代码会返回如程序清单 13-17 所示的 SOAP 响应。

程序清单 13-17 SOAP 响应

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <HelloWorldResponse xmlns="http://tempuri.org/">
      <HelloWorldResult>Hello Jason Gaylord. Your password is:
        Lights</HelloWorldResult>
    </HelloWorldResponse>
  </soap:Body>
</soap:Envelope>
```

13.6.3 使用 SOAP 1.2 请求 Web 服务

在使用 Web 服务的 ASP.NET 应用程序中，可以控制 SOAP 请求是构建为 SOAP 1.1 消息还是 SOAP 1.2 消息。程序清单 13-18 修改了前面的例子，让请求使用 SOAP 1.2，而不是使用默认的 SOAP 1.1。

程序清单 13-18 使用 SOAP 1.2 发出 SOAP 请求的 ASP.NET 应用程序

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
  protected void Page_Load(object sender, System.EventArgs e)
  {
    helloSoapHeader.HelloSoapHeader ws = new helloSoapHeader.HelloSoapHeader();
    helloSoapHeader.HelloHeader wsHeader = new helloSoapHeader.HelloHeader();
    wsHeader.Username = "Jason Gaylord";
    wsHeader.Password = "Lights";
    ws.HelloHeaderValue = wsHeader;
    ws.SoapVersion = System.Web.Services.Protocols.SoapProtocolVersion.Soap12;
    Labell.Text = ws.HelloWorld();
  }
</script>
```


在这个例子中，首先提供了 Web 服务对象的实例，并使用了 SoapVersion 属性。该属性的值是 System.Web.Services.Protocols.SoapProtocolVersion.Soap12，表示使用 SOAP 1.2。

有了这些代码，SOAP 请求就采用如程序清单 13-19 所示的结构。

程序清单 13-19 使用 SOAP 1.2 的 SOAP 请求

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Header>
    <HelloHeader xmlns="http://tempuri.org/">
      <Username>Jason Gaylord</Username>
      <Password>Lights</Password>
    </HelloHeader>
  </soap:Header>
  <soap:Body>
    <HelloWorld xmlns="http://tempuri.org/" />
  </soap:Body>
</soap:Envelope>
```

这两个例子的区别在于它们使用的 xmlns:soap 名称空间。实际上区别在于 HTTP 报头。比较 SOAP 1.1 和 1.2 消息，会发现 Content-Type 特性有所区别。另外，SOAP 1.2 HTTP 报头没有使用 soapaction 特性，因为它现在与 Content-Type 特性合并。

在 web.config 文件中使用合适的设置，就可以在 Web 服务中关闭 SOAP 1.1 或 1.2 功能，如程序清单 13-20 所示。

程序清单 13-20 关闭 SOAP 1.1 或 1.2 功能

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <webServices>
      <protocols>
        <remove name="HttpSoap"/> <!-- Removes SOAP 1.1 abilities -->
        <remove name="HttpSoap1.2"/> <!-- Removes SOAP 1.2 abilities -->
      </protocols>
    </webServices>
  </system.web>
</configuration>
```

13.7 异步使用 Web 服务

本章使用的所有 Web 服务都是同步执行的。也就是说，从 ASP.NET 应用程序的代码中发出请求后，应用程序就会完全停止运行，直到接收到 SOAP 响应为止。

对于某些请求，调用 WebMethod 并返回结果的过程需要较长时间。有时，不能控制从中请求数据的 Web 服务，也就不能控制这些服务的性能或响应时间。因此，应考虑异步使用 Web 服务。

发出异步请求的 ASP.NET 应用程序可以在最初的 SOAP 请求等待响应的同时执行其他编程任务。完成了其他任务后，ASP.NET 应用程序再从 Web 服务处获得结果。

要建立可以异步通信的 XML Web 服务，不需要执行任何额外的操作。所有的 .asmx Web 服务都

内置了与客户端异步通信的功能。程序清单 13-21 中的 Web 服务就是这样一个例子(这个程序清单包含在本章下载代码的 HelloWorldAsyncService.cs 文件中)。

程序清单 13-21 一个较慢的 Web 服务

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class HelloWorldAsyncService : System.Web.Services.WebService {
    [WebMethod]
    public string HelloWorld() {
        System.Threading.Thread.Sleep(1000);
        return "Hello World";
    }
}
```

这个 Web 服务把简单的 Hello World 返回为字符串,但在此之前,Web 服务有 1000 毫秒的暂停,这是通过使用 Sleep 方法使 Web 服务线程进入睡眠状态达到的效果。

下面查看 ASP.NET 应用程序如何异步使用这个较慢的 Web 服务,如程序清单 13-22 所示。

程序清单 13-22 异步使用 Web 服务的 ASP.NET 应用程序

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e)
    {
        helloWorldAsync.HelloWorldAsyncService ws =
            new helloWorldAsync.HelloWorldAsyncService();
        IAsyncResult myIar;
        myIar = ws.BeginHelloWorld(null, null);
        int x = 0;

        while(myIar.IsCompleted == false)
        {
            x += 1;
        }

        Label1.Text = "Result from Web service: " + ws.EndHelloWorld(myIar) +
            "<br>Local count while waiting: " + x.ToString();
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Async consumption</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
```



```

        <asp:Label ID="Label1" Runat="server"></asp:Label>
    </div>
</form>
</body>
</html>

```

在 ASP.NET 应用程序中对远程 Web 服务建立 Web 引用时, 不仅可以在 IntelliSense 中看到 WebMethod HelloWorld, 还会看到 BeginHelloWorld 和 EndHelloWorld 方法。要异步使用 Web 服务, 就必须利用 BeginHelloWorld 和 EndHelloWorld 方法。

使用 BeginHelloWorld 方法给 Web 服务发送 SOAP 请求, 但 ASP.NET 应用程序并不等待响应, 而是继续完成其他任务。在这个例子中, 它并没有做什么重要的工作, 只是计算它进入循环的次数。

从 ASP.NET 应用程序中发出 SOAP 请求后, 就可以使用 IAsyncResult 对象检查是否正在等待 SOAP 响应。为此, 应使用 mylar.IsCompleted。如果异步调用未完成, ASP.NET 应用程序就递增 x 的值, 之后再次进行检查。直到 XML Web 服务准备返回响应时, ASP.NET 应用程序才停止检查。该应用程序使用 EndHelloWorld 方法调用来检索响应。

运行这个应用程序, 结果如图 13-15 所示。

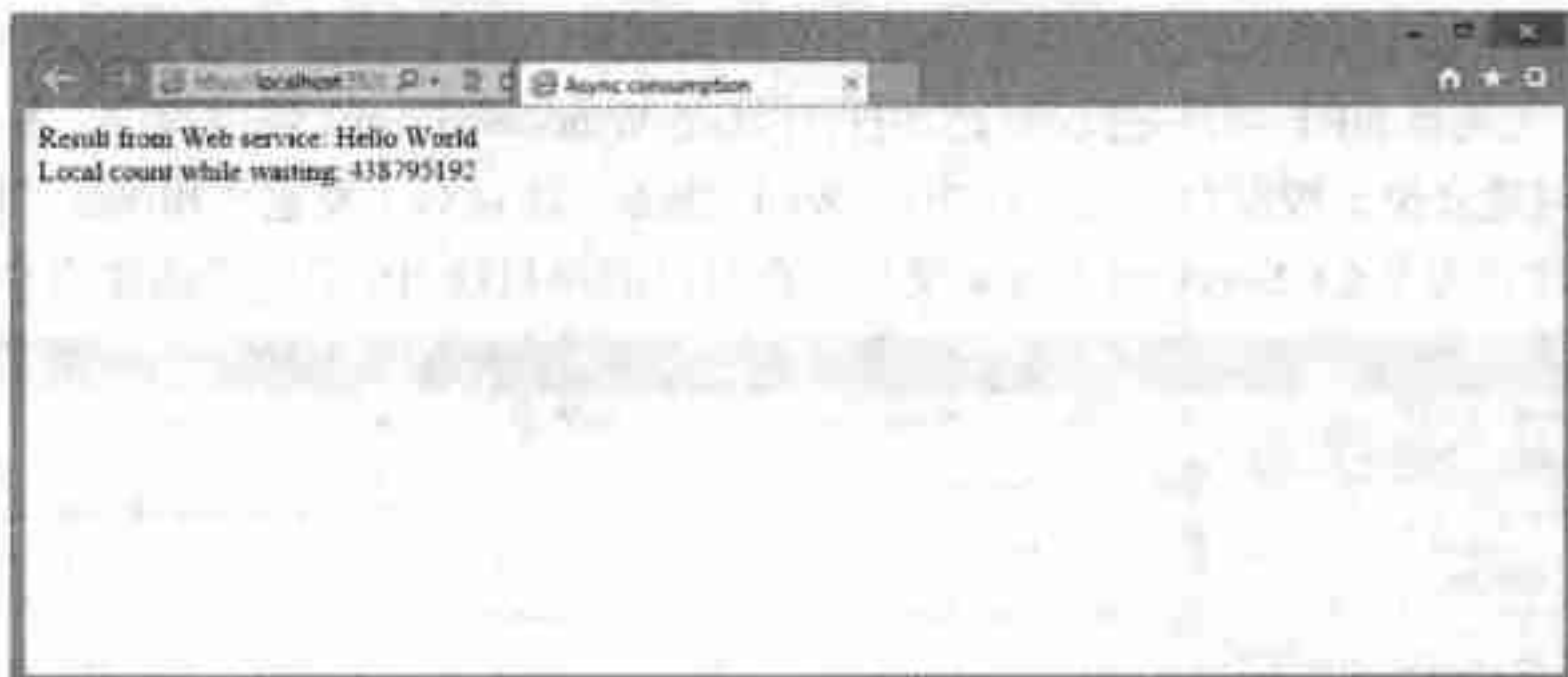


图 13-15

13.8 WCF

以前, 建立在两个点之间传递消息所需的组件不是一项简单的任务, 因为 Microsoft 提供了多种可用于此操作的技术。后来, Microsoft 创建了 Windows Communication Foundation (WCF), 帮助开发人员决定使用哪个技术建立应用程序。

WCF 是建立面向服务的应用程序的新架构。Microsoft 希望为开发人员提供一个架构, 能够快速建立并运行面向服务的正确体系结构。使用 WCF, 可以利用使分布式技术更强大的所有选项。WCF 是所有这些消息分布式技术的继任者。

13.8.1 WCF 概述

如前所述, WCF 是在 Microsoft 环境下建立分布式应用程序的一种方式。尽管分布式应用程序基于这个环境, 但这并不意味着使用者必须是 Microsoft 客户, 或者必须采用 Microsoft 组件或技术来完成该任务。另一方面, 建立 WCF 服务表示, 所建立的服务遵循前面讨论 SOA 时提出的原则, 并且这些服务没有与特定供应商关联, 因此它们几乎可以由任何人使用。



使用 Visual Studio 2012 可以建立 WCF 服务。注意这是 .NET Framework 3.0 或更高版本中的组件，因此可以运行 WCF 服务的操作系统是受限的。使用 WCF 建立的应用程序只能运行在 Windows XP SP2 及更高版本中。

如果已经熟悉 WCF，那么在 WCF 4.5 中将会发现一些改进。这些改进重点放在了提高开发人员的效率，并为一般性任务提供快速选项，诸如创建异步 Web 服务、简化 WCF 配置、提供 HTML5 WebSocket 支持。其他的新功能包括：单个端点的多种验证方法，提高文件流的性能，配置的有效性验证等。

13.8.2 建立 WCF 服务

建立 WCF 服务并不难以实现，前提是已经安装了 .NET Framework 4.5。如果使用 Visual Studio 2012，New Web Site 对话框中的项目视图就如图 13-16 所示。

以这种方式建立 WCF 项目时，会建立一个传统的类库，将该类库编译为一个 DLL，接着添加到另一个项目中。代码和项目的分离是大型项目的强大部分。也就是说，可以直接在 .NET 项目中建立 WCF 服务，无论该 .NET 项目是控制台应用程序还是 Windows Forms 应用程序。本章示例采用的方法说明了如何建立位于控制台应用程序中的 WCF 服务。注意对于要建立和部署的服务，通常最好直接把它们建立为 WCF Service Library 项目，在自己的项目或 IIS 中使用创建的 DLL。

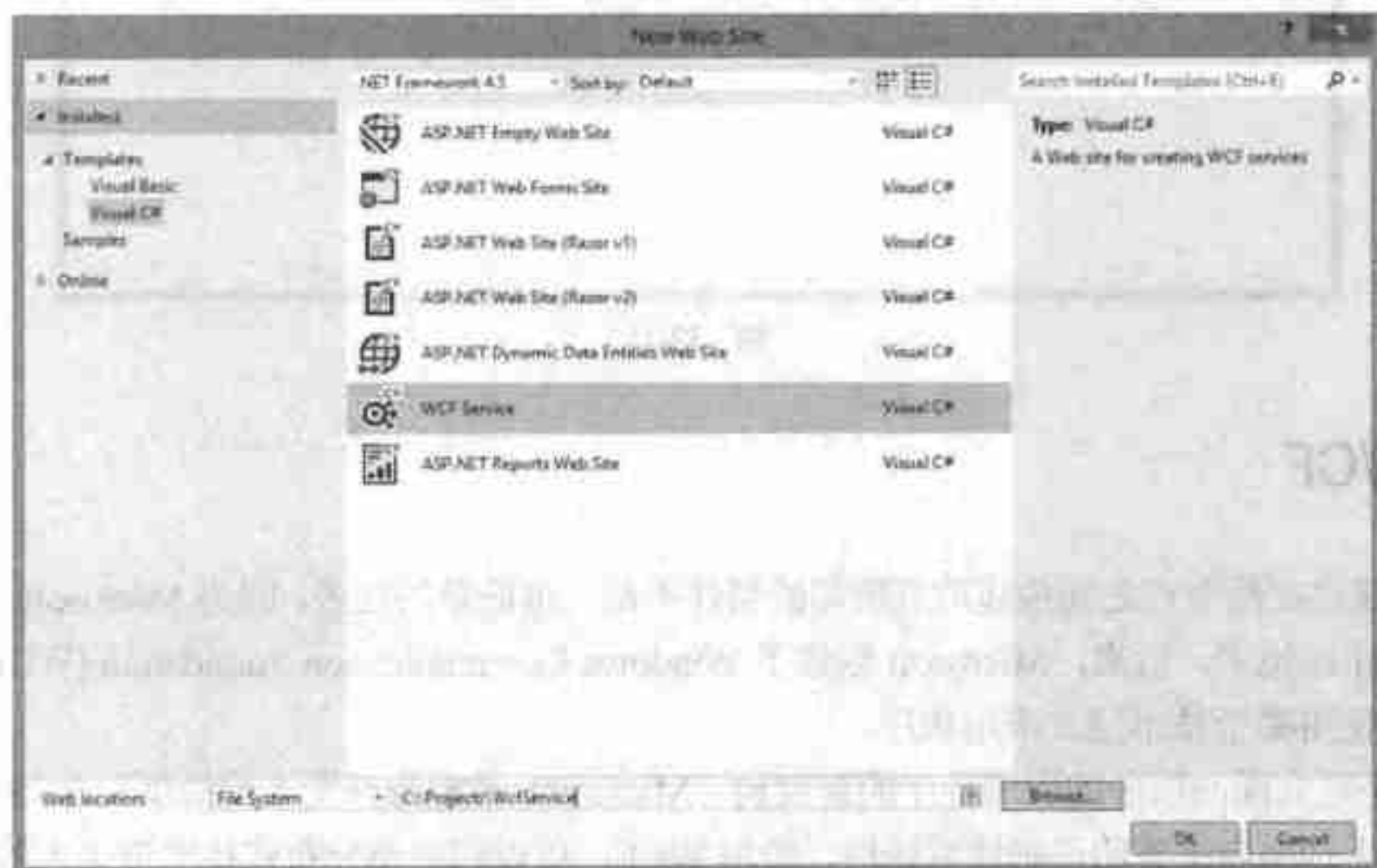


图 13-16

在开始建立 WCF 服务之前，请首先考虑建立在 WCF 架构基础上的服务的组成。

1. WCF 服务的组成

查看 WCF 服务，你会发现它由 3 部分组成：服务、一个或多个端点以及驻留该服务的环境。

服务是使用某种兼容 .NET 的语言编写的类。该类可以包含一个或多个通过 WCF 服务显示出来的方法。一个服务可以有一个或多个端点，端点用于通过服务与客户端通信。

端点本身也由 3 部分组成，Microsoft 通常把这些部分定义为 WCF 的 ABC，每个字母都表示

WCF 模型中的某个对象, 包括:

- A 表示地址(address)
- B 表示绑定(binding)
- C 表示协定(contract)

可以把 ABC 看作: A 是位置, B 是方式, C 是内容。最后, 主机环境是包含服务的地方, 它由应用程序域和过程组成。将这 3 个元素(服务、端点和主机环境)组合在一起就可以创建 WCF 服务, 如图 13-17 所示。

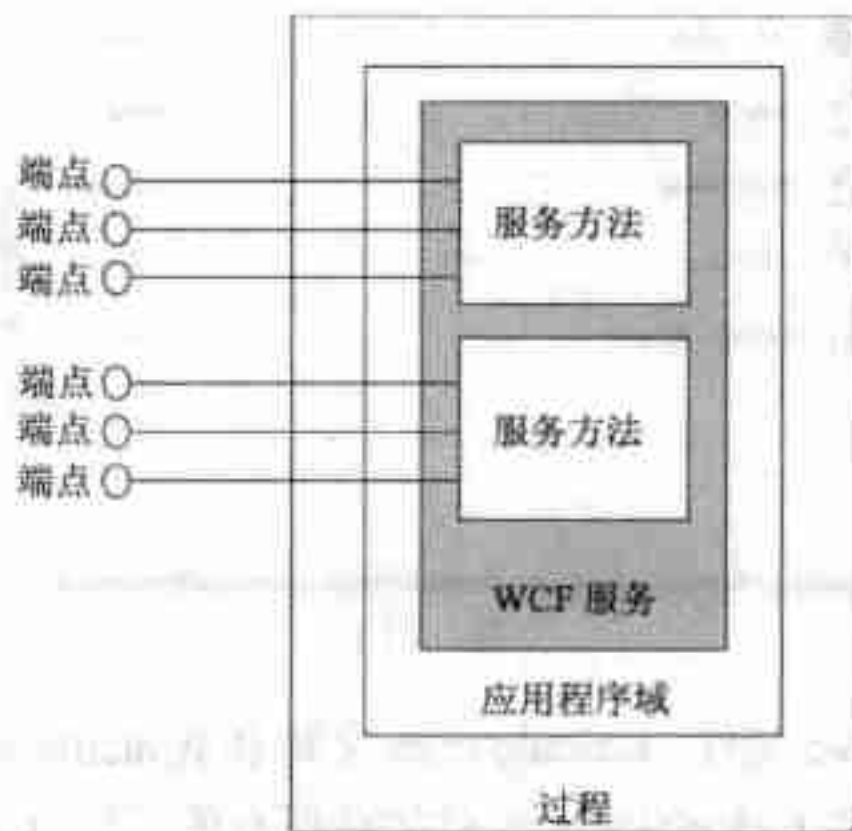


图 13-17

下一步是使用 WCF 架构创建一个基本服务。

2. 创建第一个 WCF 服务

要在驻留之前建立服务, 必须执行两个主要步骤。首先, 需要创建服务协定。其次, 必须创建数据协定。服务协定实际上是类, 其中的方法要从 WCF 服务中提取。数据协定也是类, 它指定了要从接口中提取的结构。

有了一个服务类后, 就可以把它驻留在任何地方。在 Visual Studio 2012 中运行这个服务时, 可以使用任何标准 ASP.NET 应用程序所使用的内置驻留机制。要建立第一个 WCF 应用程序, 可以从 Visual Studio 2012 菜单中选择 File | New | Web Site 命令, 把项目命名为 WCFService1。



为了保持本章的一致性, 这些项目的 C# 版本被命名为 Chapter13-WCFService-CS。

本章使用的例子演示了如何建立接口, 再建立服务本身, 从而建立 WCF 服务。

建立服务架构

第一步是在项目中创建服务架构。为此, 右击项目并从弹出的菜单中选择 Add New Item 命令。在打开的 Add New Item 对话框中, 选择 WCF Service 选项, 把服务命名为 Calculator.svc, 如图 13-18 所示。



图 13-18

这一步创建了 Calculator.svc 文件、Calculator.cs 文件和 ICalculator.cs 文件。Calculator.svc 文件是只包含页面指令的简单文件，而 Calculator.cs 文件完成所有的工作，Calculator.cs 文件是 ICalculator.cs 接口的实现。

使用接口

要创建服务，就需要服务协定。服务协定是服务的接口，包含所有的方法，以及调用方法所需的输入输出参数。为了完成这项任务，应使用 ICalculator.cs。需要创建的接口如程序清单 13-23 所示(本章下载代码中的 Chapter13-WCFService-CS 文件夹中的 ICalculator.cs)。

程序清单 13-23 创建接口

```
using System.ServiceModel;

[ServiceContract]
public interface ICalculator
{
    [OperationContract]
    int Add(int a, int b);

    [OperationContract]
    int Subtract(int a, int b);

    [OperationContract]
    int Multiply(int a, int b);

    [OperationContract]
    int Divide(int a, int b);
}
```

这是正常的接口定义，但带有几个新特性。为了访问这些需要的特性，应引用 System.ServiceModel

名称空间，这样才能访问 `ServiceContract` 和 `OperationContract` 特性。

`ServiceContract` 特性用于把类或接口定义为服务类，它需要放在类或接口的起始声明之前。在这里，上述代码中的例子基于如下接口：

```
[ServiceContract]
public interface ICalculator
{
    // Code removed for clarity
}
```

在接口中定义了 4 个方法，每个方法都通过服务协定中的 WCF 服务展示出来。因此，每个方法都要应用 `OperationContract` 特性：

```
[OperationContract]
int Add(int a, int b);
```

利用接口

下一步是创建一个实现接口的类。这个新类不仅实现了已定义的接口，还实现了服务协定。对于这个例子，把这个类添加到 `Calculator.cs` 文件中。程序清单 13-24 列出了这个接口的实现代码。

程序清单 13-24 实现接口

```
public class Calculator : ICalculator
{
    public int Add(int a, int b)
    {
        return (a + b);
    }

    public int Subtract(int a, int b)
    {
        return (a - b);
    }

    public int Multiply(int a, int b)
    {
        return (a * b);
    }

    public int Divide(int a, int b)
    {
        return (a / b);
    }
}
```

从这些新增的代码中可以看出，不必执行与 `Calculator` 类不同的操作。这是一个实现了 `ICalculator` 接口的简单类，提供了 `Add`、`Subtract`、`Multiply` 和 `Divide` 方法的实现代码。

有了接口和类后，就建立好了 WCF 服务。下一步是驻留该服务。注意这是一个简单的服务，它只提供简单的类型，没有提供复杂类型。因此，可以只建立一个服务协定，而无须完成数据协定的构建。数据协定的构建详见本章后面的内容。

在控制台应用程序中驻留 WCF 服务

接下来提取所开发的服务，把它驻留在某种类型的应用程序进程中。为此，有许多驻留选项，包括：

- 控制台应用程序
- Windows Forms 应用程序
- WPF 应用程序
- 托管的 Windows 服务
- Internet Information Services (IIS) 5.1 及更高版本
- IIS Express

如前所述，这个例子把服务驻留在 Visual Studio 2012 提供的 IIS Express 中。有两种方式可以激活驻留功能：直接对驻留行为编码，或者通过声明式编程(通常通过配置文件完成)。

编译并运行这个应用程序，结果如图 13-19 所示。

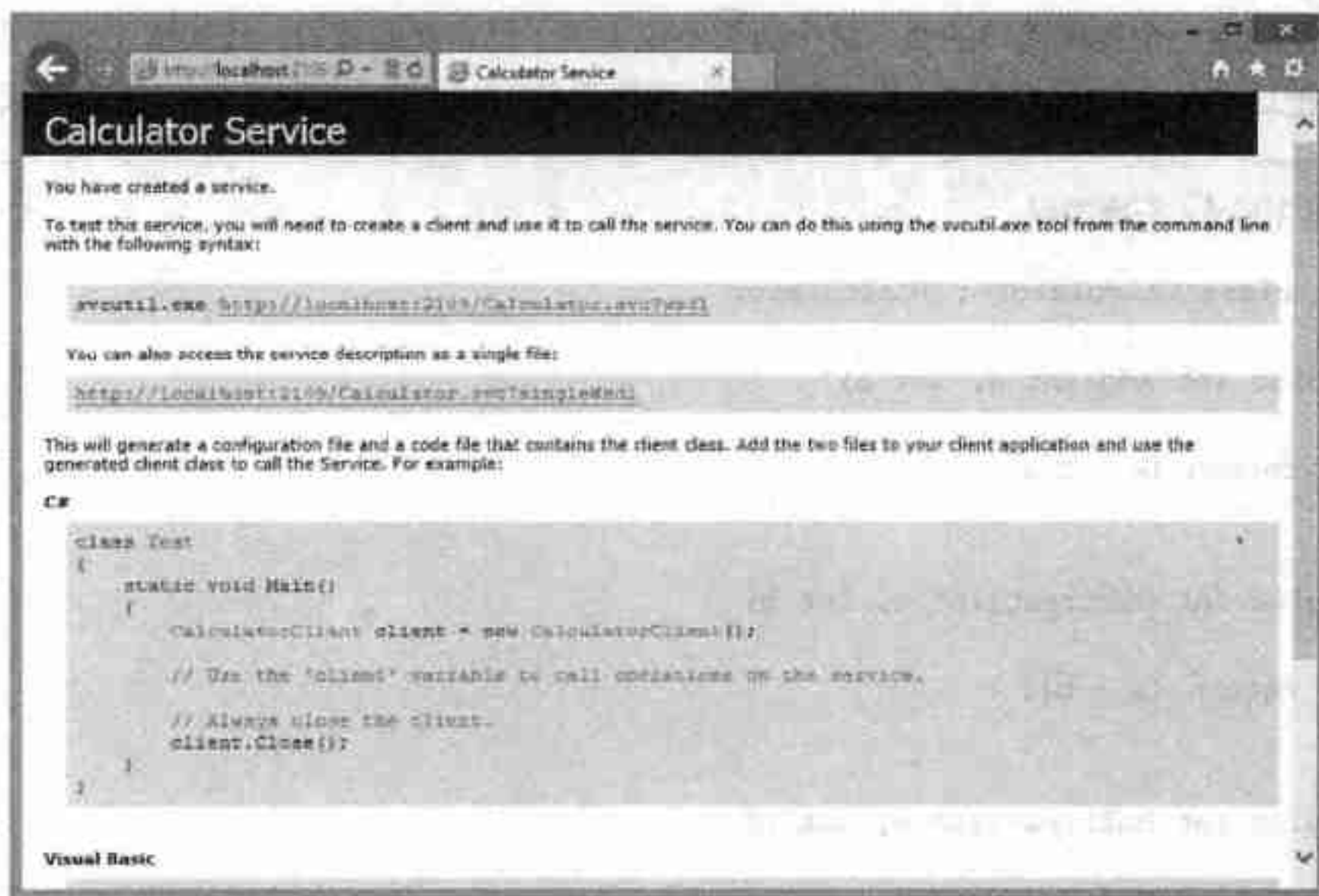


图 13-19

注意，这个结果页面非常类似于建立 ASP.NET Web 服务时的结果页面。

3. 审阅 WSDL 文档

图 13-19 显示的页面是服务的信息页面。在该图中，注意还有一个对服务的 WSDL 文件的链接。与 ASP.NET Web 服务一样，WCF 服务也可以自动生成 WSDL 文件。WCF 4.5 的特性之一就是提供了一个用于单个 WSDL 文件的选项。单击 WSDL 链接，会在浏览器中显示 WSDL，如图 13-20 所示。



图 13-20

有了这个 WSDL 文件，就可以通过 HTTP 绑定使用它定义的服务。注意文档底部的一些元素，如程序清单 13-25 所示(这个程序清单取自在浏览器中生成的 WSDL 文件)。

程序清单 13-25 WSDL 文件中显示服务端点的部分

```
<wsdl:service name="Calculator">
  <wsdl:port name="BasicHttpBinding_ICalculator"
    binding="tns:BasicHttpBinding_ICalculator">
    <soap:address location="http://localhost:2109/Calculator.svc"/>
  </wsdl:port>
</wsdl:service>
```

XML 文档中的这个元素表示，为了使用服务，终端用户需要在 HTTP 上使用 SOAP。通过在文档中使用 `<soap:address>` 元素指定这一点。使用这个简单的 WSDL 文档，就可以建立使用者来使用这个接口。

13.8.3 建立 WCF 使用者

有了使用 WCF 架构建立的 HTTP 服务后，就可以在 ASP.NET 中建立使用者应用程序，该应用程序可使用简单的 Calculator 服务。该使用者使用 SOAP 通过 HTTP 发送其请求。本节描述如何使用该服务。首先打开 Visual Studio 2012，创建一个新的 ASP.NET 应用程序。尽管这个示例使用了 ASP.NET 应用程序，但也可以使这个使用者通过 .NET 中的其他应用程序类型来调用。

把新的 ASP.NET 应用程序命名为 WCFConsumer。这个应用程序使用 Calculator 服务，因此应布置两个文本框和一个按钮来启动服务调用。对于这个示例，仅使用服务的 Add 方法。



为了保持本章的一致性，将这些项目的 C# 版本命名为 Chapter13-CustomerConsumer-CS。

13.8.4 添加服务引用

布置好 ASP.NET 页面后，就需要引用新的 WCF 服务，其引用方式类似于建立 XML Web 服务引用。在 Visual Studio 的 Solution Explorer 中右击解决方案名，从弹出的菜单中选择 Add Service Reference 命令。添加服务引用的功能在 Visual Studio 的最近 3 个版本中引入，而在以前的版本中只有 Add Reference 和 Add Web Reference 选项。

选择了 Add Service Reference 命令后，就会显示如图 13-21 所示的 Add Service Reference 对话框。



图 13-21

Add Service Reference 对话框要求输入两项内容：服务的 URI 或地址(基本上是指向 WSDL 文件的指针)以及要给引用指定的名称。给引用指定的名称用于可以和服务交互的实例化对象。

在图 13-21 中，为 Service Address 设置提供的名称用于本章前面运行的服务。在 Add Service Reference 对话框中单击 OK 按钮。这会在项目中添加 Service Reference 文件夹，其中包含一些代理文件，如图 13-22 所示。

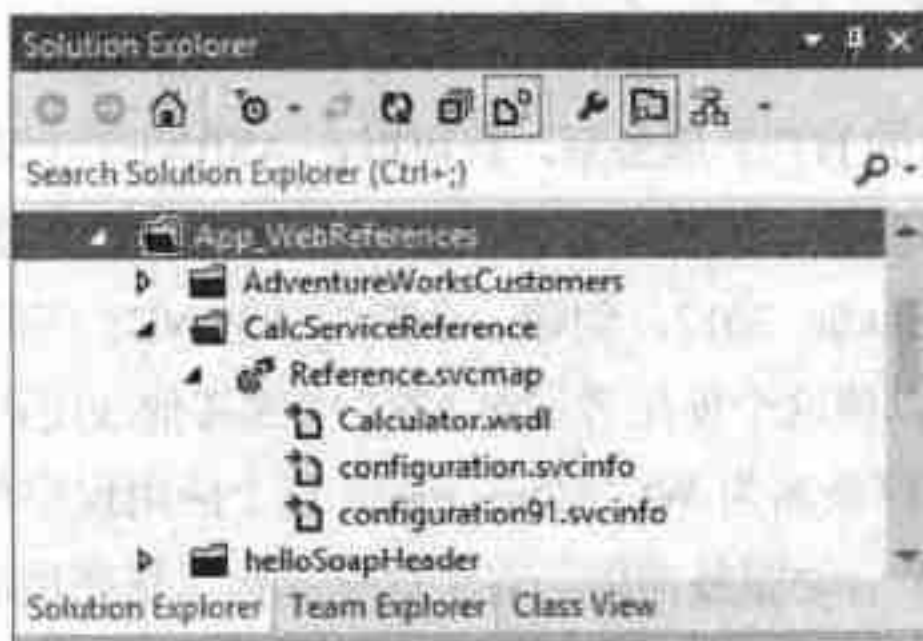


图 13-22

事实上,添加 Service Reference 文件夹后,在这个文件夹中就包含了一系列文件。另一个需要注意的是在 Service Reference 文件夹中添加的 System.ServiceModel 引用,在通过 Add Service Reference 对话框引用该服务之前并没有这个引用。

1. 配置文件的修改

查看 web.config 文件,你会发现 Visual Studio 在文档中放置了有关服务的信息,如程序清单 13-26 所示。

程序清单 13-26 Visual Studio 在 web.config 文件中添加的内容

```
<system.serviceModel>
  <bindings>
    <basicHttpBinding>
      <binding name="BasicHttpBinding_ICalculator" />
    </basicHttpBinding>
  </bindings>
  <client>
    <endpoint address="http://localhost:2109/Calculator.svc"
      binding="basicHttpBinding"
      bindingConfiguration="BasicHttpBinding_ICalculator"
      contract="CalcServiceReference.ICalculator"
      name="BasicHttpBinding_ICalculator" />
  </client>
</system.serviceModel>
```

这个配置文档的重要部分是<client>元素。该元素包含子元素<endpoint>,这个子元素定义了服务使用过程的位置和方式。

<endpoint>元素提供了服务的地址 http://localhost:2109/Calculator.svc,此外还指定了应使用可用 WCF 绑定中的哪个绑定。在这个例子中,需要使用 basicHttpBinding 绑定。即使使用 WCF 架构中已建立的绑定,在客户端中也可以定制该绑定的行为。使用<endpoint>元素的 bindingConfiguration 特性指定定义绑定行为的设置。在这个例子中,提供给 bindingConfiguration 特性的值是 BasicHttpBinding_ICalculator,它是<basicHttpBinding>元素中包含的<binding>元素的引用。

如前所述,Visual Studio 2012 使这些服务的使用相当简单。实际上,如果以前使用过 WCF 服务,就会发现配置比以前的版本容易多了。

下一步是为第一步中创建的 GUI 编写服务接口的使用代码。

2. 编写使用代码

使用接口的代码非常少。终端用户在所提供的两个文本框中各输入一个数字,然后单击按钮以调用服务,对所提供的数字执行指定的操作。程序清单 13-27 是按钮单击事件的代码。

程序清单 13-27 调用服务的按钮单击事件

```
protected void Button1_Click(object sender, EventArgs e)
{
    CalcServiceReference.CalculatorClient ws = new
    CalcServiceReference.CalculatorClient();
```



```

        int result = ws.Add(int.Parse(TextBox1.Text), int.Parse(TextBox2.Text));
        Label1.Text = result.ToString();
        ws.Close();
    }

```

这段代码非常类似于在 XML Web 服务环境下使用 Web 引用。首先是代理类的实例化，即创建 ws 对象，如下所示：

```

CalcServiceReference.CalculatorClient ws = new
    CalcServiceReference.CalculatorClient();

```

现在使用 ws 对象，IntelliSense 选项会提供相应的 Add、Subtract、Multiply 和 Divide 方法。与前面一样，请求和响应都通过 HTTP 发送为 SOAP。

13.8.5 使用数据协定

前面建立 WCF 服务时，已定义的数据协定依赖于简单类型或基本数据类型。在早期的 WCF 服务中，包含了 .NET 类型 Integer，它被映射为 XSD 类型 int。通过 WCF 生成的 WSDL 文档不可能看到已定义的输入和输出类型，但这些类型是存在的。使用单个 WSDL 文件时，这些类型是在单个 WSDL 文档中提供的。

如果不使用单个 WSDL 文件，XML 模式类型就通过导入的 .xsd 文档(Calculator.xsd 和 Calculator1.xsd)显示。这部分 WSDL 文档如程序清单 13-28 所示。

程序清单 13-28 WSDL 文档中的导入类型

```

<wsdl:types>
  <xsd:schema targetNamespace="http://tempuri.org/Imports">
    <xsd:import namespace="http://tempuri.org/"
      schemaLocation="http://localhost:2110/Calculator.svc?xsd=xsd0"/>
    <xsd:import
      namespace="http://schemas.microsoft.com/2003/10/Serialization/"
      schemaLocation="http://localhost:2110/Calculator.svc?xsd=xsd1"/>
  </xsd:schema>
</wsdl:types>

```

输入 XSD 位置 <http://localhost:2110/Calculator.svc?xsd=xsd0>，可以给出服务的输入和输出参数。例如，查看 Add 方法的定义，会看到如程序清单 13-29 所示的代码。

程序清单 13-29 在 XSD 中定义需要的类型

```

<xs:element name="Add">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="a" type="xs:int" />
      <xs:element minOccurs="0" name="b" type="xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="AddResponse">
  <xs:complexType>
    <xs:sequence>

```



```

        <xs:element minOccurs="0" name="AddResult" type="xs:int" />
    </xs:sequence>
</xs:complexType>
</xs:element>

```

这段 XML 代码表示, 其中的两个必选输入参数(a 和 b)的类型是 int。因此, 使用者得到了名为 <AddResult> 的元素, 其中包含 int 类型的值。

这个 WCF 服务的开发人员不必建立数据协定, 因为这个服务使用的是简单类型。当使用复杂类型时, 需要创建数据协定和服务协定。

1. 建立带数据协定的服务

在使用数据协定的例子中, 创建新的 WCF 服务 MyCustomDataContractService。这里仍然需要一个定义服务协定的接口和另一个实现该接口的类。除此之外, 还需要另一个定义数据协定的类。

与使用 ServiceContract 和 OperationContract 特性的服务协定一样, 数据协定使用 DataContract 和 DataMember 特性。要访问这些特性, 需要在项目中引用 System.Runtime.Serialization 名称空间, 并把这个名称空间导入文件。

这个例子中的定制类型是 Customer 类型, 如程序清单 13-30 所示(本章下载代码中的 WCF 项目的 IMyCustomDataContractService.cs 文件)。

程序清单 13-30 建立 Customer 类型

```

using System.Runtime.Serialization;
using System.ServiceModel;

[DataContract]
public class Customer
{
    [DataMember]
    public string Firstname;

    [DataMember]
    public string Lastname;
}

[ServiceContract]
public interface IMyCustomDataContractService
{
    [OperationContract]
    string HelloFirstName(Customer cust);

    [OperationContract]
    string HelloFullName(Customer cust);
}

```

这个例子也导入了 System.Runtime.Serialization 名称空间, 并且文件中的第一个类是服务的数据协定。Customer 类有两个成员 FirstName 和 LastName, 这两个属性都是 String 类型。使用 DataContract 特性把类指定为数据协定:

```
[DataContract]
public class Customer
{
    // Code removed for clarity
}
```

现在, 通过使用 `DataMember` 特性, 类中的所有属性都是数据协定的一部分:

```
[DataContract]
public class Customer
{
    [DataMember]
    public string Firstname;

    [DataMember]
    public string Lastname;
}
```

最后, 在接口以及实现了 `IHelloCustomer` 接口的类中使用 `Customer` 对象, 如程序清单 13-31 所示。

程序清单 13-31 实现接口

```
public class MyCustomDataContractService : IMyCustomDataContractService
{
    public string HelloFirstName(Customer cust)
    {
        return "Hello " + cust.Firstname;
    }

    public string HelloFullName(Customer cust)
    {
        return "Hello " + cust.Firstname + " " + cust.Lastname;
    }
}
```

2. 建立使用者

服务准备好后, 下一步就是建立使用者。首先更新用于使用 Web 服务的那个 Web 应用程序。再次右击解决方案, 从弹出的菜单中选择 `Add Service Reference` 命令。

在 `Add Service Reference` 对话框中, 为服务添加 WSDL 文件的位置, 单击 `OK` 按钮。这会像以前一样把修改添加到引用和 `web.config` 文件中, 从而允许使用服务。程序清单 13-32 中的代码说明了如果使用 `Button` 控件启动调用, 那么使用服务需要什么条件。

程序清单 13-32 通过 WCF 服务使用定制类型

```
protected void Button1_Click(object sender, EventArgs e)
{
    MyCustomDataContractServiceReference.MyCustomDataContractServiceClient ws = new
        MyCustomDataContractServiceReference.MyCustomDataContractServiceClient();
    MyCustomDataContractServiceReference.Customer myCustomer = new
        MyCustomDataContractServiceReference.Customer();
}
```

```

    myCustomer.Firstname = "Jason";
    myCustomer.Lastname = "Gaylord";
    Label1.Text = ws.HelloFullName(myCustomer);
    ws.Close();
}

```

在建立引用后，使用者应注意服务引用提供了 `MyCustomDataContractServiceClient` 对象和 `Customer` 对象，通过服务的数据协定定义这些对象。

因此，前面的代码块实例化了这两个对象，并建立了 `Customer` 对象，之后把它传送给服务提供的 `HelloFullName` 方法。

3. 查看 `HelloCustomerService` 的 WSDL 和模式

建立对 `HelloCustomer` 服务的引用时，在 WSDL 中会导入如下 XSD：

```

<wsdl:types>
  <xsd:schema targetNamespace="http://tempuri.org/Imports">
    <xsd:import namespace="http://tempuri.org/"
      schemaLocation="http://localhost:2109/MyCustomDataContractService.svc?xsd=xsd0"/>
    <xsd:import namespace="http://schemas.microsoft.com/2003/10/Serialization/"
      schemaLocation="http://localhost:2109/MyCustomDataContractService.svc?xsd=xsd1"/>
    <xsd:import namespace="http://schemas.datacontract.org/2004/07/"
      schemaLocation="http://localhost:2109/MyCustomDataContractService.svc?xsd=xsd2"/>
  </xsd:schema>
</wsdl:types>

```

`http://localhost:2019/MyCustomDataContractService.svc?xsd=xsd2` 提供了 `Customer` 对象的详细信息。这个文件的代码如下：

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:tns="http://schemas.datacontract.org/2004/07/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://schemas.datacontract.org/2004/07/"
  elementFormDefault="qualified">
  <xs:complexType name="Customer">
    <xs:sequence>
      <xs:element name="Firstname" type="xs:string" nillable="true"
        minOccurs="0"/>
      <xs:element name="Lastname" type="xs:string" nillable="true"
        minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="Customer" type="tns:Customer" nillable="true"/>
</xs:schema>

```

这段代码是 `Customer` 对象的 XSD 描述。对 WSDL 的引用包括 `Customer` 对象的 XSD 描述，并可以创建这个对象的本地实例。

使用这个模型，很容易利用自己定义的类型建立服务。

13.8.6 定义名称空间

注意本章建立的服务没有定义名称空间。如果查看生成的 WSDL 文件,就会发现所提供的名称空间是 `http://tempuri.org`。显然,我们不希望使用这个默认的名称空间,因此需要定义自己的名称空间。为此,可以使用接口的 `ServiceContract` 特性设置名称空间,如下所示:

```
[ServiceContract(Namespace="http://jasongaylord.com/ns/")]
public interface IMyCustomDataContractService
{
    [OperationContract]
    string HelloFirstName(Customer cust);

    [OperationContract]
    string HelloFullName(Customer cust);
}
```

其中, `ServiceContract` 特性使用 `Namespace` 属性来提供名称空间。

13.8.7 使用 WCF 数据服务

EDM 是数据的抽象概念模型,因为需要在代码中表示它。近年来,人们强调的是 EDM 及其优点,例如应用程序的可重用性。



第 11 章介绍了 EDM 的许多信息以及如何在 EF 中使用它。

另一个可用的模型是 WCF 数据服务架构。使用这个功能很容易为客户端应用程序创建一个接口,它可以实现所有的功能,从简单的读取功能到完整的 CRUD 模型(包括创建、读取、更新和删除功能)。WCF 数据服务架构之前称为 ADO.NET 数据服务。



WCF 数据服务一般称为使用 OData 和 WCF 提供数据。尽管这不是提供数据的唯一方式,但肯定是最简单的方式。本章后面将介绍如何在 URL 中使用这个语法构建过滤器。

WCF 数据服务可以为后台的数据源创建服务层。如果自己创建服务层,尤其是当使用完整的 CRUD 模型时,就意味着要做大量的工作。WCF 数据服务允许获得 URI 驱动的服务层。图 13-23 显示了使用 WCF 数据服务时服务层的总体体系结构。

从图 13-23 中可以看出,WCF 数据服务层并不能与数据库直接交互。但是,可以使用 EDM 层,它是数据存储和云接口之间的映射层。当使用 EDM 时,可以使用 LINQ。

WCF 数据服务允许以基于 RESTful 的服务的方式,快速实现与应用程序的底层数据源的交互。WCF 数据服务的当前版本允许使用 JSON 或基于 Atom 的 XML 来处理数据存储。

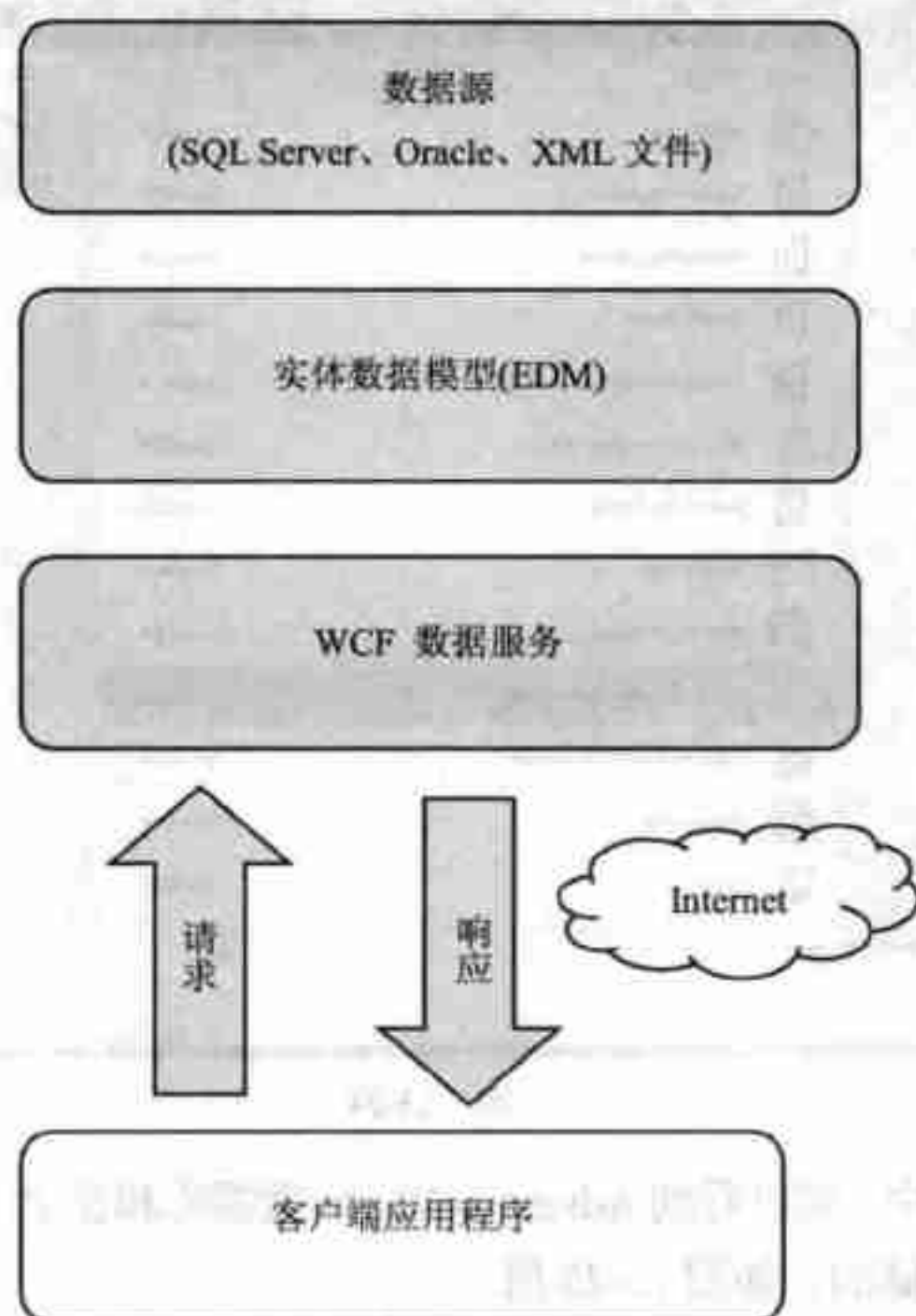


图 13-23

13.8.8 创建第一个服务

过去，要了解如何为数据库创建完整的服务层以实现创建、读取、更新和删除功能，需要花费大量的时间和精力。但是，WCF 数据服务使这项任务更加易于完成，可以从第一个例子中看出这一点。

要创建服务层，可首先使用 C# 创建一个标准的 ASP.NET Web 应用程序。这样就创建了一个标准的 Web 应用程序。由于 WCF 数据服务通过底层数据库工作，因此需要添加一个数据库。在本例中添加的是前面用过的 AdventureWorks 数据库。将该数据库放到项目的 App_Data 文件夹中。

1. 添加实体数据模型

创建数据库后，下一步是创建 WCF 数据服务将要用到的实体数据模型。为此，右击项目并从弹出的菜单中选择 Add | New Item 命令。

这会打开 Add New Item 对话框。如图 13-24 所示，在项目中添加 ADO.NET 实体数据模型。

如图 13-24 所示，将 ADO.NET Entity Data Model 文件命名为 AdventureWorks.edmx。当单击 Add 按钮创建 AdventureWorks.edmx 文件时，会显示 Entity Data Model 向导，该向导提供了创建空白 EDM 或从已有数据库创建 EDM 的选项。在本例中，选择从已有数据库(AdventureWorks)中创建 EDM 的选项。之后，在向导中单击 Next 按钮。



图 13-24

在向导的下一个界面中，可以看到 AdventureWorks 数据库和预定义的连接设置，也可以看到它们是如何在应用程序中存储的，如图 13-25 所示。

在如图 13-25 所示的屏幕截图中，注意连接字符串和映射信息的位置将存储到 web.config 文件中。在这个界面上还可以看到，在向导底部的文本框中命名模型的实例 AdventureWorksEntities。这个名称很重要，因为接下来还要用到它。

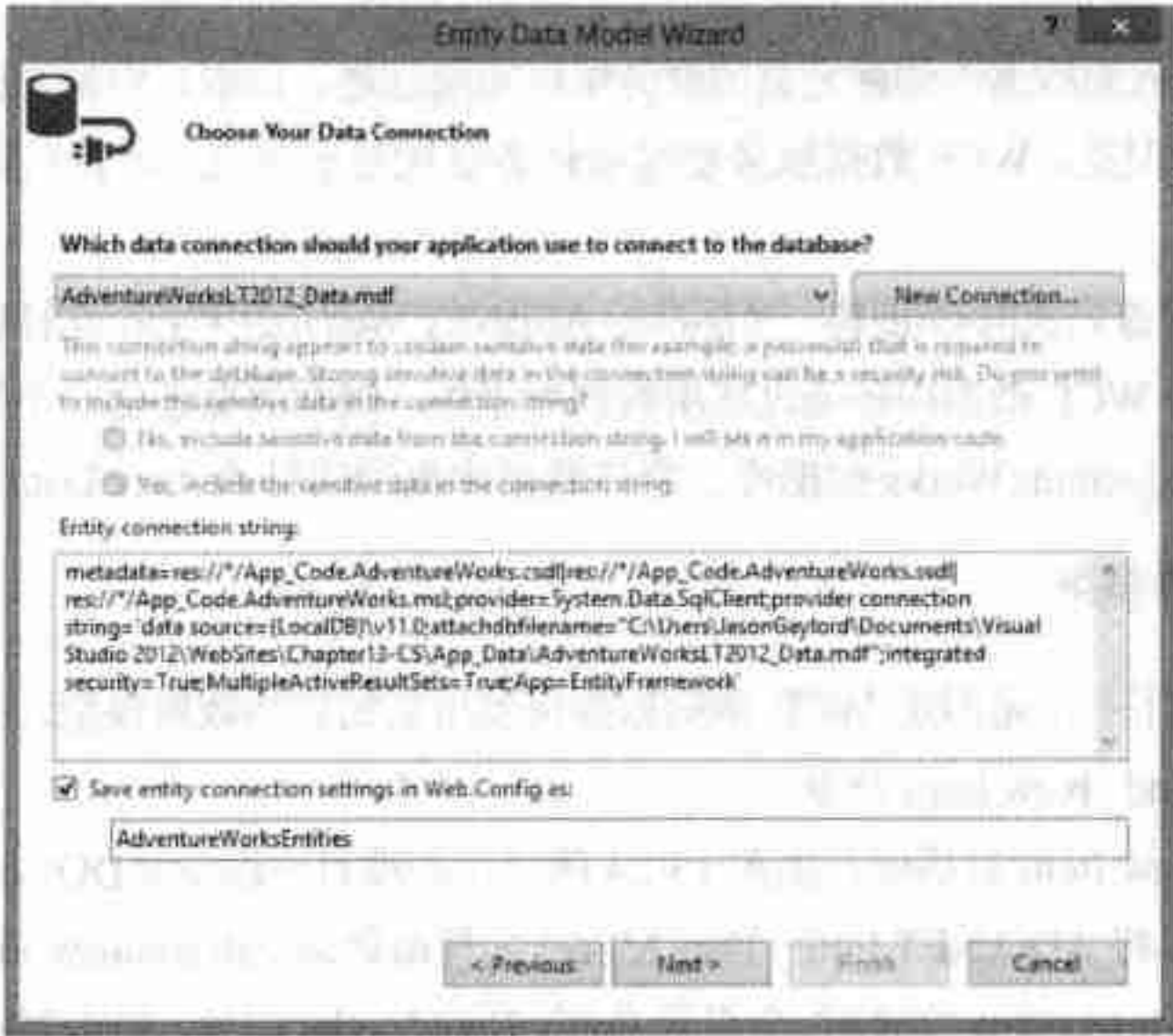


图 13-25

下一个界面要求选择该模型可能用到的表、视图或存储过程。在这个例子中，在树状视图中选中 Tables 选项旁边的复选框，以选中数据库中的所有表，如图 13-26 所示。

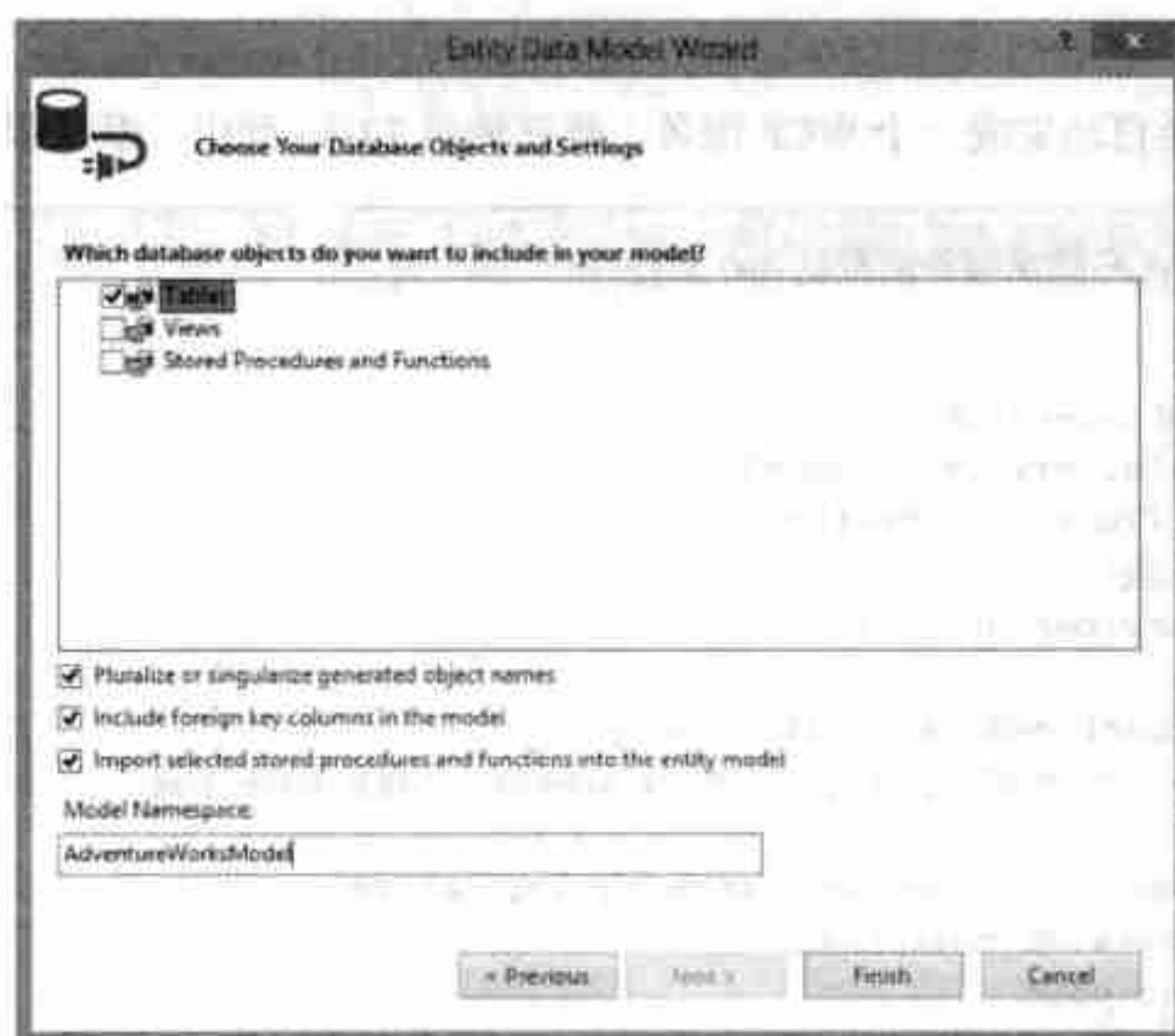


图 13-26

选中 Tables 复选框后，单击 Finish 按钮以使 Visual Studio 创建实体数据模型。Visual Studio 在 O/R 设计器中创建了模型的可视化表示。

如果查看解决方案中的 AdventureWorks.desinger.cs 文件，那么可以看到创建 EDM 时生成的所有代码。将这个类文件命名为 AdventureWorksEntities。

2. 创建服务

现在已经有了数据库和 EDM，接下来添加 WCF 数据服务。为此，在 Visual Studio 的 Solution Explorer 中右击项目，从弹出的菜单中选择 Add | New Item 命令。再次打开 Add New Item 对话框，在该对话框的中间部分选择 WCF Data Service 选项，如图 13-27 所示。



图 13-27

如图 13-27 所示，将 WCF 数据服务命名为 AdventureWorksDataService.svc。之后，单击 Add 按钮，Visual Studio 将会自动生成一个 WCF 服务。程序清单 13-33 列出了默认服务文件的代码。

程序清单 13-33 WCF 数据服务的默认.svc 文件

```
using System;
using System.Data.Services;
using System.Data.Services.Common;
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel.Web;

public class AdventureWorksDataService :
    DataService< /* TODO: put your data source class name here */ >
{
    // This method is called only once to initialize
    // service-wide policies.
    public static void
        InitializeService(DataServiceConfiguration config)
    {
        // TODO: set rules to indicate which entity sets and
        // service operations are visible, updatable, etc.
        // Examples:
        // config.SetEntitySetAccessRule("MyEntityset",
        //     EntitySetRights.AllRead);
        // config.SetServiceOperationAccessRule
        //     ("MyServiceOperation", ServiceOperationRights.All);
        config.DataServiceBehavior.MaxProtocolVersion =
            DataServiceProtocolVersion.V3;
    }
}
```

这里生成的代码是要通过 WCF 数据服务提供的内容的基本架构。如果没有完成这段代码中指定的 TODO 部分，那么它将不能运行。第一步是使用程序清单 13-34 所示的代码放入 EDM 实例的名称。

程序清单 13-34 修改 WCF 数据服务以处理 EDM

```
public class AdventureWorksDataService :
    DataService<AdventureWorksEntities>
{
    // Code removed for clarity
}
```

现在应用程序中就有了数据库、EDM 以及要处理 EDM 的服务。编译并在浏览器中打开 AdventureWorksDataService.svc 文件，可以看到如下一些 XML 代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<service xmlns:atom="http://www.w3.org/2005/Atom"
    xmlns="http://www.w3.org/2007/app"
    xml:base="http://localhost:5526/AdventureWorksDataService.svc/">
  <workspace>
    <atom:title>Default</atom:title>
```

```
</workspace>
</service>
```



如果没有看到这些 XML 代码,就需要关闭 IE 浏览器的源读取功能,方法是在浏览器中选择 Tools | Internet Options 命令。在打开的对话框中,选择 Content 选项卡,在 Feeds 部分单击 Select 按钮。在打开的对话框中,取消选中“Turn on feed reading”复选框。

前面的 XML 代码的执行结果应是模型中所有可用数据集的列表,但是在默认情况下, WCF 数据服务锁定了所有的数据集。为了在模型中解除对这些数据集的锁定,应返回到 InitializeService 函数,并添加程序清单 13-35 所示的黑体代码。

程序清单 13-35 打开服务以从可用的表中读取

```
using System;
using System.Data.Services;
using System.Data.Services.Common;
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel.Web;

public class AdventureWorksDataService :
    DataService<AdventureWorksEntities>
{
    public static void
        InitializeService(DataServiceConfiguration config)
    {
        config.SetEntitySetAccessRule("*",
            EntitySetRights.AllRead);
    }
}
```

在这个例子中,打开所有的表以进行访问。访问这些表的用户只能在这个表中读取数据,而不能进行添加或删除操作。使用星号(*)指定所有的表,并把 EntitySetRights 枚举设置为 AllRead,将底层数据的操作权限设置为只读。

在浏览器中编译并运行这个服务,可以看到如下 XML 代码:

```
<?xml version="1.0" encoding="UTF-8"?>
<service xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns="http://www.w3.org/2007/app"
  xml:base="http://localhost:5526/AdventureWorksDataService.svc/">
  <workspace>
    <atom:title>Default</atom:title>
    <collection href="BuildVersions">
      <atom:title>BuildVersions</atom:title>
    </collection>
    <collection href="ErrorLogs">
      <atom:title>ErrorLogs</atom:title>
    </collection>
```



```

    <collection href="Addresses">
      <atom:title>Addresses</atom:title>
    </collection>
    <collection href="Customers">
      <atom:title>Customers</atom:title>
    </collection>
    <collection href="CustomerAddresses">
      <atom:title>CustomerAddresses</atom:title>
    </collection>
    <collection href="Products">
      <atom:title>Products</atom:title>
    </collection>
    <collection href="ProductCategories">
      <atom:title>ProductCategories</atom:title>
    </collection>
    <collection href="ProductDescriptions">
      <atom:title>ProductDescriptions</atom:title>
    </collection>
    <collection href="ProductModels">
      <atom:title>ProductModels</atom:title>
    </collection>
    <collection href="ProductModelProductDescriptions">
      <atom:title>ProductModelProductDescriptions</atom:title>
    </collection>
    <collection href="SalesOrderDetails">
      <atom:title>SalesOrderDetails</atom:title>
    </collection>
    <collection href="SalesOrderHeaders">
      <atom:title>SalesOrderHeaders</atom:title>
    </collection>
  </workspace>
</service>

```

这段 XML 代码的输出采用 AtomPub 格式，该格式是 XML 在 WCF 数据服务中两种可用格式中的一种。另一种格式是 JSON，主要用于 JavaScript。有了如下报头，就可以检索 AtomPub 例子：

```

GET http://localhost:5526/AdventureWorksDataService.svc/ HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Language: en-US
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0)
Accept-Encoding: gzip, deflate
Host: localhost:5526
DNT: 1
Connection: Keep-Alive
Pragma: no-cache

```

把 Accept 报头修改为 application/json，例如输入 jQuery 调用服务，就会得到如下响应：

```

{ "d" : {
  "EntitySets": [
    "BuildVersions", "ErrorLogs", "Addresses", "Customers",
    "CustomerAddresses", "Products", "ProductCategories",
    "ProductDescriptions", "ProductModels",
    "ProductModelProductDescriptions", "SalesOrderDetails",

```

```

    "SalesOrderHeaders"
  }
}

```

13.8.9 查询接口

要查询接口，可以使用如下 3 个组件：URI、HTTP 报头的操作和要在查询中使用的 HTTP 动词。查询接口最常用的方式是在数据存储中执行读取操作。

回顾本章前面介绍的 HTTP 报头的例子，可以看到如程序清单 13-36 所示的代码。

程序清单 13-36 请求 HTTP 报头的示例

```

GET http://localhost:5526/AdventureWorksDataService.svc/Customers HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Language: en-US
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0)
Accept-Encoding: gzip, deflate
Host: localhost:5526
DNT: 1
Connection: Keep-Alive
Pragma: no-cache

```

返回的结果基于 Accept HTTP 报头返回的内容。调用什么方法取决于使用的 URI。在程序清单 13-36 所示的例子中，使用的 URI 是 /AdventureWorksDataService.svc/Customers，这意味着调用了在 EDM 中设置的 Customers 集合，并返回从 AdventureWorks 数据库中获得的 Product 表的内容。

程序清单 13-36 也是一条读语句，因为使用的 HTTP 动词是 GET。表 13-1 详细介绍了 HTTP 的动词列表，以及如何把它们映射为数据访问类型。

表 13-1

HTTP 动 词	数据访问类型
POST	创建
GET	读取
PUT	更新
DELETE	删除

下面介绍如何使用不同的方法查询由 WCF 数据服务提供的底层接口，以便从数据库中读取内容。

1. 读取表中数据

读取整个表的内容依赖于传送的 URI 参数。指定特定的实体集可以读取内容。例如，将下面的查询语句输入到浏览器的地址栏中：

```
http://localhost:5526/AdventureWorksDataService.svc/Customers
```

在这个例子中，只提供 URI 中的实体集 Customers，来请求整个 Customers 表的内容。图 13-28 显示了这个请求的结果。



图 13-28

下面的语法是这类请求的另一个例子：

```
http://localhost:5526/AdventureWorksDataService.svc/Products
```

在这个例子中，可以从数据库的 Products 表中得到产品的完整列表。如果查看 URI 调用中的可用表级信息，将会看到如下信息(作为例子)：

```
<id>http://localhost:4113/AdventureWorksDataService.svc/Products</id>
<title type="text">Products</title>
<updated>2012-11-28T04:43:40Z</updated>
<link rel="self" title="Products" href="Products" />
```

这里接收标题(实体名称)和作为<id>元素的完整 URI。此外，在<updated>元素中得到的时间戳显示了运行查询的时间。最后，如果编程构造 URI，那么还可以得到引用项自身的链接(除了这个链接外，在实体的关系中还存在其他链接)。

2. 从表中读取指定的项

前面的例子介绍了如何使用 URI(如 http://localhost/NorthwindDataService.svc/Products)，从表或实体集中提取整个集合。

如果查看包含在结果集合中的产品项，就会看到程序清单 13-37 所示的代码。

程序清单 13-37 从 Products 查询中查看其中一项

```
<entry xml:base="http://localhost:5526/AdventureWorksDataService.svc/"
  xmlns="http://www.w3.org/2005/Atom"
  xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
  <id>http://localhost:5526/AdventureWorksDataService.svc/Products(710)</id>
  <category term="AdventureWorksModel.Product"
    scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
```



```

<link rel="edit" title="Product" href="Products(710)" />
<link
  rel="http://schemas.microsoft.com/ado/2007/08/dataservices/
  related/ProductCategory" type="application/atom+xml;type=entry"
  title="ProductCategory" href="Products(710)/ProductCategory" />
<link
  rel="http://schemas.microsoft.com/ado/2007/08/dataservices/
  related/ProductModel" type="application/atom+xml;type=entry"
  title="ProductModel" href="Products(710)/ProductModel" />
<link
  rel="http://schemas.microsoft.com/ado/2007/08/dataservices/
  related/SalesOrderDetails" type="application/atom+xml;type=feed"
  title="SalesOrderDetails" href="Products(710)/SalesOrderDetails" />
<title />
<updated>2012-11-28T04:49:26Z</updated>
<author>
  <name />
</author>
<content type="application/xml">
  <m:properties>
    <d:ProductID m:type="Edm.Int32">710</d:ProductID>
    <d:Name>Mountain Bike Socks, L</d:Name>
    <d:ProductNumber>SO-B909-L</d:ProductNumber>
    <d:Color>White</d:Color>
    <d:StandardCost m:type="Edm.Decimal">3.3963</d:StandardCost>
    <d>ListPrice m:type="Edm.Decimal">9.5000</d>ListPrice>
    <d:Size>L</d:Size>
    <d:Weight m:type="Edm.Decimal" m:null="true" />
    <d:ProductCategoryID m:type="Edm.Int32">27</d:ProductCategoryID>
    <d:ProductModelID m:type="Edm.Int32">18</d:ProductModelID>
    <d:SellStartDate m:type="Edm.DateTime">
      2001-07-01T00:00:00
    </d:SellStartDate>
    <d:SellEndDate m:type="Edm.DateTime">
      2002-06-30T00:00:00
    </d:SellEndDate>
    <d:DiscontinuedDate m:type="Edm.DateTime" m:null="true" />
    <d:ThumbNailPhoto m:type="Edm.Binary">
      R01GODlhUAAxAPcAAAAAIAAAACAAICAAAAgIAAgACAgICAgMDAwP8AAAD/AP//AAAA//8A/wD////////wAAA
      AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
      AAAAAA AAAAAmWAAZgAAmQAAZAAA/
      wAzaAAzMwAZZgAZmQAzzAAz/wBmAABmMwBmZgBmmQBmzABm/wCZAACZMwCZgCZmQCZzACZ/wDMAADMMwDMZgDMmQ
      DMzADM/wD/AAD/MwD/ZgD/mQD/zAD//zMAADMAMzMAZjMAmTMAzDMA/zMzADMzMzMzZjMzmTMzzDMz/zNmADNmMzN
      mZjNmNmTnmzDNm/zOZADOZmzOZZjOZmTOZzDOZ/zPMADPMmzPMZjPMmTPmzDPM/zP/ADP/MzP/ZjP/mTP/zDP//2YA
      AGYAM2YAZmYAmWYAZGYA/2YzAGYzM2YzZmYzmWYzzGYz/2ZmAGZmM2ZmZmZmmWZmzGZm/2aZAGaZM2aZZmaZmWa
      ZzGaZ/2bMAGbMM2bMZmbMmWbMzGbM/2b/AGb/M2b/Zmb/mWb/zGb//5kAAJkAM5kAZpkAmZkAzJkA/5kzAJkzM5kz
      ZpkzmZkzzJkz/5lmAJlmM5lmZ
      plnmZlmzJlm/5mZAJmZM5mZzpmZmZmZzJmZ/5nMAJnMM5nMZpnMmZnMzJnM/5n/AJn/M5n/Zpn/mZn/zJn//8wAAM
      wAM8wAZswAmcwAzMwA/
      8wzAMwzM8wzZswzmcwzzMwz/8xmAMxmM8xmZsxmncxmzMxm/8yZAMyZM8yZsyZmcyZzMyZ/8zMAMzMM8zMZszMmc
      zMzMzM/8z/AMz/M8z/Zsz/mcz/zMz///8AAP8AM/8AZv8Amf8AzP8A//8zAP8zM/8zZv8zmf8zzP8z//9mAP9mM/9
      mZv9mmf9mzP9m//+ZAP+ZM/+ZZv+Zmf+ZzP+Z///MAP/MM//MZv/Mmf/
      MzP/M////AP//M///Zv//mf//zP///yH5BAEAAABAALAAAAABQADEAAAj/AP8JHEiwoMGDCBMqXMiwoCOHECNKnE1x
      osWLGDNq3Mix08ePIEOKHEmypoM
      mTKFOqXJkRBYqBLhfGZPnQ5ct/MxPmpMnQpsCZNm/CfBnTZ86gQ3HeRMoRadGlQpUqJfoUZ9KnVH9GxVhUKtC
    </d:ThumbNailPhoto>
  </m:properties>
</content>

```

```

oVaWKnZrVK9SmVMPuVHvWrfisPjd+LbuW7tmvb8t6nJ
uXIFutfbH2lSt07ta/eeOy3clTYuGtjS8yjUy5suXlmDHHdRjWIGPGIjdDBA3YL2SQVY+mvQsVLl6yqLOqfuyWt1H
ZbTv+nYl76G67H38DTs068GrS
koMSN+62+fKQqrW2Xe6aem7CSaf6fq7ceevTmcOLEh9Pvrz58+jTq1/Pvr379+8DAqA7
    </d:ThumbNailPhoto>
    <d:ThumbNailPhotoFileName>
        no_image_available_small.gif
    </d:ThumbNailPhotoFileName>
    <d:rowguid m:type="Edm.Guid">
        161c035e-21b3-4e14-8e44-af508f35d80a
    </d:rowguid>
    <d:ModifiedDate m:type="Edm.DateTime">
        2004-03-11T10:01:36.827
    </d:ModifiedDate>
  </m:properties>
</content>
</entry>

```

查看这个产品的<id>值，会看到如下信息：

```
http://localhost:5526/AdventureWorksDataService.svc/Products(710)
```

对这个特定产品的引用是 Products(710)，也就是说在 Product 集合中，我们感兴趣的是 ProductID 为 710 的项。把这个 URI 输入到浏览器的地址栏中，就只提供这个特定产品的信息。

如果查看 XML，就会发现<content>元素中包含想要查看的特定产品的所有数据。在这里，将这些数据构造为属性集合。

我们不仅可以看到这个顾客的属性列表，还可以通过使用如下 URI 声明获得单个属性：

```
http://localhost:5526/AdventureWorksDataService.svc/Products(710)/Name
```

运行上面的构造代码将只返回这个产品的 Name 属性。结果如下：

```

<?xml version="1.0" encoding="UTF-8"?>
<d:Name
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
  xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices">
  Mountain Bike Socks, L
</d:Name>

```

注意，Products(710)中的数字引用并不是索引引用，而是 ID 引用。

3. 使用关系

使用 WCF 数据服务可以方便地通过 RESTful 接口获得和处理数据库。使用 ADO.NET Entity Framework 和 WCF 数据服务的最大优点之一是它们使处理对象关系更加容易。

回顾本章前面设计的实体数据模型，你会发现这些对象有着内置的关系，该关系在 O/R 设计器中是可见的，如图 13-29 所示。

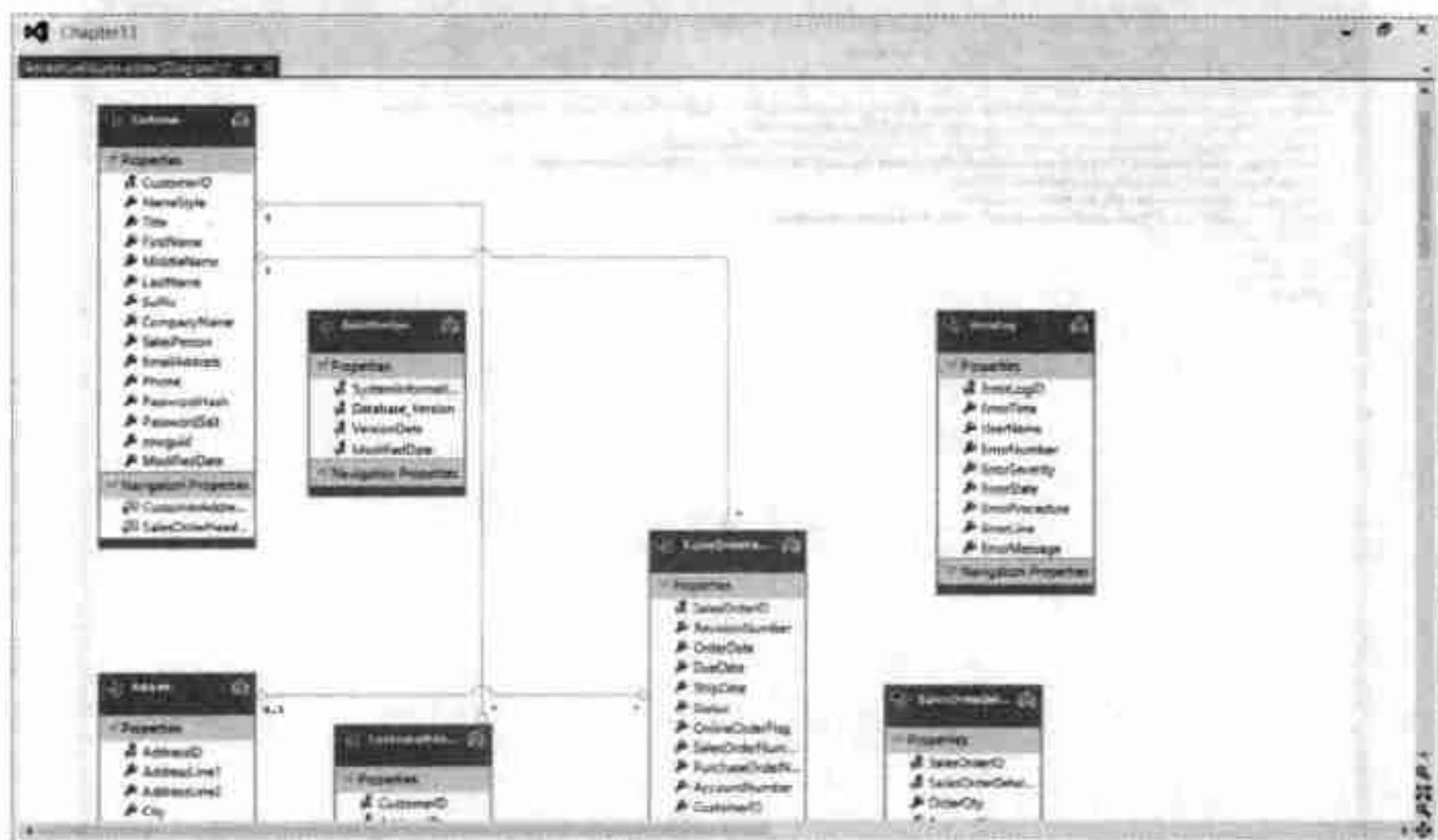


图 13-29

从图 13-29 中可以看出，Customers 对象和 SalesOrderObjects 对象之间存在关系。也可以使用 WCF 数据服务进一步操作这种对象链，因为是以 EDM 表示该对象链。

要理解如何表示这些关系，请导航到：

[http://localhost:5526/AdventureWorksDataService.svc/Customers\(1\)](http://localhost:5526/AdventureWorksDataService.svc/Customers(1))

关系如程序清单 13-38 所示。

程序清单 13-38 查看顾客关系

```
<link
  rel="http://schemas.microsoft.com/ado/2007/08/dataservices/
  related/CustomerAddresses" type="application/atom+xml;type=feed"
  title="CustomerAddresses" href="Customers(1)/CustomerAddresses" />
<link
  rel="http://schemas.microsoft.com/ado/2007/08/dataservices/
  related/SalesOrderHeaders" type="application/atom+xml;type=feed"
  title="SalesOrderHeaders" href="Customers(1)/SalesOrderHeaders" />
```

这段代码显示了这个顾客的两个关系。第二个关系是对 SalesOrderHeaders 关系的引用，可以通过这个特定<link>元素中的 rel 特性和 title 特性看到这个引用。除了这个关系外，还有一个通过<link>元素的 href 特性指向关系自身的链接。声明的引用是 Customers(1)/SalesOrderHeaders，这意味着现在可以在浏览器中输入如下 URI：

[http://localhost:5526/AdventureWorksDataService.svc/Customers\(1\)/SalesOrderHeaders](http://localhost:5526/AdventureWorksDataService.svc/Customers(1)/SalesOrderHeaders)

输入这个 URI 意味着要深入研究 ID 值为 1 的顾客以及这个顾客在系统中的订单，获得的响应结果如图 13-30 所示。



图 13-30

4. 扩展关联

到现在为止，我们通过这些<link>元素已经了解了实体关联，并能够使用新的 URI 重新执行查询，以更深入地了解这些关联的详细信息。可以根据需要在同一个查询中提取这些关联。

可以使用通过 WCF 数据服务实现的一些查询字符串参数来获得这些关联。例如，如果按以下方式执行查询：

```
http://localhost:5526/AdventureWorksDataService.svc/Products(710)
```

这个查询的结果如程序清单 13-37 所示。注意其中一个链接指向 ProductCategory 实体集。在这种模式中，要得到这个产品的类别，就必须使用 `http://localhost:5526/AdventureWorksDataService.svc/Products(710)/ProductCategory` 进行一次额外的调用以获得该项。这将会得到完全不同的结果。

但是，如果想在一次调用中获得产品的相关数据点集合，那么可以在 URI 查询中使用 `expand` 关键字：

```
http://localhost:5526/AdventureWorksDataService.svc/Products(710)
?$expand=ProductCategory
```

要使这样的查询生效，需要在查询字符串中使用一个可用的关键字，这里使用的是 `expand` 关键字。只需在关联实体集的名称后面添加 `?$expand=` 这样的格式即可。该查询的结果集如程序清单 13-39 所示。

程序清单 13-39 添加额外的实体集

```
<?xml version="1.0" encoding="utf-8" ?>
<entry xml:base="http://localhost:5526/AdventureWorksDataService.svc/"
  xmlns="http://www.w3.org/2005/Atom"
  xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
```

```

<id>
  http://localhost:5526/AdventureWorksDataService.svc/Products(710)
</id>
<category term="AdventureWorksModel.Product"
  scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"
  />
<link rel="edit" title="Product" href="Products(710)" />
<link
  rel="http://schemas.microsoft.com/ado/2007/08/dataservices/
  related/ProductCategory" type="application/atom+xml;type=entry"
  title="ProductCategory" href="Products(710)/ProductCategory">
  <m:inline>
    <entry>
      <id>
        http://localhost:5526/AdventureWorksDataService.svc/
        ProductCategories(27)
      </id>
      <category term="AdventureWorksModel.ProductCategory"
        scheme="http://schemas.microsoft.com/ado/
        2007/08/dataservices/scheme" />
      <link rel="edit" title="ProductCategory"
        href="ProductCategories(27)" />
      <link rel="http://schemas.microsoft.com/ado/
        2007/08/dataservices/related/Products"
        type="application/atom+xml;type=feed"
        title="Products" href="ProductCategories(27)/Products" />
      <link rel="http://schemas.microsoft.com/ado/
        2007/08/dataservices/related/ProductCategory1"
        type="application/atom+xml;type=feed"
        title="ProductCategory1"
        href="ProductCategories(27)/ProductCategory1" />
      <link rel="http://schemas.microsoft.com/ado/
        2007/08/dataservices/related/ProductCategory2"
        type="application/atom+xml;type=entry"
        title="ProductCategory2"
        href="ProductCategories(27)/ProductCategory2" />
    </m:inline>
  </entry>
</link>
<title />
<updated>2012-11-28T05:47:51Z</updated>
<author>
  <name />
</author>
<content type="application/xml">
  <m:properties>
    <d:ProductCategoryID m:type="Edm.Int32">
      27
    </d:ProductCategoryID>
    <d:ParentProductCategoryID m:type="Edm.Int32">
      3
    </d:ParentProductCategoryID>
    <d:Name>Socks</d:Name>
    <d:rowguid m:type="Edm.Guid">
      701019c3-09fe-4949-8386-c6ce686474e5
    </d:rowguid>
  </m:properties>
</content>

```

```

        <d:ModifiedDate m:type="Edm.DateTime">
            1998-06-01T00:00:00
        </d:ModifiedDate>
    </m:properties>
</content>
</entry>
</m:inline>
</link>
<link rel="http://schemas.microsoft.com/ado/
    2007/08/dataservices/related/ProductModel"
    type="application/atom+xml;type=entry" title="ProductModel"
    href="Products(710)/ProductModel" />
<link rel="http://schemas.microsoft.com/ado/
    2007/08/dataservices/related/SalesOrderDetails"
    type="application/atom+xml;type=feed" title="SalesOrderDetails"
    href="Products(710)/SalesOrderDetails" />
<title />
<updated>2012-11-28T05:47:51Z</updated>
<author>
    <name />
</author>
<content type="application/xml">
    <m:properties>
        <d:ProductID m:type="Edm.Int32">710</d:ProductID>
        <d:Name>Mountain Bike Socks, L</d:Name>
        <d:ProductNumber>SO-B909-L</d:ProductNumber>
        <d:Color>White</d:Color>
        <d:StandardCost m:type="Edm.Decimal">3.3963</d:StandardCost>
        <d>ListPrice m:type="Edm.Decimal">9.5000</d>ListPrice>
        <d:Size>L</d:Size>
        <d:Weight m:type="Edm.Decimal" m:null="true" />
        <d:ProductCategoryID m:type="Edm.Int32">
            27
        </d:ProductCategoryID>
        <d:ProductModelID m:type="Edm.Int32">18</d:ProductModelID>
        <d:SellStartDate m:type="Edm.DateTime">
            2001-07-01T00:00:00
        </d:SellStartDate>
        <d:SellEndDate m:type="Edm.DateTime">
            2002-06-30T00:00:00
        </d:SellEndDate>
        <d:DiscontinuedDate m:type="Edm.DateTime" m:null="true" />
        <d:ThumbNailPhoto m:type="Edm.Binary">
            R0lGODlhUAAXAPcAAAAAIAAAACAAICAAAAgIAAgACAgICAgMDAwP8AAD/AP//AAAA//8A/wD/////wAAA
            AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
            AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMwAAZgAAmQAAzAAA/wAAzAAzMwAAZgAAZmQAAzAAZ/wBmAABmMwBmZgBmmQBmzA
            Bm/wCZAACZMwCZgCZmQCZzACZ/wDMAADMMwDMZgDMmQDMzADM/wD/AAD/MwD/ZgD/mQD/zAD//zMAADMAMzMAZjM
            AmTMAzDMA/zMzADMzMzMzZjMzmTMzzDMz/zNmADNmMzMzMzjNmTNmzDNm/zOZADOZMzOZZjOZmTOZzDOZ/zPMADPM
            MzPMZjPMmTPMzDPM/zP/ADP/MzP/ZjP/mTP/zDP//2YAAGYAMZYAZmYAmWYAZGYA/2YzAGYzMZYzZmYzmWYzzGYz/
            2ZmAGZmM2ZmZmZmmWZmzGZm/2aZAGaZM2aZZmaZmWaZzGaZ/2bMAGbMM2bM2mbMmWbMzGbM/2b/AGb/M2b/Zmb/mW
            b/zGb//5kAAJkAM5kAZpkAmZkAzJkA/5kzAJkzM5kzZpkzmZkzzJkz/5lMAJlmM5lmZplmmZlmzJlm/5mZAJmZM5m
            ZZpmZmZmZzJmZ/5nMAJnMM5nMZpnMmZnMzJnM/5n/AJn/M5n/Zpn/mZn/zJn//8wAAMwAM8wAZswAmcwAzMwA/8wz
            AMwzM8wzZswzmcwzzMwz/8xmAMxmM8xmZsxmxcxmzMxm/8yZAMyZM8yZzsyZmcyZzMyZ/8zMAMzMM8zMZszMmczMz
        </d:ThumbNailPhoto>
    </m:properties>
</content>

```



```

MzM/8z/AMz/M8z/Zsz/mcz/zMz///8AAP8AM/8AZv8Amf8AzP8A//8zAP8zM/8zZv8zmf8zzP8z//9mAP9mM/9mZv
9mmf9mzP9m//+ZAP+ZM/+ZZv+Zmf+ZzP+Z///MAP/MM//MZv/Mmf/MzP/M////AP//M///Zv//mf//zP///yH5BAE
AABAALAAAAABQADEAAAJ/AP8JHEiwoMGDCBMqXMiwoCOHECNKnEixosWLGDNg3Mixo8ePIEOKHEmypoMmTKFOqXJkR
BYqBLhFGZPnQ5ct/MxPmpMnQpsCZNM/CfBnTZ86gQ3HeRMORadGlQpUqJfoUZ9KnVH9GxVhUKtCoVaWKn
ZrVK9SmVMPuVHvWrfisPjd+LbuW7tmvb8t6nJuXIFutfbH2lSt07ta/eeOy3clTYuGtjS8yjUy5suXLmDHHdRjWIG
PGIjdDBA3YL2SQVY+mvQsVL16yqLOqfuyWtlHZbTv+nYl
76G67H38DTs068GrSkoMSN+62+fKQqrW2Xe6aem7CSaf6fq7ceevTmcOLEh9Pvrz58+jTql/Pvr379+8DAgA7
  </d:ThumbnailPhoto>
  <d:ThumbnailPhotoFileName>
    no_image_available_small.gif
  </d:ThumbnailPhotoFileName>
  <d:rowguid m:type="Edm.Guid">
    161c035e-21b3-4e14-8e44-af508f35d80a
  </d:rowguid>
  <d:ModifiedDate m:type="Edm.DateTime">
    2004-03-11T10:01:36.827
  </d:ModifiedDate>
</m:properties>
</content>
</entry>

```

从这段代码中可以看出，为类别指定的<link>元素现在扩展为可以包含曾经是单独的内联调用。除了扩展单个的关联实体集外，还可以扩展多个项：

```

http://localhost:5526/AdventureWorksDataService.svc/Products(710)
  ?$expand=ProductCategory,ProductModel

```

在这个例子中，ProductCategory 和 ProductModel 部分都在 Product 实体集调用中进行扩展。

也可以深入查看嵌套的关联实体集。例如，如果 ProductCategory 实体与另一个实体 SimilarCategories 有关系，就可以把 URI 构建为：

```

http://localhost:5526/AdventureWorksDataService.svc/Products(710)
  ?$expand=ProductCategory/SimilarCategories

```

使用上面的构造代码，可以查看产品列表中 ID 为 710 的产品，并可以扩展这个产品的类别部分。之后，在类别中，与这个类别相关的类似类别也会包含在结果集中。可以方便地使用许多强大的功能深入研究嵌套关系。

5. 对结果集排序

操作来自 URI 查询的结果集的另一种方式是：将得到的结果集按照定义的特定顺序排列。可以使用 orderby 关键字作为查询字符串命令来实现排序。

```

http://localhost:5526/AdventureWorksDataService.svc/Products
  ?$orderby=Name

```

在上面的例子中，得到了一个根据实体的 Name 字段值以字母表顺序排列的完整产品列表。还可以对给定顺序的列表指定升序或降序，默认采用升序。也就是说，也可以使用下面的代码实现前面的查询：

```

http://localhost:5526/AdventureWorksDataService.svc/Products
  ?$orderby=Name asc

```

注意，URI 中的 Name 和 asc 项之间有一个空格。如果想要以相反的顺序排列，或者进行降序排列，可以使用下面的 URI 构造：

```
http://localhost:5526/AdventureWorksDataService.svc/Products
?orderby=Name desc
```

也可以使用 WCF 数据服务执行嵌套排序：

```
http://localhost:5526/AdventureWorksDataService.svc/Products
?orderby=Color asc, Name asc
```

6. 移动结果集

这个接口的终端用户可能会处理相当大的结果集，这取决于数据库的内容。如果只需要部分的数据库表并请求所有顾客(可能是 100 000 甚至更多顾客)，该怎么办？

在这种情况下，WCF 数据服务提供了一个功能，可以将内容的较小子集获取为页面，并遍历需要的页面。通过结合使用如下两个查询字符串命令实现该操作：top 和 skip。

这两个查询字符串命令在单独使用时功能也十分强大。使用 top 命令可以提取最顶端的 n 项(基于所使用的排序方法)。例如，考虑下面的命令：

```
http://localhost:5526/AdventureWorksDataService.svc/Products?$top=5
```

这里根据 ID 的值从数据库中取出最顶端的 5 个实体并返回它们。如果需要基于其他值的前 5 个实体，可以使用如下代码：

```
http://localhost:5526/AdventureWorksDataService.svc/Products
?orderby=Name desc&$top=5
```

运行修改后的代码，则会根据产品名，将排在最前面的 5 个实体返回到结果集中。

可以使用 skip 命令跳过第一组定义的项。例如，可以执行如下操作：

```
http://localhost:5526/AdventureWorksDataService.svc/Products?$skip=5
```

在这个例子中，返回紧挨最顶端 5 个实体后边的 5 个实体。关于这个命令的值，我们可能会有一些疑问，但是显然可以看出它与 top 命令一起使用时的强大功能。

```
http://localhost:5526/AdventureWorksDataService.svc/Products?$skip=10&$top=10
```

在这个例子中，跳过了最前边的 10 个实体，然后获取第 10 个实体之后紧跟的 10 个实体。也就是说，如果一个页面上有两个各包含 10 项的实体集，那么使用上述两条命令可以得到这个页面上的全部项。通过这个过程使用 URI 命令获得需要的页面数据，就可以使执行某种类型的数据库分页变得更加容易。

7. 筛选内容

最后这个命令是一个功能最强大的命令，该命令是筛选命令，它允许在检索数据库时筛选感兴趣的内容。所有这些操作都是通过使用 filter 命令实现的：

```
http://localhost:5526/AdventureWorksDataService.svc/Products
?filter=Color eq 'Red'
```

使用 filter 命令时，在命令前边添加一个美元符号(\$)，这个命令的值是 Color eq 'Red'。使用这条筛选命令，可以请求红色产品的列表。在这个例子中，数据库属性是 Color，并且在 URI 中以正确的大小写指定这个属性是非常重要的。也就是说，如果使用 color 而不是 Color，在结果集中就不会得到任何项。

将 Red 值放在单引号中，并且将运算符指定为一组字符，而不是真正的等号(=)。在使用 filter 命令时，通过 eq 字符串指定等于运算符。表 13-2 列出了可用的逻辑运算符。

表 13-2

逻辑运算符	说 明	示 例
eq	等于	Color eq 'Red'
ne	不等于	Color ne 'Red'
gt	大于	\$filter = ListPrice gt 20
ge	大于等于	\$filter = ListPrice ge 20
lt	小于	\$filter = ListPrice lt 20
le	小于等于	\$filter = ListPrice le 20
and	逻辑与	\$filter = ListPrice gt 0 and StandardCost gt 0
or	逻辑或	\$filter = ListPrice gt 0 or StandardCost lt 100
not	逻辑非	\$filter = ListPrice gt 0 not ProductName eq 'Red'

除了逻辑运算符外，还可以使用许多数学运算符，如表 13-3 所示。

表 13-3

数学运算符	说 明	示 例
add	加	\$filter = ListPrice add 5 gt 20
sub	减	\$filter = ListPrice sub 5 gt 20
mul	乘	\$filter = ListPrice mul 5 gt 20
div	除	\$filter = ListPrice div 5 gt 20
mod	取模	\$filter = ListPrice mod 100 gt 20

可以使用许多字符串函数、日期函数以及数学函数：

- substringof
- startswith
- indexof
- remove
- substring
- toupper
- concat
- hour
- month
- endswith
- length
- insert
- replace
- tolower
- trim
- day
- minute
- second

- year
- floor

- round
- ceiling

13.8.10 在 ASP.NET 中使用 WCF 数据服务

现在,我们已经知道了如何建立 WCF 数据服务,下面介绍如何在 ASP.NET 应用程序中使用该服务。注意,可以在任意类型的 .NET 应用程序中使用 WCF 数据服务,但本章主要介绍它在 ASP.NET 中的使用。

在使用数据服务时,首先在项目中创建一个标准的 ASP.NET Web 窗体。接下来,建立一个只包含样式化 GridView 服务器控件的简单页面。程序清单 13-40 显示了这个页面的代码。

程序清单 13-40 带有 GridView 控件的标准 ASP.NET 页面

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Working with Data Services</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:GridView ID="GridView1" runat="server" BackColor="White"
            BorderColor="#DEDFDE" BorderStyle="None" BorderWidth="1px"
            CellPadding="4"
            ForeColor="Black" GridLines="Vertical">
            <RowStyle BackColor="#F7F7DE" />
            <FooterStyle BackColor="#CCCC99" />
            <PagerStyle BackColor="#F7F7DE" ForeColor="Black"
                HorizontalAlign="Right" />
            <SelectedRowStyle BackColor="#CE5D5A" Font-Bold="True"
                ForeColor="White" />
            <HeaderStyle BackColor="#6B696B" Font-Bold="True"
                ForeColor="White" />
            <AlternatingRowStyle BackColor="White" />
        </asp:GridView>
    </form>
</body>
</html>
```

有了这个基本页面后,在 Visual Studio 的 Solution Explorer 中右击项目,在弹出的菜单中选择 Add Service Reference 命令,打开 Add Service Reference 对话框。

WCF 数据服务是标准的 .svc 文件,因此可以在提供的文本框中引用 AdventureWorksDataService.svc 文件。单击 Go 按钮,会显示如图 13-31 所示的对话框。

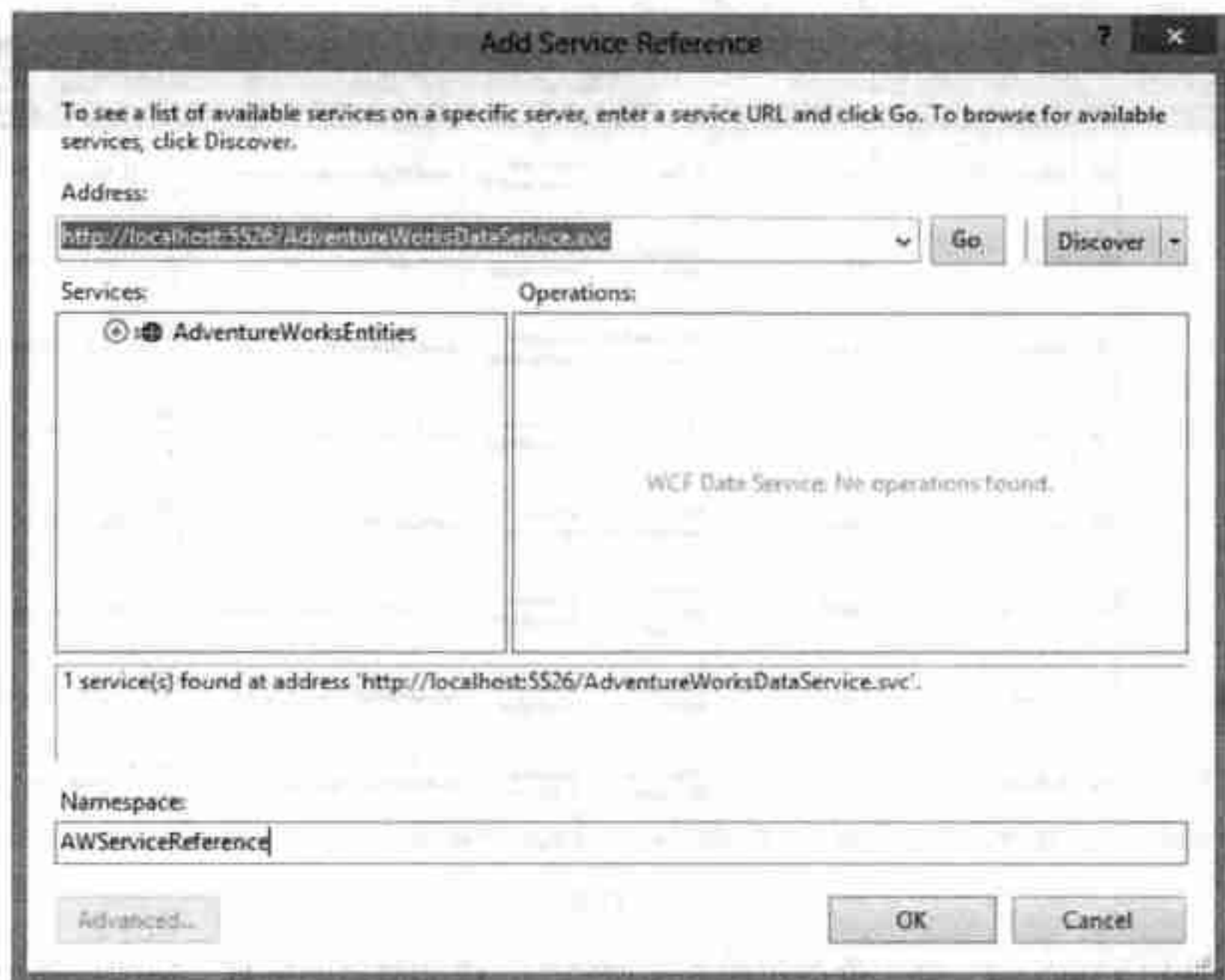


图 13-31

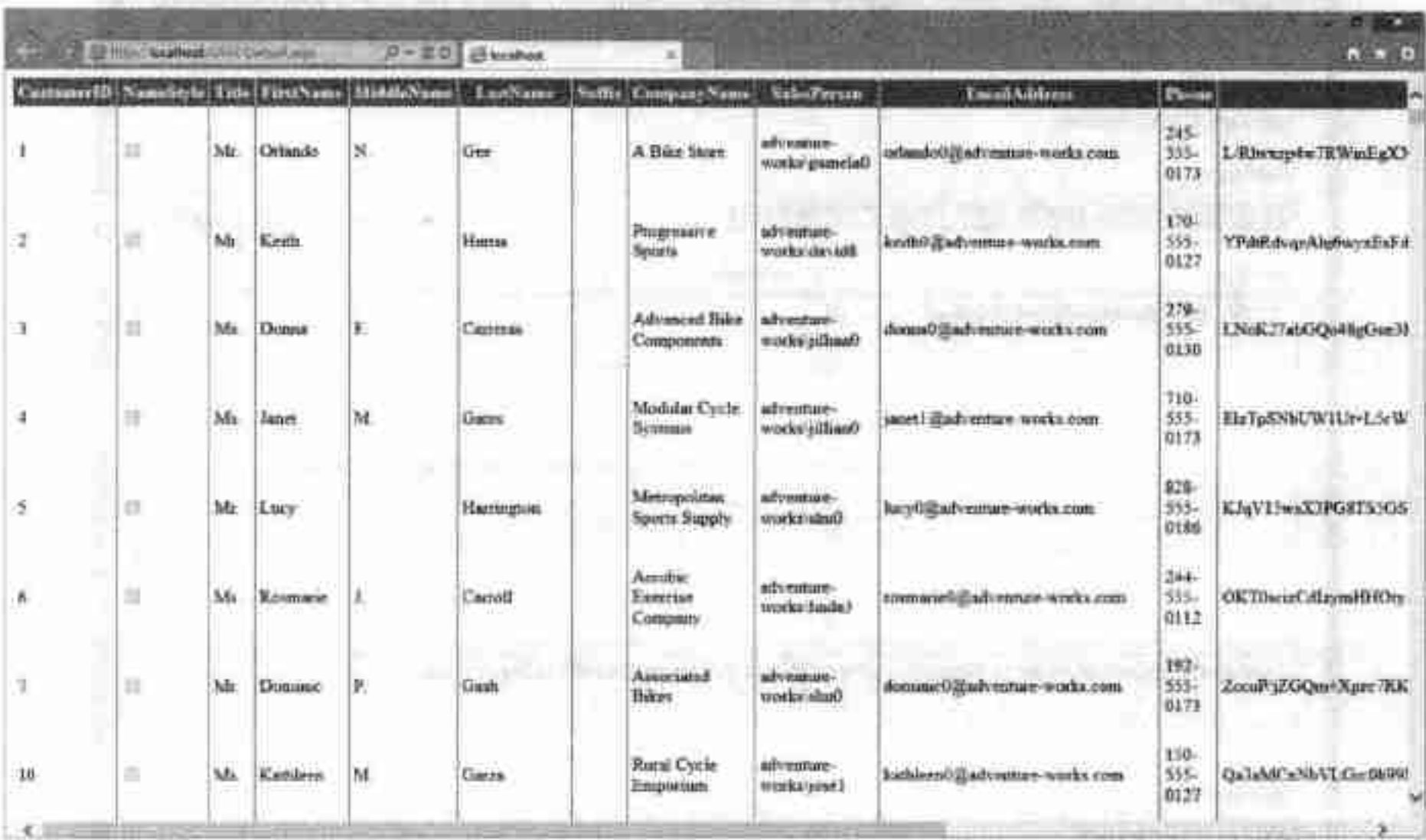
在图 13-31 中可以看到，当前所有底层对象都在对话框中显示出来。在 Namespace 文本框中，可以把引用命名为 `AWServiceReference`，然后单击 OK 按钮以保存这个配置。

下一步是在隐藏代码页面中使用这个引用，如程序清单 13-41 所示(在本章的下载代码中，这个文件是 Chapter13-WcfDSConsumer-CS 中的 `Default.aspx.cs`)。

程序清单 13-41 使用 WCF 数据服务

```
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        AdventureWorksEntities svc = new AdventureWorksEntities(new
            Uri("http://localhost:5526/AdventureWorksDataService.svc"));
        GridView1.DataSource = svc.Customers;
        GridView1.DataBind();
    }
}
```

在上面的代码中，仅仅是对实体数据模型进行了引用，通过数据服务的 URI 实现该引用。这个 EDM 实例包含与底层服务交互的所有功能的列表。在程序清单 13-41 中，使用 `svc.Customers` 返回了整个 Customers 表。之后，将结果集绑定到 GridView 控件。图 13-32 显示了绑定结果。



CustomerID	NameStyle	Title	FirstName	MiddleName	LastName	Suffix	CompanyName	SalesPerson	EmailAddress	Phone	Photo
1		Mr.	Oswaldo	N.	Gre		A Bike Store	adventure-works/gustaf0	oswald0@adventure-works.com	245-555-0173	L/R0wcrptwTRWinEgX3
2		Mr.	Keth		Huma		Progressive Sports	adventure-works/david0	keth0@adventure-works.com	170-555-0127	YPdRdvqA4ufwyzE5f4
3		Mr.	Donna	E.	Cattaneo		Advanced Bike Components	adventure-works/jillan0	donna0@adventure-works.com	279-555-0130	LN0K274tGQ04hgGse3
4		Ms.	Jane	M.	Garc		Modular Cycle Systems	adventure-works/jillian0	jane1@adventure-works.com	710-555-0173	ElzTp5NNUW1Uv+L5eW
5		Mr.	Lucy		Harrington		Metropolitan Sports Supply	adventure-works/alan0	lucy0@adventure-works.com	828-555-0186	KJqV11waXJPG8T53G5
6		Ms.	Rosmarie	J.	Carroll		Aerobic Exercise Company	adventure-works/luke0	rosmarie0@adventure-works.com	244-555-0112	OKT0scizCafzgmH00y
7		Mr.	Donnie	P.	Gash		Associated Bikes	adventure-works/alan0	donnie0@adventure-works.com	182-555-0173	ZocuPjZGQm+Xpre7KK
10		Ms.	Kathleen	M.	Garc		Rural Cycle Emporium	adventure-works/jane1	kathleen0@adventure-works.com	150-555-0127	Qa1eAdCsNbVL_Ge0d090

图 13-32

在代码中使用 LINQ 时，除了能使用像前面代码中一样的简单查询外，还可以使用一些本章前面介绍过的命令逻辑。程序清单 13-42 显示了一个针对 Customers 表的查询，通过这个查询可以得到 Customers 表中 CompanyName 包含 Bike 的顾客。

程序清单 13-42 使用 LINQ

```
AdventureWorksEntities svc = new AdventureWorksEntities(new
    Uri("http://localhost:5526/AdventureWorksDataService.svc"));

var query = svc.Customers.Where(w => w.CompanyName.Contains("Bike"));

GridView1.DataSource = query;
GridView1.DataBind();
```

在这段代码中执行 LINQ 查询，之后将这个查询绑定为 GridView 控件的 DataSource 值。这将在网格中生成另一个项列表。

13.9 ASP.NET Web API

ASP.NET 4.5 给平台带来了许多新功能，其中一项最有用的功能是 Web API。在推出 ASP.NET 4.5 之前，许多开发人员都抱怨，配置 WCF 以创建 RESTful 服务所花的时间要比开发该服务还长。所以，为了帮助提供应用程序的功能，创建了 Web API。



Web API 是 ASP.NET MVC 应用程序。如果没有 ASP.NET MVC 经验，建议在阅读完本章之前，先学习第 34 章。

Web API 是严重依赖 HTTP 的 ASP.NET MVC 应用程序，所以开发人员可以使用 HTTP 动词和有意义的路由，建立坚固的 RESTful 服务。

13.9.1 建立第一个 Web API 项目

建立 Web API 项目很简单。Visual Studio 2012 有一个很好的模板，有助于构建 Web API 项目。构建第一个 Web API 项目时会用到它。在一些情况下，开发人员可以使用 Web API 项目模板生成功能全面的 Web API，而无须编写任何代码(表示仅使用拖放功能，进行配置)。

首先，创建一个新的 ASP.NET MVC 4 项目，如图 13-33 所示。

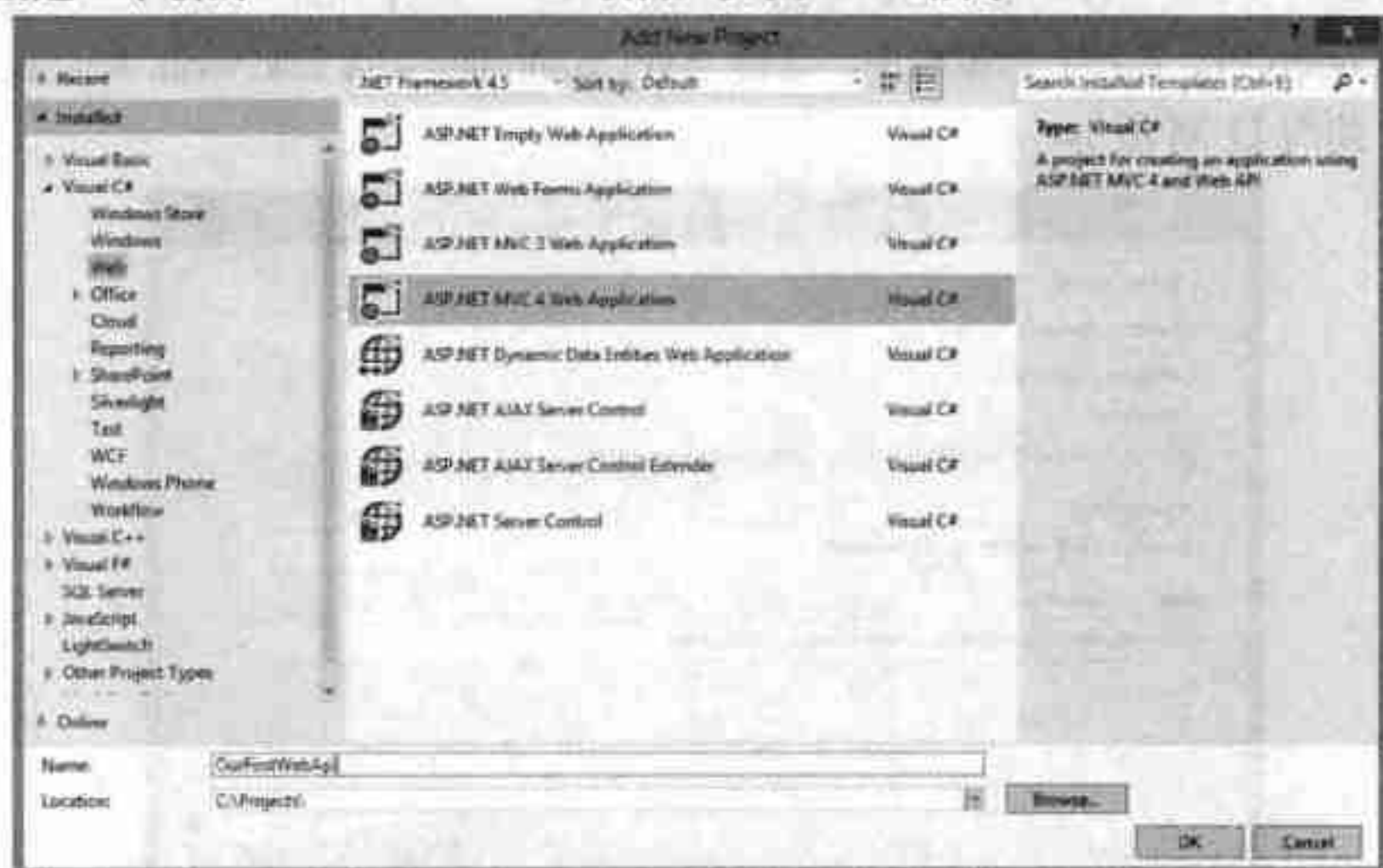


图 13-33

单击 OK 时，系统应提示定制 ASP.NET MVC 4 项目，如图 13-34 所示。对于这个例子，选择 Web API 模板。因为这是第一个服务，这个项目仅用于 API，所以给项目使用默认的 Razor 视图引擎。

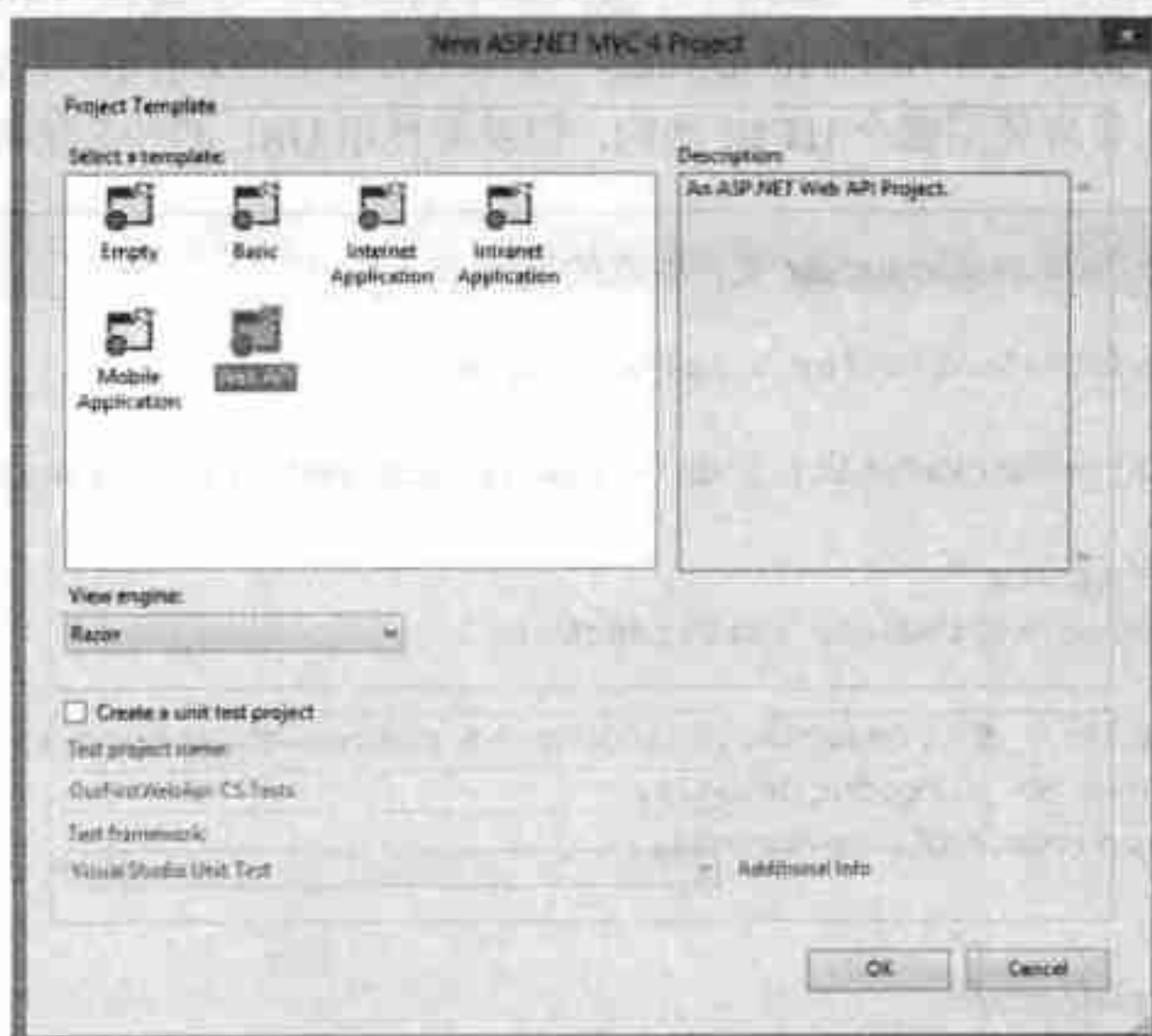


图 13-34

创建项目后，会立即进入默认控制器 `ValuesController`。删除这个控制器，因为稍后要建立自己的控制器。

在 Web 上提供 API 时，用户很有可能与数据源交互操作。此时，使用本章前面用过的数据库 `AdventureWorks`，把它放在 `App_Data` 文件夹中。

接着，创建 EF 模型，该模型类似于前面建立的、连接 WCF 的模型。但本例把 `AdventureWorks.edmx` 文件添加到 `Models` 文件夹中。在继续之前，先构建这个项目。构建项目时，模型会编译到项目的二进制文件中，在 Visual Studio IDE 中使用。

接着必须建立一个新的控制器。右击 `Controllers` 文件夹，选择 `Add | Controller`。把新控制器命名为 `ProductsController`。在控制器模板中，选择“API controller with read/write actions, using Entity Framework”，如图 13-35 所示。

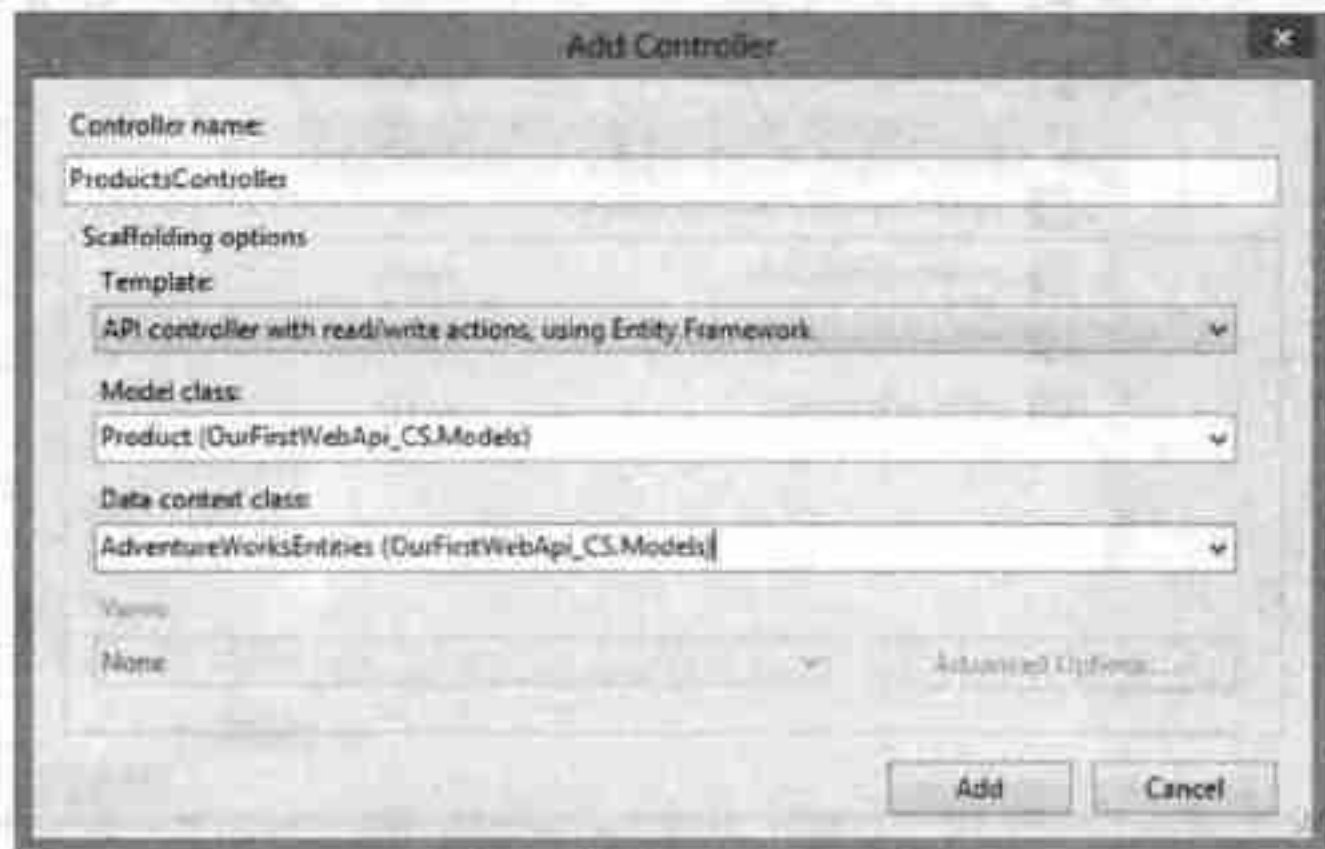


图 13-35

在 `Model class` 下拉菜单中，选择 `Product` 模型。这允许控制器绑定到数据源包含的 `Products` 表。`Data context class` 下拉菜单允许在多个数据上下文对象之间切换。本例选择 `AdventureWorkEntities`。

`ProductsController` 文件包含 API 的核心功能，如程序清单 13-43 所示。注意，每个方法的顶部都有注释，告诉开发人员应使用哪个 HTTP 动词，以及应使用 URL 的什么结构来获得预期的结果。

程序清单 13-43 给 `ProductsController` 文件生成的代码

```
public class ProductsController : ApiController
{
    private AdventureWorksEntities db = new AdventureWorksEntities();

    // GET api/Products
    public IEnumerable<Product> GetProducts()
    {
        var products = db.Products.Include(p => p.ProductCategory)
                                   .Include(p => p.ProductModel);
        return products.AsEnumerable();
    }

    // GET api/Products/5
    public Product GetProduct(int id)
    {

```

```

        Product product = db.Products.Find(id);
        if(product == null)
        {
            throw new
                HttpResponseException(Request.CreateResponse(HttpStatusCode.NotFound));
        }

        return product;
    }

// PUT api/Products/5
public HttpResponseMessage PutProduct(int id, Product product)
{
    if(ModelState.IsValid && id == product.ProductID)
    {
        db.Entry(product).State = EntityState.Modified;

        try
        {
            db.SaveChanges();
        }
        catch(DbUpdateConcurrencyException)
        {
            return Request.CreateResponse(HttpStatusCode.NotFound);
        }

        return Request.CreateResponse(HttpStatusCode.OK);
    }
    else
    {
        return Request.CreateResponse(HttpStatusCode.BadRequest);
    }
}

// POST api/Products
public HttpResponseMessage PostProduct(Product product)
{
    if(ModelState.IsValid)
    {
        db.Products.Add(product);
        db.SaveChanges();

        HttpResponseMessage response = Request.CreateResponse(HttpStatusCode.Created,
            product);
        response.Headers.Location = new Uri(Url.Link("DefaultApi",
            new { id = product.ProductID }));
        return response;
    }
    else
    {
        return Request.CreateResponse(HttpStatusCode.BadRequest);
    }
}

// DELETE api/Products/5

```



```

public HttpResponseMessage DeleteProduct(int id)
{
    Product product = db.Products.Find(id);
    if(product == null)
    {
        return Request.CreateResponse(HttpStatusCode.NotFound);
    }

    db.Products.Remove(product);

    try
    {
        db.SaveChanges();
    }
    catch(DbUpdateConcurrencyException)
    {
        return Request.CreateResponse(HttpStatusCode.NotFound);
    }

    return Request.CreateResponse(HttpStatusCode.OK, product);
}

protected override void Dispose(bool disposing)
{
    db.Dispose();
    base.Dispose(disposing);
}
}

```

13.9.2 理解 Web API 路由

注意 Web API 控制器的几个区别。首先，类没有继承 `Controller`，而继承了 `ApiController`。其次，方法没有使用 HTTP 动词特性，HTTP 动词放在方法名的第一部分，这样就可以在方法名中使用 HTTP 动词。这是通过 `ApiController` 类实现的。

如前所述，程序清单 13-43 中方法的顶部都有注释，显示了使用该方法所需的动词和期望的 URL。URL 是根据 Web API 的路由生成的。默认的 Web API 路由如程序清单 13-44 所示。

程序清单 13-44 默认的 Web API 路由

```

public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}

```

因为 Web API 使用 ASP.NET MVC，所以可以添加其他路由，或修改路由模板。

13.9.3 使用 Web API

建立了 Web API 后, 就需要由客户端使用它。因为 Web API 使用 HTTP, 所以客户端可以是移动设备, 例如 Windows Phone, 或者桌面应用程序, 或者使用另一种技术建立的 Web 应用程序。打开 Internet Explorer, 使用 <http://localhost:43059/api/products/710> 访问 Web API, 结果如图 13-36 所示。

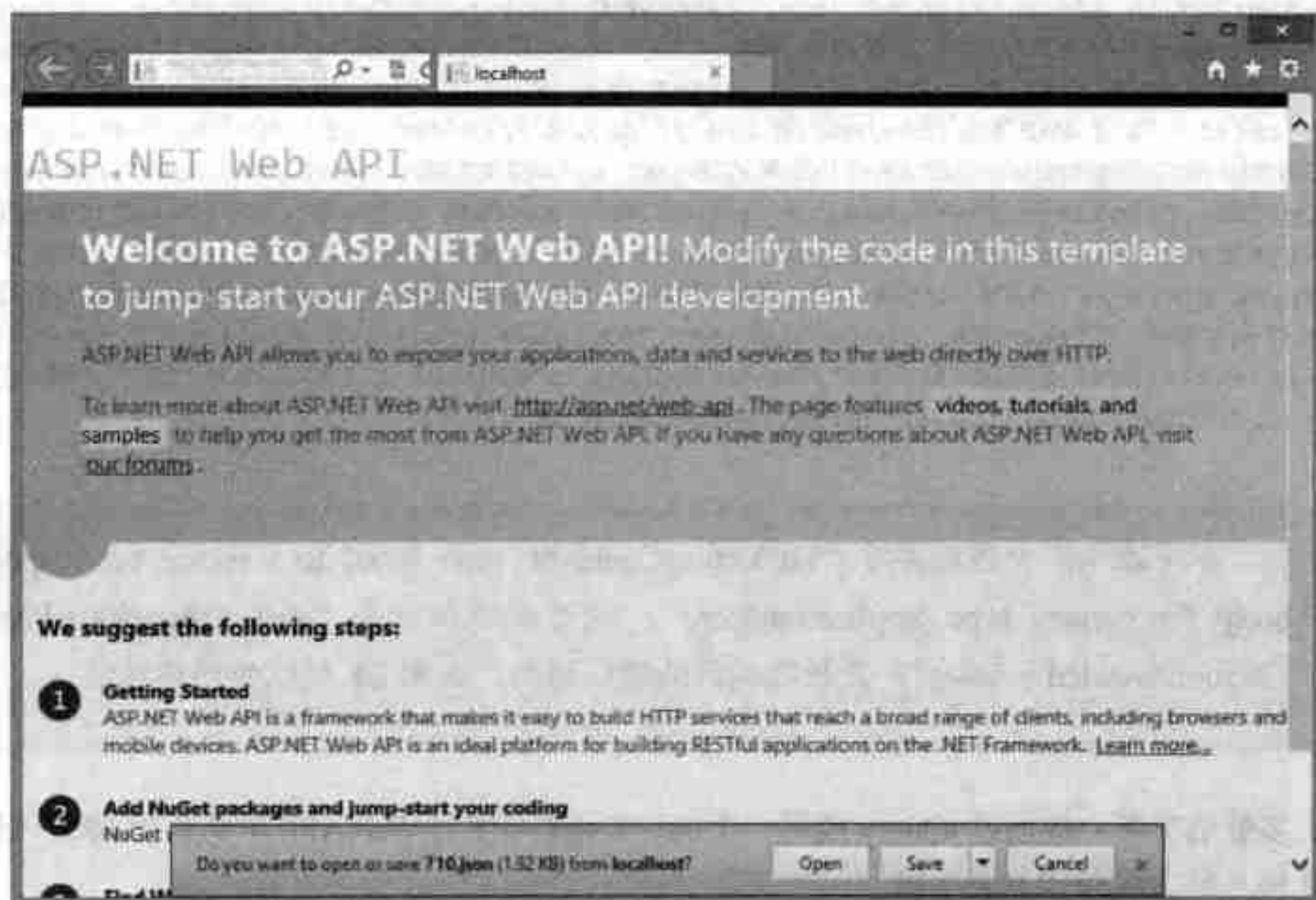


图 13-36

在这个例子中，准备下载的文件是 JSON 文件，如程序清单 13-45 中的完整 HTTP 响应所示。

程序清单 13-45 Web API 调用的完整 HTTP 响应

[illegible]


```
wAzAAAzMwAzZgAzmqAzZAAz/wBmAABmMwBmZgBmmQBmzABm/wCZAACZMwCZZgCZmQCZzACZ/wDMAADMMwDMZgDMmQ
DMzADM/wD/AAD/MwD/ZgD/mQD/zAD//z MAADMAMzMAZjMAMTMAzDMA/zMzADMzMzMzZjMzmTMzzDMz/
zNmADNmMzMzjNmTNmzDNm/zOZADOZMzOZjOZmTOZzDOZ/ z PMADPMMzPMZjPMmTPMzDPM/zP/ADP/MzP/
ZjP/mTP/zDP//2YAAGYAM2YAZmYAmWY AzGYA/2YzAGYzM2YzZmYzmWYzzGYz/2ZmAGZmM2ZmZmZmmWZmzGZm/
2aZAGaZM2aZZmaZmWaZzGaZ/2bMAGbMM2bMZmbMmWbMzGbM/2b/AGb/M2b/Zmb/mWb/zGb//5kAAJkAM5kAZpkA
mZkAzJkA/5kzAJkzM5kzZpkzmZkzzJkz/5lmAJlmM5lmZplmmZlmzJlm/5mZAJmZM5mZZpmZmZmZzJmZ/5nMAJnMM
5nMZpnMmZnMzJnM/5n/AJn/M5n/Zpn/mZn/zJn//8wAAMwAM8wAZswAmcwAzMwA/8wzAMwzM8wzZswzmcwzzMwz/8
xmAMxmM8xmZsxmmcxmzMxm/8yZAMy ZM8yZZsyZmcyZzMyZ/8zMAMzMMBzMZszMmczMzMz/8z/AMz/M8z/Zsz/
mcz/zMz///8AAP8AM/8AZv8Amf8AzP8A//8zAP8zM/8zZ v8zmf8zzP8z//9mAP9mM/9mZv9mmf9mzP9m//+ZAP
+ZM/+ZZv+Zmf+ZzP+Z///MAP/MM//MZv/Mmf/MzP/M////AP//M///Zv//mf//zP///yH5BAEAAABAALAAAAABQADE
AAAJ/AP8JHEiwoMGDCBMqXMIwocOHECNKnEixosWLGDNq3Mi xo8ePIEOKHEm ypMmTKFOqXJkRBYqBLhfGZPnQ5ct/
MxPmpMnQpsCZNM/CfBnTZ86gQ3HeRMoRadGlQpUqJfoUZ9KnVH9GxVhUK tCoVaWKnZrVK9SmVMPuVHvWrFisPjd+
LbuW7tmvb8t6nJuXIFutfbH2lSt07ta/eeOy3clTYuGtjS8yjUy5suXLMdHHdRjWIGPGIjdDBA3YL2SQVY+mvQsVL
16yqLOqfuyWtlHZbTv+nYl 76G67H38DTs068GrSkoMSN+62+fKQqrW2Xe6aem7CSaf6fq7ceevTmcOLEh9Pvrz58+
jTq1/Pvr379+8DAgA7","ThumbnailPhotoFileName":"no_image_available_small.gif","rowguid":"16
1c035e-21b3-4e14-8e44-af508f35d80a","ModifiedDate":"2004-03-11T10:01:36.827","ProductCate
gory":null,"ProductModel":null,"SalesOrderDetails":[]}
```



如果在响应中得到异常“The'ObjectContent'1' type failed to serialize the response body for content type 'application/json;”，就可以添加命令“db.Configuration.ProxyCreationEnabled = false;”，更新控制器的默认结构，关闭 EF 代理的创建功能。

在大多数情况下，应通过 jQuery 或另一个 JavaScript 库使用 Web API(详见第 25 章)。但是，可以通过其他几种方式使用 Web API。

如果使用 ASP.NET Web 窗体或 ASP.NET MVC，并且希望使用服务器端代码连接 Web API，就可以使用 HttpClient 调用来完成，HttpClient 调用则可以使用 Web API Client 库来完成。



如果没有在这个项目中引用 Web API Client 库，就可以从 NuGet Package Manager Console 上下载它；如果它已经下载为解决方案的一部分，就更新已安装这个包的项目。对于后者，可以进入 Package Manager，找到包，选择 Manage。NuGet 详见附录 G。

为了说明这一点，创建一个新的 ASP.NET Web Forms 项目 ApiConsumer。因为要使用模型中的一个表，所以需要创建一个类，以转换 JSON 结果。为了简单起见，应使用程序清单 13-46 中的两个属性。如果希望获得模型中的所有属性，就需要把它们添加到类中。

程序清单 13-46 Product 类

```
public class Product
{
    public string Name { get; set; }
    public string Color { get; set; }
}
```


接着,创建新页面 Default.aspx, 使用 HttpClient, 传递 URI /api/products/710。如果得到成功的响应,就解析结果。否则在单独的标签中显示异常,如程序清单 13-47 所示。

程序清单 13-47 调用 Web API 的 default.aspx 页面

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Net.Http" %>
<%@ Import Namespace="System.Net.Http.Headers" %>
<!DOCTYPE html>
<script runat="server">
    protected void Page_Load(Object sender, EventArgs e)
    {
        HttpClient client = new HttpClient();
        client.BaseAddress = new Uri("http://localhost:43059/");
        client.DefaultRequestHeaders.Accept.Add(
            new MediaTypeWithQualityHeaderValue("application/json"));

        HttpResponseMessage response = client.GetAsync("api/products/710").Result;
        if (response.IsSuccessStatusCode)
        {
            var product = response.Content.ReadAsAsync<Product>().Result;
            NameLabel.Text = product.Name;
            ColorLabel.Text = product.Color;
        }
        else
        {
            ErrorLabel.Text = response.ReasonPhrase;
        }
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Web API Demo</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            Product Name: <asp:Label ID="NameLabel" runat="server" /><br />
            Color: <asp:Label ID="ColorLabel" runat="server" /><br /><br />
            <asp:Label ID="ErrorLabel" runat="server" />
        </div>
    </form>
</body>
</html>
```

图 13-37 显示了收到成功的 HTTP 响应代码时的结果。

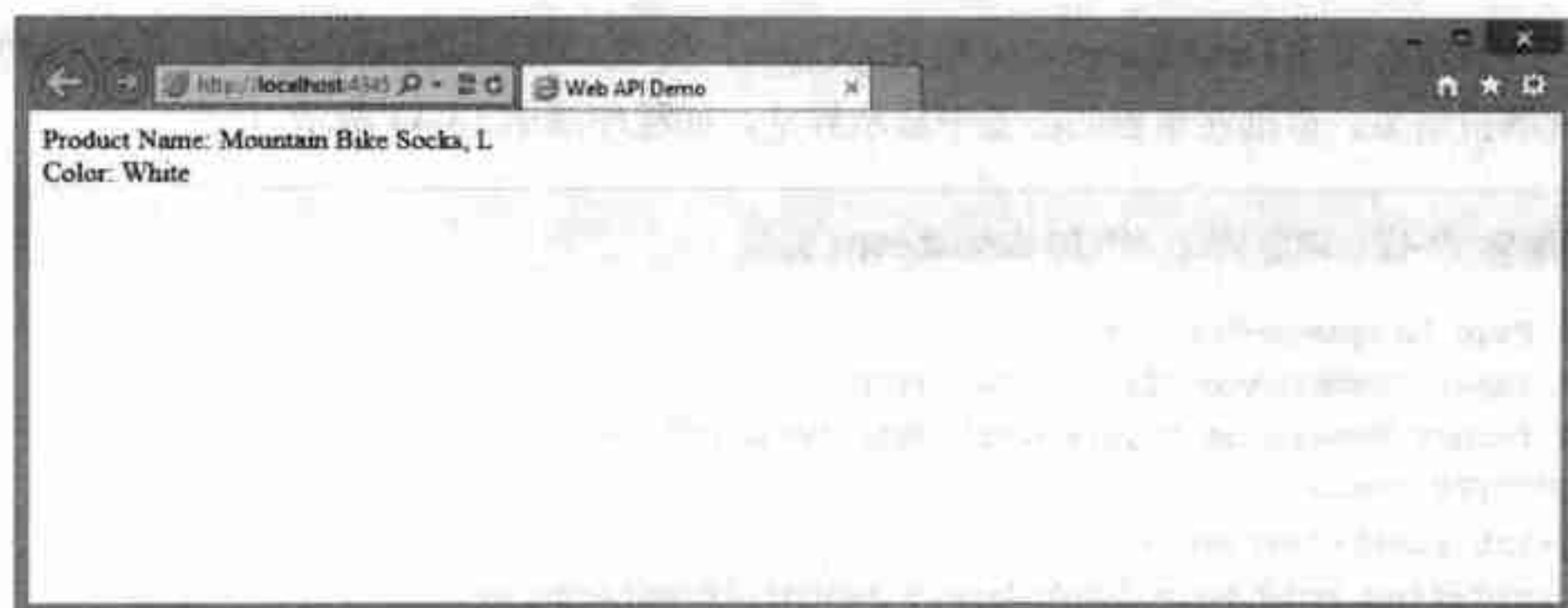


图 13-37

有时，开发人员可能不使用 Web API，但需要测试它。此时，可以看看测试 Web API 的几个工具。测试 Web API 的最流行工具是 Fiddler。Fiddler 是一个免费的实用程序，可以检查遵循 HTTP 或 HTTPS 的流量。在该实用程序中，可以创建新的 HTTP 请求报头，把请求发送给特定的 URL。



要下载 Fiddler，可以在 Internet 上搜索 Fiddler download。此外还有测试 REST API 服务调用的几个在线工具。例如，要找到其他工具，可以在 Internet 上搜索 How to test REST API service calls。

13.10 本章小结

本章概述了 .NET 平台上的 Web 服务。Web 服务本身就需要一本书来讨论。本章只探讨了如何把数据和逻辑显示为 SOAP，以及如何直接在自己建立的 ASP.NET 应用程序中使用这些 SOAP 消息。

虽然不是很详尽，但本章还是全面概述了这个架构的基础知识。如果深入研究这个技术，就会发现其功能非常强大，并可以进行扩展。

WCF 数据服务是显示数据库内容的一种强大方式。本章介绍了在接口上使用命令以筛选指定内容的方法。许多命令都可以用作 URI 的一部分以获得特定的结果集。但是，通过使用代码，尤其是 LINQ，也可以在处理 ASP.NET 页面时从要用到的接口中获得结果。

最后，本章概述了 ASP.NET 4.5 中的新 Web API，这是迄今为止创建 RESTful Web 服务的最简单解决方案。

第Ⅳ部分

提 供 程 序

- 第 14 章 提供程序模型概述
- 第 15 章 扩展提供程序模型

第 14 章

提供程序模型概述

本章要点

- 提供程序的定义
- ASP.NET 4.5 中的提供程序模型
- 配置提供程序

建立应用程序时，ASP.NET 提供程序模型是要理解的一个重要架构。ASP.NET 是为 Internet 建立应用程序的一种方式。这意味着，应用程序的显示代码要通过 HTTP 传输，而 HTTP 是一种无状态的协议。ASP.NET 在断开连接的体系结构中工作。这个模型的本质意味着，请求传入该模型后，模型会作出响应，然后把响应发送回去。在该体系结构上，ASP.NET 不区分请求。包含 ASP.NET 应用程序的服务器只是响应传送给它的所有请求。

这就是说，建立 Web 应用程序的开发人员必须考虑用户在使用应用程序时，如何在各个请求之间保存状态。要为用户保存状态，就必须在某种数据存储中记录用户的状态。可以通过几种方式实现该操作，但这些方式都不太完美。因此，必须选择一种合适的方法。



有关维护 ASP.NET 应用程序中状态的内容，可参阅第 21 章。

可以通过以下几种方法来存储状态，包括：

- 应用程序状态
- 会话状态
- 高速缓存对象

可以在服务器上使用这些方法，还可以使用自己的定制方法，例如使用自己的定制模式把状态存储在数据库中。也可以把状态写回客户端，或者直接写入客户端计算机，或者把状态放在响应的 HTML 输出中。其他方法包括：

- Cookies

- 查询字符串
- 隐藏字段
- 视图状态

无论是 ASP.NET 中内置的提供程序，还是由你自己定制的提供程序，都能够很好地管理状态。接下来我们查看提供程序是什么以及如何使用它。



与本书的其他章节不同，本章包含所有的程序清单和纯文本文件。在大多数情况下，文本文件只包含程序清单，且可能不是完整的配置文件。

14.1 提供程序概述

这些方法都不错，但它们大都比较基础，生命周期也较短。ASP.NET 4.5 包含了一些系统，如成员资格和角色管理系统，它们可以在多个请求/响应事务处理过程中处理用户的状态。实际上，这些系统需要的状态管理功能大大超出了以前的状态管理方法中所能提供的有限时间帧。因此，这些系统必须使用更高级的模式记录状态，在 ASP.NET 中很容易做到这一点。只要使用提供程序，就可以利用比较高级的模式把状态记录到数据存储中。



提供程序是允许对数据存储、进程等进行编程访问的对象。

ASP.NET 中的会话默认以 InProc 方式存储，即存储在运行 ASP.NET 的进程中。在 ASP.NET 中，可以改变用于 Session 对象的提供程序，这会改变存储会话的位置。存储会话信息的提供程序有：

- InProc
- StateServer
- SQLServer

在 InProc 中，可以使用 StateServer 把会话存储在一个进程中，而该进程与运行 ASP.NET 的进程是完全独立的。如果 ASP.NET 进程停止，这就会保护会话进程。还可以使用 SQLServer 选项把会话存储到磁盘(如数据库)上。这个方法可以直接把会话存储在 SQL Server 数据库中。那么，如何改变用于会话的提供程序？这有两种方式。

一种方式是使用如图 14-1 所示的 IIS(Internet Information Services)管理器。另一种方式则是直接进入系统配置文件(如 machine.config 文件)或应用程序配置文件(如 web.config 文件)。在文件中，改变会话状态提供程序的名称，该名称在配置文档的<sessionState>部分使用。

ASP.NET 的以后版本使提供程序模型更进了一步，下面将对此进行讨论。

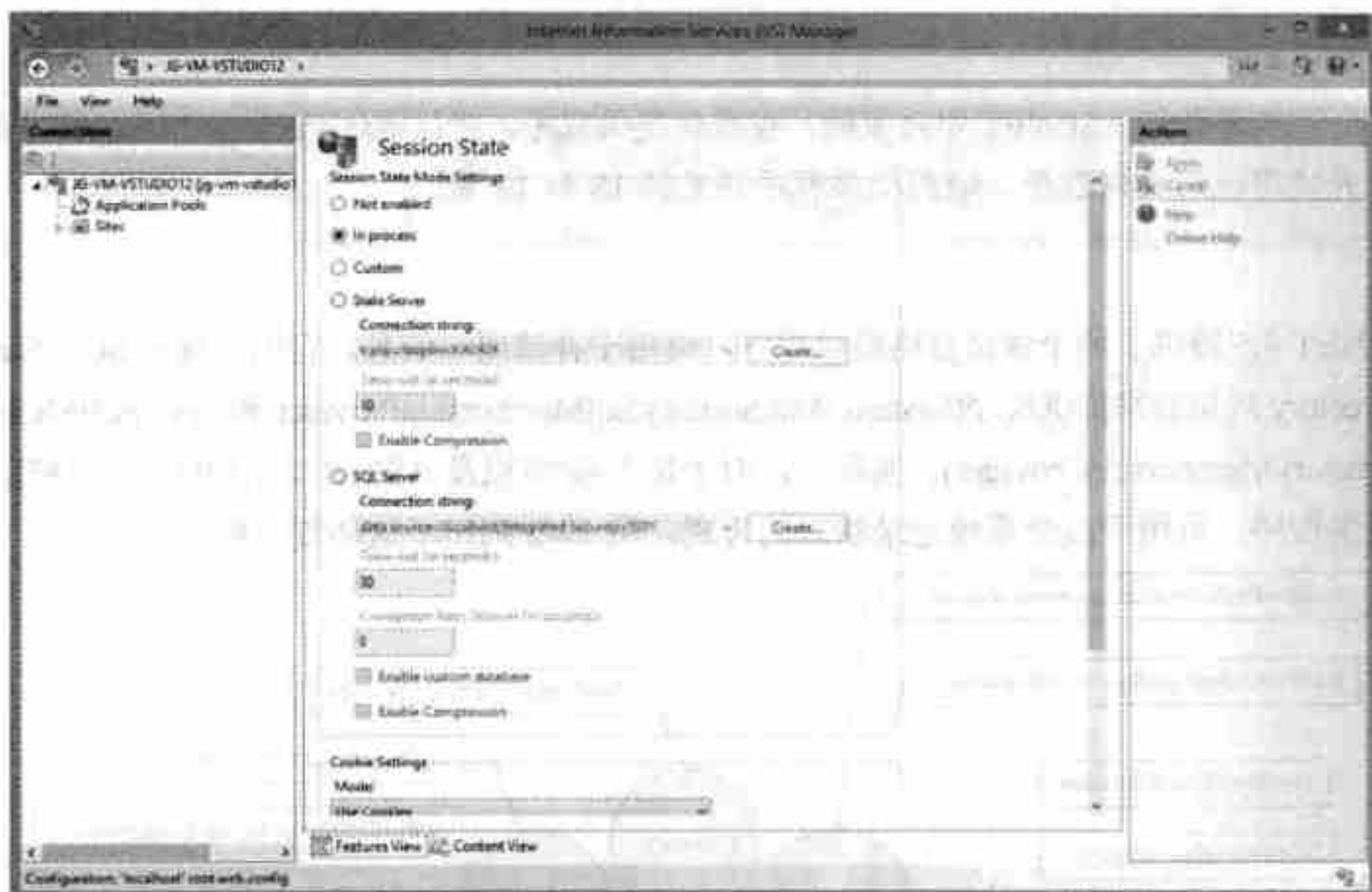


图 14-1

14.2 ASP.NET 4.5 中的提供程序模型

在开发 ASP.NET 时, 用户希望除了 InProc、StateServer 和 SQLServer 这 3 个选项之外, 还能采用其他方式存储会话。例如, 用户需求是添加可以在 Oracle 数据库中存储会话的提供程序。如果开发小组为 Oracle 添加了提供程序, 就要为其他数据库和数据存储方法添加更多的提供程序。因此, 开发小组没有为每个可能的数据存储方法添加提供程序, 而是设计了提供程序模型, 允许用户添加自己希望的提供程序。于是就诞生了 ASP.NET 中新的提供程序模型。

目前 ASP.NET 4.5 引入了许多新系统, 它们都需要某种状态存储方式。另外, 许多新系统要求, 不是使用较脆弱的模式记录状态(这是会话的默认存储方式), 而是把状态存储在比较具体的数据存储中, 如数据库或 XML 文件。这也使访问应用程序的用户的状态有较长的生命周期——这是这些系统的另一个要求。

ASP.NET 4.5 中基于提供程序模型的系统要求有较高级的状态管理, 这些系统包括:

- 成员资格
- 角色管理
- 站点导航
- 个性化
- 健康监控 Web 事件
- 配置文件保护

成员资格系统是允许 ASP.NET 在某种类型的用户存储中创建、删除和编辑应用程序用户的一种方式。显然, 开发人员希望为各种用户群使用不受数量限制的不同数据存储, 因此需要一种简单的方式来改变 ASP.NET 应用程序底层的用户存储。于是, ASP.NET 4.5 中的提供程序模型应运而生。



本章介绍 ASP.NET 中内置的、经典的提供程序，建议通过 Package Manager 下载并使用通用提供程序。通用提供程序详见第 18 和 19 章。

ASP.NET 4.5 提供了两个成员资格提供程序，可用于存储用户信息。它们分别是 SQL Server 和 Active Directory 成员资格提供程序(System.Web.Security.SqlMembershipProvider 和 System.Web.Security.ActiveDirectoryMembershipProvider)。实际上，对于每个系统(以及 ASP.NET 1.x 的一些系统)，都有一系列提供程序，可用于改变系统记录状态的方式。图 14-2 列出了这些提供程序。

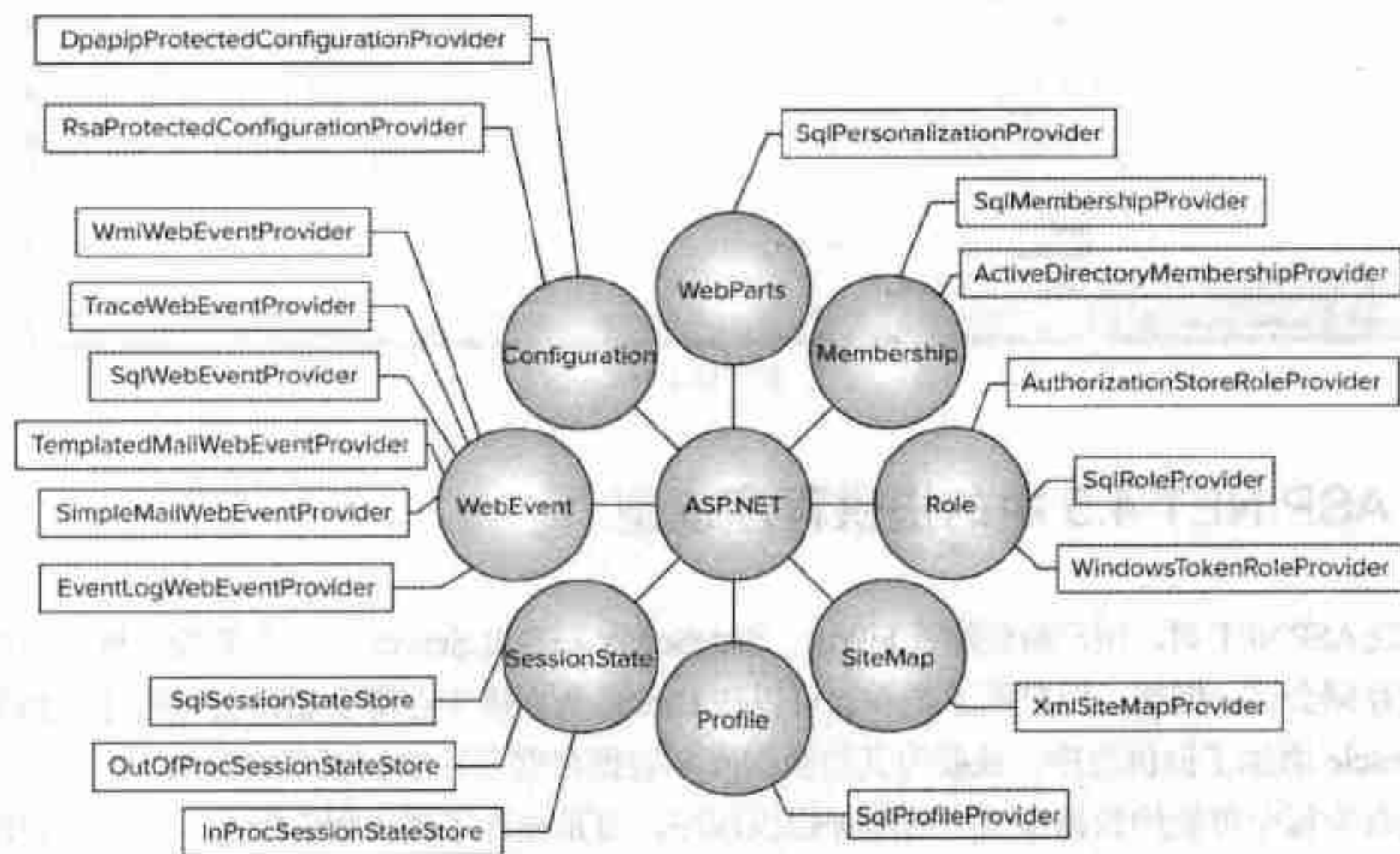


图 14-2

从图 14-2 中可以看出，ASP.NET 提供了大量的提供程序。一些系统只有一个提供程序，如 Profile 系统就只包含一个连接 SQL Server 的提供程序。另一些系统包含多个提供程序，如 WebEvent 系统就包含 6 个不同的提供程序。下面将介绍如何设置 SQL Server，以使用本章列出的多个提供程序。对于许多提供程序(但不是全部)，可以把 SQL Server 2005、2008 或 2012 作为后台数据存储。本章后面将逐个介绍 ASP.NET 4.5 中的提供程序。

14.2.1 设置提供程序以使用 SQL Server 2005、2008 或 2012

许多提供程序都可以使用 SQL Server。例如，成员资格、角色管理、个性化和其他系统都可以很好地使用 SQL Server。但是，在默认情况下，这些系统只能使用 LocalDB，而不能使用比较成熟的 SQL Server 版本，如 SQL Server 2005、2008 或 2012。

为了使用这些数据库，必须使用 aspnet_regsql.exe 工具建立数据库。使用 aspnet_regsql.exe 创建必需的表、角色、存储过程和提供程序需要的其他项。要启动该工具，请选择 Start | All Programs | Microsoft Visual Studio 2012 | Visual Studio Tools | Visual Studio 2012 Command Prompt 命令，打开 Visual Studio 2012 的命令提示符，以访问 ASP.NET SQL Server 安装向导。ASP.NET SQL Server 安装

向导是一个很容易使用的工具，它可以帮助设置 SQL Server，使之用于 ASP.NET 4.5 中的许多系统，例如成员资格、角色管理和个性化系统。该设置向导为创建数据库提供了两种方式：使用命令行工具或使用 GUI 工具。下面首先介绍该工具的命令行版本。

1. ASP.NET SQL Server 安装向导的命令行工具

该设置向导的命令行版本为开发人员提供了数据库创建方式的优化控制。从这个工具的命令行开始使用它不是很困难，不必担心。

如果安装了 Visual Studio 2012，就可以从 Visual Studio Command Prompt 启动该工具。在命令提示符中输入 `aspnet_regsql.exe -?`，就会显示该工具的所有命令行选项的列表。

表 14-1 描述了建立 SQL Server 实例以使用该个性化架构的一些可用选项。

表 14-1

命令选项	说 明
-?	显示可用选项命令的列表
-W	使用向导模式。如果没有使用其他参数，就使用默认安装模式
-S <server>	指定要使用的 SQL Server 实例
-U <login>	指定要登录到 SQL Server 的用户名。如果使用这个选项，也要同时使用 -P 命令
-P <password>	指定用于登录到 SQL Server 的密码。如果使用这个选项，也要同时使用 -U 命令
-E	提供使用当前 Windows 验证凭据的说明
-C	指定连接 SQL Server 的连接字符串。如果使用这个选项，就可以避免使用 -U 和 -P 命令，因为它们已在连接字符串中指定
-A all	对 ASP.NET 提供的所有 SQL Server 操作添加支持，包括成员资格、角色管理、Profile、站点计数器和页面/控件个性化
-A p	添加对使用 Profile 系统的支持
-R all	删除对以前安装的所有 SQL Server 操作的支持，包括成员资格、角色管理、Profile、站点计数器和页面/控件个性化
-R p	从 SQL Server 中删除对 Profile 功能的支持
-d <database>	指定用于应用程序服务的数据库名。如果没有指定数据库名，就使用 aspnetdb
-sqlxportonly <filename>	将这个命令和其他命令结合使用，不会修改 SQL Server 数据库实例，而会生成 SQL 脚本，以添加或删除指定的功能。这个命令会在一个文件中创建脚本，该文件将使用命令中指定的名称作为文件名

为了使用这个命令行工具修改 SQL Server，以使用个性化提供程序，可以输入如下命令：

```
aspnet_regsql.exe -A all -E
```

输入上述命令后，命令行工具就会创建 ASP.NET 4.5 系统所需的所有功能。结果显示在该工具中，如图 14-3 所示。



图 14-3

完成这一操作后, 就可以看到在 SQL Server 2012 的 Microsoft SQL Server Management Studio 中创建了新数据库 aspnetdb(用于本例的数据库)。现在就有了用于所有 ASP.NET 系统的表, 这些系统可以利用 SQL Server, 如图 14-4 所示。

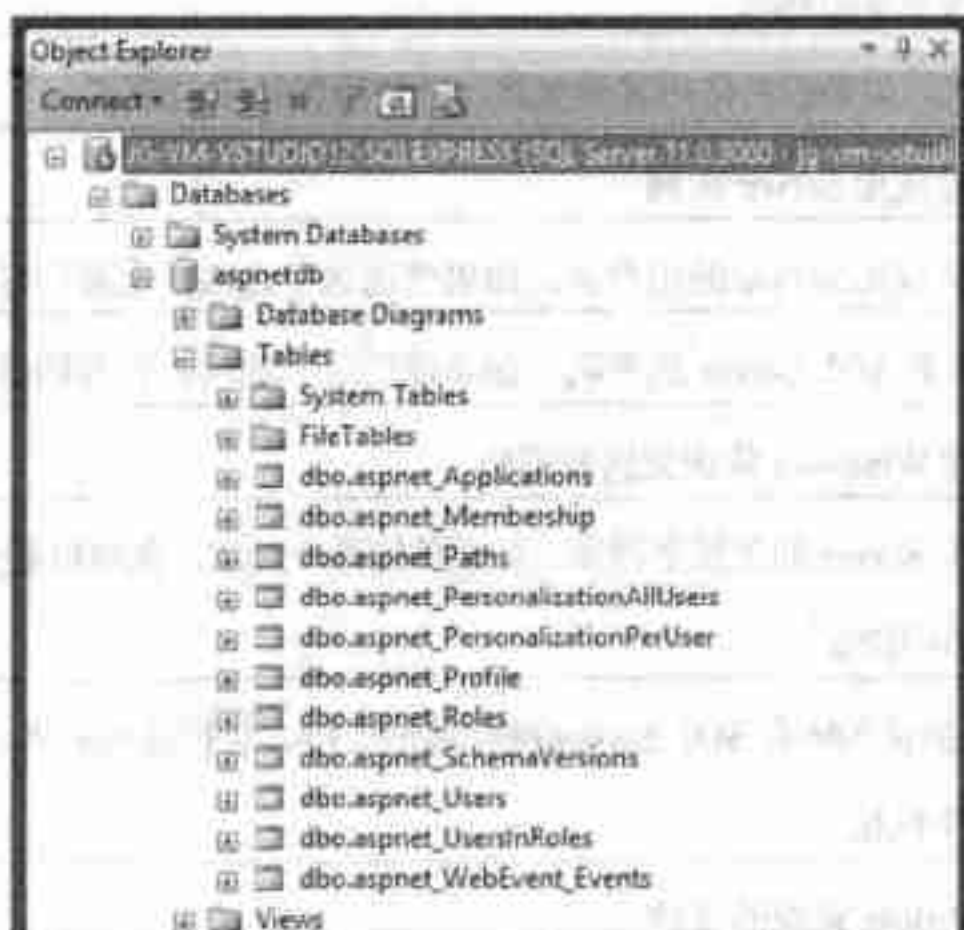


图 14-4

与 ASP.NET SQL Server 安装向导的 GUI 版本相比, 使用命令行工具的一个优点在于, 可以在数据库中只安装自己感兴趣的功能, 而不是像 GUI 版本那样必须安装所有的功能。例如, 如果只有成员资格系统需要与 SQL Server 2012 通信, 而其他系统(如角色管理和个性化系统)不需要, 就可以配置该设置向导, 只在数据库中建立成员资格系统需要的表、角色、存储过程和其他项。如果只为成员资格系统建立数据库, 那么可以在命令行上输入下面的命令:

```
aspnet_regsql.exe -A m -E
```

2. ASP.NET SQL Server 安装向导的 GUI 工具

除了通过命令行使用 ASP.NET SQL Server 安装向导之外, 还可以使用该设置向导的 GUI 版本。要启动 GUI 版本, 可以在 Visual Studio 命令提示符中输入下面的代码:

```
aspnet_regsql.exe
```

这样就会显示 ASP.NET SQL Server 设置向导的欢迎界面，如图 14-5 所示。

单击 Next 按钮，进入一个新的屏幕，其中提供两个选项：在 SQL Server 中安装管理功能和删除这些管理功能，如图 14-6 所示。



图 14-5

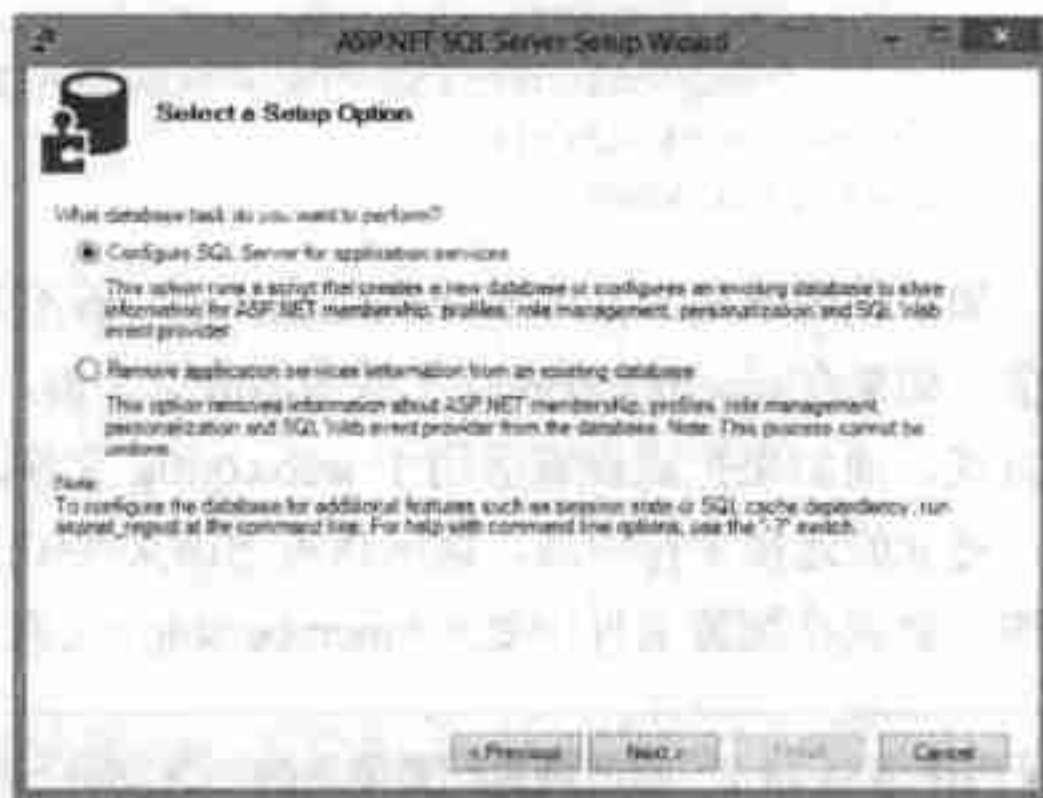


图 14-6

选择 Configure SQL Server for application services 单选按钮，单击 Next 按钮。所出现的屏幕(如图 14-7 所示)要求输入 SQL Server 的登录凭据和数据库名以执行操作。Database 选项是 <default>，表示向导将创建数据库 aspnetdb。如果要选择另一个文件夹，例如应用程序的数据库，可以选择相应的选项。

完成服务器和数据库的选择后，单击 Next 按钮。所出现的屏幕如图 14-8 所示，要求确认前面的设置。如果一切都配置正确，就单击 Next 按钮，否则就单击 Previous 按钮，更正设置。

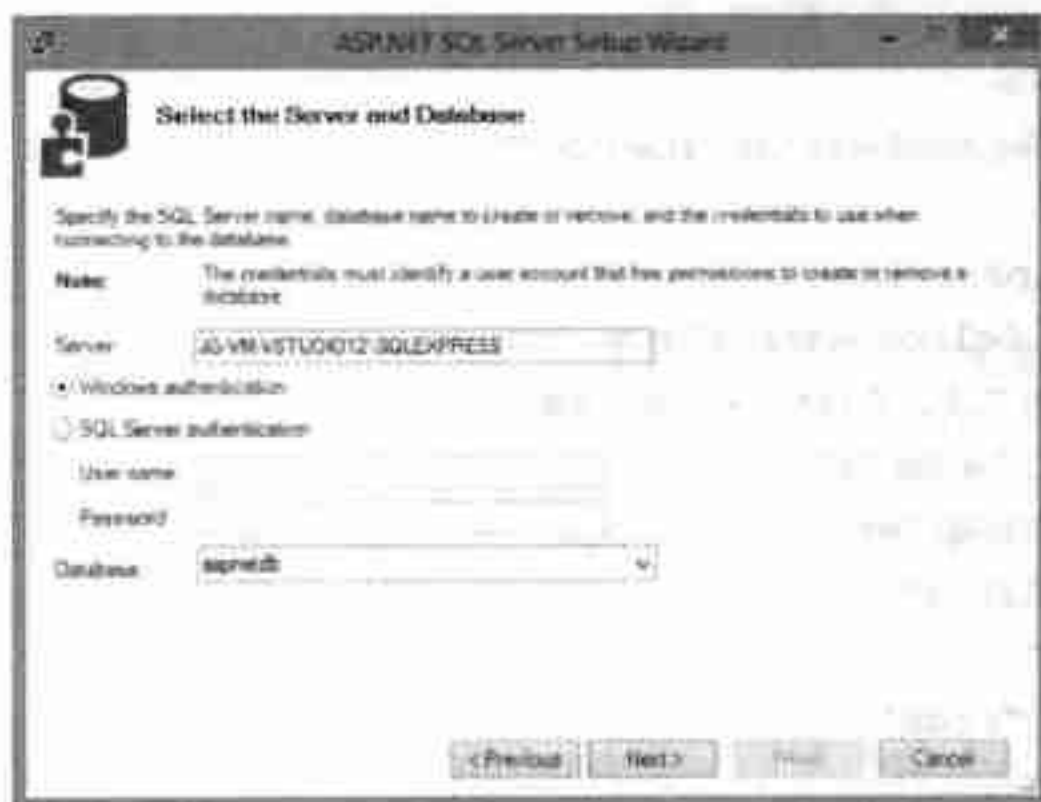


图 14-7

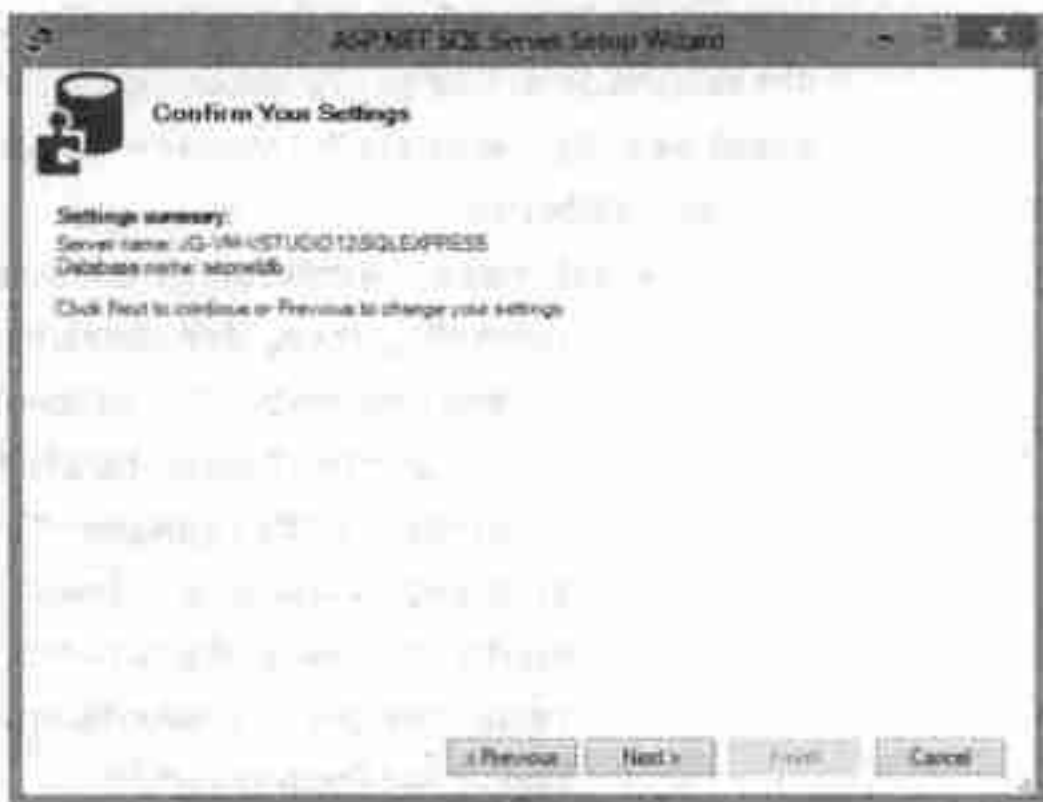


图 14-8

完成该操作后，就会收到一条通知，说明一切都已正确配置。

3. 把默认的提供程序连接到新的 SQL Server 实例

安装了全功能的 SQL Server 以使用 ASP.NET 提供的各个系统后，就在 machine.config 或 web.config 文件中创建了到数据库的连接字符串，如程序清单 14-1 所示。

程序清单 14-1 在 machine.config 或 web.config 文件中修改连接字符串以使用 SQL Server 2012

```
<configuration>
  <connectionStrings>
    <add name="LocalSqlServer"
      connectionString="Data Source=127.0.0.1;Integrated Security=SSPI" />
    </connectionStrings>
  </configuration>
```

如果使用的是 SQL Server 的远程实例，而不是应用程序所在服务器上的实例，就要修改所提供的值。如果在 machine.config 文件中修改这个值，就会改变每个 ASP.NET 应用程序使用该提供程序的方式。而如果把该设置应用于 web.config 文件，就只影响使用该实例的本地应用程序。

建立好连接字符串后，就应该考虑该文件中的 <providers> 部分。例如，如果使用成员资格提供程序，就要在配置文件中处理 <membership> 元素。修改 SQL Server 的设置如程序清单 14-2 所示。

程序清单 14-2 修改通过配置使用的 SQL Server

```
<?xml version="1.0"?>
<configuration>
  <connectionStrings>
    <add name="LocalSqlServer"
      connectionString=
        "Data Source=127.0.0.1;Integrated Security=SSPI;Initial Catalog=aspnetdb;" />
    </connectionStrings>

  <system.web>
    <compilation debug="false" targetFramework="4.5" />
    <httpRuntime targetFramework="4.5" />
    <membership defaultProvider="AspNetSqlMembershipProvider">
      <providers>
        <add name="AspNetSqlMembershipProvider"
          type="System.Web.Security.SqlMembershipProvider,
            System.Web, Version=4.0.0.0, Culture=neutral,
            PublicKeyToken=b03f5f7f11d50a3a"
          connectionStringName="LocalSqlServer"
          enablePasswordRetrieval="false"
          enablePasswordReset="true"
          requiresQuestionAndAnswer="true"
          applicationName="/"
          requiresUniqueEmail="false"
          passwordFormat="Hashed"
          maxInvalidPasswordAttempts="5"
          minRequiredPasswordLength="7"
          minRequiredNonalphanumericCharacters="1"
          passwordAttemptWindow="10"
          passwordStrengthRegularExpression="" />
      </providers>
    </membership>
  </system.web>
</configuration>
```


完成了这些修改后, SQL Server 2012 实例就是一个可用于应用程序的提供程序。这个提供程序实例的名称是 `AspNetSqlMembershipProvider`。该实例还使用了在程序清单 14-1 中定义的连接字符串 `LocalSqlServer`。

注意程序清单 14-2 中一些重要的特性声明。第一个特性声明是成员资格系统使用的提供程序通过 `<membership>` 主节点中的 `defaultProvider` 特性定义。使用这个特性可以指定提供程序是内置的提供程序还是自己建立的定制提供程序,或是从第三方接收的提供程序。有了程序清单 14-2 中的代码,成员资格提供程序就可以使用 SQL Server 2012(如本例所示),而不必使用 SQL Server Express 版本文件。

下面介绍 ASP.NET 4.5 中内置的所有提供程序,先从成员资格提供程序开始介绍。

14.2.2 成员资格提供程序

利用成员资格系统可以方便地管理 ASP.NET 应用程序中的用户。与 ASP.NET 提供的大多数系统一样,成员资格系统也包含一系列服务器控件,可以与定义好的提供程序交互,以获取或记录提供程序在数据库中定义的信息。因为在服务器控件与用于检索和记录数据的数据存储之间存在提供程序,所以可以方便地让控件在另一完全不同的后台工作,只需通过简单修改 ASP.NET 应用程序中的配置来改变整个系统的底层提供程序(这里是成员资格系统)即可。这对服务器控件没有影响。

如前所述, ASP.NET 4.5 提供了两个成员资格提供程序:

- `System.Web.Security.SqlMembershipProvider`: 使用成员资格系统连接 SQL Server 2005/2008/2012 和 SQL Server Express Edition。
- `System.Web.Security.ActiveDirectoryMembershipProvider`: 使用成员资格系统连接 Active Directory(在 Windows Server 中可用)。

这两个成员资格提供程序类继承了 `MembershipProvider` 基类,如图 14-9 所示。

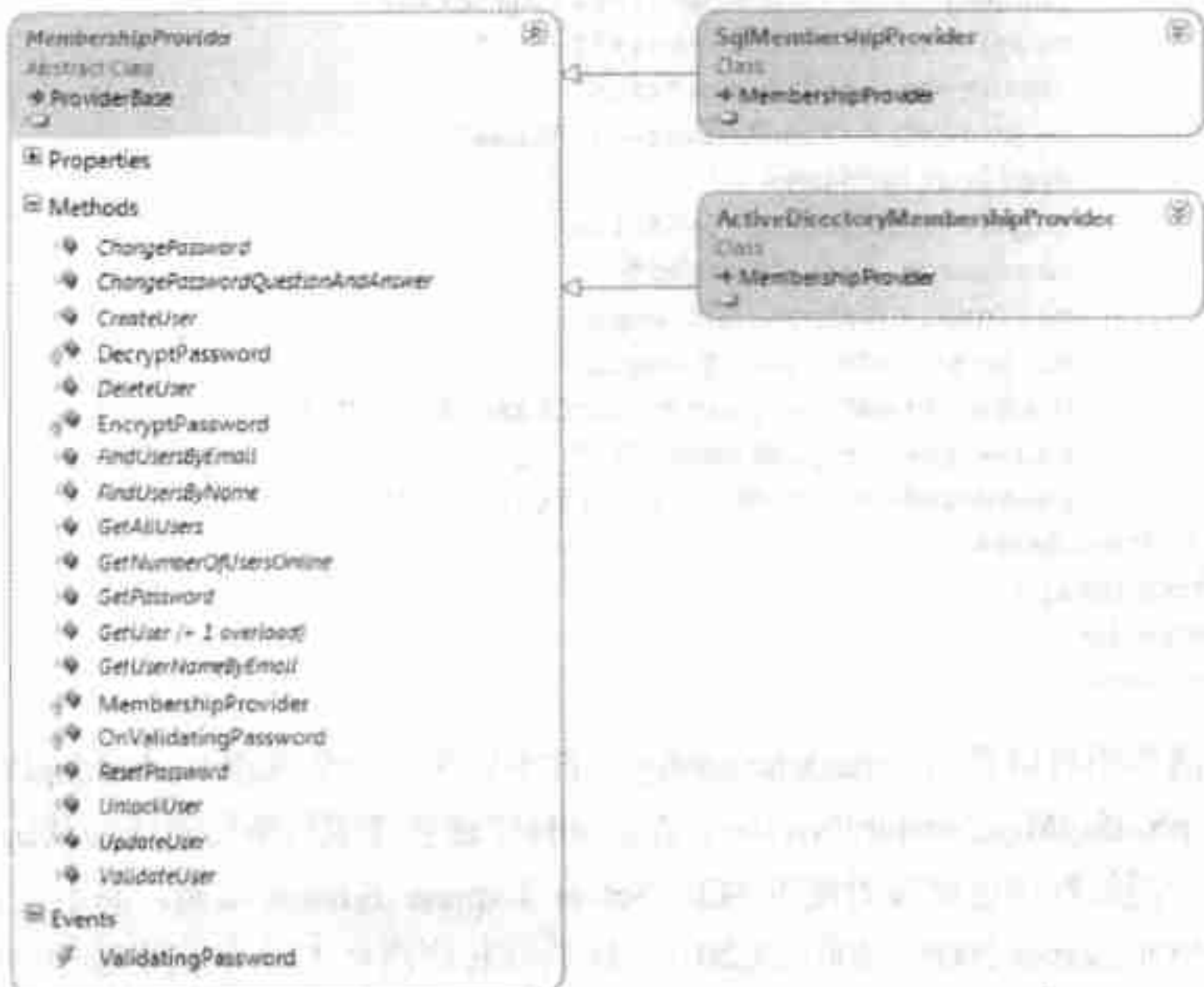


图 14-9



ASP.NET 4.5 带有这两个提供程序，还有另外两个成员资格选项。第一个是 SimpleMembership，目前，它只能用于 ASP.NET MVC 4，不能直接用于 ASP.NET Web 窗体。第二个是使用 DotNetOpenAuth。这个选项与 ASP.NET 4.5 中的默认模板捆绑在一起，可通过 NuGet 使用。NuGet 是一个第三方库。这两个选项详见第 19 章。

下面分别介绍这两个提供程序。

1. System.Web.Security.SqlMembershipProvider

默认提供程序是 SqlMembershipProvider 实例。在应用服务器上，每个 ASP.NET 应用程序的默认声明位于 machine.config 文件中。machine.config 文件位于 C:\Windows\Microsoft.NET\Framework\v4.0.30319\Config 目录下。程序清单 14-3 显示了这个提供程序的定义，该定义包含在 machine.config 文件中。

程序清单 14-3 SqlMembershipProvider 实例的声明

```
<configuration>
  <system.web>
    <membership defaultProvider="AspNetSqlMembershipProvider">
      <providers>
        <add name="AspNetSqlMembershipProvider"
              type="System.Web.Security.SqlMembershipProvider,
                  System.Web, Version=4.0.0.0, Culture=neutral,
                  PublicKeyToken=b03f5f7f11d50a3a"
              connectionStringName="LocalSqlServer"
              enablePasswordRetrieval="false"
              enablePasswordReset="true"
              requiresQuestionAndAnswer="true"
              applicationName="/"
              requiresUniqueEmail="false"
              passwordFormat="Hashed"
              maxInvalidPasswordAttempts="5"
              minRequiredPasswordLength="7"
              minRequiredNonalphanumericCharacters="1"
              passwordAttemptWindow="10"
              passwordStrengthRegularExpression="" />
      </providers>
    </membership>
  </system.web>
</configuration>
```

从这个程序清单中可以看出，machine.config 文件中定义了一个 SqlMembershipProvider 实例，这个实例名为 AspNetSqlMembershipProvider。在该文件中还包含成员资格系统的默认行为设置。在默认情况下，这个提供程序也配置为使用 SQL Server Express Edition 实例，而不是全功能的 SQL Server 版本，如 SQL Server 2005、2008 或 2012。因为在程序清单 14-3 的提供程序声明中已定义了 connectionStringName 特性，并将其设置为 LocalSqlServer。并且还在 machine.config 文件中定义了 LocalSqlServer，如程序清单 14-4 所示。

程序清单 14-4 定义 LocalSqlServer 实例

```

<configuration>
  <connectionStrings>
    <clear />
    <add name="LocalSqlServer"
      connectionString="data source=.\SQLEXPRESS;Integrated
      Security=SSPI;AttachDBFilename=|DataDirectory|aspnetdb.mdf;
      User Instance=true"
      providerName="System.Data.SqlClient" />
    </connectionStrings>
  </configuration>

```

可以看出, 连接字符串信息指明数据源是一个本地 SQL Server Express Edition 文件(.mdf 文件)。当然, SqlMembershipProvider 功能并不是只能使用这种文件类型, 还可以使用 SQL Server 2005、2008 或 2012(如本章前面所示)。

2. System.Web.Security.ActiveDirectoryMembershipProvider

ASP.NET 4.5 提供的成员资格系统还可以将这个系统连接到 Microsoft Active Directory 实例, 甚至连接到 Active Directory Application Mode(ADAM, 这是个独立的目录产品)。因为在 machine.config 文件的 SqlMembershipProvider 中定义了默认的成员资格提供程序, 所以必须在应用程序的 web.config 文件中重写这些设置。

在 web.config 文件中创建 ActiveDirectoryMembershipProvider 的指定实例之前, 必须先定义到 Active Directory 存储的连接字符串, 如程序清单 14-5 所示。

程序清单 14-5 定义到 Active Directory 存储的连接字符串

```

<configuration>
  <connectionStrings>
    <add name="ADConnectionString"
      connectionString=
      "LDAP://domain.myAdServer.com/CN=Users,DC=domain,DC=testing,DC=com" />
    </connectionStrings>
  </configuration>

```

有了这个连接字符串, 就可以在 web.config 文件中创建 ActiveDirectoryMembershipProvider 实例, 把它关联到这个连接字符串, 如程序清单 14-6 所示。

程序清单 14-6 定义 ActiveDirectoryMembershipProvider 实例

```

<configuration>
  <connectionStrings>
    <add name="ADConnectionString"
      connectionString=
      "LDAP://domain.myAdServer.com/CN=Users,DC=domain,DC=testing,DC=com" />
    </connectionStrings>
  <system.web>
    <membership
      defaultProvider="AspNetActiveDirectoryMembershipProvider">
    <providers>

```



```

    <add name="AspNetActiveDirectoryMembershipProvider"
        type="System.Web.Security.ActiveDirectoryMembershipProvider,
        System.Web, Version=1.0.3600, Culture=neutral,
        PublicKeyToken=b03f5f7f11d50a3a"
        connectionStringName="ADConnectionString"
        connectionUsername="UserWithAppropriateRights"
        connectionPassword="PasswordForUser"
        connectionProtection="Secure"
        enablePasswordReset="true"
        enableSearchMethods="true"
        requiresQuestionAndAnswer="true"
        applicationName="/"
        description="Default AD connection"
        requiresUniqueEmail="false"
        clientSearchTimeout="30"
        serverSearchTimeout="30"
        attributeMapPasswordQuestion="department"
        attributeMapPasswordAnswer="division"
        attributeMapFailedPasswordAnswerCount="singleIntAttribute"
        attributeMapFailedPasswordAnswerTime="singleLargeIntAttribute"
        attributeMapFailedPasswordAnswerLockoutTime="singleLargeIntAttribute"
        attributeMapEmail = "mail"
        attributeMapUsername = "userPrincipalName"
        maxInvalidPasswordAttempts = "5"
        passwordAttemptWindow = "10"
        passwordAnswerAttemptLockoutDuration = "30"
        minRequiredPasswordLength="7"
        minRequiredNonalphanumericCharacters="1"
        passwordStrengthRegularExpression=
            "(?=.{6,}) (?=.*\d) (?=.*\W) (?=.{1,})" />
    </providers>
</membership>
</system.web>
</configuration>

```

上述程序清单中的特性并不都是必需的，这个程序清单只提供了 `ActiveDirectoryMembershipProvider` 的一些可用特性。实际上，可以使用最简单的形式声明这个实例，如下所示：

```

<membership
  defaultProvider="AspNetActiveDirectoryMembershipProvider">
  <providers>
    <add name="AspNetActiveDirectoryMembershipProvider"
        type="System.Web.Security.ActiveDirectoryMembershipProvider,
        System.Web, Version=1.0.3600, Culture=neutral,
        PublicKeyToken=b03f5f7f11d50a3a"
        connectionStringName="ADConnectionString" />
  </providers>
</membership>

```

只要有了 `SqlMembershipProvider` 或 `ActiveDirectoryMembershipProvider`，在配置好成员资格系统服务器控件(如 `Login` 服务器控件)和成员资格 API 后，就可以通过建立的提供程序记录和检索它们的信息。这就是 ASP.NET 小组建立的提供程序模型的强大功能所在。本章后面介绍其他提供程序时，还将继续探讨提供程序模型的功能。

14.2.3 角色提供程序

用户(使用 ASP.NET 成员资格系统)登录到系统之后, ASP.NET 角色管理系统允许利用用户所属的角色, 给他授予整个应用程序的某访问权限。与其他系统一样, ASP.NET 4.5 中的角色管理系统也提供了一系列提供程序, 可以使用简单的方式存储和检索角色信息。当然, 这并不意味着只能使用角色管理系统中的 3 个提供程序之一。可以扩展某个已建立的提供程序, 甚至可以创建自己的定制提供程序。

ASP.NET 4.5 默认为角色管理系统提供了如下 3 个提供程序:

- **System.Web.Security.SqlRoleProvider**: 使用 ASP.NET 角色管理系统连接 SQL Server 2005/2008/2012 以及 Microsoft SQL Server Express Edition。
- **System.Web.Security.WindowsTokenRoleProvider**: 连接 ASP.NET 角色管理系统和内置的 Windows 安全组系统。
- **System.Web.Security.AuthorizationStoreRoleProvider**: 连接 ASP.NET 角色管理系统和 XML 文件、Active Directory 和 Active Directory Application Mode (ADAM) 存储。

角色管理的这 3 个类都继承 RoleProvider 基类, 如图 14-10 所示。

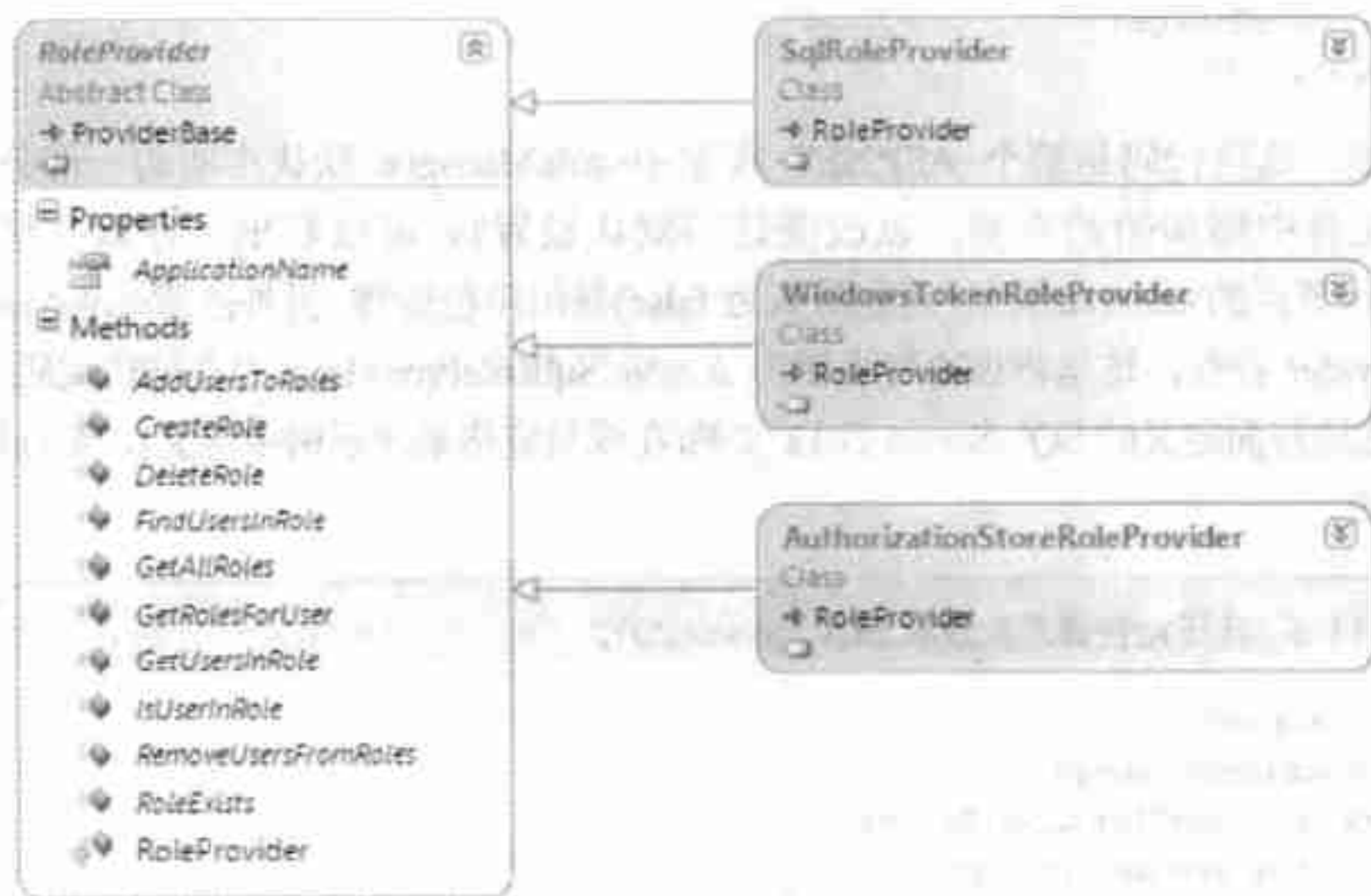


图 14-10

1. System.Web.Security.SqlRoleProvider

ASP.NET 中的角色管理系统默认使用 SQL Server Express Edition 文件, 这一点与成员资格系统一样。对 SQL Server Express Edition 文件的连接使用 SqlRoleProvider, 但很容易配置 SQL Server 2005、2008 或 2012 服务器, 以通过 SqlRoleProvider 使用角色管理系统。本章开始部分介绍了建立全功能的 SQL Server 的过程。

在 machine.config.comments 文件中查看 SqlRoleProvider 实例, 应注意程序清单 14-7 中定义的语法。machine.config.comments 文件提供了 machine.config 文档, 并列出了 ASP.NET 框架中默认设置的细节。

程序清单 14-7 SqlRoleProvider 实例的声明

```

<configuration>
  <roleManager enabled="false" cacheRolesInCookie="false"
    cookieName=".ASPXROLES" cookieTimeout="30" cookiePath="/"
    cookieRequireSSL="false" cookieSlidingExpiration="true"
    cookieProtection="All" defaultProvider="AspNetSqlRoleProvider"
    createPersistentCookie="false" maxCachedResults="25">
    <providers>
      <clear />
      <add connectionStringName="LocalSqlServer" applicationName="/"
        name="AspNetSqlRoleProvider"
        type="System.Web.Security.SqlRoleProvider, System.Web,
        Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
        />
      <add applicationName="/" name="AspNetWindowsTokenRoleProvider"
        type="System.Web.Security.WindowsTokenRoleProvider, System.Web,
        Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
        />
    </providers>
  </roleManager>
</configuration>

```

如前所述,这段代码是整个 ASP.NET 框架中<roleManager>默认声明的一部分(注意可以在 web.config 文件中添加新的声明,以改变这些默认设置)。可以看出,在默认情况下,通过<roleManager>节点的 enabled 特性(其值默认为 false)禁用角色管理。另外注意<roleManager>元素中的 defaultProvider 特性,这里将该特性设置为 AspNetSqlRoleProvider。也在这段代码中定义这个提供程序。要连接前面定义的 SQL Server 2012 实例(在成员资格系统示例中定义),可以使用程序清单 14-8 中的语法。

程序清单 14-8 连接角色管理系统和 SQL Server 2012

```

<configuration>
  <connectionStrings>
    <add name="LocalSqlServer"
      connectionString=
        "Data Source=127.0.0.1;Integrated Security=SSPI;Initial Catalog=aspnetdb;" />
  </connectionStrings>
  <system.web>
    <roleManager enabled="true" cacheRolesInCookie="true"
      cookieName=".ASPXROLES" cookieTimeout="30" cookiePath="/"
      cookieRequireSSL="false" cookieSlidingExpiration="true"
      cookieProtection="All" defaultProvider="AspNetSqlRoleProvider"
      createPersistentCookie="false" maxCachedResults="25">
      <providers>
        <clear />
        <add connectionStringName="LocalSqlServer" applicationName="/"
          name="AspNetSqlRoleProvider"
          type="System.Web.Security.SqlRoleProvider, System.Web,
          Version=4.0.0.0, Culture=neutral,
          PublicKeyToken=b03f5f7f11d50a3a" />
      </providers>
    </roleManager>
  </system.web>
</configuration>

```



```

        </roleManager>
    </system.web>
</configuration>

```

有了这些代码，就可以连接 SQL Server 2012。下面讨论角色管理系统的第二个提供程序。

2. System.Web.Security.WindowsTokenRoleProvider

Windows 操作系统有内置的角色系统。在使用基于内联网的应用程序(所有的用户都指定了已定义好的角色)时，使用 Windows 安全组系统是非常理想的方式。当然，如果关闭 ASP.NET 应用程序的匿名验证功能，把应用程序配置为使用 Windows Authentication，Windows 安全组系统的性能将达到最佳。



有关 ASP.NET 应用程序的 Windows Authentication 的论述，详见第 20 章。

在使用 WindowsTokenRoleProvider 时有一些限制。它是一个只读的提供程序，因为 ASP.NET 不允许修改在 Windows 安全组系统中应用的设置。也就是说，在使用这个提供程序时，通过 RoleProvider 抽象类提供的所有方法并不都是可用的。在 WindowsTokenRoleProvider 类中，只能使用 IsUserInRole 和 GetUsersInRole 方法。

要配置 WindowsTokenRoleProvider 实例，可以使用程序清单 14-9 中定义的语法。

程序清单 14-9 WindowsTokenRoleProvider 实例

```

<configuration>
  <system.web>
    <authentication mode="Windows" />
    <roleManager defaultProvider="WindowsProvider"
      enabled="true" cacheRolesInCookie="false">
      <providers>
        <add name="WindowsProvider"
          type="System.Web.Security.WindowsTokenRoleProvider" />
      </providers>
    </roleManager>
  </system.web>
</configuration>

```

必须在 <roleManager> 元素中使用 defaultProvider 特性声明默认的提供程序，才能修改 SqlRoleProvider 关联中的指定提供程序。

3. System.Web.Security.AuthorizationStoreRoleProvider

在 ASP.NET 的默认安装中，最后一个角色提供程序是 AuthorizationStoreRoleProvider。这个角色提供程序类可以在 Authorization Manager 策略存储中存储角色，这种存储也称为 AzMan 存储。与 WindowsTokenRoleProvider 一样，AuthorizationStoreRoleProvider 的使用也存在一些限制，因为它不支持任何 AzMan 业务规则。

要使用 AuthorizationStoreRoleProvider，必须先在 web.config 文件中建立一个连接，以连接 AzMan

使用的 XML 数据存储, 如程序清单 14-10 所示。

程序清单 14-10 建立到 AzMan 策略存储的连接

```
<configuration>
  <connectionStrings>
    <add name="LocalPolicyStore"
      connectionString="msxml://~\App_Data\SampleStore.xml" />
  </connectionStrings>
</configuration>
```

注意在使用这些基于 XML 的策略文件时, 最好把它们存储在 App_Data 文件夹中。存储在 App_Data 文件夹中的文件不能在浏览器中删除。

建立了连接字符串后, 下一步就是配置 AuthorizationStoreRoleProvider 实例, 这需要使用程序清单 14-14 中定义的语法。

程序清单 14-11 定义 AuthorizationStoreRoleProvider 实例

```
<configuration>
  <connectionStrings>
    <add name="MyLocalPolicyStore"
      connectionString="msxml://~\App_Data\datafilename.xml" />
  </connectionStrings>
  <system.web>
    <authentication mode="Windows" />
    <identity impersonate="true" />
    <roleManager defaultProvider="AuthorizationStoreRoleProvider"
      enabled="true"
      cacheRolesInCookie="true"
      cookieName=".ASPROLES"
      cookieTimeout="30"
      cookiePath="/"
      cookieRequireSSL="false"
      cookieSlidingExpiration="true"
      cookieProtection="All" >
      <providers>
        <clear />
        <add name="AuthorizationStoreRoleProvider"
          type="System.Web.Security.AuthorizationStoreRoleProvider"
          connectionStringName="MyLocalPolicyStore"
          applicationName="SampleApplication"
          cacheRefreshInterval="60"
          scopeName="" />
      </providers>
    </roleManager>
  </system.web>
</configuration>
```

下面介绍 ASP.NET 的个性化提供程序。

14.2.4 个性化提供程序

与 ASP.NET 中的成员资格系统一样, 个性化系统(也称为 Profile 系统)是另一个基于提供程序模

型的系统。这个系统在查看应用程序的终端用户和为该用户集中存储的数据之间建立关联。如前所述，这些个性化属性以每个用户为基础进行存储和维护。ASP.NET 为数据的存储提供了提供程序 `System.Web.Profile.SqlProfileProvider`，它使用 ASP.NET 个性化系统连接 SQL Server 2005/2008/2012 以及 Microsoft SQL Server Express Edition。

个性化系统类继承了 `ProfileProvider` 基类，如图 14-14 所示。

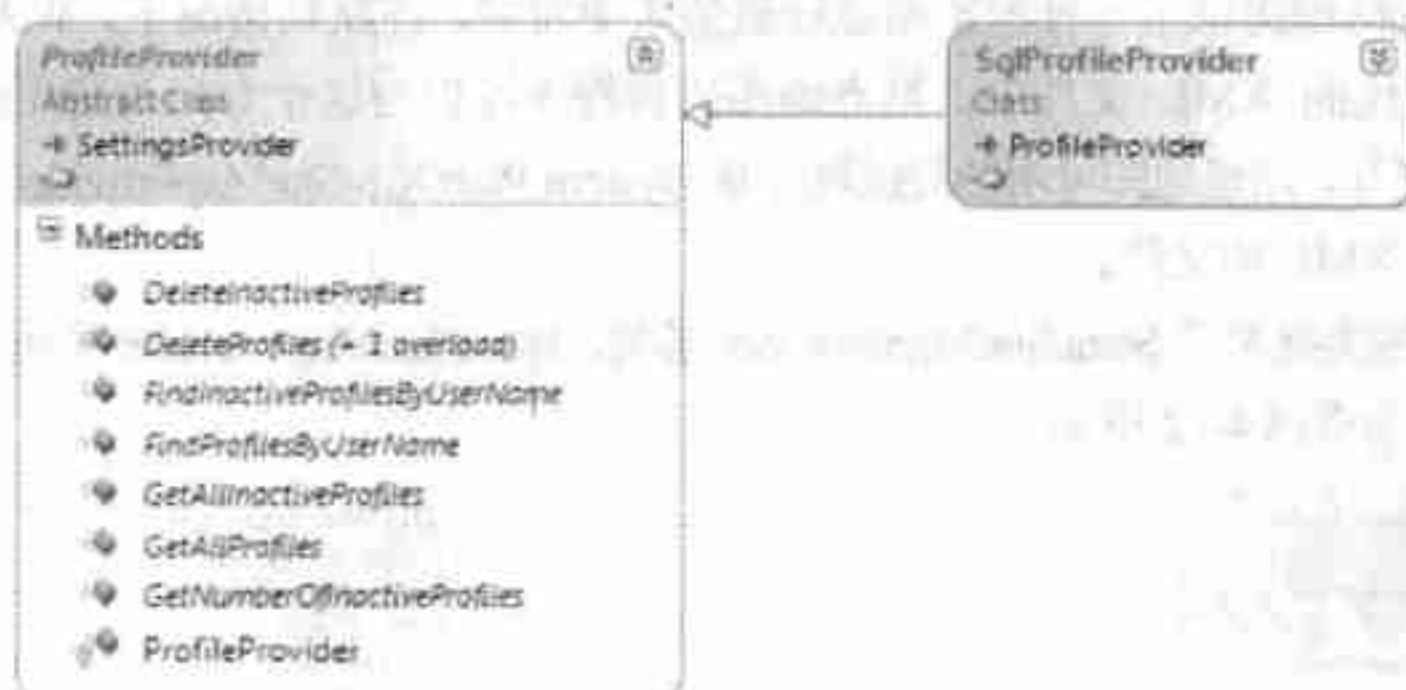


图 14-11

与前面介绍的其他提供程序一样，`SqlProfileProvider` 默认连接的是 Microsoft SQL Server Express Edition 文件。但是，可以修改连接以使用 SQL Server 2005、2008 或 2012。例如，如果连接的是 SQL Server 2012 数据库，可以在 `web.config` 文件中定义连接，把 `SqlProfileProvider` 声明关联到这个连接字符串，如程序清单 14-12 所示。

程序清单 14-12 连接 `SqlProfileProvider` 和 SQL Server 2012

```

<configuration>
  <connectionStrings>
    <add name="LocalSqlServer"
      connectionString="Data Source=127.0.0.1;Integrated Security=SSPI" />
  </connectionStrings>
  <system.web>
    <profile>
      <providers>
        <clear />
        <add name="AspNetSqlProfileProvider"
          connectionStringName="LocalSqlServer" applicationName="/"
          type="System.Web.Profile.SqlProfileProvider, System.Web,
            Version=4.0.0.0, Culture=neutral,
            PublicKeyToken=b03f5f7f11d50a3a" />
      </providers>
      <properties>
        <add name="FirstName" />
        <add name="LastName" />
        <add name="LastVisited" />
        <add name="Age" />
        <add name="Member" />
      </properties>
    </profile>
  </system.web>
</configuration>
  
```


要在 SQL Server 数据库中存储配置文件信息，必须配置这个数据库以创建相应的表、存储过程和其他项。本章前面已论述过这项任务。

14.2.5 站点地图提供程序

与刚才讨论的个性化提供程序一样，ASP.NET 4.5 也为使用站点地图提供了一个提供程序。ASP.NET 使用站点地图提供了一种维护站点导航的集中方式。在默认情况下，Web 应用程序的导航定义位于一个结构化的 XML 文件中。站点地图提供程序可以与这个专门为应用程序创建的 XML 文件.sitemap 进行交互。站点地图可用的提供程序是 `System.Web.XmlSiteMapProvider`，它使用 ASP.NET 导航系统连接基于 XML 的文件。

站点地图系统的类继承了 `StaticSiteMapProvider` 基类，`StaticSiteMapProvider` 基类是 `SiteMapProvider` 基类的部分实现，如图 14-12 所示。



图 14-12

这是第一个默认不连接 SQL Server 数据库的提供程序。该提供程序使用的是一个静态的 XML 文件。这个 XML 文件采用一种特殊的模式，详见第 17 章。

配置 `XmlSiteMapProvider` 的代码如程序清单 14-13 所示。

程序清单 14-13 在 web.config 文件中定义 `XmlSiteMapProvider` 实例

```

<configuration>
  <system.web>
    <siteMap defaultProvider="MyXmlSiteMapProvider" enabled="true">
      <providers>
        <add name="MyXmlSiteMapProvider"
          description="SiteMap provider that reads in .sitemap files."
        />
      />
    />
  />
</configuration>
  
```

```

type="System.Web.XmlSiteMapProvider, System.Web, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
siteMapFile="AnotherWeb.sitemap" />
</providers>
</siteMap>
</system.web>
</configuration>

```

在严格设计的 web.sitemap 文件中, XmlSiteMapProvider 只允许指定单个根元素。XML 文件查找的默认文件名是 web.sitemap, 但在 web.config 的提供程序声明中, 使用 siteMapFile 特性可以改变这个默认设置, 如程序清单 14-13 所示。

14.2.6 会话状态提供程序

为用户存储会话状态的可用模式包括 InProc、StateServer、SQLServer, 甚至 Custom。每种模式都有其优缺点。在确定使用哪个会话状态模式时, 应全盘考虑每个选项。



有关这些会话状态模式的信息, 详见第 21 章。

这个提供程序模型与本章前面讨论的提供程序模型有点不同。SessionStateModule 类是处理程序, 用于加载一种会话状态模式。会话状态模式的定义如下:

- **System.Web.SessionState.InProcSessionStateStore:** 在与 ASP.NET 工作者进程相同的进程中存储会话。这是执行会话状态管理的最佳方法。
- **System.Web.SessionState.OutOfProcSessionStateStore:** 在与 ASP.NET 工作者进程不同的另一个进程中存储会话。该模式比较安全, 但性能不如 InProc 模式。
- **System.Web.SessionState.SqlSessionStateStore:** 在 SQL Server 中存储会话。这是存储会话最安全的方法, 但也是 3 种模式中性能最差的方法。

这 3 种会话状态管理模式如图 14-13 所示。



图 14-13

接下来将逐个介绍这 3 种能够在 ASP.NET 4.5 应用程序中使用的模式。

1. System.Web.SessionState.InProcSessionStateStore

InProcSessionStateStore 是 ASP.NET 的默认模式。在这种模式下,生成的会话保存在 ASP.NET 工作者进程使用的相同进程中(aspnet_wp.exe 或 w3wp.exe)。这种模式的性能最佳,但也存在一些问题。因为会话存储在同 一个进程中,只要工作者进程再循环,就会破坏所有的会话。工作者进程再循环有许多原因,例如修改了 web.config 文件、Global.asax 文件,或者修改了 IIS 中需要在指定时间过后再循环进程的设置。

在 web.config 文件中配置使用 InProc 模式的代码如程序清单 14-14 所示。

程序清单 14-14 在 web.config 文件中为会话状态管理定义 InProc 模式

```
<configuration>
  <system.web>
    <sessionState mode="InProc">
    </sessionState>
  </system.web>
</configuration>
```

可以看出,这种模式相当简单。下一个要讨论的方法是进程外模式,也称为 StateServer 模式。

2. System.Web.SessionState.OutOfProcSessionStateStore

除了 InProc 模式之外,StateServer 模式是存储会话状态的进程外方法。这个方法的性能不像把会话存储在 ASP.NET 工作者进程中的模式那么好,因为该方法在进程之间跳转,以处理当前使用的进程。尽管性能不如 InProc 模式,但使用 OutOfProcSessionStateStore 比使用 InProcSessionStateStore 运行会话更可靠。即使应用程序的工作者进程需要再循环,该应用程序所使用的会话也仍然会保留。这对于主要依靠会话的应用程序来说至关重要。

使用 OutOfProcSessionStateStore 方法的示例如程序清单 14-15 所示。

程序清单 14-15 使用 OutOfProcSessionStateStore 方法在进程外运行会话

```
<configuration>
  <system.web>
    <sessionState mode="StateServer"
      stateConnectionString="tcpip=127.0.0.1:42424">
    </sessionState>
  </system.web>
</configuration>
```

在使用 StateServer 模式时,还必须使用 stateConnectionString 特性定义存储会话的位置。在这个例子中,使用了本地服务器,也就是说,会话存储在同一台计算机上,但存储在另一个完全不同的进程中。也可以把该特性的值设置为某个 IP 地址,将会话存储在另一台服务器上。除了 IP 地址之外,注意还使用了端口 42424,这是当会话使用 StateServer 模式时所需要的端口。有关改变 StateServer 的端口的讨论,详见第 21 章。

3. System.Web.SessionState.SqlSessionStateStore

在 ASP.NET 中,会话状态管理的最后一个提供程序是 SqlSessionStateStore。这肯定是 3 种模式

中最有弹性的一种，但也是 3 种模式中性能最差的一种。如果使用这种会话状态存储方式，就必须相应建立数据库。第 21 章介绍了建立数据库的方法。

要把应用程序配置为使用 `SqlSessionStateStore`，就必须配置 `web.config` 文件，如程序清单 14-16 所示。

程序清单 14-16 在 `web.config` 文件中定义 `SqlSessionStateStore`

```
<configuration>
  <system.web>
    <sessionState mode="SQLServer"
      allowCustomSqlDatabase="true"
      sqlConnectionString="Data Source=127.0.0.1;
      database=MyCustomASPStateDatabase;Integrated Security=SSPI">
    </sessionState>
  </system.web>
</configuration>
```

下面探讨用于 Web 事件体系结构的提供程序。

14.2.7 Web 事件提供程序

在 ASP.NET 4.5 提供的所有系统中，健康监控系统的提供程序比其他任何系统都多。健康监控系统允许 ASP.NET 应用程序管理员评估正在运行的 ASP.NET 应用程序的健康状况，捕获事件(错误和其他可能的触发器)，再通过某个提供程序存储它们。这些事件称为 Web 事件。通过健康监控系统可以监控许多事件，也就是说，可以记录验证失败/成功、生成的所有错误、ASP.NET 工作者进程的信息、请求数据、响应数据等。记录项就是指使用某个提供程序把数据记录到某类数据库中。

在默认情况下，ASP.NET 4.5 为健康监控系统提供了 7 个提供程序，这比其他任何 ASP.NET 系统都多。这些提供程序的定义如下：

- `System.Web.Management.EventLogWebEventProvider`：使用 ASP.NET 的健康监控系统在 Windows 事件日志中记录安全操作错误和其他错误。
- `System.Web.Management.SimpleMailWebEventProvider`：使用 ASP.NET 的健康监控系统，通过电子邮件发送错误信息。
- `System.Web.Management.TemplatedMailWebEventProvider`：类似于 `SimpleMailWebEventProvider`，使用 `TemplatedMailWebEventProvider` 可以在模板化的电子邮件中发送错误信息。使用标准的 `.aspx` 页面定义模板。
- `System.Web.Management.SqlWebEventProvider`：使用 ASP.NET 的健康监控系统在 SQL Server 中存储错误信息。与 ASP.NET 中其他系统的 SQL 提供程序一样，`SqlWebEventProvider` 也默认在 SQL Server Express Edition 中存储错误信息。
- `System.Web.Management.TraceWebEventProvider`：使用 ASP.NET 的健康监控系统把错误信息发送给 ASP.NET 页面跟踪系统。
- `System.Web.Management.IISTraceWebEventProvider`：使用 ASP.NET 的健康监控系统把错误信息发送给 IIS 跟踪系统。
- `System.Web.Management.WmiWebEventProvider`：连接 ASP.NET 的健康监控系统和 Windows Management Instrumentation (WMI) 事件提供程序。

ASP.NET 健康监控系统的这 7 个提供程序继承 WebEventProvider 或 BufferedEventProvider 基类 (BufferedEventProvider 又继承 WebEventProvider)，如图 14-14 所示。

WebEventProvider 类与 BufferedEventProvider 类有什么区别？最大的区别是 WebEventProvider 类在事件发生时写入事件，而 BufferedEventProvider 类则一直保存事件，直到收集了一定数量的事件后才把它们写入数据库，或者放在电子邮件中成批发送。如果使用 SqlWebEventProvider 类，就进行这种成批处理，而不是在每个 Web 事件发生时都让提供程序建立对数据库的连接，并写入事件。

下面将逐个探讨健康监控系统的这 7 个提供程序。

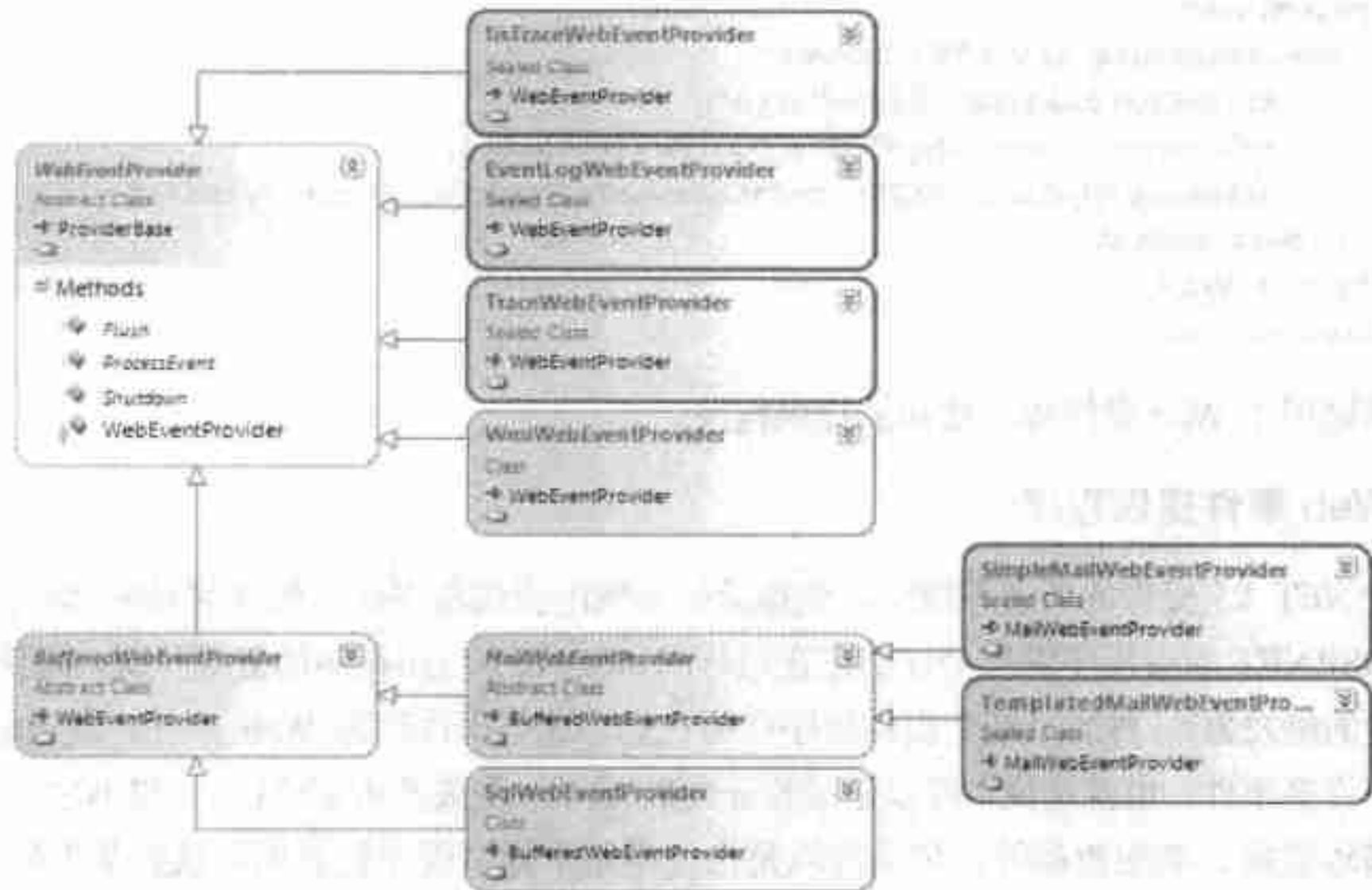


图 14-14

1. System.Web.Management.EventLogWebEventProvider

传统上，管理员和开发人员都在内置的 Windows 事件日志中记录和查看系统错误和应用程序错误。可以通过 Event Viewer 查看事件日志中的项。在 Control Panel 窗口中选择 Administration Tools 选项，再选择 Event Viewer 选项，即可打开这个基于 GUI 的事件查看工具。

在默认情况下，健康监控系统使用 Windows 事件日志记录已在服务器的配置文件中指定的项，或者在应用程序的 web.config 文件中指定的项。如果在服务器上安装了 .NET Framework，在 CONFIG 文件夹的 web.config.comments 文件中就会看到 EventLogWebEventProvider，代码如程序清单 14-17 所示。

程序清单 14-17 在 web.config.comments 文件中声明的 EventLogWebEventProvider

```
<configuration>
  <system.web>
    <healthMonitoring heartbeatInterval="0" enabled="true">
      <bufferModes>
        <!-- Removed for clarity -->
      </bufferModes>
      <providers>
        <clear />
      </providers>
    </healthMonitoring>
  </system.web>
</configuration>
```



```

    <add name="EventLogProvider"
        type="System.Web.Management.EventLogWebEventProvider,
        System.Web, Version=4.0.0.0, Culture=neutral,
        PublicKeyToken=b03f5f7f11d50a3a" />
    <!-- Removed for clarity -->
</providers>
<profiles>
    <!-- Removed for clarity -->
</profiles>
<rules>
    <add name="All Errors Default" eventName="All Errors"
        provider="EventLogProvider" profile="Default" minInstances="1"
        maxLimit="Infinite" minInterval="00:01:00" custom="" />
    <add name="Failure Audits Default" eventName="Failure Audits"
        provider="EventLogProvider" profile="Default" minInstances="1"
        maxLimit="Infinite" minInterval="00:01:00" custom="" />
</rules>
<eventMappings>
    <!-- Removed for clarity -->
</eventMappings>
</healthMonitoring>
</system.web>
</configuration>

```

从程序清单 14-17 中可以看出，在健康监控系统中可以应用许多设置。根据定义好的规则和事件映射，这些项会记录到保存应用程序的服务器的日志中。仔细查看程序清单中的<rules>部分，它指定了要监控的错误类型。在这个部分，应利用健康监控系统来捕获两类错误：All Errors Default 和 Failure Audits Default。

如果触发<rules>部分定义的一个错误，并且该错误被健康监控系统捕获，就会记录它。记录该错误的位置取决于指定的提供程序。在<rules>部分的<add>元素中使用的提供程序特性确定该提供程序。在程序清单 14-17 的例子中，EventLogProvider 是指定的提供程序。也就是说，使用 Windows 事件日志记录这两类错误。



在使用其他提供程序时，应注意健康监控系统的操作与本章前面介绍的其他系统不同。使用 ASP.NET 4.5 的健康监控系统可以一次指定多个提供程序。也就是说，可以在 web.config 文件中指定，错误不仅记录到 Windows 事件日志中，还可以通过其他指定的提供程序记录到其他数据存储中。例如，对于相同的 Web 事件类型，可以把要记录的 Web 事件同时赋予 Windows 事件日志和 SQL Server。

2. System.Web.Management.SimpleMailWebEventProvider

有时，在应用程序中发生错误时，管理员或相关的开发人员希望通过电子邮件获得通知。除了使用 EventLogWebEventProvider 把事件记录到磁盘中之外，还可以使用 SimpleMailWebEventProvider 通过电子邮件发出错误通知。由这个提供程序的名称可知，该电子邮件的结构比较简单。程序清单 14-18 说明除了把错误写入 Windows 事件日志之外，如何添加电子邮件通知。

程序清单 14-18 SimpleMailWebEventProvider 定义

```

<configuration>
  <system.web>
    <healthMonitoring heartbeatInterval="0" enabled="true">
      <bufferModes>
        <add name="Website Error Notification"
          maxBufferSize="100"
          maxFlushSize="20"
          urgentFlushThreshold="1"
          regularFlushInterval="00:01:00"
          urgentFlushInterval="00:01:00"
          maxBufferThreads="1" />
      </bufferModes>
    <providers>
      <clear />
      <add name="EventLogProvider"
        type="System.Web.Management.EventLogWebEventProvider,
          System.Web, Version=4.0.0.0, Culture=neutral,
          PublicKeyToken=b03f5f7f11d50a3a" />
      <add name="SimpleMailProvider"
        type="System.Web.Management.SimpleMailWebEventProvider,
          System.Web, Version=4.0.0.0, Culture=neutral,
          PublicKeyToken=b03f5f7f11d50a3a"
        from="website@company.com"
        to="admin@company.com"
        cc="adminLevel2@company.com" bcc="director@company.com"
        bodyHeader="Warning!"
        bodyFooter="Please investigate ASAP."
        subjectPrefix="Action required."
        buffer="true"
        bufferMode="Website Error Notification"
        maxEventLength="4096"
        maxMessagesPerNotification="1" />
    </providers>
    <profiles>
      <!-- Removed for clarity -->
    </profiles>
    <rules>
      <add name="All Errors Default" eventName="All Errors"
        provider="EventLogProvider" profile="Default" minInstances="1"
        maxLimit="Infinite" minInterval="00:01:00" custom="" />
      <add name="Failure Audits Default" eventName="Failure Audits"
        provider="EventLogProvider" profile="Default" minInstances="1"
        maxLimit="Infinite" minInterval="00:01:00" custom="" />
      <add name="All Errors Simple Mail" eventName="All Errors"
        provider="SimpleMailProvider" profile="Default" />
      <add name="Failure Audits Default" eventName="Failure Audits"
        provider="SimpleMailProvider" profile="Default" />
    </rules>
    <eventMappings>
      <!-- Removed for clarity -->
    </eventMappings>
  </system.web>
</configuration>

```

```

    </healthMonitoring>
  </system.web>
</configuration>

```

在这个例子中，捕获了发生的错误，这些错误不仅被写入事件日志，还被通过电子邮件发送给在提供程序定义中指定的终端用户。SimpleMailWebEventProvider 的一个有趣的地方是，这个类继承了 BufferedWebEventProvider，而不是像 EventLogWebEventProvider 那样继承了 WebEventProvider。继承 BufferedWebEventProvider 意味着先让健康监控系统建立错误通知集合，然后再发送它们。<bufferModes>部分定义了缓存的工作方式。

3. System.Web.Management.TemplatedMailWebEventProvider

顾名思义，前面的 SimpleMailWebEventProvider 就是发送一封简单的、基于文本的电子邮件。要发送一封比较美观的、包含更多信息的电子邮件，可以使用 TemplatedMailWebEventProvider。与 SimpleMailWebEventProvider 一样，只需在<healthMonitoring>部分定义相应的提供程序即可。该定义的模型如程序清单 14-19 所示。

程序清单 14-19 TemplatedMailWebEventProvider 定义

```

<providers>
  <clear />
  <add name="EventLogProvider"
    type="System.Web.Management.EventLogWebEventProvider,
    System.Web, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b03f5f7f11d50a3a" />
  <add name="TemplatedMailProvider"
    type="System.Web.Management.TemplatedMailWebEventProvider,
    System.Web, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b03f5f7f11d50a3a"
    template="..\mailtemplates/errornotification.aspx"
    from="website@company.com"
    to="admin@company.com"
    cc="adminLevel2@company.com" bcc="director@company.com"
    bodyHeader="Warning!"
    bodyFooter="Please investigate ASAP."
    subjectPrefix="Action required."
    buffer="true"
    bufferMode="Website Error Notification"
    maxEventLength="4096"
    maxMessagesPerNotification="1" />
</providers>

```

这个提供程序声明与 SimpleMailWebEventProvider 的最大区别如程序清单 14-19 中的加粗部分所示。TemplatedMailWebEventProvider 的 template 特性指定了模板的位置，该模板将用于创建电子邮件，并将其从健康监控系统中发送出去。

4. System.Web.Management.SqlWebEventProvider

在许多情况下，捕获和记录应用程序中发生的 Web 事件时需要把事件写入磁盘。此时，EventLogWebEventProvider 是理想的提供程序，因为它会把这些 Web 事件自动写入 Windows 事件日志。但是，

有时也会希望把这些 Web 事件写入磁盘,此时,一种比较好的替代方式是把这些 Web 事件写入 SQL Server(也可以同时写入事件日志)。

写入 SQL Server 比写入 Windows 事件日志好一些。应用程序在 Web 场中运行时,我们希望把 Web 场中发生的所有错误都写入一个地方。此时,最好把通过健康监控系统捕获的所有 Web 事件都写入一个 SQL Server 实例,Web 场中的所有服务器都可以连接该 SQL Server 实例。

与本章前面基于 SQL Server 的其他提供程序一样,在默认情况下,SqlWebEventProvider 也把 SQL Server Express Edition 用作底层数据库。要连接全功能的 SQL Server 版本,需要定义好的连接,如程序清单 14-20 所示。

程序清单 14-20 定义好的 LocalSqlServer 实例

```
<configuration>
  <connectionStrings>
    <add name="LocalSqlServer"
        allowCustomSqlDatabase="true"
        connectionString="Data Source=127.0.0.1;Integrated Security=SSPI" />
  </connectionStrings>
</configuration>
```

有了这个连接,下一步就是在 web.config 文件的 SqlWebEventProvider 声明中使用它,如程序清单 14-21 所示。

程序清单 14-21 使用 SqlWebEventProvider 把 Web 事件写入 SQL Server 2012

```
<configuration>
  <system.web>
    <healthMonitoring>
      <!-- Other nodes removed for clarity -->
      <providers>
        <clear />
        <add name="SqlWebEventProvider"
            type="System.Web.Management.SqlWebEventProvider, System.Web"
            connectionStringName="LocalSqlServer"
            maxEventDetailsLength="1073741823"
            buffer="true"
            bufferMode="SQL Analysis" />
      </providers>
    </healthMonitoring>
  </system.web>
</configuration>
```

事件现在记录到 SQL Server 2012 中。SqlWebEventProvider 的优点在于,与 SimpleMailWebEventProvider 和 TemplatedMailWebEventProvider 一样,SqlWebEventProvider 也继承了 BufferedWebEventProvider。因此,可以成批写入 Web 事件,而不必一个一个地写入。使用提供程序声明中的 buffer 和 bufferMode 特性实现该功能。它可以与在<healthMonitoring>声明的<bufferMode>部分应用的设置一起使用。

5. System.Web.Management.TraceWebEventProvider

调试 ASP.NET 应用程序的一种方法是使用内置于系统中的跟踪功能。利用跟踪功能可以查看请求、应用程序状态、cookie、控件树、窗体集合等方面的细节。通过 `TraceWebEventProvider` 对象，可以把 Web 事件输出到跟踪结果中。在配置文件中设置 `TraceWebEventProvider` 实例的代码如程序清单 14-22 所示。

程序清单 14-22 使用 `TraceWebEventProvider` 把 Web 事件写入跟踪结果

```
<configuration>
  <system.web>
    <healthMonitoring>
      <!-- Other nodes removed for clarity -->
      <providers>
        <clear />
        <add name="TraceWebEventProvider"
              type="System.Web.Management.TraceWebEventProvider, System.Web"
              maxEventLength="4096"
              maxMessagesPerNotification="1" />
      </providers>
    </healthMonitoring>
  </system.web>
</configuration>
```

注意，即使有了提供程序，也必须给提供程序指定要捕获的错误。通过健康监控系统的 `<rules>` 部分完成该操作。

`IisTraceWebEventProvider` 与 `TraceWebEventProvider` 相同，只是它把跟踪信息传送给 IIS，而不是 ASP.NET 跟踪系统。

6. System.Web.Management.WmiWebEventProvider

内置于健康监控系统的最后一个提供程序是 `WmiWebEventProvider`。它可以将来自健康监控系统的 Web 事件映射为 Windows Management Instrumentation (WMI) 事件。将事件传送给 WMI 子系统时，可以把事件表示为对象。

在默认情况下，`WmiWebEventProvider` 已经设置好，只需把自己感兴趣的 Web 事件映射为在健康监控声明的 `<rules>` 部分已经声明的 `WmiWebEventProvider` 即可。这个声明位于服务器上 .NET Framework 的 CONFIG 文件夹的 `web.config.comments` 文件中，如程序清单 14-23 所示 (`WmiWebEventProvider` 以粗体显示)。

程序清单 14-23 `web.config.comments` 文件中的 `WmiWebEventProvider` 定义

```
<configuration>
  <system.web>
    <healthMonitoring>
      <!-- Other nodes removed for clarity -->
      <providers>
        <clear />
        <add name="EventLogProvider"
              type="System.Web.Management.EventLogWebEventProvider,
```

```

        System.Web,Version=4.0.0.0,Culture=neutral,
        PublicKeyToken=b03f5f7f11d50a3a" />
        <add connectionStringName="LocalSqlServer"
            maxEventDetailsLength="1073741823" buffer="false"
            bufferMode="Notification" name="SqlWebEventProvider"
            type="System.Web.Management.SqlWebEventProvider,
            System.Web,Version=4.0.0.0,Culture=neutral,
            PublicKeyToken=b03f5f7f11d50a3a" />
        <add name="WmiWebEventProvider"
            type="System.Web.Management.WmiWebEventProvider,
            System.Web,Version=4.0.0.0,Culture=neutral,
            PublicKeyToken=b03f5f7f11d50a3a" />
    </providers>
</healthMonitoring>
</system.web>
</configuration>

```

注意,健康监控系统使用提供程序模型的优点是,允许为系统捕获的 Web 事件使用多个提供程序。

14.2.8 提供程序的配置

ASP.NET 4.5 的一项强大功能是,允许加密配置文件的各个部分。可以加密 web.config 文件中定义好的 ASP.NET 部分和放在该文件中的定制部分。这就可以避免敏感的配置信息被读取应用程序文件库的人看到。

在默认情况下,ASP.NET 4.5 提供了两个配置提供程序,它们的定义如下:

- **System.Configuration.DpapiProtectedConfigurationProvider**: 使用内置于 Windows 操作系统的数据保护 API(Data Protection API, DPAPI)加密和解密配置部分。
- **System.Configuration.RsaProtectedConfigurationProvider**: 使用 RSA 公钥加密算法加密和解密配置部分。

这两个用于加密和解密配置部分的提供程序继承了 ProtectedConfigurationProvider 基类,如图 14-15 所示。

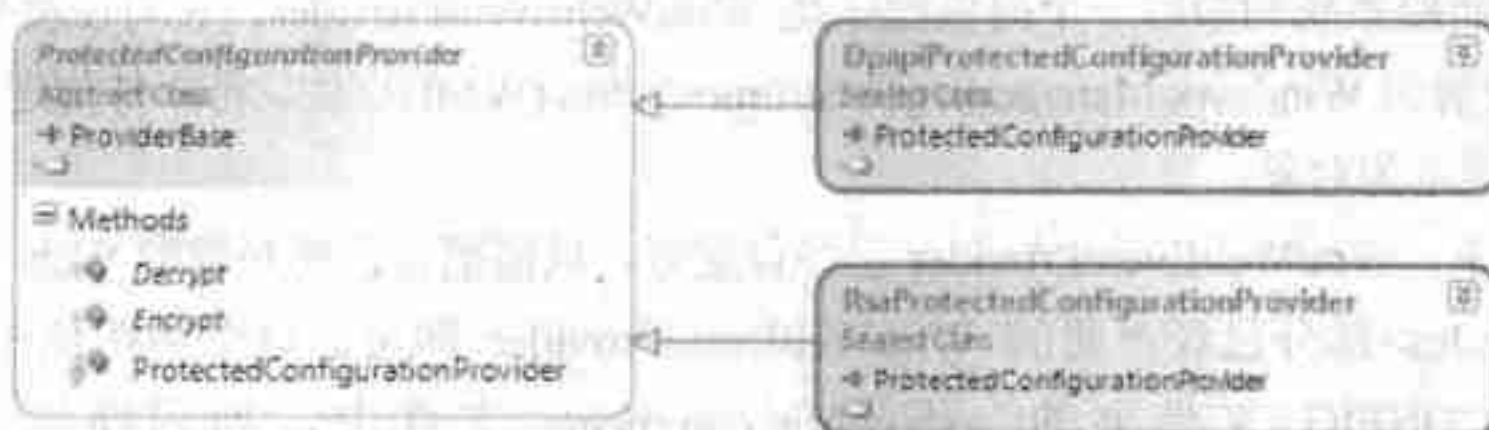


图 14-15



有关使用这两个提供程序加密和解密配置部分的内容,详见第 28 章。

下面逐个介绍这两个提供程序。

1. System.Configuration.DpapiProtectedConfigurationProvider

DpapiProtectedConfigurationProvider 类可以使用 Windows Data Protection API (DPAPI)加密和解

密配置部分。这个提供程序可以在每台计算机上执行这些加密和解密任务。它不适用于 Web 场。如果在 Web 场的配置文件中受保护的配置，就应使用 `RsaProtectedConfigurationProvider`。

在服务器的 `machine.config` 文件中，包含 `DpapiProtectedConfigurationProvider` 和 `RsaProtectedConfigurationProvider` 的定义。`RsaProtectedConfigurationProvider` 被设置为默认的配置提供程序。要把 `DpapiProtectedConfigurationProvider` 建立为默认的配置提供程序，可以使用应用程序的 `web.config` 文件，或者在 `machine.config` 文件中修改 `<configProtectedData>` 节点的 `defaultProvider` 特性。`web.config` 文件的改动如程序清单 14-24 所示。

程序清单 14-24 在 `web.config` 文件中使用 `DpapiProtectedConfigurationProvider`

```
<configuration>
  <configProtectedData defaultProvider="DataProtectionConfigurationProvider">
    <providers>
      <clear />
      <add name="DataProtectionConfigurationProvider"
        type="System.Configuration.DpapiProtectedConfigurationProvider,
        System.Configuration, Version=4.0.0.0,
        Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
        description="Uses CryptProtectData and CryptUnProtectData Windows
        APIs to encrypt and decrypt"
        useMachineProtection="true"
        keyEntropy="RandomStringValue" />
    </providers>
  </configProtectedData>
</configuration>
```

在配置文件的 `<configProtectedData>` 部分定义提供程序。注意该配置部分在 `<system.web>` 部分的外部。

这个提供程序定义的两个主要特性如下：

- `useMachineProtection` 特性默认设置为 `true`，表示服务器上的所有应用程序都共享加密和解密配置部分的相同含义。这也表示驻留在同一台计算机上的应用程序可以互相进行加密和解密。把 `useMachineProtection` 特性设置为 `false`，表示加密和解密操作仅在单个应用程序上进行。这个设置也说明，必须修改应用程序运行的账户，才能使其不同于服务器上的其他应用程序。
- `keyEntropy` 特性为防止一个应用程序解密另一个应用程序的配置部分提供了一种轻量级方式。该特性可以使任意字符串值参与加密和解密过程。

2. `System.Configuration.RsaProtectedConfigurationProvider`

加密和解密配置部分的默认提供程序是 `RsaProtectedConfigurationProvider`。这个设置在应用服务器的 `machine.config` 文件中。`machine.config` 文件中的相应代码如程序清单 14-25 所示。

程序清单 14-25 `machine.config` 文件中的 `RsaProtectedConfigurationProvider` 声明

```
<configuration>
  <configProtectedData defaultProvider="RsaProtectedConfigurationProvider">
    <providers>
```



```

<add name="RsaProtectedConfigurationProvider"
    type="System.Configuration.RsaProtectedConfigurationProvider,
    System.Configuration, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b03f5f7f11d50a3a"
    description="Uses RsaCryptoServiceProvider to encrypt and decrypt"
    keyContainerName="NetFrameworkConfigurationKey" cspProviderName=""
    useMachineContainer="true" useOAEP="false" />
<add name="DataProtectionConfigurationProvider"
    type="System.Configuration.DpapiProtectedConfigurationProvider,
    System.Configuration, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b03f5f7f11d50a3a"
    description="Uses CryptProtectData and CryptUnProtectData
    Windows APIs to encrypt and decrypt"
    useMachineProtection="true" keyEntropy="" />
</providers>
</configProtectedData>
</configuration>

```

RsaProtectedConfigurationProvider 使用 Triple-DES 加密算法来加密配置文件中指定的部分。这个提供程序只有几个可用特性：

- keyContainerName 特性是用于加密和解密过程的指定键容器。这个提供程序默认使用内置于 .NET Framework 中的默认键容器，但是通过该特性，也可以很容易地将应用程序切换为另一个键容器。
- 只有在指定定制的加密服务提供程序(CSP)以使用 Windows Cryptographic API (CAPI)时，才使用 cspProviderName 特性。此时，应把 CSP 的名称指定为 cspProviderName 特性的值。
- useMachineContainer 特性可以指定是使用在整个计算机上都有效的键容器，还是使用用户特定的键容器。这个特性类似于 DpapiProtectedConfigurationProvider 中的 useMachineProtection 特性。
- UseOAEP 特性指定在加密和解密过程中是否启用 Optional Asymmetric Encryption and Padding (OAEP)功能。

提供程序的配置

如本章前面所述，可以将 ASP.NET 4.5 中的这些系统关联到许多可用的提供程序。另外，还可以通过提供程序的特性，配置所关联的提供程序的行为。很容易通过用于整个系统的配置文件(如 machine.config 文件)或仅用于某个应用程序的配置文件(如 web.config 文件)来实现该操作。

14.3 本章小结

本章介绍了提供程序模型的基础知识，以及使用各种 ASP.NET 系统时可用的提供程序。我们应掌握每个系统可用的内置提供程序，以及如何细调每个提供程序的行为。

提供程序模型可以增加一层抽象，允许用户确定用于各个系统的底层数据存储。例如，可以确定是把成员资格和角色管理信息存储在 SQL Server 中，还是存储在 Oracle 中，而无须对业务或表示逻辑做任何修改。

下一章将介绍如何更进一步地利用提供程序模型。

第 15 章

扩展提供程序模型

本章要点

- 修改和扩展提供程序
- 建立自己的提供程序

第 14 章介绍了 ASP.NET 4.5 中的提供程序模型,解释了如何将它用于成员资格和角色管理系统。如第 14 章所述,ASP.NET 4.5 中的这些系统需要在相当长的时间内维护某种类型的用户状态。它们对状态存储的时间间隔和安全需求比早期仅使用 Session 对象的系统更高。另外,ASP.NET 4.5 还提供了一系列提供程序,用作底层连接器,以满足这些系统对状态管理的数据存储要求。

.NET Framework 4.5 默认安装中的提供程序包含了使用这些系统所需的状态管理数据存储的最常见方式。但与 .NET 中的大多数对象一样,可以定制和扩展已有的提供程序。

本章将介绍一些扩展 ASP.NET 4.5 中提供程序模型的方式,探讨两个扩展提供程序模型的示例。但是,本章将首先讨论一些比较简单的方式,以修改和扩展 .NET Framework 4.5 默认安装中已有的提供程序。

15.1 提供程序是较大体系结构中的一层

第 14 章介绍过,提供程序可以定义 ASP.NET 4.5 中许多系统的数据访问层。它们还可以定义核心的业务逻辑实现方式,指定如何操作或处理数据。它们可以使用各种控件和 API,使这些系统以统一的方式组合起来,而无论提供程序采用何种底层数据存储方法。提供程序模型也允许从一个提供程序转换到另一个提供程序,而不会影响底层的控件和与提供程序交互的 API。这个模型如图 15-1 所示。

在图 15-1 中,成员资格系统中使用的控件和成员资格 API 都使用指定的提供程序。改变底层的提供程序不会改变控件或 API,但可以明确修改这些项的操作方式(稍后可以看到)。也可以只修改这些项所需的状态管理的存储位置。

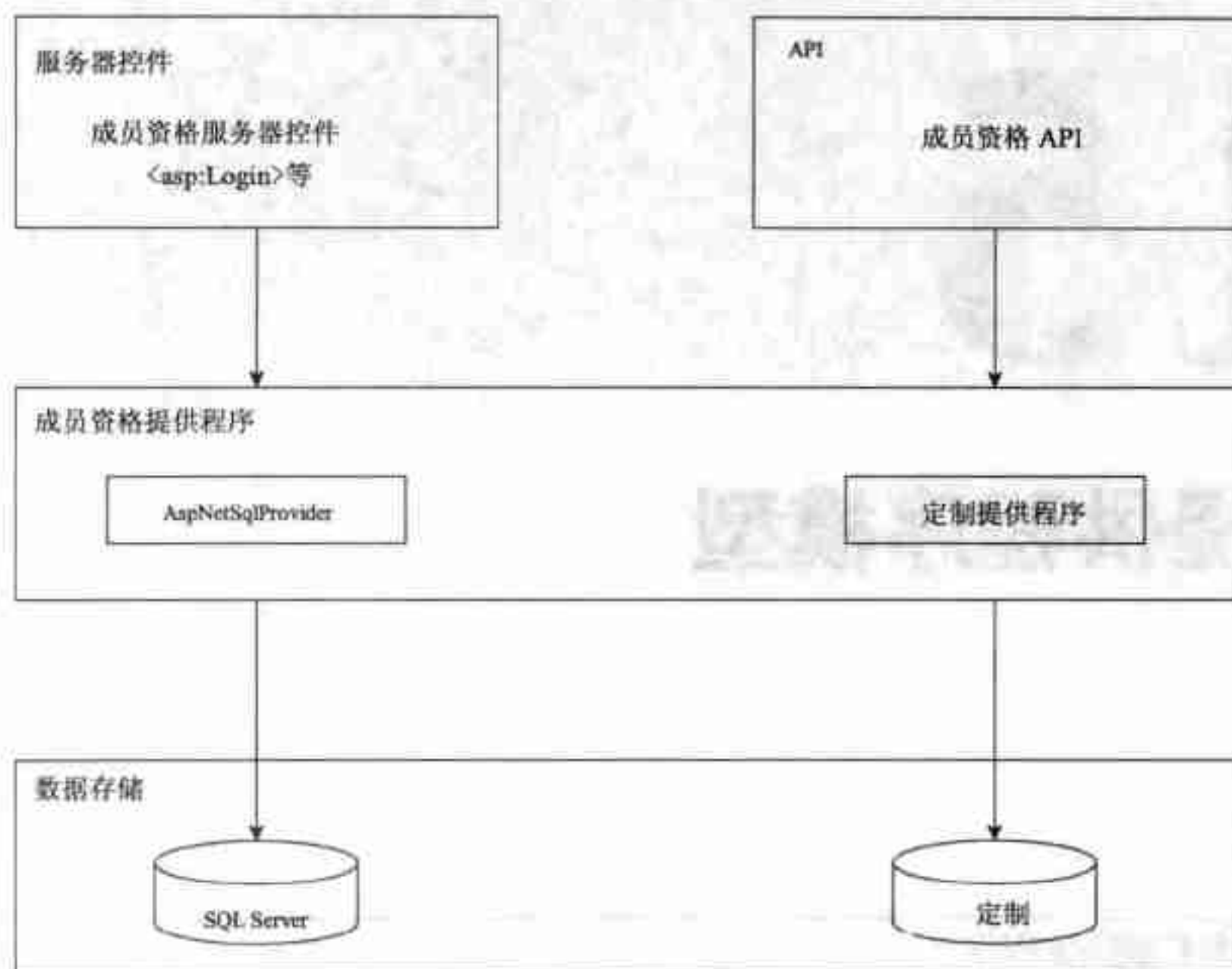


图 15-1



除了本章介绍的示例之外，还可以使用 ASP.NET 通用提供程序(在 Package Manager 中安装 Microsoft.AspNet.Providers 包)。通用提供程序没有修改 API，但替换了内置提供程序中的数据存储机制。如第 14 章所述，这个主题详见第 18 和 19 章。

15.2 通过基于特性的编程修改提供程序的行为

要修改 .NET Framework 4.5 内置的提供程序的行为，可能最简单的方式就是通过基于特性的编程。在 ASP.NET 4.5 中，可以通过使用特性修改相当高级的行为。使用 `machine.config` 文件或 `web.config` 根文件中的提供程序定义，可以改变提供程序的行为。本章将举例说明如何修改 `SqlMembershipProvider`。

15.2.1 通过 `SqlMembershipProvider` 建立简单的密码结构

在使用 `SqlMembershipProvider` 实例创建用户时，无论是使用 SQL Server Express 还是 SQL Server 2005/2008/2012，都要注意创建用户所需要的密码是一种半强类型化的密码。在通过 ASP.NET Web Site Administration 工具创建用户时，这一点非常明显，如图 15-2 所示。ASP.NET Web Site Administration 工具详见附录 D。

在这个屏幕上尝试输入一个密码，但屏幕提示，该密码不满足应用程序的要求。密码的最小长度是 7 个字符，而且其中至少要有一个非数字的字符。也就是说，需要输入像 `Micro$oft` 这样的密码。

这类行为是由成员资格提供程序指定的，而不是由成员资格系统中使用的控件或 API 指定的。该要求的定义在 C:\WINDOWS\Microsoft.NET\Framework\v4.0.xxxx\CONFIG 下的 machine.config.comments 文件中。这个定义如程序清单 15-1 所示。

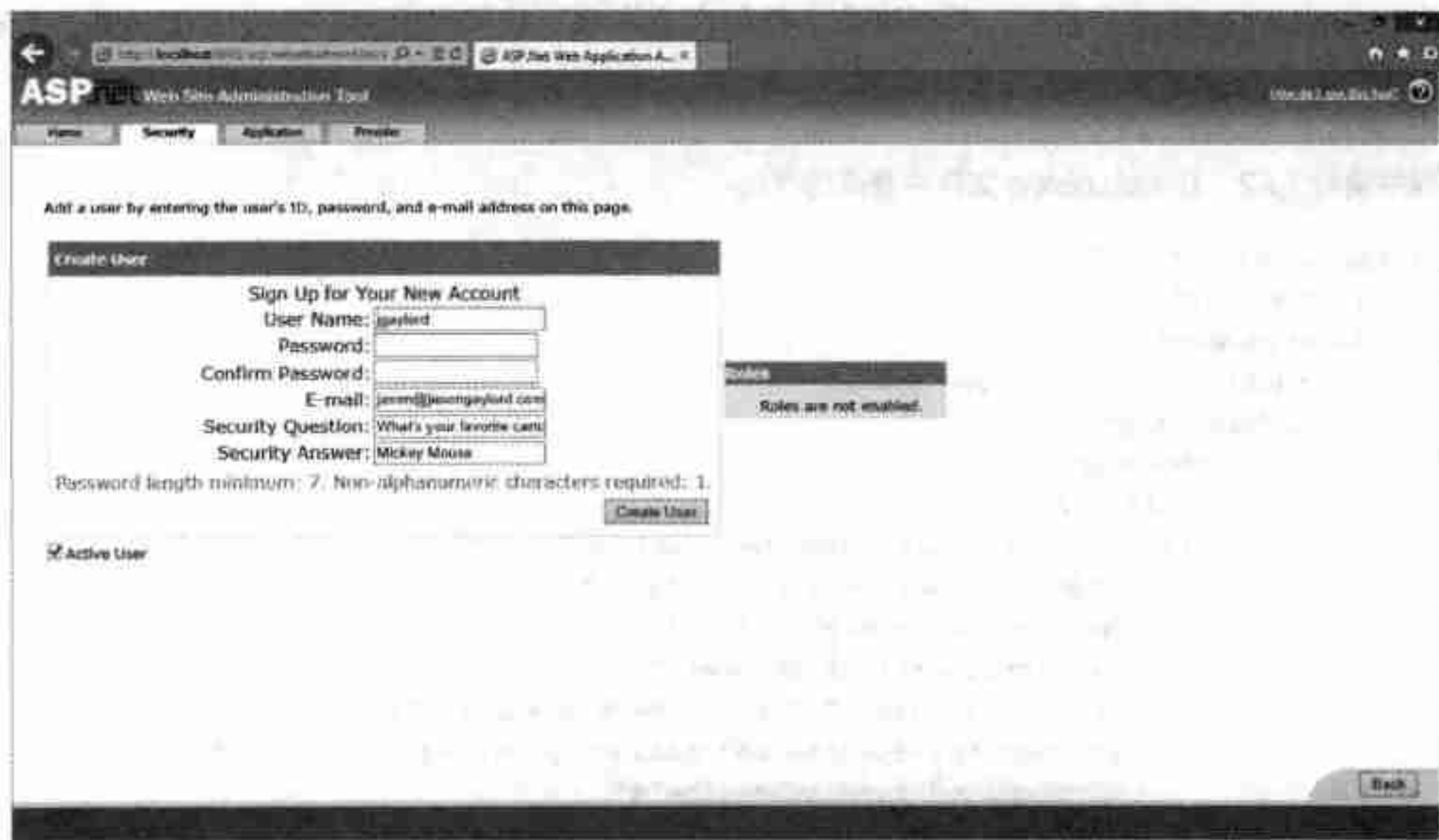


图 15-2

程序清单 15-1 SqlMembershipProvider 实例声明

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <membership defaultProvider="AspNetSqlMembershipProvider"
      userIsOnlineTimeWindow="15" hashAlgorithmType="">
      <providers>
        <clear />
        <add connectionStringName="LocalSqlServer"
          enablePasswordRetrieval="false"
          enablePasswordReset="true" requiresQuestionAndAnswer="true"
          applicationName="/" requiresUniqueEmail="false"
          passwordFormat="Hashed" maxInvalidPasswordAttempts="5"
          minRequiredPasswordLength="7"
          minRequiredNonalphanumericCharacters="1"
          passwordAttemptWindow="10"
          passwordStrengthRegularExpression=""
          name="AspNetSqlMembershipProvider"
          type="System.Web.Security.SqlMembershipProvider, System.Web,
            Version=4.0.0.0, Culture=neutral,
            PublicKeyToken=b03f5f7f11d50a3a" />
      </providers>
    </membership>
  </system.web>
</configuration>
```

下面查看这个提供程序的特性,注意 `minRequiredPasswordLength` 和 `minRequiredNonalphanumericCharacters` 特性定义了上述行为。要在服务器上的每个应用程序中改变该行为,只需修改这个文件中的这些值。但是,建议在各个应用程序的 `web.config` 文件中修改这些值,如程序清单 15-2 所示(因为 `web.config` 修改了多次,所以这个文件保存为本章下载代码中的 `Listing15-02.xml`,以便于下载)。

程序清单 15-2 在 `web.config` 文件中修改特性值

```
<?xml version="1.0" ?>
<configuration>
  <system.web>
    <authentication mode="Forms" />
    <membership>
      <providers>
        <clear />
        <add connectionStringName="LocalSqlServer"
          enablePasswordRetrieval="false"
          enablePasswordReset="true"
          requiresQuestionAndAnswer="true"
          applicationName="/" requiresUniqueEmail="false"
          passwordFormat="Hashed" maxInvalidPasswordAttempts="5"
minRequiredPasswordLength="4"
minRequiredNonalphanumericCharacters="0"
          passwordAttemptWindow="10"
          name="AspNetSqlMembershipProvider"
          type="System.Web.Security.SqlMembershipProvider,
            System.Web, Version=4.0.0.0, Culture=neutral,
            PublicKeyToken=b03f5f7f11d50a3a" />
      </providers>
    </membership>
  </system.web>
</configuration>
```

在这个例子中,通过 `minRequiredPasswordLength` 和 `minRequiredNonalphanumericCharacters` 特性改变了对密码的要求。密码允许的最小长度是 4 个字符,并且这些字符都可以是数字和字母(特殊字符包括!、\$、#等)。

在应用程序的 `web.config` 文件中重新定义提供程序是一个相当简单的过程。在程序清单 15-2 中, `<membership>` 元素非常类似于 `machine.config` 文件中的 `<membership>` 元素。

在定义自己的 `SqlMembershipProvider` 实例时有两个选择。第一个选择如程序清单 15-2 所示,重新定义 `machine.config` 文件中的 `SqlMembershipProvider` 实例(即 `AspNetSqlMembershipProvider`,其值来自于提供程序声明中的 `name` 特性)。如果采用这种方法,就必须清理以前定义的 `AspNetSqlMembershipProvider` 实例,在 `<providers>` 部分使用 `<clear />` 节点重新定义 `AspNetSqlMembershipProvider`。否则就会抛出错误,说明已经定义过这个提供程序。

在清理了以前定义的 `AspNetSqlMembershipProvider` 实例后,使用 `<add>` 元素重新定义这个提供程序。在程序清单 15-2 中,使用 `minRequiredPasswordLength` 和 `minRequiredNonalphanumericCharacters` 特性的新值(以粗体显示的部分)重新定义了对密码的要求。

在定义自己的 `SqlMembershipProvider` 实例时,另一个选择是为 `<add>` 元素中定义的提供程序指

定唯一的 `name` 特性值。如果采用这种方法,就必须使用 `defaultProvider` 特性,把这个新命名的实例指定为成员资格系统的默认提供程序。该方法如程序清单 13-3 所示(因为 `web.config` 修改了多次,所以这个文件保存为本章下载代码中的 `Listing15-03.xml`,以便于下载)。

程序清单 15-3 定义自己的 `SqlMembershipProvider` 实例

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <authentication mode="Forms" />
    <membership defaultProvider="JasonsSqlMembershipProvider">
      <providers>
        <add connectionStringName="LocalSqlServer"
          enablePasswordRetrieval="false"
          enablePasswordReset="true"
          requiresQuestionAndAnswer="true"
          applicationName="/" requiresUniqueEmail="false"
          passwordFormat="Hashed" maxInvalidPasswordAttempts="5"
          minRequiredPasswordLength="4"
          minRequiredNonalphanumericCharacters="0"
          passwordAttemptWindow="10"
          name="JasonsSqlMembershipProvider"
          type="System.Web.Security.SqlMembershipProvider,
            System.Web, Version=4.0.0.0, Culture=neutral,
            PublicKeyToken=b03f5f7f11d50a3a" />
      </providers>
    </membership>
  </system.web>
</configuration>
```

在这个例子中, `machine.config` 文件中的 `SqlMembershipProvider` 实例(使用 `JasonsSqlMembershipProvider` 名称定义)没有重新定义,而是在 `web.config` 文件中定义了一个全新的实例 `MyVeryOwnAspNetSqlMembershipProvider`。

15.2.2 通过 `SqlMembershipProvider` 建立强类型的密码结构

下面介绍如何使密码的结构更复杂。当然,完成这项任务有两种方法。一种方法是使用前面介绍的 `minRequiredPasswordLength` 和 `minRequiredNonalphanumericCharacters` 特性,使密码达到需要的长度(较长的密码一般比较安全),并包含指定数量的非数字字母的字符(这也会增加安全性)。

另一种方法是使用 `passwordStrengthRegularExpression` 特性。如果 `minRequiredPasswordLength` 和 `minRequiredNonalphanumericCharacters` 特性不能提供需要的密码结构,那么最好使用 `passwordStrengthRegularExpression` 特性。

下面是使用这个特性的一个示例。假定要求用户的密码包含:

- 至少一个大写字母
- 至少一个小写字母
- 至少一个数字
- 至少一个特殊字符
- 至少 8 个字符

可以把提供程序定义为如程序清单 15-4 所示(因为 web.config 修改了多次, 所以这个文件保存为本章下载代码中的 Listing15-04.xml, 以便于下载)。

程序清单 15-4 web.config 文件中改变密码结构的提供程序实例

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <authentication mode="Forms" />
    <membership defaultProvider="JasonsSqlMembershipProvider">
      <providers>
        <add connectionStringName="LocalSqlServer"
          enablePasswordRetrieval="false"
          enablePasswordReset="true"
          requiresQuestionAndAnswer="true"
          applicationName="/" requiresUniqueEmail="false"
          passwordFormat="Hashed" maxInvalidPasswordAttempts="5"
          passwordStrengthRegularExpression=
            "(?=^.{8,}$)(?=.*\d)(?=.*\W+)(?![.\n])(?=.*[A-Z])(?=.*[a-z]).*$"
          passwordAttemptWindow="10"
          name="JasonsSqlMembershipProvider"
          type="System.Web.Security.SqlMembershipProvider,
            System.Web, Version=4.0.0.0, Culture=neutral,
            PublicKeyToken=b03f5f7f11d50a3a" />
      </providers>
    </membership>
  </system.web>
</configuration>
```

这里没有使用 minRequiredPasswordLength 和 minRequiredNonalphanumericCharacters 特性, 而使用了 passwordStrengthRegularExpression 特性, 其值是:

```
(?=^.{8,}$)(?=.*\d)(?=.*\W+)(?![.\n])(?=.*[A-Z])(?=.*[a-z]).*$
```



正则表达式有几个在线资源。如第 6 章所述, 查找正则表达式最流行的一个 Internet 站点是 RegExLib, 网址是 www.regexlib.com。

这就说明, 有许多方式可以修改 .NET Framework 4.5 安装中已有的提供程序的行为。可以调整内置于架构中的许多提供程序, 使用基于特性的编程方式, 使它们满足自己的要求。SqlMembershipProvider 示例就演示了该过程, 对其他提供程序也可以进行类似的简单修改。

15.3 ProviderBase 类

所有的提供程序都以某种方式派生于 System.Configuration.Provider 名称空间中的 ProviderBase 类。ProviderBase 是一个抽象类, 用于为继承

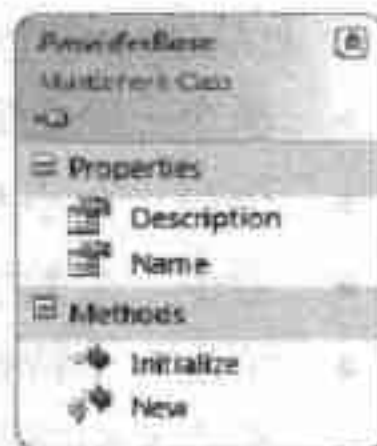


图 12-3

它的提供程序定义基本模板。注意 `ProviderBase` 抽象类包含的内容很少，如图 15-3 所示。

如前所述，这个类的内容不多。它只是提供程序的根类，可以让提供程序初始化自身。

`Name` 属性用于提供友好的名称，例如 `AspNetSqlRoleProvider`。`Description` 属性用于指定提供程序的文本描述，以后可能由管理工具使用该文本描述。`ProviderBase` 类的主要成员是 `Initialize()` 方法，它的构造代码如下：

```
public virtual void Initialize(string name, NameValueCollection config)
```

注意 `Initialize()` 方法的两个参数。第一个参数是 `name`，它就是配置文件中赋予提供程序声明的 `name` 特性值。`config` 参数的类型是 `NameValueCollection`，这是一个名/值对的集合。这些名/值对就是在配置文件的提供程序声明中定义为各种特性及相关值的项。

查看 ASP.NET 4.5 默认安装中包含的提供程序，就会发现每个提供程序都定义了一个可以继承的类，该类实现了 `ProviderBase` 抽象类。例如，在成员资格系统的模型中，`SqlMembershipProvider` 继承了 `MembershipProvider`，而 `MembershipProvider` 实现了 `ProviderBase`。该模型如图 15-4 所示。

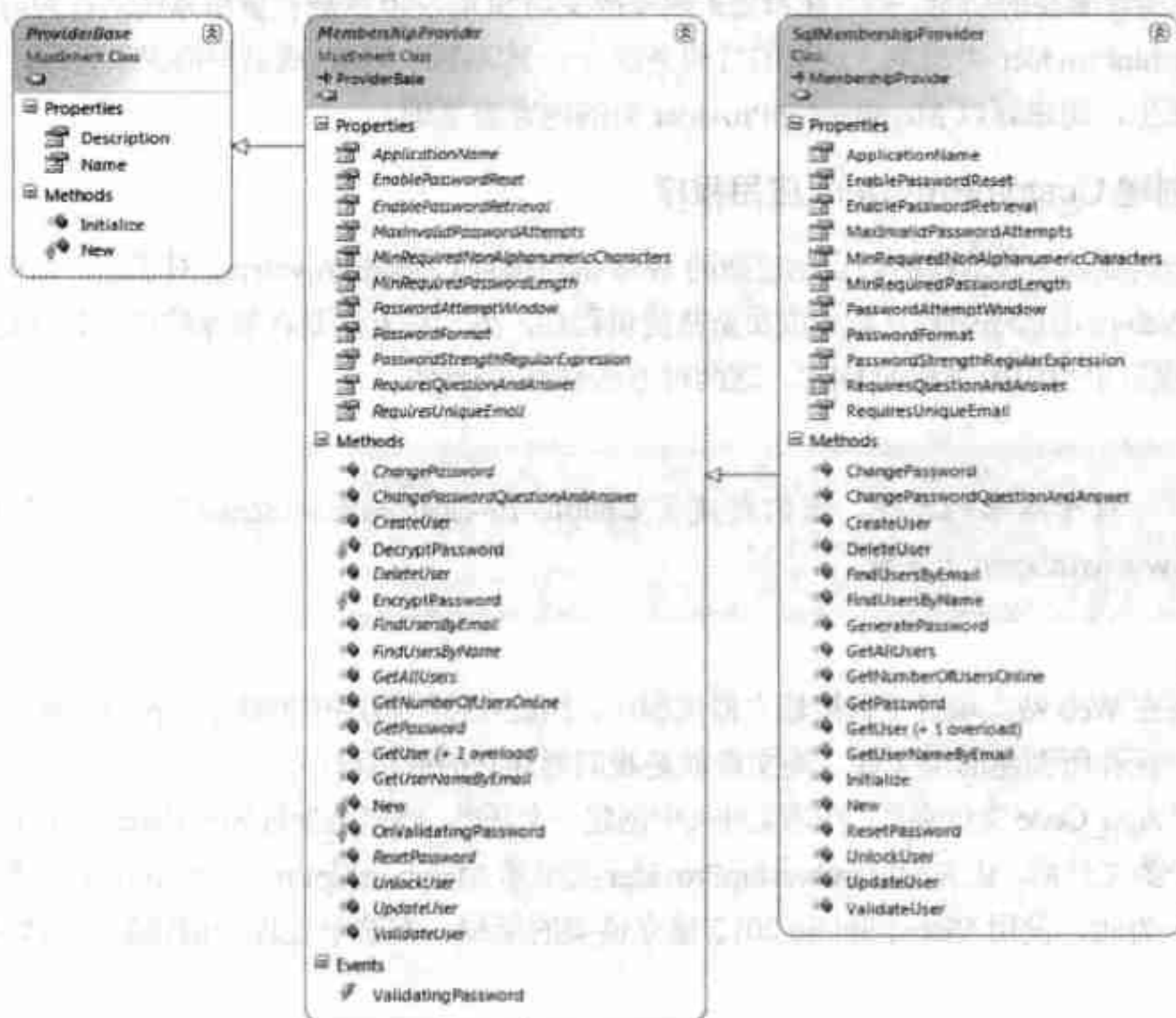


图 15-4

注意，各个系统都实现了一个特定的基本提供程序以供使用。不可能使用一个提供程序满足所有系统的需要。如图 15-4 所示，`MembershipProvider` 实例有一些 ASP.NET 成员资格系统所需的特定功能，其方法肯定不是角色管理系统或 Web Parts 功能所需的。

有了这些基本提供程序，在创建自己的定制提供程序以使用 ASP.NET 的成员资格系统时，就有两种选择：首先，可以实现 `ProviderBase` 类，从头开始创建自己的提供程序。这里不推荐这种方法，因为已经有操作各个系统的抽象类。其次，可以实现 `MembershipProvider`，以它提供的模型为基础，

这是比较好的方法。如果使用某个版本的 SQL Server，并且只想改变该提供程序的底层行为，就可以继承 `SqlMembershipProvider`，再修改该继承类的行为。下面将通过示例来介绍扩展提供程序模型的各种方式。

15.4 建立自己的提供程序

现在介绍关于建立自己的提供程序以在 ASP.NET 应用程序中使用的过程。实际上，提供程序并不难建立(后面将介绍)，甚至可以直接在 ASP.NET 4.5 项目中创建。下面的示例将演示如何建立一个可以在 XML 文件中使用的成员资格提供程序。一些较小的 Web 站点常常需要创建这样的提供程序。而较大的 Web 站点和基于 Web 的应用程序一般要使用某种类型的数据库，而不是使用 XML 文件来管理用户。

在建立自己的成员资格提供程序时有两种方法：从 `SqlMembershipProvider` 类或 `MembershipProvider` 类派生，以实现需要的功能。只有在希望扩展或改变与 SQL 交互的成员资格系统的行为时，才应从 `SqlMembershipProvider` 类派生。这里的目的是建立一个只读的 XML 成员资格提供程序，因此不应从这个类派生，而最好以 `MembershipProvider` 类的内容为基础。

15.4.1 创建 CustomProviders 应用程序

这个示例要以用户选择的语言创建新的 Web 站点项目 `CustomProviders`。对于这个示例来说，需要直接在 Web 应用程序中建立新的成员资格提供程序。另一种方法是在类库项目中建立提供程序，再在 Web 项目中引用所生成的 DLL。这两种方法都是可行的。



对于本章的示例，我们创建了 `Chapter15-CustomProviders-CS`，代码可以从 www.wrox.com 上下载。

这里要在 Web 站点项目中直接建立提供程序，因此在应用程序中创建了 `App_Code` 文件夹。在该文件夹中保存所创建的类文件。类文件就是我们要建立的提供程序。

创建好 `App_Code` 文件夹后，在该文件夹中创建一个新类，将该类命名 `XmlMembershipProvider.cs`。建立好这个类文件后，让 `XmlMembershipProvider` 类继承 `MembershipProvider`，并确定要重写哪些方法和属性。为此，使用 Visual Studio 2012 建立该类的架构。在这个过程中创建的代码如程序清单 15-5 所示。

程序清单 15-5 `XmlMembershipProvider` 类的开头部分

```
namespace Chapter15_CustomProviders_CS.App_Code
{
    public class XmlMembershipProvider : MembershipProvider
    {
    }
}
```

上面只对新类 `XmlMembershipProvider` 做了一处修改。这个新类继承了 `MembershipProvider`。

15.4.2 构造需要的类架构

为了让 Visual Studio 2012 使用需要的方法和属性建立 XmlMembershipProvider 类, 执行下面的步骤。在 C# 中, 需要右击代码中的 MembershipProvider 语句, 并从可用选项中选择 Implement Abstract Class。另一种选择是将光标放在文档窗口中的 MembershipProvider 语句上, 然后从 Visual Studio 菜单中选择 Edit | IntelliSense | Implement Abstract Class 命令。执行上述操作后, Visual Studio 的文档窗口就会显示类的全部架构代码。程序清单 15-6 列出了创建 XmlMembershipProvider 类时生成的代码。

程序清单 15-6 Visual Studio 为 XmlMembershipProvider 类生成的代码

```
namespace Chapter15_CustomProviders_CS.App_Code
{
    public class XmlMembershipProvider : MembershipProvider
    {
        public override string ApplicationName
        {
            get
            {
                throw new NotImplementedException();
            }
            set
            {
                throw new NotImplementedException();
            }
        }

        public override bool ChangePassword(string username, string oldPassword,
            string newPassword)
        {
            throw new NotImplementedException();
        }

        public override bool ChangePasswordQuestionAndAnswer(string username, string password,
            string newPasswordQuestion, string newPasswordAnswer)
        {
            throw new NotImplementedException();
        }

        public override MembershipUser CreateUser(string username, string password,
            string email, string passwordQuestion, string passwordAnswer, bool isApproved,
            object providerUserKey, out MembershipCreateStatus status)
        {
            throw new NotImplementedException();
        }

        public override bool DeleteUser(string username, bool deleteAllRelatedData)
        {
            throw new NotImplementedException();
        }

        public override bool EnablePasswordReset
        {

```

```
        get { throw new NotImplementedException(); }
    }

    public override bool EnablePasswordRetrieval
    {
        get { throw new NotImplementedException(); }
    }

    public override MembershipUserCollection FindUsersByEmail(string emailToMatch,
        int pageIndex, int pageSize, out int totalRecords)
    {
        throw new NotImplementedException();
    }

    public override MembershipUserCollection FindUsersByName(string usernameToMatch,
        int pageIndex, int pageSize, out int totalRecords)
    {
        throw new NotImplementedException();
    }

    public override MembershipUserCollection GetAllUsers(int pageIndex, int pageSize,
        out int totalRecords)
    {
        throw new NotImplementedException();
    }

    public override int GetNumberOfUsersOnline()
    {
        throw new NotImplementedException();
    }

    public override string GetPassword(string username, string answer)
    {
        throw new NotImplementedException();
    }

    public override MembershipUser GetUser(string username, bool userIsOnline)
    {
        throw new NotImplementedException();
    }

    public override MembershipUser GetUser(object providerUserKey, bool userIsOnline)
    {
        throw new NotImplementedException();
    }

    public override string GetUserNameByEmail(string email)
    {
        throw new NotImplementedException();
    }

    public override int MaxInvalidPasswordAttempts
    {

```

```
        get { throw new NotImplementedException(); }
    }

    public override int MinRequiredNonAlphanumericCharacters
    {
        get { throw new NotImplementedException(); }
    }

    public override int MinRequiredPasswordLength
    {
        get { throw new NotImplementedException(); }
    }

    public override int PasswordAttemptWindow
    {
        get { throw new NotImplementedException(); }
    }

    public override MembershipPasswordFormat PasswordFormat
    {
        get { throw new NotImplementedException(); }
    }

    public override string PasswordStrengthRegularExpression
    {
        get { throw new NotImplementedException(); }
    }

    public override bool RequiresQuestionAndAnswer
    {
        get { throw new NotImplementedException(); }
    }

    public override bool RequiresUniqueEmail
    {
        get { throw new NotImplementedException(); }
    }

    public override string ResetPassword(string username, string answer)
    {
        throw new NotImplementedException();
    }

    public override bool UnlockUser(string userName)
    {
        throw new NotImplementedException();
    }

    public override void UpdateUser(MembershipUser user)
    {
        throw new NotImplementedException();
    }
}
```



```

        public override bool ValidateUser(string username, string password)
        {
            throw new NotImplementedException();
        }
    }
}

```

代码非常多!有了这个架构后,下一步就应建立一些项,供 Visual Studio 建立的提供程序使用。首先建立 XML 文件,它包含允许访问应用程序的所有用户。

15.4.3 创建 XML 用户数据存储

这是一个 XML 成员资格提供程序,目的是从 XML 文件中读取用户信息,而不是从 SQL Server 数据库中读取,因此必须定义提供程序可以使用的 XML 文件结构。这个示例使用的结构如程序清单 15-7 所示(各个项目中的 App_Data\Users.xml)。

程序清单 15-7 用于存储用户名和密码的 XML 文件

```

<?xml version="1.0" encoding="utf-8" ?>
<Users>
  <User>
    <Username>JasonGaylord</Username>
    <Password>Reindeer</Password>
    <Email>jason@jasongaylord.com</Email>
    <DateCreated>12/10/2012</DateCreated>
  </User>
  <User>
    <Username>ScottHanselman</Username>
    <Password>YabbaDabbaDo</Password>
    <Email>scott@outlook.com</Email>
    <DateCreated>12/02/2012</DateCreated>
  </User>
  <User>
    <Username>ChristianWenz</Username>
    <Password>BamBam</Password>
    <Email>christian@outlook.com</Email>
    <DateCreated>01/11/2013</DateCreated>
  </User>
</Users>

```

这个 XML 文件只保存了 3 个用户实例,它们都包含了用户名、密码、电子邮件地址和创建用户的日期。因为是数据文件,所以应把该文件放在 ASP.NET 应用程序的 App_Data 文件夹中。该文件的名称可任意设置,这里将它命名为 Users.xml。

本章的后面在验证用户时,将从这个 XML 文件中提取这些值。

15.4.4 在 web.config 文件中定义提供程序实例

第 14 章在配置文件(如 machine.config 或 web.config 文件)中定义了一个提供程序及其行为。这个提供程序是为单个应用程序实例定义的,因此本例在应用程序的 web.config 文件中定义了另一个提供程序。

默认的提供程序是 SqlMembershipProvider,它是在服务器的 machine.config 文件中定义的。对

于这个示例，必须重写这个设置，建立新的默认提供程序。web.config 文件中的 XML 成员资格提供程序声明如程序清单 15-8 所示。

程序清单 15-8 在 web.config 文件中定义 XmlMembershipProvider

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.5" />
    <httpRuntime targetFramework="4.5" />
    <authentication mode="Forms"/>
    <membership defaultProvider="XmlFileProvider">
      <providers>
        <add name="XmlFileProvider" type="XmlMembershipProvider"
          xmlUserDatabaseFile="/App_Data/Users.xml"/>
      </providers>
    </membership>
  </system.web>
</configuration>
```

在这个程序清单中，默认提供程序定义为 XmlFileProvider。因为这个提供程序的名称没有在父配置文件中指定，所以必须在 web.config 文件中定义 XmlFileProvider。

使用 defaultProvider 特性可以定义用于成员资格系统的提供程序名称，这里是 XmlFileProvider。接着在 <providers> 部分使用 <add> 元素定义 XmlFileProvider 实例。<add> 元素为提供程序指定名称 XmlFileProvider，它还指向提供程序的类(或类型)，这里指向的是刚才创建的架构类 XmlMembershipProvider。



提供程序类可能需要完全限定的类名，这表示类名要使用对应的名称空间作为前缀。

除了已经使用的这些特性之外，还可以在提供程序声明中创建需要的特性。但是，无论创建什么类型的提供程序，都必须在提供程序中指定特性，并操作通过这些特性提供的值。在简单的 XmlMembershipProvider 中，只有定制特性 xmlUserDatabaseFile，它指向用户数据库 XML 文件的位置。对于这个提供程序而言，这是一个可选特性。如果没有为 xmlUserDatabaseFile 提供值，就使用默认值。在程序清单 15-8 中，为要使用的 XML 文件提供了一个值。注意 xmlUserDatabaseFile 只是文件名。

在该例中没有列出特性 applicationName，但它是一个可用的特性，因为在 XmlMembershipProvider 类中指定了该特性。该特性指向 XmlMembershipProvider 实例要定位的应用程序。其默认值也放在配置文件内这个默认提供程序的声明中，如下所示：

```
applicationName="/"
```

15.4.5 不实现 MembershipProvider 类的方法和属性

现在查看 XmlMembershipProvider 类。下一步是实现提供程序需要的方法或属性。不需要实际

使用这个架构包含的方法，而只需扩展感兴趣的方法。例如，如果不允许通过编程访问的方式改变密码(这会改变采用编程访问方式的控件)，可以不执行某个操作。或者如果有人试图实现该方法，就会抛出异常，如程序清单 15-9 所示。

程序清单 15-9 通过抛出异常，不实现某个可用的方法

```
public override bool ChangePassword(string username, string oldPassword,
    string newPassword)
{
    throw new NotSupportedException();
}
```

如果调用 `ChangePassword` 方法，就会抛出 `NotSupportedException` 异常。如果不想抛出异常，可以返回 `false` 值，不执行其他操作(但这会使试图实现该提供程序的开发人员感到困惑，不理解该方法的底层逻辑)，如程序清单 15-10 所示。

程序清单 15-10 通过返回 false 值，不实现某个可用的方法

```
public override bool ChangePassword(string username, string oldPassword,
    string newPassword)
{
    return false;
}
```

本章不打算介绍 `XmlMembershipProvider` 的每个操作，因此，读者可以自己试着使用 `MembershipProvider` 派生实例的可用方法和属性，对不打算使用的项进行必要的修改。

15.4.6 实现 `MembershipProvider` 类的方法和属性

下面实现 `MembershipProvider` 类的一些方法和属性，以使 `XmlMembershipProvider` 类能够发挥作用。首先要定义一些私有变量，供类中的多个方法使用。这些变量声明如程序清单 15-11 所示。

程序清单 15-11 在 `XmlMembershipProvider` 类中声明一些私有变量

```
public class XmlMembershipProvider : MembershipProvider
{
    private string _AppName;
    private Dictionary<string, MembershipUser> _MyUsers;
    private string _FileName;

    // Code removed for clarity
}
```

上面声明的变量是类中多个方法需要的变量。`_AppName` 变量定义了使用 XML 成员资格提供程序的应用程序，在本例中是本地应用程序。XML 文件中所有的成员都放在某种类型的集合中。这个例子使用字典泛型类 `_MyUsers`。最后，这个示例使用 `_FileName` 变量指向要使用的文件。

1. 定义 ApplicationName 属性

定义了私有变量后, 下一步就是定义 ApplicationName 属性。这里要使用第一个私有变量 AppName。ApplicationName 的属性定义如程序清单 15-15 所示。

程序清单 15-12 定义 ApplicationName 属性

```
public override string ApplicationName
{
    get
    {
        return _AppName;
    }
    set
    {
        _AppName = value;
    }
}
```

定义了 ApplicationName 属性后, 就要在 web.config 文件的提供程序声明(XmlFileProvider)中检索已定义的值。

2. 扩展 Initialize 方法

现在扩展 Initialize 方法, 使它在 web.config 文件的提供程序声明中读取定制特性及其关联值。在 XmlMembershipProvider 类的类架构中, 注意 Initialize 方法没有包含在可选项的列表中。

在第一次初始化提供程序时, 会调用 Initialize 方法。不需要重写这个方法, 因此, 在类架构的声明中没有包含它。为了把 Initialize 方法放在 XmlMembershipProvider 类中, 只需在类中输入 public override, 然后通过 IntelliSense 打开 Initialize 方法, 如图 15-5 所示。



图 15-5

以这种方式将 `Initialize` 方法放在类中是非常简单的操作。从 IntelliSense 的列表中选择 `Initialize` 方法，按下 `Enter` 键，就会在代码中显示该方法的基本构造代码，如程序清单 15-13 所示。

程序清单 15-13 `Initialize` 方法的开头部分

```
public override void Initialize(string name,
    System.Collections.Specialized.NameValueCollection config)
{
    base.Initialize(name, config);
}
```

`Initialize` 方法带有两个参数。第一个参数是参数名，第二个参数是 `web.config` 文件的提供程序声明中的名/值对集合。该集合包括所有的特性及其值，例如 `xmlUserDatabaseFile` 特性及包含用户信息的 XML 文件名。使用 `config` 可以访问这些已定义好的值。

在 `XmlFileProvider` 实例中，包含了 `applicationName` 特性和 `xmlUserDatabaseFile` 特性，如程序清单 15-14 所示。

程序清单 15-14 扩展 `Initialize()` 方法

```
public override void Initialize(string name,
    System.Collections.Specialized.NameValueCollection config)
{
    base.Initialize(name, config);

    _AppName = config["applicationName"];

    if(String.IsNullOrEmpty(_AppName))
    {
        _AppName = "/";
    }

    _FileName = config["xmlUserDatabaseFile"];

    if(String.IsNullOrEmpty(_FileName))
    {
        _FileName = "/App_Data/Users.xml";
    }
}
```

上面的代码除了使用 `MyBase.Initialize` 方法执行初始化操作之外，还使用 `config` 检索 `applicationName` 特性和 `xmlUserDatabaseFile` 特性的值。无论如何，都应首先检查特性值是否为空。如果在 `web.config` 文件的提供程序声明中没有指定这些特性，就使用 `String.IsNullOrEmpty` 方法给它们赋予默认值，在 `XmlFileProvider` 实例中就是这种情况。因为实际上没有声明 `XmlFileProvider` 声明中的 `applicationName` 特性，所以把默认值 `"/` 赋予它。

对于 `xmlUserDatabaseFile` 特性，如果没有在 `web.config` 文件中给它赋值，提供程序就会在 `App_Data` 文件夹中查找 XML 文件 `Users.xml`。

3. 验证用户

成员资格提供程序的一项很重要的功能就是验证用户(即检查用户的身份)，通过 ASP.NET 的

Login 服务器控件实现用户验证。这个控件利用了 `Membership.ValidateUser` 方法，而该方法又利用了 `XmlMembershipProvider` 类的 `ValidateUser` 方法。

建立了 `Initialize` 方法和私有变量后，就可以为提供程序编写一些功能。`ValidateUser` 方法的实现代码如程序清单 15-15 所示。

程序清单 15-15 实现 `ValidateUser` 方法

```
public override bool ValidateUser(string username, string password)
{
    if (String.IsNullOrEmpty(username) || String.IsNullOrEmpty(password))
    {
        return false;
    }

    try
    {
        ReadUserFile();

        MembershipUser mu;

        if (_MyUsers.TryGetValue(username.ToLower(), out mu))
        {
            if (mu.Comment == password)
            {
                return true;
            }
        }

        return false;
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message.ToString());
    }
}
```

`ValidateUser` 方法带有两个参数：用户名和密码(它们的类型都是 `String`)。`ValidateUser` 方法返回的值是 `Boolean` 类型——`True` 或 `False`，表示验证过程是否成功完成。

在 `ValidateUser` 方法中，首先进行检查，以确定是否在请求中遗漏了用户名或密码。如果在请求中遗漏了某一项，就返回 `False` 值。

如果 XML 文件中包含了用户名和密码，就进入 Try Catch 块。将 XML 文件中的用户信息读入 `_MyUsers` 变量的过程是由 `ReadUserFile` 方法完成的。稍后探讨这个方法，但注意 `_MyUsers` 变量是 `Dictionary` 泛型类的一个实例。其关键是用户名的小写字符串值，该值的类型是 `MembershipUser`，这是通过成员资格系统提供的类型。

使用 XML 文件中的所有用户填充 `_MyUsers` 对象后，就创建了一个 `MembershipUser` 实例。这个对象是 `TryGetValue` 操作的结果。`MembershipUser` 不包含用户的密码，因此，`ReadUserFile` 方法把用户的密码赋予 `MembershipUser` 类的 `Comment` 属性。如果在字典集合中找到了用户名，就将该 `MembershipUser` 实例的密码与 `Comment` 属性的值进行比较。如果它们相同，`ValidateUser` 方法的返回值就是 `True`。

可以看出，这个方法依赖于 `ReadUserFile` 方法的结果。下面就讨论 `ReadUserFile` 方法。

4. 建立 ReadUserFile 方法

ReadUserFile 方法读取 XML 文件的内容, 而 XML 文件包含应用程序的所有用户。这是一个定制方法, 其工作在 ValidateUser 方法的外部完成。也就是说, ReadUserFile 方法可以在其他方法(如 GetAllUsers 方法)中重用。ReadUserFile 方法只是读取 XML 文件的内容, 把所有的用户放在 `_MyUsers` 变量中, 如程序清单 15-16 所示。

程序清单 15-16 使用 ReadUserFile()方法获取应用程序的所有用户

```
private void ReadUserFile()
{
    if(_MyUsers == null)
    {
        lock(this)
        {
            _MyUsers = new Dictionary<string, MembershipUser>();
            var query = from users in
                        XElement.Load(
                            HostingEnvironment.MapPath(_FileName)).Elements("User")
                        select users;

            foreach(var user in query)
            {
                MembershipUser mu = new MembershipUser(Name,
                    user.Element("Username").Value,
                    null,
                    user.Element("Email").Value,
                    String.Empty,
                    user.Element("Password").Value,
                    true,
                    false,
                    DateTime.Parse(user.Element("DateCreated").Value),
                    DateTime.Now,
                    DateTime.Now,
                    DateTime.Now,
                    DateTime.Now);

                _MyUsers.Add(mu.UserName.ToLower(), mu);
            }
        }
    }
}
```



为了运行代码, 需要导入 `System.Xml`、`System.Xml.Linq` 和 `System.Web.Hosting` 名称空间。

ReadUserFile 方法首先在所运行的线程上锁定要执行的操作。这是 ASP.NET 的独特功能。在编写自己的提供程序时, 一定要使用线程安全的代码。对于在 ASP.NET 中编写的大多数项, 如 `HttpModule` 或 `HttpHandler`(详见第 30 章), 都不需要使用线程安全的代码。这些项可能在多个线程

上有多个请求，每个向 `HttpModule` 或 `HttpHandler` 发出请求的线程都有这些项的唯一实例。

与 `HttpHandler` 不同，ASP.NET 应用程序仅创建和使用提供程序的单个实例。如果向应用程序发出多个请求，所有这些线程都将试图访问应用程序所包含的这个唯一的提供程序实例。因为多个请求有可能同时访问提供程序实例，所以应以线程安全的方式创建提供程序。在执行文件 I/O 操作等任务时，使用锁定操作就可以达到上述目的。为了锁定访问，可以在 `ReadUserFile()` 方法中使用 `lock` 语句。

这段构造代码的优点是在应用程序中只运行单个提供程序实例。使用 XML 文件的内容填充 `_MyUsers` 对象后，就不需要重新填充该对象。给请求者发送响应后，提供程序实例不会消失，而是包含在内存中，供多个请求使用。这就是在读取 XML 文件之前检查 `_MyUsers` 是否包含值的原因。

如果 `_MyUsers` 为空，就使用 `XmlDocument` 对象获取文档中的每个 `<User>` 元素。对于文档中的每个 `<User>` 元素，把值赋予一个 `MembershipUser` 对象。该 `MembershipUser` 对象带有如下参数：

```
public MembershipUser (
    string providerName,
    string name,
    Object providerUserKey,
    string email,
    string passwordQuestion,
    string comment,
    bool isApproved,
    bool isLockedOut,
    DateTime creationDate,
    DateTime lastLoginDate,
    DateTime lastActivityDate,
    DateTime lastPasswordChangedDate,
    DateTime lastLockoutDate
)
```

在这段构造代码中，我们并没有为每一项提供值，而是使用 `XmlNode` 对象从 XML 文件中获取构造代码需要的值。在 `MembershipUser` 对象中填充需要的值后，就使用下面的语句将这些项添加到 `_MyUsers` 对象中：

```
_MyUsers.Add(mu.UserName.ToLower(), mu)
```

建立了 `ReadUserFile()` 方法后，就可以在 `ValidateUser()` 方法和其他方法中使用它。一旦填充 `_MyUsers` 集合，就不需要再次填充它，其他方法仍然可以使用它。下面探讨如何在 ASP.NET 应用程序中使用前面演示的提供程序。

15.4.7 使用 `XmlMembershipProvider` 进行用户登录

如果读者按照本章前面的步骤完成了示例，使用 `XmlMembershipProvider` 类就不需要做太多的工作。现在应有一个 XML 数据文件，它包含了应用程序的所有用户(关于这个文件请参阅程序清单 15-7)，在应用程序的 `web.config` 文件中还包含 `XmlFileProvider` 的声明(对 `web.config` 文件的修改请参阅程序清单 15-8)。当然，另一个必须具有的项是应用程序的 `App_Code` 文件夹中的 `XmlMembershipProvider.cs` 类。但是，如果把提供程序建立为类库，就要确保在 ASP.NET 应用程序中正确引用所创建的 DLL(即 DLL 位于 `Bin` 文件夹中)。有了这些项后，使用提供程序就非常简单。

下面举一个简单的例子。创建 `Default.aspx` 页面，其中只包含文本 “You are authenticated!”

接着，创建 `Login.aspx` 页面，在该页面上拖放一个 `Login` 服务器控件。除了放置该控件外，不

需要再对 Login.aspx 页面做其他修改。用户现在就可以登录到应用程序。



有关成员资格系统的详细信息，可参阅第 19 章，其中包括成员资格系统提供的各个服务器控件的介绍。

在微型 ASP.NET 应用程序中有了这两个文件后，就要对 web.config 文件进行一些小的修改，以允许使用窗体验证方式，拒绝所有匿名用户浏览任意页面。相关的代码如程序清单 15-17 所示。

程序清单 15-17 在 web.config 文件中拒绝匿名用户浏览应用程序

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.5" />
    <httpRuntime targetFramework="4.5" />
    <authentication mode="Forms"/>
    <authorization>
      <deny users="?" />
    </authorization>
    <!-- Other settings removed for clarity -->
  </system.web>
</configuration>
```

现在运行 Default.aspx 页面，会立即转向 Login.aspx 页面(应在应用程序中创建这个文件，其中只包含一个 Login 服务器控件)。在该页面上，输入 XML 文件中的一组用户名和密码。该操作很简单！

对于 ASP.NET 4.5 中的基于提供程序的模型，一项优点是使用提供程序的控件并不知道对底层提供程序的这些修改。在这个例子中，使用只读的 XML 提供程序替代删除的默认 SqlMembershipProvider，而 Login 服务器控件对此一无所知。当终端用户单击 Login 服务器控件中的 Log In 按钮时，该控件仍然利用 Membership.ValidateUser 方法，而该方法利用了刚才创建的 XmlMembershipProvider。显然，这是个很强大的模型。

15.5 扩展预定义的提供程序

除了从某个抽象的基类(如 MembershipProvider)建立自己的提供程序之外，还可以扩展 ASP.NET 中预定义的提供程序。

例如，有时可能需要将成员资格系统、角色管理系统与 SQL Server 结合使用，但想改变默认提供程序(SqlMembershipProvider 或 SqlRoleProvider)的底层工作方式。如果要使用底层的数据存储，而该数据存储已由某个提供程序使用，修改提供程序的工作方式就比从头开始建立全新的提供程序好得多。

扩展预定义的提供程序的另一个优点是，不需要重写提供程序的所有内容。如果只修改内置提供程序的一种行为，只需要重写几个方法，就可以使修改过的操作简单、快速地应用到应用程序中。

下面就扩展一个内置的提供程序，以改变该提供程序的底层功能。

15.5.1 用新的 LimitedSqlRoleProvider 提供程序限制角色功能

假设要在 ASP.NET 应用程序中使用角色管理系统，并为该系统使用 SQL Server 后台数据库。我们希望限制开发人员可以在其应用程序中创建的角色，并去除在系统中将用户添加到某个角色的功能。

本例不是以 RoleProvider 抽象类为基础建立角色提供程序，而是从 SqlRoleProvider 中派生提供程序，再修改创建角色和给角色添加用户的几个方法。

与前面一样，该例在 App_Code 文件夹的应用程序中创建新的提供程序。但实际上，如果要在公司范围内使用这个提供程序，就应创建一个类库项目，这样开发小组就可以使用 DLL，而不是使用一个可修改的类文件。

在 App_Code 文件夹中创建类文件 LimitedSqlRoleProvider.cs，这个类要继承 SqlRoleProvider，因此其结构如程序清单 15-18 所示。

程序清单 15-18 LimitedSqlRoleProvider 类的开头部分

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Security;

namespace Chapter15_CustomProviders_CS.App_Code
{
    public class LimitedSqlRoleProvider : SqlRoleProvider
    {
    }
}
```

该类的创建与 XmlMembershipProvider 类的创建相似。在创建 XmlMembershipProvider 类时，可以使用 Visual Studio 建立整个类架构，以重写所有需要的方法和属性，使新类能够运行。在本例中，如果要在 Visual Studio 中完成这些操作，就会出错，因为本例没有使用抽象类。我们不需要重写很多方法和属性，因为本例继承的类是从抽象类派生的，只需重写需要使用的方法和属性即可。

要在 Visual Studio 中查看这些方法和属性，应输入 public override，IntelliSense 就会在下拉列表中显示可用的方法和属性，如图 15-6 所示。

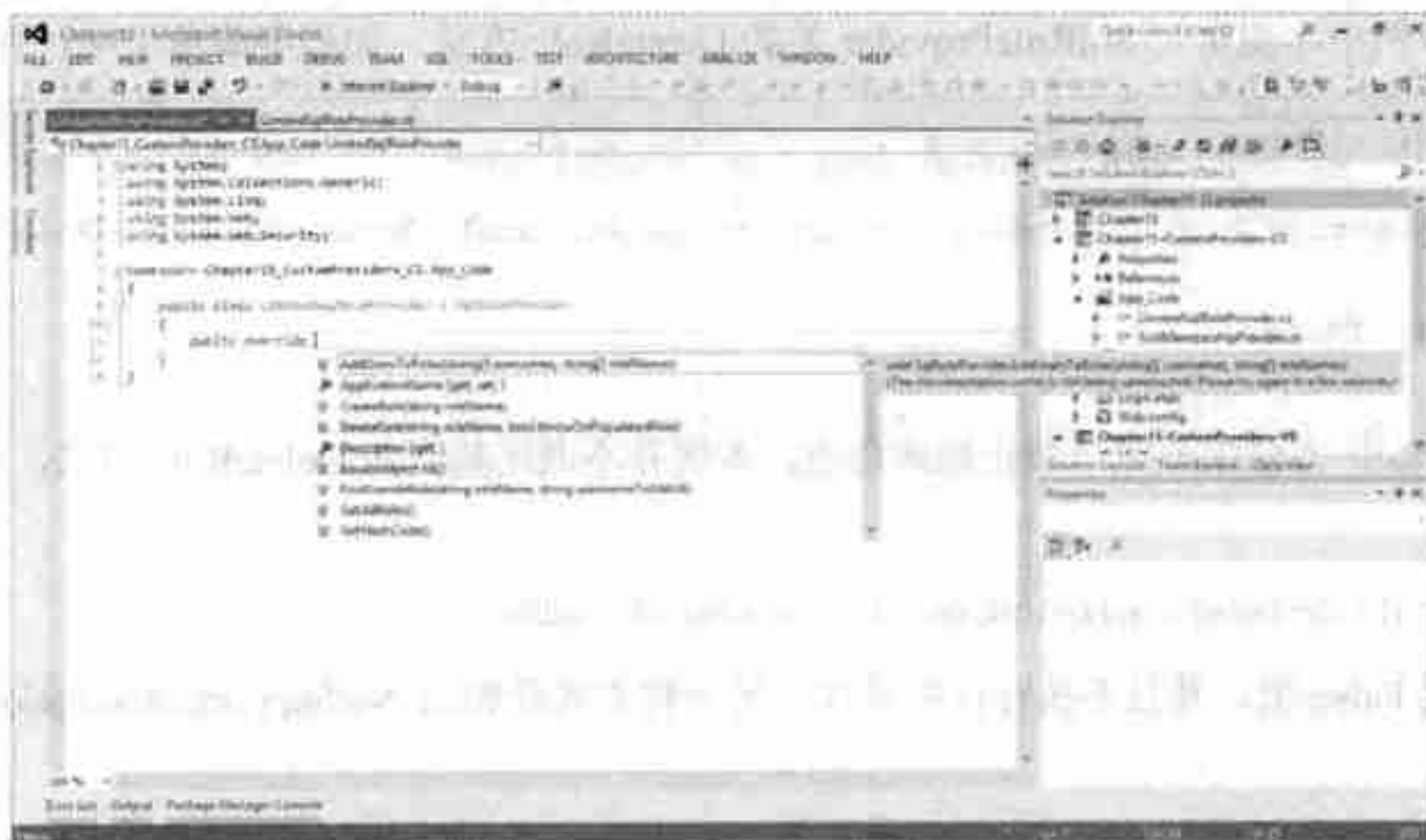


图 15-6

对于本例，只需重写 `CreateRole`、`AddUsersToRoles` 和 `DeleteRole` 方法，详见下面的内容。

1. CreateRole 方法

`SqlRoleProvider` 类的 `CreateRole` 方法允许开发人员给系统添加角色。这个方法只带有一个参数：角色名，这是一个字符串值。本例不允许开发人员随意创建角色，因此这个提供程序将开发人员能创建的角色限制为 `Administrator` 和 `Manager`。为此，`CreateRole` 方法的代码如程序清单 15-19 所示。

程序清单 15-19 在 `CreateUser` 方法中只允许创建 `Administrator` 和 `Manager` 角色

```
public override void CreateRole(string roleName)
{
    if(roleName == "Administrator" || roleName == "Manager")
    {
        base.CreateRole(roleName);
    }
    else
    {
        throw new
            ProviderException("Role creation limited to only Administrator and Manager");
    }
}
```



为了使这段代码能够运行，需要导入 `System.Configuration.Provider` 名称空间。

在这个方法中，首先进行检查，确定所创建的角色是否为 `Administrator` 或 `Manager`。如果不是，就抛出 `ProviderException` 异常，告诉开发人员允许创建的角色。

如果所创建的角色是 `Administrator` 或 `Manager`，就调用基类 `SqlRoleProvider` 的 `CreateRole` 方法。

2. DeleteRole 方法

如果只允许使用本提供程序的开发人员创建某些角色，那么我们可能不希望他们删除任何已创建的角色。本例中希望重写 `SqlRoleProvider` 类的 `DeleteRole` 方法，如程序清单 15-20 所示。

程序清单 15-20 禁用 `DeleteRole` 方法

```
public override bool DeleteRole(string roleName, bool throwOnPopulatedRole)
{
    return false;
}
```

在 `DeleteRole` 方法中，完全禁止删除角色。本例并不调用基类的 `DeleteRole` 方法，而是返回如下内容：

```
return this.DeleteRole(roleName, throwOnPopulatedRole);
```

此处返回 `False` 值，并且不执行任何操作。另一种方式是抛出 `NotSupportedException` 异常，如下所示：

```
throw new NotSupportedException();
```

3. AddUsersToRoles 方法

在可以重写的方法中，注意只有一个方法允许把任意多个用户添加到任意多个角色中。Roles 类的许多方法都映射了该方法。在 Roles 类中，注意 AddUserToRole 方法、AddUserToRoles 方法、AddUsersToRole 方法和 AddUsersToRoles 方法。这些方法都映射到基类 RoleProvider 的 AddUsersToRoles 方法。

例如，假设只允许开发人员将用户添加到 Manager 角色中，但不允许添加到 Administrator 角色中，就可以构建如程序清单 15-21 所示的方法。

程序清单 15-21 禁止把用户添加到某个特定角色中

```
public override void AddUsersToRoles(string[] usernames, string[] roleNames)
{
    foreach(string roleItem in roleNames)
    {
        if(roleItem == "Administrator")
        {
            throw new ProviderException("You are not authorized to add any users" +
                " to the Administrator role");
        }
    }

    base.AddUsersToRoles(usernames, roleNames);
}
```

这个重写方法遍历所提供的所有角色，如果字符串数组中包含的某个角色是 Administrator，就抛出 ProviderException 异常，通知开发人员不允许把用户添加到这个角色中。还可以对基类 RoleProvider 的 RemoveUsersFromRoles 方法采用相同的操作，但这里没有列出该方法。

15.5.2 使用新的 LimitedSqlRoleProvider 提供程序

有了新的提供程序后，就需要对 web.config 文件进行一些修改，以便在 ASP.NET 应用程序中使用这个提供程序。在 web.config 文件中为使用这个提供程序而添加的代码如程序清单 15-22 所示。

程序清单 15-22 在 web.config 文件中为使用这个提供程序进行一些修改

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <roleManager defaultProvider="LimitedProvider" enabled="true">
      <providers>
        <add connectionStringName="LocalSqlServer" applicationName="/"
          name="LimitedProvider"
          type="LimitedSqlRoleProvider" />
      </providers>
    </roleManager>
  </system.web>
</configuration>
```

必须为 defaultProvider 特性提供值，并在<providers>部分进一步定义该提供程序，才能在应用

程序中定义它。还需要将 `enabled` 特性设置为 `true`，才能激活该提供程序。在默认情况下，角色管理系统是禁用的。

使用 `<add>` 元素，可以添加使用了 `LimitedSqlRoleProvider` 类的提供程序实例。这个提供程序派生于 `SqlRoleProvider` 类，因此必须使用该提供程序需要的特性，如 `connectionStringName` 特性，它指向用于连接指定 SQL 实例的连接字符串。

有了新的 `LimitedSqlRoleProvider` 实例并在 `web.config` 文件中定义它之后，就可以按照往常的方式在应用程序中使用 `Roles` 类，但注意该类的操作方式与一般的 `SqlRoleProvider` 有所不同。

为了理解 `Roles` 类的操作方式，构建一个简单的 ASP.NET 页面，其中包含一个 `TextBox` 控件、一个 `Button` 控件和一个 `Label` 服务器控件。该页面如程序清单 15-23 所示。

程序清单 15-23 使用 `Roles.CreateRole` 方法

```
<%@ Page Language="C#" %>
<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        try
        {
            Roles.CreateRole(TextBox1.Text);
            Label1.Text = "Role successfully created.";
        }
        catch(Exception ex)
        {
            Label1.Text = ex.Message.ToString();
        }
    }
</script>
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>>Main Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        Role Name:<br />
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
        <br />
        <asp:Button ID="Button1" runat="server" Text="Create Role"
            OnClick="Button1_Click" /><br />
        <br />
        <asp:Label ID="Label1" runat="server"></asp:Label>
    </div>
    </form>
</body>
</html>
```

这个简单的 ASP.NET 页面允许用户在文本框中输入一个字符串，并使用这个字符串尝试创建一个新角色。注意只要创建的角色不是 `Administrator` 或 `Manager`，页面就会出错。因此，在调用

Roles.CreateRole 方法时，如果不遵循提供程序定义的规则，就会产生错误。实际上，运行这个页面并输入不是 Administrator 或 Manager 的角色，就会显示如图 15-7 所示的窗口。



图 15-7

为了说明这个提供程序的工作情况，创建另一个 ASP.NET 页面，它允许将用户添加到某个角色中。如前所述，可以使用许多方法实现该操作，但本例使用 Roles.AddUserToRole 方法，如程序清单 15-24 所示。

程序清单 15-24 尝试通过新的角色提供程序将用户添加到角色中

```
<%@ Page Language="C#" %>
<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        try
        {
            Roles.AddUserToRole(TextBox1.Text, TextBox2.Text);
            Label1.Text = "User successfully added to role";
        }
        catch (Exception ex)
        {
            Label1.Text = ex.Message.ToString();
        }
    }
</script>
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Main Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        Add the following user:<br />
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
        <br />
        To role:<br />
        <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox><br />
```

```
<br />
<asp:Button ID="Button1" runat="server" Text="Add User to Role"
    OnClick="Button1_Click" /><br />
<br />
<asp:Label ID="Label1" runat="server"></asp:Label>
</div>
</form>
</body>
</html>
```

在这个例子中，提供了两个文本框。第一个文本框要求输入用户名，第二个文本框要求输入角色，以便将用户添加到该角色中。按钮单击事件的处理代码使用了 `Roles.AddUserToRole` 方法。本章前面建立了提供程序，因此如果试图把用户添加到 Administrator 角色中，就会抛出错误，如图 15-8 所示。

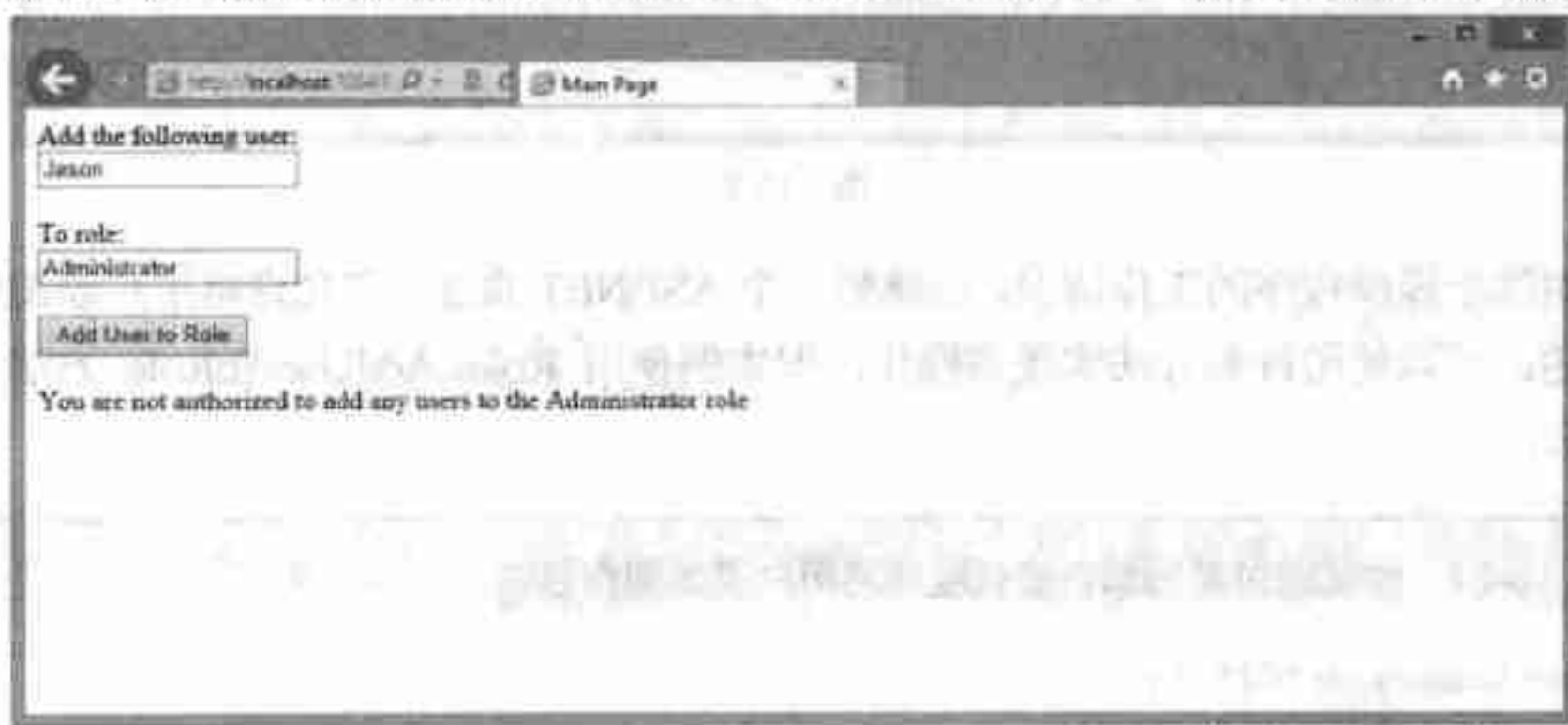


图 15-8

本例试图把 Jason 添加到 Administrator 角色中。当然，这会抛出错误，并返回在提供程序中定义的错误消息。

15.6 本章小结

本章和第 14 章都介绍了提供程序模型，以及它对目前创建的 ASP.NET 4.5 应用程序的意义。ASP.NET 提供了大量的提供程序，以便与 ASP.NET 中的许多系统进行通信，但可用的提供程序不止这些。我们还可以创建自己的提供程序，甚至扩展系统中已有提供程序的功能。

本章介绍了两类提供程序。首先创建了自己的提供程序，为的是使用成员资格系统，将用户数据保存在 XML 数据存储中，然后使用示例来说明如何扩展 `SqlRoleProvider` 类(ASP.NET 中已有的类)，从而改变这个提供程序的底层行为。

第 V 部分

ASP.NET 特性

- 第 16 章 使用母版页
- 第 17 章 站点导航
- 第 18 章 个性化
- 第 19 章 成员资格和角色管理
- 第 20 章 安全性

第 16 章

使用母版页

本章要点

- 编写母版页和内容页面
- 使用母版页指定默认内容
- 以编程方式指定母版页
- 嵌套母版页
- 适应不同浏览器的母版页
- 理解母版页的事件顺序
- 使用母版页和 ASP.NET AJAX

可视化继承是 ASP.NET 提供的一个重要功能,可使用它建立 Web 页面。该功能在 ASP.NET 2.0 中首次引入。实际上,可以创建一个模板页面,作为应用程序中任意数量 ASP.NET 内容页面的基础。这些模板页面称为母版页,可以使应用程序更容易建立和管理,从而提高效率。Visual Studio 2012 为母版页提供了完全的设计器支持,开发人员可以使用的功能也更丰富。例如,即使有嵌套的母版页,Visual Studio 也为页面在浏览器中的显示效果提供了所见即所得的真实视图。本章将详细介绍如何在应用程序中最大限度地使用母版页,首先将介绍母版页的优点。

16.1 需要母版页的原因

目前,大多数 Web 站点在整个应用程序或应用程序的大多数页面中都有有一些公共元素。例如,在 Reuters News 网站(www.reuters.com)的首页上,就可以看到整个 Web 站点都使用的一些公共元素。这些公共区域如图 16-1 所示。

在这个屏幕截图中,注意页面上的页眉区域、导航区域和页脚区域。实际上,在整个应用程序中,几乎每个页面都使用了这些相同的元素。甚至在母版页出现以前,我们也需要通过多种方式将这些元素放在每个页面上,但在大多数情况下,这么做比较困难。

一些开发人员简单地把这些公共区域的代码复制并粘贴到需要它们的每个页面上。这是可行的

方法，但相当麻烦——每次需要对应用程序的某个公共部分进行修改时，就必须在每个页面上重复这个修改，该操作非常枯燥，并且效率很低。



图 16-1

以前使用传统 ASP 时，一种流行的方法是把所有的公共部分都放在 include 文件中，然后把这个文件放在页面中，如下所示：

```
<!-- #include virtual="/myIncludes/header.asp" -->
```

使用 include 文件存在的问题是，必须考虑在 include 头文件中新打开的 HTML 标记。这些标记必须在主文档或 include 页脚文件中关闭。通常很难使所有的 HTML 标记有序，尤其是在多人完成一个项目的情况下更是如此。Web 页面有时会显示奇怪的结果，因为其中包含不正确或不存在的起始标记或结束标记。也很难在可视化的设计器中使用 include 文件。使用 include 文件不允许开发人员像在浏览器中那样查看整个页面。最后，开发人员开发了各个部分中的页面，并希望这些部分可以按照计划组合在一起。开发人员的很多时间都用在了 include 文件中起始和结束标记的精雕细琢上。

自 2000 年推出 ASP.NET 1.0 后，开发人员开始使用用户控件来封装 Web 页面的公共部分。例如，建立一个 Web 页面，把页眉、导航和页脚部分的代码拖放到需要它们的每个页面上，就可以在

该 Web 页面上包含这些部分。

这种技术是可行的，但也存在一些问题。在 Visual Studio 2005 和 ASP.NET 2.0 推出以前，用户控件导致的问题与 include 文件存在的问题类似。在 Web 页面的设计视图中工作时，页面的公共区域在 Visual Studio .NET 2002 和 2003 中只显示为灰框。这将很难建立页面：在浏览器中编译和运行页面之前，不能显示正在建立的页面。用户控件也存在与 include 文件相同的问题：必须在两个不同的文件中匹配起始和结束 HTML 标记。我们更喜欢用户控件，而不是 include 文件，但用户控件并不是在整个应用程序中使用的最佳模板。Visual Studio 通过在设计视图中显示用户控件的内容来解决一些问题。如果仅在 Web 页面上包含较少的公共部分，使用用户控件就比较理想；但在处理较大的页面模板时，它们仍然比较麻烦。

针对 include 文件 and 用户控件存在的问题，ASP.NET 小组开发出了母版页，这是把模板应用于应用程序的一种新方式。它们将开发人员解决该问题的方式颠倒过来。母版页位于所开发的页面外部，而用户控件位于页面的内部。这些母版页清晰地划分各个页面的公共区域和每个页面独有的内容区域。母版页使用起来很简单，并且很有趣。下面查看 ASP.NET 中有关母版页的一些基础知识。

16.2 母版页的基础知识

母版页是提供模板的一种简单方式，该模板可以由应用程序中的任意多个 ASP.NET 页面使用。在使用母版页时，要创建母版页文件，母版页文件是由子页面或内容页面引用的模板。母版页使用 .master 文件扩展名；而内容页面使用 .aspx 文件扩展名，并且在文件的 Page 指令中声明。

把需要在模板中共享的内容放在 .master 文件中，包括 Web 应用程序使用的页眉、导航和页脚区域。内容页面包含除母版页元素之外的其他页面元素。在运行时，ASP.NET 引擎会把这些元素合并到页面上，显示给终端用户。图 16-2 说明了这个过程。

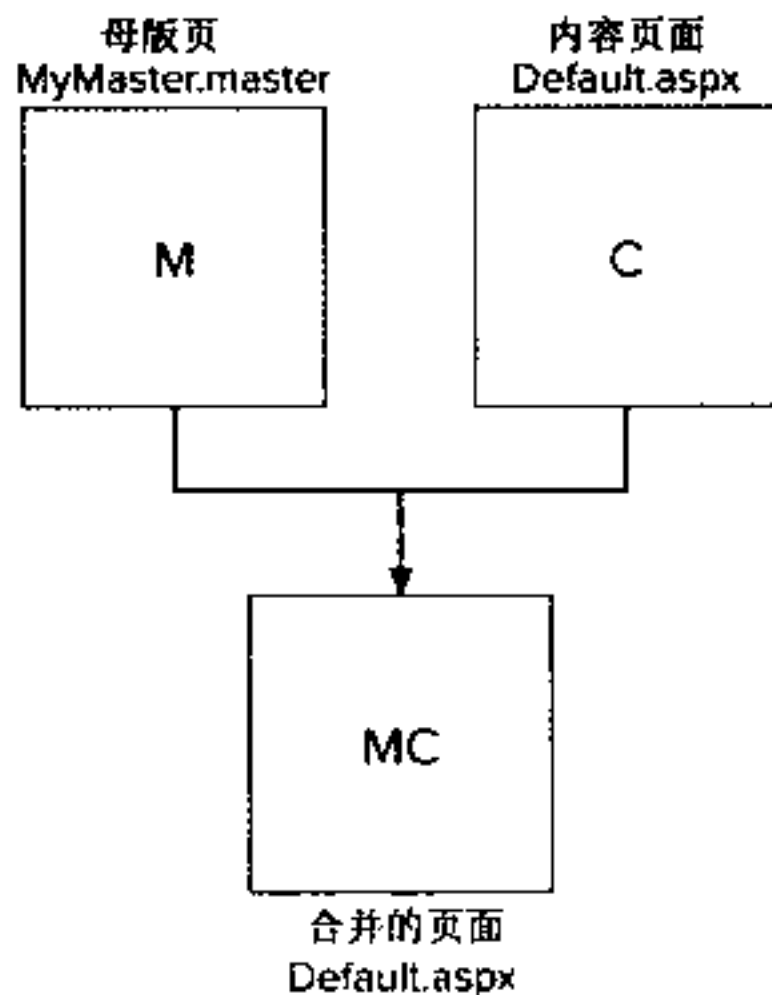


图 16-2

使用母版页的一个优点是，在创建内容页面时可以在 IDE 中看到模板。如果在处理页面时可以看到整个页面，就很容易开发出使用模板的内容页面。在处理内容页面时，所有模板项都以灰色显示，不能编辑。可以修改的项会清晰地显示在模板中。这些可处理的区域称为内容区域，最初是在

母版页中定义的。在母版页中，指定了内容页面可以使用的页面区域。在母版页中可以有多个内容区域，图 16-3 显示了带两个内容区域的母版页。

在图 16-3 所示的屏幕截图中，页面上有两个已定义的区域，它们就是内容区域。在页面的设计视图中，内容区域用浅色的虚线框表示，虚线框代表 `ContentPlaceHolder` 控件。如果把鼠标停放在内容区域，就会在控件的上方显示该控件的名称(但比较浅)。该停放操作也显示在图 16-3 中。

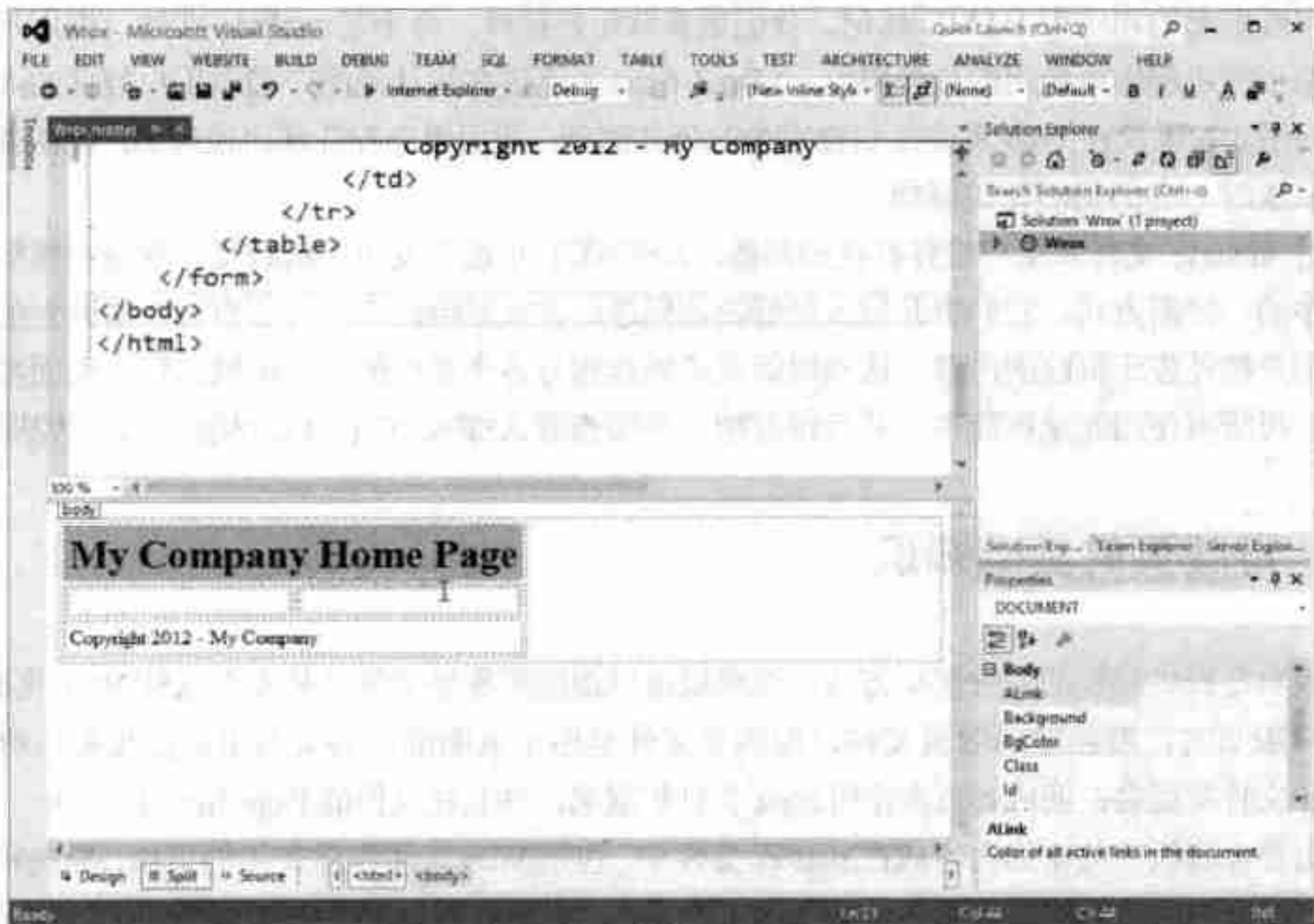


图 16-3

许多公司和组织发现，使用母版页是很理想的方式，因为该技术清晰地建立了业务需求模型。许多公司都在其内联网上使用相同的外观和操作方式，这些公司现在可以为各个部门提供 `master` 文件，在创建内联网的部门部分时使用该文件。这个功能使公司很容易在整个内联网上保持一致的外观和操作方式。

16.3 编写母版页

现在看看如何建立如图 16-3 所示的母版页。可以在任意基于文本的编辑器中创建母版页，但建议使用 Visual Studio 2012 或 Visual Studio Express 2012 for Web。本章将介绍如何使用 Visual Studio 2012 完成该操作。

将母版页添加到项目中的方式与常规的 `.aspx` 页面相同：在给应用程序添加新文件时，选择 `Master Page` 选项，如图 16-4 所示。

因为母版页与其他 `.aspx` 页面一样，所以 `Add New Item` 对话框也允许选择使用内联编码模型来编写母版页，或者把母版页的代码放在单独的文件中。没有将服务器代码放在单独的文件中，这意味着要给创建的页面使用内联编码模型。这个选项会创建一个 `master` 页面。选择把代码放在单独的文件中，这意味着要对创建的页面使用新的隐藏代码模型。选中 `“Place code in separate file”` 复选框

会创建一个 .master 页面和一个相关联的 .master.cs 文件。还可以选择 **Select master page** 选项，把母版页嵌入另一个母版页中，这个选项详见本章后面的内容。



图 16-4

使用内联编码模型的母版页示例如程序清单 16-1 所示。

程序清单 16-1 母版页示例

```
<%@ Master Language="C#" %>
<script runat="server">
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>My Company Master Page</title>
  <asp:ContentPlaceHolder id="head" runat="server">
  </asp:ContentPlaceHolder>
</head>
<body>
  <form id="form1" runat="server">
    <table cellpadding="3" border="1">
      <tr style="background:silver">
        <td colspan="2">
          <h1>My Company Home Page</h1>
        </td>
      </tr>
      <tr>
        <td>
          <asp:ContentPlaceHolder ID="ContentPlaceHolder1"
            runat="server">
          </asp:ContentPlaceHolder>
        </td>
      </tr>
    </table>
  </form>
</body>
</html>
```

```

        <td>
            <asp:ContentPlaceholder ID="ContentPlaceholder2"
                runat="server">
            </asp:ContentPlaceholder>
        </td>
    </tr>
    <tr>
        <td colspan="2">
            Copyright 2012 - My Company
        </td>
    </tr>
</table>
</form>
</body>
</html>

```

这是一个简单的母版页。在 Visual Studio 2012 中创建母版页的一个优点是可以在源代码视图中处理母版页，但也可以切换到设计视图中，像创建其他 ASP.NET 页面那样创建母版页。

首先查看母版页的代码。第一行是指令：

```
<%@ Master Language="C#" %>
```

这里没有像一般的.aspx 页面那样使用 Page 指令，而是使用针对母版页的 Master 指令。这个母版页只使用特性 Language。本例中 Language 特性的值是 C#。

可以像编写其他.aspx 页面那样编写母版页的其他部分。可以使用服务器控件、原始的 HTML 和文本、图像、事件或其他用于.aspx 页面的内容。也就是说，母版页也可以有 Page_Load 事件和其他适当的事件。

注意，在程序清单 16-1 所示的代码中使用了<asp:ContentPlaceholder>服务器控件。这个控件定义了模板的区域，在该区域中，内容页面可以放置其内容：

```

<tr>
    <td>
        <asp:ContentPlaceholder ID="ContentPlaceholder1" runat="server">
        </asp:ContentPlaceholder>
    </td>
    <td>
        <asp:ContentPlaceholder ID="ContentPlaceholder2" runat="server">
        </asp:ContentPlaceholder>
    </td>
</tr>

```

在这个母版页中，定义了内容页面可以放置其内容的两个区域。该母版页包含页眉区域和页脚区域。它还在页面中定义了两个区域，任何继承该页面的内容页面都可以在这两个区域中放置其内容。下面查看内容页面如何使用这个母版页。

16.4 编写内容页面

应用程序中有了母版页后，就可以把这个新模板用于应用程序的内容页面。在 Solution Explorer 中右击应用程序并选择 Add New Item 命令，以在应用程序中创建新的内容页面。

要创建内容页面或把这个母版页用作模板页面,可以在 Add New Item 对话框的选项列表中选择 Web Form 选项,如图 16-5 所示。但是,没有创建常用的 Web 窗体,而是选中 Select master page 复选框。这样,以后就可以把这个 Web 窗体关联到某个母版页了。

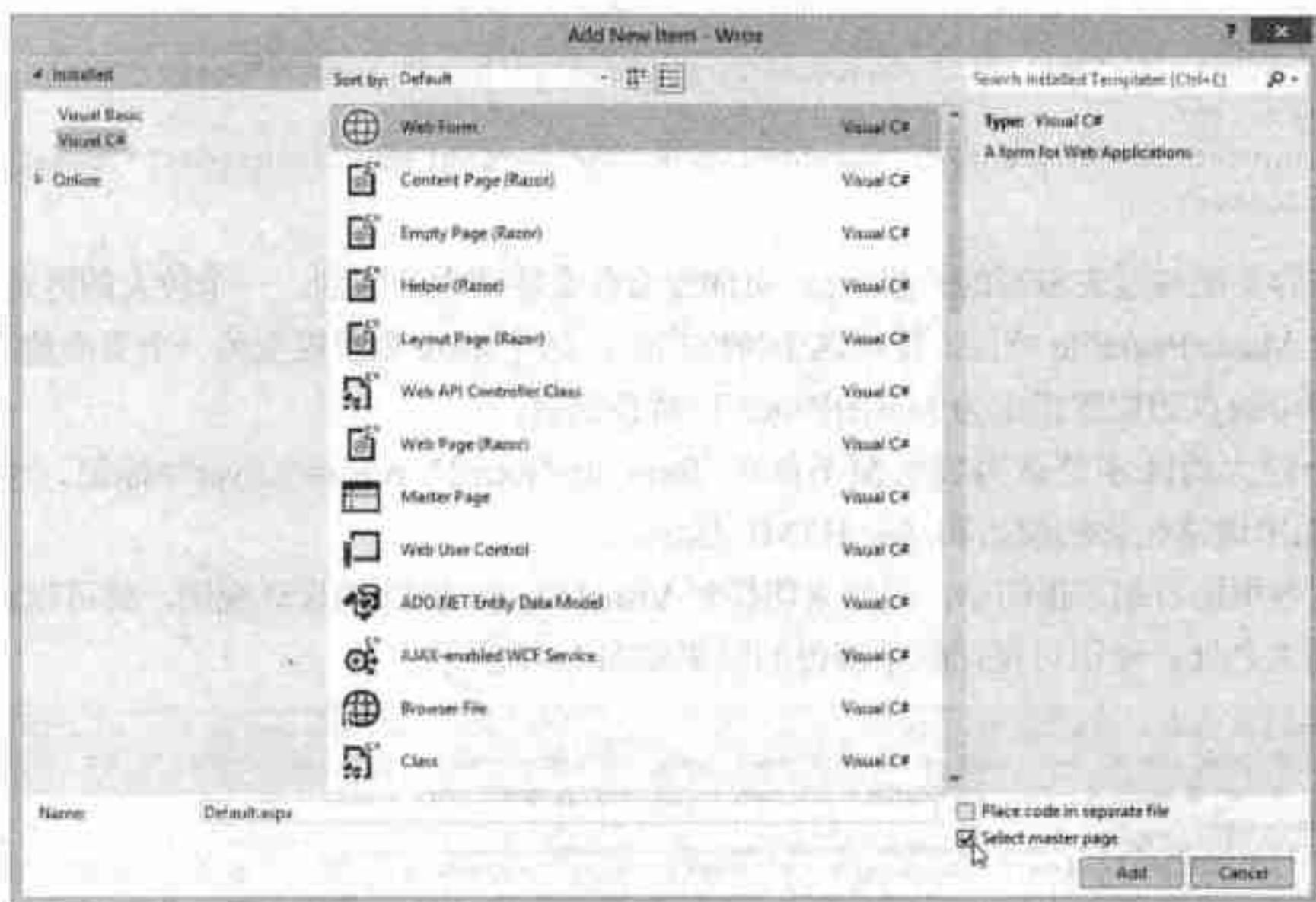


图 16-5

对内容页面命名后,单击 Add New Item 对话框中的 Add 按钮,就会显示 Select a Master Page 对话框,如图 16-6 所示。

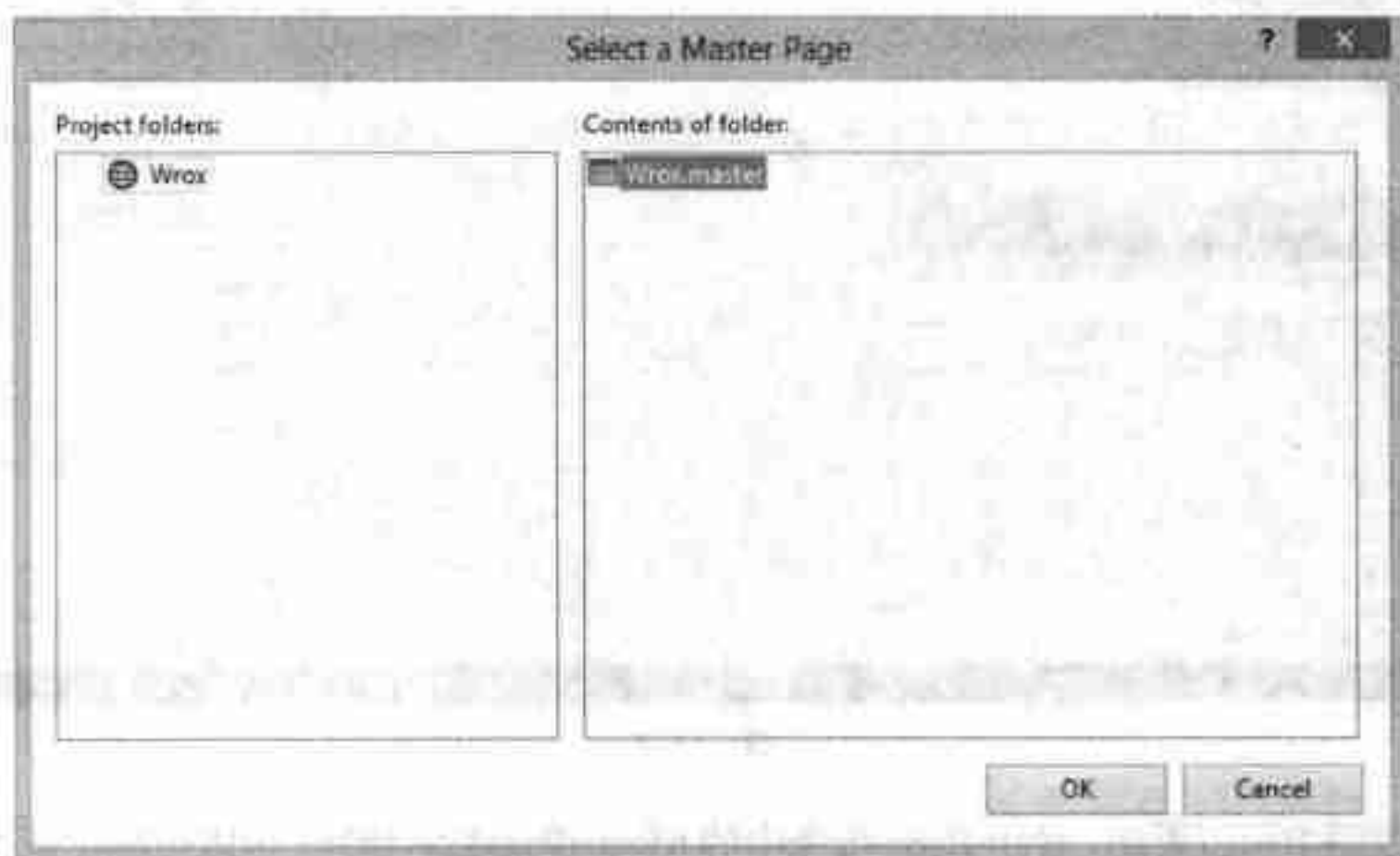


图 16-6

这个对话框可以选择用于建立内容页面的母版页,从应用程序包含的可用母版页中选择。对于这个例子,选择在程序清单 16-1 中创建的新母版页,单击 OK 按钮。这将创建一个内容页面,这是一个简单的.aspx 页面,文件中只包含几行代码,如程序清单 16-2 所示。

程序清单 16-2 创建的内容页面

```

<%@ Page Language="C#" MasterPageFile="~/Wrox.master" Title="" %>
<script runat="server">
</script>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server">
</asp:Content>

```

这个内容页面与过去编写的一般.aspx 页面没有什么特别大的区别。一个较大的区别是在 Page 指令中包含 MasterPageFile 特性。使用这个特性指示, 这个.aspx 页面根据另一个页面建立其控件。应用程序中母版页的位置指定为 MasterPageFile 特性的值。

另一个较大的区别是该内容页面不包含<form id="form1" runat="server">标记, 也不包含一般.aspx 页面中通常包含的起始和结束 HTML 标记。

这个内容页面看起来很简单, 但如果切换到 Visual Studio 2012 的设计视图, 就可以看出使用内容页面的强大之处。使用可视化继承获得的结果如图 16-7 所示。

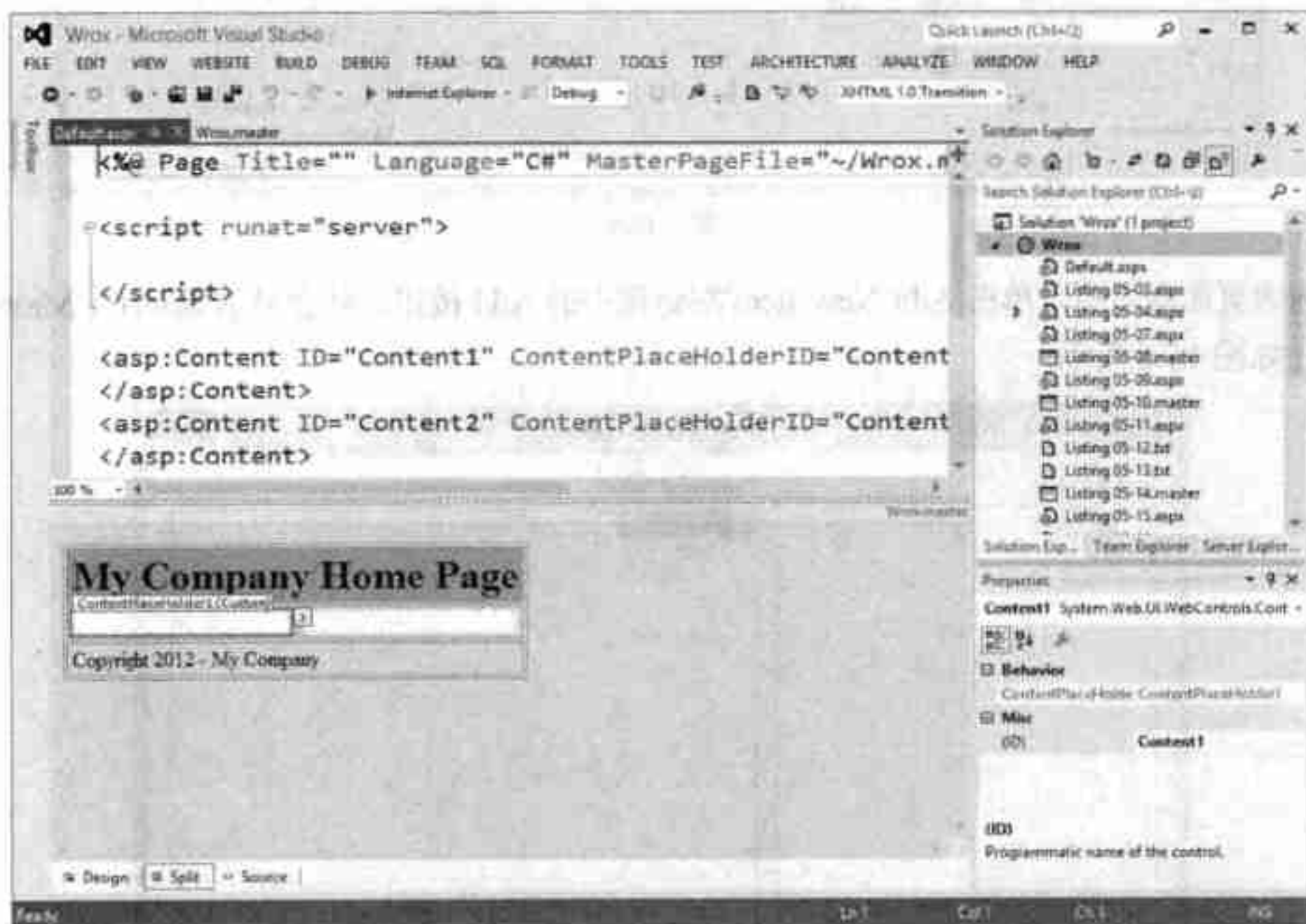


图 16-7

从这个屏幕截图中可以看出, 使用 Page 指令中的 MasterPageFile 特性, 可以可视化地继承 Wrox.master 文件中的所有内容。在 Visual Studio 的设计视图中还可以看到, 引用的母版页名称显示在设计视图页面的右上角。如果单击表示继承自母版页的灰色区域, 光标就会显示为不允许执行操作的图标, 如图 16-8 所示(光标在标题中的单词“Page”上)。



图 16-8

在母版页中定义的所有公共区域都显示为灰色，而在母版页中使用`<asp:ContentPlaceHolder>`服务器控件指定的内容区域则清晰显示，并可以在内容页面中用于其他内容。可以在这些定义好的内容区域中添加任何内容，就好像在处理常规的.aspx 页面一样。把这个.master 页面用于内容页面的例子请参见程序清单 16-3。

程序清单 16-3 使用 Wrox.master 的内容页面

```
<%@ Page Language="C#" MasterPageFile="~/Wrox.master" %>
<script runat="server">
    protected void Button1_Click(object sender, System.EventArgs e)
    {
        Label1.Text = "Hello " + HttpUtility.HtmlEncode(TextBox1.Text) + "!";
    }
</script>
<asp:Content ID="Content1" ContentPlaceHolderId="ContentPlaceHolder1"
    runat="server">
    <b>Enter your name:</b><br />
    <asp:Textbox ID="TextBox1" runat="server" />
    <br />
    <br />
    <asp:Button ID="Button1" runat="server" Text="Submit"
        OnClick="Button1_Click" /><br />
    <br />
    <asp:Label ID="Label1" runat="server" Font-Bold="True" />
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderId="ContentPlaceHolder2"
    runat="server">
    <asp:Image ID="Image1" runat="server" ImageUrl="wrox.gif" />
</asp:Content>
```

我们能够立刻看出一些区别。如前所述，这个页面没有`<form id="form1" runat="server">`标记，也没有任何起始和结束 HTML 标记。这些标记都没有包含进来，因为它们位于母版页中。还要注意服务器控件`<asp:Content>`：

```
<asp:Content ID="Content1" ContentPlaceHolderId="ContentPlaceHolder1" runat="server">
    ...
</asp:Content>
```

`<asp:Content>`服务器控件是定义好的内容区域，映射为母版页上的特定服务器控件`<asp:ContentPlaceHolder>`。在这个例子中，`<asp:Content>`服务器控件映射为母版页上 ID 为 `ContentPlaceHolder1` 的`<asp:ContentPlaceHolder>`服务器控件。在内容页面上，不必指定内容的位置，因为已经在母版页中定义该位置。因此，只需把适当的内容放在所提供的内容部分，让母版页完成大部分工作。

与处理一般的.aspx 页面一样，可以为内容页面创建事件处理程序。在本例中，只使用了一个事件处理程序，即终端用户提交窗体时的按钮单击事件处理程序。所创建的.aspx 页面包含母版页和内

容页面, 如图 16-9 所示。

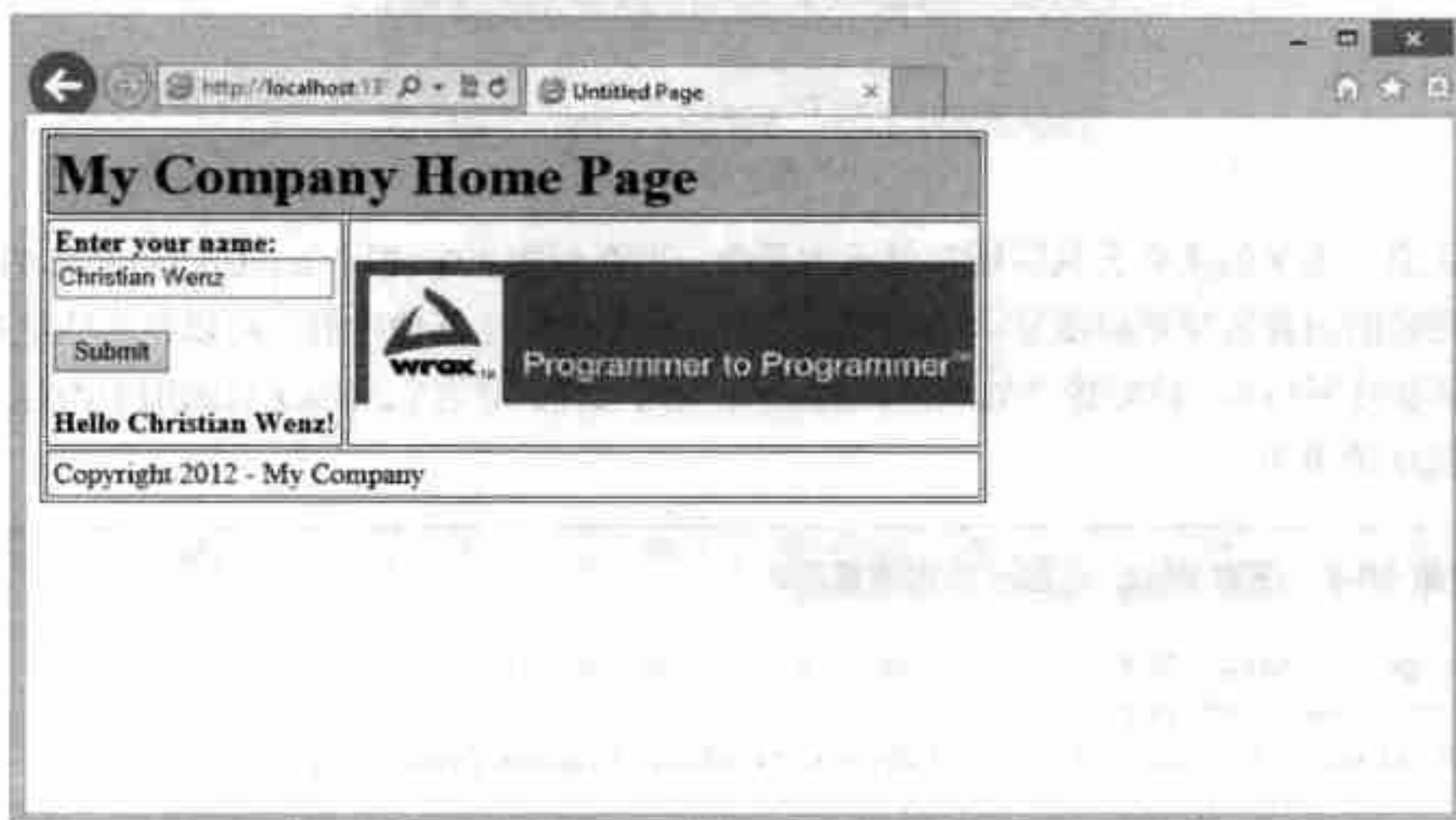


图 16-9

16.4.1 混合页面类型和语言

一个有趣的方面是: 在使用母版页时, 既不使用特定的编码模型(内联或隐藏代码模型), 也不使用特定的语言。因此, 可以在应用程序中自由混合这些元素, 它们会工作得很好。

可以使用前面创建的母版页, 使用内联编码模型创建该母版页, 然后使用隐藏代码模型创建内容页面。程序清单 16-4 显示了一个内容页面, 它使用的 Web 窗体是通过隐藏代码创建的。

程序清单 16-4 使用隐藏代码模型的内容页面

```
<%@ Page Language="C#" MasterPageFile="~/Wrox.master" AutoEventWireup="false"
    CodeFile="MyContentPage.aspx.cs" Inherits="MyContentPage" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1"
    runat="server">
    <b>Enter your name:</b><br />
    <asp:Textbox ID="TextBox1" runat="server" />
    <br />
    <br />
    <asp:Button ID="Button1" runat="server" Text="Submit" /><br />
    <br />
    <asp:Label ID="Label1" runat="server" Font-Bold="True" />
</asp:Content>
<asp:Content ID="Content3" ContentPlaceHolderID="ContentPlaceHolder2"
    runat="server">
    <asp:Image ID="Image1" runat="server" ImageUrl="wrox.gif" />
</asp:Content>
public partial class MyContentPage : System.Web.UI.Page
{
    protected void Button1_Click(object sender, System.EventArgs e)
```



```

{
    Label1.Text = "Hello " + HttpUtility.HtmlEncode(TextBox1.Text) + "!";
}
}

```



尽管母版页使用的是内联编码模型，但仍然可以轻松地使用隐藏代码模型创建内容页面(如程序清单 16-4 所示的页面)。这些页面仍然将会很好地工作。

不仅可以在使用母版页时混合编码模型，还可以给母版页或内容页面混合使用编程语言。使用 C# 语言建立母版页，并不意味着必须在使用该母版页的所有内容页面中都使用 C#。也可以使用 Visual Basic 建立内容页面。例如，使用 C# 创建一个母版页，它使用 Page_Load 事件处理程序。然后，用 Visual Basic 创建内容页面。完成这些操作后，运行页面，它会工作得很好。这就是说，即使母版页使用的是某种 .NET 语言，通过该母版页建立应用程序的编程小组也可以使用任意 .NET 语言。这就是 .NET Framework 提供的开放性。

16.4.2 指定要使用的母版页

本章前面已介绍过，很容易在页面级指定要使用哪个母版页。在内容页面的 Page 指令中，只需使用 MasterPageFile 特性：

```
<%@ Page Language="C#" MasterPageFile="~/Wrox.master" %>
```

除了在页面级指定要使用的母版页之外，还可以采用第二种方式，在应用程序的 web.config 文件中指定要使用的母版页，如程序清单 16-16 所示。

程序清单 16-5 在 web.config 文件中指定母版页

```

<configuration>
  <system.web>
    <pages masterPageFile="~/Wrox.master" />
  </system.web>
</configuration>

```

在 web.config 文件中指定母版页，会使在应用程序中创建的所有内容页面都继承指定的母版页。如果在 web.config 文件中声明母版页，就可以创建使用这个母版页的任意多个内容页面。以这种方式指定母版页后，就可以使用如下方式构造内容页面的 Page 指令：

```
<%@ Page Language="C#" %>
```

只要在内容页面中声明另一个不同的母版页，就可以覆盖为整个应用程序指定的母版页：

```
<%@ Page Language="C#" MasterPageFile="~/MyOtherCompany.master" %>
```

在 web.config 文件中指定母版页，并不表示要在所有的.aspx 页面中都使用这个母版页。如果创建一个常规的 Web 窗体并运行它，ASP.NET 将知道该页面不是内容页面，因此把它作为一般的.aspx 页面运行。

16.4.3 将母版页应用于页面子集

如果只把母版页应用于页面子集(如包含在应用程序特定文件夹中的页面),就可以在 web.config 文件中使用<location>元素,如程序清单 16-6 所示。

程序清单 16-6 在 web.config 文件中为特定的文件夹指定母版页

```
<configuration>
  <location path="AdministrationArea">
    <system.web>
      <pages masterPageFile="~/WroxAdmin.master" />
    </system.web>
  </location>
</configuration>
```

在 web.config 文件中添加<location>部分后,就指定 AdministrationArea 文件夹将使用另一主文件模板,这是使用<location>元素的 path 特性指定的。path 特性的值可以是文件夹名,也可以是页面,如 AdminPage.aspx。

16.4.4 使用页面标题

在应用程序中创建内容页面时,所有的内容页面都默认为自动使用在母版页中声明的标题。例如,前面的母版页使用 My Company Master Page 标题。通过这个母版页创建的所有内容页面都使用 My Company Master Page 标题。在内容页面的@Page 指令中使用 Title 特性来指定页面的标题,就可以避免使用母版页的标题。也可以在内容页面中通过编程来操作页面标题。为此,应在内容页面的代码中使用 Master 对象。Master 对象的 Title 特性用于设置内容页面的页面标题,如程序清单 16-7 所示。

程序清单 16-7 为内容页面编写定制的页面标题

```
<%@ Page Language="C#" MasterPageFile="~/Wrox.master" %>
<script runat="server">
  protected void Page_LoadComplete(object sender, EventArgs e)
  {
    Master.Page.Title = "This page was generated on: " +
      DateTime.Now.ToString();
  }
</script>
```

16.4.5 使用母版页中的控件和属性

在内容页面中使用母版页时,可以访问母版页上的控件和属性。母版页由内容页面引用时,会提供 Master 特性。使用这个特性可以获取母版页中包含的控件值或定制特性。

例如,在母版页上创建 GUID(唯一标识符),在使用该母版页的内容页面上可以获得该 GUID。在这个例子中,使用了程序清单 16-1 中创建的母版页,但添加了一个 Label 服务器控件和 Page_Load 事件,如程序清单 16-8 所示。

程序清单 16-8 在第一次请求时创建 GUID 的母版页

```
<%@ Master Language="C#" %>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        if(!Page.IsPostBack)
        {
            Label1.Text = System.Guid.NewGuid().ToString();
        }
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>My Company Master Page</title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
        <table cellpadding="3" border="1">
            <tr bgcolor="silver">
                <td colspan="2">
                    <h1>My Company Home Page</h1>
                    <b>User's GUID:&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~
                        <asp:Label ID="Label1" runat="server" /></b>
                    ~
                </td>
            </tr>
            <tr>
                <td>
                    <asp:ContentPlaceHolder ID="ContentPlaceholder1"
                        runat="server">
                    </asp:ContentPlaceHolder>
                </td>
                <td>
                    <asp:ContentPlaceHolder ID="ContentPlaceholder2"
                        runat="server">
                    </asp:ContentPlaceHolder>
                </td>
            </tr>
            <tr>
                <td colspan="2">
                    Copyright 2012 - My Company
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

现在，母版页上有一个可以从内容页面访问的 Label 控件：

- 使用 FindControl 方法
- 使用定制属性

有两种方式可完成这项任务。第一种方式是使用母版页提供的 `FindControl` 方法，如程序清单 16-9 所示。

程序清单 16-9 获取内容页面上 Label 控件的 Text 值

```
<%@ Page Language="C#" MasterPageFile="~/Wrox.master" %>
<script runat="server">
    protected void Page_LoadComplete(object sender, EventArgs e)
    {
        Label1.Text = (Master.FindControl("Label1") as Label).Text;
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label2.Text = "<b>Hello " + TextBox1.Text + "!</b>";
    }
</script>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1"
    runat="server">
    <b>Your GUID number from the master page is:<br />
    <asp:Label ID="Label1" runat="server" /></b><p>
    <b>Enter your name:</b><br />
    <asp:Textbox ID="TextBox1" runat="server" />
    <br />
    <br />
    <asp:Button ID="Button1" runat="server" Text="Submit"
        OnClick="Button1_Click" /><br />
    <br />
    <asp:Label ID="Label2" runat="server" Font-Bold="True" /></p>
</asp:Content>
<asp:Content ID="Content3" ContentPlaceHolderID="ContentPlaceHolder2"
    runat="server">
    <asp:Image ID="Image1" runat="server" ImageUrl="Wrox.gif" />
</asp:Content>
```

在这个例子中，程序清单 16-8 中的母版页首先创建了一个 GUID，该 GUID 在母版页自身的 Label 服务器控件中存储为文本值。这个 Label 控件的 ID 是 `Label1`。母版页仅在第一次请求内容页面时生成这个 GUID，然后就可以使用这个值填充内容页面的控件。

内容页面的一个有趣的方面是，把代码放在 `Page_LoadComplete` 事件处理程序中，就可以获取母版页上的 GUID 值。在 ASP.NET 中，该事件在 `Page_Load` 事件触发后会立刻触发。事件的触发顺序将在本章的后面介绍，但内容页面中的 `Page_Load` 事件总是在母版页中的 `Page_Load` 事件之前触发。为了获得新的 GUID（它在母版页的 `Page_Load` 事件中创建），必须在 `Page_Load` 事件之后触发的一个事件中获得 GUID，此时就应使用 `Page_LoadComplete` 事件。因此，在内容页面的 `Page_LoadComplete` 事件中填充内容页面上的 Label 服务器控件。注意，内容页面上的 Label 控件与母版页上的 Label 控件有相同的 ID，因此该 ID 不能区分它们。可以使用 `Master` 属性区分它们。

如前所述，还有另一种方式：采用这种方式不仅可以获得母版页上的服务器控件，还可以访问母版页上可能显示的任何定制属性。在程序清单 16-10 所示的母版页中，为页面的 `<h1>` 部分使用了一个定制属性。

程序清单 16-10 显示了一个定制属性的母版页

[illegible]

```

        </tr>
    </table>
</form>
</body>
</html>

```

在这个母版页中，母版页有 `PageHeadingTitle` 属性。给这个属性赋予默认值 “My Company”，然后把它放在母版页文件的某个 `<h1>` 元素之间的 HTML 标记中，这会使其默认值变成母版页上使用的页面标题。尽管母版页已经有用于标题的值，但使用这个母版页的所有内容页面都可以重写这个 `<h1>` 标题，具体过程如程序清单 16-11 所示。

程序清单 16-11 重写母版页中属性的内容页面

```

<%@ Page Language="C#" MasterPageFile="~/Wrox.master" %>
<%@ MasterType VirtualPath="~/Wrox.master" %>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        Master.PageHeadingTitle = "My Company-Division X";
    }
</script>

```

在内容页面中，可以使用 `Master` 属性给母版页中的属性赋值。这是很简单的操作。我们不仅可以获得母版页中的所有公有属性，还可以获得母版页包含的所有方法。

为此，需要使用 `MasterType` 页面指令。该指令允许对母版页进行强类型化的引用，通过 `Master` 对象访问母版页的属性。

前面介绍了如何使用 `FindControl` 方法获得母版页上的服务器控件。`FindControl` 方法工作得很好，但它是后期绑定方法，如果从标记中删除控件，调用该方法就会失败。从 `FindControl` 方法返回对象时，应使用防御性的编码方式，不断地检查 `null`。通过上面介绍的机制(使用程序清单 16-10 中的公有属性)，可以采用另一种方式获得母版页上的服务器控件。这种方式更高效。

为此，只需把服务器控件设置为公有属性，如程序清单 16-12 所示。

程序清单 16-12 把母版页中的服务器控件设置为公有属性

```

<%@ Master Language="C#" %>
<script runat="server">
    public Label MasterPageLabel1
    {
        get
        {
            return Label1;
        }
        set
        {
            Label1 = value;
        }
    }
</script>

```


在这个例子中，公有属性 `MasterPageLabel1` 提供了通过 `Label1` 的 ID 访问 `Label` 控件的方法。现在就可以在内容页面上创建 `MasterPageLabel1` 属性的实例，重写 `Label` 服务器控件的任意属性。因此，如果要增大母版页创建的 GUID 的尺寸，并在 `Label1` 服务器控件中显示它，只需修改 `Label` 控件的 `Font.Size` 属性，如程序清单 16-13 所示。

程序清单 16-13 重写母版页上 `Label` 控件的属性

```
<%@ Page Language="C#" MasterPageFile "~/Wrox.master" %>
<%@ MasterType VirtualPath "~/Wrox.master" %>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        Master.MasterPageLabel1.Font.Size = 25;
    }
</script>
```

这种方法是在内容页面上获得母版页上服务器控件的最高效方式。

16.5 在母版页中指定默认内容

如前所述，母版页可以指定内容页面能够使用的内容区域。母版页可以只包含一块内容区域，也可以包含多块内容区域。内容区域的一个好处是，在创建母版页时，可以指定内容区域的默认内容。如果选择不重写这个默认内容，它就可以由内容页面使用。程序清单 16-14 中的母版页就在内容区域中指定了一些默认内容。

程序清单 16-14 在母版页中指定默认内容

```
<%@ Master Language="C#" %>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>My Company</title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
        <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
            Here is some default content.
        </asp:ContentPlaceHolder><p>
        <asp:ContentPlaceHolder ID="ContentPlaceHolder2" runat="server">
            Here is some more default content.
        </asp:ContentPlaceHolder></p>
    </form>
</body>
</html>
```

要在母版页的内容区域中放置默认内容，只需把它放在母版页的 `ContentPlaceHolder` 服务器控件上。继承这个母版页的所有内容页面也会继承默认内容。程序清单 16-15 中的内容页面重写了这个母版页的一块内容区域。

程序清单 16-15 在内容页面中重写一些默认内容

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master" %>
<asp:Content ID="Content3" ContentPlaceHolderId="ContentPlaceHolder2" runat="server">
    Here is some new content.
</asp:Content>
```

这段代码创建了一个页面，其中一块内容区域的内容来自母版页本身，其他内容来自内容页面，如图 16-10 所示。

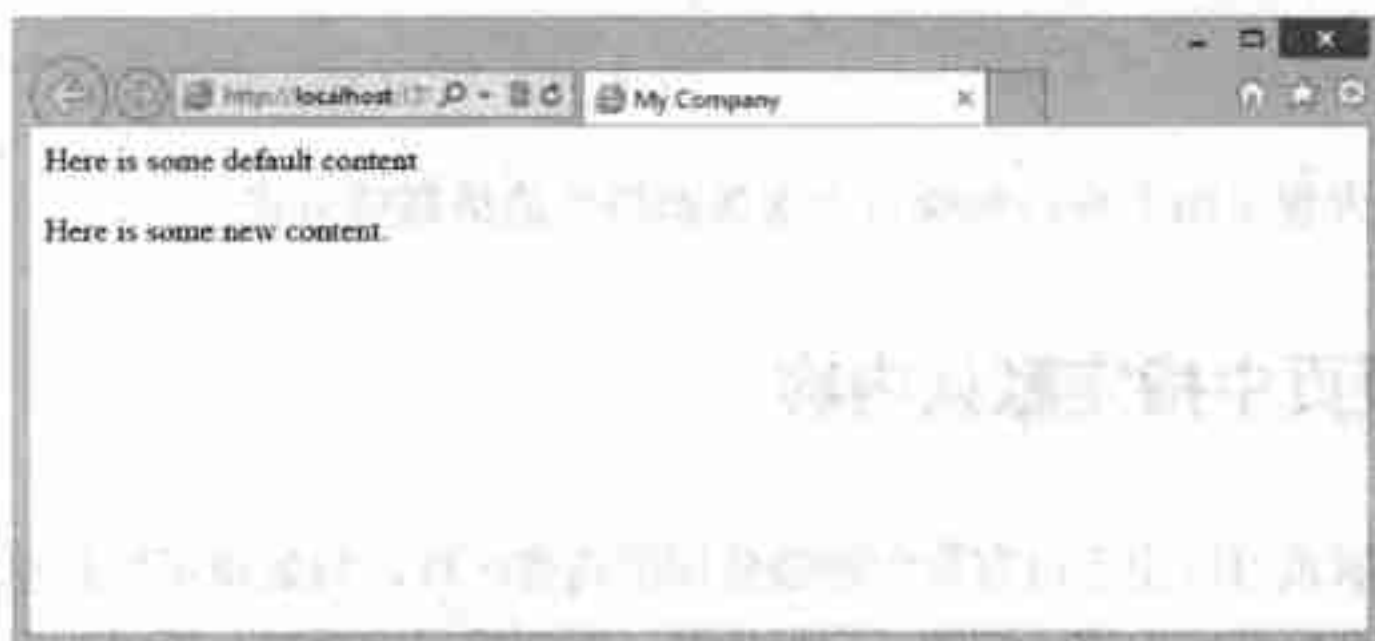


图 16-10

在 Visual Studio 2012 的设计视图中处理内容区域时，另一个有趣之处在于，智能标记允许用户简单地处理默认内容。

第一次处理内容页面时，注意所有的默认内容最初都放在 `Content` 服务器控件中。单击该控件的智能标记，从弹出的菜单中选择 `Create Custom Content` 命令，就可以改变默认内容。这个选项允许重写母版页的内容，并插入自定义的内容。在内容区域中放置一些定制内容后，智能标记就会显示另一个选项 `Default to Master's Content`。这个选项允许把母版页的默认内容返回到内容区域中，并删除已放在内容区域中的定制内容，这样就很容易返回为默认内容。如果选择这个选项，就会得到一个警告，指出这样操作会删除放在 `Content` 服务器控件中的定制内容，如图 16-11 所示。

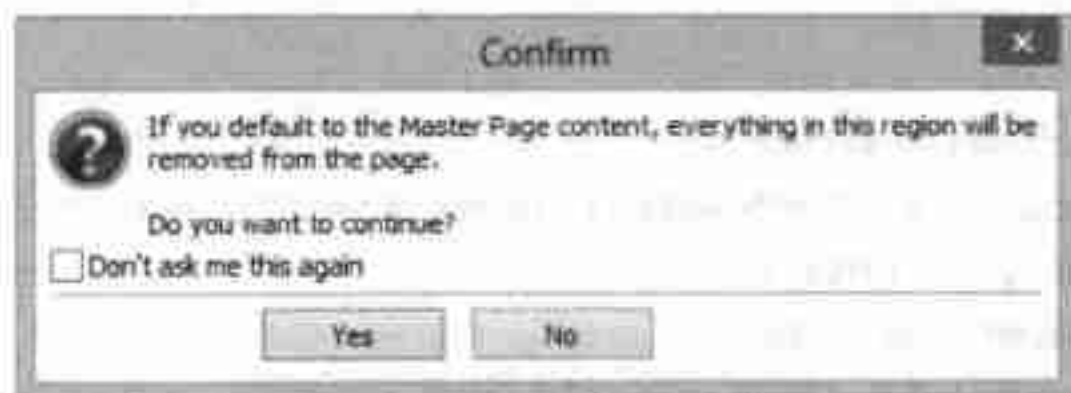


图 16-11

改变 `Content` 服务器控件中的默认内容后，结果如图 16-12 所示。

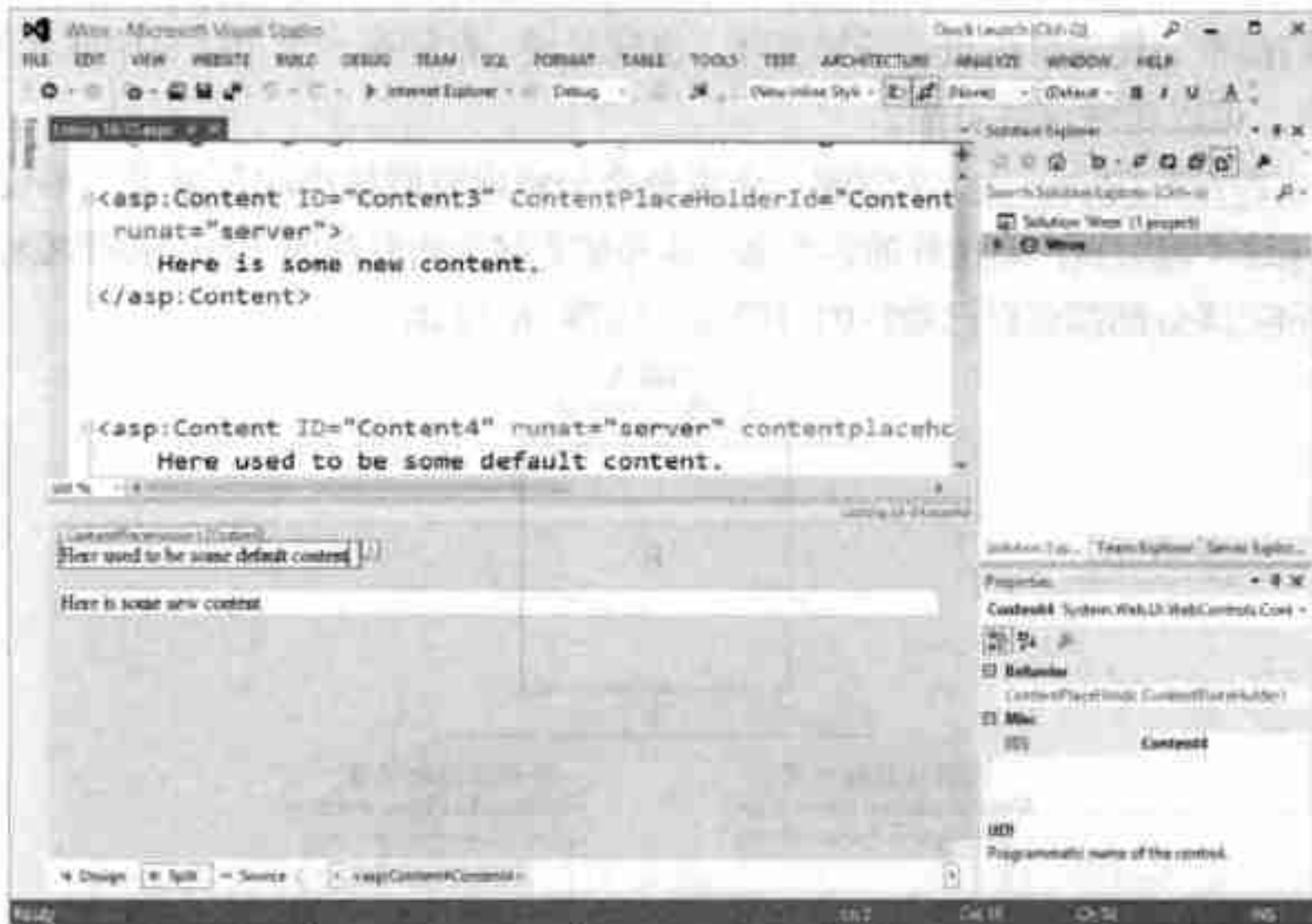


图 16-12

16.6 以编程方式将母版页赋予内容页面

在任何内容页面上，都可以轻松地以编程方式指定母版页。使用 `Page.MasterPageFile` 属性就可以把母版页赋予内容页面。无论是否已经在 `@Page` 指令中指定了另一个母版页，都可以使用该属性。

为此，要通过 `Page_PreInit` 事件使用这个属性。`Page_PreInit` 事件是访问 `Page` 生存期时最早触发的事件。因此，应在该事件中指定内容页面使用的母版页。在使用母版页时，`Page_PreInit` 是一个重要的事件，因为这是在把母版页和内容页面合并到单个实例中之前能够同时影响这两个页面的唯一地方。程序清单 16-16 描述了如何把母版页以编程方式赋予内容页面。

程序清单 16-16 使用 `Page_PreInit` 事件将母版页以编程方式赋予内容页面

```
<%@ Page Language="C#" %>
<script runat="server">
    protected void Page_PreInit(object sender, EventArgs e)
    {
        Page.MasterPageFile = "~/MyMasterPage.master";
    }
</script>
```

在这个例子中，当动态生成页面时就会在页面构造过程的开始位置把母版页赋予内容页面。注意，内容页面必须包含 `Content` 控件，否则就会抛出错误。

16.7 母版页的嵌套

本章前面已经创建了一个内容页面可以使用的母版页。但是，大多数公司和组织都存在许多部

门和小组，他们都希望使用母版页的各种变体，也就是说，希望在一个母版页中嵌套另一个母版页。在 ASP.NET 中，可以实现这种页面。

例如，假定路透社(Reuters)正在创建一个由整个公司内联网使用的母版页。不仅路透社希望在整个公司内实现这个母版页，路透社的各个部门也希望直接为内联网的子部分提供模板。例如，路透社的欧洲分部和美国分部都有自己独特的母版页，如图 16-13 所示。

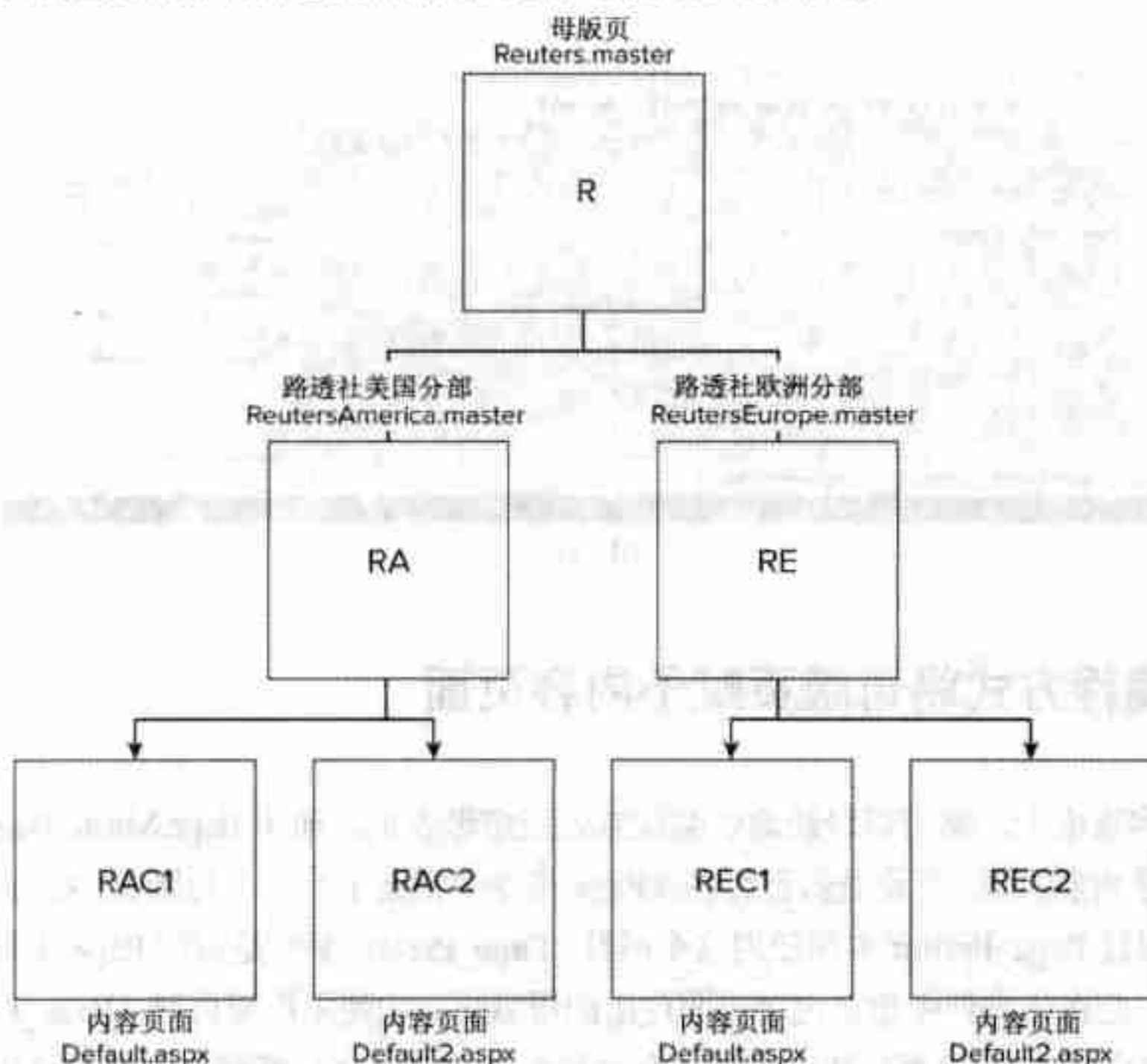


图 16-13

为此，路透社的欧洲分部和美国分部都各自创建了一个母版页，该页面继承了全局母版页，如程序清单 16-17 所示。文件 ReutersMain.master 在本书第 16 章的下载代码中。

程序清单 16-17 主母版页

```

<%@ Master Language="C#" %>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Reuters</title>
  <asp:ContentPlaceHolder id="head" runat="server">
  </asp:ContentPlaceHolder>
</head>
<body>
  <form id="form1" runat="server">
    <p><asp:Label ID="Label1" runat="server" BackColor="LightGray"
      BorderColor="Black" BorderWidth="1px" BorderStyle="Solid"
      Font-Size="XX-Large"> Reuters Main Master Page </asp:Label></p>
    <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
    </asp:ContentPlaceHolder>
  </form>
</body>
</html>
  
```

```

</form>
</body>
</html>

```

这是一个简单的母版页，但非常适合于说明这个嵌套功能的工作原理。主母版页在公司内全局使用，其 ContentPlaceHolder 服务器控件的 ID 是 ContentPlaceHolder1。

创建子母版页或嵌套的母版页时，使用的方式与建立其他母版页相同。在 Add New Item 对话框中选择 Master Page 选项，确保选中 Select master page 复选框，如图 16-14 所示。这会再次打开允许选择母版页的对话框。

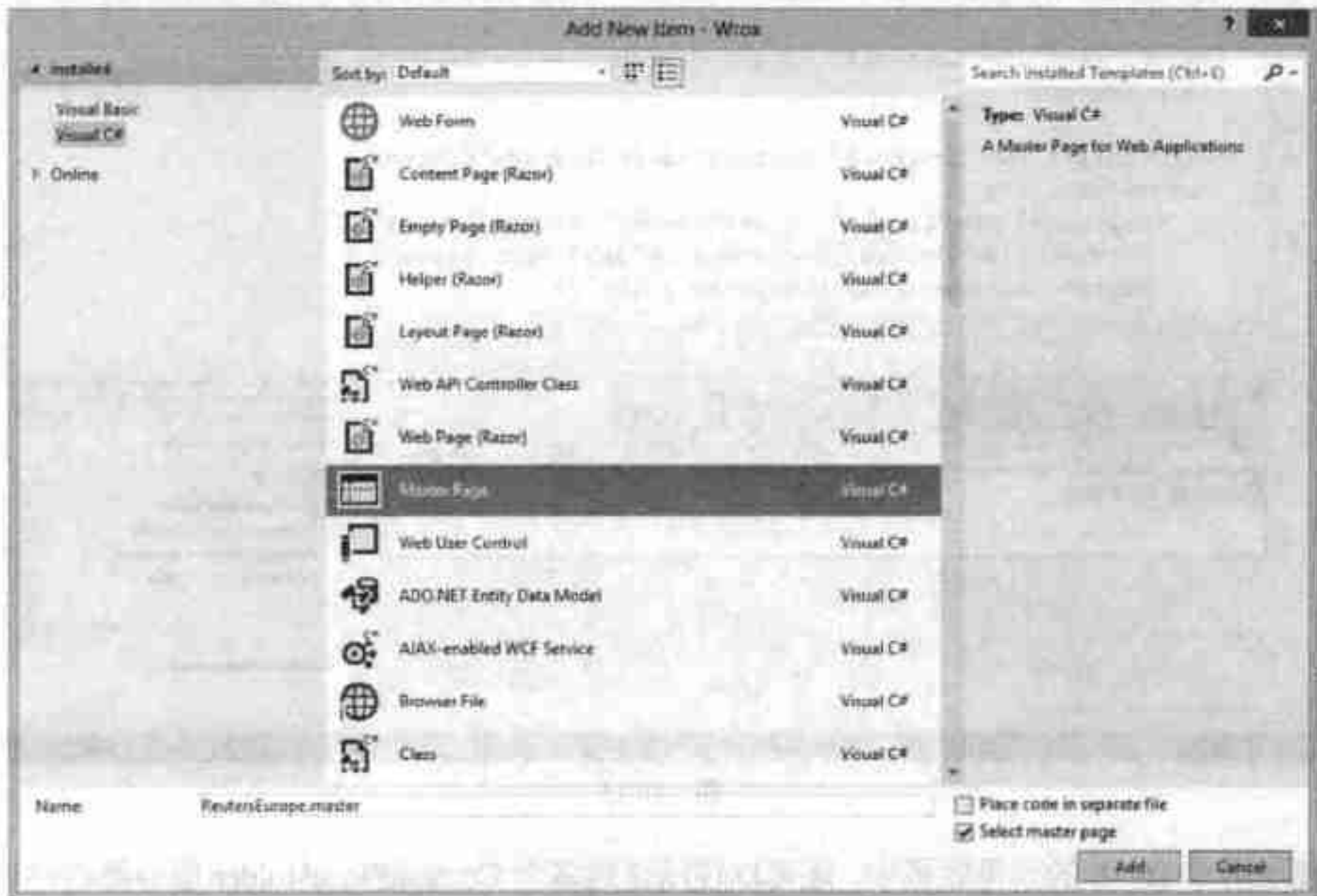


图 16-14

程序清单 16-18 说明了如何在子母版页文件中使用这个主母版页。文件 ReutersEurope.master 在本书第 16 章的下载代码中。

程序清单 16-18 子母版页

```

<%@ Master Language="C#" MasterPageFile="~/ReutersMain.master" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
  <asp:Label ID="Label1" runat="server" BackColor="#E0E0E0" BorderColor="Black"
    BorderStyle="Dotted" BorderWidth="2px" Font-Size="Large">
    Reuters Europe </asp:Label><br /><hr />
  <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
  </asp:ContentPlaceHolder>
</asp:Content>

```

仔细查看这个页面，你会发现它与使用母版页的普通.aspx 页面没有什么太大区别。MasterPageFile

特性的用法也是相同的，只是使用了@Master 指令，而不是@Page 指令。接着，Content2 控件也使用了 Content 控件的 ContentPlaceHolderId 属性。这个属性把该内容区域输入到主母版页中定义的 ContentPlaceHolder1 内容区域中。

ASP.NET 的功能之一是可以直接在 Visual Studio 2012 的设计视图中查看嵌套的母版页。图 16-15 在 Visual Studio 2012 的设计视图中显示了嵌套的母版页。

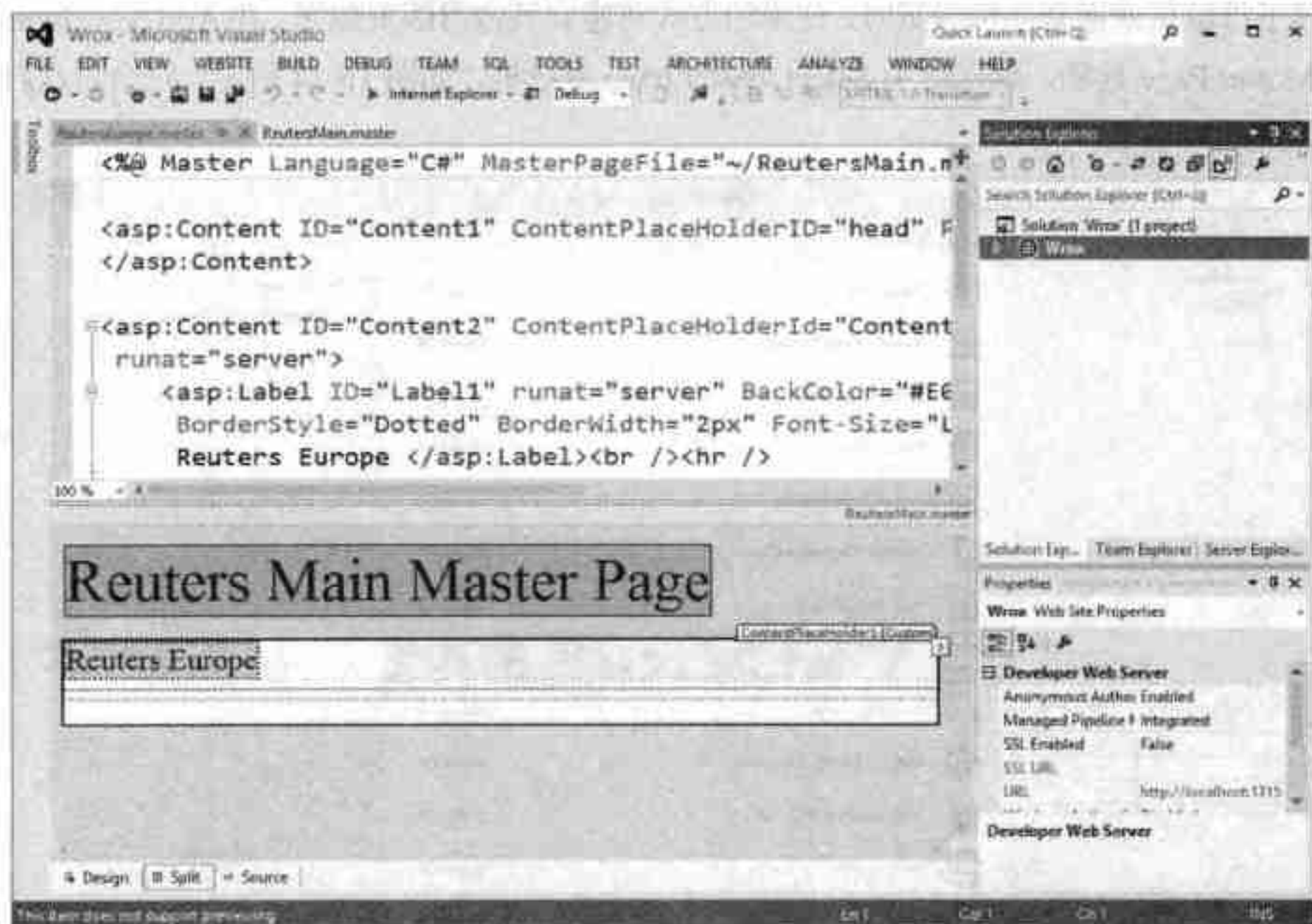


图 16-15

在程序清单 16-18 的子母版页中，还可以使用任意多个 ContentPlaceHolder 服务器控件。使用这个母版页的所有内容页面都可以使用这些控件。程序清单 16-19 中的内容页面就使用了这个子母版页，即 ReutersEurope.master。

程序清单 16-19 内容页面 Default.aspx

```
<%@ Page Language="C#" MasterPageFile="~/ReutersEurope.master" %>
<asp:Content ID="Content1" ContentPlaceHolderId="ContentPlaceHolder1" runat="server">
    Hello World
</asp:Content>
```

可以看出，在这个内容页面中，Page 指令中的 MasterPageFile 属性值是我们创建的子母版页。继承 ReutersEurope 母版页实际上是把两个母版页(ReutersMain.master 和 ReutersEurope.master)合并到单个母版页中。这个内容页面的 Content 控件也指向子母版页中定义的内容区域，在代码中使用 ContentPlaceHolderId 属性时可以看到这一点。最后得到的是一个不太美观的页面，如图 16-16 所示。

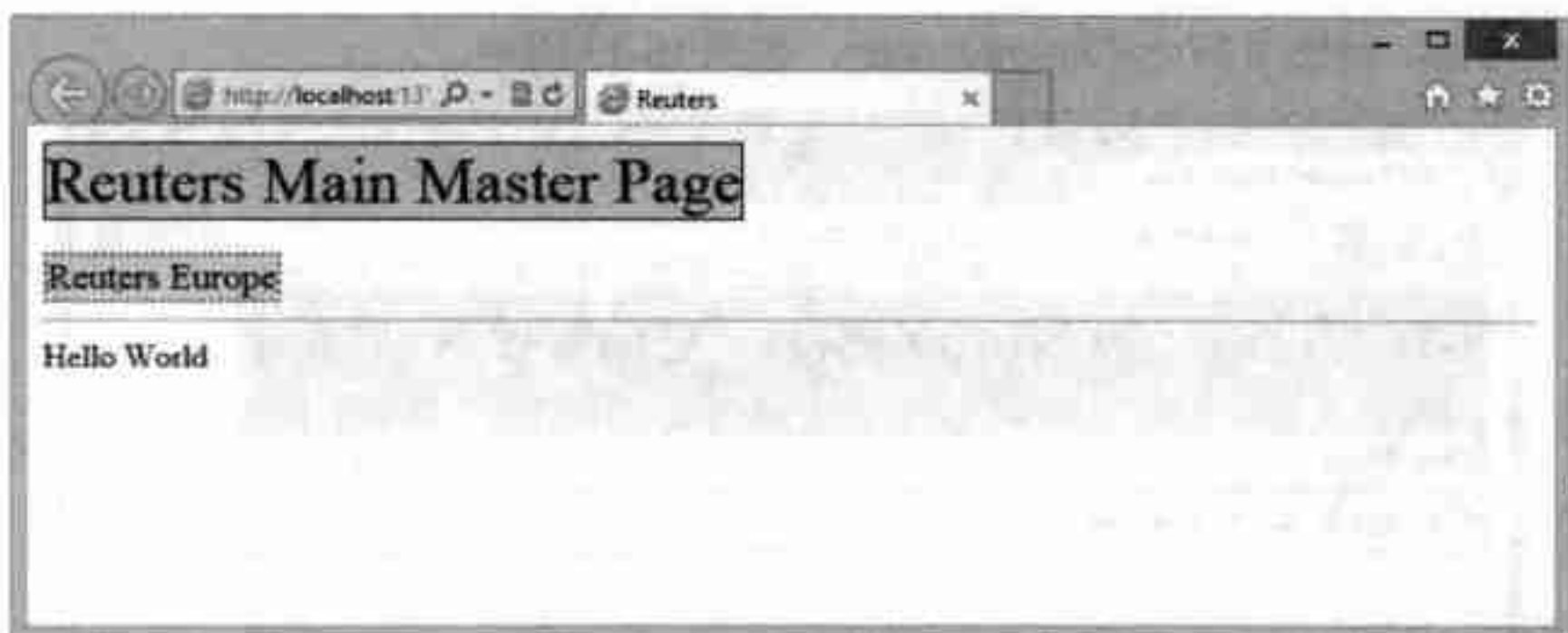


图 16-16



可以看出,也可以创建使用子母版页的内容页面。但这并不总是正确的解决方案,有时甚至适得其反。嵌套母版页的结构可能变得非常复杂,最终会过度依赖继承。另外,一个母版页的微小变化,就可能对所有派生的内容页面产生很大的影响。

16.8 浏览器特定的母版页

在许多情况下,可以使用许多不同的浏览器查看开发人员建立的应用程序。一些访问者在 Internet Explorer 中显示应用程序,其他一些访问者则使用 Firefox 或 Google Chrome 查看,还有一些访问者在平板电脑或手机上调用应用程序。

因此,ASP.NET 允许在内容页面上使用多个母版页。根据终端用户使用的浏览器类型,ASP.NET 引擎会使用相应的母版页文件。因此,我们需要创建能适用于特定浏览器的母版页,从而充分利用特定浏览器提供的功能,给终端用户提供最佳的浏览体验。使用多个母版页的功能如程序清单 16-20 所示。

程序清单 16-20 可以处理多个母版页的内容页面

```
<%@ Page Language="C#" MasterPageFile="~/Wrox.master"
    Firefox:MasterPageFile="~/WroxFirefox.master"
    Safari:MasterPageFile="~/WroxSafari.master"
    Opera:MasterPageFile="~/WroxOpera.master"
    Chrome:MasterPageFile="~/WroxChrome.master" %></asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderId="ContentPlaceHolder1"
    runat="server">
    Hello World
</asp:Content>
```

从这个示例内容页面中可以看出,它可以处理 3 个不同的母版页文件。第一个母版页文件使用 MasterPageFile 属性。这是用于不符合任何其他选项的页面的默认设置。也就是说,如果请求者不是 Firefox、Chrome、Safari 或 Opera 浏览器,就使用默认的母版页——Wrox.master。但是,如果请求者

是 Chrome 浏览器, 就使用 WroxChrome.master, 如图 16-17 所示。

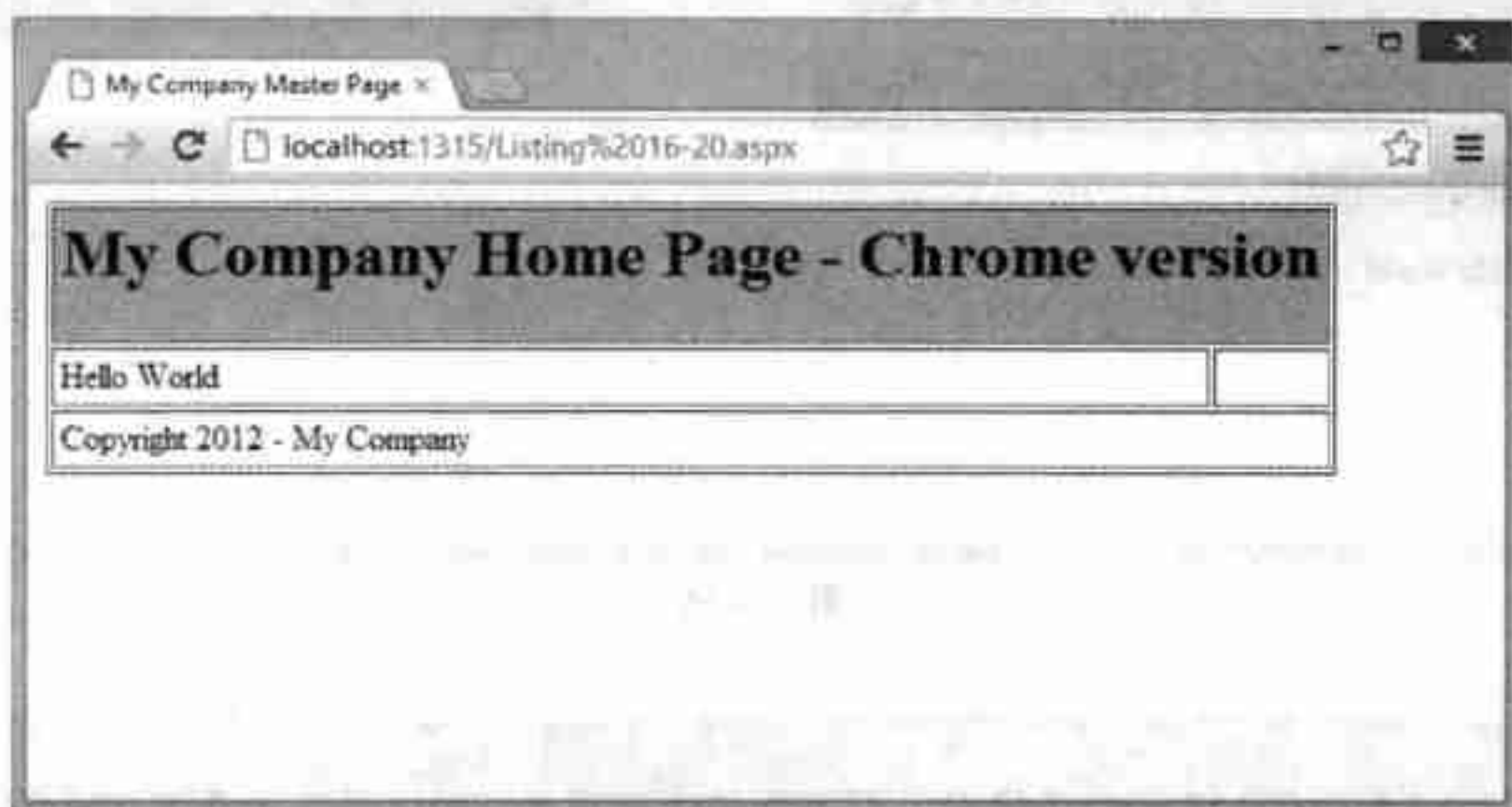


图 16-17

在产品服务器上有一个关于可用浏览器的列表, 应用程序位于产品服务器的 C:\Windows\Microsoft .NET\Framework\v4.0.xxxx\CONFIG\Browsers 下。一些可用的浏览器如下:

- Blackberry
- Chrome
- Default
- Firefox
- Gateway
- Generic
- IE
- IEMobile
- iPhone
- Opera
- Safari
- UCBrowser

当然, 也可以添加所需的任何其他.browser 文件。

16.9 事件的触发顺序

在处理母版页和内容页面时, 可以使用相同的事件(如 Page_Load)。但一定要知道哪些事件先触发, 哪些事件后触发。这是母版页最重要的一个方面, 因为其事件顺序不同于常规页面的事件顺序。

当把两个类合并为单个页面类时, 需要特定的顺序。当终端用户在浏览器上请求内容页面时, 事件的触发顺序如下:

- (1) 母版页子控件的初始化: 先初始化母版页包含的所有服务器控件。
- (2) 内容页面子控件的初始化: 初始化内容页面包含的所有服务器控件。
- (3) 母版页的初始化: 初始化母版页。
- (4) 内容页面的初始化: 初始化内容页面。

- (5) 内容页面的加载: 加载内容页面(这是 Page_Load 事件, 后跟 Page_LoadComplete 事件)。
- (6) 母版页的加载: 加载母版页(这也是 Page_Load 事件, 后跟 Page_LoadComplete 事件)。
- (7) 母版页子控件的加载: 把母版页中的服务器控件加载到页面中。
- (8) 内容页面子控件的加载: 把内容页面中的服务器控件加载到页面中。

在建立应用程序时应注意事件触发顺序。例如, 如果要在特定的内容页面中使用母版页包含的服务器控件值, 就不能从内容页面的 Page_Load 事件中提取这些服务器控件的值。这是因为这个事件在母版页的 Page_Load 事件之前触发。这个问题促成在 .NET Framework 中创建了 Page_LoadComplete 事件。内容页面的 Page_LoadComplete 事件在母版页的 Page_Load 事件之后触发。因此, 可以使用这个触发顺序在母版页中获得控件值, 但在触发内容页面的 Page_Load 事件时, 该控件没有值。

16.10 高速缓存母版页

在处理普通的.aspx 页面时, 可以使用下面的构造代码(或其变体)对页面应用输出高速缓存:

```
<%@ OutputCache Duration="10" Varybyparam="None" %>
```

这会在服务器的内存中把页面缓存 10 秒。许多开发人员使用输出高速缓存来提高 ASP.NET 页面的性能。它也可以用于包含不会很快过时的数据的页面。

在使用母版页时, 如何把输出高速缓存应用于 ASP.NET 页面? 首先, 不能把高速缓存只应用于母版页, 也不能把 OutputCache 指令放在母版页上。如果这么做, 在页面的第二次检索时就会出错, 因为应用程序找不到缓存的页面。

为了在使用母版页时应用输出高速缓存, 应把 OutputCache 指令放在内容页面上。这会高速缓存内容页面的内容和母版页的内容(此时母版页只是单个页面)。放在母版页上的 OutputCache 指令不会让母版页出错, 但也不会被缓存。这个指令只能用于内容页面。

在使用高速缓存功能方面, ASP.NET 4.5 的另一新的有趣功能是, ASP.NET 现在可以在控件级别控制视图状态。可以使用通常包含多种视图状态的控件(例如 GridView 等)来实现该功能, 也可以通过 ContentPlaceHolder 控件来使用这个新功能。

例如, 可以构造如下代码:

```
<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server" ViewStateMode="Disabled">
</asp:ContentPlaceHolder>
```

在这个例子中, 如果页面的剩余部分在使用视图状态, 那么 ContentPlaceHolder1 就不使用视图状态。ViewStateMode 属性的可用值包括 Disabled、Enabled 和 Inherit。Disabled 值用于关闭控件的视图状态, Enabled 值用于打开控件的视图状态, Inherit 值用于获取 @Page 指令中分配的值。删除视图状态可以提高页面的性能。



第 22 章详细介绍了 ASP.NET 的缓存功能。

16.11 ASP.NET AJAX 和母版页

目前,许多大型 ASP.NET 应用程序都使用母版页以及这项技术提供的功能,以建立模板化的 Web 站点。ASP.NET 把 ASP.NET AJAX 作为默认安装的一部分,并且母版页和 Ajax 集成得非常好。



有关 ASP.NET AJAX 的内容,详见本书第 23 章。

使用 AJAX 功能的每个页面都必须在页面上放置 ScriptManager 控件。如果要使用 AJAX 的页面是一个利用了母版页的内容页面,那么还必须把 ScriptManager 控件放在母版页上。



一个页面上只能有一个 ScriptManager 控件。

创建支持 Ajax 的母版页并不困难。为了做到这一点,只需将 ScriptManager 服务器控件添加到母版页上即可,如程序清单 16-21 所示。

程序清单 16-21 支持 Ajax 的母版页

```
<%@ Master Language="C#" %>
<script runat="server">
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
<asp:ContentPlaceHolder id="head" runat="server">
</asp:ContentPlaceHolder>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:ScriptManager ID="ScriptManager1" runat="server" />
<asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">
</asp:ContentPlaceHolder>
</div>
</form>
</body>
</html>
```

从程序清单 16-21 中可以看出,这个 Ajax 母版页与标准母版页的唯一区别是包含了 ScriptManager 服务器控件。如果母版页拥有 Ajax 功能,即使内容页面根本不使用 Ajax,也需要包含 ScriptManager 服务器控件。

如果要把常用的 JavaScript 项放在 Web 应用程序的所有页面上,使用母版页上的 ScriptManager 控件也是有帮助的。例如,程序清单 16-22 说明了如何通过母版页方便地在每个页面上包含 JavaScript。

程序清单 16-22 通过母版页包含脚本

```

<%@ Master Language="C#" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title></title>
  <asp:ContentPlaceHolder id="head" runat="server">
  </asp:ContentPlaceHolder>
</head>
<body>
  <form id="form1" runat="server">
  <div>
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Scripts>
        <asp:ScriptReference Path="myScript.js" />
      </Scripts>
    </asp:ScriptManager>
    <asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">
    </asp:ContentPlaceHolder>
  </div>
</form>
</body>
</html>

```

在这个例子中，myScript.js 文件现在位于每个使用了这个 Ajax 母版页的内容页面上。如果内容页面还需要使用 Ajax 功能，就不能把另一个 ScriptManager 控件放在该页面上。相反，内容页面需要使用已经位于母版页上的 ScriptManager 控件。

这就是说，如果内容页面需要给 ScriptManager 控件添加其他的项，就可以使用 ScriptManagerProxy 服务器控件访问母版页上的 ScriptManager 控件。使用 ScriptManagerProxy 控件可以给 ScriptManager 添加任意项，这些项专门用于包含它们的内容页面实例。

例如，程序清单 16-23 说明了如何通过 ScriptManagerProxy 控件向内容页面添加额外的脚本。

清单 16-23 使用 ScriptManagerProxy 控件添加额外的项

```

<%@ Page Language="C#" MasterPageFile="~/AjaxMaster.master" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
  <asp:ScriptManagerProxy ID="ScriptManagerProxy1" runat="server">
    <Scripts>
      <asp:ScriptReference Path="myOtherScript.js" />
    </Scripts>
  </asp:ScriptManagerProxy>
</asp:Content>

```

在这个例子中，内容页面使用 ScriptManagerProxy 控件添加额外的脚本。ScriptManagerProxy 控件的工作方式与 ScriptManager 控件一样，只是它专门用于使用母版页的内容页面。ScriptManagerProxy 控件会与页面的 ScriptManager 控件交互，执行必要的操作。

16.12 本章小结

在创建使用公共页眉、页脚或导航部分(几乎每个页面都拥有)的应用程序时,使用母版页是非常好的解决方案。母版页很容易实现,并且通过改变一个文件就可以把更改应用于应用程序的每个页面,从而大大方便了包含成千上万个页面的大型应用程序的管理。

本章介绍了 ASP.NET 中的母版页,说明了如何在 Web 应用程序中建立和使用母版页。除了基础知识之外,本章还介绍了母版页的事件触发顺序、高速缓存和特定于容器的特定母版页。最后,在处理模板化的应用程序时,母版页应是我们的首选。

第 17 章

站点导航

本章要点

- 使用.sitemap 文件和 SiteMapPath 服务器控件
- 使用 TreeView 和 Menu 服务器控件
- 利用 URL 映射
- 本地化 Web.sitemap 文件和过滤站点导航的结果

我们开发的 Web 应用程序一般都有多个页面。通常要以某种方式创建许多相互连接的 Web 页面。如果在页面之间建立导航系统，终端用户就很容易以直观的方式操作应用程序。

目前，必须从许多不同的方式中选择一种来将应用程序的路径展示给终端用户。当继续给整个应用程序添加页面时，站点导航这一艰巨任务将更加复杂。

目前在 Web 应用程序中建立导航的方法是在页面上散布超链接。一般使用包含文件或用户控件将超链接添加到 Web 页面中。也可以将超链接直接硬编码到页面上，显示在页面的标题中或边栏上。当移动页面或修改页面名称时，导航的难度会加大。有时，开发人员不得不进入应用程序的每个页面以修改导航的某些方面。

ASP.NET 4.5 引入了导航系统来解决这个问题，从而使终端用户对应用程序的管理变得非常简单。ASP.NET 的这个功能非常复杂，但它也可以很简单，我们可以深入控制其工作方式的各个方面。

站点导航系统可以在一个 XML 文件中定义整个站点，该 XML 文件称为站点地图。在定义了站点地图后，就可以使用 SiteMap 类来编程处理它。ASP.NET 中站点地图功能的另一个方面是：提供了一个专门用于处理站点地图文件的数据提供程序，它可以把站点地图绑定到一系列基于导航的服务器控件。本章介绍 ASP.NET 4.5 导航系统中的所有这些组件。下面将首先探讨站点地图。

17.1 基于 XML 的站点地图

站点地图并不是必需的元素(如后面所述)，但在使用 ASP.NET 4.5 的导航系统时，通常首先要为应用程序建立站点地图。站点地图是站点结构的 XML 描述。

使用站点地图可以定义应用程序中所有页面的导航结构,以及它们的相互关系。如果根据 ASP.NET 的站点地图标准来定义站点地图,就要使用 SiteMap 类或 SiteMapDataSource 控件与导航信息交互。使用 SiteMapDataSource 控件可以把站点地图文件中的信息绑定到各种数据绑定控件,包括 ASP.NET 提供的导航服务器控件。

要为应用程序创建新的站点地图文件,可以将站点地图或 XML 文件添加到应用程序中,把 XML 文件命名为 Web.sitemap。如果选择了 SiteMap 选项,那么就已经添加了这个文件。该文件命名为 Web,扩展名是.sitemap。程序清单 17-1 演示了.sitemap 文件的一个例子(本章下载代码中的 Web.sitemap)。

程序清单 17-1 Web.sitemap 文件示例

```
<?xml version="1.0" encoding="utf-8" ?>

<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode title="Home" description="Home Page" url="Default.aspx">
    <siteMapNode title="News" description="The Latest News" url="News.aspx">
      <siteMapNode title="U.S." description="U.S. News"
        url="News.aspx?cat=us" />
      <siteMapNode title="World" description="World News"
        url="News.aspx?cat=world" />
      <siteMapNode title="Technology" description="Technology News"
        url="News.aspx?cat=tech" />
      <siteMapNode title="Sports" description="Sports News"
        url="News.aspx?cat=sport" />
    </siteMapNode>
    <siteMapNode title="Finance" description="The Latest Financial Information"
      url="Finance.aspx">
      <siteMapNode title="Quotes" description="Get the Latest Quotes"
        url="Quotes.aspx" />
      <siteMapNode title="Markets" description="The Latest Market Information"
        url="Markets.aspx">
        <siteMapNode title="U.S. Market Report"
          description="Looking at the U.S. Market" url="MarketsUS.aspx" />
        <siteMapNode title="NYSE"
          description="The New York Stock Exchange" url="NYSE.aspx" />
      </siteMapNode>
      <siteMapNode title="Funds" description="Mutual Funds"
        url="Funds.aspx" />
    </siteMapNode>
    <siteMapNode title="Weather" description="The Latest Weather"
      url="Weather.aspx" />
  </siteMapNode>
</siteMap>
```

这个文件有什么功能?它提供了逻辑结构,ASP.NET 现在可以在导航系统的其他地方使用该结构。下面查看这个文件的构造方式。

这个 XML 文件的根节点是<siteMap>元素,该文件中只能有一个<siteMap>元素。在这个<siteMap>元素中有一个根元素<siteMapNode>,这一般是应用程序的起始页面。在程序清单 17-1 所在的文件

中，根元素<siteMapNode>指向起始页面 Default.aspx:

```
<siteMapNode title="Home" description="Home Page" url="Default.aspx">\
```

表 17-1 描述了<siteMapNode>元素中最常见的一些特性。

表 17-1

特 性	说 明
title	title 特性提供链接的文本描述。这里使用的 String 值是用于链接的文本
description	description 特性不仅说明链接的作用，而且被用于链接上的 ToolTip 特性。ToolTip 特性是终端用户把光标停留在链接上几秒后显示出来的黄色方框
url	url 特性描述了文件在解决方案中的位置。如果文件在根目录下，就使用文件名，如“Default.aspx”。如果文件位于子文件夹下，就在这个特性的 String 值中包含该文件夹，如“MySubFolder/Markets.aspx”

添加了第一个<siteMapNode>后，就可以在根元素<siteMapNode>中嵌套任意多个<siteMapNode>元素。还可以在该结构中通过为父<siteMapNode>元素创建子元素<siteMapNode>来生成更多的链接层次。

程序清单 17-1 中的例子给应用程序提供了如下导航结构：

```
Home
  News
    U.S.
    World
    Technology
    Sports
  Finance
    Quotes
    Markets
      U.S. Market Report
      NYSE
    Funds
  Weather
```

可以看出，这个结构在某些地方有 3 层。使用该文件的最简单方式是利用 ASP.NET 的 SiteMapPath 服务器控件，SiteMapPath 服务器控件是专门用来处理.sitemap 文件的。

17.2 SiteMapPath 服务器控件

通过 ASP.NET 提供的服务器控件 SiteMapPath，很容易使用刚才创建的.sitemap 文件。这个控件位于 Visual Studio 2012 IDE 的 Navigation 部分。

SiteMapPath 控件可以创建导航功能，用户可能以前创建过，也可能在互联网上的 Web 页面上看到过。SiteMapPath 控件会创建所谓的“面包屑导航(breadcrumb navigation)”，这是一种线性路径，定义了终端用户在导航结构中的位置。Web 站点 newegg.com 如图 17-1 所示，其中就使用了这种类型的导航。

这类导航系统的作用是向终端用户显示它们与站点其他内容间的相互关系。传统上,编写这类导航系统是比较困难的;但现在有了 SiteMapPath 服务器控件,就很容易编写这类导航系统。例如下面要介绍某(假想的)新闻网站的站点结构。



图 17-1

首先创建一个应用程序,它包含程序清单 17-1 中创建的 Web.sitemap 文件。之后创建 Web 窗体 MarketsUS.aspx,这个文件在 Web.sitemap 文件中被定义为应用程序的最底层文件。

SiteMapPath 控件很容易使用,甚至不需要使用数据源控件将之绑定到 Web.sitemap 文件,以获得该文件中的所有信息。只需把一个 SiteMapPath 控件拖放到 MarketsUS.aspx 页面上,就会得到程序清单 17-2。

程序清单 17-2 使用 Web.sitemap 文件和 SiteMapPath 服务器控件

```
<%@ Page Language="C#" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Using the SiteMapPath Server Control</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:SiteMapPath ID="Sitemappath1" runat="server">
    </asp:SiteMapPath>
  </form>
</body>
</html>
```

这段代码确实很简单。运行这个页面,结果如图 17-2 所示。

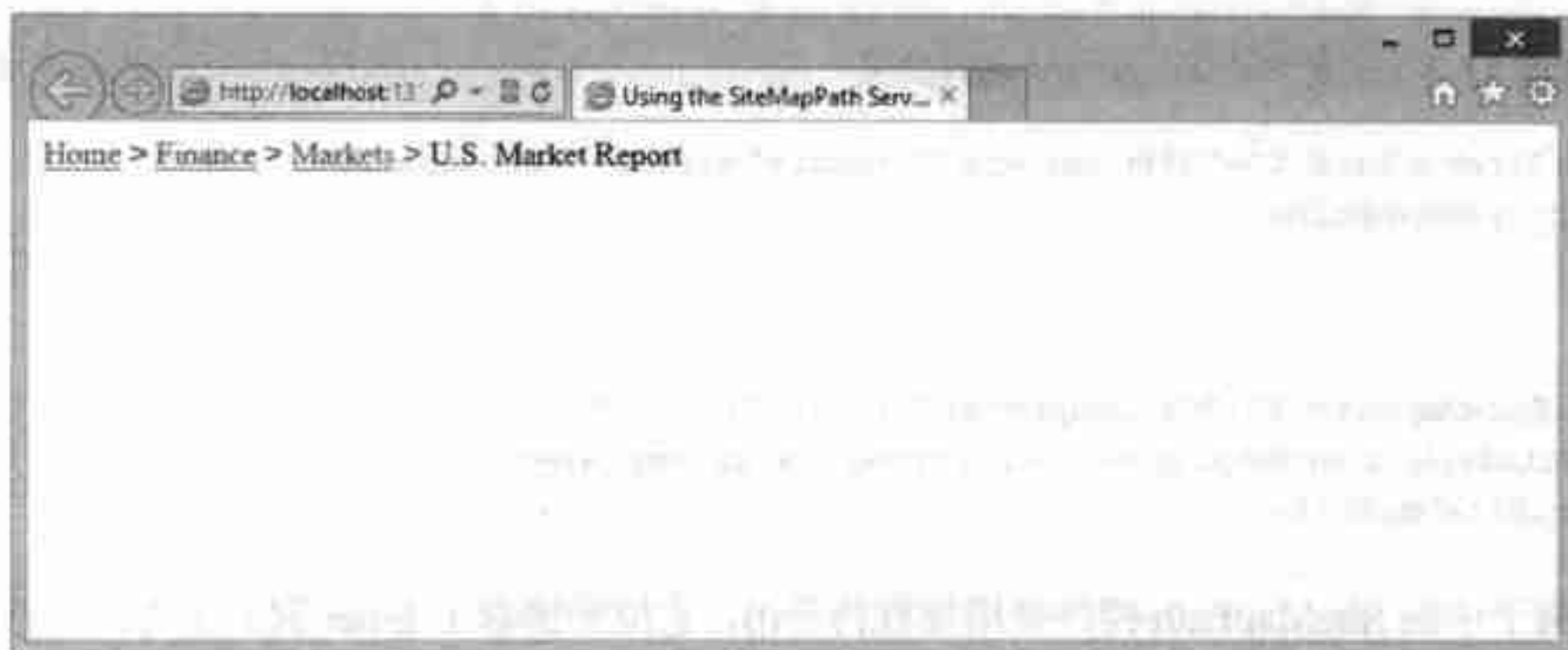


图 17-2

这个屏幕截图显示 MarketsUS.aspx 页面上的美国市场报告。终端用户可以看到，这个页面位于站点的 Markets 部分，而 Markets 部分在站点的 Finance 部分。通过面包屑导航，终端用户能理解站点的结构并得知他们在站点中的位置，从而可以快速选择链接，以导航到站点的任意位置。

如果把光标停留在 Finance 链接上几秒钟，就会出现工具提示，如图 17-3 所示。

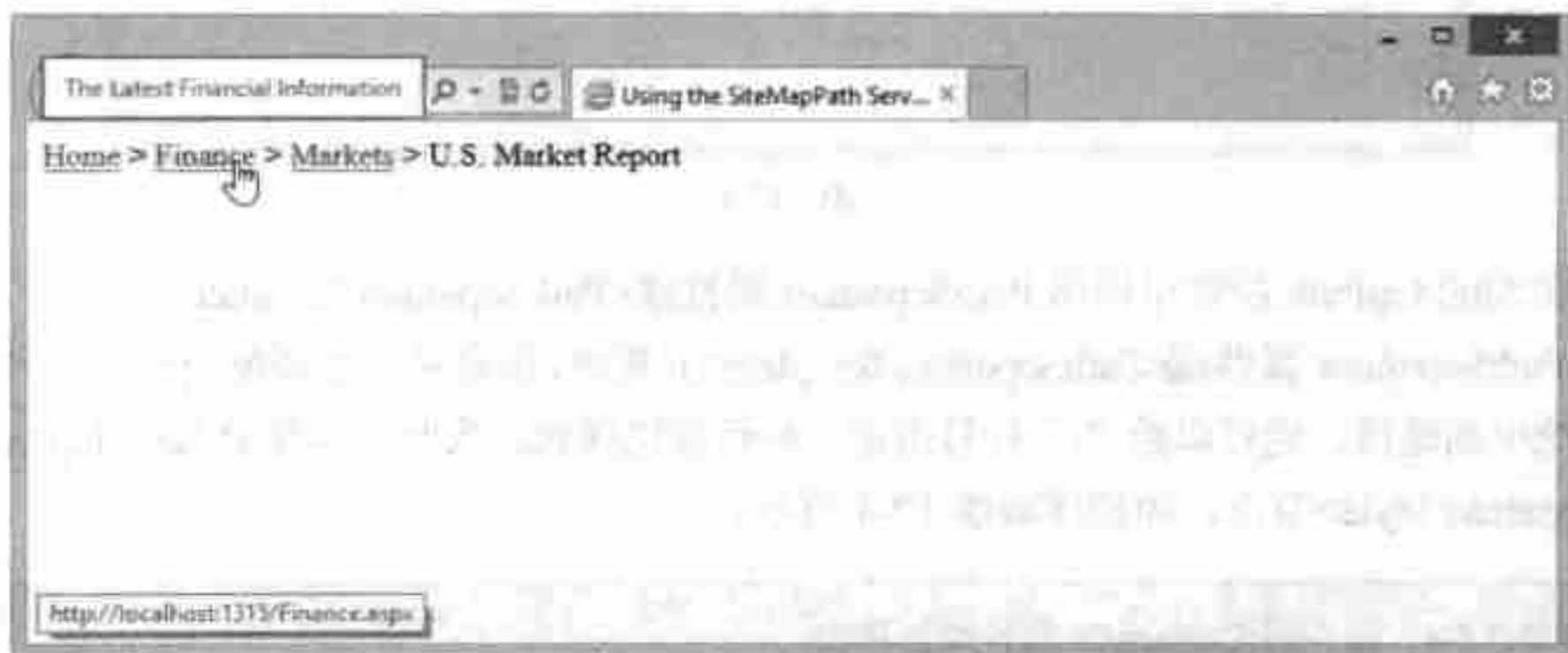


图 17-3

这个工具提示的内容是 The Latest Financial Information，它来自 Web.sitemap 文件中 <siteMapNode> 元素的 description 特性：

```
<siteMapNode title="Finance" description="The Latest Financial Information"
url="Finance.aspx">
```

SiteMapPath 控件能自动工作，不需要用户的参与。只需把这个基本控件添加到页面上，该控件就会自动创建刚才所看到的面包屑导航系统。但是，可以使用下面介绍的属性来修改该控件的外观和行为。

17.2.1 PathSeparator 属性

SiteMapPath 控件的一个重要的样式属性是 PathSeparator。在默认情况下，SiteMapPath 控件使用大于号(>)来分隔链接元素。给 PathSeparator 属性重新指定值，就可以改变分隔符。程序清单 17-3 演示了这个属性的用法。

程序清单 17-3 改变 PathSeparator 属性的值

```
<asp:SiteMapPath ID="Sitemappath1" runat="server" PathSeparator=" | ">
</asp:SiteMapPath>
```

或

```
<asp:SiteMapPath ID="Sitemappath1" runat="server">
  <PathSeparatorTemplate> | </PathSeparatorTemplate>
</asp:SiteMapPath>
```

这个例子中的 SiteMapPath 控件使用竖杠符号(|)，它位于键盘上 Enter 键的上方。运行代码，结果如图 17-4 所示。

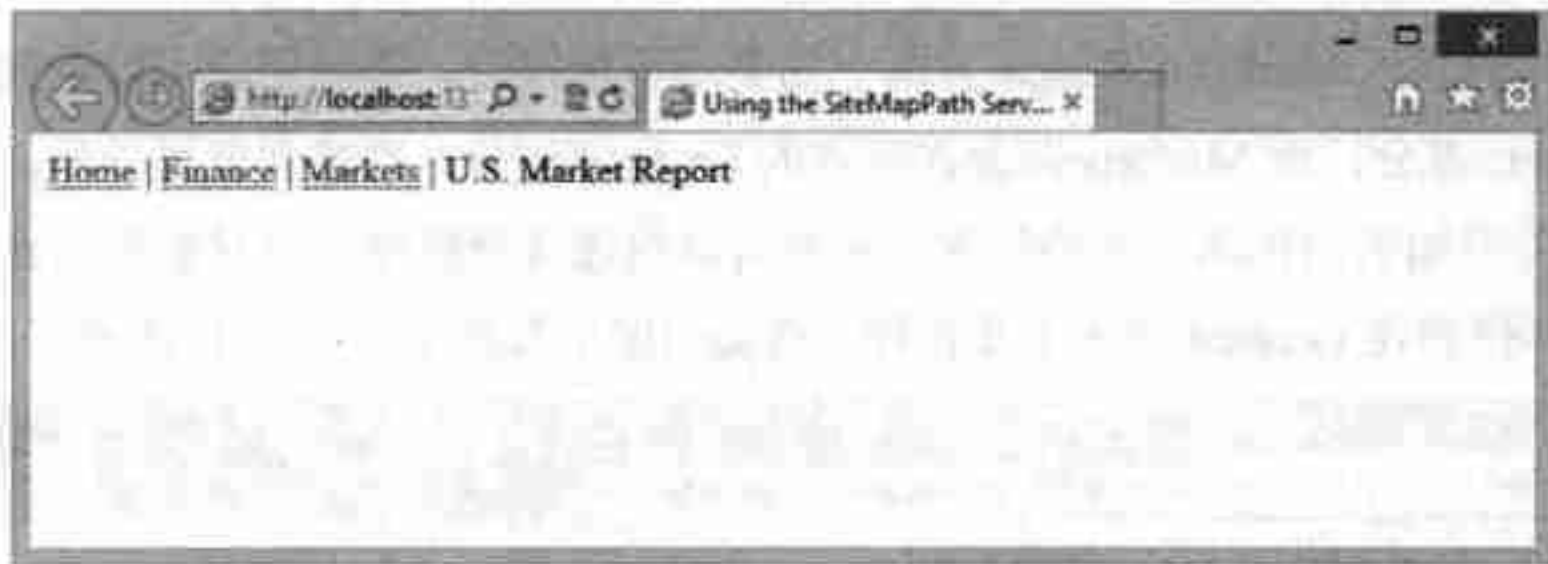


图 17-4

可以在 SiteMapPath 控件中使用 PathSeparator 属性或<PathSeparatorTemplate>元素。

使用 PathSeparator 属性或<PathSeparatorTemplate>元素时，很容易指定要使用什么符号分隔面包屑导航系统中的链接。也可以给“|”符号指定一些可视化样式。为此，需要给 SiteMapPath 控件添加<PathSeparatorStyle>节点，如程序清单 17-4 所示。

程序清单 17-4 给 PathSeparator 属性添加样式

```
<asp:SiteMapPath ID="Sitemappath1" runat="server" PathSeparator=" | ">
  <PathSeparatorStyle Font-Bold="true" Font-Names="Verdana" ForeColor="#663333"
    BackColor="#cccc66"></PathSeparatorStyle>
</asp:SiteMapPath>
```

这似乎不太美观，但是演示了使用<PathSeparatorStyle>元素和 SiteMapPath 控件可以改变分隔符元素的可视化外观。结果如图 17-5 所示。

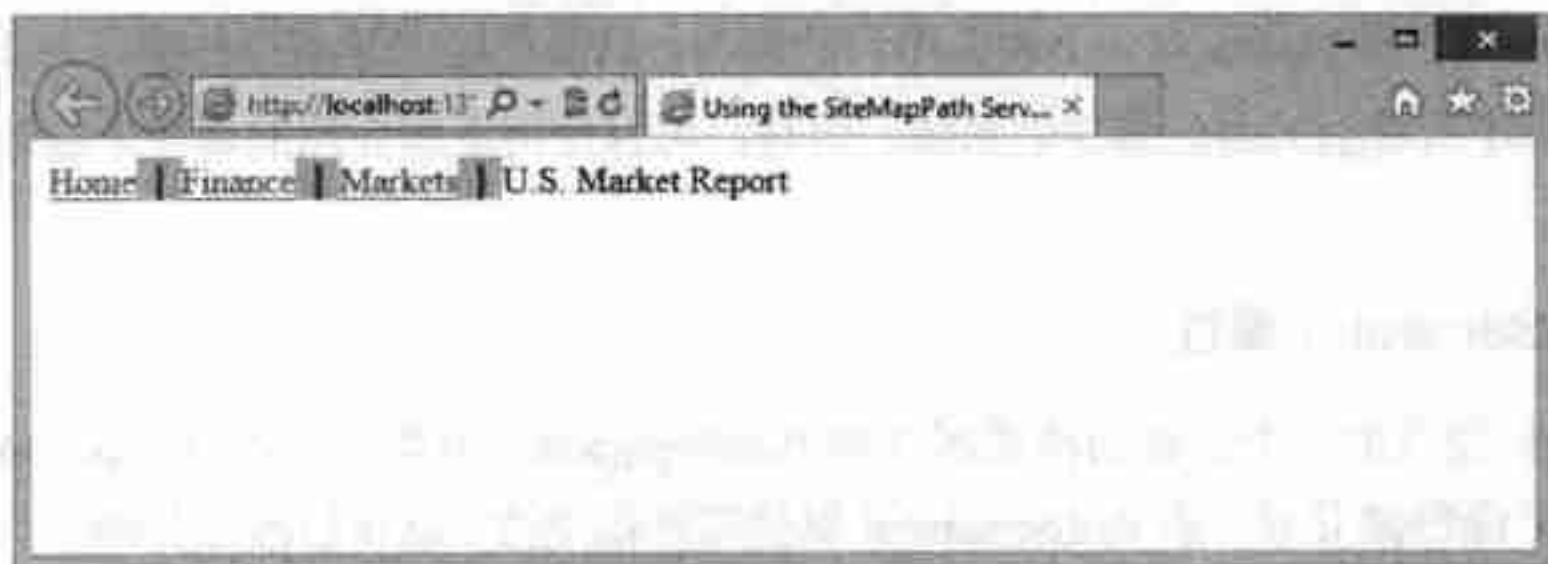


图 17-5

使用这些结构，还可以把图像作为分隔符添加进来，如程序清单 17-5 所示。

程序清单 17-5 把图像用作分隔符

```
<%@ Page Language="C#" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Using the SiteMapPath Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:SiteMapPath ID="SiteMapPath1" runat="server">
            <PathSeparatorTemplate>
                <asp:Image ID="Image1" runat="server" ImageUrl="divider.gif" />
            </PathSeparatorTemplate>
        </asp:SiteMapPath>
    </form>
</body>
</html>
```

要把图像用作链接之间的分隔符，可以使用<PathSeparatorTemplate>元素，在该元素中放置 Image 控件。实际上，可以在 SiteMapPath 控件生成的导航链接之间放置任意类型的控件。

17.2.2 PathDirection 属性

使用 SiteMapPath 控件时，另一个有趣的属性是 PathDirection，这个属性用于改变输出中生成的链接的方向。这个属性只有两个值：RootToCurrent 和 CurrentToRoot。

Root 链接是显示中的第一个链接，它通常是首页。Current 链接是当前显示的页面的链接。这个属性默认设置为 RootToCurrent。如果在示例中把该属性的值改为 CurrentToRoot，就会生成如图 17-6 所示的结果。

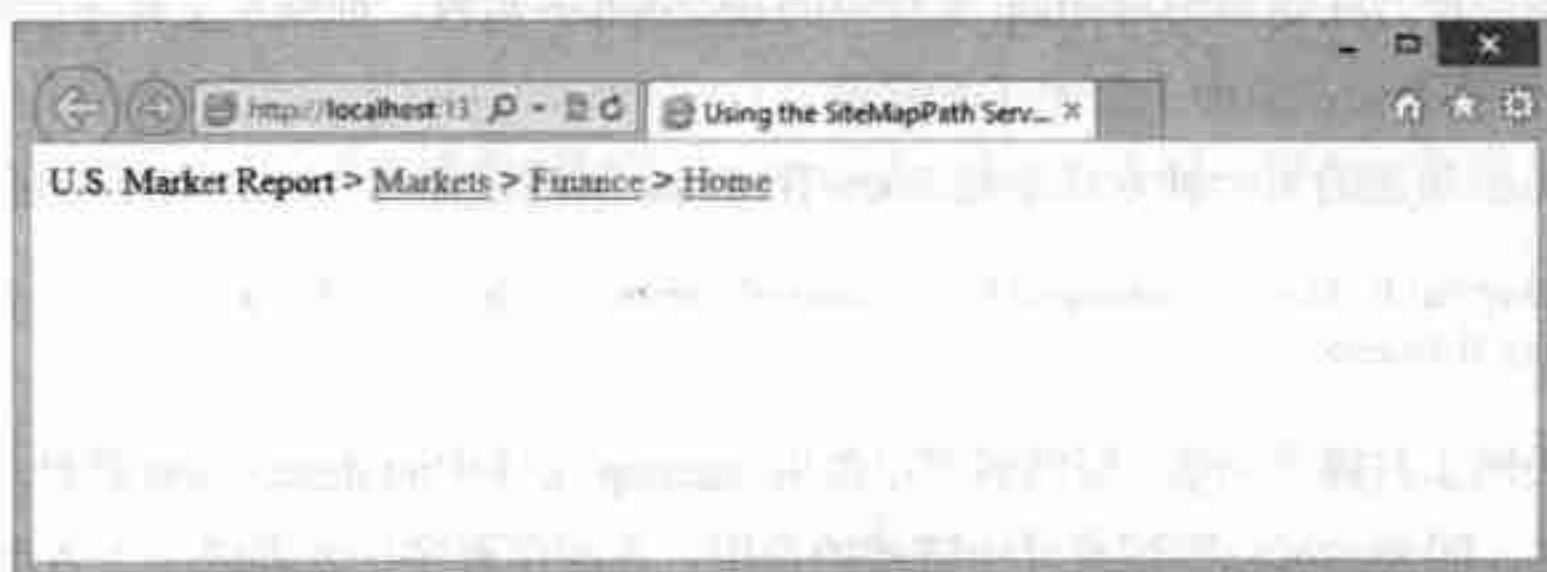


图 17-6

17.2.3 ParentLevelsDisplayed 属性

在一些情况下，导航系统会有许多层。在程序清单 17-1 所示的站点地图中，有 3 层页面，这不是很复杂的结构。但是，一些人处理的站点可能有许多层页面。此时，使用 SiteMapPath 控件就不明智。因为这么做会显示非常长的页面列表。

在这种情况下，可以使用 SiteMapPath 控件的 ParentLevelsDisplayed 属性。设置该属性后，会只

显示指定深度的页面。因此，如果使用 SiteMapPath 控件和程序清单 17-1 中的 Web.sitemap，就可以把 ParentLevelsDisplayed 属性的值设置为 3，此时看不到页面有什么变化，但它已经显示为 3 层页面深度的路径。如果把这个值改为 2，SiteMapPath 控件就会构建为：

```
<asp:SiteMapPath ID="Sitemappath1" runat="server" ParentLevelsDisplayed="2">
</asp:SiteMapPath>
```

注意，这个改动的结果如图 17-7 所示。SiteMapPath 控件只显示两层页面深度，没有显示首页链接。

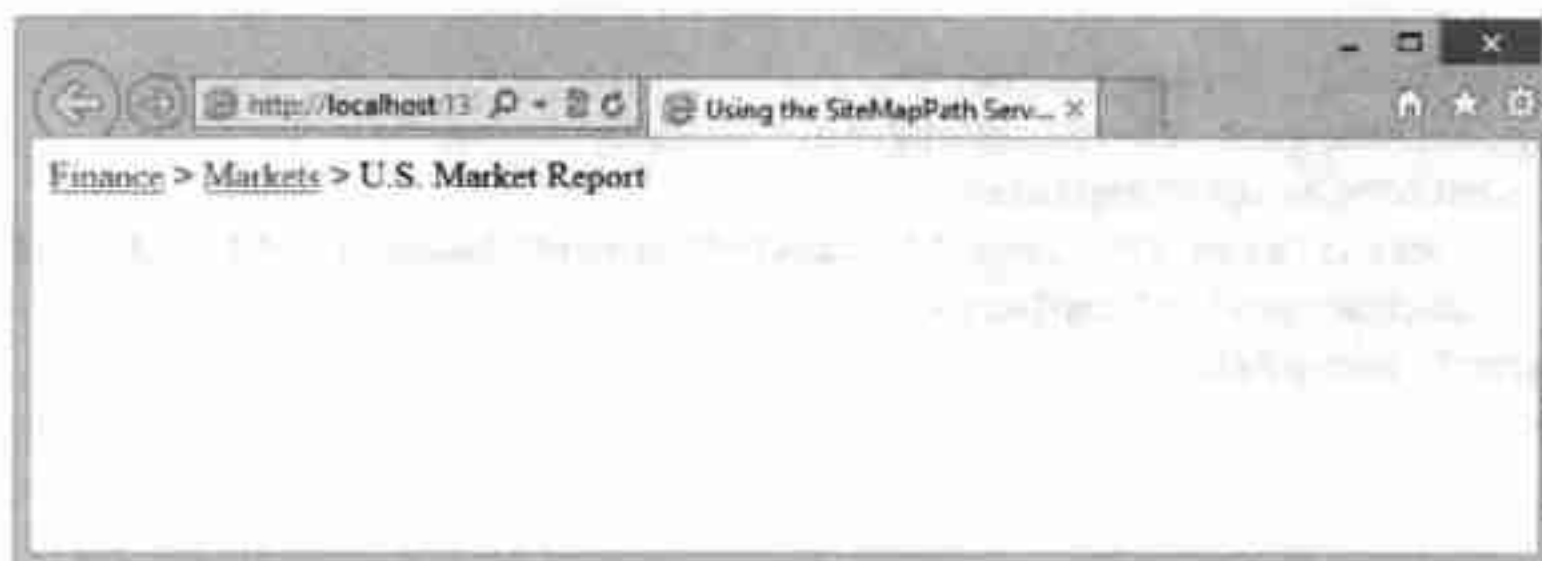


图 17-7

在默认情况下，所显示的链接数量没有限制，因此 SiteMapPath 控件会根据站点地图文件中指定的值生成特定数量的链接。

17.2.4 ShowToolTips 属性

默认情况下，如果在 Web.sitemap 文件中使用了 description 特性，SiteMapPath 控件就会为每个链接生成工具提示。工具提示是终端用户把光标停留在 SiteMapPath 控件中的一个链接上时出现在屏幕上的文本，本章前面已介绍过这个功能。

我们常常不希望 SiteMapPath 控件为它生成的链接显示工具提示。此时，可以使用两种方式关闭这个功能。第一种方式是忽略.sitemap 文件中的 description 特性。如果从文件中删除了这些特性，SiteMapPath 控件就不会在页面上显示工具提示。

关闭工具提示功能的另一种方式是把 ShowToolTips 属性设置为 false，如下所示：

```
<asp:SiteMapPath ID="Sitemappath1" runat="server" ShowToolTips="false">
</asp:SiteMapPath>
```

这样就会关闭工具提示功能，但仍然可以使用.sitemap 文件中的 description 特性。可能还要使用 description 特性，因为它可以跟踪文件中链接的作用。在应用程序中处理成千上万个链接时，该功能就很有帮助。

17.2.5 SiteMapPath 控件的子元素

前面已经介绍过 SiteMapPath 控件的<PathSeparatorStyle>和<PathSeparatorTemplate>子元素，但 SiteMapPath 控件还包含其他子元素。表 17-2 中列出了它的所有子元素。

表 17-2

子 元 素	说 明
CurrentNodeStyle	在当前显示的页面中, 给 SiteMapPath 导航系统中的链接应用样式
CurrentNodeTemplate	在当前显示的页面中, 给 SiteMapPath 导航系统中的链接应用模板
NodeStyle	给 SiteMapPath 导航系统中的所有链接应用样式。在 CurrentNodeStyle 或 RootNodeStyle 元素中应用的设置会替代这里的设置
NodeTemplate	给 SiteMapPath 导航系统中的所有链接应用模板。在 CurrentNodeStyle 或 RootNodeStyle 元素中应用的设置会替代这里的设置
PathSeparatorStyle	给 SiteMapPath 导航系统中的链接分隔符应用样式
PathSeparatorTemplate	给 SiteMapPath 导航系统中的链接分隔符应用模板
RootNodeStyle	给 SiteMapPath 导航系统中的第一个链接(根链接)应用样式
RootNodeTemplate	给 SiteMapPath 导航系统中的第一个链接应用模板

17.3 TreeView 服务器控件

TreeView 服务器控件是一个功能非常丰富的控件, 可以显示数据的层次结构, 因此很适合显示包含在.sitemap 文件中的内容。图 17-8 说明了该控件如何显示本章前面一直在使用的站点地图(程序清单 17-1)的内容。该图首先在屏幕顶部显示了一个完全折叠的 TreeView 控件, 然后在下方显示完全展开的第二个 TreeView 控件。

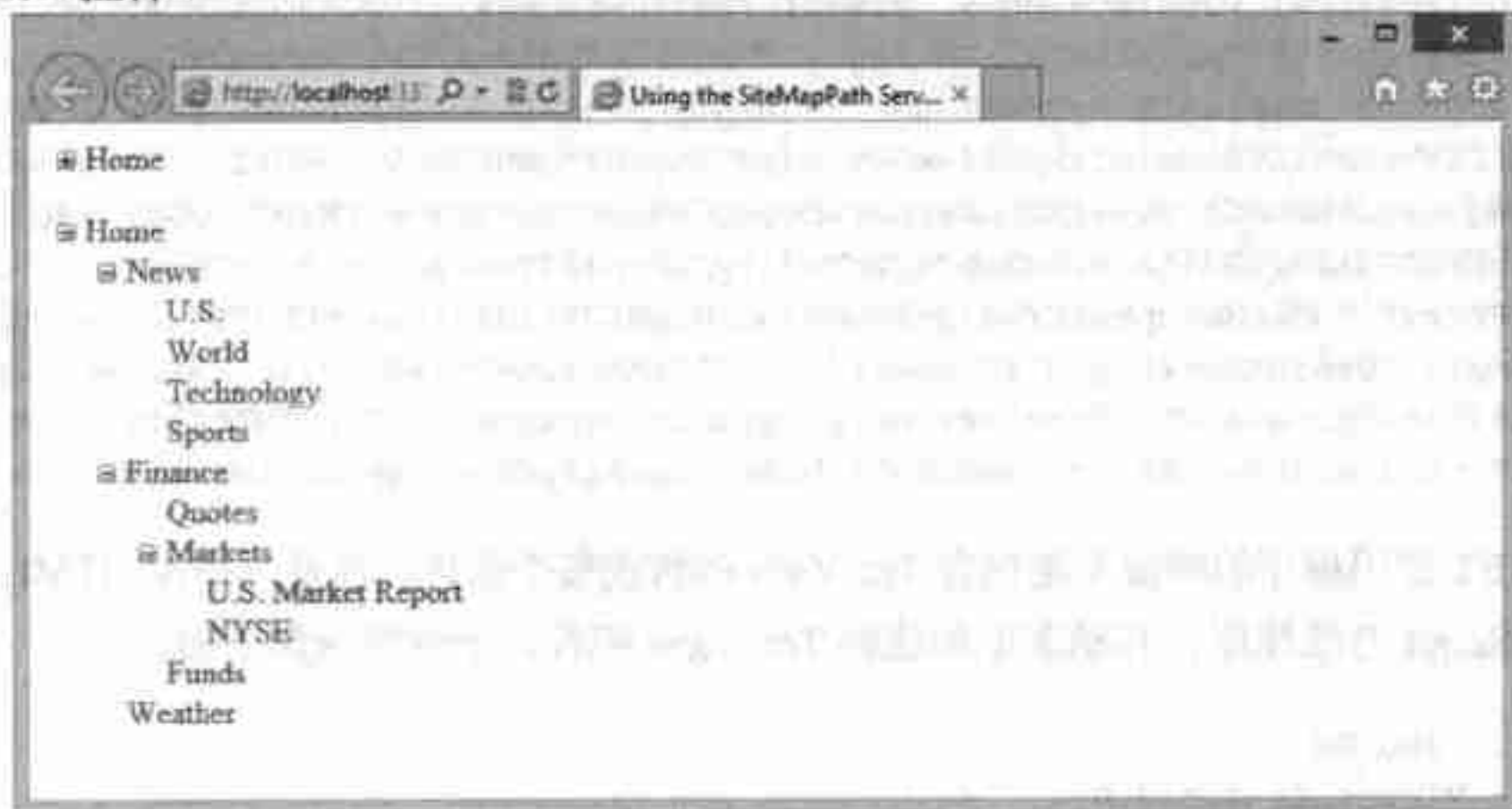


图 17-8

这个控件可以通过它的可扩展、可折叠架构动态加载要显示的节点。如果该控件可以显示树型视图的输出和一些客户端脚本, 那么当有人展开了该控件中的一个节点以显示选中项的子节点时, 该控件就可以向服务器回送调用。如果站点导航系统比较大, 该功能就很理想。此时, 动态加载 TreeView 控件的节点可以大大提高性能。回送功能的优点是: 在底层实现, 并且不需要完全刷新 ASP.NET 页面。当然, 只有浏览器接受带有 TreeView 控件的客户端代码时, 才能使用这个功能。如果浏览器不接受这些客户端代码, 该控件就只显示相应的内容(显示 TreeView 控件需要的所有信

息)。它只能处理客户端脚本的客户执行这些基于 JavaScript 的回送。

如果在受某种 HTTP 嗅探器(用于监控网络上的通信)监控的页面上运行 TreeView 控件, 那么肯定可以看到 TreeView 控件显示的内容。



大多数 Web 浏览器都带有便于嗅探网络通信的内置工具; Firefox 用户应从 getfirebug.com 上安装 Firebug 扩展, 另一个很好的嗅探器是 Fiddler, 可以从 fiddler2.com 网站上获得。

如果浏览器支持客户端脚本, 在展开 TreeView 控件的一个节点后, HTTP 请求如下所示:

```
POST /Navigation/Default.aspx HTTP/1.1
Accept: */*
Accept-Language: en-us
Referrer: http://localhost:1882/Navigation/Default.aspx
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0)

DNT: 1
Host: localhost:1882
Content-Length: 904
Proxy-Connection: Keep-Alive
Pragma: no-cache

__EVENTTARGET=&__EVENTARGUMENT=&TreeView1_ExpandState=c&TreeView1_SelectedNode=Tree
Viewlt0&TreeView1_PopulateLog=&__VIEWSTATE=%2FwEPDwUKLTU0ODk0OTE2Mg9kFgICBA9kFgICAw
88KwAJAgAPFggeDU5ldmVYRXhwYW5kZWZkHgtfIURhdGFkY3VudGVjdGVkTm9kZQULVHJlZVZpZ
XcxddAeCUxhc3RjbmlleAIBZAgUKWACBQmWojAUKWACFhAeBFRleHQFBHhVUeBVZhbHVlBQRib21lHgtO
YXZpZ2F0ZVYyYUUYL05hdmlnYXRpb24vRGVmYXVsdC5hc3B4HgdUb29sVG1wBQlIb21lIFBhZ2UeCERhdGF
QYXRoBRgubmF2aWdhZGlubi9kZWZhdWw0LmFzcHgeCURhdGFkY3VudGVjdGVkZx4QUG9wdWxhdG
VPbkRlbWFuZGdkZBgBBR5fX0NvbnRyb2xzUmVxdWlyZVBvc3RCYWNrS2V5X18WAwURTG9naW4xJFJlbWVtY
mVyTWUFF0xvZ2luMSRMb2dpbkltYWdlQnV0dG9uBQlUcmVlVmlldzFtwszVpUMxFTDtpERnNjgEIkWWbg%3
D%3D&Login1$UserName=&Login1$Password=&__CALLBACKID=TreeView1&__CALLBACKPARAM=0%7C1
%7Ctft%7C4%7CHome24%7C%2Fnavigation%2Fdefault.aspxHome&__EVENTVALIDATION=%2FwEWBgKg
8Yn8DwKUvNalDwL666vYDAKC0q%2BkBgKnz4ybCAKn5fLxBaSy6WQwPagNZsHisWROJfuiopOe
```

ASP.NET 应用程序的响应不是包含 TreeView 控件的整个页面, 而是一小段 HTML, 由页面上的一个 JavaScript 方法使用, 并动态地加载到 TreeView 控件。示例响应如下所示:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/8.0
Date: Sat, 10 Nov 2012 17:55:03 GMT
X-AspNet-Version: 4.0.30319
Cache-Control: private, no-store
Content-Type: text/html; charset=utf-8
Content-Length: 1756
Connection: Close

112|/wEWCgKg8Yn8DwKUvNalDwL666vYDAKC0q+kBgKnz4ybCAKn5fLxBQKAgtPaBALEmcbhCgK8nZDfCAL
M/ZK8AR/nFcl4nlPgp6HcFlU6YiFBfoNM14|nn|<div id="TreeViewln6Nodes"
style="display:none;">
```

```

<table cellpadding="0" cellspacing="0" style="border-width:0;">
  <tr>
    <td><div style="width:20px;height:1px"></div></td><td><div
      style="width:20px;height:1px">
      </div></td><td><div style="width:20px;height:1px">
      </div></td><td></td><td style="white-space:nowrap;">
      <a href="/Navigation/MarketsUSasdf.aspx"
        title="Looking at the U.S. Market" id="TreeView1t12"
        style="text-decoration:none;">U.S. Market Report</a></td>
    </tr>
  </table><table cellpadding="0" cellspacing="0" style="border-width:0;">
    <tr>
      <td><div style="width:20px;height:1px"></div></td><td><div
        style="width:20px;height:1px">
        </div></td><td><div style="width:20px;height:1px">
        </div></td><td>
        </td><td style="white-space:nowrap;">
      <a href="/Navigation/NYSE.aspx" title="The New York Stock Exchange"
        id="TreeView1t13" style="text-decoration:none;">NYSE</a></td>
    </tr>
  </table>
</div>

```

回送功能相当强大,但如果要禁用它(甚至包括为具有该功能的浏览器禁用它),只需把 `TreeView` 控件的 `PopulateNodesFromClient` 属性设置为 `false`(默认值为 `true`)即可。

`TreeView` 控件的可定制性很高,但首先看看如何使用程序清单 17-1 中的 `.sitemap` 文件创建该控件的默认版本。这个例子继续使用前面创建的 `MarketsUS.aspx` 页面。

首先在页面上创建一个 `SiteMapDataSource` 控件。在使用 `TreeView` 控件显示 `.sitemap` 文件的内容时,必须使用数据源控件。`TreeView` 控件不像 `SiteMapPath` 控件那样能自动绑定到站点地图文件。

有了一个基本的 `SiteMapDataSource` 控件后,就把一个 `TreeView` 控件放在页面上,将 `DataSourceId` 属性设置为 `SiteMapDataSource1`。完成上述操作后,代码如程序清单 17-6 所示。

程序清单 17-6 一个基本的 `TreeView` 控件

```

<%@ Page Language="C#" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">

```



```

<title>Using the TreeView Server Control</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:SiteMapPath ID="SiteMapPath1" runat="server">
    </asp:SiteMapPath>
    <br /><p>
    <asp:TreeView ID="TreeView1" runat="server"
      DataSourceID="SiteMapDataSource1">
    </asp:TreeView>
    <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" /></p>
  </form>
</body>
</html>

```

运行页面，展开 TreeView 控件，结果如图 17-9 所示。

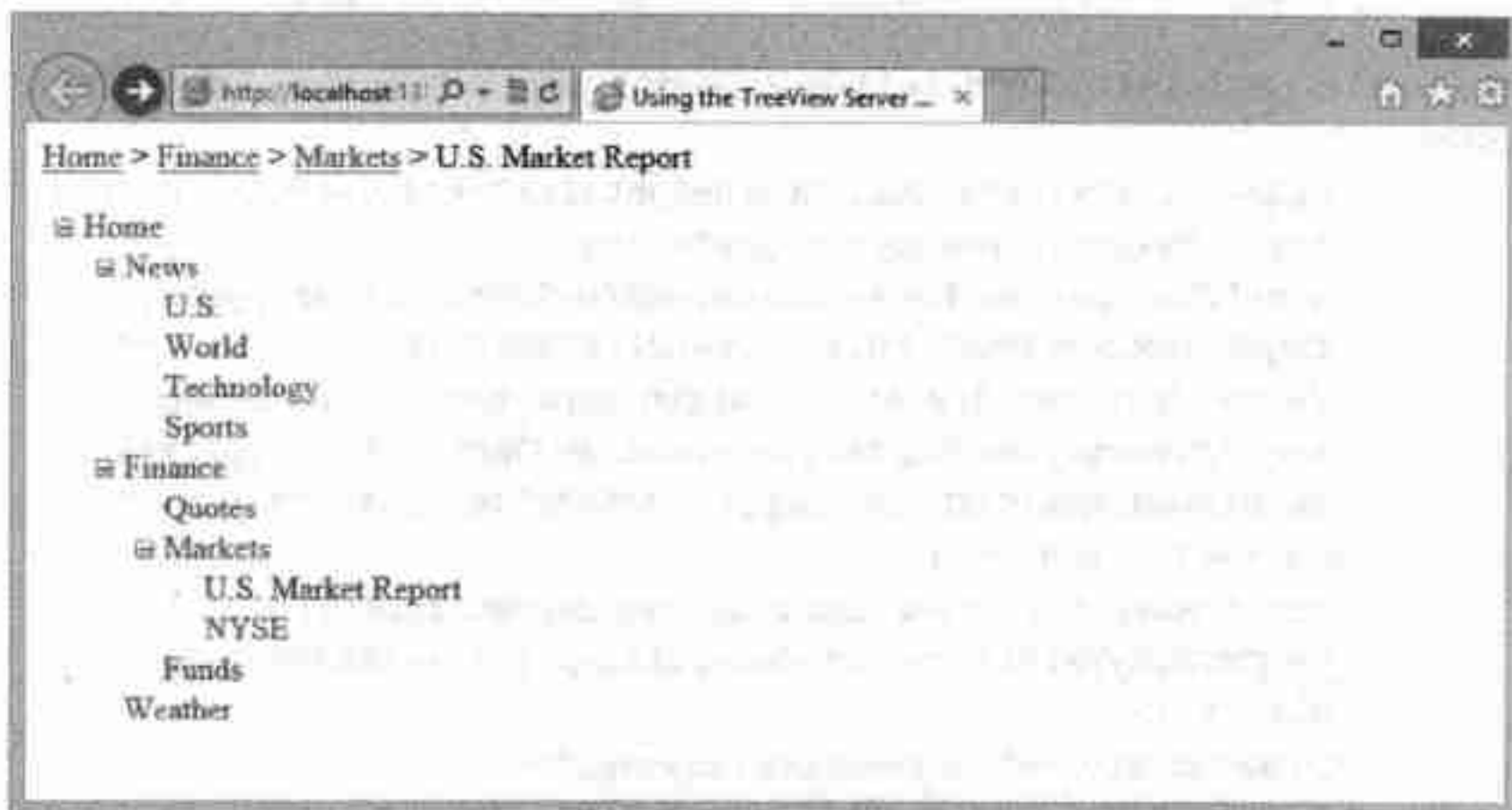


图 17-9

这是一个非常基本的 TreeView 控件。这个控件的优点是允许进行高度定制，甚至可以使用 ASP.NET 4.5 中预定义的样式。

17.3.1 标识 TreeView 控件的内置样式

如前所述，TreeView 控件有许多内置样式。利用这些内置样式的最佳方式是在页面的设计视图中进行操作。在 Visual Studio 2012 的设计视图中，单击 TreeView 服务器控件右上角的箭头，找到 Auto Format 选项。单击这个选项，就可以使用许多样式。选择其中一个样式就会修改 TreeView 控件的代码，使之适应选中的样式。例如，从选项列表中选择 MSDN，我们创建的 TreeView 控件就转换为应用 MSDN 样式，如程序清单 17-7 所示。

程序清单 17-7 应用了 MSDN 样式的 TreeView 控件

```

<asp:TreeView ID="TreeView1" runat="server" DataSourceID="SiteMapDataSource1"
  ImageSet="Msdn" NodeIndent="10">
  <ParentNodeStyle Font-Bold="False" />
  <HoverNodeStyle BackColor="#CCCCCC" BorderColor="#888888" BorderStyle="Solid"
    Font-Underline="True" />

```



```

<SelectedNodeStyle BackColor="White" BorderColor="#888888" BorderStyle="Solid"
  BorderWidth="1px" Font-Underline="False" HorizontalPadding="3px"
  VerticalPadding="1px" />
<NodeStyle Font-Names="Verdana" Font-Size="8pt" ForeColor="Black"
  HorizontalPadding="5px" NodeSpacing="1px" VerticalPadding="2px" />
</asp:TreeView>

```

使用这些内置样式，就可以完全改变 TreeView 控件的外观和操作方式。运行这段代码，结果如图 17-10 所示。

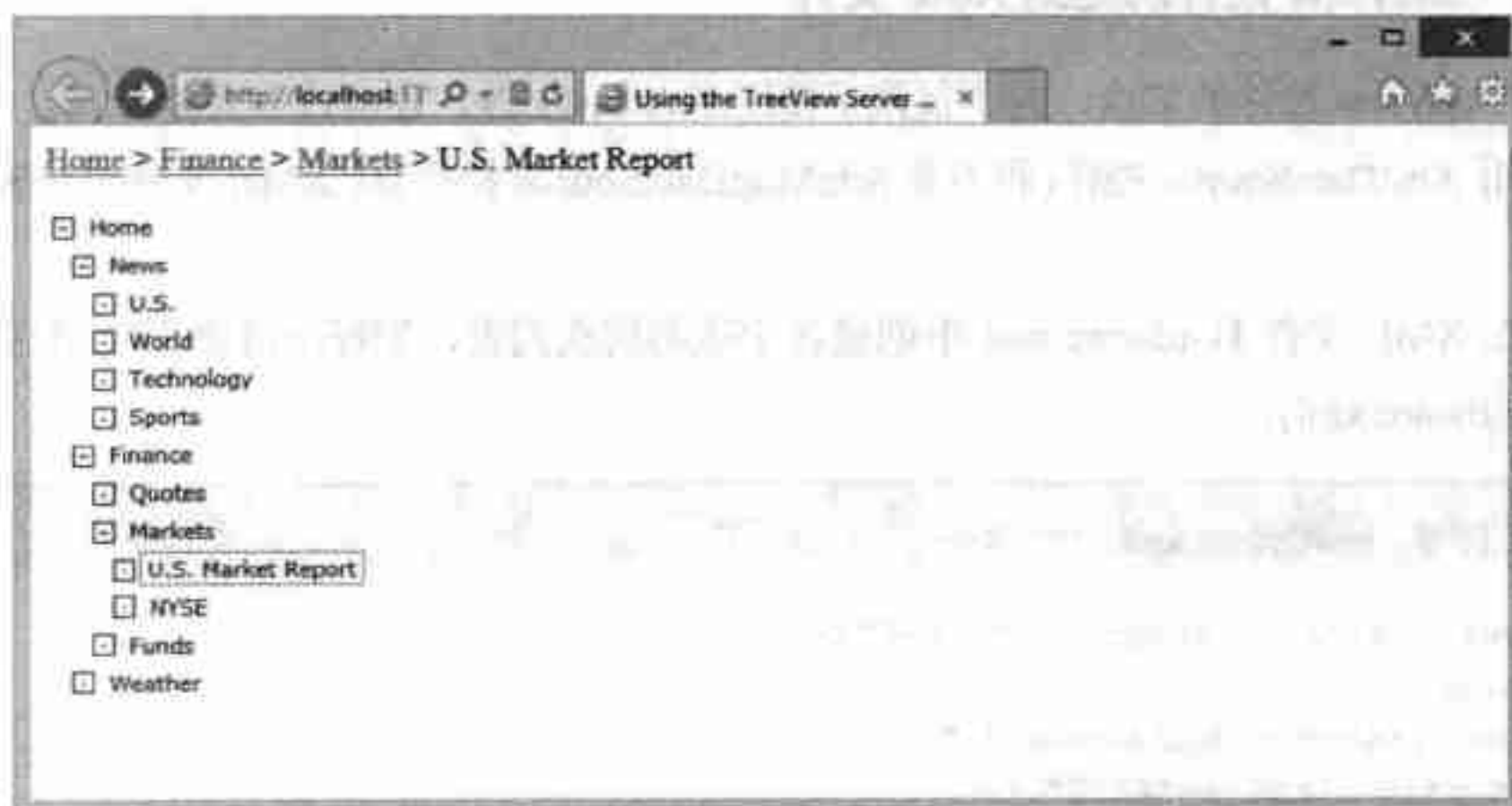


图 17-10

17.3.2 研究 TreeView 控件的各个部分

为了掌握 TreeView 控件的用法，你必须理解该控件创建的层次结构树的每个部分所使用的术语。

TreeView 控件中的每个元素或每个条目都称为节点。层次结构中最上面的节点是根节点。TreeView 控件可以有多个根节点。在层次结构中，如果任何节点(包括根节点在内)的下面还有节点，就将该节点称为父节点。父节点下面的直接节点则称为子节点。每个父节点可以有一个或多个子节点。如果节点不包含子节点，就称为叶节点。

下面根据前面的站点地图详细说明这些术语的用法：

Home - Root node, parent node

News - Parent node, child node

U.S. - Child node, leaf node

World - Child node, leaf node

Technology - Child node, leaf node

Sports - Child node, leaf node

Finance - Parent node, child node

Quotes - Child node, leaf node

Markets - Parent node, child node

U.S. Market Report - Child node, leaf node

NYSE - Child node, leaf node

Funds - Child node, leaf node

Weather - Child node, leaf node

在这个列表中,可以看出每个节点是什么,以及在节点层次结构中如何表示它。例如,U.S. Market Report 节点是一个叶节点,表示它没有与之相关的子节点。但是,它还是 Markets 节点的子节点,Markets 节点是 U.S. Market Report 节点的父节点。如果正在处理 Markets 节点,那么它还是 Finance 节点的子节点,Finance 节点是它的父节点。要点是站点地图层次结构中的每个节点都与层次结构中的其他节点相关。必须理解这些关系,才能编程处理这些节点(如本章后面所述),用于处理它们的方法包括 RootNode、CurrentNode 和 ParentNode。

17.3.3 将 TreeView 控件绑定到 XML 文件

要填充 TreeView 控件的节点,并不仅限于使用.sitemap 文件,还有许多其他方式。一种比较好的方式是使用 XmlDataSource 控件(而不是 SiteMapDataSource 控件)从 XML 文件中填充 TreeView 控件。

例如,在 XML 文件 Hardware.xml 中创建各个项的层次列表,如程序清单 17-8 所示(本章下载代码中的 Hardware.xml)。

程序清单 17-8 Hardware.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Hardware>
  <Item Category="Motherboards">
    <Option Choice="ASUS" />
    <Option Choice="MSI" />
  </Item>
  <Item Category="Memory">
    <Option Choice="2048mb" />
    <Option Choice="4096mb" />
    <Option Choice="8192mb" />
  </Item>
  <Item Category="HardDrives">
    <Option Choice="250GB" />
    <Option Choice="500GB" />
    <Option Choice="750GB" />
  </Item>
  <Item Category="Drives">
    <Option Choice="DVD Burner" />
    <Option Choice="Blu-ray" />
    <Option Choice="Blu-ray Burner" />
  </Item>
</Hardware>
```

可以看出,这个列表不能用于站点导航,但允许终端用户从选项的层次列表中选择。这个 XML 文件分为 4 个选项类别: Motherboards、Memory、HardDrives 和 Drives。要把 TreeView 控件绑定到这个 XML 文件,可以使用 XmlDataSource 控件指定要使用的 XML 文件的位置。然后,在 TreeView 控件中使用<asp:TreeNodeBinding>元素指定要绑定 XML 文件中的哪些元素,以填充 TreeView 控件的节点,如程序清单 17-9 所示。

程序清单 17-9 把 TreeView 控件绑定到 Hardware.xml 文件

```

<%@ Page Language="C#" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Latest Hardware</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:TreeView ID="TreeView1" runat="server" DataSourceID="XmlDataSource1">
            <DataBindings>
                <asp:TreeNodeBinding DataMember="Hardware"
                    Text="Computer Hardware" />
                <asp:TreeNodeBinding DataMember="Item" TextField="Category" />
                <asp:TreeNodeBinding DataMember="Option" TextField="Choice" />
            </DataBindings>
        </asp:TreeView>
        <asp:XmlDataSource ID="XmlDataSource1" runat="server"
            DataFile="Hardware.xml">
        </asp:XmlDataSource>
    </form>
</body>
</html>

```

此处要介绍的第一项是<asp:XmlDataSource>控件，它与前面的<asp:SiteMapDataSource>控件一样简单，但使用 DataFile 属性指向 Hardware.xml 文件。

下一步是创建一个 TreeView 控件，将它绑定到这个 XML 文件。可以把默认的 TreeView 控件直接绑定到 XmlDataSource 控件，如下所示：

```
<asp:TreeView ID="TreeView1" runat="server" DataSourceID="XmlDataSource1" />
```

完成上述操作后，结果并不正确，如图 17-11 所示。



图 17-11

可以看出，TreeView 控件已绑定到 Hardware.xml 文件，但查看 TreeView 控件中的节点就会发

现，它只显示了文件中 XML 元素的名称。这不是我们想要的，因此应在 TreeView 控件中使用 <DataBindings>元素指定如何绑定到 XML 文件。

<DataBindings>元素封装了一个或多个 TreeNodeBinding 对象。TreeNodeBinding 对象的两个比较重要的属性是 DataMember 和 TextField。DataMember 属性指向 TreeView 控件要查找的 XML 元素的名称，TextField 属性指定 TreeView 在该 XML 元素中查找的 XML 属性。如果操作正确，使用 <DataBindings>元素会得到如图 17-12 所示的结果。



图 17-12

在程序清单 17-9 中，可以重写 XML 文件中根节点的文本值<Hardware>，在 TreeView 控件中把它显示为 Computer Hardware：

```
<asp:TreeNodeBinding DataMember="Hardware" Text="Computer Hardware" />
```

17.3.4 在 TreeView 中选择多个选项

如前所述，TreeView 控件不用于导航，但可以用于其他用途。在许多情况下，TreeView 控件可以显示层次结构列表，让终端用户从中选择一项或多项。

TreeView 控件的一项重要内置功能是在列表把复选框放在层次结构项中的节点旁边。这些复选框可以让终端用户进行多项选择。TreeView 控件的 ShowCheckBoxes 属性可用于在项列表中给许多不同类型的节点旁边创建复选框。ShowCheckBoxes 属性的值如表 17-3 所示。

表 17-3

值	说 明
All	给 TreeView 控件中的每个节点应用复选框
Leaf	给没有子元素的节点应用复选框
None	不给 TreeView 控件中的任何节点应用复选框
Parent	只给 TreeView 控件中的父节点应用复选框。父节点至少有一个与之相关的子节点
Root	给 TreeView 控件中的每个根节点应用复选框

在使用 ShowCheckBoxes 属性时, 可以在控件中明确设置它, 如下所示:

```
<asp:TreeView ID="Treeview1" runat="server" Font-Underline="false"
  DataSourceID="XmlDataSource1" ShowCheckBoxes="Leaf">
  ...
</asp:TreeView>
```

或者使用下面的代码以编程方式设置它:

```
TreeView1.ShowCheckBoxes = TreeNodeTypes.Leaf;
```

为了演示如何在 TreeView 控件中使用复选框, 下面继续扩展程序清单 17-9 中的示例。创建一个层次结构列表, 让人们从列表中选择多项以接收更多的信息, 如程序清单 17-10 所示。

程序清单 17-10 在节点的层次结构列表中对叶节点应用复选框

```
<%@ Page Language="C#" %>
<script runat="server">
  protected void Button1_Click(object sender, System.EventArgs e)
  {
    if(TreeView1.CheckedNodes.Count > 0)
    {
      Labell.Text = "We are sending you information on:<p>";
      foreach(TreeNode node in TreeView1.CheckedNodes)
      {
        Labell.Text += node.Text + " " + node.Parent.Text + "<br>";
      }
    }
    else
    {
      Labell.Text = "You didn't select anything. Sorry!";
    }
  }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Latest Hardware</title>
</head>
<body>
  <form runat="server">
    Please select the items you are interested in:
    <p>
    <asp:TreeView ID="TreeView1" runat="server" Font-Underline="False"
      DataSourceID="XmlDataSource1" ShowCheckBoxes="Leaf">
      <DataBindings>
        <asp:TreeNodeBinding DataMember="Hardware"
          Text="Computer Hardware" />
        <asp:TreeNodeBinding DataMember="Item" TextField="Category" />
        <asp:TreeNodeBinding DataMember="Option" TextField="Choice" />
      </DataBindings>
    </asp:TreeView>
    <p>
    <asp:Button ID="Button1" runat="server" Text="Submit Choices"
      OnClick="Button1_Click" />
  </form>
</body>
</html>
```

```

</p>
<asp:XmlDataSource ID="XmlDataSource1" runat="server"
    DataFile="Hardware.xml">
</asp:XmlDataSource>
</p>
<asp:Label ID="Label1" runat="server" />
</form>
</body>
</html>

```

在这个例子中，首先把 ShowTextBoxes 属性设置为 Leaf，表示在 TreeView 控件中仅为不包含任何子节点的项添加复选框。带复选框的项应是展开层次结构列表后的最后一项。

设置了这个属性后，就要在 Button1_Click 事件中处理终端用户选择的项。首先要检查是否选择了项：

```

if (TreeView1.CheckedNodes.Count > 0) {
    ...
}

```

在这个例子中，回送时选中节点的数量必须大于 0，表示至少选择了一项。如果大于 0，就可以执行 if 语句中的代码，然后 if 语句会填充页面上的 Label 控件。要使用选中节点的数据填充 Label 控件，需要使用一条 foreach 语句，如下所示：

```

foreach (TreeNode node in TreeView1.CheckedNodes) {
    ...
}

```

这会创建一个 TreeNode 对象实例，在选中节点的 TreeView1 集合中检查每个 TreeNode 对象。

对于选中的每个节点，需要获取节点及其父节点的 Text 值，以进一步填充 Label 控件，如下所示：

```
Label1.Text += node.Text + " " + node.Parent.Text + "<br>";
```

最后，页面生成的结果如图 17-13 所示。



图 17-13

17.3.5 在 TreeView 控件中指定定制的图标

可以对 TreeView 控件进行高度定制。本章前面提到,通过指定内置样式很容易定义 TreeView 控件的外观和操作方式。应用其中一个样式,可以显著地改变该控件的外观。一处最引人注目的改动是 TreeView 控件中用于节点的图标。还可以把自己的图标应用于节点层次结构列表中的节点,但这不如选择 TreeView 控件中内置的样式那么简单。

TreeView 控件包含如表 17-4 所示的属性,这些属性可以为控件的节点指定图像。

表 17-4

属 性	说 明
CollapseImageUrl	如果展开后节点显示出有子节点,而且可以折叠,就对该节点应用定制的图像
ExpandImageUrl	如果节点可以展开以显示其子节点,就对该节点应用定制的图像
LeafNodeStyle-ImageUrl	如果节点没有子节点,并且位于节点层次结构链的最后,就对该节点应用定制的图像
NoExpandImageUrl	如果编程时节点不能展开,或者它本身是叶节点,就对该节点应用定制的图像。该属性主要用于添加空格,使叶节点与其父节点对齐
ParentNodeStyle-ImageUrl	仅对 TreeView 控件中的父节点应用定制的图像
RootNodeStyle-ImageUrl	仅对 TreeView 控件中的根节点应用定制的图像

程序清单 17-11 是这些属性的用法示例。

程序清单 17-11 对 TreeView 控件应用定制的图像

```
<asp:TreeView ID="TreeView1" runat="server" Font-Underline="False"
    DataSourceId="XmlDataSource1"
    CollapseImageUrl="Images/CollapseImage.png"
    ExpandImageUrl="Images/ExpandImage.png"
    LeafNodeStyle-ImageUrl="Images/LeafImage.png">
    <DataBindings>
        <asp:TreeNodeBinding DataMember="Hardware" Text="Computer Hardware" />
        <asp:TreeNodeBinding DataMember="Item" TextField="Category" />
        <asp:TreeNodeBinding DataMember="Option" TextField="Choice" />
    </DataBindings>
</asp:TreeView>
```

在控件的节点前面指定这 3 幅图像,会重写使用加号(+)和减号(-)来展开和折叠节点的默认值。当默认不使用任何图像时,还可以为任意叶节点使用一幅图像来覆盖默认值。运行程序清单 17-11 的代码,结果如图 17-14 所示(当然,效果取决于读者使用的图像)。



图 17-14

17.3.6 指定用于连接节点的线

TreeView 控件把项的层次结构列表显示给终端用户，在默认情况下，这些层次结构项之间的关系显示得不是很清晰，因此有时我们希望把它们显示得更清晰一些。一种方式是在父节点和子节点之间显示连接线，这就需要把 TreeView 控件的 ShowLines 属性设置为 True(该属性默认设置为 False):

```
<asp:TreeView ID="TreeView1" runat="server" Font-Underline="False"
  DataSourceId="XmlDataSource1" ShowCheckBoxes="Leaf" ShowLines="True">
  ...
</asp:TreeView>
```

执行这段代码得到的结果如图 17-15 所示。



图 17-15

如果 ShowLines 属性设置为 True，还可以在 TreeView 控件中定义自己的线和图像。这是很容易实现的操作，因为 Visual Studio 2012 提供了 ASP.NET TreeView Line Image Generator 工具。这个工

具可以可视化地设计线的外观以及相关的展开和折叠图像。建立好自己的线和图像后，该工具就会创建所有必需的文件，供 TreeView 控件使用。

要获得这个工具，可以进入文件的设计视图，单击页面上 TreeView 控件的智能标记。找到 Customize Line Images 选项，只有选中了 Show Lines 复选框才能看到这个选项。单击该选项，就会打开 ASP.NET TreeView Line Image Generator 对话框，如图 17-16 所示。

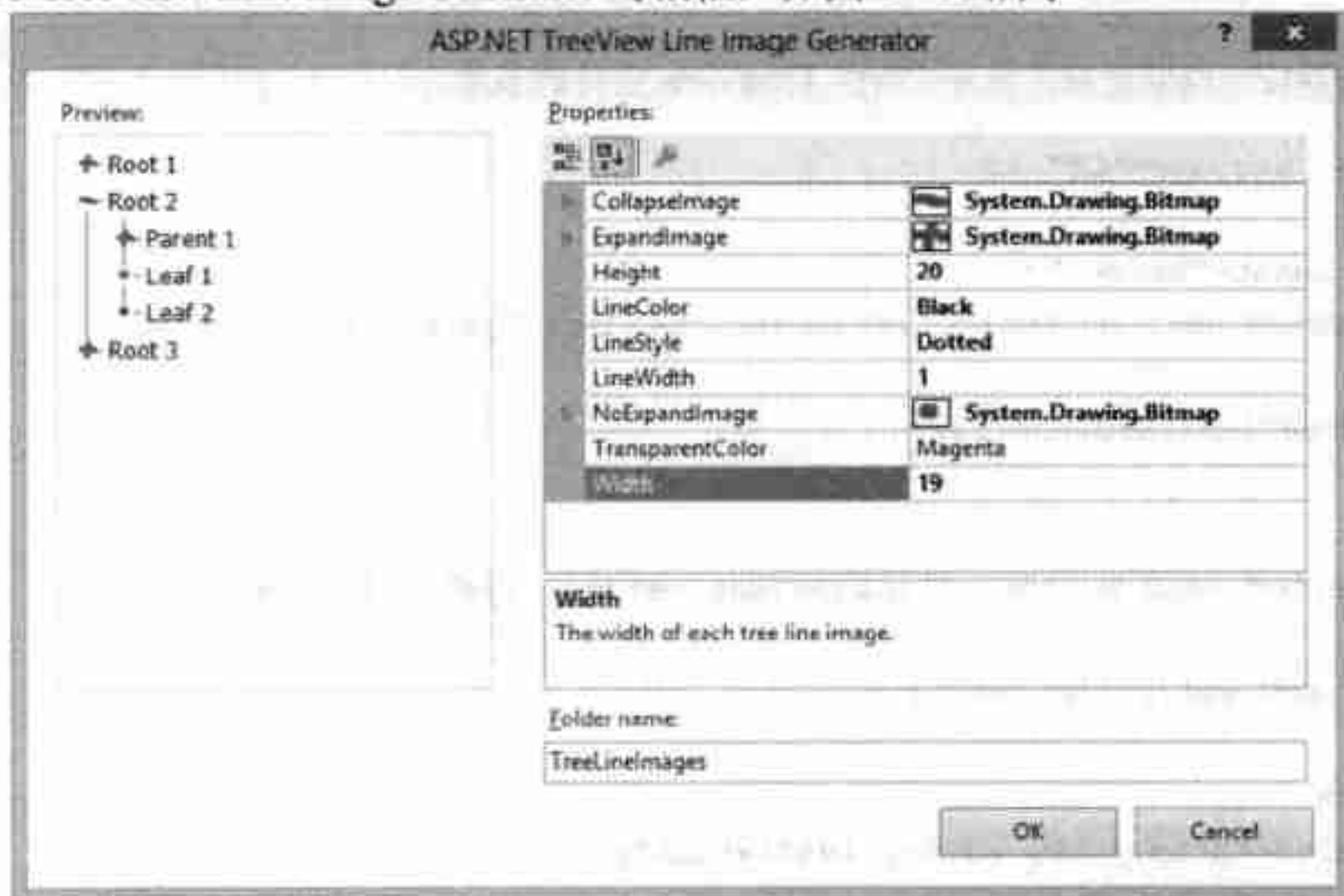


图 17-16

在这个对话框中，可以为需要 Expand、Collapse 或 NoCollapse 图标的节点选择图像。还可以指定连接节点的线的颜色和样式。在创建自己的样式时，对话框会根据所使用的样式直接显示 TreeView 控件的示例输出。最后一步是选择这个对话框创建的文件输出。完成这一步后，单击 OK 按钮，在对话框指定的文件夹中就会生成一个很长的新文件列表。默认情况下，ASP.NET TreeView Line Image Generator 把输出文件夹命名为 TreeLineImages，但用户可以把该文件夹命名为任意名称。如果项目中不存在这个文件夹，Visual Studio 就会创建它。之后，TreeView 控件就可以通过设置 LineImagesFolder 属性来使用新的图像和样式，如下所示：

```
<asp:TreeView ID="TreeView1" runat="server" ShowLines="True"
DataSourceId="SiteMapDataSource1" LineImagesFolder="TreeViewLineImages">
```

其中重要的属性显示为粗体。ShowLines 属性必须设置为 True。之后，该属性使用前面显示的默认设置。如果指定了位置，就使用 LineImagesFolder 属性从中提取定制的图像和样式。可以看出，这里该属性指向新文件夹 TreeViewLineImages，其中包含了我们创建的所有新图像和样式。查看该文件夹，了解一下该工具生成了哪些输出。

17.3.7 以编程方式使用 TreeView 控件

前面介绍了如何声明性地使用 TreeView 控件。ASP.NET 的优点之一是不但可以声明性地使用它的组件，还可以以编程方式操作这些控件。

TreeView 控件有相关的 TreeView 类，可以完全在代码中管理 TreeView 控件及其操作方式。下面介绍如何使用较常见的方式编程控制 TreeView。

1. 以编程方式展开和折叠节点

使用 `TreeView` 控件可以在层次结构中以编程方式展开或折叠节点。为此, 要使用 `TreeView` 类的 `ExpandAll` 或 `CollapseAll` 方法。程序清单 17-12 显示了前面在程序清单 17-6 中使用的 `TreeView` 控件, 但在它的上面添加了两个按钮, 用于启动节点的展开和折叠功能。

程序清单 17-12 以编程方式展开或折叠 `TreeView` 控件的节点

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, System.EventArgs e)
    {
        TreeView1.ExpandAll();
    }

    protected void Button2_Click(object sender, System.EventArgs e)
    {
        TreeView1.CollapseAll();
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>TreeView Control</title>
</head>
<body>
    <form id="Form1" runat="server">
        <p>
            <asp:Button ID="Button1" runat="server" Text="Expand Nodes"
                OnClick="Button1_Click" />
            <asp:Button ID="Button2" runat="server" Text="Collapse Nodes"
                OnClick="Button2_Click" />
            <br />
            <br />
            <asp:TreeView ID="TreeView1" runat="server"
                DataSourceId="SiteMapDataSource1">
            </asp:TreeView>
            <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" /></p>
        </form>
    </body>
</html>
```

运行这个页面, 会在 `TreeView` 控件的上面显示两个按钮。单击第一个按钮, 调用 `ExpandAll` 方法, 展开整个节点列表。单击第二个按钮, 调用 `CollapseAll` 方法, 把所有的节点折叠起来, 如图 17-17 所示。



图 17-17

程序清单 17-12 中的例子非常经典,但它只根据终端用户的操作(终端用户单击按钮)来展开和折叠节点。最好可以编程启动这些操作。

我们可能希望把 `TreeView1.CollapseAll()` 命令放在 `Page_Load` 事件中,但如果这么做,代码就不能运行。因此,应使用 `TreeView` 控件的 `OnDataBound` 属性:

```
<asp:TreeView ID="TreeView1" runat="server"
    DataSourceId="SiteMapDataSource1" OnDataBound="TreeView1_DataBound">
</asp:TreeView>
```

这个属性的值指向代码中的一个方法,如下所示:

```
protected void TreeView1_DataBound(object sender, System.EventArgs e)
{
    TreeView1.CollapseAll();
}
```

现在运行页面,注意在浏览器上第一次加载页面时, `TreeView` 控件会完全折叠。

还可以展开或折叠树中特定的节点,而不是展开或折叠整个列表。为此,使用刚才创建的 `TreeView1_DataBound` 方法。使用程序清单 17-1 中的站点地图,修改 `TreeView1_DataBound` 方法,如程序清单 17-13 所示。

程序清单 17-13 编程展开特定的节点

```
protected void TreeView1_DataBound(object sender, System.EventArgs e)
{
    TreeView1.CollapseAll();
    TreeView1.FindNode("Home").Expand();
    TreeView1.FindNode("Home/Finance").Expand();
    TreeView1.FindNode("Home/Finance/Markets").Expand();
}
```


在这个例子中, 使用 `FindNode` 方法并展开找到的节点。`FindNode` 方法带有一个字符串参数, 其值是要引用的节点和节点路径。例如, `TreeView1.FindNode("Home/Finance").Expand()` 展开 `Finance` 节点。要找到节点, 必须指定从根节点到要处理的节点(这里是 `Finance` 节点)的完整路径。在站点地图路径结构中, 使用正斜杠分隔站点地图路径的各个节点。

注意, 必须依次展开每个节点, 直到到达 `Finance` 节点为止。如果在 `TreeView1_DataBound` 方法中只使用 `TreeView1.FindNode("Home/Finance/Markets").Expand()`, `Markets` 节点会展开, 但它上面的父节点 `Finance` 和 `Home` 不会展开, 并且在调用页面时看不到展开的 `Markets` 节点。

如果不使用 `Expand` 方法, 还可以把 `Expanded` 属性设置为 `True`, 如程序清单 17-14 所示。

程序清单 17-14 使用 `Expanded` 属性编程展开节点

```
protected void TreeView1_DataBound(object sender, System.EventArgs e)
{
    TreeView1.CollapseAll();
    TreeView1.FindNode("Home").Expanded = true;
    TreeView1.FindNode("Home/Finance").Expanded = true;
    TreeView1.FindNode("Home/Finance/Markets").Expanded = true;
}
```

这里主要讨论的是 `Expand` 方法和 `Expanded` 属性, 但是还可以使用 `Collapse` 方法以编程方式来折叠节点。然而, 实际上不存在 `Collapsed` 属性, 而是可以把 `Expanded` 属性设置为 `False`。

2. 添加节点

使用 `TreeView` 控件可以完成的另一项有趣任务是给整个层次结构以编程方式添加节点。`TreeView` 控件由 `TreeNode` 对象的集合组成。如前面的例子所示, `Finance` 节点实际上是可以编程处理的 `TreeNode` 对象。还可以向该控件添加其他 `TreeNode` 对象。

通常情况下, `TreeNode` 对象有 `Text` 和 `Value` 属性。`Text` 属性是 `TreeView` 控件中显示给终端用户的内容。`Value` 属性是附加的数据项, 可用于关联 `TreeNode` 对象。另一个可以使用的属性(假定 `TreeView` 控件是导航链接列表)是 `NavigateUrl`。程序清单 17-15 演示了如何在程序清单 17-1 使用的站点地图中以编程方式添加节点。

程序清单 17-15 给 `TreeView` 控件以编程方式添加节点

```
<%@ Page Language="C#" %>

<script runat="server">
protected void Button3_Click(object sender, System.EventArgs e)
{
    TreeNode myNode = new TreeNode();
    myNode.Text = TextBox1.Text;
    myNode.NavigateUrl = TextBox2.Text;
    TreeView1.FindNode("Home/Finance/Markets").ChildNodes.Add(myNode);
}
</script>
```



```

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>TreeView Control</title>
</head>
<body>
    <form id="Form1" runat="server">
        <p>
            <asp:Button ID="Button1" runat="server" Text="Expand Nodes"
                OnClick="Button1_Click" />
            <asp:Button ID="Button2" runat="server" Text="Collapse Nodes"
                OnClick="Button2_Click" /></p>
        <p>
            <strong>Text of new node:</strong>
            <asp:TextBox ID="TextBox1" runat="server">
            </asp:TextBox>
        </p>
        <p>
            <strong>Destination URL of new node:</strong>
            <asp:TextBox ID="TextBox2" runat="server">
            </asp:TextBox>
            <br />
            <br />
            <asp:Button ID="Button3" runat="server" Text="Add New Node"
                OnClick="Button3_Click" />
        </p>
        <p>
            <asp:TreeView ID="TreeView1" runat="server"
                DataSourceId="SiteMapDataSource1">
            </asp:TreeView></p>
        <p>
            <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" /></p>
    </form>
</body>
</html>

```

这个页面包含两个文本框和一个新的 Button 控件。第一个文本框用于填充新建节点的 Text 属性，第二个文本框用于填充新建节点的 NavigateUrl 属性。

如果运行页面，那么单击 Expand Nodes 按钮就可以展开整个层次结构。接着给 Markets 节点添加子节点。要以编程方式添加新节点，应与前面一样使用 FindNode 方法查找 Markets 节点。找到该节点后，就可以使用 ChildNodes.Add() 方法，传送 TreeNode 对象实例，以此来添加子节点。提交第一个文本框中的 NASDAQ 和第二个文本框中的 Nasdaq.aspx，会使 TreeView 控件显示如图 17-18 所示的外观。

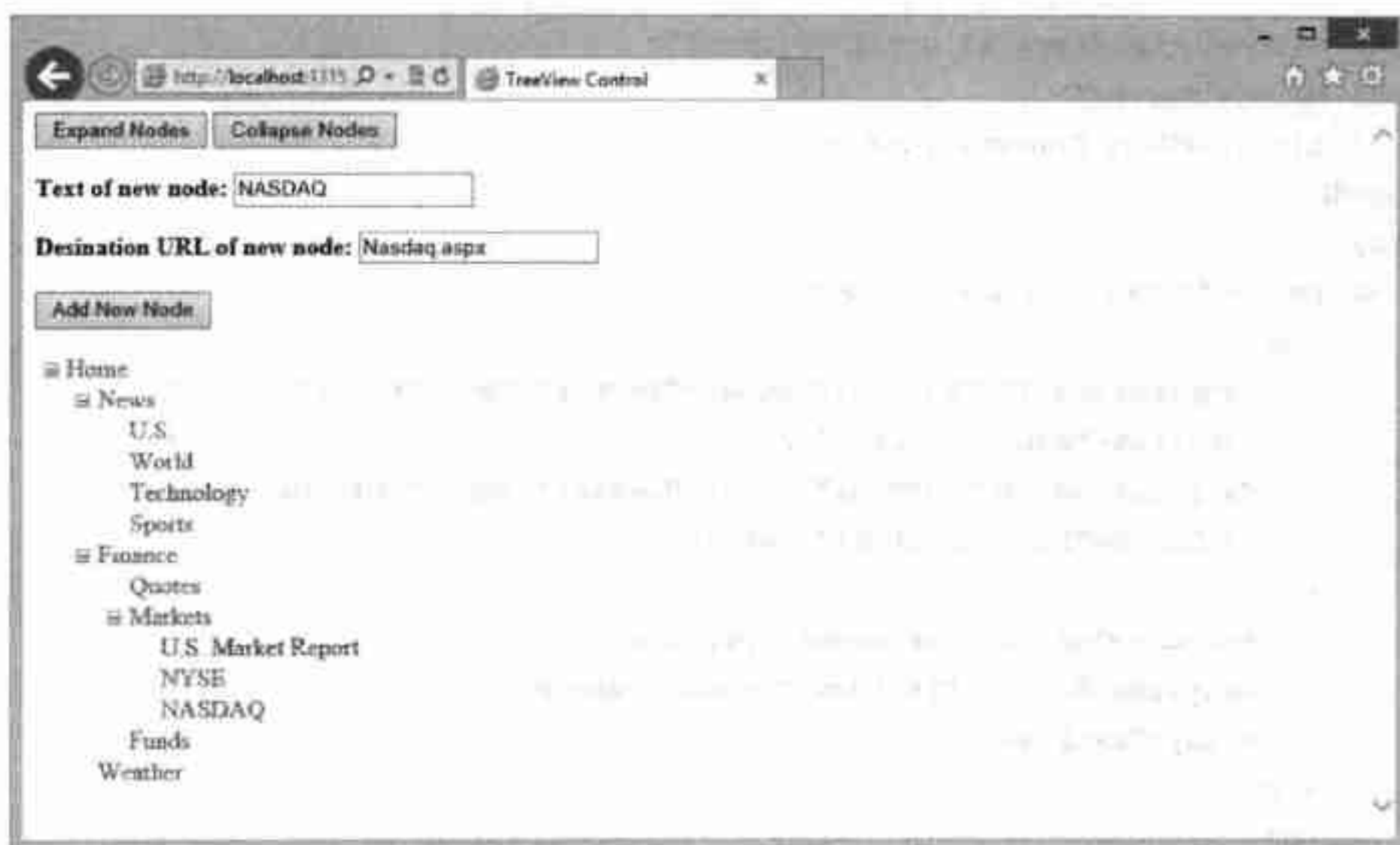


图 17-18

添加了节点后，即使把层次结构树折叠起来，再重新展开，该节点也仍然会被添加进去。也可以给 Markets 节点添加任意多个子节点。注意，虽然可以编程修改节点，但并不能改变数据源(XML 文件或.sitemap 文件)的内容。这些数据源在整个过程中保持不变。

17.4 Menu 服务器控件

ASP.NET 4.5 还有一个更酷的导航控件，即 Menu 服务器控件。这个控件很适合让终端用户在较大的选项层次结构中导航，但在这个过程中很少利用浏览器的资源。图 17-19 显示了在完全折叠或完全展开层次结构的一个分支时 Menu 控件的外观。

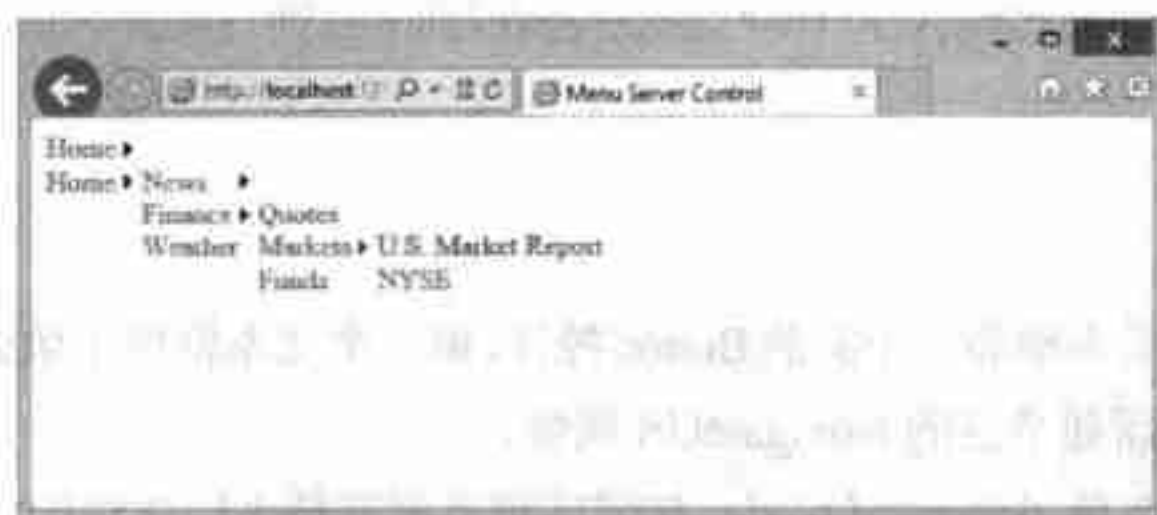


图 17-19

从图 17-19 中可以看出，所显示的第一个 Menu 控件是 Home 链接，在该链接的右边有一个小箭头，这个箭头表示在层次结构中这个顶层链接还包含更多的选项。第二个 Menu 控件显示了终端用户进入站点地图提供的一个分支时控件的默认外观。

当有许多选项时，无论这些选项是终端用户可以选择的选项，还是在应用程序中提供的导航点，使用 Menu 控件都非常理想。Menu 控件可以提供许多选项，并且在该过程中几乎不占用空间。

在 ASP.NET 应用程序中使用 Menu 控件是很简单的。Menu 控件要使用 SiteMapDataSource 控件。可以将 SiteMapDataSource 控件和 Menu 控件拖放到 Visual Studio 2012 设计界面上，使用 Menu 控件

的 DataSourceID 属性把它们连接起来。还可以直接在代码中创建和连接它们。程序清单 17-16 是最简单的 Menu 控件的示例。

程序清单 17-16 Menu 控件的简单用法

```
<%@ Page Language="C#" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Menu Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
        <asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1">
        </asp:Menu>
    </form>
</body>
</html>
```

在这个例子中，使用 SiteMapDataSource 控件自动处理应用程序的 Web.sitemap 文件。该例包含的另一项是 Menu 控件，它使用经典的 ID 和 runat 属性，并使用 DataSourceID 属性把自己连接到从 SiteMapDataSource 控件提取出来的数据。

默认的 Menu 控件非常简单，通过重新定义该控件的属性，可以高度定制它的外观和操作方式。下面就介绍几个修改 Menu 控件外观和操作方式的例子。

17.4.1 对 Menu 控件应用不同的样式

Menu 控件默认显示为平面外观。如果要保留该外观，可以使用该控件的默认属性设置，也可以改变字号和样式，使之符合站点的要求。实际上，可以有許多方式来修改这个控件，使它的外观比较新颖独特，并且符合站点的要求。可以定制这个控件的外观，也可以使用该控件附带的预定义样式。

1. 使用预定义样式

Visual Studio 2012 包含一些可以用于 Menu 控件的预定义样式，从而将外观和操作方式快速应用于显示的菜单项。所提供的样式有 Classic 和 Professional 等。要应用这些预定义样式，可以在页面的设计视图中使用 Menu 控件。在设计视图中，突出显示 Menu 控件，展开该控件的智能标记，此时可以看到用于处理该控件的一系列选项。要改变该控件的外观和操作方式，可单击 Auto Format 链接，选择一个样式。

执行这个操作会应用一系列样式属性，改变该控件的代码。例如，如果选择 Classic 选项，结果如程序清单 17-17 所示。

程序清单 17-17 把样式应用于 Menu 控件时代码的改动

```
<asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1"
    BackColor="#B5C7DE" ForeColor="#284E98"
    Font-Names="Verdana" Font-Size="0.8em" StaticSubMenuIndent="10px">
```



```

DynamicHorizontalOffset="2">
  <StaticSelectedStyle BackColor="#507CD1"></StaticSelectedStyle>
  <StaticMenuItemStyle HorizontalPadding="5"
    VerticalPadding="2"></StaticMenuItemStyle>
  <DynamicMenuStyle BackColor="#B5C7DE"></DynamicMenuStyle>
  <DynamicSelectedStyle BackColor="#507CD1"></DynamicSelectedStyle>
  <DynamicMenuItemStyle HorizontalPadding="5"
    VerticalPadding="2"></DynamicMenuItemStyle>
  <DynamicHoverStyle ForeColor="White" Font-Bold="True"
    BackColor="#284E98"></DynamicHoverStyle>
  <StaticHoverStyle ForeColor="White" Font-Bold="True"
    BackColor="#284E98"></StaticHoverStyle>
</asp:Menu>

```

在 Menu 控件中,有许多新增的样式可以改变显示在该控件中的菜单项。图 17-20 说明了这个样式选项显示在浏览器中的情况。

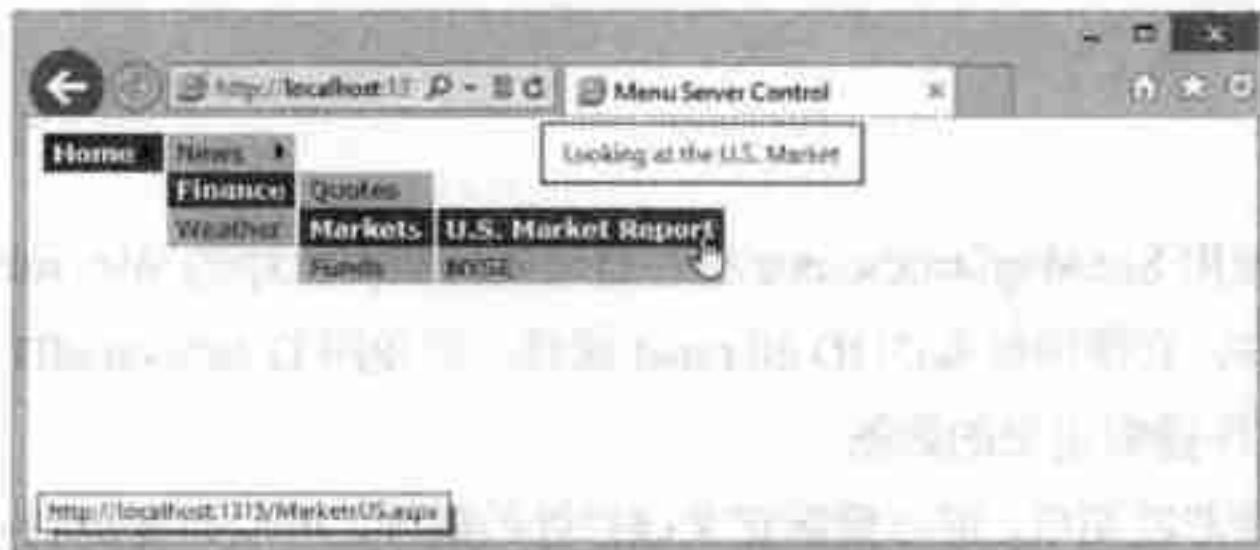


图 17-20

2. 改变静态项的样式

Menu 控件把层次结构中的项看成静态或动态的。这个例子中的静态项是生成页面时显示的 Home 链接。动态链接是用户在把光标停留在菜单中的 Home 链接上时动态出现的项。可以改变菜单中这两类节点的样式。

要把某个样式应用于静态链接,必须给 Menu 控件添加静态样式元素。Menu 控件包含如下静态样式元素:

- <StaticHoverStyle>
- <StaticItemTemplate>
- <StaticMenuItemStyle>
- <StaticMenuStyle>
- <StaticSelectedStyle>

在这个列表中,重要的选项有<StaticHoverStyle>和<StaticMenuItemStyle>元素。<StaticHoverStyle>元素用于定义当终端用户把光标停留在菜单中的静态项上时该静态项的样式。<StaticMenuItemStyle>元素用于定义当终端用户没有把光标停留在菜单中的静态项上时该静态项的样式。

程序清单 17-18 添加了一个样式,在终端用户把光标停留在静态项上时应用该样式。

程序清单 17-18 在 Menu 控件中给静态项添加光标停留样式

```

<asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1">
  <StaticHoverStyle BackColor="DarkGray" BorderColor="Black" BorderStyle="Solid"

```

```

        BorderWidth="1"></StaticHoverStyle>
</asp:Menu>

```

当终端用户把光标停留在菜单中的静态项上时，这个例子给该项添加了背景色和边框，结果如图 17-21 所示。



图 17-21

3. 给动态项添加样式

给 Menu 控件的动态项添加样式与给静态项添加样式一样简单。Menu 控件包含许多不同的元素，可用于修改动态项的外观，如下所示：

- <DynamicHoverStyle>
- <DynamicItemTemplate>
- <DynamicMenuItemStyle>
- <DynamicMenuStyle>
- <DynamicSelectedStyle>

这些元素修改菜单项的方式与它们的静态版本相同，但只改变从静态项中动态弹出的项。程序清单 17-19 是把光标停留样式应用于动态项的一个例子。

程序清单 17-19 把光标停留样式应用于 Menu 控件中的动态项

```

<asp:Menu ID="Menu1" runat="server" DataSourceID="Sitemapdatasource1">
  <StaticHoverStyle BackColor="DarkGray" BorderColor="Black" BorderStyle="Solid"
    BorderWidth="1"></StaticHoverStyle>
  <DynamicHoverStyle BackColor="DarkGray" BorderColor="Black" BorderStyle="Solid"
    BorderWidth="1"></DynamicHoverStyle>
</asp:Menu>

```

运行这段代码，生成的结果如图 17-22 所示。

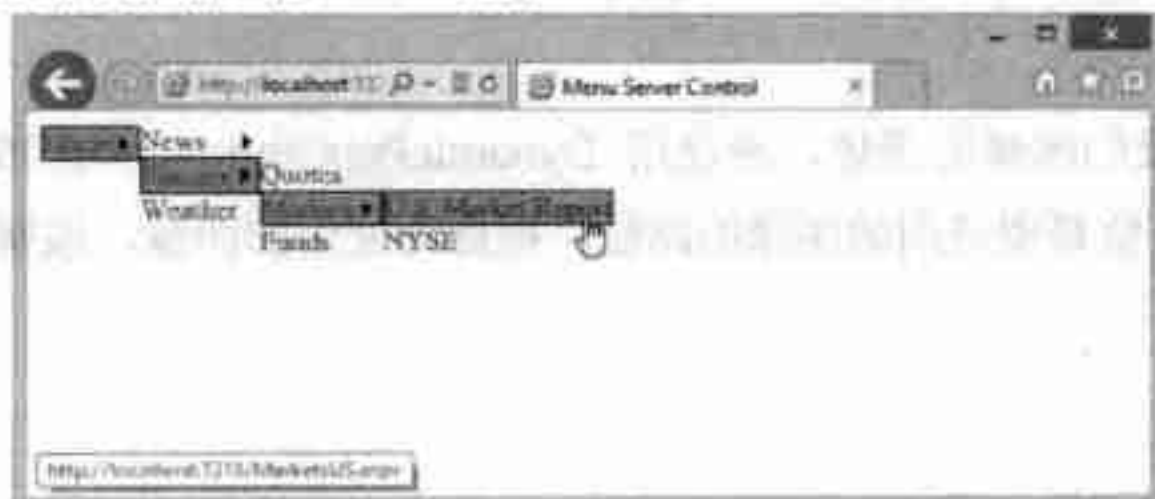


图 17-22

4. 修改菜单项的布局

在默认情况下, 动态的菜单项从左到右显示。也就是说, 菜单中的项在展开时会以垂直方式显示。实际上, 可以控制这种行为, 还可以使用另一种方式。

另一种方式是把第一层菜单项直接显示在第一个静态项的下面(即水平方向)。使用 Menu 控件的 Orientation 属性可以改变这种行为, 如程序清单 17-20 所示。

程序清单 17-20 使菜单项水平显示

```
<asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1"
    Orientation="Horizontal">
</asp:Menu>
```

运行这段代码, 生成的结果如图 17-23 所示。

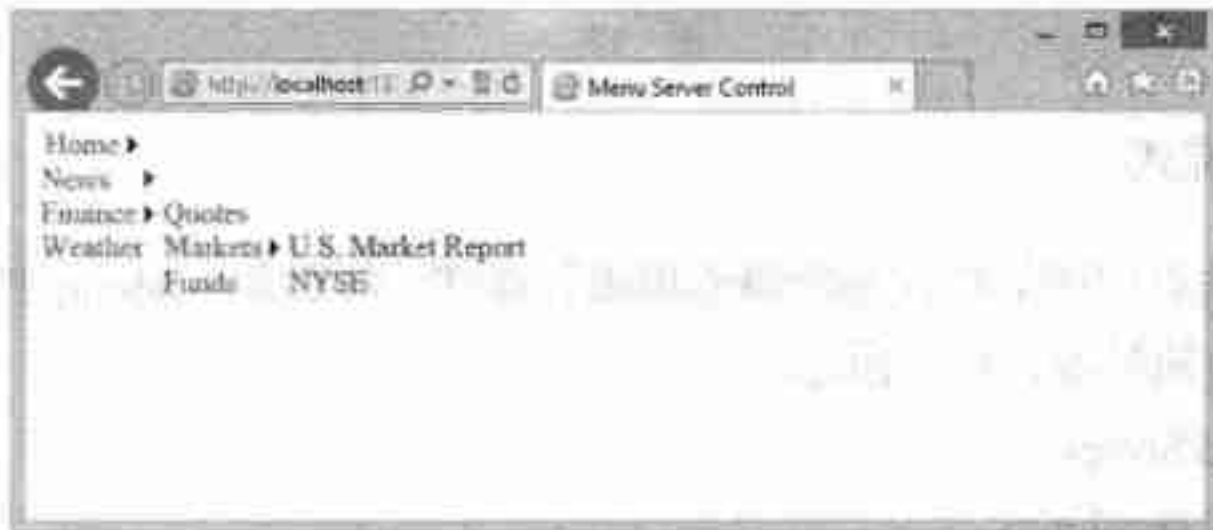


图 17-23

Orientation 属性的值只能是 Horizontal 或 Vertical, 其默认值是 Vertical。

5. 改变弹出符号

在默认情况下, 生成的菜单项无论是静态项还是动态项, 都使用箭头表示弹出符号, 如图 17-24 所示。



图 17-24

不一定要使用这个箭头符号, 实际上, 可以方便地改用一幅图像。程序清单 17-21 说明了如何完成这项任务。

程序清单 17-21 使用定制图像

```
<asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1"
    Orientation="Horizontal" DynamicPopOutImageUrl="Images/myArrow.png"
    StaticPopOutImageUrl="Images/myArrow.png">
</asp:Menu>
```

要把弹出符号改为我们选择的图像, 应使用 DynamicPopOutImageUrl 或 StaticPopOutImageUrl 属性。这些属性的 String 值是要使用的图像的路径。根据所使用的图像, 该属性会生成如图 17-25 所示的结果。



图 17-25

6. 使用图像分隔菜单项

Menu 控件的另一样式选项可以给菜单项添加一幅分隔符图像。根据分隔符图像所处的位置, 可以使用 `StaticBottomSeparatorImageUrl`、`StaticTopSeparatorImageUrl`、`DynamicBottomSeparatorImageUrl` 和 `DynamicTopSeparatorImageUrl` 属性。

例如, 如果要把分隔符图像只放在动态菜单项的下面, 应使用 `DynamicBottomSeparatorImageUrl` 属性, 如程序清单 17-22 所示。

程序清单 17-22 对动态项应用分隔符图像

```
<asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1"
  DynamicBottomSeparatorImageUrl="Images/myDivider.png">
</asp:Menu>
```

Menu 控件中所有定义分隔符图像的属性都带有一个 `String` 值, 它指向图像的位置。程序清单 17-22 的结果如图 17-26 所示。

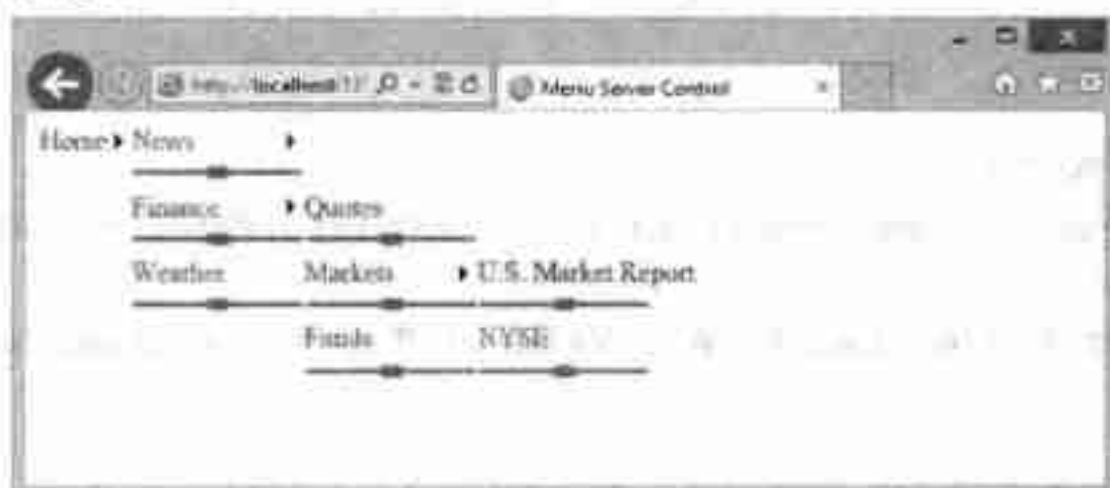


图 17-26

17.4.2 Menu 事件

Menu 控件包含如下事件:

- `DataBinding`
- `DataBound`
- `Disposed`
- `Init`
- `Load`
- `MenuItemClick`
- `MenuItemDataBound`
- `PreRender`

- Unload

需要注意的一个事件是 MenuItemClick, 这个事件如程序清单 17-23 所示, 它在终端用户单击一个菜单项时执行一些操作。

程序清单 17-23 使用 MenuItemClick 事件

```
protected void Menu1_MenuItemClick(object sender, MenuEventArgs e)
{
    // Code for event here
}
```

这个委托使用了 MenuEventArgs 数据类, 允许访问所选菜单项的文本和值。

17.4.3 把 Menu 控件绑定到 XML 文件

与 TreeView 控件一样, 也可以把 Menu 控件绑定到 ASP.NET 4.5 提供的其他数据源控件的项。大多数开发人员都喜欢使用 Menu 控件, 让终端用户导航到 URL 目的地, 还可以使用 Menu 控件让用户作出选择。

例如, 采用前面创建的 XML 文件 Hardware.xml, 该文件在本章前面的程序清单 17-8 中与 TreeView 控件一起使用。而在这个例子中, Menu 控件与 XmlDataSource 控件一起使用。当终端用户从菜单中选择一项时, 就使用选中的项填充页面上的列表框。该控件的代码如程序清单 17-24 所示。

程序清单 17-24 将 Menu 控件绑定到 XML 文件上

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Menu1_MenuItemClick(object sender, MenuEventArgs e)
    {
        Listbox1.Items.Add(e.Item.Parent.Value + " : " + e.Item.Value);
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Menu Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Menu ID="Menu1" runat="server" DataSourceID="XmlDataSource1"
            OnMenuItemClick="Menu1_MenuItemClick">
            <DataBindings>
                <asp:MenuItemBinding DataMember="Item"
                    TextField="Category"></asp:MenuItemBinding>
                <asp:MenuItemBinding DataMember="Option"
                    TextField="Choice"></asp:MenuItemBinding>
            </DataBindings>
        </asp:Menu>
        <p>
            <asp:ListBox ID="Listbox1" runat="server">
```

```

</asp:ListBox></p>
<asp:xmldatasource ID="XmlDataSource1" runat="server"
  datafile="Hardware.xml" />
</form>
</body>
</html>

```

在这个例子中，没有像前面的 TreeView 控件一样使用<asp:TreeNodeBinding>元素，Menu 控件使用<asp:MenuItemBinding>元素来连接 XML 文件 Hardware.xml 中列出的项。另外，Menu 控件的根元素<asp:Menu>现在包含 OnMenuItemClick 属性，它指向事件委托 Menu1_MenuItemClick。

Menu1_MenuItemClick 委托包含数据类 MenuEventArgs，它能获得选中的子元素和父元素的值。在这个例子中，使用这两个元素的值填充 ListBox 控件，如图 17-27 所示。



图 17-27

17.5 SiteMap 数据提供程序

ASP.NET 4.5 提供了一系列 DataSource 控件形式的数据提供程序。本章前面介绍的其中一个 DataSource 控件是 SiteMapDataSource，这个 DataSource 控件可处理站点地图和绑定到其上的控件。

一些控件不需要 SiteMapDataSource 控件来绑定到应用程序的站点地图(站点地图一般存储在 Web.sitemap 文件中)。本章的前面介绍了使用 SiteMapPath 控件时不需要 SiteMapDataSource 控件。SiteMapPath 控件可以直接使用 Web.sitemap 文件，不需要这个数据提供程序。

但是，一些导航控件，如 TreeView 控件和 DropDownList 控件，就需要 SiteMapDataSource 控件来提取站点导航信息。

SiteMapDataSource 控件的使用很简单，如本章的例子所示。SiteMapDataSource 控件的最简单形式如下所示：

```
<asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
```

SiteMapDataSource 控件的这种形式只能获得树型层次结构的信息(如前面所示)。注意，有许多属性都可以改变数据在绑定到数据输出的控件中的显示方式。

17.5.1 ShowStartingNode 属性

ShowStartingNode 属性确定是否使用所提取的节点对象集合获得.sitemap 文件的根节点。这个属

性是布尔值，并且默认设置为 True。如果使用程序清单 17-1 中的 Web.sitemap 文件，就可以构建如程序清单 17-25 中所示的 SiteMapDataSource 控件，从集合中删除根节点。

程序清单 17-25 从获得的节点集合中删除根节点

```
<%@ Page Language="C#" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Menu Server Control</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server"
      ShowStartingNode="False" />
    <asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1">
    </asp:Menu>
  </form>
</body>
</html>
```

这段代码生成的菜单如图 17-28 所示。



图 17-28

在这个屏幕截图中，根节点已被删除，所显示的菜单从根节点的子节点开始。

17.5.2 StartFromCurrentNode 属性

StartFromCurrentNode 属性让 SiteMap 数据提供程序只提取从页面的当前节点开始的节点集合。该属性默认设置为 False，表示 SiteMap 数据提供程序总是提取所有的节点(从根节点开始到当前节点)。

对于这个例子，使用程序清单 17-1 中的.sitemap 文件创建页面 Markets.aspx。这个页面在节点集合的层次结构中是 Finance 节点的子节点，它还有两个子节点 U.S. Market Report 和 NYSE。把 StartFromCurrentNode 属性设置为 True 的示例如程序清单 17-26 所示。

程序清单 17-26 使用 StartFromCurrentNode 属性的 Markets.aspx 页面

```
<%@ Page Language="C#" %>

<html xmlns="http://www.w3.org/1999/xhtml">
```

```

<head runat="server">
    <title>Menu Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server"
            StartFromCurrentNode="True" />
        <asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1">
        </asp:Menu>
    </form>
</body>
</html>

```

添加这个属性后的结果如图 17-29 所示。

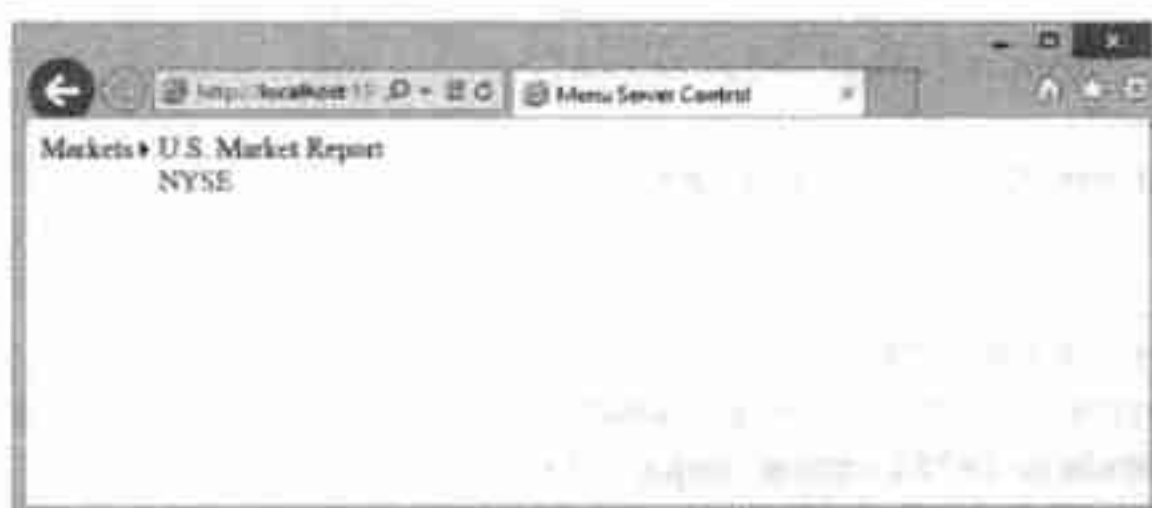


图 17-29

17.5.3 StartingNodeOffset 属性

StartingNodeOffset 属性是整数值，表示层次结构集合的起点。这个属性默认设置为 0，表示通过 SiteMapDataSource 控件获取的节点集合从根节点开始。其他值会提供与根节点的偏移量，并把该偏移量指定的节点作为新的起点。在程序清单 17-1 的例子中，可以看出集合是从 Default.aspx 的 Home 页面开始的，Default.aspx 在本章的许多例子中都会用到。

如果把这个属性的值设置为 1，集合的起点就与默认的起点(Default.aspx 的 Home 页面)之间相隔 1 个位置。例如，如果使用 SiteMapDataSource 控件的页面是 MarketsUS.aspx，节点集合就从 Finance 页面(Finance.aspx)开始。

```

Home    Offset 0
News    Offset 1
  U.S.   Offset 2
  World  Offset 2
  Technology Offset 2
  Sports Offset 2
Finance Offset 1
  Quotes Offset 2
  Markets Offset 2
    U.S. Market Report Offset 3
    NYSE Offset 3
  Funds Offset 2
  Weather Offset 1

```

在上述层次结构中，可以看到每个节点与根节点的偏移量。因此，如果把 StartingNodeOffset 属性设置为 1，并且浏览到 U.S. Market Report 页面，节点集合就从 Finance 页面(Finance.aspx)开始，

而根节点的其他子节点(News 和 Weather)不会出现在节点集合中, 因为在这个层次结构中, Finance.aspx 页面是所请求页面的直接路径。

17.5.4 StartingNodeUrl 属性

StartingNodeUrl 属性可以指定以.sitemap 文件中找到的页面作为节点集合的起点。该属性的值默认为空, 但当设置为 Finance.aspx 时, 集合就从 Finance 页面开始, 并把这个页面作为节点集合的根节点。程序清单 17-27 显示了使用 StartingNodeUrl 属性的一个例子。

程序清单 17-27 使用 StartingNodeUrl 属性

```
<%@ Page Language="C#" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Menu Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server"
            StartingNodeUrl="Finance.aspx" />
        <asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1">
        </asp:Menu>
    </form>
</body>
</html>
```

当在程序运行过程中遇到 StartingNodeUrl 属性值时, 就把其值与 Web.sitemap 文件中的 url 属性比较。如果它们匹配, 就把匹配的页面作为通过 SiteMapDataSource 控件获得的节点集合的根节点。

17.6 SiteMap API

SiteMap 类是站点导航结构在内存中的表示。这个类非常适合于编程处理站点的层次结构。SiteMap 类有几个属性可以使导航结构的处理非常简单, 这些属性如表 17-5 所示。

表 17-5

属 性	说 明
CurrentNode	为当前页面提取 SiteMapNode 对象
RootNode	从站点导航结构的根节点及其他节点中提取 SiteMapNode 对象
Provider	为当前的站点地图提取默认的 SiteMapProvider 对象
Providers	提取名为 SiteMapProvider 的对象集合

程序清单 17-28 是一个使用 SiteMap 对象的例子, 它演示了如何使用 Markets.aspx 页面中的 CurrentNode 对象。

程序清单 17-28 使用 CurrentNode 对象

```

<%@ Page Language="C#" %>

<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e)
    {
        Label1.Text = SiteMap.CurrentNode.Description + "<br>" +
            SiteMap.CurrentNode.HasChildNodes + "<br>" +
            SiteMap.CurrentNode.NextSibling.ToString() + "<br>" +
            SiteMap.CurrentNode.ParentNode.ToString() + "<br>" +
            SiteMap.CurrentNode.PreviousSibling.ToString() + "<br>" +
            SiteMap.CurrentNode.RootNode.ToString() + "<br>" +
            SiteMap.CurrentNode.Title + "<br>" +
            SiteMap.CurrentNode.Url;
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>SiteMapDataSource</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Label ID="Label1" runat="server"></asp:Label>
    </form>
</body>
</html>

```

这段代码使用 SiteMap 类和 CurrentNode 对象来处理与当前页面相关的一些信息。运行这个页面，结果将如下所示：

```

The Latest Market Information
True
Funds
Finance
Quotes
Home
Markets
/SiteNavigation/Markets.aspx

```

使用 CurrentNode 属性可以为 SiteMapPath 控件创建样式，如程序清单 17-29 所示。

程序清单 17-29 使用 CurrentNode 属性创建定制的导航结构

```

<%@ Page Language="C#" %>

<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e)
    {
        Hyperlink1.Text = SiteMap.CurrentNode.ParentNode.ToString();
        Hyperlink1.NavigateUrl = SiteMap.CurrentNode.ParentNode.Url;

        Hyperlink2.Text = SiteMap.CurrentNode.PreviousSibling.ToString();
        Hyperlink2.NavigateUrl = SiteMap.CurrentNode.PreviousSibling.Url;
    }

```

```

        Hyperlink3.Text = SiteMap.CurrentNode.NextSibling.ToString();
        Hyperlink3.NavigateUrl = SiteMap.CurrentNode.NextSibling.Url;
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>SiteMapDataSource</title>
</head>
<body>
    <form id="form1" runat="server">
        Move Up:
        <asp:Hyperlink ID="Hyperlink1" runat="server"></asp:Hyperlink><br />
        &lt;-- <asp:Hyperlink ID="Hyperlink2" runat="server"></asp:Hyperlink> |
        <asp:Hyperlink ID="Hyperlink3" runat="server"></asp:Hyperlink> --&gt;
    </form>
</body>
</html>

```

运行页面，可以得到定制的导航结构，如图 17-30 所示。



图 17-30

17.7 URL 映射

随着应用程序不断扩展，Web 页面使用的 URL 会变得相当复杂。有时可以基于通过 URL 提供的查询字符串来显示内容已改变的 Web 页面，例如：

```
http://www.asp.net/forums/view.aspx?forumid=12&categoryid=6
```

在其他情况下，Web 页面位于文件夹层次结构中较深的层次，终端用户以后在浏览器上打开该页面时，必须输入或记住很繁琐的 URL。有时，我们还希望一组页面的外观看起来是同一页面或目的地。

此时，可以使用 ASP.NET 功能——URL 映射，将复杂的 URL 映射为简单的 URL。为此，要使用 `<urlMappings>` 元素在 `web.config` 文件中应用的设置，如程序清单 17-30 所示。

程序清单 17-30 使用 `<urlMappings>` 元素映射 URL

```
<configuration>
```

```
    <system.web>
```

```
        <urlMappings>
```



```

    <add url="~/Content.aspx" mappedUrl="~/SystemNews.aspx?categoryid=5" />
  </urlMappings>

</system.web>

</configuration>

```

在这个例子中提供了虚假的 URL——Content.aspx，它被映射为比较复杂的 URL——SystemNews.aspx?categoryid=5。建立了这个映射后，当终端用户输入简单的 URL Content.aspx 时，应用程序就知道应调用较复杂的 URL，即 SystemNews.aspx?categoryid=5 页面。完成这个复杂 URL 的调用不需要在浏览器中修改 URL。甚至在页面加载完毕后，浏览器仍然会把 Content.aspx 页面显示为目的地，从而在某种意义上欺骗终端用户。



在这种情况下，终端用户总是会被路由到 SystemNews.aspx?categoryid=5，即使 Content.aspx 页面存在也是如此。因此，一定要映射到应用程序不包含的页面。

17.8 站点地图的本地化

改进的资源文件(.resx)是本地化 ASP.NET 应用程序的一种好方法。本书的第 32 章介绍了这种使用 ASP.NET 对 Web 应用程序进行本地化的方法。这里主要介绍如何把本地化功能应用于应用程序的页面，但是没有进一步演示如何将本地化功能应用于诸如 Web.sitemap 文件这样的具体项。

17.8.1 为本地化构建 Web.sitemap 文件

可以把本地化指令应用于 ASP.NET Web 应用程序的页面上，也可以使用相同的架构在 Web.sitemap 文件中完成本地化任务。为了说明这一点，程序清单 17-31 构建了一个 Web.sitemap 文件，它有点类似于程序清单 17-1 中的 Web.sitemap 文件，但要简单得多。

程序清单 17-31 为本地化创建一个基本的.sitemap 文件

```

<?xml version="1.0" encoding="utf-8" ?>

<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0"
  enableLocalization="true">
  <siteMapNode url="Default.aspx" resourceKey="Home">
    <siteMapNode url="News.aspx" resourceKey="News">
      <siteMapNode url="News.aspx?cat=us" resourceKey="NewsUS" />
      <siteMapNode url="News.aspx?cat=world" resourceKey="NewsWorld" />
      <siteMapNode url="News.aspx?cat=tech" resourceKey="NewsTech" />
      <siteMapNode url="News.aspx?cat=sport" resourceKey="NewsSport" />
    </siteMapNode>
  </siteMapNode>
</siteMap>

```

程序清单 17-31 是一个相当简单的 Web.sitemap 文件。为了在 Web.sitemap 文件中支持本地化功能，必须使用<siteMap>元素中的 enableLocalization 特性，把它设置为 true，以启用这个功能。之后，

就可以使用<siteMapNode>元素定义每个导航节点。但在这个例子中,由于要在各个.resx文件中定义这些导航部分的内容(尤其是title和description特性),因此不需要在这个文件中重复定义这些项。也就是说,这个例子只需要定义url特性。但要注意,还可以通过.resx文件定义这个特性,从而根据终端用户定义的区域性设置,把它们定向到不同的页面。

下一个要注意的特性是<siteMapNode>元素中使用的resourceKey,它在执行的各个.resx文件中使用和定义。以下面的<siteMapNode>元素为例:

```
<siteMapNode url="News.aspx" resourceKey="News">
    . . .
</siteMapNode>
```

在这个例子中,resourceKey的值(以及.resx文件中使用的键)是News。这表示可以使用下面的语法,在.resx文件中定义title和description特性的值:

```
News.Title
News.Description
```

有了Web.sitemap文件,下一步就是对web.config文件进行一些小的修改。

17.8.2 修改web.config文件

有了Web.sitemap文件,下一步就是给web.config文件添加一些内容。为了让Web应用程序自动检测访问各个页面的用户所使用的区域性设置,需要在@Page指令中设置culture和uiCulture特性,或者在web.config文件的<globalization>元素中设置这些特性,以支持自动检测。

在处理导航和Web.sitemap文件时,最好在web.config文件中进行这些修改,使它们在应用程序的每个页面上自动发挥作用。将修改添加到web.config文件可以简化操作,因为不需要给每个页面添加这些修改。

要进行这些修改,可以打开web.config文件,添加<globalization>元素,如程序清单17-32所示。

程序清单 17-32 给web.config文件添加区域性检测功能

```
<configuration>
  <system.web>

    <globalization culture="auto" uiCulture="auto" />

  </system.web>
</configuration>
```

为了支持自动检测功能,只需把culture和uiCulture特性设置为auto即可。还可以把值定义为auto:en-US,这表示支持自动区域性检测功能,但如果所定义的区域性在各个资源文件中未找到,就把en-US(美国英语)用作默认区域性设置。但是,因为我们要定义一组Web.sitemap默认值,所以不需要这样做。

接下来需要创建程序集资源文件,以定义Web.sitemap文件使用的值。

17.8.3 创建程序集资源(.resx)文件

要创建一组用于Web.sitemap文件的程序集资源文件,应在项目中创建文件夹App_GlobalResources。如果使用的是Visual Studio 2012或Visual Studio Express 2012 for Web,就可以右击项目并选择Add

ASP.NET Folder | App_GlobalResources 命令来添加这个文件夹。

添加文件夹后, 下一步是给这个文件夹添加两个程序集资源文件, 第一个文件命名为 Web.sitemap.resx, 第二个文件命名为 Web.sitemap.de.resx。添加这两个文件的目的是, 让 Web.sitemap 文件包含一组在 Web.sitemap.resx 文件中定义的、用于 Web.sitemap 文件的默认值, 以及这些值的德语版本, 这些版本包含在 Web.sitemap.de.resx 文件中。

如果文件名中有 de 值, 就表示该文件由把首选语言设置为 de-De(或其变体, 例如 de-AT、de-CH) 的用户使用。这些设置的其他值如表 17-6 所示。

表 17-6

.resx 文件	对应的区域性设置
Web.sitemap.resx	当终端用户的区域性设置不能通过另一个.resx 文件识别时, 使用默认值
Web.sitemap.en.resx	用于所有英语用户的资源文件
Web.sitemap.en-gb.resx	用于所有大不列颠英语用户的资源文件
Web.sitemap.fr-ca.resx	用于所有加拿大法语用户的资源文件
Web.sitemap.ru.resx	用于所有俄语用户的资源文件

有了 Web.sitemap.resx 文件和 Web.sitemap.de.resx 文件后, 下一步是给这些文件填充值。为此, 可直接使用前面在 Web.sitemap 文件中定义的键。图 17-31 显示了结果。

Name	Value
Home.Description	Startseite
Home.Title	Start
News.Description	Die aktuellsten Neuigkeiten
News.Title	Neuigkeiten
NewsUS.Description	Neuigkeiten aus den USA
NewsUS.Title	USA-News
NewsWorld.Description	Neuigkeiten aus dem Rest der Welt
NewsWorld.Title	Welt-News
NewsTech.Description	Neuigkeiten aus dem Bereich Technologie
NewsTech.Title	Technologie
NewsSport.Description	Neuigkeiten aus der Welt des Sports
NewsSport.Title	Sport

Name	Value
Home.Description	Homepage
Home.Title	Home
News.Description	Get the latest news
News.Title	Our News
NewsUS.Description	News from the US of A
NewsUS.Title	U.S. News
NewsWorld.Description	News from around the world
NewsWorld.Title	World
NewsTech.Description	Technology news
NewsTech.Title	Technology
NewsSport.Description	Sport news
NewsSport.Title	Sports

图 17-31

IDE 认为这些不是有效的标识符, 但应用程序仍然可以使用这个模型。有了这些文件后, 就可以测试本地化的工作情况。

17.8.4 测试结果

在应用程序中创建一个页面, 并在该页面上放置一个 TreeView 服务器控件。除了 TreeView 控件之外, 还要放置一个 SiteMapDataSource 控件, 以处理前面创建的 Web.sitemap 文件。设置 TreeView 控件的属性 DataSourceID="SiteMapDataSource1", 如本章前面所示, 将这两个控件关联起来。

如果在 Internet Explorer 中把语言设置为 en-us(美国英语), 结果就如图 17-32 所示。



图 17-32

在浏览器中打开这个页面, 就会检查请求的区域性设置。由于本例定义的唯一细化设置是用于使用 de(德语)区域性设置的用户, 因此使用默认的 Web.sitemap.resx。于是, 使用 Web.sitemap.resx

文件填充 TreeView 控件的值,如图 17-32 所示。如果请求者把区域性设置为 de,就会得到完全不同的结果。

要测试这种情况,可以修改 IE 中使用的首选语言。首先,在 IE 中选择 Tools | Internet Options 命令,在第一个选项卡(General)中,单击对话框底部的 Languages 按钮,打开 Language Preference 对话框。单击 Add 按钮,在选项列表中添加 German 语言设置。最后,使用 Move Up 按钮把 German 选项移动到列表的最上面。结果如图 17-33 所示。如果使用的是 Firefox 浏览器,就导航到 Tools | Options | Content,单击 Languages 部分的 Choose 按钮,修改语言设置。



图 17-33

有了这个设置后,运行页面,TreeView 控件就会显示如图 17-34 所示的结果。

现在当请求页面时,区域性就被设置为 de,对应的是 Web.sitemap.de.resx 文件,而不是默认的 Web.sitemap.resx 文件。



图 17-34

17.9 安全补偿

如果完成了本章前面的示例,你就会注意到,我们还没有探讨<siteMapNode>标记的 roles 特性。这个特性很强大,可以为包含在导航系统中的项提供授权模型。也就是说,可以只向有特定权限的用户显示导航项。用于这个操作的常见术语是“安全补偿(security trimming)”。本节就将介绍如何把安全补偿功能应用于在 ASP.NET 4.5 中建立的应用程序。

这个功能是两个 ASP.NET 4.5 系统在站点导航系统中彼此交互的一个优秀示例。只有激活 ASP.NET 4.5 角色管理系统,安全补偿功能才会起作用。角色管理系统详见第 19 章,一定要阅读这一章,因为本节不会详细探讨这个系统。

为了说明安全补偿功能在 ASP.NET 应用程序中的作用,本节将介绍如何限定只允许包含在指定应用程序角色中的用户访问应用程序的管理系统。

17.9.1 启动管理员的角色管理功能

要应用安全补偿,首先应建立应用程序以处理角色。这个过程相当简单。为此,一种简便的方法是为应用程序打开 ASP.NET Web Site Administration Tool,在这个基于 Web 的工具中直接激活角

色管理功能。要打开这个管理工具，可以在 Visual Studio 的 Solution Explorer 菜单中单击 ASP.NET Configuration 按钮，这个按钮的图标是一个锤子和一个球体。

启动 ASP.NET Web Site Administration Tool 后，选择 Security 选项卡，这会显示一个页面，在该页面上可以为应用程序启动成员资格和角色管理系统。

首先，激活并启动角色管理系统，再激活成员资格系统。成员资格系统详见第 19 章。打开成员资格系统后，在应用程序中建立一些用户。如果一个用户登录到应用程序中，就给他赋予一个指定的角色。角色赋予操作会改变站点导航系统的显示。

ASP.NET Web Site Administration Tool 的 Security 选项卡如图 17-35 所示。

在这个页面上，选择 Enable roles 链接，就可以启用角色管理系统。之后，页面会通知我们，该系统没有角色。要为站点导航系统创建需要的角色，可选择 Create or Manage roles 链接。这会显示一个新页面，在其中可以创建管理员角色。对于这个例子，将创建的角色命名为 Admin。

添加了 Admin 角色后，单击 Back 按钮，选择应用程序使用的验证类型。应确保选择了 From the internet 选项，这样就可以在系统中创建用户。在默认情况下，创建的用户存储在 ASP.NET 为应用程序创建的 Microsoft SQL Server Express Edition 文件中，所以必须安装数据库服务器。也可以在 web.config 文件中提供定制的连接字符串，把 ASP.NET 配置为使用另一个数据库服务器或类型。

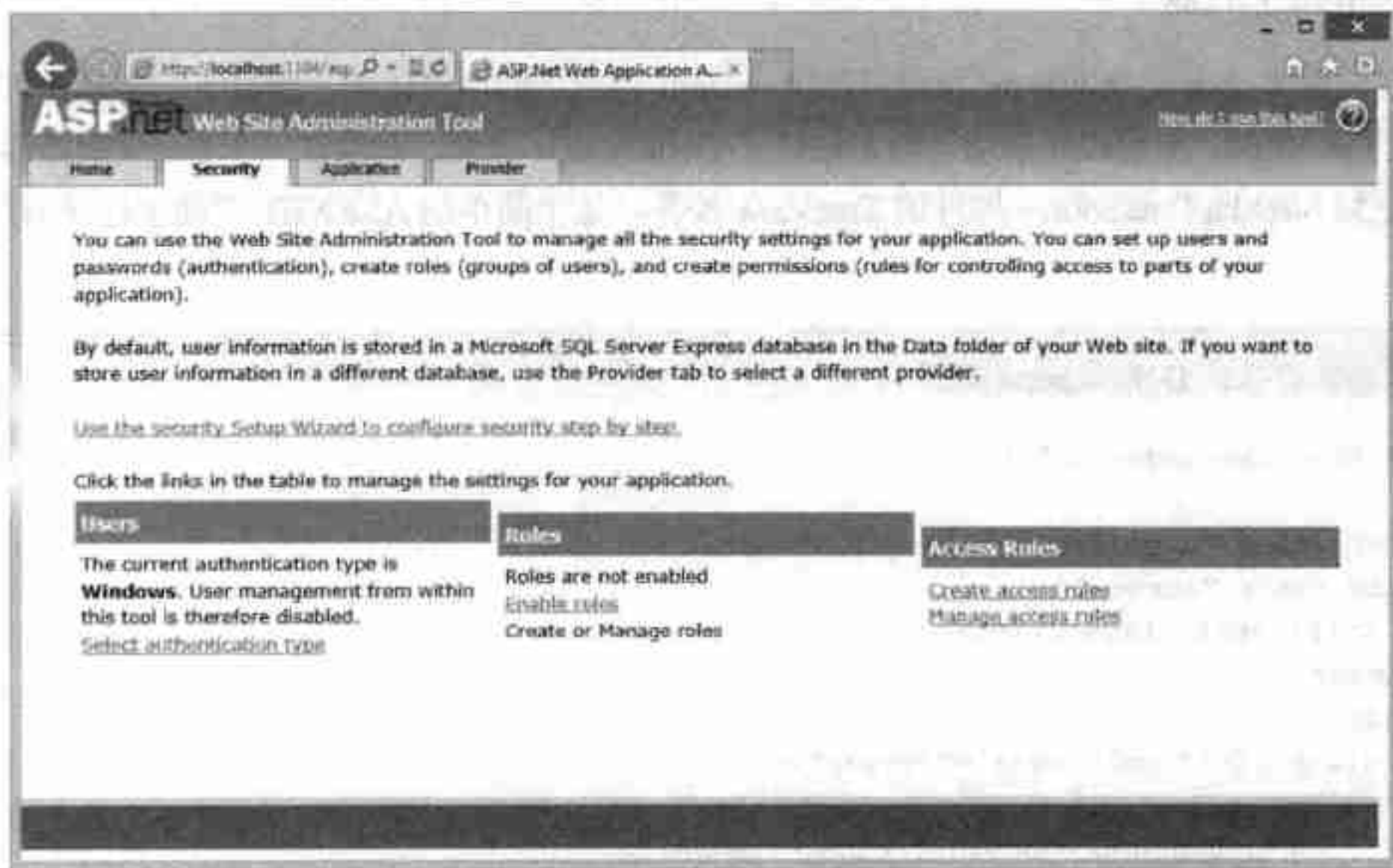


图 17-35

选择了验证类型后，就可以创建新用户，在创建用户的屏幕上选择 Admin 角色，就可以把该用户放在 Admin 角色中。

之后，就可以检查 web.config 文件中的设置是否正确，如程序清单 17-33 所示。

程序清单 17-33 在 web.config 文件中启用角色管理系统

```
<configuration>
  <system.web>
    <authentication mode="Forms" />
    <roleManager enabled="true" />
  </system.web>
</configuration>
```

```
</system.web>  
</configuration>
```

17.9.2 建立管理员的配置部分

下一步是为管理员建立一个页面。这个例子把该页面命名为 `AdminOnly.aspx`，它只包含一个简单的字符串值，表示欢迎管理员进入本页面。只有包含在 `Admin` 角色中的用户才能锁定该页面。为此，要在 `web.config` 文件中设置相应的值，如程序清单 17-34 所示。

程序清单 17-34 在 web.config 文件中锁定 AdminOnly.aspx 页面

```
<configuration>  
  <location path="AdminOnly.aspx">  
    <system.web>  
      <authorization>  
        <allow roles="Admin" />  
        <deny users="*" />  
      </authorization>  
    </system.web>  
  </location>  
</configuration>
```

现在，因为 `AdminOnly.aspx` 页面只能由 `Admin` 角色中的用户访问，所以下一步是允许用户登录应用程序。为此，应用程序的演示版本创建了 `Default.aspx` 页面，它包含一个 `Login` 服务器控件和一个绑定到 `SiteMapDataSource` 控件的 `TreeView` 控件。这个简单的 ASP.NET 页面如程序清单 17-35 所示。

程序清单 17-35 Default.aspx 页面

```
<%@ Page Language="C#" %>  
  
<html xmlns="http://www.w3.org/1999/xhtml">  
  <head runat="server">  
    <title>Main Page</title>  
  </head>  
  <body>  
    <form id="form1" runat="server">  
      <div>  
        <asp:Login ID="Login1" runat="server">  
        </asp:Login>  
        <br />  
        <asp:TreeView ID="TreeView1" runat="server"  
          DataSourceID="SiteMapDataSource1" ShowLines="True">  
        </asp:TreeView>  
        <br />  
        <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />  
      </div>  
    </form>  
  </body>  
</html>
```

有了 `Default.aspx` 页面后，就要对程序清单 17-1 中的 `Web.sitemap` 文件进行另一处修改。在这

个例子中需要添加一个<siteMapNode>元素，以使用新的 AdminOnly.aspx 页面。该节点如下所示：

```
<siteMapNode title="Administration" description="The Administrators page"
url="AdminOnly.aspx" roles="Admin" />
```

因为 Login 控件使用验证功能，所以需要引用 jQuery，或者禁用隐含验证功能(参见第 6 章)，后者可以用下面的 web.config 设置实现：

```
<appSettings>
  <add key="ValidationSettings:UnobtrusiveValidationMode" value="None" />
</appSettings>
```

将所有项添加到应用程序后，就要为站点导航系统启用安全补偿功能。

17.9.3 启用安全补偿功能

在默认情况下，安全补偿功能是禁用的。即使在 web.config 文件的<siteMapNode>元素中应用了 roles 特性值，安全补偿功能也是不能工作的。要启用安全补偿功能，必须调整站点导航系统的提供程序声明。

要对 XmlSiteMapProvider 进行必要的修改，应在配置链的较高层次进行这些修改，例如修改 machine.config 或默认的 web.config 文件；也可以在较低层次进行修改，例如修改应用程序的 web.config 文件。在本例中修改 web.config 文件。

为了在 web.config 文件中修改 XmlSiteMapProvider，首先要清除已声明的实例。之后，重新声明 XmlSiteMapProvider 的一个新实例，但这次要启用安全补偿功能，如程序清单 17-36 所示。

程序清单 17-36 启用提供程序中的安全补偿功能

```
<configuration>
  <system.web>
    <siteMap>
      <providers>
        <clear />
        <add siteMapFile="web.sitemap" name="AspNetXmlSiteMapProvider"
          type="System.Web.XmlSiteMapProvider, System.Web, Version=4.0.0.0,
            Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
            securityTrimmingEnabled="true" />
      </providers>
    </siteMap>
  </system.web>
</configuration>
```

从本例中可以看出，定义了一个新的 XmlSiteMapProvider，其 securityTrimmingEnabled 特性设置为 true(参见代码中以粗体显示的部分)。启用了安全补偿功能后，就可以在站点导航系统中使用<siteMapNode>元素中的 roles 特性了。

下面进行测试。运行 Default.aspx 页面。首先显示的页面不包含到页面的管理部分的链接，如图 17-36 所示。

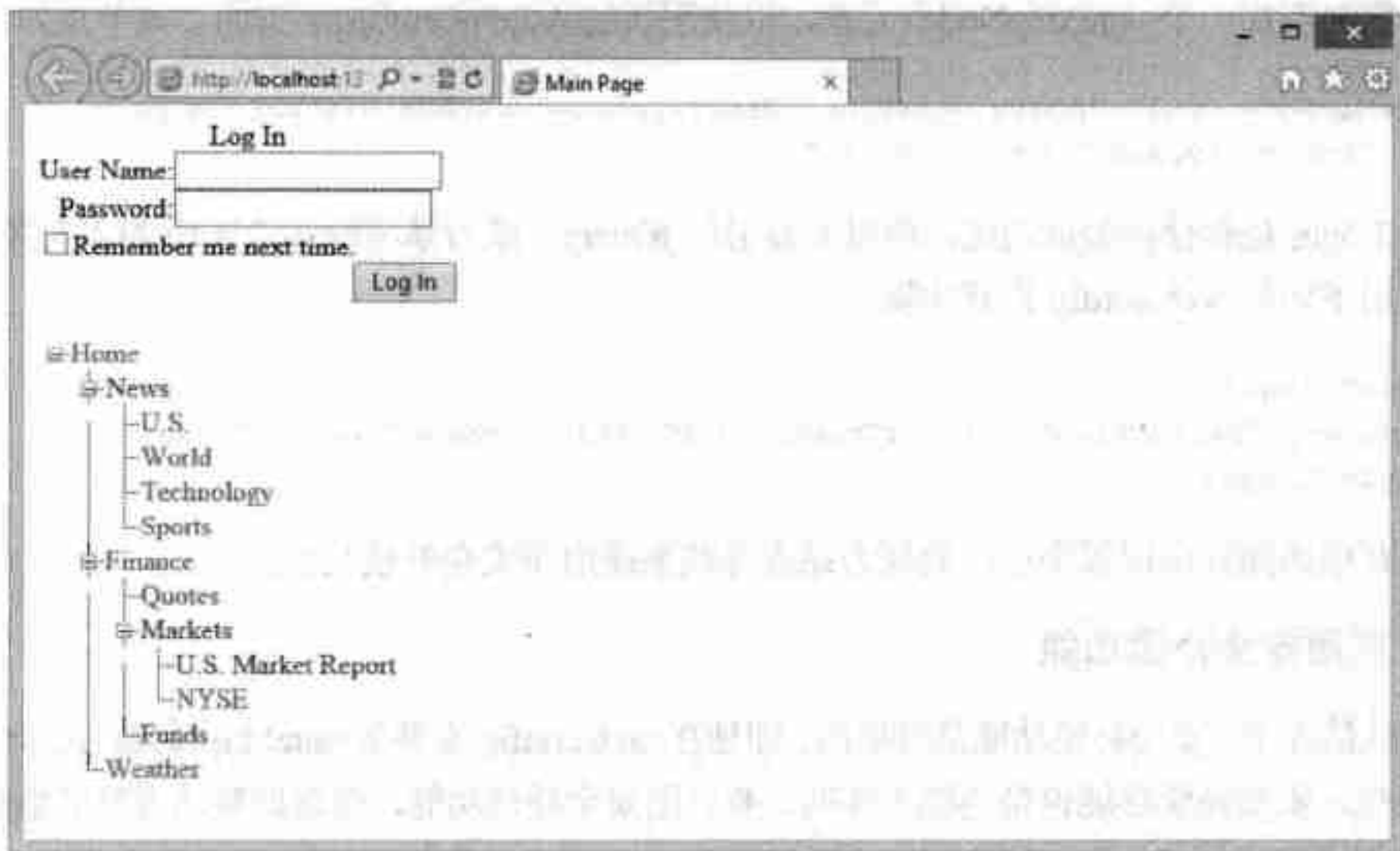


图 17-36

在图 17-36 中，TreeView 控件中没有显示 Administration 链接。现在使用 Admin 角色中包含的用户身份登录到应用程序，就会看到站点导航系统已改变，反映了用户的角色，如图 17-37 所示。

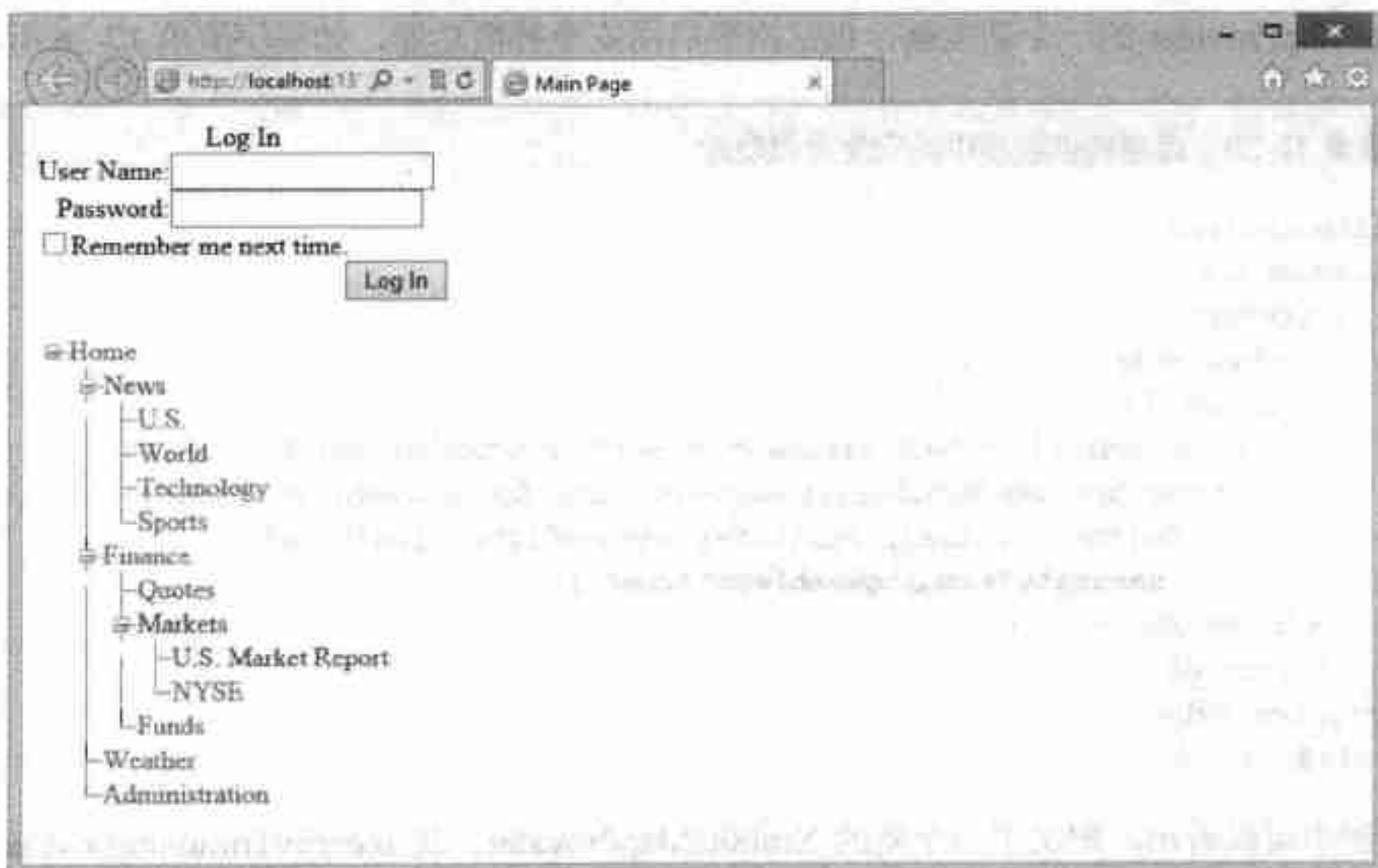


图 17-37

安全补偿功能非常适用于根据用户在角色管理系统中的角色，自动改变显示出来的站点导航。

17.10 嵌套站点地图文件

不需要把所有的站点导航信息都放在一个 Web.sitemap 文件中。实际上，可以把这些信息分别放

在多个.sitemap 文件中,再把它们合并到一个.sitemap 文件中,这称为嵌套.sitemap 文件。

例如,假定使用程序清单 17-1 中的站点地图文件,我们要在 Entertainment 区域中添加大量的站点导航信息。此时就可以把所有的新信息放在当前的 Web.sitemap 文件中,或者将所有的 Entertainment 链接放在另一个站点地图文件中,在主站点地图文件中引用该文件即可。

嵌套站点地图文件是很简单的工作。只需创建一个新的.sitemap 文件,命名为 Entertainment.sitemap,然后把这个文件放在应用程序中。程序清单 17-37 显示了该文件。

程序清单 17-37 Entertainment.sitemap 文件

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode url="Entertainment.aspx" title="Entertainment"
    description="The Entertainment Page">
    <siteMapNode url="Movies.aspx" title="Movies"
      description="The Latest in Movies" />
    <siteMapNode url="Fashion.aspx" title="Fashion"
      description="The Latest in Fashion" />
  </siteMapNode>
</siteMap>
```

可以把 Entertainment.sitemap 文件放在主站点地图文件所在的根目录中。现在,以程序清单 17-1 中的站点地图文件为基础,在列表的底部进行如程序清单 17-38 所示的修改。

程序清单 17-38 给 Web.sitemap 文件添加新内容

```
<siteMapNode siteMapFile="Entertainment.sitemap" />
```

在此没有使用标准的 url、title 和 description 特性,而是只使用 siteMapFile 特性指向在这个主站点地图文件中包含的另一个站点地图文件。运行这个页面,结果如图 17-38 所示。

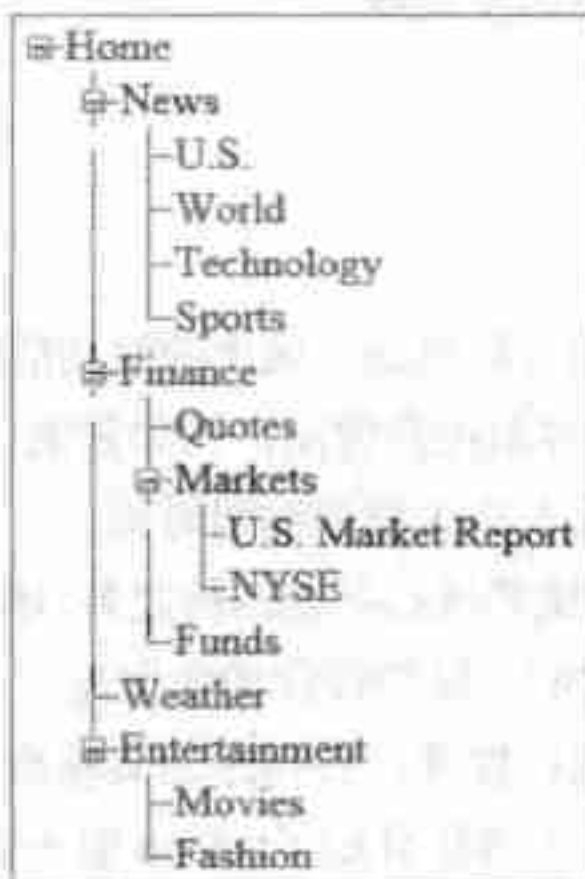


图 17-38

嵌套站点地图文件的另一种方法是在站点地图提供程序模型定义中建立第二个提供程序,再使用<siteMapNode>元素中的 provider 特性引用该声明。为此,应在 web.config 文件中添加一个新的站点地图提供程序引用,如程序清单 17-39 所示。

程序清单 17-39 在同一个 Web.sitemap 文件中使用另一个站点地图提供程序

```

<configuration>
  <system.web>

    <siteMap>
      <providers>
        <add siteMapFile="Entertainment.sitemap" name="AspNetXmlSiteMapProvider2"
          type="System.Web.XmlSiteMapProvider, System.Web, Version=4.0.0.0,
            Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
      </providers>
    </siteMap>

  </system.web>
</configuration>

```

在这段代码中，定义了第二个提供程序。这并不表示必须在<provider>部分使用<clear />元素，而只需定义使用了新名称的提供程序即可。本例中的提供程序名是 AspNetXmlSiteMapProvider2。另外，在这个提供程序定义中，siteMapFile 特性用于指向要使用的站点地图文件名。

之后，就可以在 Web.sitemap 文件中使用<siteMapNode>元素中的 provider 特性来引用该声明。要以这种方式添加 Entertainment.sitemap 文件，<siteMapNode>元素应如程序清单 17-40 所示。

程序清单 17-40 在 Web.sitemap 文件中使用第二个提供程序

```

<siteMapNode provider="AspNetXmlSiteMapProvider2" />

```

这也会得到如图 17-38 所示的结果。使用 provider 特性除了提供嵌套站点地图文件的另一种方式之外，还可以提供强大的功能。如果建立了一个新的站点地图提供程序，从另一个非 XML 文件的数据源中提取站点地图导航信息，就可以在主 Web.sitemap 文件中混合这些结果。最终结果中的数据项可以来自两个或多个完全不同的数据源。

17.11 本章小结

本章介绍了 ASP.NET 4.5 提供的导航机制。这个导航功能的核心是在 XML 文件中详细描述导航结构，而该 XML 文件可以由各种导航控件使用，例如新的 TreeView 控件和 SiteMapPath 控件。新导航机制的强大功能可以节省开发人员大量的编码时间。

除了探讨 ASP.NET 4.5 中导航系统的核心基础结构之外，本章还描述了 TreeView 和 SiteMapPath 控件，以及如何在应用程序中使用它们。这些控件的优点是，可以分层显示导航系统，让终端用户轻松漫游站点。另外，很容易修改这些控件，获得超出其标准外观和功能的特性。

除了 TreeView 服务器控件之外，本章还介绍了 Menu 服务器控件。这两个控件有许多相似之处，因为它们都提供了查看层次结构数据的方式。

最后，本章介绍了如何进行 URL 映射，以及如何本地化 Web.sitemap 文件，根据用户在角色管理系统中的角色，过滤站点导航内容的结果。

第18章

个 性 化

本章要点

- 探究 ASP.NET 的个性化功能
- 使用个性化属性
- 使用个性化提供程序存储数据
- 处理匿名用户的需求

许多 Web 应用程序都必须通过当前浏览页面的终端用户的信息进行定制。过去,开发人员通常通过 cookie、Session 对象或 Application 对象给浏览页面的终端用户存储个性化属性。cookie 可以永久地存储数据项,因此在终端用户返回 Web 页面时,可以提取所有与他相关的设置,以供应用程序再次使用。cookie 不是永久存储用户数据的最佳方式,因为不是所有的计算机都接受它们,专业的终端用户也很容易修改它们。

如第 19 章所述,ASP.NET 的成员资格和角色管理功能是 ASP.NET 存储用户信息的简便方式。开发人员如何使用相同的机制存储定制信息呢?

ASP.NET 4.5 提供了一个杰出的功能:个性化。这个最新版本中的 ASP.NET 个性化引擎可以把浏览页面的终端用户和为终端用户存储的所有数据项自动关联起来。以每个用户为基础维护的个性化属性存储在服务器而不是客户端上。这些数据项可以方便地放在自己选择的数据存储中(如 SQL Server),因此终端用户可以在以后浏览站点时浏览这些个性化属性。

这个功能非常适合于创建高度定制的、用于特定用户的站点,并且无须事先建立任何设置,因为已建立好相应的设置。个性化功能也是 ASP.NET 小组提高开发人员效率,从而更便于他们完成工作的一种方式。

18.1 个性化模型

ASP.NET 4.5 中的个性化模型非常简单,与 ASP.NET 中的大多数模型一样,它也是一个可扩展的模型。图 18-1 勾画出了个性化模型。

从图 18-1 中可以看出, 这个模型包括 3 层。首先查看个性化模型的中间层: 个性化服务层。这一层包含 Profile API, 这个 Profile API 层可以把终端用户的数据项编程到较低层的数据存储中。

内置于 ASP.NET 的一些控件可以利用个性化功能存储页面设置信息, 用户还可以使用这个新引擎存储自己的数据项。与 Web Parts 一样, 可以在 ASP.NET 页面中使用这些数据项。

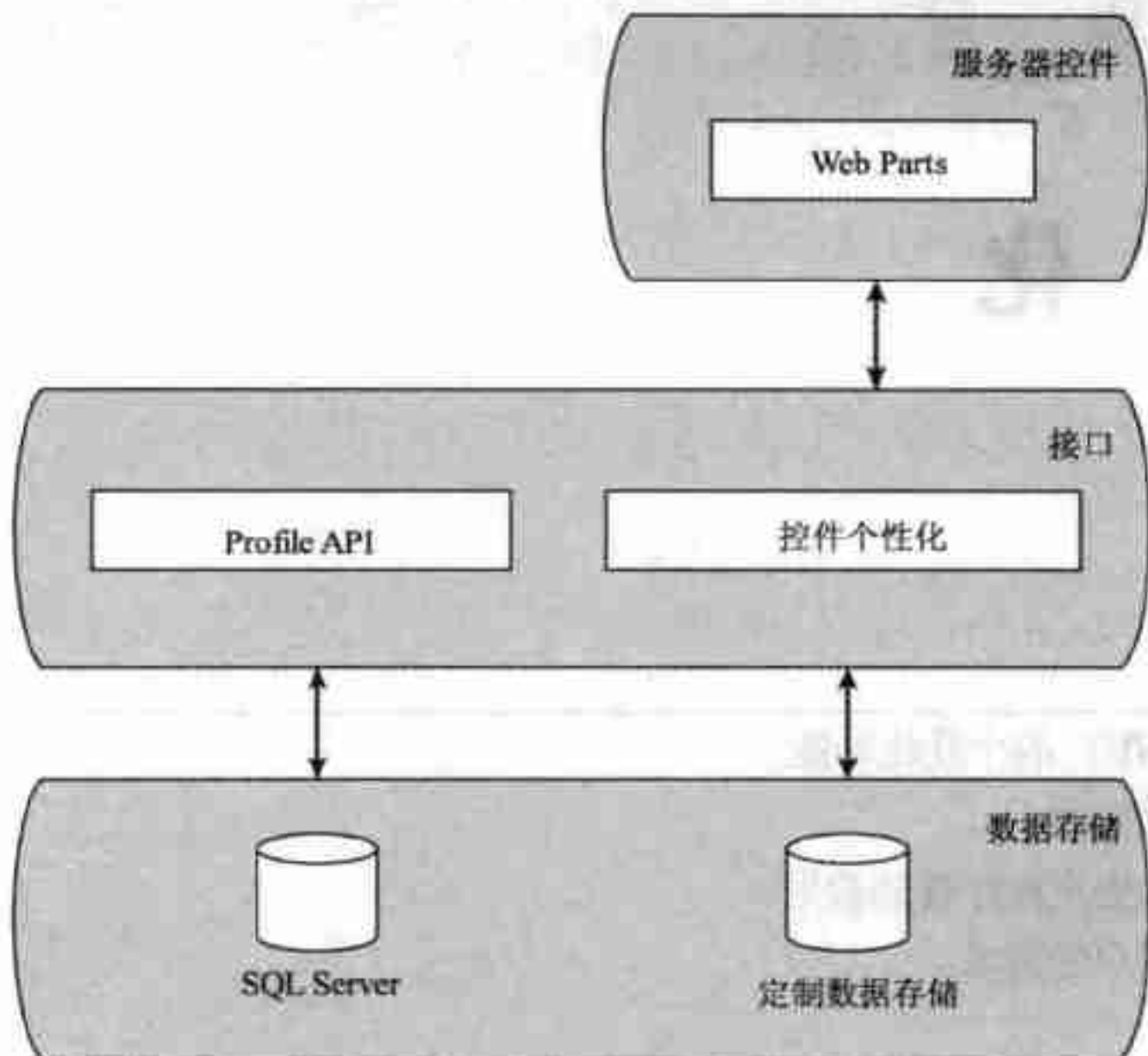


图 18-1

在个性化服务层的下面是用于处理 SQL Server 及 SQL Server Express Edition 文件的默认个性化数据提供程序。在应用 ASP.NET 4.5 的个性化功能时, 不仅可以这个数据存储, 还可以扩展模型, 为个性化引擎创建定制的数据提供程序。



第 15 章介绍了如何建立自己的提供程序。

前面简要介绍了个性化模型, 下面开始使用它创建一些存储的个性化属性, 以便在以后的应用程序中使用。

18.2 创建个性化属性

创建定制的个性化属性的优点是, 它们的创建非常容易。创建好这些属性后, 就可以对它们进行强类型化的访问。还可以创建仅由授权用户使用的个性化属性, 以及一些匿名用户可以使用的个性化属性。这些数据项是非常强大的, 主要是因为可以立即在应用程序中使用它们, 而无须建立底层的基础结构以支持它们。作为使用 ASP.NET 个性化系统的一个例子, 本章首先创建一些简单的个性化属性。以后再介绍如何在应用程序中使用这些个性化属性。

18.2.1 添加简单的个性化属性

当添加个性化属性时，必须确定要存储用户的哪些数据项。对于这个例子，创建几个可以在应用程序中使用的用户数据项，假定要存储用户的如下信息：

- 名
- 姓
- 上一次访问时间
- 年龄
- 成员资格状态

ASP.NET 总是把配置存储在 XML 文件中，ASP.NET 4.5 个性化引擎也不例外。所有涉及终端用户的这些定制数据项都在应用程序的 web.config 文件中定义和存储，如程序清单 18-1 所示(本章下载代码中的 web.config)。

程序清单 18-1 在 web.config 文件中创建个性化属性

```
<configuration>
  <system.web>
    <profile>
      <properties>
        <add name="FirstName" />
        <add name="LastName" />
        <add name="LastVisited" />
        <add name="Age" />
        <add name="Member" />
      </properties>
    </profile>
    <authentication mode="Windows" />
  </system.web>
</configuration>
```

在 web.config 文件的<system.web>部分创建了<profile>部分，以使用 ASP.NET 4.5 个性化引擎。在 web.config 文件的<profile>部分再创建<properties>部分。在<properties>部分，可以定义要个性化引擎存储的所有属性。

从这个代码示例中可以看出，使用<add>元素定义简单的属性是相当容易的。这个元素附带 name 特性，用于包含要存储的属性名。

首先假定访问使用这些属性建立的页面，该页面已经通过 Windows 验证(验证和授权详见下一章)。本章的后面将说明如何把个性化属性应用于匿名用户。把个性化属性应用于匿名用户的功能在默认情况下是禁用的。

定义好这些个性化属性之后，使用它们与定义它们一样简单。下面探讨如何在应用程序中使用这些定义。

18.2.2 使用个性化属性

在 web.config 文件中定义了个性化属性后，就可以在代码中使用这些项。例如，可以创建一个简单的表单，要求终端用户输入一些信息。用户提交表单时，数据就存储在个性化引擎中。

为此，应建立一个新的网站(本章后面将介绍 Web 应用程序和个性化属性在 Web 应用程序中较

难使用的原因), 给它添加程序清单 18-1 和 18-2 中的代码。再设置站点的属性, 禁用 `Anonymous`, 启用 `Windows` 验证(这应是默认设置)。这就允许 ASP.NET 存储当前用户的个性化信息。匿名用户也在本章后面介绍。

与验证功能一样, ASP.NET 默认也把 SQL Server Express 用作数据存储, 所以它也需要设置。程序清单 18-2 说明了如何访问在程序清单 18-1 中建立的个性化信息。

程序清单 18-2 使用定义好的个性化属性

```
<%@ Page Language="C#" %>
<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        if (Page.User.Identity.IsAuthenticated)
        {
            Profile.FirstName = TextBox1.Text;
            Profile.LastName = TextBox2.Text;
            Profile.Age = TextBox3.Text;
            Profile.Member = Radiobuttonlist1.SelectedItem.Text;
            Profile.LastVisited = DateTime.Now.ToString();
            Label1.Text = "<p>Stored information includes:</p><p>" +
                "First name: " + Profile.FirstName +
                "<br />Last name: " + Profile.LastName +
                "<br />Age: " + Profile.Age +
                "<br />Member: " + Profile.Member +
                "<br />Last visited: " + Profile.LastVisited + "</p>";
        }
        else
        {
            Label1.Text = "You must be authenticated!";
        }
    }
</script>
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Storing Personalization</title>
</head>
<body>
    <form id="form1" runat="server">
        <p>First Name:
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox></p>
        <p>Last Name:
        <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox></p>
        <p>Age:
        <asp:TextBox ID="TextBox3" runat="server" Width="50px"
        MaxLength="3"></asp:TextBox></p>
        <p>Are you a member?
        <asp:RadioButtonList ID="Radiobuttonlist1" runat="server">
            <asp:ListItem Value="1">Yes</asp:ListItem>
            <asp:ListItem Value="0" Selected="True">No</asp:ListItem>
        </asp:RadioButtonList></p>
        <p><asp:Button ID="Button1" runat="server" Text="Submit"
        OnClick="Button1_Click" />
    </form>
</body>
</html>
```

```

</p>
<hr /><p>
<asp:Label ID="Label1" runat="server"></asp:Label></p>
</form>
</body>
</html>

```

个性化属性的使用方式类似于过去使用 Session 对象的方式，但注意这里存储和获取的个性化属性都不基于键。因此，在使用它们时不需要记住键名。

个性化系统存储的所有项都会被强制转换为.NET 数据类型。这些项默认存储为 String 类型，可以对这些项进行早期绑定访问。要存储某一项，只需使用 Profile 对象直接填充个性化属性：

```
Profile.FirstName = TextBox1.Text
```

要提取相同的信息，只需使用 Profile 类的相应属性即可，如下所示：

```
Label1.Text = Profile.FirstName
```

在源代码视图中使用 Profile 类和所有其他个性化属性的优点是，在建立页面时，这种方法可以提供 IntelliSense。在源代码视图中使用 Profile 类时，已定义的所有项都通过 IntelliSense 功能显示为可用选项，如图 18-2 所示。

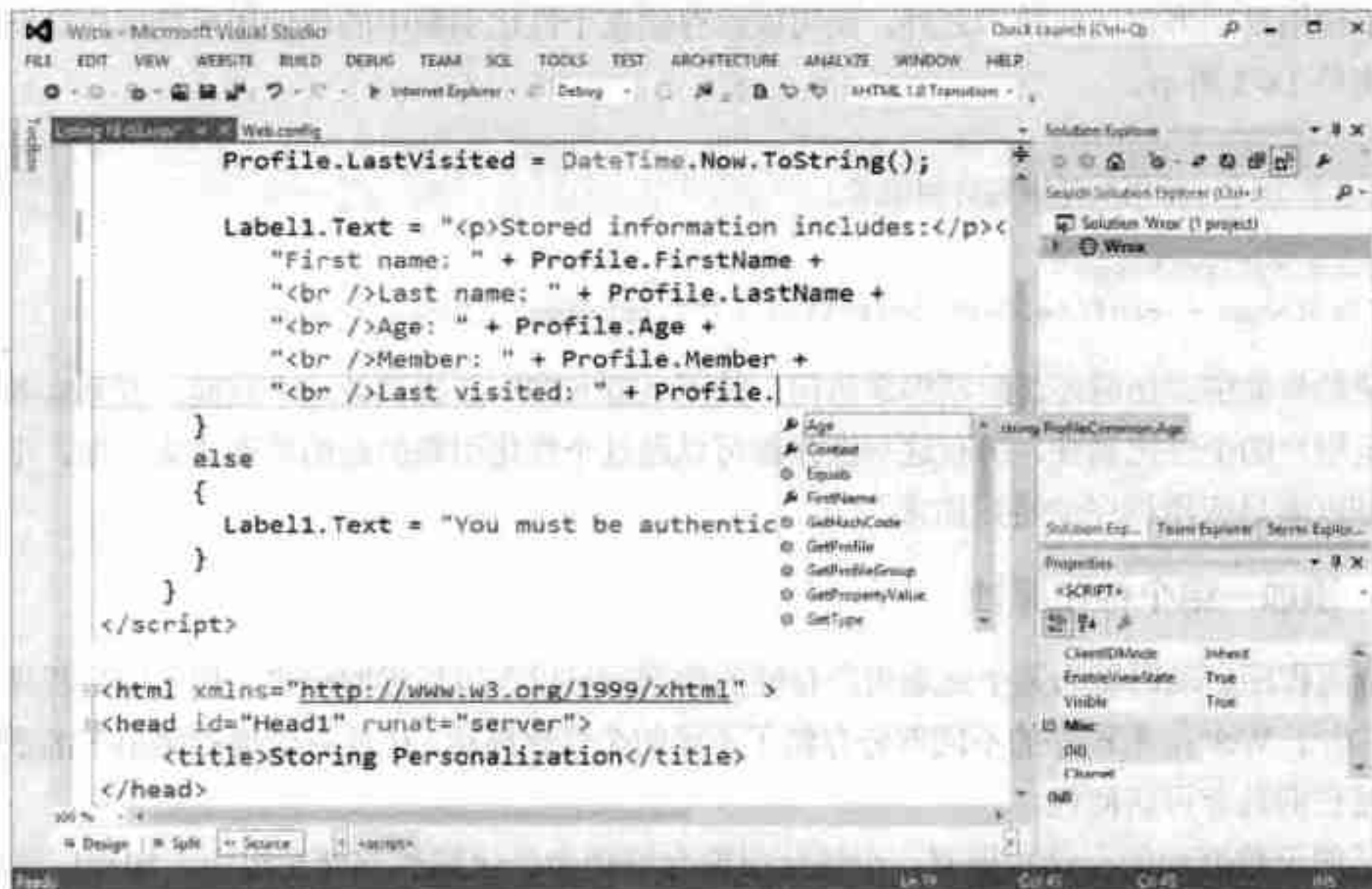


图 18-2

所有这些属性都可以在 IntelliSense 中访问，因为 Profile 类是隐藏的，所以在把对个性化的修改保存到 web.config 文件中时，会在后台动态编译 Profile 类。将这些项保存到 web.config 文件中后，就可以在整个应用程序中以强类型化的方式使用这些属性。

运行程序清单 18-2 中的页面，结果如图 18-3 所示。可能需要在网站设置中激活 Windows 验证功能，并禁用匿名验证功能。

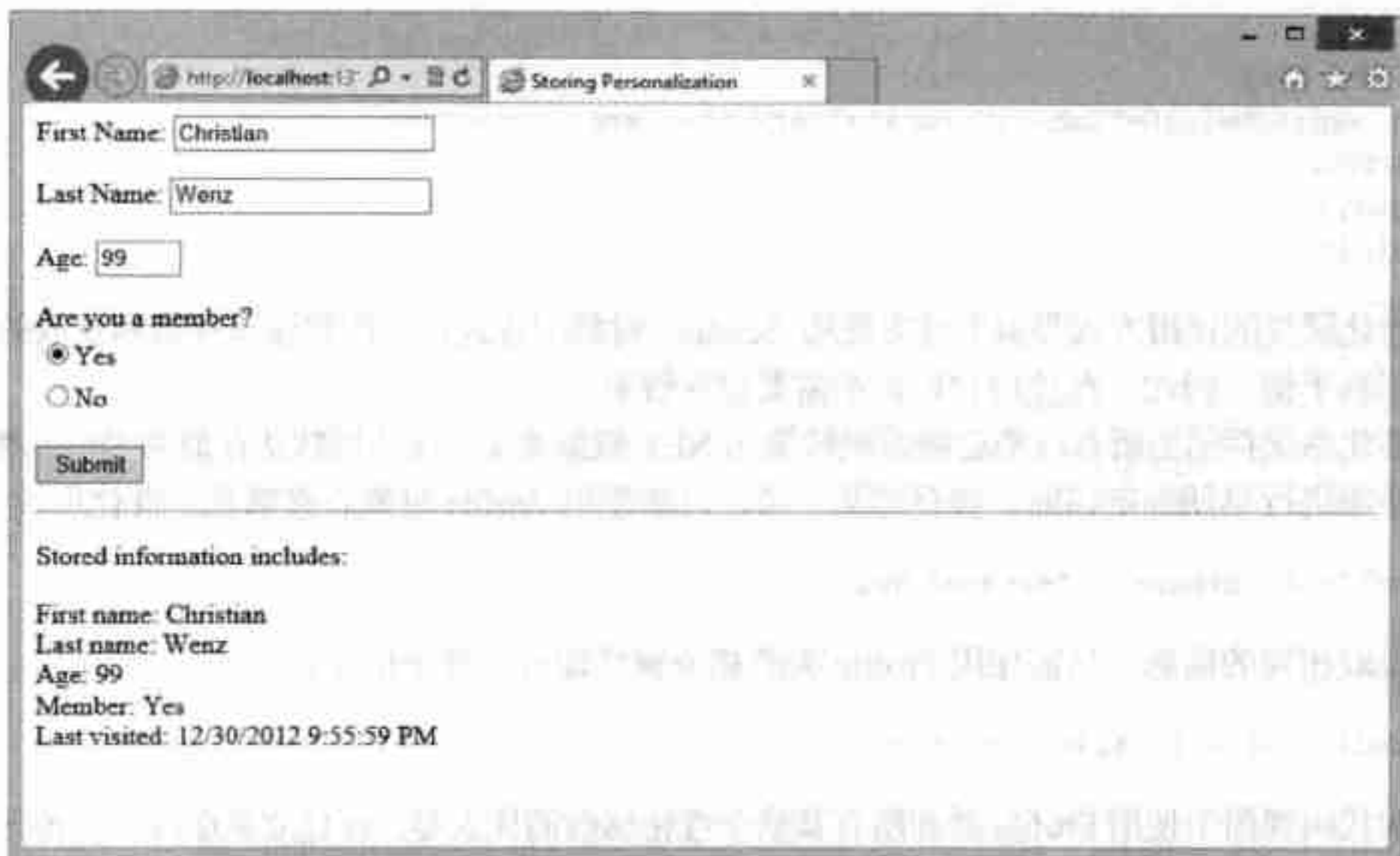


图 18-3

除了使用早期绑定访问技术之外，还可以给存储在个性化引擎中的项使用后期绑定访问技术，如程序清单 18-3 所示。

程序清单 18-3 使用后期绑定访问技术

```
string myFirstName;  
myFirstName = Profile.GetPropertyValue("FirstName").ToString();
```

无论是早期绑定访问还是后期绑定访问，使用 ASP.NET 4.5 提供的这个功能，都很容易存储和获取特定用户的个性化属性。所有这些操作都可以通过个性化引擎的最简单形式来实现。下面看看如何定制以满足应用程序的特定需求。

18.2.3 添加一组个性化属性

在应用程序启动时就为某个终端用户存储的数据项可以在以后提取出来，用于应用程序中的其他页面。由于 Web 应用程序的不同部分存储了不同的个性化属性，因此有时要存储很多的数据项集合，并使它们具有可访问性。

为了便于数据项的存储和提取，个性化引擎允许把个性化属性存储在组中，如程序清单 18-4 所示。

程序清单 18-4 在 web.config 文件中创建个性化组

```
<configuration>  
  <system.web>  
    <profile>  
      <properties>  
        <add name="FirstName" />  
        <add name="LastName" />  
        <add name="LastVisited" />  
      </properties>  
    </profile>  
  </system.web>  
</configuration>
```



```

<add name="Age" />
<group name="MemberDetails">
  <add name="Member" />
  <add name="DateJoined" />
  <add name="PaidDuesStatus" />
  <add name="Location" />
</group>
<group name="FamilyDetails">
  <add name="MarriedStatus" />
  <add name="DateMarried" />
  <add name="NumberChildren" />
  <add name="Location" />
</group>
</properties>
</profile>
<authentication mode="Windows" />
</system.web>
</configuration>

```

程序清单 18-4 中的代码存储在 web.config 文件中，其中列出了两个组。第 1 个组是 MemberDetails，定义了 4 项；第 2 个组是 FamilyDetails，定义了另外 3 个相关的项。使用 <properties> 部分的 <group> 元素定义个性化组。使用 name 特性指定组的名称，就像指定 <add> 元素一样。可以根据需要或采用推荐的最佳实践方式来定义任意多个组。

18.2.4 使用分组的个性化属性

在程序清单 18-4 中，一些项没有在组中定义。可以把组中定义的属性与没有在组中定义的属性混合在一起。仍然可以使用前面说明的方式访问程序清单 18-4 中未在组中定义的项：

```
Label1.Text = Profile.FirstName
```

现在，使用个性化组，可以通过嵌套的名称空间以逻辑方式访问已定义的项：

```

Label1.Text = Profile.MemberDetails.DateJoined
Label2.Text = Profile.FamilyDetails.MarriedStatus

```

对于这个例子，在每个已定义的个性化组中，使用逻辑方式访问两个不同的项。对于在应用程序的 web.config 文件中定义的属性，每个组都有同名的属性 Location，这是允许的结构，因为使用个性化组定义它们。采用这种结构，可以通过指定相应的组来获取各个 Location 属性：

```

Label1.Text = Profile.MemberDetails.Location
Label2.Text = Profile.FamilyDetails.Location

```

18.2.5 为个性化属性定义类型

默认情况下，在存储个性化属性时，这些属性都创建为 System.String 类型。通过 web.config 文件中的配置设置可以很容易把该类型改为其他类型。要定义个性化属性的名称和类型，可以使用 <properties> 部分的 <add> 元素的 type 特性，如程序清单 18-5 所示。

程序清单 18-5 为个性化属性定义类型

```
<properties>
```

```

<add name="FirstName" type="System.String" />
<add name="LastName" type="System.String" />
<add name="LastVisited" type="System.DateTime" />
<add name="Age" type="System.Int32" />
<add name="Member" type="System.Boolean" />
</properties>

```

前两个属性 `FirstName` 和 `LastName` 的类型被强制转换为 `System.String`, 这不是必需的操作。即使忽略这一步, 它们也会转换为 `String` 类型, 因为这是个性化系统中属性的默认类型(假定没有定义其他类型)。下一个个性化属性是 `LastVisited`, 它被定义为 `System.DateTime` 类型, 用于存储终端用户上次访问页面的日期和时间。除此之外, 使用特定的 .NET 数据类型定义其他个性化属性。

这是首选方法, 因为在编写应用程序以使用自己定义的个性化属性时, 可以对它们进行类型检查。

18.2.6 使用定制的类型

从上面为个性化属性定义类型的例子中可以看出, 很容易定义属性并将其强制转换为 .NET Framework 中可用的数据类型。可以很容易在 `web.config` 文件中定义诸如 `System.Integer`、`System.String`、`System.DateTime`、`System.Byte` 和 `System.Boolean` 这样的项, 如何定义复杂的类型呢?

定义使用定制类型的个性化属性与定义使用简单类型的个性化属性一样简单。定制类型可以存储复杂的项, 例如从一个应用程序传送给另一个应用程序的购物车信息或其他状态信息。程序清单 18-6 显示了类 `ShoppingCart`, 后面要在个性化属性定义中使用它(本章下载代码中的 `ShoppingCart.cs`)。

程序清单 18-6 创建要用作个性化类型的类

```

using System;
[Serializable]
public class ShoppingCart
{
    private string PID;
    private string CompanyProductName;
    private int Number;
    private decimal Price;
    private DateTime DateAdded;
    public ShoppingCart() {}
    public string ProductID
    {
        get { return PID; }
        set { PID = value; }
    }
    public string ProductName
    {
        get { return CompanyProductName; }
        set { CompanyProductName = value; }
    }
    public int NumberSelected
    {
        get { return Number; }
        set { Number = value; }
    }
    public decimal ItemPrice

```



```

    {
        get { return Price; }
        set { Price = value; }
    }
    public DateTime DateItemAdded
    {
        get { return DateAdded; }
        set { DateAdded = value; }
    }
}

```

这个简单的购物车结构现在可以在终端用户浏览电子商务站点时存储购物车。甚至可以将购物车保存起来，等待终端用户下次返回站点。一定要注意，这个类需要把 `Serializable` 属性放在类声明的前面，这样才能确保将信息正确转换为 XML 或二进制。

下面看看如何在 `web.config` 文件中为个性化属性指定这种复杂类型(例如 `ShoppingCart` 类型)，如程序清单 18-7 所示。

程序清单 18-7 为个性化属性使用复杂类型

```

<properties>
  <add name="FirstName" type="System.String" />
  <add name="LastName" type="System.String" />
  <add name="LastVisited" type="System.DateTime" />
  <add name="Age" type="System.Int32" />
  <add name="Member" type="System.Boolean" />
  <add name="Cart" type="ShoppingCart" serializeAs="Binary" />
</properties>

```

就像在个性化数据存储中存储基本数据类型一样，这个结构也可以存储定制类型，以自己选择的格式把它们序列化到最终的数据存储中。这里把 `ShoppingCart` 对象序列化到数据存储的二进制对象中。`serializeAs` 特性的值可以是：

- **Binary**：序列化对象，并将其存储为所选数据存储中的二进制数据。
- **ProviderSpecific**：根据提供程序的指令存储对象。这表示不是由个性化引擎确定如何序列化对象，而是由个性化提供程序确定。
- **String**：默认设置。把个性化属性存储为所选数据存储中的字符串。
- **XML**：提取对象，把它序列化为 XML 格式，再存储到所选的数据存储中。

18.2.7 提供默认值

除了定义个性化属性的数据类型之外，还可以在 `web.config` 文件中直接定义它们的默认值。在默认情况下，我们创建的个性化属性没有值，但很容易使用 `<add>` 元素的 `defaultValue` 特性给它们指定默认值。定义默认值的过程如程序清单 18-8 所示。

程序清单 18-8 定义个性化属性的默认值

```

<properties>
  <add name="FirstName" type="System.String" />
  <add name="LastName" type="System.String" />
  <add name="LastVisited" type="System.DateTime" />

```



```
<add name="Age" type="System.Int32" />
<add name="Member" type="System.Boolean" defaultValue="false" />
</properties>
```

在这个例子中，只给一个个性化属性提供了默认值。最后一个个性化属性 `Member` 的默认值指定为 `false`。也就是说，在给个性化属性数据库添加新的终端用户时，要定义 `Member` 属性而不是使用空值。

18.2.8 把个性化属性指定为只读

也可以把个性化属性指定为只读。为此，只需给 `<add>` 元素添加 `readOnly` 特性：

```
<add name="StartDate" type="System.DateTime" readOnly="true" />
```

要把个性化属性指定为只读，应将 `readOnly` 特性的值设置为 `true`。该特性的默认值是 `false`。

18.3 匿名个性化

ASP.NET 的功能之一是允许匿名的终端用户利用该框架提供的个性化功能。如果站点需要某种形式的注册，这就是很重要的功能。例如，许多电子商务站点允许匿名终端用户在注册之前就在该站点上购物，使用该站点的购物车。

18.3.1 支持终端用户的匿名身份

匿名个性化功能默认为禁用，因为它会消耗流行站点上的数据库资源。因此，允许匿名个性化的第一步是使用 `web.config` 文件中的相应设置启用这个功能。还需要根据属性在 `web.config` 文件中的具体定义进行一些修改，以确定是否允许对应用程序进行匿名个性化。

如程序清单 18-9 所示，可以利用 `<anonymousIdentification>` 元素启用匿名身份识别功能，让个性化引擎识别未知的终端用户。

程序清单 18-9 允许进行匿名身份识别

```
<configuration>
  <system.web>
    <anonymousIdentification enabled="true" />
  </system.web>
</configuration>
```

为了对可能访问应用程序的终端用户启用匿名身份识别，应在 `web.config` 文件的 `<system.web>` 节点下添加 `<anonymousIdentification>` 元素。然后在 `<anonymousIdentification>` 元素中使用 `enabled` 特性，把它的值设置为 `true`。该特性默认设置为 `false`。

启用匿名身份识别功能后，ASP.NET 会给每个访问应用程序的匿名用户使用唯一的标识符。这个标识符会随每个请求一起发送，但在终端用户由 ASP.NET 验证身份后，该标识符就会被删除。

对于匿名用户，其信息在终端用户的计算机上默认存储为 `cookie`。其他信息(给匿名用户提供的个性化属性)存储在服务器的特定数据存储中。

为了演示匿名身份识别功能，请关闭示例应用程序的 Windows 验证，而改为使用窗体验证，如

程序清单 18-10 所示。

程序清单 18-10 关闭 Windows 验证，使用窗体验证

```
<configuration>
  <system.web>
    <anonymousIdentification enabled="true" />
    <authentication mode="Forms" />
  </system.web>
</configuration>
```

有了这段代码，然后运行程序清单 18-2 中的页面，就会看到如程序清单 18-11 中所示的标题。

程序清单 18-11 在 HTTP 标题中设置匿名 cookie

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/8.0
Date: Sun, 30 Dec 2012 21:02:33 GMT
X-AspNet-Version: 4.0.30319
X-SourceFiles:
=?UTF-8?B?QzpcVXNlcncQ2hyaXN0aWwFuXERvY3VtZW50c1xWaXNlYWwgU3RlZGlvIDIwMTJcV2ViU2l0ZXNcV3JveFwMaXN0aWw5nIDE4LTAyLmFzcHg=?=
X-Powered-By: ASP.NET
Set-Cookie: .ASPXANONYMOUS=HduG02IdzgEkAAAAMzA4NzIxZDUtZDk4ZC00ODQyLTk3ZDIyY2RhNzI5N2NiODZjq7SGklbzPMq49DZkIgxPDBBXlk7ZD2mP5lC5e4xcPFYl; expires=Sun, 10-Mar-2013 07:42:33 GMT; path=/; HttpOnly
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 1419
Connection: Close
```

在这个 HTTP 标题中，将 cookie(即.ASPXANONYMOUS)设置为散列值，在后面供 ASP.NET 个性化系统进行检索。

1. 为匿名身份识别修改 cookie 的名称

cookie 默认使用的名称是.ASPXANONYMOUS。在 web.config 文件的<anonymousIdentification>元素中，使用 cookieName 特性可以修改这个 cookie 的名称，如程序清单 18-12 所示。

程序清单 18-12 修改 cookie 的名称

```
<configuration>
  <system.web>
    <anonymousIdentification
      enabled="true"
      cookieName=".ASPXWenzWebApplication" />
  </system.web>
</configuration>
```

2. 修改存储 cookie 的时间长度

另外，存储在终端用户计算机上的 cookie 默认存储 100 000 分钟(大约 70 天)。如果要修改这个

值, 可以在<anonymousIdentification>元素中使用 cookieTimeout 特性, 如程序清单 18-13 所示。

程序清单 18-13 修改存储 cookie 的时间长度

```
<configuration>
  <system.web>
    <anonymousIdentification
      enabled="true"
      cookieTimeout="1440" />
  </system.web>
</configuration>
```

在这个例子中, 将 cookieTimeout 的值改为 1440, 即 1440 分钟(一天)。如果不想把终端用户的身份保留太长时间, 这就是保持购物车的理想时间。

3. 改变标识符的存储方式

尽管使用 cookie 存储匿名标识符, 但是可以改变这种方法。cookie 是获得身份的首选方式, 但不使用 cookie 也可以获得身份。其他方式有使用 URI 或设备配置文件。程序清单 18-14 是使用 URI 保存标识符的一个例子。

程序清单 18-14 指定 cookie 的存储方式

```
<configuration>
  <system.web>
    <anonymousIdentification
      enabled="true"
      cookieless="UseUri" />
  </system.web>
</configuration>
```

除了 UseUri 之外, 其他选项包括 UseCookies、AutoDetect 和 UseDeviceProfile。下面将介绍这些选项:

- **UseCookies:** 这是默认设置。如果没有设置值, ASP.NET 就假定使用这个值。UseCookies 表示 cookie 放在终端用户的计算机上, 用于身份识别。
- **UseUri:** 这个值表示 cookie 不存储在终端用户的计算机上, 而是把唯一标识符放在页面的 URL 中。如果开发人员想避免把 cookie 放在终端用户的计算机上, 这个选项就很适合, 但会创建古怪的 URL, 在终端用户给页面添加书签以便以后浏览时, 这可能会成为问题。
- **AutoDetect:** 使用这个值会让 ASP.NET 引擎确定是给匿名身份识别使用 cookie 还是 URL。对每个用户都要进行一次确认, 因此该选项执行起来比其他两个选项糟糕一些。ASP.NET 必须在确认使用哪个方法前检查终端用户。如果必须让终端用户关闭 cookie(目前很少见), 最好使用 AutoDetect 代替 UseUri。
- **UseDeviceProfile:** 为发出请求的设备或浏览器配置标识符。

4. 查看存储的匿名标识符

ASP.NET 使用全局唯一的 GUID, 使匿名标识符保持唯一。现在还可以获取这个唯一标识符以

供自己使用。要获取 GUID，就要使用 AnonymousID 属性改进 Request 对象。AnonymousID 属性返回一个 String 类型的值，它可以用于代码中，如下所示：

```
Label1.Text = Request.AnonymousID
```

18.3.2 使用匿名身份识别事件

在创建匿名用户的过程中，注意可以使用 Global.asax 文件中的一个重要事件来管理该过程：AnonymousIdentification_Creating。

使用 AnonymousIdentification_Creating 事件，可以进行终端用户的身份识别。例如，如果不想使用 GUID 唯一地标识终端用户，就可以在这个事件中改变标识值。

为此，使用 AnonymousIdentificationEventArgs 类型的事件代理创建该事件，如程序清单 18-15 所示。

程序清单 18-15 改变匿名用户的唯一标识符

```
public void AnonymousIdentification_Creating(object sender,
    AnonymousIdentificationEventArgs e)
{
    e.AnonymousID = "Bubbles " + DateTime.Now;
}
```

AnonymousIdentificationEventArgs 事件委托的 AnonymousID 属性用于唯一地标识匿名用户。现在不使用 GUID 唯一地标识匿名用户：

```
d13fafec-244a-4d21-9137-b213236ebedb
```

而是在 AnonymousIdentification_Creating 事件中把 AnonymousID 属性改为：

```
Bubbles 12/31/2012 2:07:33 PM
```

18.3.3 个性化属性的匿名选项

如果试图使匿名功能发挥作用，就可能会得到如图 18-4 所示的错误。

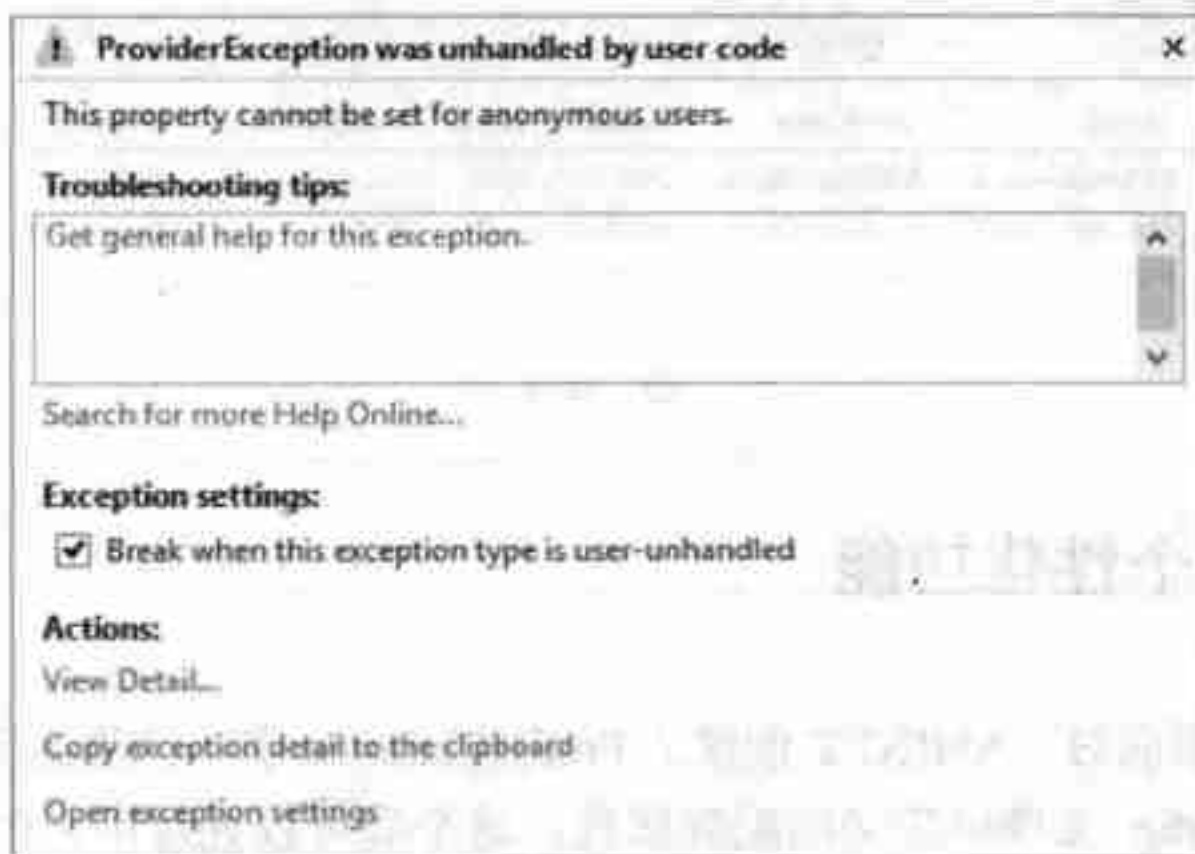


图 18-4

有了处理匿名用户的功能后,还必须给浏览页面的匿名用户指定支持哪些个性化属性。为此,在 web.config 文件的<properties>部分,给已定义属性的<add>元素添加 allowAnonymous 特性,如程序清单 18-16 所示。

程序清单 18-16 为个性化属性启用匿名功能

```
<properties>
  <add name="FirstName" type="System.String" />
  <add name="LastName" type="System.String" />
  <add name="LastVisited" type="System.DateTime" allowAnonymous="true" />
  <add name="Age" type="System.Integer" />
  <add name="Member" type="System.Boolean" />
</properties>
```

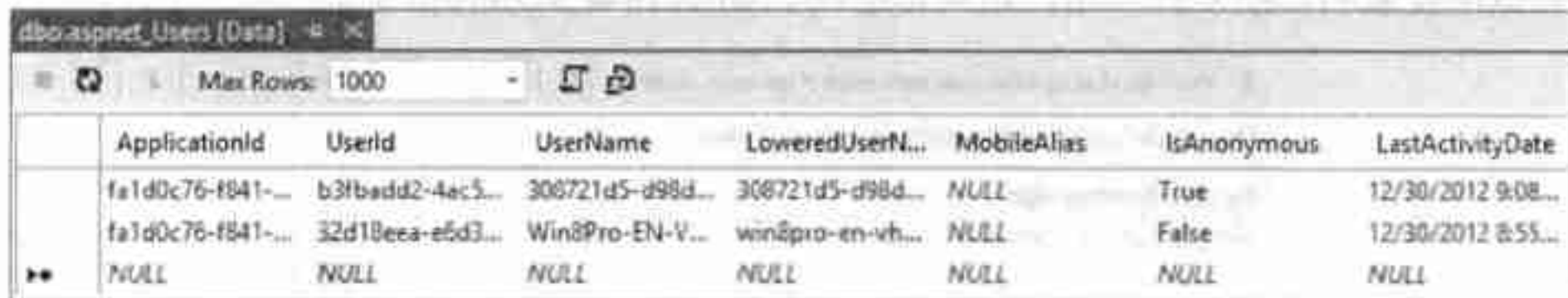
在这个例子中,设置 LastVisited 属性以允许通过把 allowAnonymous 特性设置为 true 来支持匿名用户访问。由于这是处理匿名用户的唯一属性,因此其他已定义的属性都不为这类用户存储信息。如果仍然要检查该用户是否已在代码中得到了身份验证,确定已将添加了 allowAnonymous 特性的一行代码注释掉,以使示例能正常运行。

18.3.4 有关匿名用户配置信息的警告

考虑到前面介绍的有关匿名用户的所有内容,我们应对采用什么方法存储匿名用户的配置信息非常谨慎。存储匿名用户的配置信息会大量占用所使用的数据存储。例如在下面的例子中,使用 SQL Server Express Edition 存储了一个已验证用户的配置信息和一个匿名用户的配置信息。这会把这两个用户的信息存放在 aspnet_Profile 和 aspnet_Users 表中。

aspnet_Users 表中的两个用户如图 18-5 所示。

在图 18-5 中,匿名用户在表中的第一行显示。这个用户有个很古怪的名称,就是前面显示的 Request.AnonymousID。两个用户的另一区别是表中的 IsAnonymous 列。匿名用户的这一列设置为 true,而已验证的用户设置为 false。因为数据库可能会很快填满大量的匿名用户信息,所以应考虑真正需要为这类用户存储什么信息。



ApplicationId	UserId	UserName	LoweredUserN...	MobileAlias	IsAnonymous	LastActivityDate
fa1d0c76-f841-...	b3fbadd2-4ac5-...	308721d5-d98d...	308721d5-d98d...	NULL	True	12/30/2012 9:08...
fa1d0c76-f841-...	32d18eea-e5d3-...	Win8Pro-EN-V...	win8pro-en-vh...	NULL	False	12/30/2012 8:55...
NULL	NULL	NULL	NULL	NULL	NULL	NULL

图 18-5

18.4 编程访问个性化功能

在调用 ASP.NET 页面时,ASP.NET 创建了 ProfileCommon 类,该类继承了 ProfileBase 类,用于强类型化在 web.config 文件中定义的配置属性。这个类可以处理用户的配置文件存储,使用 ProfileBase 类的 GetPropertyValue 和 SetPropertyValue 方法获取和设置配置属性。

ASP.NET 使用 ProfileModule 类提供了获取特定 Profile 事件所需的钩子(hook),使用 ProfileModule 类

在页面的 Profile 对象中创建和存储配置文件信息。

ProfileModule 类有 3 个事件,可用于处理用户的配置文件。这些事件分别是 MigrateAnonymous、Personalize 和 ProfileAutoSaving,主要用于身份验证。前面介绍了如何在应用程序中处理匿名用户,因此下面首先探讨如何把这些用户变成通过验证的用户——因为我们常常要移动它们的配置属性,修改它们的状态。

18.4.1 迁移匿名用户

在处理匿名用户时,必须把匿名用户变成注册用户。例如,终端用户填满了购物车后,就可以在站点上注册,以购买这些商品。此时,终端用户就要从匿名用户变成注册用户。

因此,ASP.NET 提供了 Profile_MigrateAnonymous 事件处理程序,将匿名用户变成注册用户。Profile_MigrateAnonymous 事件需要一个 ProfileMigrateEventArgs 类型的数据类。将该事件放在处理迁移的页面上或 Global.asax 文件中(可以在应用程序的任何地方使用)。这个事件的使用如程序清单 18-17 所示。

程序清单 18-17 为特定的个性化属性迁移匿名用户

```
public void Profile_MigrateAnonymous(object sender, ProfileMigrateEventArgs e)
{
    ProfileCommon anonymousProfile = Profile.GetProfile(e.AnonymousID);
    Profile.LastVisited = anonymousProfile.LastVisited;
}
```

在这个例子中创建了一个 ProfileCommon 对象,使用访问页面的匿名用户的配置信息填充它。之后就可以使用该对象获取匿名用户的所有配置属性。这表示可以把匿名用户的配置信息移到已验证用户的配置系统中,填充其配置信息。

程序清单 18-17 演示了如何把个性化属性从匿名用户迁移到新的注册用户。除了迁移单个属性之外,还可以迁移个性化组中的属性,如程序清单 18-18 所示。

程序清单 18-18 为个性化组中的项迁移匿名用户

```
public void Profile_MigrateAnonymous(object sender, ProfileMigrateEventArgs e)
{
    ProfileCommon au = Profile.GetProfile(e.AnonymousID);
    if(au.MemberDetails.DateJoined != String.Empty) {
        Profile.MemberDetails.DateJoined = DateTime.Now.ToString();
        Profile.FamilyDetails.MarriedStatus = au.FamilyDetails.MarriedStatus;
    }
    AnonymousIdentificationModule.ClearAnonymousIdentifier();
}
```

使用 Global.asax 文件中的这个事件,可以实现匿名用户在应用程序中注册时有逻辑地迁移他们。迁移过程还允许选择要迁移的项,并按照自己的需要修改其值。

18.4.2 配置信息的个性化

除了处理 Global.asax 文件中的匿名用户之外,还可以使用编程的方式个性化从个性化存储中提取出来的配置信息。为此,需要使用 Profile_Personalize 事件,这个事件的使用示例如程序清单 18-19 所示。

程序清单 18-19 个性化所提取的配置信息

```

public void Profile_Personalize(object sender, ProfileEventArgs args)
{
    ProfileCommon checkedProfile;
    if(User == null) { return; }
    checkedProfile = (ProfileCommon)ProfileBase.Create(User.Identity.Name);
    if(DateTime.Now.IsDaylightSavingTime()) {
        checkedProfile = checkedProfile.GetProfile("TimeDifferenceUser");
    }
    else {
        checkedProfile = checkedProfile.GetProfile("TimeUser");
    }
    if(checkedProfile != null) {
        args.Profile = checkedProfile;
    }
}

```

在这个例子中, 根据特定的参数(无论是 Daylight Savings Time 还是其他值), 可以把特定的配置信息赋予用户。为此, 可使用 ProfileBase.Personalize 事件, 该事件通常位于 Global.asax 页面中。

18.4.3 确定是否继续自动保存配置信息

在使用 ASP.NET 提供的配置信息功能时, 页面会在执行结束时把配置信息自动保存到特定的数据存储中。这个功能默认为启用状态(设置为 true), 但是可以在 web.config 文件中使用 <profile> 节点的 automaticSaveEnabled 特性把它设置为 false, 如程序清单 18-20 所示。

程序清单 18-20 使用 automaticSaveEnabled 特性

```

<profile automaticSaveEnabled="false">
  <properties>
    <add name="FirstName" />
    <add name="LastName" />
    <add name="LastVisited" />
    <add name="Age" />
    <add name="Member" />
  </properties>
</profile>

```

如果把 automaticSaveEnabled 特性的值设置为 false, 就必须亲自调用 ProfileBase.Save 方法。但在大多数情况下, 应把这个特性设置为 true。在发出页面请求并处理该请求后, 就会触发 ProfileModule.ProfileAutoSaving 事件。也可以利用这个事件来禁用自动保存功能, 如程序清单 18-21 所示。将该事件放在 Global.asax 文件中。

程序清单 18-21 使用 ProfileAutoSaving 事件禁用自动保存功能

```

public void Profile_ProfileAutoSaving(object sender, ProfileAutoSaveEventArgs args)
{
    if(Profile.PaidDueStatus.HasChanged) {
        args.ContinueWithProfileAutoSave = true;
    }
}

```

```

    } else {
        args.ContinueWithProfileAutoSave = false;
    }
}

```

在这个例子中，触发 Profile_ProfileAutoSaving 事件时，就可以在该事件中修改一些行为。程序清单 18-21 检查 Profile.PaidDueStatus 属性是否已改变。如果已改变，就继续启用配置系统的自动保存功能；如果 Profile.PaidDueStatus 属性没有改变，就禁用自动保存功能。

18.4.4 在 Web Application Projects 中使用配置信息

尽管访问配置属性很方便，但在默认情况下，只有 ASP.NET Web Site Projects (WSP) 支持这个功能，而更常用的是 ASP.NET Web Application Projects (WAP)。如果使用其中一个模板，就不能对属性进行快捷访问。使用 WSP 时，ASP.NET 会自动创建代理类，其中包含了所有的配置属性；而 WAP 不包含它们。

但是，有一种较容易的方法来模拟属性的快捷访问：创建自己的配置类。ASP.NET 团队的前成员 Jon Galloway 在博客日记中做了详细描述，网址是 <http://weblogs.asp.net/jgalloway/archive/2008/01/19/writing-a-custom-asp-net-profile-class.aspx>，这里只介绍基本步骤。

首先需要创建一个派生自 ProfileBase 的类，在其中添加需要的配置选项属性。使用 SettingsAllowAnonymous 特性甚至可以支持匿名访问。程序清单 18-22 是一个示例，它实现了本章使用的配置属性的简化版本。

程序清单 18-22 在 Web Application Projects 中使用配置属性

```

using System.Web.Profile;
using System.Web.Security;

namespace Wrox
{
    public class UserProfile : ProfileBase
    {
        public static UserProfile GetUserProfile(string username)
        {
            return Create(username) as UserProfile;
        }

        public static UserProfile GetUserProfile()
        {
            return Create(Membership.GetUser().UserName) as UserProfile;
        }

        [SettingsAllowAnonymous(false)]
        public string FirstName
        {
            get { return base["FirstName"] as string; }
            set { base["FirstName"] = value; }
        }

        [SettingsAllowAnonymous(false)]
        public string LastName
    }
}

```

```

    {
        get { return base["LastName"] as string; }
        set { base["LastName"] = value; }
    }

    [SettingsAllowAnonymous(true)]
    public DateTime LastVisited
    {
        get { return base["LastVisited"] as DateTime; }
        set { base["LastVisited"] = value; }
    }
}

```

接着需要确保在 web.config 中，配置属性来自于刚才创建的类。程序清单 18-23 中的设置说明了具体做法。

程序清单 18-23 在 web.config 中引用配置类

```

<profile defaultProvider="AspNetSqlProfileProvider" inherits="Wrox.UserProfile">
</profile>

```

最后，程序清单 18-19 中的 API 提供了对配置属性的访问。

18.5 个性化提供程序

如本章前面的图 18-1 所示，个性化模型的中间层是个性化 API 层，它与一系列默认的数据提供程序通信。个性化模型默认使用 SQL Server Express Edition 文件存储我们定义的个性化属性。我们不仅可以使这种数据存储，还可以使用 SQL Server 数据提供程序处理 SQL Server 7.0 及以后版本。除了 SQL Server 数据提供程序之外，如果这些数据存储不能满足要求，该体系结构还允许用户创建自己的数据提供程序。

18.5.1 使用 SQL Server Express Edition

SQL Server 数据提供程序可以使用 SQL Server Express Edition 文件。SQL Server 数据提供程序是 ASP.NET 提供的个性化系统使用的默认提供程序。在与 Visual Studio 2012 一起使用时，IDE 把 aspnetdb.mdf 文件放在应用程序的 App_Data 文件夹中。

查看 machine.config 文件，注意用于处理个性化引擎如何使用这个数据库的部分。在 LocalSqlServer 文件的第一个引用中，可以在该文件的 <connectionStrings> 部分找到该文件的连接字符串，如程序清单 18-24 所示。

程序清单 18-24 给 SQL Server Express Edition 文件添加连接字符串

```

<configuration>
  <connectionStrings>
    <clear />
    <add name="LocalSqlServer"
      connectionString="data source=.\SQLEXPRESS;Integrated Security=SSPI;

```



```

        AttachDBFilename=|DataDirectory|aspnetdb.mdf;User Instance=true"
        providerName="System.Data.SqlClient" />
    </connectionStrings>
</configuration>

```

在这个例子中，定义了连接字符串 LocalSqlServer。文件的位置由 connectionString 特性指定，它指向文件的相对路径。这就表示在创建的每个利用个性化功能的应用程序中，默认的 SQL Server 提供程序应位于应用程序的 App_Data 文件夹中，并且以 aspnetdb.mdf 命名。

通过 <connectionStrings> 部分的 LocalSqlServer 声明指定 SQL Server Express Edition 文件的连接字符串。在 machine.config 文件的 <profile> 部分可以看到个性化引擎对该连接字符串的引用。<profile> 部分包含一个子部分，其中列出了可用于该个性化引擎的所有提供程序，如程序清单 18-25 所示。

程序清单 18-25 添加新的 SQL Server 数据提供程序

```

<configuration>
  <system.web>
    <profile>
      <providers>
        <add name="AspNetSqlProfileProvider"
            connectionStringName="LocalSqlServer" applicationName="/"
            type="System.Web.Profile.SqlProfileProvider, System.Web,
            Version=4.0.0.0, Culture=neutral,
            PublicKeyToken=b03f5f7f11d50a3a" />
      </providers>
    </profile>
  </system.web>
</configuration>

```

在这个例子中，使用 <add> 元素添加了一个提供程序。在这个元素中，connectionStringName 特性指向程序清单 18-22 中 connectionString 特性中声明的内容。

可以指定一个完全不同的 SQL Server Express Edition 文件，替代在 machine.config 文件中指定的文件。首先，创建一个连接字符串，指向新的 SQL Server Express Edition 文件，该文件是 aspnetdb.mdb 文件的模板化版本。此时，可以使用 connectionString 指向这个新文件。如果在 machine.config 文件中修改这些值，那么位于服务器上的所有 ASP.NET 应用程序都会使用这个指定的文件。但如果只修改 web.config 文件，那么只有使用 web.config 文件的应用程序才会使用这个新数据存储。服务器上的其他应用程序不会改变。

18.5.2 使用 SQL Server

使用 SQL Server Express Edition 文件处理个性化架构是相当容易的。但在处理性能和可靠性要求较高的大型应用程序时，就应使用 SQL Server 个性化提供程序和 SQL Server 7.0 或以后版本(包括 2000、2005、2008 和 2012)。如果这种数据存储可用，就应总是使用这个选项，而不是使用默认的 SQL Server Express Edition 文件。

如果通过 SQL Server Express Edition 文件使用 SQL Server 个性化提供程序，就会觉得这项工作很简单。个性化提供程序的工作是一流的，不需要任何设置或配置。但使用 SQL Server 个性化提供程序和 SQL Server 的全功能版本就是另一番光景。虽然这并不困难，但在使用前必须建立和配置 SQL Server。

ASP.NET 4.5 提供了为个性化架构建立和配置 SQL Server 的两种方式。一种方式是使用 ASP.NET SQL Server 设置向导,另一种方式是运行 .NET Framework 4.5 提供的一些 SQL Server 脚本。



有关 ASP.NET SQL Server 设置向导的论述,详见第 14 章。

要使用设置向导为 ASP.NET 4.5 个性化功能建立 SQL Server,必须在 Visual Studio 2012 Command Prompt 中启动 `aspnet_regsql.exe`,打开该工具。在命令提示行中输入 `aspnet_regsql.exe`,打开 ASP.NET SQL Server 设置向导的 GUI。如果按照设置向导的步骤进行操作,就可以为许多 ASP.NET 系统建立 SQL Server 实例,包括个性化系统。

1. 使用 SQL 脚本安装个性化功能

为个性化架构建立和配置 SQL Server 的另一个选项是利用这些工具和向导使用的 SQL 脚本。查看 `C:\WINDOWS\Microsoft .NET\Framework\v4.0.xxxx\`,从中可以发现安装和移除脚本——`InstallPersonalization.sql` 和 `UninstallPersonalization.sql`。运行这些脚本可以为数据库提供运行个性化架构所需要的表。在运行个性化脚本(或其他新的 ASP.NET 系统脚本)之前,必须先运行 `InstallCommon.sql` 脚本。

2. 为 SQL Server 配置提供程序

现在我们已经为使用个性化系统建立了 SQL Server 数据库,下一步是重新定义个性化提供程序,使之使用这个实例,而不是使用默认的 SQL Server Express Edition 文件。

在应用程序的 `web.config` 文件中完成这项任务。这里要配置提供程序,把这个提供程序实例定义为要使用的提供程序。程序清单 18-26 显示了在 `web.config` 文件的 `<profile>` 部分添加的内容。

程序清单 18-26 将 `SqlProfileProvider` 连接到 SQL Server

```
<configuration>
  <connectionStrings>
    <add name="LocalSql2012Server"
      connectionString="data source=127.0.0.1;Integrated Security=SSPI" />
  </connectionStrings>
  <profile defaultProvider="AspNetSql2012ProfileProvider">
    <providers>
      <clear />
      <add name="AspNetSql2012ProfileProvider"
        connectionStringName="LocalSql2012Server" applicationName="/"
        type="System.Web.Profile.SqlProfileProvider, System.Web,
          Version=4.0.0.0, Culture=neutral,
          PublicKeyToken=b03f5f7f11d50a3a" />
    </providers>
    <properties>
      <add name="FirstName" />
      <add name="LastName" />
      <add name="LastVisited" />
      <add name="Age" />
      <add name="Member" />
    </properties>
  </profile>
</configuration>
```

```

    </properties>
  </profile>
</configuration>

```

对这个配置定义的唯一修改是使用 `defaultProvider` 特性, 给它指定值, 即要使用的提供程序的名称, 这里是新创建的 SQL Server 提供程序 `AspNetSql2012ProfileProvider`。还可以通过修改 `<profile>` 元素, 对 `machine.config` 文件进行这样的修改, 如程序清单 18-27 所示。

程序清单 18-27 在 `machine.config` 文件中把 SQL Server 用作提供程序

```

<configuration>
  <system.web>
    ...
    <profile enabled="true" defaultProvider="AspNetSql2012ProfileProvider">
    ...
    </profile>
    ...
  </system.web>
</configuration>

```

此修改将迫使服务器上的所有应用程序都使用这个新的 SQL Server 提供程序, 而不是使用默认的 SQL Server 提供程序(除非在应用程序的 `web.config` 文件中重写这条命令)。

18.5.3 使用多个提供程序

不仅可以使使用单个数据存储或提供程序, 还可以使用任意多个提供程序。甚至可以为每个定义好的属性指定个性化提供程序。也就是说, 可以给大多数属性使用默认提供程序, 还可以让几个属性使用完全不同的提供程序, 如程序清单 18-28 所示。

程序清单 18-28 使用不同的提供程序

```

<configuration>
  <system.web>
    <profile
      defaultProvider="AspNetSqlProvider">
      <properties>
        <add name="FirstName" />
        <add name="LastName" />
        <add name="LastVisited" />
        <add name="Age" />
        <add name="Member" provider="AspNetSql2012ProfileProvider" />
      </properties>
    </profile>
  </system.web>
</configuration>

```

在这个例子中, 指定默认提供程序 `AspNetSqlProvider`。除非指定另一个提供程序, 否则就使用这个提供程序。改变这个设置的唯一属性是 `Member`。`Member` 属性使用一个完全不同的个性化提供程序。这里, 该属性通过 `<add>` 元素的 `provider` 特性使用 Access 提供程序(`AspNetSql2012ProfileProvider`)。通过这个特性可以为每个已定义属性指定特定的提供程序。

18.5.4 使用通用提供程序

ASP.NET 4.5 的配置功能与 SQL Server 关联在一起。准确地说,是有此依赖关系的配置提供程序,为了使 API 更灵活、更容易用于其他数据库系统,微软提供了所谓的通用提供程序。可惜,它们不是 ASP.NET 标准发布包的一部分,但可以在 NuGet 上用作独立的包。包名是 Microsoft.AspNet.Providers(注意该包的旧版称为 System.Web.Providers),NuGet 包的首页是 <http://nuget.org/packages/Microsoft.AspNet.Providers>。在 Visual Studio 中启动 NuGet Package Manager(使用 Tools | Library Package Manager | Package Manager Console)时,可以执行如下命令,把包安装到当前项目中:

```
Install-Package Microsoft.AspNet.Providers
```

图 18-6 显示了安装的典型控制台输出。之后, System.Web.Providers.dll 程序集就是项目的一部分了。

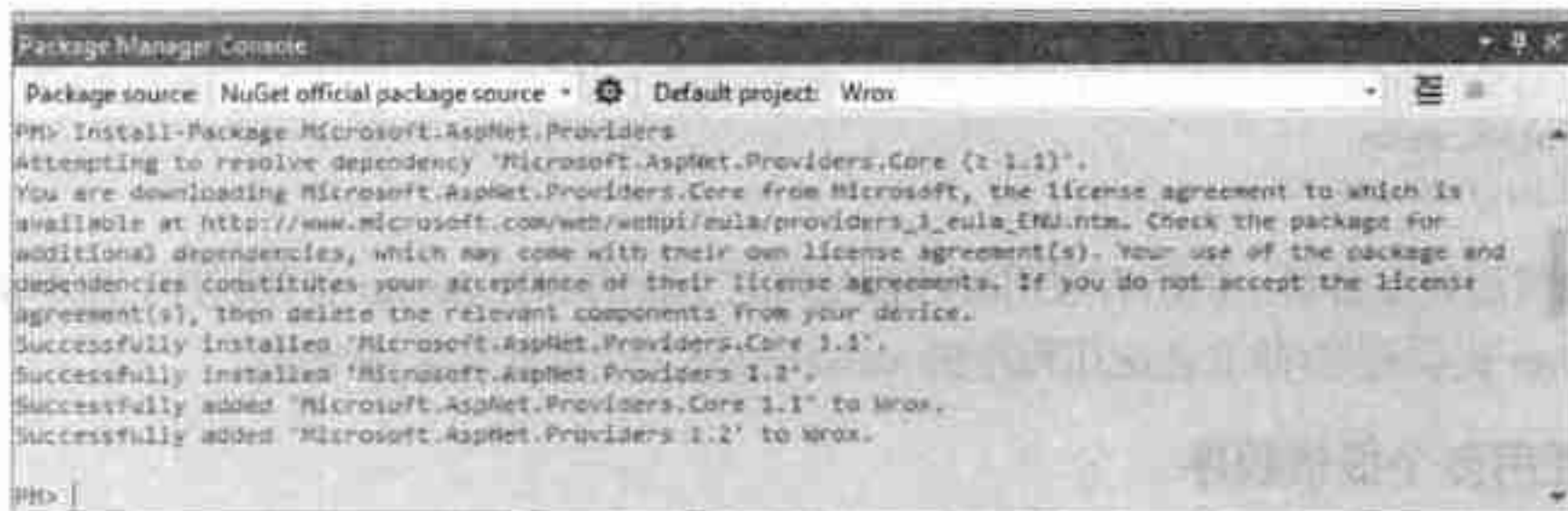


图 18-6

为了实际使用新的程序集,只需通过 web.config 配置 ASP.NET。技巧是使用 System.Web.Providers 程序集中定义的提供程序,这样 ASP.NET 就可以选择数据,处理支持的数据库了,支持的数据库列表目前关注的是微软的 Azure 云产品,但以后会添加更多的数据库。程序清单 18-29 是示例配置。

程序清单 18-29 在 web.config 中使用通用提供程序

```
<configuration>
  <connectionstrings>
    <add connectionstring="..." name="DefaultConnection" />
  </connectionstrings>
  <system.web>
    <profile defaultprovider="DefaultProfileProvider">
      <providers>
        <add name="DefaultProfileProvider" applicationname="/"
              type="System.Web.Providers.DefaultProfileProvider"
              connectionStringname="DefaultConnection" />
      </providers>
    </profile>
  </system.web>
</configuration>
```

18.6 管理应用程序的配置信息

在建立使用配置信息的 ASP.NET 应用程序时,我们将很快意识到,需要一种方式在应用程序的

生命周期中管理所有的配置信息。在 ASP.NET MMC 管理单元或 ASP.NET Web Site Administration Tool 中, 都不允许删除指定用户的配置信息, 甚至不能清除一段时间内未激活的所有用户的配置信息。

ASP.NET 4.5 提供了管理应用程序存储的配置信息的方式, 即使用 .NET 中可用的类 `ProfileManager`。使用 `ProfileManager` 类可以建立管理功能, 全面管理应用程序存储的配置信息。该类可以访问相应的属性值, 例如个性化系统使用的提供程序名或当前的应用程序名, 还可以使用 `ProfileManager` 类的许多方法检索与用户配置信息相关的其他各类信息。通过 `ProfileManager` 类, 可以对已存储的配置信息执行操作, 例如清理旧配置信息数据库。

18.6.1 ProfileManager 类的属性

`ProfileManager` 类的属性如表 18-1 所示。

表 18-1

属 性	说 明
<code>ApplicationName</code>	获取或设置所操作的应用程序名
<code>AutomaticSaveEnabled</code>	获取或设置一个布尔值, 表示是否在页面执行的最后存储配置信息
<code>Enabled</code>	获取或设置一个布尔值, 表示应用程序是否能使用个性化系统
<code>Provider</code>	获取用于个性化系统的提供程序名
<code>Providers</code>	获取用于 ASP.NET 应用程序的所有提供程序的集合

这些属性包含一些与个性化系统有关的信息以及可用于个性化系统的提供程序, 可以把这些属性集成到我们建立的管理系统中。下面介绍 `ProfileManager` 类可用的方法。

18.6.2 ProfileManager 类的方法

`ProfileManager` 类有许多方法, 有助于管理应用程序的用户配置信息。这些方法如表 18-2 所示。

表 18-2

方 法	说 明
<code>DeleteInactiveProfiles</code>	可以删除过了指定时间仍然一次也没有激活的配置信息
<code>DeleteProfile</code>	可以删除指定的配置信息
<code>DeleteProfiles</code>	可以删除配置信息的集合
<code>FindInactiveProfilesByUserName</code>	根据指定的日期查找指定用户名的所有未激活的配置信息
<code>FindProfilesByUserName</code>	查找指定用户名的所有配置信息
<code>GetAllInactiveProfiles</code>	获取过了指定日期仍然没有激活的配置信息
<code>GetAllProfiles</code>	获取所有配置信息的集合
<code>GetNumberOfInactiveProfiles</code>	获取过了指定日期仍然没有激活的配置信息数
<code>GetNumberOfProfiles</code>	获取系统中的配置信息总数

从这个方法列表中可以看出, 可以执行许多操作来管理存储在数据库中的配置信息。

下面为 ASP.NET 应用程序建立配置信息管理页面, 在这个例子中将其建立为 ASP.NET 页面,

也可以将其简单构建为控制台应用程序。

18.6.3 建立配置信息管理页面

要为应用程序建立简单的配置信息管理页面，可以用程序清单 18-30 中的代码，在应用程序中创建一个 ASP.NET 页面，这个页面将用于管理存储在该应用程序的数据库中的配置信息。

这个页面包含许多控件，但最重要的控件是 DropDownList，它包含数据库中具有配置信息的所有实体的用户名。根据对应用程序执行的操作，你可能会多次看到相同的用户名，因为一个用户可以在数据库中有多个配置信息。

使用 DropDownList 控件可以选择一个用户，查看他在数据库中存储的配置信息。在这个页面上，还可以删除该用户的配置信息。实际上可以对 ProfileManager 类执行许多操作，但这个例子只演示了一些非常基本的操作。

配置信息管理页面的代码如程序清单 18-30 所示。

程序清单 18-30 ProfileManager.aspx 页面

```
<%@ Page Language="C#" %>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        if (DropDownList1.Items.Count == 0)
        {
            WriteDropDownList();
            WriteUserOutput();
        }
    }
    protected void DeleteButton_Click(object sender, EventArgs e)
    {
        ProfileManager.DeleteProfile(DropDownList1.Text.ToString());
        DropDownList1.Items.Clear();
        WriteDropDownList();
        WriteUserOutput();
    }
    protected void SelectButton_Click(object sender, EventArgs e)
    {
        WriteUserOutput();
    }
    protected void WriteUserOutput()
    {
        int outputInt;
        ProfileInfoCollection pic = new ProfileInfoCollection();
        if (DropDownList1.Text != "") {
            pic = ProfileManager.FindProfilesByUserName
                (ProfileAuthenticationOption.All,
                 DropDownList1.Text.ToString(), 0, 1, out outputInt);
        }
        DetailsView1.DataSource = pic;
        DetailsView1.DataBind();
    }
    protected void WriteDropDownList()
    {
```



```

int outputInt;
ProfileInfoCollection pic = ProfileManager.Provider.GetAllProfiles
    (ProfileAuthenticationOption.All, 0, 10000, out outputInt);
foreach(ProfileInfo proInfo in pic)
{
    ListItem li = new ListItem();
    li.Text = proInfo.UserName.ToString();
    DropDownList1.Items.Add(li);
}
Label1.Text = outputInt.ToString();
}
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>ProfileAdmin Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <b>Profile Manager<br />
        </b>
        <br />
        Total number of users in system:
        <asp:Label ID="Label1" runat="server"></asp:Label><br />
        &nbsp;<br />
        <asp:DropDownList ID="DropDownList1" runat="server">
        </asp:DropDownList>&nbsp;<br />
        <asp:Button ID="SelectButton" runat="server"
            OnClick="SelectButton_Click"
            Text="Get User Profile Information" /><br />
        <br />
        <asp:DetailsView ID="DetailsView1" runat="server" CellPadding="4"
            ForeColor="#333333" GridLines="None"
            Height="50px">
            <FooterStyle BackColor="#1C5E55" Font-Bold="True" ForeColor="White" />
            <EditRowStyle BackColor="#7C6F57" />
            <PagerStyle BackColor="#666666" ForeColor="White"
                HorizontalAlign="Center" />
            <HeaderStyle BackColor="#1C5E55" Font-Bold="True" ForeColor="White" />
            <AlternatingRowStyle BackColor="White" />
            <CommandRowStyle BackColor="#C5BBAF" Font-Bold="True" />
            <RowStyle BackColor="#E3EAEB" />
            <FieldHeaderStyle BackColor="#D0D0D0" Font-Bold="True" />
        </asp:DetailsView>
        <br />
        <asp:Button ID="DeleteButton" runat="server"
            Text="Delete Selected User's Profile Information"
            OnClick="DeleteButton_Click" />
    </div>
    </form>
</body>
</html>

```

18.6.4 研究配置信息管理页面的代码

在配置信息管理页面的代码中, 注意 `ProfileManager` 类用于执行两个不同的操作。

首先, 使用 `ProfileManager` 类的 `GetAllProfiles` 方法填充页面上的 `DropDownList` 控件。这个方法的构造代码如下:

```
GetAllProfiles(
    authenticationOption,
    pageIndex,
    pageSize,
    totalRecords)
```

`GetAllProfiles` 方法有几个参数。第一个参数允许用户指定是为系统中的所有配置信息使用这个方法, 还是只为系统中包含的匿名用户或已验证用户的配置信息使用该方法。本例使用这个方法提取出所有的配置信息。利用 `ProfileAuthenticationOption` 枚举完成该操作。`GetAllProfiles` 方法的另外两个参数要求指定页面索引和要从数据库中提取的记录数。这里没有提取所有的记录(因为要考虑所提取数据的字节数), 而是指定提取第一个数据页面(使用索引 0), 这个页面包含前 10 000 条记录(这基本上是本应用程序中的所有记录)。`GetAllProfiles` 方法的最后一个参数是要提取的总记录数, 可用于应用程序的任何位置或用于遍历记录。配置选项管理页面在 `Label1` 服务器控件中显示总记录数。

`GetAllProfiles` 方法返回一个 `ProfileInfoCollection` 对象, 它是 `ProfileInfo` 对象的集合。遍历 `ProfileInfoCollection` 中的所有 `ProfileInfo` 对象, 可以提取出某用户配置信息中的一些重要属性。本例仅使用 `ProfileInfo` 对象的 `UserName` 属性填充页面上的 `DropDownList` 控件。

终端用户从下拉列表中选择用户时, 就要使用 `FindProfilesByUserName` 方法显示所选用户的配置信息, 这个方法也返回一个 `ProfileInfoCollection` 对象。

要删除 `DropDownList` 控件中所选用户的配置信息, 只需使用 `DeleteProfile` 方法, 给它传送所选用户的姓名, 如下所示:

```
ProfileManager.DeleteProfile(DropDownList1.Text.ToString())
DropDownList1.Items.Clear()
WriteDropDownList()
WriteUserOutput()
```

从系统中删除配置信息后, 用户的姓名就不再出现在下拉列表中(因为重新绘制了 `DropDownList` 控件)。查看数据库的 `aspnet_Profile` 表, 会发现所选用户的配置信息被删除。但要注意该用户(即使是匿名用户)仍然存储在 `aspnet_Users` 表中。

如果不仅要删除用户的配置信息, 还要从 `aspnet_Users` 表中删除该用户, 就应调用 `Membership` 类的 `DeleteUser` 方法, 如下所示:

```
ProfileManager.DeleteProfile(DropDownList1.Text.ToString())
Membership.DeleteUser(DropDownList1.Text.ToString())
DropDownList1.Items.Clear()
WriteDropDownList()
WriteUserOutput()
```

上面的代码使用 `DeleteUser` 方法从 `aspnet_Users` 表中删除了所选的用户。使用 `DeleteUser` 方法的其他构造函数也可以完成相同的任务, 如下所示:

```

Membership.DeleteUser(DropDownList1.Text.ToString(), True)
DropDownList1.Items.Clear()
WriteDropDownList()
WriteUserOutput()

```

这个 DeleteUser 方法的第二个参数从 aspnetdb.mdf 数据库的所有表中删除了与该用户相关的所有数据。

18.6.5 运行配置信息管理页面

编译并运行配置信息管理页面，结果如图 18-7 所示。

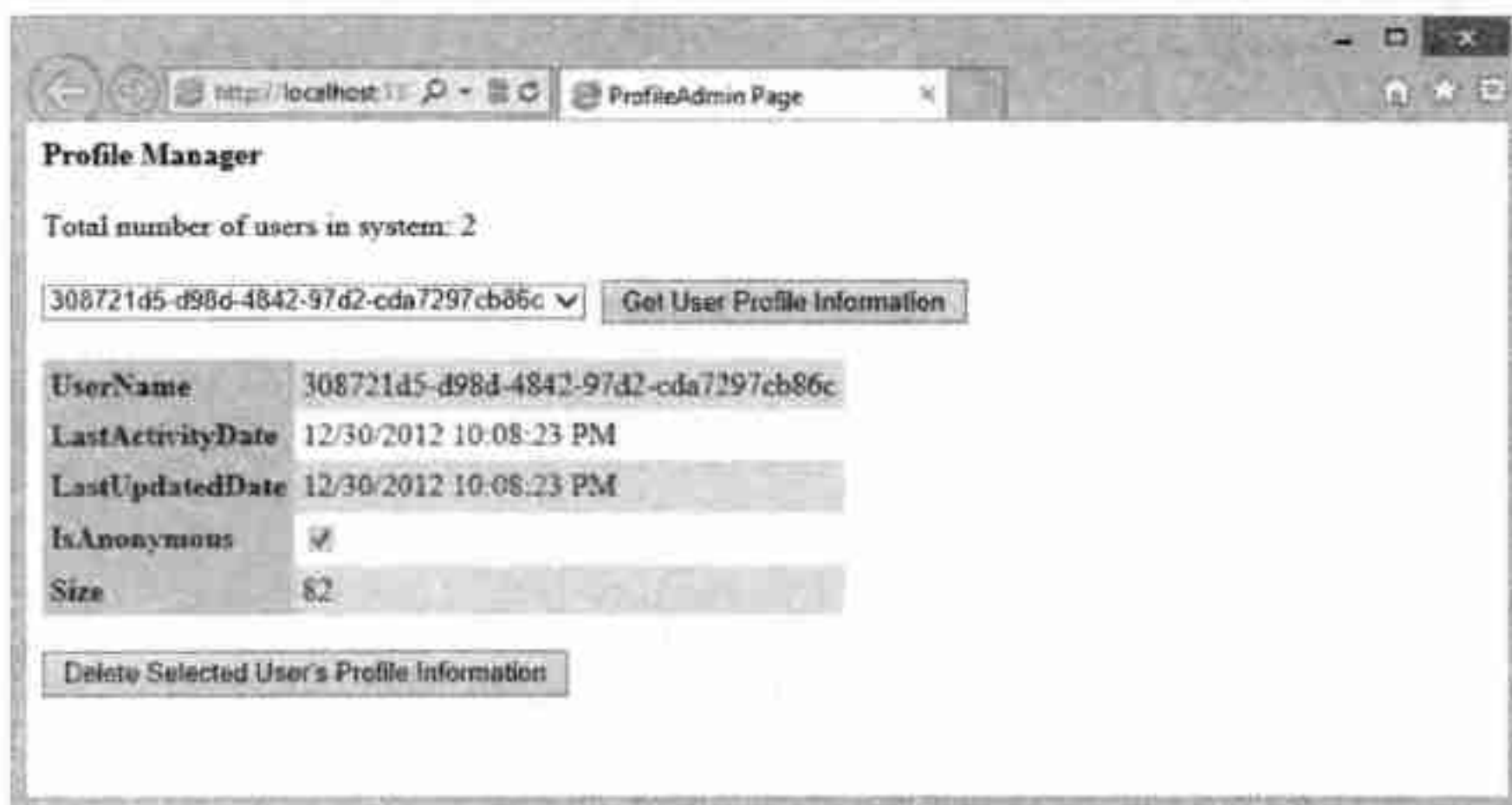


图 18-7

在这个屏幕上可以看到，该页面正在处理匿名用户(根据用户名的 GUID)，并且选中了 IsAnonymous 列。在这个页面上，单击相应的按钮就可以删除该用户的配置信息。

18.7 本章小结

ASP.NET 4.5 提供的个性化功能可以非常容易地使 Web 应用程序对每个终端用户(无论是已通过验证的用户还是匿名用户)来说都是独一无二的。这个新系统可以存储所有内容，包括 .NET Framework 提供的基本数据类型和定制类型。这个系统比使用 Session 或 Application 对象的功能更多，扩展性更好。可以通过 ASP.NET 中几个内置的个性化提供程序来存储数据。这些提供程序包括与 SQL Server Express Edition 文件或 SQL Server 连接的提供程序。还可以使用 ProfileManager 类管理系统的配置信息，这包括根据需要监控和删除配置信息。

第 19 章

成员资格和角色管理

本章要点

- 管理 ASP.NET 4.5 身份验证和授权
- 添加和管理角色
- 使用 ASP.NET 的 Login 服务器控件
- 创建定制的站点注册过程
- 回顾成员资格和角色 API 以及使用它们的控件
- 集成 OAuth/OpenID 验证功能

用户的身份验证和授权是许多 Web 站点和基于浏览器的应用程序的重要功能。传统上，处理 Windows 窗体应用程序(胖客户端)时，使用 Windows 集成验证；处理基于浏览器的应用程序(瘦客户端)时，使用窗体验证。

窗体验证可以获取未验证的请求，使用 HTTP 客户端重定向功能，把它们重定向到 HTML 表单。用户提供登录信息，然后提交表单。应用程序验证请求后，用户就会接收到 HTTP cookie，用于后续的请求。这种验证在许多方面都比较好，但需要开发人员建立所有的元素，甚至管理整个系统的后台机制。这对许多开发人员来说是一项艰巨的任务，在大多数情况下非常费时。

ASP.NET 4.5 包含身份验证和授权管理服务，用于处理需要访问 Web 页面或应用程序的用户的登录、身份验证、授权和管理。这个杰出的成员资格和角色管理服务是一个易于实现的架构，使用 SQL Server 作为后台数据存储。这个架构也包含一个 API，可以编程访问成员资格和角色管理的功能。另外，许多用于成员资格和角色管理的服务器控件简化了 Web 应用程序的创建，Web 应用程序将这些服务提供的功能组合在一起。

在介绍 ASP.NET 4.5 的成员资格和角色管理功能之前，先了解一下身份验证和授权的基本规则是很重要的。下面是对它们的简要介绍：

- **身份验证：**身份验证是确定用户身份的过程。在用户通过身份验证后，开发人员就可以确定已被验证的用户是否有权继续操作。如果没有进行身份验证过程，就不能给用户授予实体权限。在 ASP.NET 4.5 中，使用成员资格服务提供身份验证。

- **授权:** 授权过程确定通过身份验证的用户是否可以访问应用程序的所有部分、访问应用程序的特定部分或只访问应用程序提供的特定数据集。在对用户或组进行身份验证和授权时,可以根据用户类型或首选项来定制站点。在 ASP.NET 4.5 中,使用角色管理服务提供授权。

19.1 ASP.NET 4.5 的身份验证

ASP.NET 4.5 提供的成员资格管理服务可以验证用户的身份,以允许其访问一个页面或整个站点。ASP.NET 管理服务不仅为管理用户提供了一套 API,还提供了一些服务器控件来使用 API。这些服务器控件可以通过身份验证过程来处理终端用户。本章稍后将介绍这些控件的功能。

19.1.1 为成员资格建立 Web 站点

在使用 ASP.NET 4.5 提供的的安全控件之前,首先必须建立应用程序来使用成员资格服务。具体操作取决于如何处理提供的的安全架构。

从 Visual Studio 2010 开始,内置的 `SqlMembershipProvider` 实例用来存储应用程序中注册用户的详细信息。Visual Studio 2012 用于 ASP.NET MVC 的模板默认为新的 `DefaultMembershipProvider`,但 ASP.NET Web Forms 模板仍使用 `SqlMembershipProvider`。从 API 的角度来看,本章使用什么提供程序并不重要。

为了进行演示,本章的例子使用基于窗体的验证。对于这些例子,可以假定应用程序对公众开放,以用于注册和浏览。如果这是基于内联网的应用程序(即所有的用户都在内部的网络上),就可以使用 Windows 集成验证来验证用户的身份。



与第 18 章介绍的 ASP.NET 配置支持相同,也可以使用通用提供程序引入更多的功能。

ASP.NET 4.5 提供了一个数据提供程序模型,用于处理与多种底层数据存储交互所需要的具体管理功能。图 19-1 是 ASP.NET 4.5 成员资格服务的示意图。

从图 19-1 中可以看出,与 ASP.NET 的其他提供程序模型相同,成员资格提供程序也可以访问各种底层数据存储。在图 19-1 中可以看到内置的 SQL Server 数据存储。也可以建立自己的成员资格提供程序,以访问其他任何处理用户凭据的定制数据存储。在成员资格提供程序的上面是一组安全服务器控件,它们利用底层成员资格提供程序授予的访问权限在身份验证过程中管理用户。

从 System.Web.Security 到 System.Web.ApplicationServices

如果是从 ASP.NET 的以前版本迁移过来,就会注意到一些 ASP.NET 成员资格类型已经从 `System.Web.dll` 移到 `System.Web.ApplicationServices.dll`。在当前使用的网站模型中,你不会注意到这个改变,因为 ASP.NET 在编译过程中会自动使用新的 DLL。只需确保使用 ASP.NET 4.5 进行编译即可。

对于要迁移到 ASP.NET 4.5 的旧 Web 应用程序项目，需要在项目中添加对 System.Web.ApplicationServices.dll 的引用。http://www.asp.net/whitepapers/aspnet4/breaking-changes#0.1_Toc256770156 上的 breaking changes 白皮书列出了已删除类型的完整列表。

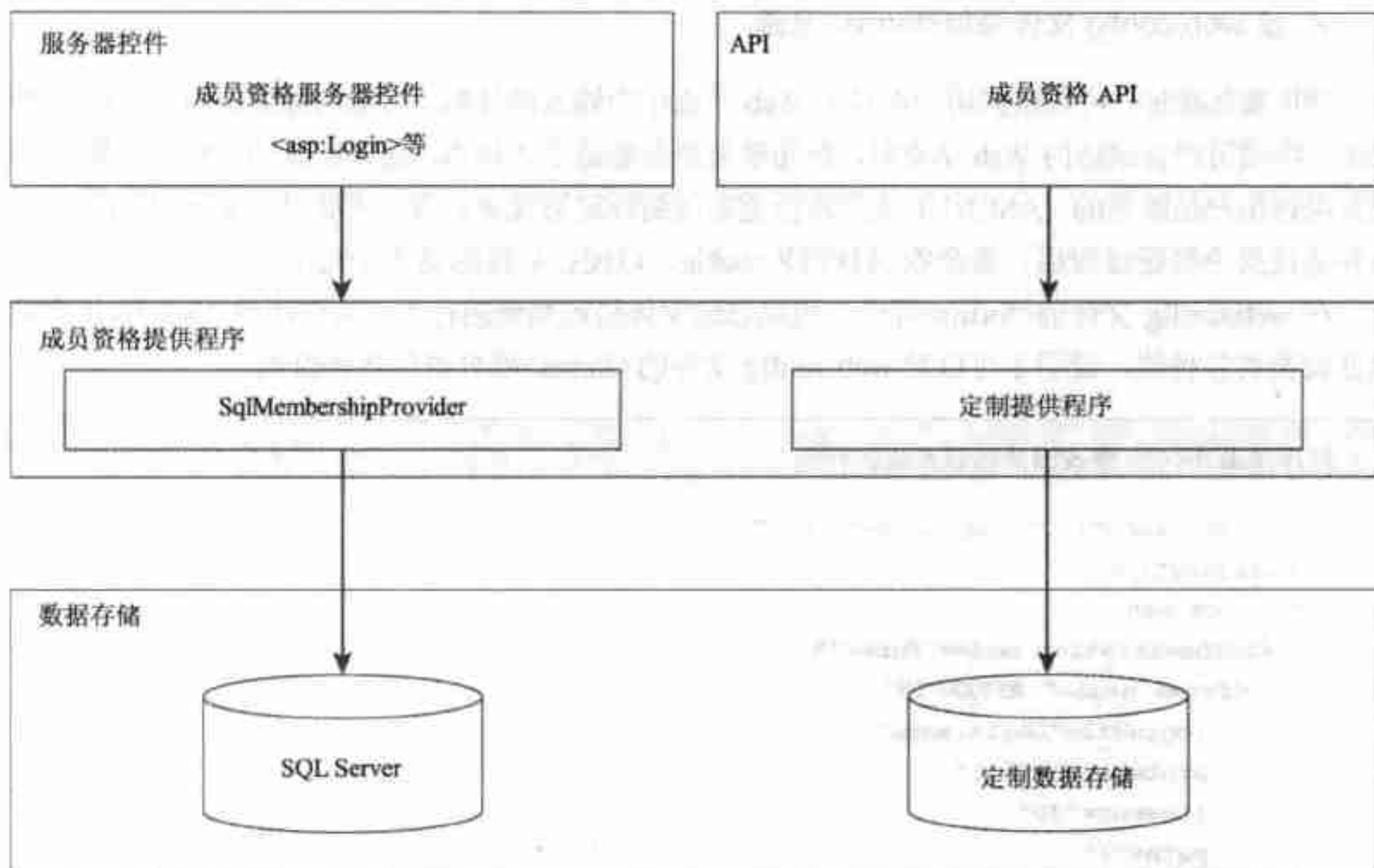


图 19-1

1. 给 web.config 文件添加<authentication>元素

为了在 Web 应用程序中给成员资格服务提供窗体验证支持，首先要在 web.config 文件中启用该功能。如果在应用程序中没有 web.config 文件，就创建该文件。接着在该文件中添加如程序清单 19-1 所示的代码部分。

程序清单 19-1 给 web.config 文件添加窗体验证

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.web>
    <authentication mode="Forms" />
  </system.web>
</configuration>
  
```

给 web.config 文件添加<authentication>元素后，就启用了使用 ASP.NET 4.5 提供的成员资格服务所需要的所有功能。要使用这个元素启用窗体验证，只需给 mode 特性指定值 Forms。这是一个窗体验证示例，但是 mode 特性的其他可选值有 Windows、Passport 和 None。

IIS 验证模式包含基本验证、摘要验证和 Windows 集成验证。Passport 验证指向微软提供的一项集中式服务，该服务为任意成员站点提供了登录和核心配置信息服务。使用 Passport 需要付费，微

软已经弃用。

本例中 mode 特性被设置为 Forms，因此可以进行下一步，把用户添加到数据存储中。此时，还可以对 web.config 文件进行一些修改，以改变窗体验证系统的行为。下面将介绍这些修改。

2. 给 web.config 文件添加<forms>元素

使用窗体验证，可以根据用户在基于 Web 的表单中输入的凭据，为他们提供访问站点或资源的权限。终端用户尝试访问 Web 站点时，使用匿名身份验证进入站点，这是默认的身份验证模式。如果发现该用户是匿名的，ASP.NET 就会将他重定向到特定的登录页面。终端用户输入相应的登录信息并通过身份验证过程后，就会收到 HTTP cookie，以便在后续的请求中使用。

在 web.config 文件的<forms>部分，可以修改窗体验证系统的行为。程序清单 19-2 列出了窗体验证设置的各种值，说明了可以对 web.config 文件的<forms>部分进行各种修改。

程序清单 19-2 修改窗体验证系统的行为

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.web>
    <authentication mode="Forms">
      <forms name=".ASPXAUTH"
        loginUrl="Login.aspx"
        protection="All"
        timeout="30"
        path="/"
        requireSSL="false"
        slidingExpiration="true"
        cookieless="UseDeviceProfile" />
    </authentication>
  </system.web>
</configuration>
```

可以根据需要设置它们，并且许多选项可以使用不同于此处显示的值。如前所述，这些值不是必需的。只需按照程序清单 19-1 中所示的配置进行设置，就可以立刻使用成员资格服务。

程序清单 19-2 中有一些有趣的设置。在 web.config 文件中添加<forms>元素，可以改变窗体验证系统的工作方式，但一定要把<forms>元素嵌套在<authentication>元素中。下面列出了<forms>元素的特性：

- name: 定义在终端用户通过身份验证后发送给他们的 cookie 的名称，这个 cookie 的名称默认是 ASPXAUTH。
- loginUrl: 如果没有找到有效的身份验证 cookie (ASPXAUTH 或其他 cookie)，就指定重定向 HTTP 请求的页面位置，让用户登录。该特性默认设置为 Login.aspx。
- protection: 指定应用于 cookie 的保护级别，cookie 在终端用户通过身份验证后存储在他的计算机上。可选值包括 All、None、Encryption 和 Validation，我们应总是使用 All。
- timeout: 定义 cookie 的有效期(分钟)，默认值是 30 分钟。
- path: 指定应用程序发出的 cookie 的路径。
- requireSSL: 定义是否要求凭据通过加密线路(SSL)发送，而不是通过明文发送。

- **slidingExpiration**: 指定 cookie 的超时是否在可变化的范围内。默认值是 true。这表示终端用户的 cookie 在应用程序的最后一个请求发出 30 分钟(或 timeout 特性指定的时间)后才过期。如果 slidingExpiration 特性的值设置为 false, cookie 就在第一个请求发出 30 分钟后过期。
- **cookieless**: 指定 ASP.NET 处理 cookie 的方式。其值可以是 UseDeviceProfile、UseCookies、AutoDetect 和 UseUri, 默认值是 UseDeviceProfile。这个值根据设备的用户代理检测是否使用了 cookie。UseCookies 要求所有的请求都在 cookie 中存储凭据。AutoDetect 会自动检测信息是存储在客户端的 cookie 中, 还是存放在 URI 中(通过首先发送测试 cookie 来进行检测)。最后, UseUri 强制 ASP.NET 把信息存储在所有实例的 URI 中。

启用窗体验证后, 下一步就是给 SQL Server Express Edition 数据存储 aspnetdb.mdf 添加用户。

19.1.2 添加用户

为了给成员资格服务添加用户, 可以把用户注册到 SQL Server Express Edition 数据存储中。第一个问题是: 该数据存储位于何处?



当然, SQL Server 有许多版本可以用于本书的例子。但是, 本章使用成员资格系统在创建用户时使用默认数据库。

成员资格系统的 SQL Server 提供程序可以使用专门为成员资格服务(和其他 ASP.NET 系统, 如角色管理系统)构建的 SQL Server Express Edition 文件。如果文件不存在, ASP.NET 会自动创建。要创建 aspnetdb.mdf 文件, 应使用 ASP.NET 服务器控件, 这些控件利用成员资格服务来强制创建这个文件。如果应用程序需要 aspnetdb.mdf 文件, ASP.NET 4.5 就会在 App_Data 文件夹中创建它。

有了数据存储后, 就可以开始给数据存储添加用户。

1. 使用 CreateUserWizard 服务器控件

CreateUserWizard 服务器控件可以与成员资格服务结合起来使用, 这个控件和本章涉及的其他控件都在 Visual Studio 2012 工具箱的 Login 部分。CreateUserWizard 控件可以把注册用户放在数据存储中, 以便以后进行检索。如果应用程序中的页面允许终端用户注册到站点上, 那么至少需要从用户那里获取登录名和密码, 并且把这些值存放在数据存储中。这样, 终端用户以后就可以访问这些项, 以使用成员资格系统登录到应用程序。

为了更便于完成任务, CreateUserWizard 控件能完全控制注册工作。程序清单 19-3 是使用该控件的一个简单示例。

程序清单 19-3 允许终端用户注册到站点

```
<%@ Page Language="C#" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Creating Users</title>
</head>
<body>
```



```

<form id="form1" runat="server">
  <asp:CreateUserWizard ID="CreateUserWizard1" runat="server"
    BorderWidth="1px" BorderColor="#FFDFAD" BorderStyle="Solid"
    BackColor="#FFFBD6" Font-Names="Verdana">
    <TitleTextStyle Font-Bold="True" BackColor="#990000"
      ForeColor="White"></TitleTextStyle>
  </asp:CreateUserWizard>
</form>
</body>
</html>

```

这个页面只使用了 CreateUserWizard 控件，该控件可以为 Web 应用程序注册终端用户。此处的 CreateUserWizard 控件应用了一些样式，但它可以非常简单，如下所示：

```

<asp:CreateUserWizard ID="CreateUserWizard1" runat="server">
</asp:CreateUserWizard>

```

运行代码，就会向终端用户显示一个表单，如图 19-2 所示。

图 19-2

这个屏幕截图显示了由终端用户填充的表单，其中包含用户信息，如用户名、密码、电子邮件地址和安全问答部分。单击 Create User 按钮，就可以把这些已定义的用户信息放在数据存储中。

通过该控件提供的用户名和密码可以让终端用户通过 Login 服务器控件登录到应用程序。该表单中还包含一个采用 CreateUser 服务器控件形式的 Confirm Password 文本框，用于确保密码拼写正确。包含电子邮件地址文本框是为了在终端用户忘记登录凭据时，把凭据通过电子邮件及时地发送给他。最后，安全问答部分用于在修改凭据或用户信息前验证终端用户的身份；或者在验证用户身份之后通过浏览器向用户提供凭据或用户信息。

终端用户单击这个表单中的 Create User 按钮后，需要确认存储信息，如图 19-3 所示。

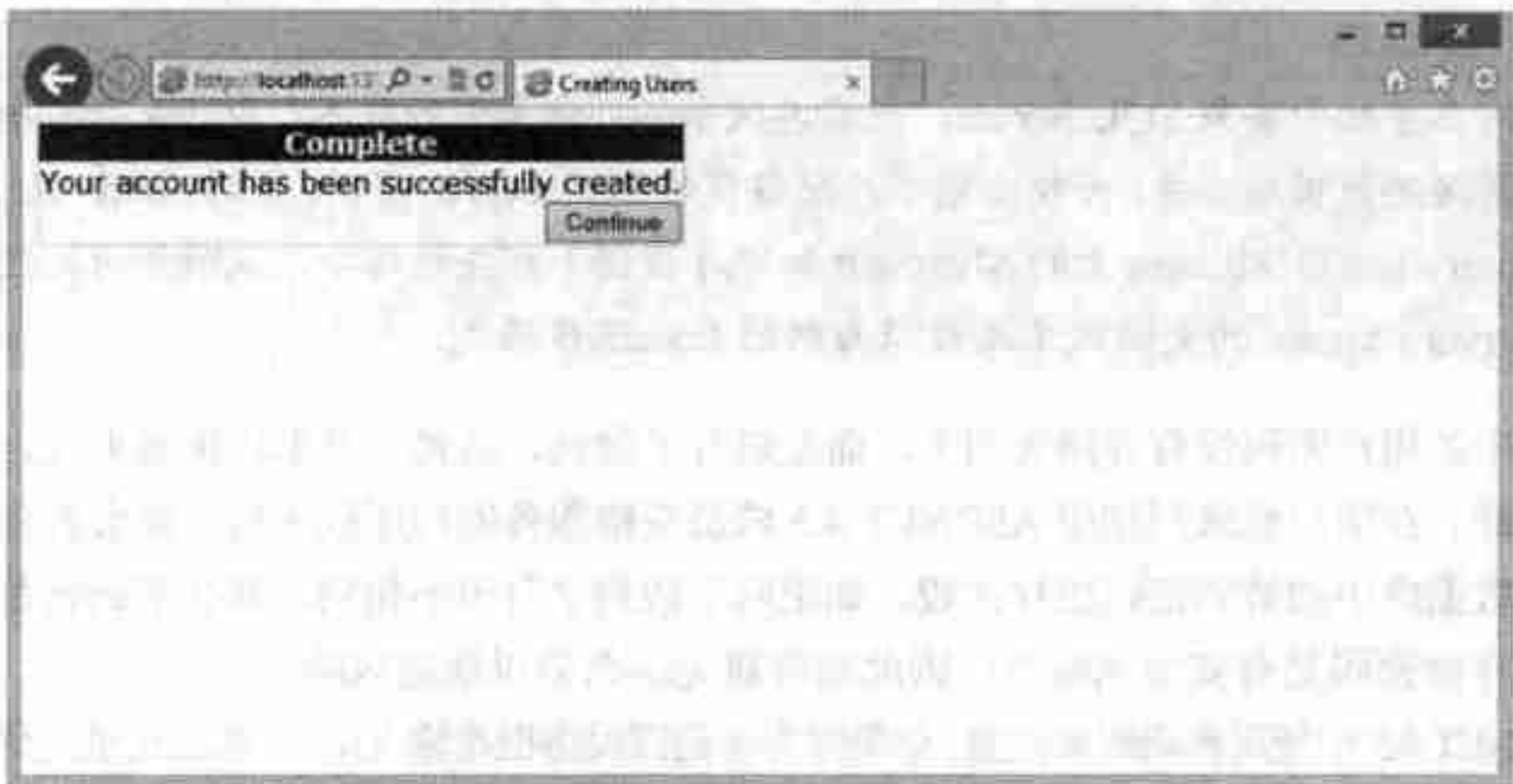


图 19-3

2. 查看用户的存储位置

使用 CreateUserWizard 控件给成员资格服务添加用户后，就要查看这些信息存储在什么地方。如果使用 Visual Studio 创建存储用户信息的 SQL Server Express Edition 文件，就会在上面的例子运行并完成图 19-3 中的表单时创建该文件。例子运行完成后，可以单击 Solution Explorer 中的 Refresh 按钮以查找 aspnetdb.mdf 文件，它位于项目的 App_Data 文件夹下。这个文件包含许多不同的表，这里只对 aspnet_Membership 表感兴趣。

在 Server Explorer 中右击 aspnet_Membership 表并选择 Show Table Data 命令，打开该表，输入的用户就会显示在系统中，如图 19-4 所示。

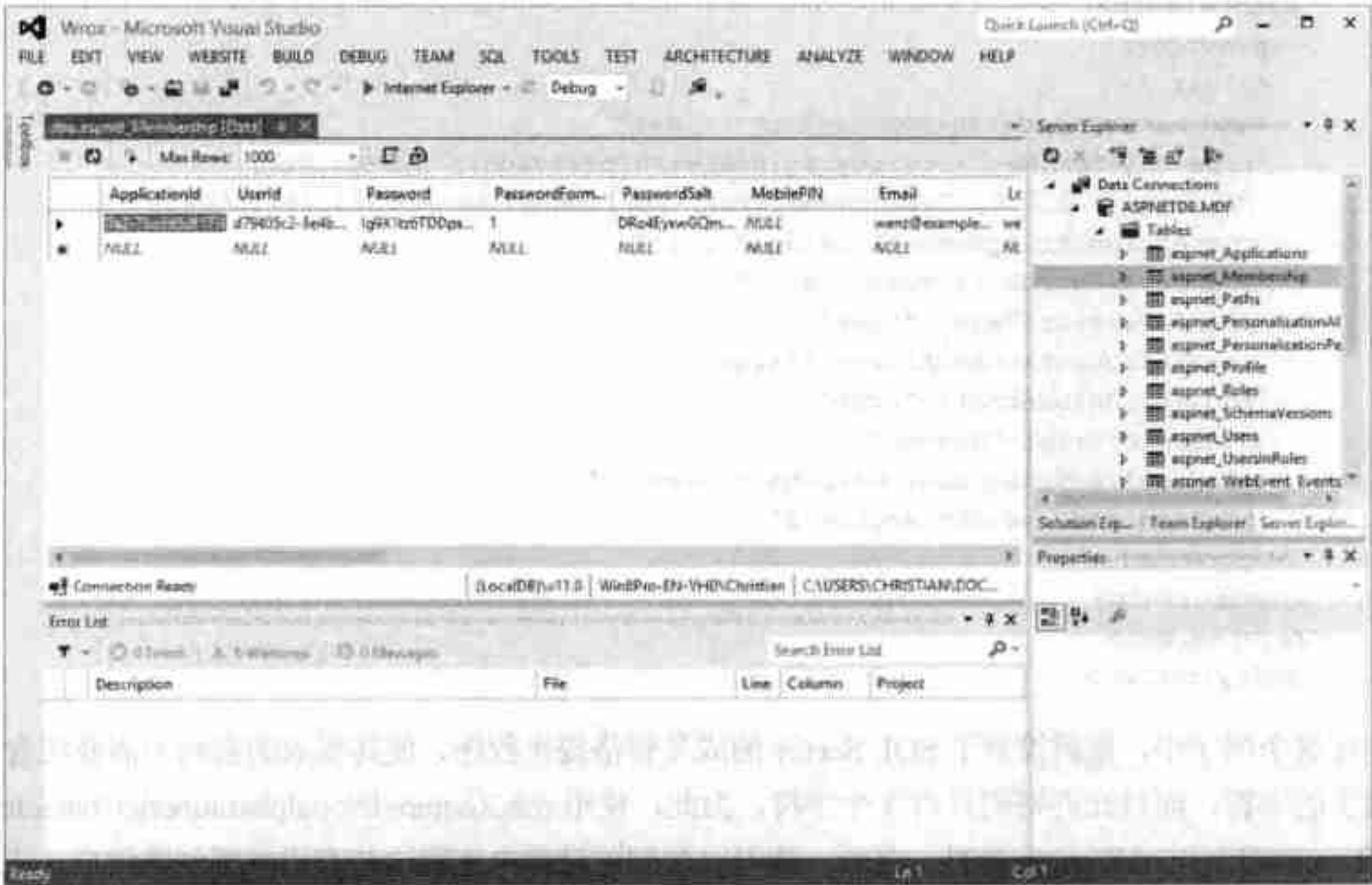


图 19-4

更新到 LocalDB

如果没有在系统中安装 SQL Server，或者其版本不匹配文件的版本，Server Explorer 就可能拒绝打开前面讲述的数据库文件，并指出它不匹配当前的 SQL Server 实例。此时，<http://msdn.microsoft.com/en-us/library/aa873188.aspx> 上的 Microsoft 知识库提供了相关的指令，说明如何把数据库文件升级到 SQL Server Express 的更新版本或升级为新的 LocalDB 格式。

这个表中的用户密码没有存储为明文，而是进行了散列，这是一种单向加密形式，从而使密码不容易被破解。在用户登录到使用 ASP.NET 4.5 成员资格服务的应用程序时，就会散列他的密码，并与存储在数据库中的散列密码进行比较。如果两个散列字符串不相同，那么密码就是不匹配的。以明文形式存储密码是有安全风险的，因此始终避免这么做以规避风险。

在 ASP.NET 4.5 中使用密码时要注意：如果因为密码错误而很难输入用户，那么可能是因为 ASP.NET 默认要求使用强密码。输入到系统中的所有密码都至少要有 7 个字符，其中至少包含一个非数字和字母的字符(如[,], !, @, #, \$)。这种组合的密码示例如下：

```
Sh@rd#2#cr@ck!
```

尽管这类密码比较安全，但像这样的密码有时很难记住。实际上，可以改变成员资格提供程序的工作方式，在 web.config 文件中重新设置它，使其不要求这么复杂的密码，如程序清单 19-4 所示。但注意这也可以使攻击者更容易猜出密码。

程序清单 19-4 在 web.config 文件中修改成员资格提供程序

```
<configuration>
  <system.web>
    <membership>
      <providers>
        <clear />
        <add name="AspNetSqlMembershipProvider"
          type="System.Web.Security.SqlMembershipProvider, System.Web,
            Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
          connectionStringName="LocalSqlServer"
          enablePasswordRetrieval="false"
          enablePasswordReset="true"
          requiresQuestionAndAnswer="false"
          requiresUniqueEmail="true"
          passwordFormat="Hashed"
          minRequiredNonalphanumericCharacters="0"
          minRequiredPasswordLength="3" />
      </providers>
    </membership>
  </system.web>
</configuration>
```

在这个例子中，重新设置了 SQL Server 的成员资格提供程序，使其要求的密码不需要包含非字母数字的字符，而且允许密码只有 3 个字符。为此，使用 minRequiredNonalphanumericCharacters 和 minRequiredPasswordLength 特性。之后，就可以在配置设置中使用这些密码规则创建用户。成员资格提供程序的修改详见本章后面的内容。

3. 使用 CreateUserWizard 控件

使用 CreateUserWizard 控件时, 应注意 ContinueButtonClick 和 CreatedUser 事件。成功创建用户后, 单击第二个页面上的 Continue 按钮, 就会触发 ContinueButtonClick 事件, 如程序清单 19-5 所示。注意程序清单 19-5 和 19-6 不能单独运行, 因此没有包含在本章的下载代码中。

程序清单 19-5 ContinueButtonClick 事件

```
protected void CreateUserWizard1_ContinueButtonClick(object sender, EventArgs e)
{
    Response.Redirect("Default.aspx");
}
```

在这个例子中, 通过 CreateUserWizard 控件提供的表单把用户添加到成员资格服务中后, 该用户单击 Continue 按钮就可以重定向到应用程序的另一个页面。使用简单的 Response.Redirect 语句可实现该操作。在使用这个事件时, 必须给<asp:CreateUserWizard>控件添加 OnContinueButtonClick="CreateUserWizard1_ContinueButtonClick"。

在数据存储中成功创建用户时, 就会触发 CreateUser 事件。这个事件的用法如程序清单 19-6 所示。

程序清单 19-6 CreateUser 事件

```
protected void CreateUserWizard1_CreatedUser(object sender, EventArgs e)
{
    // Code here
}
```

如果希望在用户注册到该服务中时执行其他操作, 就可以使用这个事件。

4. 在注册过程中合并个性化属性

在第 18 章对个性化功能的介绍中, 很容易使用 ASP.NET 4.5 中的个性化管理系统以及存储用户特定的信息。CreateUserWizard 控件提供的注册过程非常适合于从用户处获取这些信息以直接存储在个性化系统中。该获取过程可以合并到代码中。

如第 18 章所述, 首先是在应用程序的 web.config 文件中定义一些个性化信息, 如程序清单 19-7 所示。

程序清单 19-7 在 web.config 文件中创建个性化属性

```
<configuration>
  <system.web>
    <profile>
      <properties>
        <add name="FirstName" />
        <add name="LastName" />
        <add name="LastVisited" />
        <add name="Age" />
        <add name="Member" />
      </properties>
    </profile>
  </system.web>
</configuration>
```

```

    </system.web>
</configuration>

```

在 web.config 文件中定义了这些属性后,就可以使用它们在 ASP.NET 成员资格系统中创建用户。使用 CreateUserWizard 控件可以创建一个过程,该过程的第一步要求用户输入用户名和密码,第二步要求用户输入一些自定义的个性化信息。程序清单 19-8 显示了完成该任务的 CreateUserWizard 控件。

程序清单 19-8 将个性化属性用于 CreateUserWizard 控件

```

<%@ Page Language="C#" %>
<script runat="server">
    protected void CreateUserWizard1_CreatedUser(object sender, EventArgs e)
    {
        ProfileCommon pc = new ProfileCommon();
        pc.Initialize(CreateUserWizard1.UserName.ToString(), true);
        pc.FirstName = Firstname.Text;
        pc.LastName = Lastname.Text;
        pc.Age = Age.Text;
        pc.Save();
    }
</script>
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Creating Users with Personalization</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:CreateUserWizard ID="CreateUserWizard1" runat="server"
            BorderWidth="1px" BorderColor="#FFDFAD" BorderStyle="Solid"
            BackColor="#FFFBD6" Font-Names="Verdana"
            LoginCreatedUser="true" OnCreatedUser="CreateUserWizard1_CreatedUser" >
            <WizardSteps>
                <asp:WizardStep ID="WizardStep1" runat="server"
                    Title="Additional Information" StepType="Start">
                    <table width="100%"><tr><td>
                        Firstname: </td><td>
                            <asp:TextBox ID="Firstname" runat="server"></asp:TextBox>
                        </td></tr><tr><td>
                        Lastname: </td><td>
                            <asp:TextBox ID="Lastname" runat="server"></asp:TextBox>
                        </td></tr><tr><td>
                        Age: </td><td>
                            <asp:TextBox ID="Age" runat="server"></asp:TextBox>
                        </td></tr></table>
                </asp:WizardStep>
                <asp:CreateUserWizardStep runat="server"
                    Title="Sign Up for Your New Account">
                </asp:CreateUserWizardStep>
                <asp:CompleteWizardStep runat="server" Title="Complete">

```



```

        </asp:CompleteWizardStep>
    </WizardSteps>
    <StepStyle BorderColor="#FFDFAD" Font-Names="Verdana"
        BackColor="#FFFBD6" BorderStyle="Solid"
        BorderWidth="1px"></StepStyle>
    <TitleTextStyle Font-Bold="True" BackColor="#990000"
        ForeColor="White"></TitleTextStyle>
</asp:CreateUserWizard>
</form>
</body>
</html>

```

按照 CreateUserWizard 控件的默认实例的定义对标准注册过程进行这些修改后，注册系统就包含了使用 ProfileCommon 对象存储和提取属性的请求。然后使用 ProfileCommon.Initialize 方法初始化当前用户的属性值。接着通过 ProfileCommon 对象对配置属性进行强类型化的访问，以设置属性值。所有的值都设置完成后，使用 Save 方法结束该过程。

使用 ProfileCommon 类

如第 18 章所述，只能在网站模型中直接、方便地访问配置信息，而不能在 Web 应用程序模型中实现这个操作。因此，如果有一个网站，且在 web.config 中定义了如程序清单 19-7 所示的配置属性，就可以使用 ProfileCommon 类。如果使用了 Web 应用程序模型，就必须使用辅助类，如第 18 章所述。

可以在 CreateUserWizard 控件中使用<WizardSteps>元素定义定制的步骤。在这个元素中，可以利用所选的任何方式，构建一系列注册步骤。在<WizardSteps>部分，定义了 3 个步骤，如程序清单 19-8 所示。第一步是定制步骤，使用<asp:WizardStep>控件请求终端用户的个性化属性。在<asp:WizardStep>控件中，布置了一个表，创建了一个定制表单。

程序清单 19-8 中定义了另外两步：使用<asp:CreateUserWizardStep>控件创建用户；使用<asp:CompleteWizardStep>控件确认新用户的创建。这些步骤的顺序就是它们显示给终端用户的顺序。以希望的方式创建这些步骤后，就可以使用 CreateUserWizard 控件的 CreatedUser 事件存储定制属性：

```

protected void CreateUserWizard1_CreatedUser(object sender, EventArgs e)
{
    ProfileCommon pc = new ProfileCommon();
    pc.Initialize(CreateUserWizard1.UserName.ToString(), true);
    pc.FirstName = Firstname.Text;
    pc.LastName = Lastname.Text;
    pc.Age = Age.Text;
    pc.Save();
}

```

不仅可以在一个步骤中请求用户输入个人信息，还可以把这些项直接合并到<asp:CreateUserWizardStep>步骤中。一种简单的方式是切换到页面的设计视图，打开 CreateUserWizard 控件的智能标记。然后单击 Customize Create User Step 链接，如图 19-5 所示。

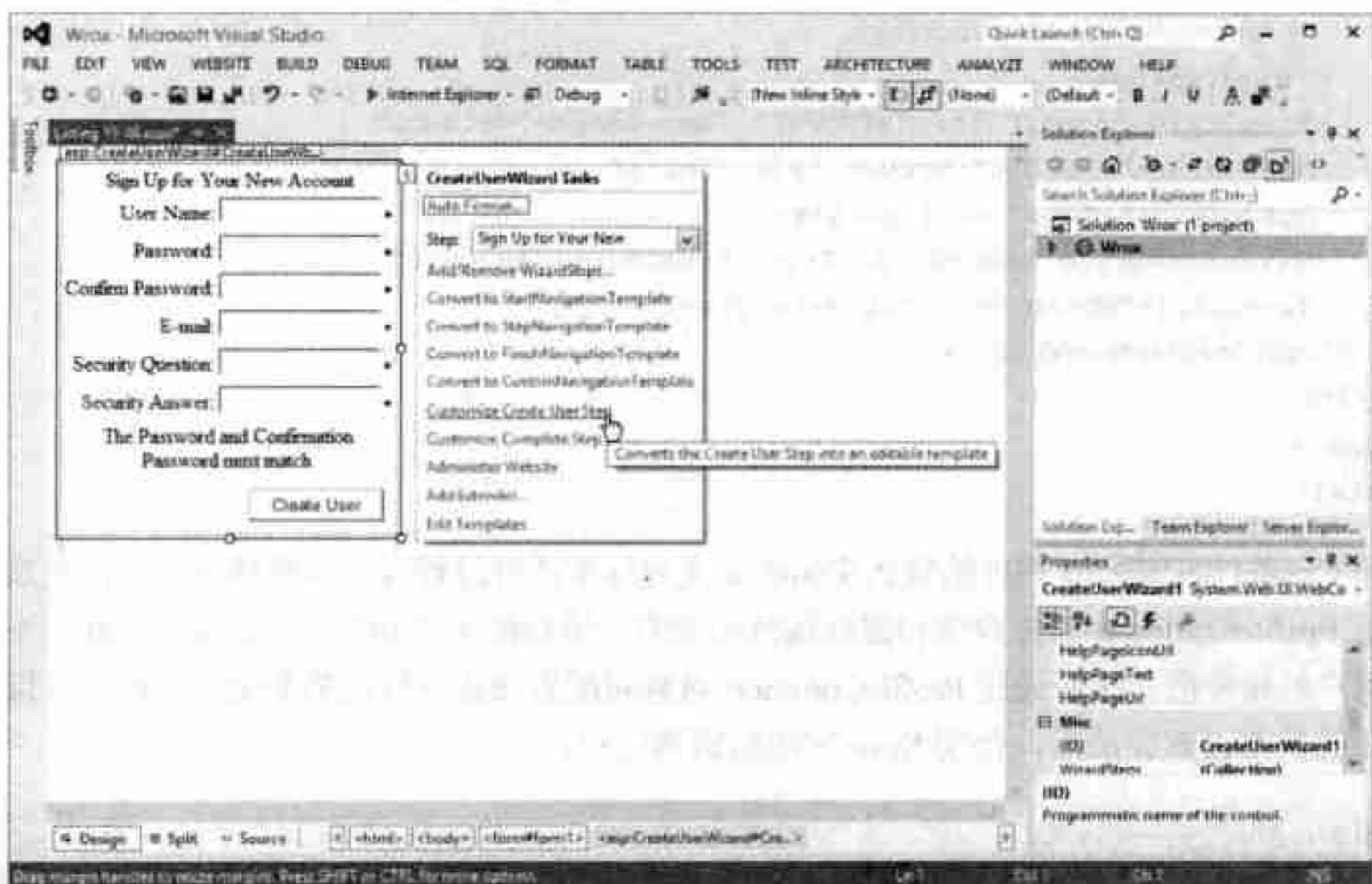


图 19-5

单击 **Customize Create User Step** 链接后，会在 `<asp:CreateUserWizardStep>` 控件的 `<ContentTemplate>` 部分详细列出这一步的内容。在 `<ContentTemplate>` 元素中，可以看到一个用于创建新用户的完全默认的表单。此时，可以自由修改该表单，添加自己的部分，请求终端用户输入其个人信息。在这个详细表单中，也可以删除项。例如，如果不想请求用户输入安全问答，就可以从表单中删除这两项(必须在成员资格提供程序定义中禁用问答请求)。通过修改这个默认表单，可以为终端用户完全定制注册过程，如图 19-6 所示。

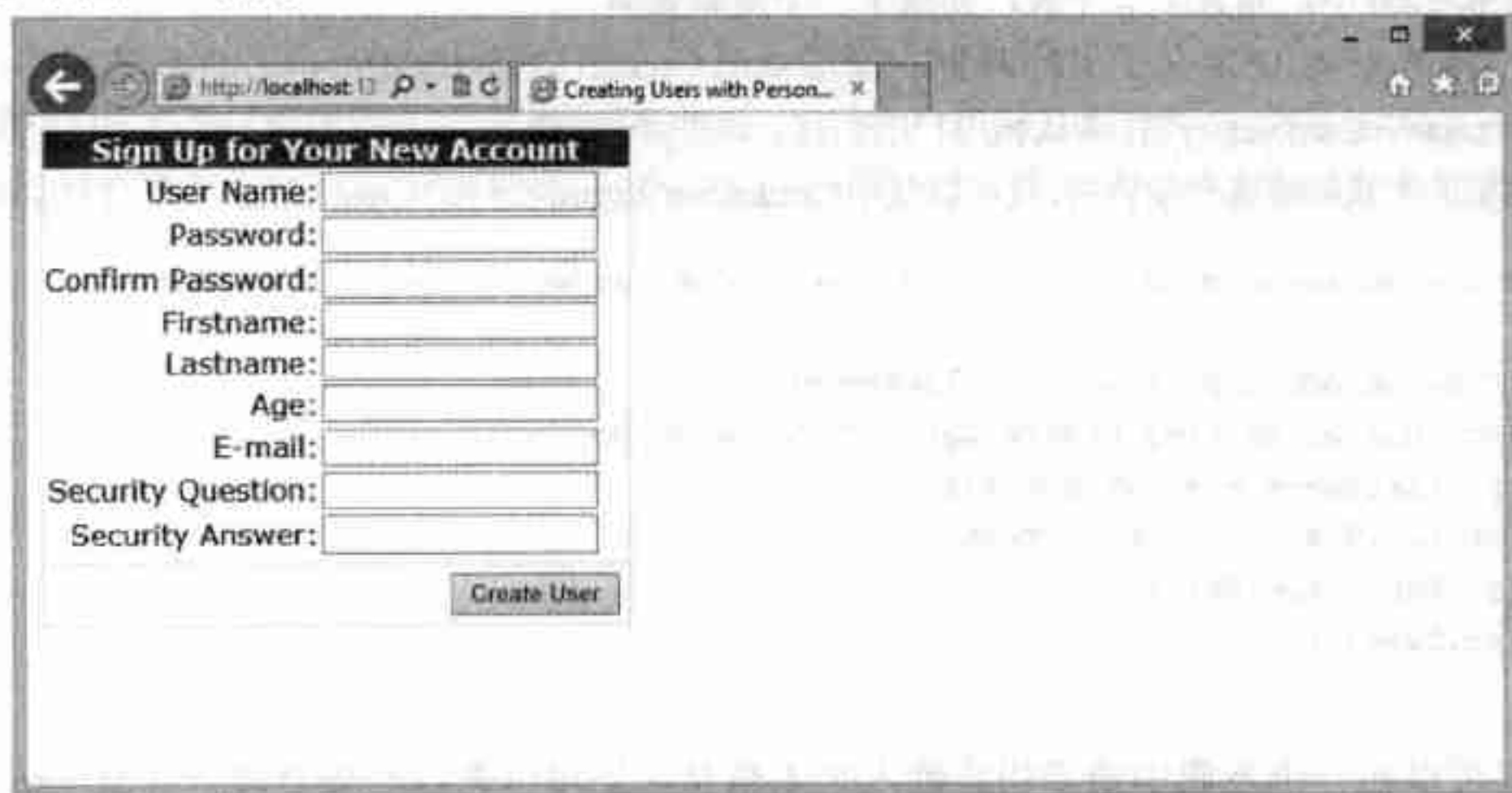


图 19-6

5. 以编程方式添加用户

不仅可以使服务器控件给成员资格服务注册或添加新用户，ASP.NET 4.5 还提供了

Membership API 来编程执行这项任务。该功能非常适合于为了将用户添加到服务中而创建自己的机制, 或者适合于修改使用 ASP.NET 1.0/1.1 创建的 Web 应用程序。

Membership API 包含用于将用户添加到服务中的 CreateUser 方法, 该方法有如下 4 种不同的签名:

```
Membership.CreateUser(username As String, password As String)
Membership.CreateUser(username As String, password As String,
email As String)
Membership.CreateUser(username As String, password As String,
email As String, passwordQuestion As String,
passwordAnswer As String, isApproved As Boolean,
ByRef status As System.Web.Security.MembershipCreateStatus)
Membership.CreateUser(username As String, password As String,
email As String, passwordQuestion As String,
passwordAnswer As String, isApproved As Boolean, providerUserKey As Object
ByRef status As System.Web.Security.MembershipCreateStatus)
```

可以使用这个方法创建用户。这个方法的优点是, 不需要创建 Membership 类的实例, 而是可以直接使用。CreateUser 方法的简单用法如程序清单 19-9 所示(不要运行这个程序清单, 而应继续阅读, 完成更多的准备工作)。

程序清单 19-9 以编程方式创建用户

```
<%@ Page Language="C#" %>
<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        try
        {
            Membership.CreateUser(TextBox1.Text.ToString(),
                TextBox2.Text);
            Label1.Text = "Successfully created user " + TextBox1.Text;
        }
        catch (MembershipCreateUserException ex)
        {
            Label1.Text = "Error: " + ex.ToString();
        }
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Creating a User</title>
</head>
<body>
    <form id="form1" runat="server">
        <h1>Create User</h1>
        <p>Username<br />
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        </p>
        <p>Password<br />
        <asp:TextBox ID="TextBox2" runat="server"
            TextMode="Password"></asp:TextBox>
        </p>
```



```

<p>
  <asp:Button ID="Button1" runat="server" Text="Create User"
    OnClick="Button1_Click" />
</p>
<p>
  <asp:Label ID="Label1" runat="server"></asp:Label>
</p>
</form>
</body>
</html>

```

使用 CreateUserWizard 控件或 Membership API 中的 CreateUser 方法，可以相当方便地为 Web 应用程序创建用户。

在这行代码中，如果在使用 CreateUser 方法创建用户时出现问题，就会抛出 MembershipCreateUserException 异常。在这个例子中，我们故意抛出了一个异常(因为配置要求提供安全问答)，该异常会写入屏幕的一个 Label 服务器控件。将异常写到屏幕的一个示例如下所示：

```

Error: System.Web.Security.MembershipCreateUserException: The password-answer
supplied is invalid. at System.Web.Security.Membership.CreateUser(String username,
String password, String email) at System.Web.Security.Membership.CreateUser(String
username, String password) at ASP.default_aspx.Button1_Click(Object sender,
EventArgs e) in c:\Documents and Settings\Christian\My Documents\Visual Studio
2012\WebSites\Wrox\Default.aspx:line 10

```

这些详细信息最好不要显示给终端用户，而应给他们返回一条比较简单的消息，其构造如下所示：

```
Label1.Text = "Error: " + ex.Message.ToString();
```

这会得到如下简单结果：

```
Error: The password-answer supplied is invalid.
```

还可以使用 MembershipCreateUserException 异常捕获特定的错误，并返回合适的消息，如程序清单 19-10 所示。

程序清单 19-10 捕获特定的 MembershipCreateUserException 值

```

<%@ Page Language="C#" %>
<script runat="server">
  protected void Button1_Click(object sender, EventArgs e)
  {
    try
    {
      Membership.CreateUser(TextBox1.Text, TextBox2.Text);
      Label1.Text = "Successfully created user " + TextBox1.Text;
    }
    catch (MembershipCreateUserException ex)
    {
      switch (ex.StatusCode)
      {
        case MembershipCreateStatus.DuplicateEmail:
          Label1.Text = "You have supplied a duplicate email address.";

```



```

        break;
    case MembershipCreateStatus.DuplicateUserName:
        Label1.Text = "You have supplied a duplicate username.";
        break;
    case MembershipCreateStatus.InvalidEmail:
        Label1.Text = "You have not supplied a proper email address.";
        break;
    default:
        Label1.Text = "ERROR: " + ex.Message.ToString();
        break;
    }
}
}
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Creating a User</title>
</head>
<body>
    <form id="form1" runat="server">
        <h1>Create User</h1>
        <p>Username<br />
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        </p>
        <p>Password<br />
        <asp:TextBox ID="TextBox2" runat="server"
            TextMode="Password"></asp:TextBox>
        </p>
        <p>
            <asp:Button ID="Button1" runat="server" Text="Create User"
                OnClick="Button1_Click" />
        </p>
        <p>
            <asp:Label ID="Label1" runat="server"></asp:Label>
        </p>
    </form>
</body>
</html>

```

在这个例子中，可以查找在 `CreateUser` 过程中发生的特定错误。这里的代码只查找了 3 个特定的项，但错误代码列表可以包括如下内容：

- `MembershipCreateStatus.DuplicateEmail`
- `MembershipCreateStatus.DuplicateProviderUserKey`
- `MembershipCreateStatus.DuplicateUserName`
- `MembershipCreateStatus.InvalidAnswer`
- `MembershipCreateStatus.InvalidEmail`
- `MembershipCreateStatus.InvalidPassword`
- `MembershipCreateStatus.InvalidProviderUserKey`
- `MembershipCreateStatus.InvalidQuestion`
- `MembershipCreateStatus.InvalidUserName`

- MembershipCreateStatus.ProviderError
- MembershipCreateStatus.Success
- MembershipCreateStatus.UserRejected

除了通过定义接下来的操作来给用户发送较好的错误报告之外，还可以使用这些事件执行任何需要的操作。

6. 修改将用户注册到应用程序的方式

我们要确定用户如何注册到应用程序中，以及所选的成员资格提供程序要求用户输入什么信息。在 machine.config 文件中建立默认的成员资格提供程序及其应用的设置。如果打开服务器上的 machine.config 文件，就会看到如程序清单 19-11 所示的代码。

程序清单 19-11 machine.config 文件中成员资格提供程序的设置

```
<membership>
  <providers>
    <add name="AspNetSqlMembershipProvider"
      type="System.Web.Security.SqlMembershipProvider, System.Web,
        Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
      connectionStringName="LocalSqlServer"
      enablePasswordRetrieval="false"
      enablePasswordReset="true"
      requiresQuestionAndAnswer="true"
      applicationName="/"
      requiresUniqueEmail="false"
      passwordFormat="Hashed"
      maxInvalidPasswordAttempts="5"
      minRequiredPasswordLength="7"
      minRequiredNonalphanumericCharacters="1"
      passwordAttemptWindow="10"
      passwordStrengthRegularExpression="" />
  </providers>
</membership>
```

machine.config 文件的这个部分显示了 ASP.NET 4.5 中的默认成员资格提供程序 AspNetSqlProvider。如果要添加其他成员资格提供程序并在服务器上使用，就应把它们添加到 machine.config 文件的 <membership> 部分。如果仅将这些提供程序用于某个应用程序实例，就可以把它们添加到应用程序的 web.config 文件中。

SqlMembershipProvider 定义的重要特性包括 enablePasswordRetrieval、enablePasswordReset、requiresQuestionAndAnswer、requiresUniqueEmail 和 PasswordFormat。表 19-1 定义了这些特性。

表 19-1

特 性	说 明
enablePasswordRetrieval	确定提供程序是否支持密码的获取。这个特性的值是布尔值，默认值是 False。将该特性设置为 False 时，不能检索密码，但可以使用新的随机密码替代
enablePasswordReset	确定提供程序是否支持密码的重置。这个特性的值是布尔值，默认值是 True
requiresQuestionAndAnswer	在创建用户时，指定提供程序是否需要问答组合。这个特性的值是布尔值，默认值是 False

(续表)

特 性	说 明
requiresUniqueEmail	在创建用户时, 定义提供程序是否需要指定唯一的电子邮件。这个特性的值是布尔值, 默认值是 False。将该特性设置为 True 时, 只有唯一的电子邮件地址可以输入到数据存储中。
passwordFormat	定义密码存储到数据存储中的格式, 值可以是 Hashed、Clear 和 Encrypted, 默认值是 Hashed。散列密码使用 SHA1 加密方式, 而加密的密码使用 Triple-DES 加密方式。

除了这些在 machine.config 文件中定义的项之外, 还可以在 web.config 文件中重新定义它们(这会重写 machine.config 文件中的设置)。

19.1.3 请求凭据

使用 ASP.NET 提供的成员资格服务允许用户访问 Web 应用程序之后, 就可以给这些用户提供登录到站点的方式。该操作不需要我们做什么工作。在介绍允许用户访问应用程序的控件之前, 应对 web.config 文件进行一些修改。

1. 使用<authorization>元素禁用访问功能

在添加<authorization>和<forms>元素(程序清单 19-1 和 19-2)以对 web.config 文件进行修改之后, 浏览应用程序中任意页面的每个用户就可以访问该 Web 应用程序。为了禁用公开访问功能, 必须拒绝未经验证的用户访问站点的页面。

拒绝未经验证的用户访问站点的代码如程序清单 19-12 所示。

程序清单 19-12 拒绝未经验证的用户

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.web>
    <authentication mode="Forms" />
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</configuration>
```

使用<authorization>和<deny>元素, 可以拒绝指定的用户访问 Web 应用程序, 或者(如本例所示)简单地拒绝所有未经验证的用户(这就是问号的含义)。

因为除验证用户之外的所有人都会被拒绝访问站点, 所以我们希望使应用程序的访问者能轻松地变成验证用户。为此, 需要使用 Login 服务器控件。

2. 使用 Login 服务器控件

Login 服务器控件允许未经验证的用户提供能在某种数据存储中通过验证的登录凭据, 把他们转变为验证用户。在前面的例子中, 把 SQL Server Express Edition 用作数据存储, 但是也可以方便地使用 SQL Server 的全功能版本。

使用 Login 控件的第一步是创建标题为 Login.aspx 的新 Web 页面, 这是重定向未验证用户以获

得其凭据的默认页面。在 web.config 文件中修改<forms>元素的 returnUrl 特性值, 就可以改变这种行为。

Login.aspx 页面只需要一个<asp:Login>控件, 就可以获取把终端用户变成验证用户所需的所有信息, 如程序清单 19-13 所示(本章下载代码中的 Login.aspx)。

程序清单 19-13 使用 Login 控件为终端用户提供登录

```
<%@ Page Language="C#" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Login Page</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:Login ID="Login1" runat="server">
    </asp:Login>
  </form>
</body>
</html>
```

在这里建立的页面中, 如果未验证用户单击了应用程序中的另一个页面, 就会将它重定向到 Login.aspx 页面。可以在浏览器的地址栏中看到 ASP.NET 跟踪 URL 中的位置的情况:

```
http://localhost:1315/Membership/Login.aspx?ReturnUrl=%2fMembership%2fDefault.aspx
```

使用 Login 控件的登录页面如图 19-7 所示。

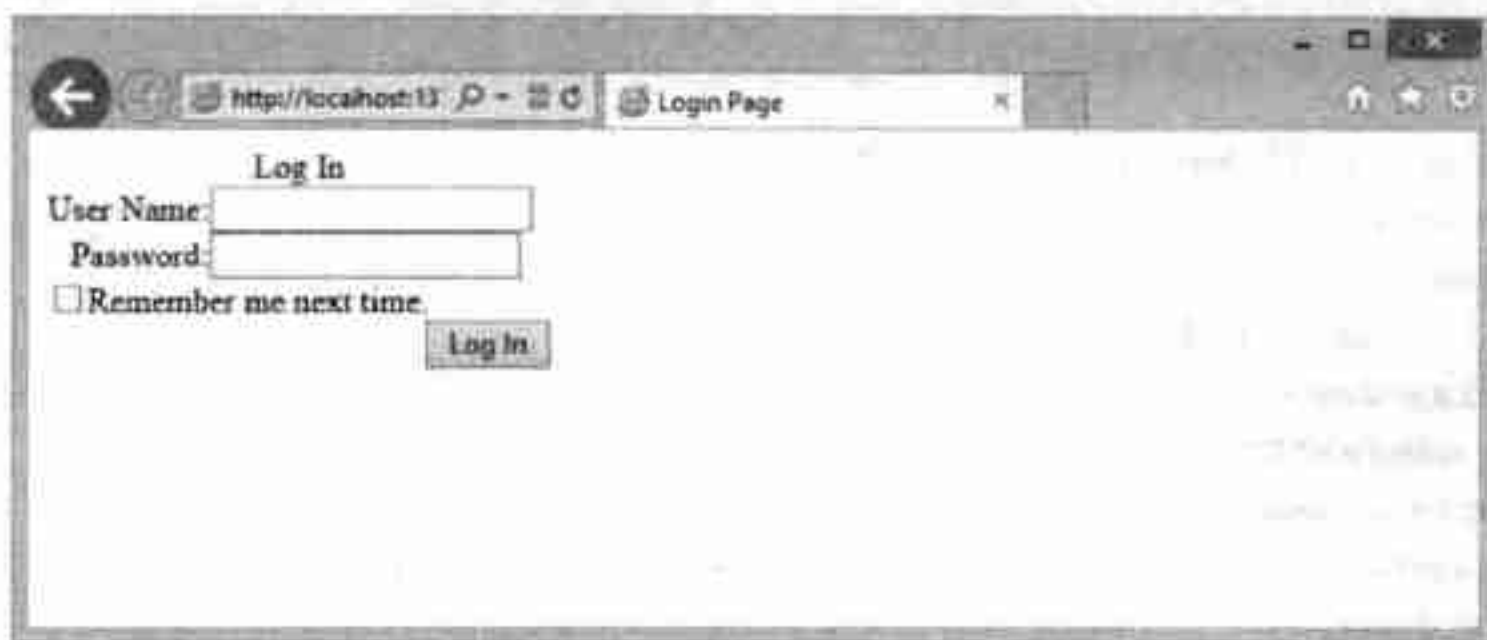


图 19-7

在图 19-7 中, Login 控件要求用户输入用户名和密码。其中的复选框允许把 cookie 存储在客户端, cookie 可以让终端用户绕过以后的登录过程。把 Login 控件的 DisplayRememberMe 属性设置为 False, 就可以删除该复选框和用于记住用户的相关文本。

除了 DisplayRememberMe 属性之外, 还可以使用 RememberMeText 和 RememberMeSet 属性来操作 Login 控件。RememberMeText 属性很容易理解, 因为它的值只定义了复选框旁边的文本。但 RememberMeSet 属性比较有趣, 这个属性为布尔值(默认为 False), 指定用户使用 Login 控件登录后, 是否在客户端的计算机上设置永久的 cookie。如果该属性设置为 True, 而 DisplayRememberMe 属性也设置为 True, 那么在浏览器中生成 Login 控件时就默认选中复选框。如果 DisplayRememberMe 属性设置为 False(表示终端用户看不到复选框或不能选择永久保存登录 cookie 的选项), RememberMeSet 属性

设置为 True，就自动在用户的计算机上设置 cookie，而不需要用户知道或选择。采用这种方法时必须慎重，因为终端用户有时会使用公共计算机，这种方法意味着把授权 cookie 设置在公共计算机上。

cookie 会保留在客户端的计算机上，直到用户从应用程序中注销为止(如果提供这个选项)。有了永久保存的 cookie，并假定终端用户没有从应用程序中注销，该用户在返回应用程序时就永远不需要再次登录，因为他的凭据由 cookie 中的内容提供。终端用户登录到应用程序后，就会返回到最初要访问的页面。

还可以修改 Login 控件的外观和操作方式，修改方式与其他控件一样。一种方式是在该控件的智能标记中单击 Auto Format 链接，打开一个修改该控件外观和操作方式的选项列表，如图 19-8 所示。

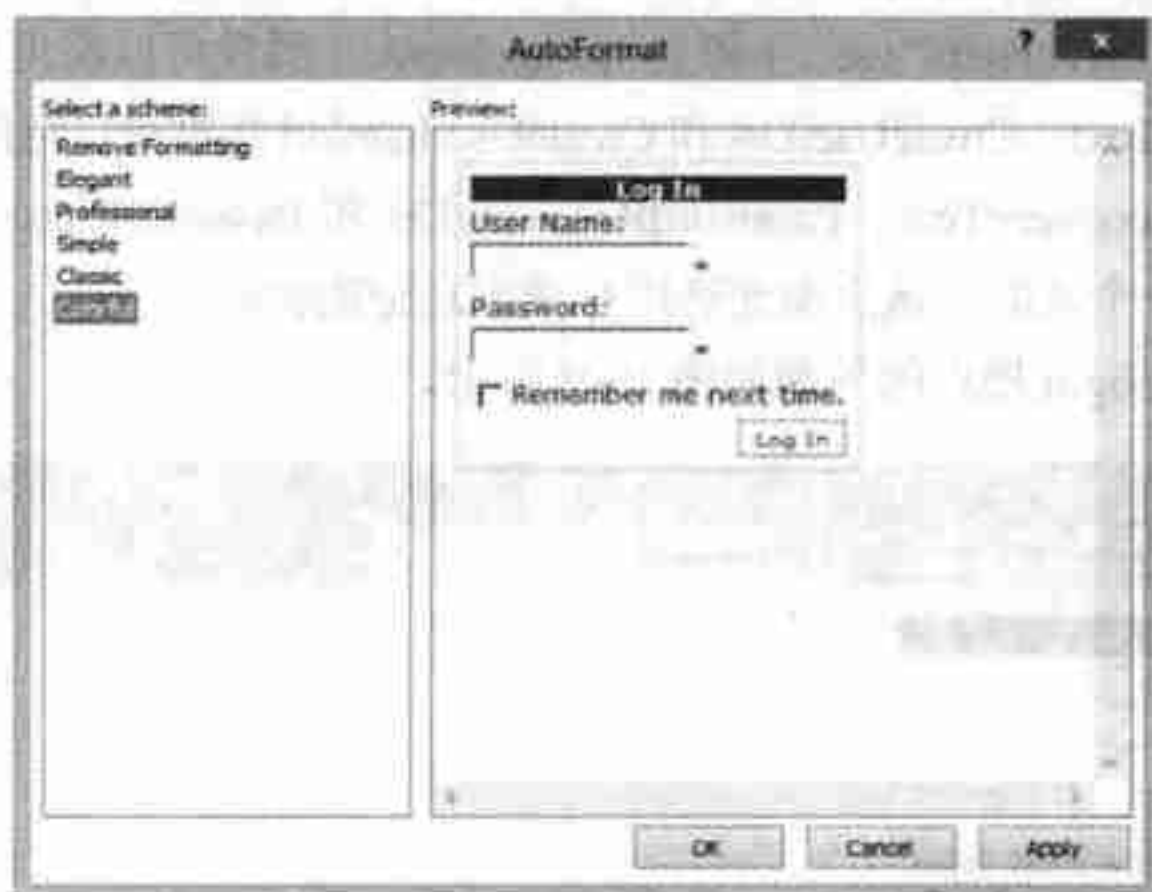


图 19-8

例如，选择 Colorful 选项并修改代码。程序清单 19-14 显示了为该选项生成的代码。

程序清单 19-14 格式化的 Login 控件

```
<asp:Login ID="Login1" runat="server" BackColor="#FFFBD6"
  BorderColor="#FFDFAD" BorderPadding="4" BorderStyle="Solid"
  BorderWidth="1px" Font-Names="Verdana" Font-Size="0.8em"
  ForeColor="#333333" TextLayout="TextOnTop">
  <TextBoxStyle Font-Size="0.8em" />
  <LoginButtonStyle BackColor="White" BorderColor="#CC9966"
    BorderStyle="Solid" BorderWidth="1px" Font-Names="Verdana"
    Font-Size="0.8em" ForeColor="#990000" />
  <InstructionTextStyle Font-Italic="True" ForeColor="Black" />
  <TitleTextStyle BackColor="#990000" Font-Bold="True" Font-Size="0.9em"
    ForeColor="White" />
</asp:Login>
```

在这个程序清单中，许多子元素用于修改控件显示的特定项。Login 控件包含的样式元素如下：

- <CheckboxStyle>
- <FailureTextStyle>
- <HyperLinkStyle>
- <InstructionTextStyle>
- <LabelStyle>

- <LoginButtonStyle>
- <TextBoxStyle>
- <TitleTextStyle>
- <ValidatorTextStyle>

Login 控件的许多属性都可以修改控件的外观和操作方式。一处有趣的修改是可以在控件的底部添加一些链接，以访问其他资源。通过这些链接，用户可以获得帮助或注册到应用程序，从而可以得到提供的登录凭据。

可以提供链接来完成以下功能：

- 使用 HelpPageText、HelpPageUrl 和 HelpPageImageUrl 属性可以将用户重定向到帮助页面。
- 使用 CreateUserText、CreateUserUrl 和 CreateUserImageUrl 属性可以将用户重定向到注册页面。
- 使用 PasswordRecoveryText、PasswordRecoveryUrl 和 PasswordRecoveryImageUrl 属性可以将用户重定向到一个页面，该页面允许用户重置忘记的密码。

使用这些链接时，Login 控件的外观如图 19-9 所示。

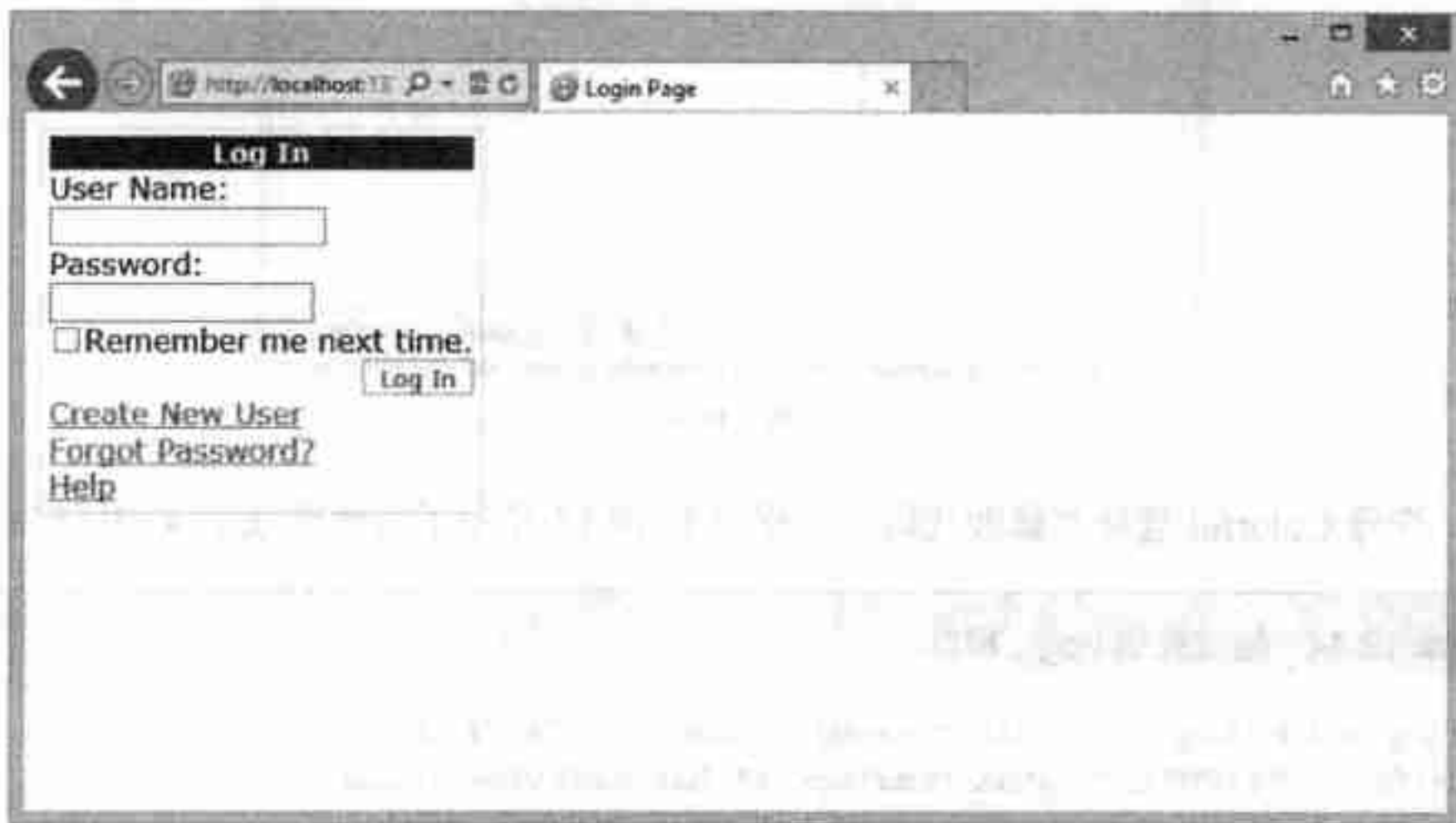


图 19-9

3. 编程登录用户

除了使用 Login 控件的预置机制之外，还可以使用 Membership 类编程完成这项任务。为了验证接收到的凭据，可以使用这个类的 ValidateUser 方法。ValidateUser 方法的签名如下所示：

```
Membership.ValidateUser(username As String, password As String)
```

这个方法的用法如程序清单 19-19 所示。

程序清单 19-15 编程验证用户的凭据

```
if (Membership.ValidateUser (TextBox1.Text, TextBox2.Text)) {
    FormsAuthentication.RedirectFromLoginPage (TextBox1.Text, false);
}
else {
```



```

    Label1.Text = "You are not registered with the site.";
}

```

如果用户的凭据通过验证, `ValidateUser` 方法就返回布尔值 `True`, 否则就返回 `False`。在程序清单 19-19 的代码中, 使用 `RedirectFromLoginPage` 方法将凭据通过验证的终端用户退出登录页面。这个方法的参数是用户名和布尔值, 布尔值指定是否通过 cookie 设置来永久保存凭据。

4. 锁定输入错误密码的用户

在为用户提供登录表单时, 一定要防止用户多次尝试输入错误的密码。如果怀有恶意的终端用户知道一个用户名, 就会多次尝试使用不同的密码来访问应用程序。一定要阻止这类行为, 不能让恶意的终端用户使用该用户名尝试输入数百个可能的密码。

ASP.NET 内置了这类行为的防御措施。在 `aspnet_Membership` 表中, 有两列用于防止这类行为: `FailedPasswordAttemptCount` 和 `FailedPasswordAttemptWindowStart`。

默认情况下, 在一个最长显示 10 分钟的窗口中, 至多只能在一次登录中尝试给同一个用户名输入 5 次密码。第 5 次尝试失败后, 账户就会被锁定。要实现这个功能, 需要在 ASP.NET 中将 `IsLockedOut` 列设置为 `True`。

可以控制允许尝试输入密码的次数和应用程序中登录窗口的显示时间, 在 `machine.config` 文件的 `SqlMembershipProvider` 声明中定义这两项。可以在服务器的配置文件或应用程序的 `web.config` 文件中修改这些值。在 `web.config` 文件中修改这些值的过程如程序清单 19-19 所示。

程序清单 19-16 在提供程序声明中修改密码尝试的相应值

```

<configuration>
  <system.web>
    <membership defaultProvider="AspNetSqlMembershipProvider">
      <providers>
        <clear />
        <add connectionStringName="LocalSqlServer"
          applicationName="/"
          maxInvalidPasswordAttempts="3"
          passwordAttemptWindow="15"
          name="AspNetSqlMembershipProvider"
          type="System.Web.Security.SqlMembershipProvider, System.Web,
            Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
      </providers>
    </membership>
  </system.web>
</configuration>

```

要确定允许尝试输入密码的次数, 可以使用 `maxInvalidPasswordAttempts` 特性。本例中把该特性的值改为 3, 表示在账户被锁定之前, 用户可以(在指定的窗口显示时间内)输入 3 次密码。`maxInvalidPasswordAttempts` 特性的默认值是 5。可以使用 `passwordAttemptWindow` 特性将允许输入密码的窗口的显示时间设置为 15 分钟。这个特性的默认值是 10, 因此本例将窗口显示时间增加了 5 分钟。

有了这些项后, 下一步就是进行测试。程序清单 19-17 是一个测试示例, 它假定已建立了一个应用程序并为其添加了一个用户。

程序清单 19-17 测试密码尝试输入次数的示例页面

```

<%@ Page Language="C#" %>
<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        if(CheckBox1.Checked == true)
        {
            MembershipUser user = Membership.GetUser(TextBox1.Text);
            user.UnlockUser();
        }
        if(Membership.ValidateUser(TextBox1.Text, TextBox2.Text))
        {
            Label1.Text = "You are logged on!";
        }
        else
        {
            MembershipUser user = Membership.GetUser(TextBox1.Text);
            Label1.Text = "Locked out value: " + user.IsLockedOut.ToString();
        }
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Login Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <h1>Login User</h1>
        <p>
            <asp:CheckBox ID="CheckBox1" runat="server" Text="Unlock User" />
        </p>
        <p>
            Username<br />
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        </p>
        <p>Password<br />
            <asp:TextBox ID="TextBox2" runat="server"
                TextMode="Password"></asp:TextBox>
        </p>
        <p>
            <asp:Button ID="Button1" runat="server" Text="Login"
                OnClick="Button1_Click" />
        </p>
        <p>
            <asp:Label ID="Label1" runat="server"></asp:Label>
        </p>
    </div>
    </form>
</body>
</html>

```

这个页面包含两个文本框，分别用于输入用户名和密码。在这两个文本框的上面是一个复选框，

在账户因输入了错误的密码而被锁定后，可以使用该复选框解除锁定用户。

运行这个页面，对用户输入 3 次不正确的密码，就会得到如图 19-10 所示的结果。

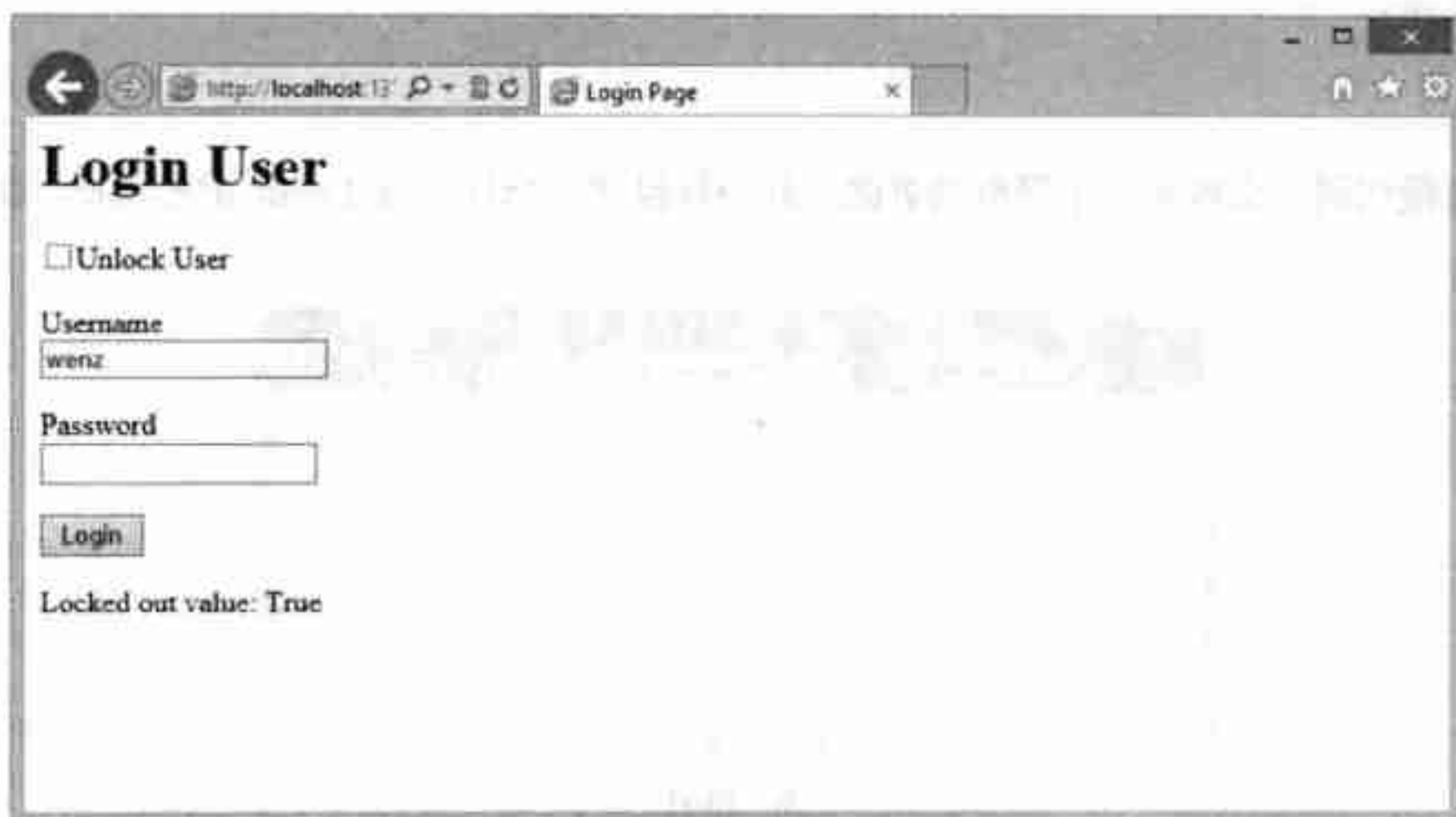


图 19-10

可以通过 `MembershipUser` 对象来读取 `IsLockOut` 属性。这个对象允许编程访问包含在 `aspnet_Membership` 表中的用户数据。在本例中获取 `IsLockOut` 属性后，把它显示在屏幕上。`MembershipUser` 对象也有许多方法，其中一个是 `UnlockUser` 方法，如果在按钮单击事件中选中了复选框，就会调用这个方法。



确保在继续前再次给用户解锁，否则就不再能登录到应用程序。

19.1.4 处理验证用户

用户通过验证后，就可以使用 ASP.NET 4.5 中提供的许多不同的服务器控件和方法以处理用户信息。在这个工具集合中，还包含 `LoginStatus` 和 `LoginName` 控件。

1. LoginStatus 服务器控件

`LoginStatus` 服务器控件可以让用户单击链接，在站点上登录或注销。作为这个控件的一个示例，从 `web.config` 文件中删除 `<deny>` 元素，以使站点的页面可以由未验证的用户访问。然后编写 `Default.aspx` 页面，相应的代码如程序清单 19-18 所示。

程序清单 19-18 LoginStatus 控件的登录和注销功能

```
<%@ Page Language="C#" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Login or Logout</title>
</head>
```



```

<body>
  <form id="form1" runat="server">
    <asp:LoginStatus ID="LoginStatus1" runat="server" />
  </form>
</body>
</html>

```

运行这段代码，会得到一个简单的页面，其中只包含一个标题为 Login 的超链接，如图 19-11 所示。



图 19-11

单击 Login 超链接会进入 Login.aspx 页面，应在该页面上提供凭据。提供凭据之后，就会重定向到 Default.aspx 页面，但现在页面包含一个标题为 Logout 的超链接，如图 19-12 所示。用户未通过验证时，LinkStatus 控件显示一个链接，而在用户通过验证后显示另一个链接。单击 Logout 超链接，会注销用户，重新绘制 Default.aspx 页面，但这次会显示 Login 超链接。



图 19-12

2. LoginName 服务器控件

LoginName 服务器控件可以显示验证用户的用户名，这是目前该控件很常见的一种用途。作为该控件用法的一个示例，修改 Default.aspx 页面，使其在用户登录时包含验证用户的登录名，如程序清单 19-19 所示。

程序清单 19-19 显示验证用户的用户名

```

<%@ Page Language="C#" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Login or Logout</title>
</head>
<body>

```

```

<form id="form1" runat="server">
  <asp:LoginStatus ID="LoginStatus1" runat="server" />
  <p><asp:LoginName ID="LoginName1" runat="server"
    Font-Bold="True" Font-Size="XX-Large" /></p>
</form>
</body>
</html>

```

当用户登录到应用程序并被返回到 Default.aspx 页面时, 就会显示他的用户名和 LoginStatus 控件生成的超链接, 如图 19-13 所示。



图 19-13

除了显示登录用户的用户名之外, 还可以使用 LoginName 控件的 FormatString 属性添加文本。例如, 要在用户名的旁边显示一条欢迎消息, 可以按照如下代码所示构建 LoginName 控件:

```

<asp:LoginName ID="LoginName1" runat="Server"
  FormatString="Welcome to our Website {0}!" />

```

还可以在页面事件中只使用如下构造代码:

```

LoginName1.FormatString = "Welcome to the site {0}!";

```

在生成页面时, ASP.NET 会使用登录用户的用户名替换字符串的 {0} 部分。结果如下所示:

```

Welcome to the site Christian!

```

如果不想在使用 LoginName 控件时显示用户名, 只须忽略字符串的 {0} 部分即可。该控件会把 FormatString 属性的值放在页面上。

19.1.5 显示在线用户数

成员资格服务的一项优秀功能是显示给定时刻的在线用户数。这是门户网站和论坛中非常常见的选项, 以此向站点的访问者说明该站点的受欢迎程度。

为了显示在线用户数, 可以使用 Membership 类提供的 GetNumberOfUsersOnline 方法。将程序清单 19-20 中的代码添加到图 19-11 所示的 Default.aspx 页面。

程序清单 19-20 显示在线用户数

```

<%@ Page Language="C#" %>
<script runat="server">
  protected void Page_Load(object sender, EventArgs e)
  {

```

```

        Labell.Text = Membership.GetNumberOfUsersOnline().ToString();
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Login or Logout</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:LoginStatus ID="LoginStatus1" runat="server" />
        <p><asp:LoginName ID="LoginName1" runat="server"
            Font-Bold="True" Font-Size="XX-Large" /></p>
        <p>There are <asp:Label ID="Labell" runat="server" Text="0" />
            users online.</p>
    </form>
</body>
</html>

```

生成页面时，会显示过去 15 分钟内登录的用户数，图 19-14 就是生成结果的一个示例。



图 19-14

可以看出，过去 15 分钟内有两位用户登录。这个 15 分钟的时间段是在 `machine.config` 文件的 `<membership>` 元素中确定的：

```

<membership userIsOnlineTimeWindow="15" >
</membership>

```

`userIsOnlineTimeWindow` 默认设置为 15。这里指定的数字以分钟为单位。要增加这个时间，只须增大该值即可。除了在 `machine.config` 文件中指定这个数字之外，还可以在 `web.config` 文件中设置这个数字。

19.1.6 处理密码

我们经常会登录网络，并且会在 Internet 的许多不同的 Web 站点上使用用户名和密码组合。因此，终端用户常常会忘记密码或要更改密码。ASP.NET 提供了使用成员资格服务的两个新服务器控件，让终端用户可以修改密码或找回忘记的密码。

1. ChangePassword 服务器控件

ChangePassword 服务器控件允许终端用户直接在浏览器上修改密码。程序清单 19-21 显示了

ChangePassword 控件的用法。

程序清单 19-21 允许用户修改密码

```
<%@ Page Language="C#" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Change Your Password</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:LoginStatus ID="LoginStatus1" runat="server" />
    <p><asp:ChangePassword ID="ChangePassword1" runat="server">
      </asp:ChangePassword></p>
  </form>
</body>
</html>
```

这是<asp:ChangePassword>控件的一种相当简单的用法。运行这个页面，结果如图 19-15 所示。



图 19-15

ChangePassword 控件会生成一个表单，要求用户输入以前的密码。它还要求终端用户将新密码输入两次。如果用户已登录，单击 Change Password 按钮就会修改密码。如果终端用户没有登录到应用程序，他就会被重定向到登录页面，只有登录用户才能修改密码。修改密码后，会通知终端用户，如图 19-16 所示。

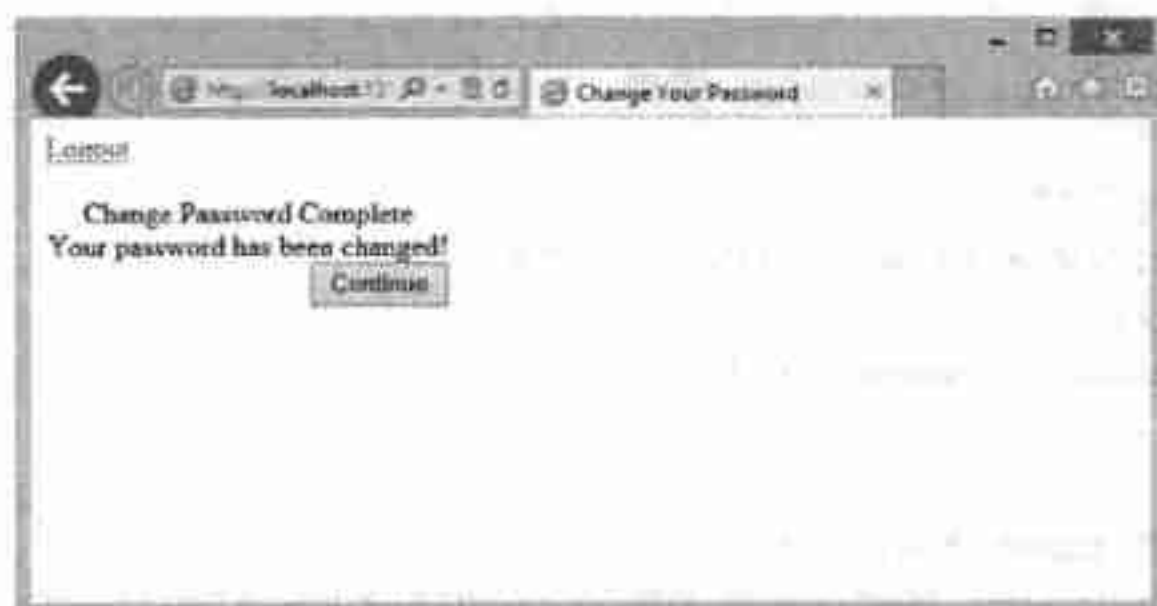


图 19-16

终端用户可以修改密码是因为成员资格提供程序的 enablePasswordReset 特性设置为 true。为了禁用这个功能，可以把 enablePasswordReset 特性设置为 false。

在终端用户修改密码时，还可以指定构建密码必须遵循的规则。例如，要求密码包含特定数量

的字符，或者要求在字母字符之外使用数字和/或特殊字符。使用 `NewPasswordRegularExpression` 属性可以指定新密码需要遵循的构建规则，如下所示：

```
NewPasswordRegularExpression="@\" (?=.{6,}) (?=(.*\d){1,}) (?=(.*\W){1,}) "
```

利用这个正则表达式来检查终端用户创建的新密码。如果不匹配正则表达式，就可以使用 `NewPasswordRegularExpressionErrorMessage` 属性(ASP.NET 中一个名称很长的属性)在控件输出中显示一条错误消息。

2. PasswordRecovery 服务器控件

人们很容易忘记密码。因此，应提供从数据存储中提取密码的方式。`PasswordRecovery` 服务器控件提供了一种简单的方式来完成该任务。

密码恢复通常通过电子邮件将终端用户的密码发送给用户。因此，需要搭建 SMTP 服务器(可能与应用服务器相同)。在 `web.config` 文件中配置 SMTP 服务器，如程序清单 19-22 所示。

程序清单 19-22 在 `web.config` 文件中配置通过电子邮件发送的密码

```
<configuration>
  <system.web>
    <!-- Removed for clarity -->
  </system.web>
  <system.net>
    <mailSettings>
      <smtp from="wenz@example.com">
        <network host="localhost" port="25"
          defaultCredentials="true" />
      </smtp>
    </mailSettings>
  </system.net>
</configuration>
```

正确建立了 `<mailSettings>` 元素后，就可以开始使用 `PasswordRecovery` 控件。`PasswordRecovery` 控件的简单用法如程序清单 19-23 所示。

程序清单 19-23 使用 `PasswordRecovery` 控件

```
<%@ Page Language="C#" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Getting Your Password</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:PasswordRecovery ID="PasswordRecovery1" runat="server">
      <MailDefinition From="wenz@example.com">
      </MailDefinition>
    </asp:PasswordRecovery>
  </form>
</body>
</html>
```

<asp:PasswordRecovery>元素需要<MailDefinition>子元素。<MailDefinition>元素包含要发送给终端用户的电子邮件的详细信息。最低要求是使用 From 属性,它提供了电子邮件中 From 部分的电子邮件地址。这个属性的 String 值应是电子邮件地址。<MailDefinition>元素的其他属性包括:

- BodyFileName
- CC
- From
- IsBodyHtml
- Priority
- Subject

运行这个页面, PasswordRecovery 控件会要求输入用户的用户名,如图 19-17 所示。

有了用户名,成员资格服务就提取终端用户以前输入的问题和答案,并生成如图 19-18 所示的视图。



图 19-17

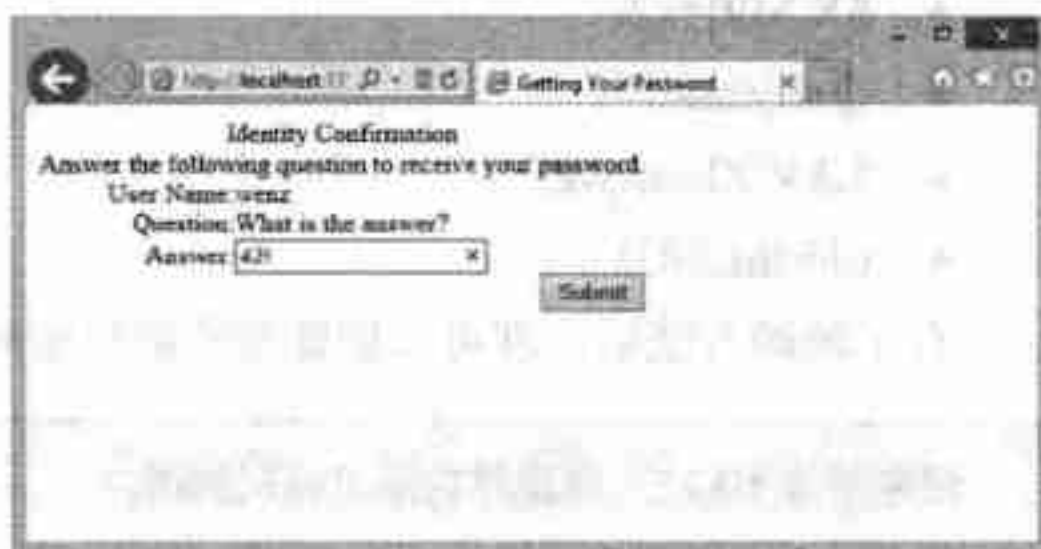


图 19-18

如果问题回答正确(注意答案区分大小写),就生成一封包含密码的电子邮件,并发送给终端用户。如果问题回答不正确,就显示一条错误消息。当然,如果禁用成员资格系统中的问答功能,就不使用问题。



如果没有可用的邮件服务器,建议使用 <http://smtp4dev.codeplex.com/> 上的开源邮件服务器模拟器,它不是实际发送电子邮件,而只是显示它们。这样调试这个功能会容易一些。

成员资格服务数据存储没有存储实际的密码,而只存储了密码的散列版本。因此我们得不到最初的密码,而只能得到 ASP.NET 生成的新密码。

为了能把实际密码发送回用户,必须修改密码在成员资格服务数据存储中的存储方式。可以通过改变成员资格数据提供程序的 PasswordFormat 特性(如本章前面所述)来完成该操作。除了 Hashed 值之外,该特性的另外两个值是 Clear 和 Encrypted。把该特性的值改为 Clear 或 Encrypted,就可以将密码以可读的格式发送回终端用户。

3. 生成随机密码

一些应用程序在创建用户时必须生成随机密码。ASP.NET 提供了辅助方法,用于获取随机密码。

程序清单 19-24 就是创建一个辅助方法以获得随机密码的示例。

程序清单 19-24 生成随机密码

```
protected string GeneratePassword()
{
    string returnPassword;
    returnPassword = Membership.GeneratePassword(10, 3);
    return returnPassword;
}
```

要在 ASP.NET 中生成随机密码, 可以使用 `GeneratePassword` 辅助方法。这个方法可以生成指定长度的随机密码, 还可以指定密码应至少包含多少个非数字字母的字符。这个例子使用该方法 5 次, 生成了如下结果(当然, 读者的运行结果应与此不同):

- D](KQg6s2[
- \$X.M9]*x2-
- Q+llY2#zD%
- %kWZL@zy&f
- o]&lhL#iU1

有了辅助方法后, 就可以创建带有随机密码的用户, 如程序清单 19-25 所示。

程序清单 19-25 创建带有随机密码的用户

```
Membership.CreateUser(TextBox1.Text, GeneratePassword());
```

19.2 ASP.NET 4.5 的授权

处理了要访问 Web 应用程序的用户的注册和身份验证之后, 下一步就是授权。用户可以查看什么内容? 要承担什么角色? 这些对于任何 Web 应用程序来说都是很重要的问题。下面首先介绍如何给验证用户显示某些项, 给未验证用户显示另外一些项。

19.2.1 使用 LoginView 服务器控件

LoginView 服务器控件可以控制用户可以查看页面某个部分的什么信息。使用 LoginView 控件可以指定验证用户查看页面的哪些部分, 未验证用户查看页面的哪些部分。程序清单 19-26 显示了这个控件的一个例子。

程序清单 19-26 通过 LoginView 控件控制用户可以查看的信息

```
<%@ Page Language="C#" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Changing the View</title>
</head>
<body>
    <form id="form1" runat="server">
```

```

<asp:LoginStatus ID="LoginStatus1" runat="server" />
<p>
<asp:LoginView ID="LoginView1" runat="server">
  <LoggedInTemplate>
    Here is some REALLY important information that you should know
    about all those people that are not authenticated!
  </LoggedInTemplate>
  <AnonymousTemplate>
    Here is some basic information for you.
  </AnonymousTemplate>
</asp:LoginView></p>
</form>
</body>
</html>

```

<asp:LoginView>是一个模板化控件，它有两个子元素<LoggedInTemplate>和<AnonymousTemplate>。在本例中，在<AnonymousTemplate>部分定义的信息是供未验证用户查看的，如图 19-19 所示。

在<LoggedInTemplate>部分定义验证用户查看的内容，这些内容与未验证用户查看的内容完全不同，如图 19-20 所示。

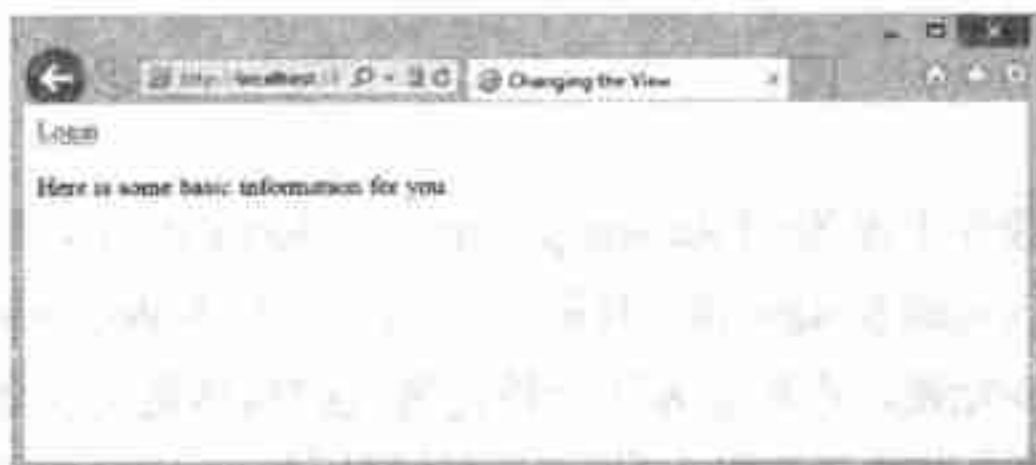


图 19-19

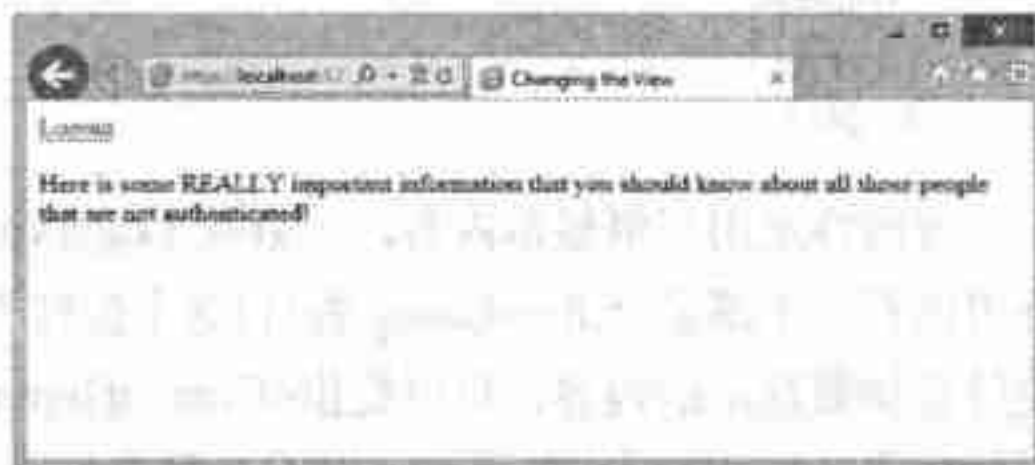


图 19-20

在这两个模板中只放置了简单的 ASCII 文本，但可以在其中放置其他任何内容，包括其他服务器控件。这就表示在模板化的部分可以显示页面的所有部分，包括表单。

除了使用 LoginView 控件的<LoggedInTemplate>和<AnonymousTemplate>元素之外，还可以激活页面的某些部分或实体的特定内容，这些实体是某个角色的一部分，例如，某人是 Admin 组的成员。为此，可以使用 LoginView 控件的<RoleGroups>部分，如程序清单 19-27 所示。

程序清单 19-27 为特定的组提供视图

```

<%@ Page Language="C#" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Changing the View</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:LoginStatus ID="LoginStatus1" runat="server" />
    <p>
      <asp:LoginView ID="LoginView1" runat="server">
        <LoggedInTemplate>

```

```

        Here is some REALLY important information that you should know
        about all those people that are not authenticated!
    </LoggedInTemplate>
    <AnonymousTemplate>
        Here is some basic information for you.
    </AnonymousTemplate>
    <RoleGroups>
        <asp:RoleGroup Roles="Admins">
            <ContentTemplate>
                You are an Admin!
            </ContentTemplate>
        </asp:RoleGroup>
        <asp:RoleGroup Roles="CoolPeople">
            <ContentTemplate>
                You are cool!
            </ContentTemplate>
        </asp:RoleGroup>
    </RoleGroups>
</asp:LoginView><p>
</form>
</body>
</html>

```

要向特定用户组显示内容，可以在 LoginView 控件中添加<RoleGroups>元素。<RoleGroups>部分可以有一个或多个 RoleGroup 控件(这个控件不在 Visual Studio 的工具箱中)。为了使用 RoleGroup 控件提供要显示的内容，可以提供<ContentTemplate>元素，它可以为属于特定角色的实体定义要显示的内容。在<ContentTemplate>部分放置什么内容完全取决于开发人员。可以放置原始文本(如本例中所示)，甚至其他 ASP.NET 控件。

注意在<RoleGroups>部分放置已定义角色的顺序。用户登录到站点时，首先要检查他们是否匹配某个已定义的角色。最先匹配的角色是用于 LoginView 控件的视图，即使用户匹配多个角色，也首先考虑该视图。也可以在<asp:RoleGroups>控件的 Roles 属性中放置多个角色，如下所示：

```

<asp:RoleGroup Roles="CoolPeople, HappyPeople">
    <ContentTemplate>
        You are cool or happy (or both)!
    </ContentTemplate>
</asp:RoleGroup>

```

19.2.2 为角色管理建立 Web 站点

除了前面介绍的成员资格服务之外，ASP.NET 还提供了终端用户管理服务的另一个方面：ASP.NET 角色管理服务。成员资格服务涵盖了应用程序的身份验证的所有方面，而角色管理服务涵盖的是授权。与成员资格服务可以使用前面列出的任一数据提供程序一样，角色管理服务也可以使用 SQL Server 提供程序(SqlRoleProvider)和任意定制的提供程序。事实上，这个服务与成员资格服务在许多方面都是兼容的。图 19-21 简单显示了角色管理服务的一些特殊之处。

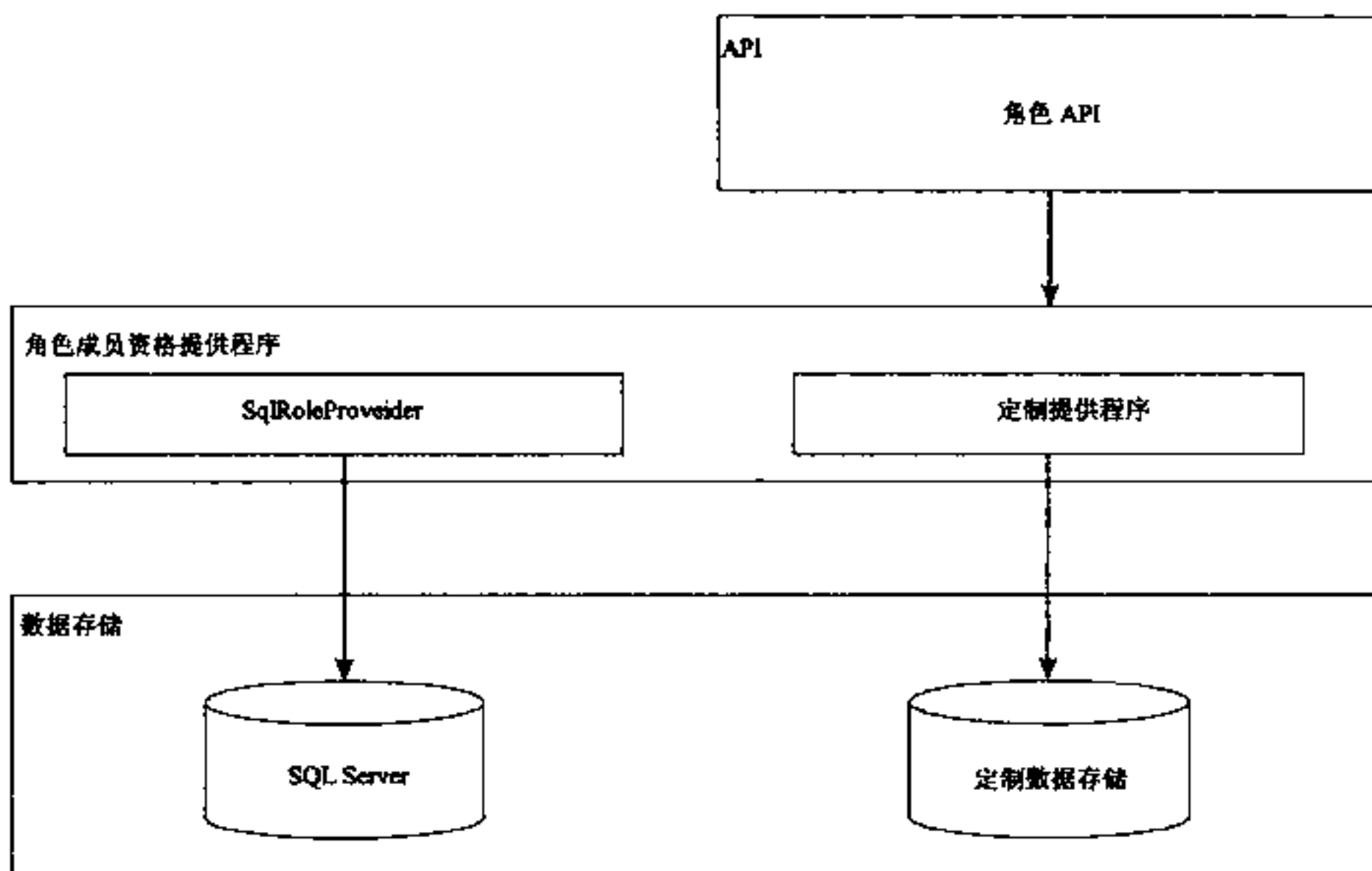


图 19-21

1. 修改<roleManager>部分

使用角色管理服务的第一步是在 `machine.config` 或 `web.config` 文件中修改角色管理提供程序的行为。在 `machine.config.comments` 文件中，有一部分专门用于处理角色管理服务，如程序清单 19-28 所示。

程序清单 19-28 `machine.config.comments` 文件中的角色管理提供程序设置

```

<roleManager
  enabled="false"
  cacheRolesInCookie="false"
  cookieName=".ASPXROLES"
  cookieTimeout="30"
  cookiePath="/"
  cookieRequireSSL="false"
  cookieSlidingExpiration="true"
  cookieProtection="All"
  defaultProvider="AspNetSqlRoleProvider"
  createPersistentCookie="false"
  maxCachedResults="25">
  <providers>
    <clear />
    <add connectionStringName="LocalSqlServer" applicationName="/"
      name="AspNetSqlRoleProvider" type="System.Web.Security.SqlRoleProvider,
      System.Web, Version=4.0.0.0, Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a" />
    <add applicationName="/" name="AspNetWindowsTokenRoleProvider"
      type="System.Web.Security.WindowsTokenRoleProvider, System.Web,
      Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
  </providers>
</roleManager>
  
```

角色管理服务在 machine.config.comments 文件中定义其设置，如上面的程序清单所示。可以直接在 machine.config 文件中修改这些设置，也可以在 web.config 文件中执行一些修改以重写任意一个较高层面的设置(这些修改只用于当前的应用程序)。

在<roleManager>元素中定义主要设置。<roleManager>元素的一些特性如表 19-2 所示。

表 19-2

特 性	说 明
enabled	定义是否为应用程序激活角色管理服务。这个特性的值是布尔值，默认设置为 False。也就是说，角色管理服务默认为禁用。这可以避免用户从 ASP.NET 1.0/1.1 迁移到 ASP.NET 2.0、3.5、4 或 4.5 时出现的变化。因此，必须首先在 machine.config 或 web.config 文件中把这个值改为 True
cacheRolesInCookie	定义用户的角色是否可以在客户端计算机的 cookie 中存储。这个特性的值是布尔值，默认设置为 True。这是很理想的情况，因为从 cookie 中提取角色可避免 ASP.NET 通过角色管理提供程序查找用户的角色。如果要为所有的实例通过提供程序提取角色，就可以把该特性设置为 False
cookieName	定义用于发送给终端用户的 cookie 的名称，cookie 用于存储角色管理信息。cookie 默认命名为 ASPXROLES，不能改变这个名称
cookieTimeout	定义 cookie 的有效期(分钟)，默认值是 30 分钟
cookieRequireSSL	确定是否要求角色管理信息通过加密线路(SSL)发送，而不是作为明文发送。默认值是 False
cookieSlidingExpiration	指定 cookie 的超时是否在可变化的范围内，默认值是 True。这就表示，在应用程序的最后一个请求发出 30 分钟(或 cookieTimeout 特性指定的时间)后，终端用户的 cookie 才会过期。如果 cookieSlidingExpiration 特性的值设置为 false，cookie 就会在第一个请求发出 30 分钟后过期
createPersistentCookie	指定 cookie 是否会过期还是永远处于激活状态，默认值是 False，由于安全原因，永久的 cookie 并不总是很合适
cookieProtection	指定要应用于 cookie 的保护级别，cookie 存储在终端用户的计算机上，用于管理信息。其设置可以是 All、None、Encryption 和 Validation，我们应总是使用 All 设置
defaultProvider	定义用于角色管理服务的提供程序，在 ASP.NET 4.0 中该特性默认设置为 AspNetSqlRoleProvider，在 ASP.NET 4.5 中该特性默认设置为 DefaultRoleProvider

2. 修改 web.config 文件

下一步是配置 web.config 文件，使其可以使用角色管理服务。应用程序的某些页面或子部分可能只可以由具有特定角色的人访问。为了管理这种访问，应在 web.config 文件中定义访问权限。需要进行的修改如程序清单 19-29 所示。

程序清单 19-29 修改 web.config 文件

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.web>
    <compilation targetFramework="4.5" />
    <roleManager enabled="true"/>
    <authentication mode="Forms" />
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</configuration>
```

```

    </authorization>
</system.web>
<location path="AdminPage.aspx">
    <system.web>
        <authorization>
            <allow roles="AdminPageRights" />
            <deny users="*" />
        </authorization>
    </system.web>
</location>
</configuration>

```

web.config 文件做了两件事情：首先，第一个<system.web>部分的功能与本章前面的成员资格服务的对应部分没有区别。<deny>元素拒绝所有未经验证用户的访问。

web.config 文件的第二部分相当有趣。<location>元素用于定义应用程序中某个页面(AdminPage.aspx)的访问权限。在本例中，只有包含在 AdminPageRights 角色中的用户才能访问该页面，而其他用户(无论他们是否已通过身份验证)都不能访问该页面。使用星号(*)作为<deny>元素的 users 特性值时，就表示所有的用户(无论他们是否已通过身份验证)都不能访问指定的资源。但是，使用<allow>元素可以改变这种拒绝访问情况，该元素允许包含在特定角色中的用户对指定的资源进行访问。

19.2.3 添加和检索应用程序角色

有了 machine.config 或 web.config 文件后，就可以给角色管理服务添加角色。角色管理服务与成员服务一样，都使用数据存储来存储用户的信息。这些例子主要使用 SQL Server Express Edition 作为提供程序，因为它是默认的提供程序。

角色管理服务和成员资格服务的一处主要区别是，角色管理服务没有使用服务器控件。管理应用程序的角色和用户的角色信息是通过 Roles API 或 ASP.NET 4.5 提供的 Web Site Administration Tool 实现的。程序清单 19-30 说明了如何使用新方法给服务添加角色。

程序清单 19-30 给应用程序添加角色

```

<%@ Page Language="C#" %>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            ListBoxDataBind();
        }
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        Roles.CreateRole(TextBox1.Text);
        ListBoxDataBind();
    }

    protected void ListBoxDataBind()
    {
        ListBox1.DataSource = Roles.GetAllRoles();
    }

```



```

        ListBox1.DataBind();
    }
</script>
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Role Manager</title>
</head>
<body>
    <form id="form1" runat="server">
        <h1>Role Manager</h1>
        Add Role:<br />
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <p><asp:Button ID="Button1" runat="server" Text="Add Role to Application"
            OnClick="Button1_Click" /></p>
        Roles Defined:<br />
        <asp:ListBox ID="ListBox1" runat="server">
            </asp:ListBox>
        </form>
    </body>
</html>

```

这个例子允许在文本框中输入角色，再把它们提交给角色管理服务。角色管理服务包含的角色会显示在列表框中，如图 19-22 所示。

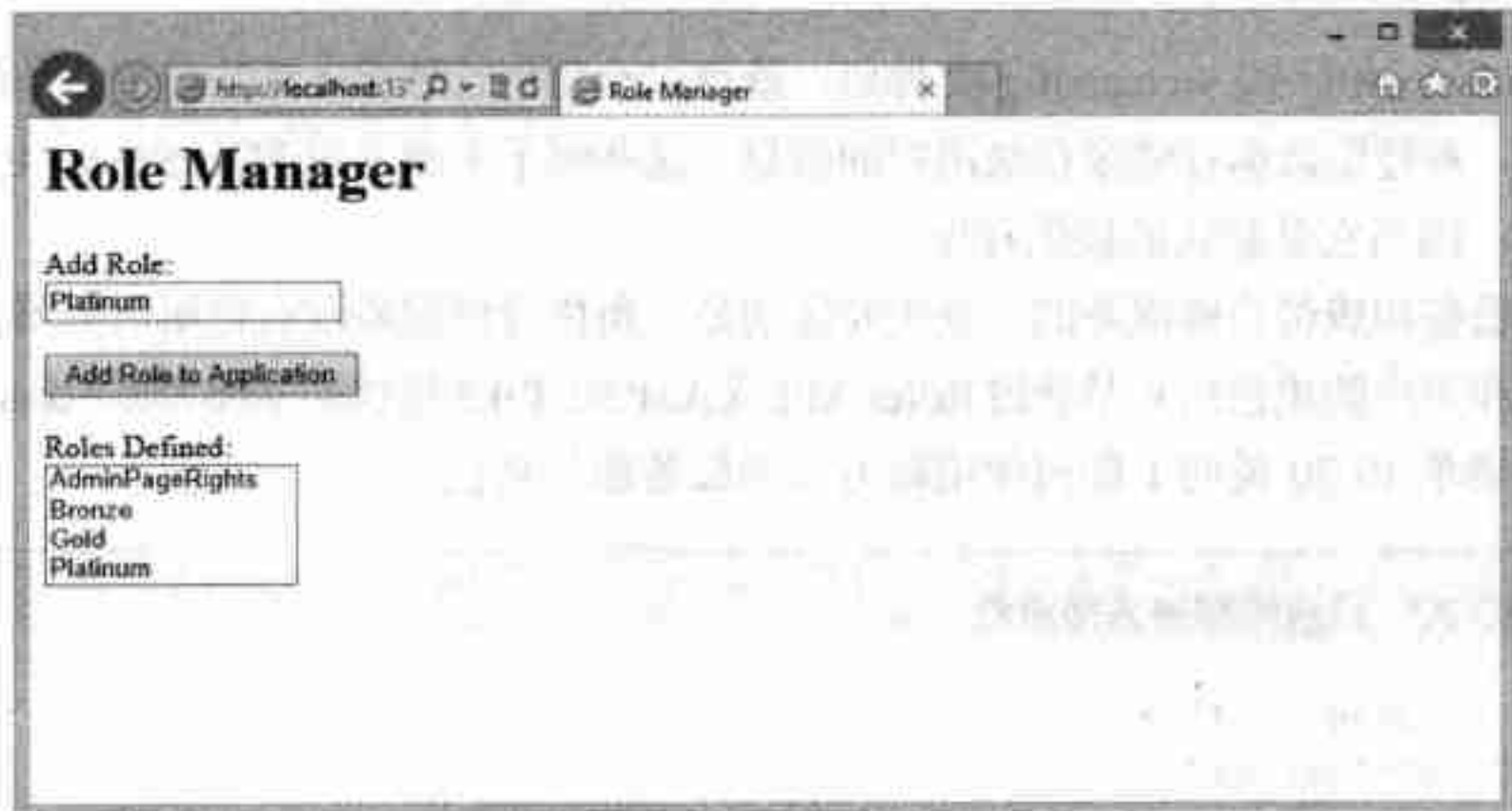


图 19-22

要把角色输入到管理服务中，只需使用 Roles 类的 CreateRole 方法。与 Membership 类一样，不需要实例化 Roles 类。要把角色添加到角色管理服务中，也可使用 CreateRole 方法，它只带一个字符串参数，即角色名：

```
Roles.CreateRole(rolename As String)
```

使用这个方法可以创建任意多个角色，但每个角色必须是唯一的，否则就会抛出异常。

要在应用程序的角色管理服务中提取角色(如上述例子的列表框中显示的一组角色)，可以使用 Roles 类的 GetAllRoles 方法。这个方法返回服务中所有角色的 String 集合：

```
Roles.GetAllRoles()
```

19.2.4 删除角色

我们一直在给服务添加角色，但有时也需要从服务中删除角色。在角色管理服务中删除角色与添加角色一样简单。要删除角色，可以使用 `DeleteRole` 方法的一种签名。`DeleteRole` 方法的第一种签名带一个字符串参数：角色名。第二种签名带角色名和一个布尔值，该布尔值表示在某个角色中包含一个或多个成员时是否抛出异常(这样就不会在角色中有用户时无意删除该角色)。

```
Roles.DeleteRole(rolename As String)
Roles.DeleteRole(rolename As String, throwOnPopulatedRole As Boolean)
```

程序清单 19-31 基于程序清单 19-30 中的一部分代码。对于这个例子，再添加一个按钮，它会触发第二个按钮单击事件，该事件从服务中删除角色。

程序清单 19-31 从应用程序中删除角色

```
protected void DeleteButton_Click(object sender, EventArgs e)
{
    foreach(ListItem li in ListBox1.Items) {
        if(li.Selected == true) {
            Roles.DeleteRole(li.ToString());
        }
    }
    ListBox.DataBind();
}
```

这个例子从 `ListBox` 控件中删除了选中的项。如果选择了多个选项(表示在 `ListBox` 控件中设置了属性 `SelectionMode="Multiple"`)，就会在 `foreach` 循环中依次删除服务中的每个角色。尽管我们使用 `Roles.DeleteRole(li.ToString())` 删除角色，但还可以使用 `Roles.DeleteRole(li.ToString(), True)`，以确保在角色包含成员时不删除该角色。

19.2.5 给角色添加用户

现在已经有了角色，并且可以在需要时删除这些角色，下一步就是给所创建的角色添加用户。如果角色没有与之相关的用户，这个角色就没有什么用处。要把用户添加到角色中，可以使用下面的构造代码：

```
Roles.AddUserToRole(username As String, rolename As String)
```

要把一个用户同时添加到多个角色中，可以使用下面的构造代码：

```
Roles.AddUserToRoles(username As String, rolenames() As String)
```

要把多个用户添加到一个角色中，可以使用下面的构造代码：

```
Roles.AddUsersToRole(usernames() As String, rolename As String)
```

最后，要把多个用户添加到多个角色中，可以使用下面的构造代码：

```
Roles.AddUsersToRoles(usernames() As String, rolenames() As String)
```

参数可以是集合，无论参数是 `usernames()` 还是 `rolenames()`，在方法中都是 `String` 数组。

19.2.6 获取某个角色的所有用户

在角色管理服务中查找信息是很容易的。可以确定某个角色中包含哪些用户，还可以确定某个用户包含在哪些角色中。

可以使用具体的方法实现这两种情况。首先看看如何确定包含在某个角色中的所有用户，如程序清单 19-32 所示。

程序清单 19-32 查找某个角色中的用户

```
<%@ Page Language="C#" %>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            DropDownDataBind();
        }
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        GridView1.DataSource = Roles.GetUsersInRole(DropDownList1.SelectedValue);
        GridView1.DataBind();
        DropDownDataBind();
    }

    protected void DropDownDataBind()
    {
        DropDownList1.DataSource = Roles.GetAllRoles();
        DropDownList1.DataBind();
    }
</script>

</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Role Manager</title>
</head>
<body>
    <form id="form1" runat="server">
        Roles:
        <asp:DropDownList ID="DropDownList1" runat="server">
        </asp:DropDownList>
        <asp:Button ID="Button1" runat="server" Text="Get Users In Role"
            OnClick="Button1_Click" />
        <br />
        <br />
        <asp:GridView ID="GridView1" runat="server">
        </asp:GridView>
    </form>
</body>
</html>
```


这个页面创建了一个下拉列表，其中包含应用程序的所有角色。单击按钮会显示所选角色的所有用户。使用 `GetUsersInRole` 方法确定特定角色的用户。这个方法带有一个字符串参数，它表示角色的名称：

```
Roles.GetUsersInRole(rolename As String)
```

运行代码，生成的页面如图 19-23 所示。



图 19-23

19.2.7 获取包含某个用户的所有角色

为了确定包含某个用户的所有角色，可创建一个页面，其中只包含一个文本框和一个按钮。在文本框中输入用户的名称，单击按钮就会提取角色，并填充到 `GridView` 控件中。按钮单击事件(所有的操作都在这个事件中进行)如程序清单 19-33 所示。

程序清单 19-33 获取包含某个用户的所有角色

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        GridView1.DataSource = Roles.GetRolesForUser(TextBox1.Text);
        GridView1.DataBind();
    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Role Manager</title>
</head>
<body>
    <form id="form1" runat="server">
        User:
        <asp:TextBox ID="TextBox1" runat="server" />
        <asp:Button ID="Button1" runat="server" Text="Get Roles of User"
            OnClick="Button1_Click" />
        <br />
        <br />
        <asp:GridView ID="GridView1" runat="server">
        </asp:GridView>
```

```

    </form>
</body>
</html>

```

上面的代码会生成如图 19-24 所示的结果。

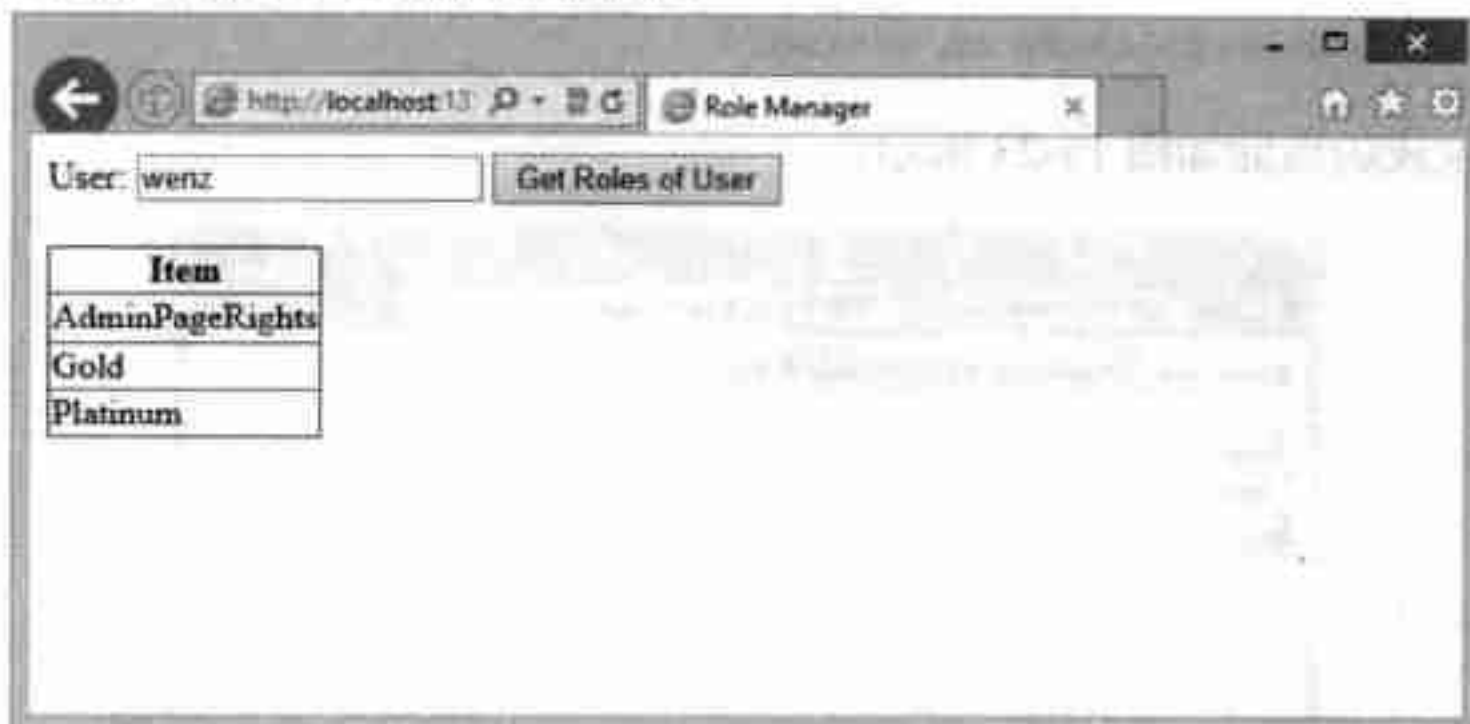


图 19-24

要获得包含某个用户的角色，可以使用 `GetRolesForUser` 方法。这个方法有两个签名。第一个签名在上面的例子中演示过，它带有一个表示用户名的字符串参数。另一个签名不带任何参数，返回登录到成员资格服务的用户所属的角色。

19.2.8 从角色中删除用户

除了给角色添加用户外，还可以轻易地从角色中删除用户。要从角色中删除用户，可以使用下面的构造代码：

```
Roles.RemoveUserFromRole(username As String, rolename As String)
```

要从多个角色中同时删除一个用户，可以使用下面的构造代码：

```
Roles.RemoveUserFromRoles(username As String, rolenames() As String)
```

要从一个角色中删除多个用户，可以使用下面的构造代码：

```
Roles.RemoveUsersFromRole(usernames() As String, rolename As String)
```

最后，要从多个角色中删除多个用户，可以使用下面的构造代码。

```
Roles.RemoveUsersFromRoles(usernames() As String, rolenames() As String)
```

参数显示为集合，无论是 `usernames()` 还是 `rolenames()`，这些参数在方法中都是 `String` 数组。

19.2.9 检查角色中的用户

最后一项可以执行的操作是检查某个用户是否在角色中。可以采用两种方式执行该检查。第一种方式是使用 `IsUserInRole` 方法。`IsUserInRole` 方法带两个参数——用户名和角色名：

```
Roles.IsUserInRole(username As String, rolename As String)
```

这个方法返回一个布尔值以表示用户的状态，其用法见程序清单 19-34。

程序清单 19-34 检查用户的角色状态

```

if(Roles.IsUserInRole(TextBox1.Text, "AdminPageRights"))
{
    // perform action here
}

```

除了 `IsUserInRole` 方法之外, 另一种方式是使用 `FindUsersInRole` 方法。这个方法可以在某个角色中根据名称搜索所有的用户。`FindUsersInRole` 方法带有两个参数: 角色名和用户名。它们都是 `String` 类型的值:

```
Roles.FindUsersInRole(rolename As String, username As String)
```

程序清单 19-35 是这个方法的一个示例。

程序清单 19-35 在某个角色中查找特定的用户

```

<%@ Page Language="C#" %>
<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        GridView1.DataSource =
            Roles.FindUsersInRole("AdminPageRights", TextBox1.Text);
        GridView1.DataBind();
    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Role Manager</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <asp:Button ID="Button1" runat="server" Text="Button"
            OnClick="Button1_Click" />
        <p><asp:GridView ID="GridView1" runat="server">
            </asp:GridView></p>
    </form>
</body>
</html>

```

19.2.10 角色的高速缓存方式

默认情况下, 从角色管理服务底层的数据存储中提取用户的角色后, 就可以在客户端计算机上把这些角色作为 `cookie` 存储。这样每次应用程序需要用户的角色状态时就不必访问数据存储。使用 `cookie` 总是有一些风险, 因为终端用户可以操纵 `cookie` 来访问应用程序中拒绝某个用户访问的信息或部分。

尽管角色可以高速缓存在 `cookie` 中, 但在默认情况下, 它们一次只能高速缓存 30 分钟。可以使用几种方式处理这个角色 `cookie`, 其中一些方式有助于更好地保护应用程序。

对应用程序的一种保护是在终端用户登录到站点时, 使用 `Roles API` 的 `DeleteCookie` 方法删除

这个角色 cookie，如程序清单 19-36 所示。

程序清单 19-36 在验证终端用户的身份时删除其角色 cookie

```
if (Membership.ValidateUser (TextBox1.Text, TextBox2.Text)
{
    Roles.DeleteCookie ();
    FormsAuthentication.RedirectFromLoginPage (TextBox1.Text, false);
}
else {
    Label1.Text = "You are not registered with the site.";
}
```

使用 Roles.DeleteCookie 方法可以从客户端计算机中删除任意用于定义用户角色的 cookie。如果终端用户再次登录到站点，在应用程序中重新验证他的角色是不会有问题的，不需要依赖 cookie 的内容。这一步为站点提供了更多的保护。

19.3 使用 SimpleMembership API

的确，喜欢 Membership API 的人不多，讨厌它的人很多。它是在 10 余年前构思的，但对于简单的任务而言过于复杂，也没有为目前 Web 上很常见的一些身份验证机制作好准备。稍后介绍现代的身份验证机制，但首先学习在 ASP.NET 中验证身份的一种较简单方式。

微软利用 WebMatrix 给初学者和学生提供了一个相当强大的 IDE，官方的 PR 就这么称呼它。这可能意味着，不应在商业项目中使用 WebMatrix(尽管允许这么做)，但对于较简单的应用程序可以快速获得结果。

随 ASP.NET Web Pages 技术一起发布的成员资格和角色部分是 WebMatrix 的组成部分。第 35 章详细介绍了可用的功能，但本章简要讨论 API 的一个特定方面，它也可以用于 ASP.NET Web 窗体应用程序。

SimpleMembership 和 SimpleRoles API 允许编程服务成员资格和角色信息，但使用的接口比常规的 Membership 和 RolesAPI 简单得多。下面是 SimpleMembership 类的一些方法：

- WebSecurity.ChangePassword
- WebSecurity.CreateUserAndAccount
- WebSecurity.Login
- WebSecurity.Logout
- WebSecurity.RequireAuthenticatedUser

与前面使用的功能相比，其基本功能没有什么新东西，但语法简单一些。例如，如果用户没有进行身份验证，RequireAuthenticatedUser 方法就自动重定向到登录页面。

Matthew M. Osborn 就这个主题写了一篇博客(<http://blog.osbornm.com/archive/2010/07/21/using-simplmembership-with-asp.net-webpages.aspx>)。Jon Galloway 在 <http://weblogs.asp.net/jgalloway/archive/2012/08/29/simplmembership-membership-providers-universal-providers-and-the-new-asp-net-4-5-web-forms-and-asp-net-mvc-4-templates.aspx> 上提供了更多细节和技术背景。

在当前处理的合作项目中，很少看到简单的 API，但它们很有趣，可能适合某些项目，所以这里提及了它们。

19.4 使用 Web Site Administration Tool

本章介绍的许多操作也可以使用 Web Site Administration Tool 完成,如图 19-25 所示。选择 Visual Studio 2012 菜单中的 Website | ASP.NET Configuration 命令,就可以使用 ASP.NET Web Site Administration Tool。

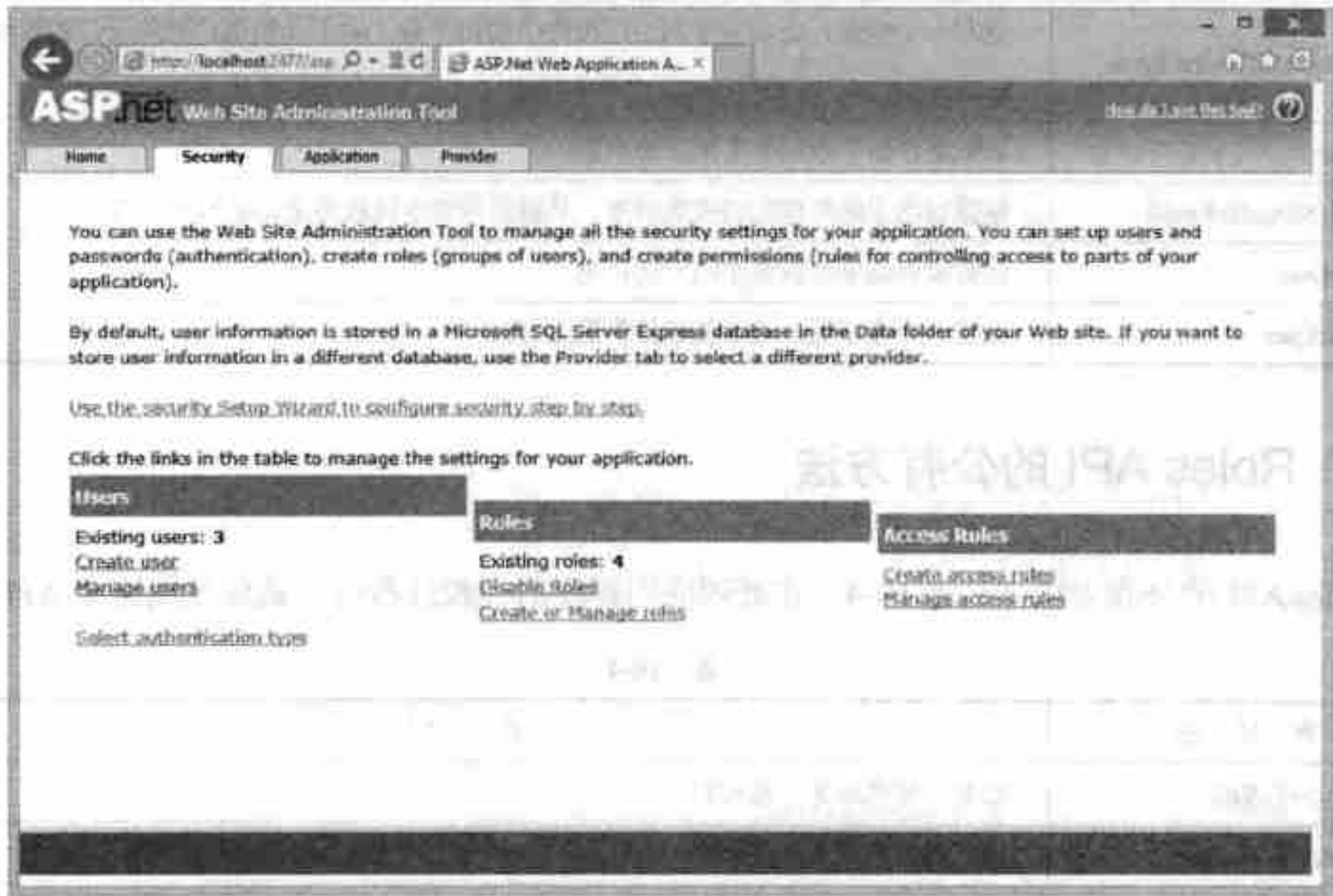


图 19-25

尽管使用这个工具可以很容易地完成所有的操作,但我们常常在应用程序中执行这些操作。在编写 ASP.NET 应用程序时,了解所有可能的方式是非常重要的。

有关 Web Site Administration Tool 的讨论,详见附录 D。

19.5 Membership API 的公有方法

Membership API 的公有方法详见表 19-3。在进行应用程序的身份验证过程时,就要使用这个 API。

表 19-3

方 法 名	说 明
CreateUser	在指定的数据存储中添加新用户
DeleteUser	从数据存储中删除指定的用户

(续表)

方 法 名	说 明
FindUsersByEmail	返回一个用户集合，这些用户的电子邮件地址与给定的电子邮件地址相匹配
FindUsersByName	返回一个用户集合，这些用户的用户名与给定的用户名相匹配
GeneratePassword	生成指定长度的随机密码
GetAllUsers	返回数据存储中包含的所有用户的集合
GetNumberOfUsersOnline	返回一个整数，表示登录到应用程序中的用户数。用于计数用户数的窗口的时间限制在 <code>machine.config</code> 或 <code>web.config</code> 文件中指定
GetUser	从数据存储中返回某个用户的信息
GetUserNameByEmail	根据对电子邮件地址的搜索结果，从数据存储中提取特定记录的用户名
UpdateUser	在数据存储中更新某个用户的信息
ValidateUser	返回一个布尔值，表示某组凭据是否有效

19.6 Roles API 的公有方法

Roles API 的公有方法详见表 19-4。在处理应用程序的授权过程时，就要使用这个 API。

表 19-4

方 法 名	说 明
AddUsersToRole	给某个角色添加一组用户
AddUsersToRoles	给一组角色添加一组用户
AddUserToRole	给某个角色添加某个用户
AddUserToRoles	给一组角色添加某个用户
CreateRole	给指定的数据存储添加新角色
DeleteCookie	删除客户端计算机上用于存储用户所属角色的 cookie
DeleteRole	从数据存储中删除某个角色。使用这个方法的相应参数，还可以控制无论角色是否包含用户时是否删除角色
FindUsersInRole	返回一组用户，他们的用户名匹配给定的用户名
GetAllRoles	返回数据存储中存储的所有角色的集合
GetRolesForUser	返回包含某个用户的角色集合
IsUserInRole	返回一个布尔值，表示用户是否包含在某个角色中
RemoveUserFromRole	从某个角色中删除某个用户
RemoveUserFromRoles	从一组角色中删除某个用户
RemoveUsersFromRole	从某个角色中删除一组用户
RemoveUsersFromRoles	从一组角色中删除一组用户
RoleExists	返回一个布尔值，表示数据存储中是否有某个角色

19.7 集成 OAuth 和 OpenID 身份验证

前面提到, ASP.NET 的身份验证系统非常强大, 由于有了提供程序模型和通用提供程序(参见第 18 章)等方法, 该系统非常灵活。而且, 它变得越来越好了。许多网站在验证身份时求助于开放标准。为此, 最常用的两种方式是:

- **OpenID:** 由 OpenID Foundation (<http://openid.net/foundation>) 创建。一般方式是用户用身份提供程序注册, 例如 Google 或 Yahoo! 的提供程序。接着, 这些用户想登录到 OpenID 支持的任意第三方网站时, 就会重定向到提供程序的站点, 在那里登录, 得到标记, 接着就可以使用标记在第三方网站上验证身份。
- **OAuth** 由 Internet Engineering Task Force (IETF, <http://ietf.org/>) 控制。身份验证过程类似于 OpenID, 但 OAuth 不进行真正的身份验证, 而是给用户提供访问标记而不是身份凭证的证明。



维基百科上的 OAuth 文章很好地描述了 OpenID 和 OAuth 之间的区别, 网址是 http://en.wikipedia.org/wiki/OAuth#OpenID_vs._pseudo-authentication_using_OAuth。

幸好, 标准的细节不那么重要, 因为下面介绍一种非常方便的方式, 将这些标准植入可信的、真正的 ASP.NET 身份验证系统中。

19.7.1 使用 OpenID

首先是坏消息: ASP.NET 没有内置对 OpenID 或 OAuth 的支持。但来自微软的外部包可以非常容易地实现这些标准。本节讨论 OpenID, 不然, 从 ASP.NET API 的角度来看, 它与 OAuth 区别很大。Google 的 OpenID 服务很容易实现。使用 OpenID 和 OAuth 的最方便方式是使用 Visual Studio 2012 附带的 ASP.NET Web Forms Application 模板, 如图 19-26 所示。

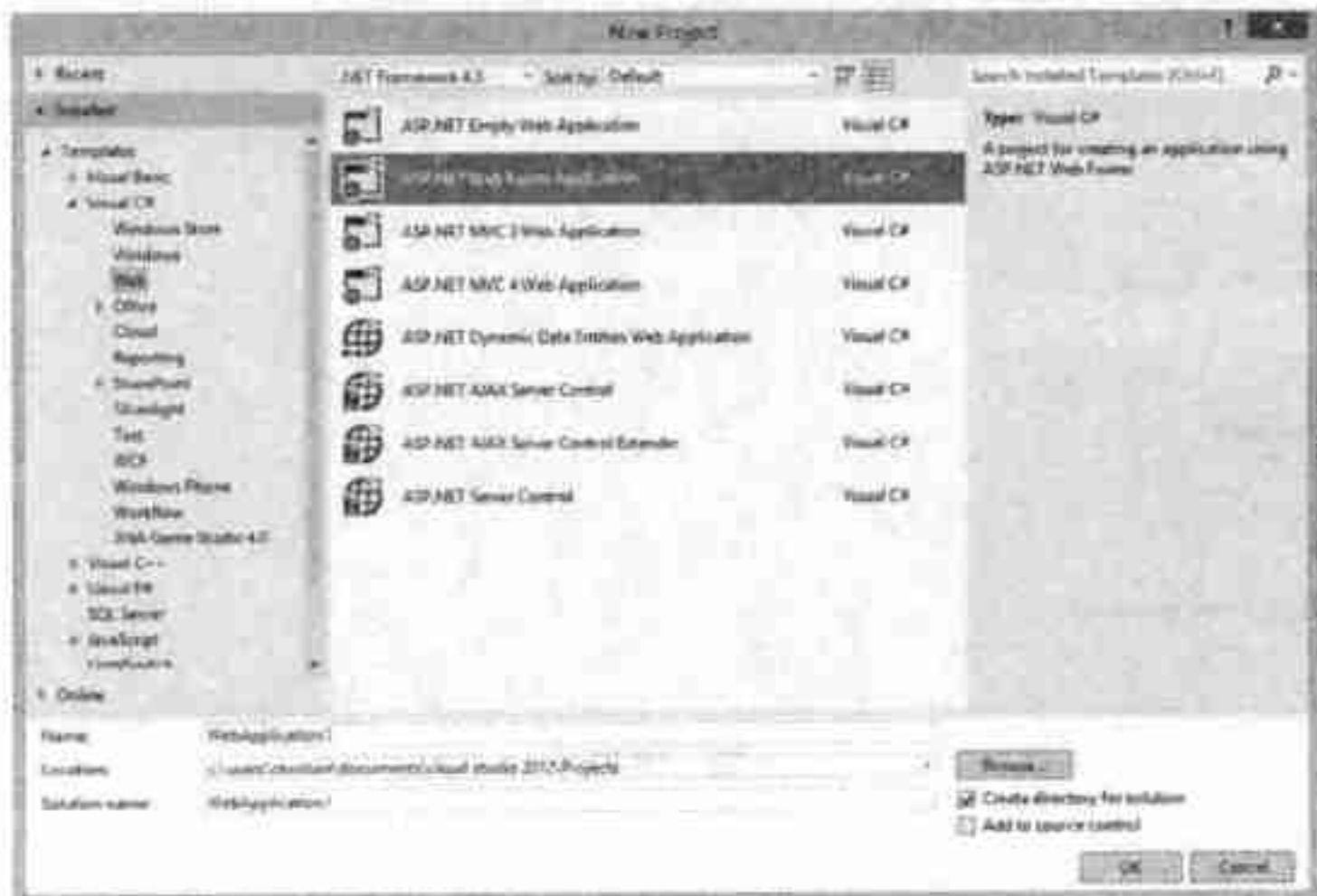


图 19-26

用这个模板创建一个新的 Web 应用程序, 查看 App_Start/AuthConfig.cs 文件。其中几行被注释了的代码行提示这个模板可以做什么。取消对调用 `OpenAuth.AuthenticationClients.AddGoogle()` 的代码的注释。程序清单 19-37 显示了相关的部分。

程序清单 19-37 在 AuthConfig.cs 中激活 OpenID 和 OAuth

```
//OpenAuth.AuthenticationClients.AddTwitter(  
//    consumerKey: "your Twitter consumer key",  
//    consumerSecret: "your Twitter consumer secret");  
  
//OpenAuth.AuthenticationClients.AddFacebook(  
//    appId: "your Facebook app id",  
//    appSecret: "your Facebook app secret");  
  
//OpenAuth.AuthenticationClients.AddMicrosoft(  
//    clientId: "your Microsoft account client id",  
//    clientSecret: "your Microsoft account client secret");  
  
OpenAuth.AuthenticationClients.AddGoogle();
```

接着, 运行应用程序, 单击右上角的登录链接, 就会显示如图 19-27 所示的登录屏幕。在该图中, 以按钮的形式提供了几个登录机制(Facebook、Twitter、Microsoft、Google), 但在读者的系统中只有 Google 是可用的(这很好, 因为它是唯一一个不需要进一步配置就能工作的提供程序)。

单击 Google 按钮, 就会重定向到 Google 站点(看看图 19-28 中的 URL), 该站点要求提供 Google 凭证。

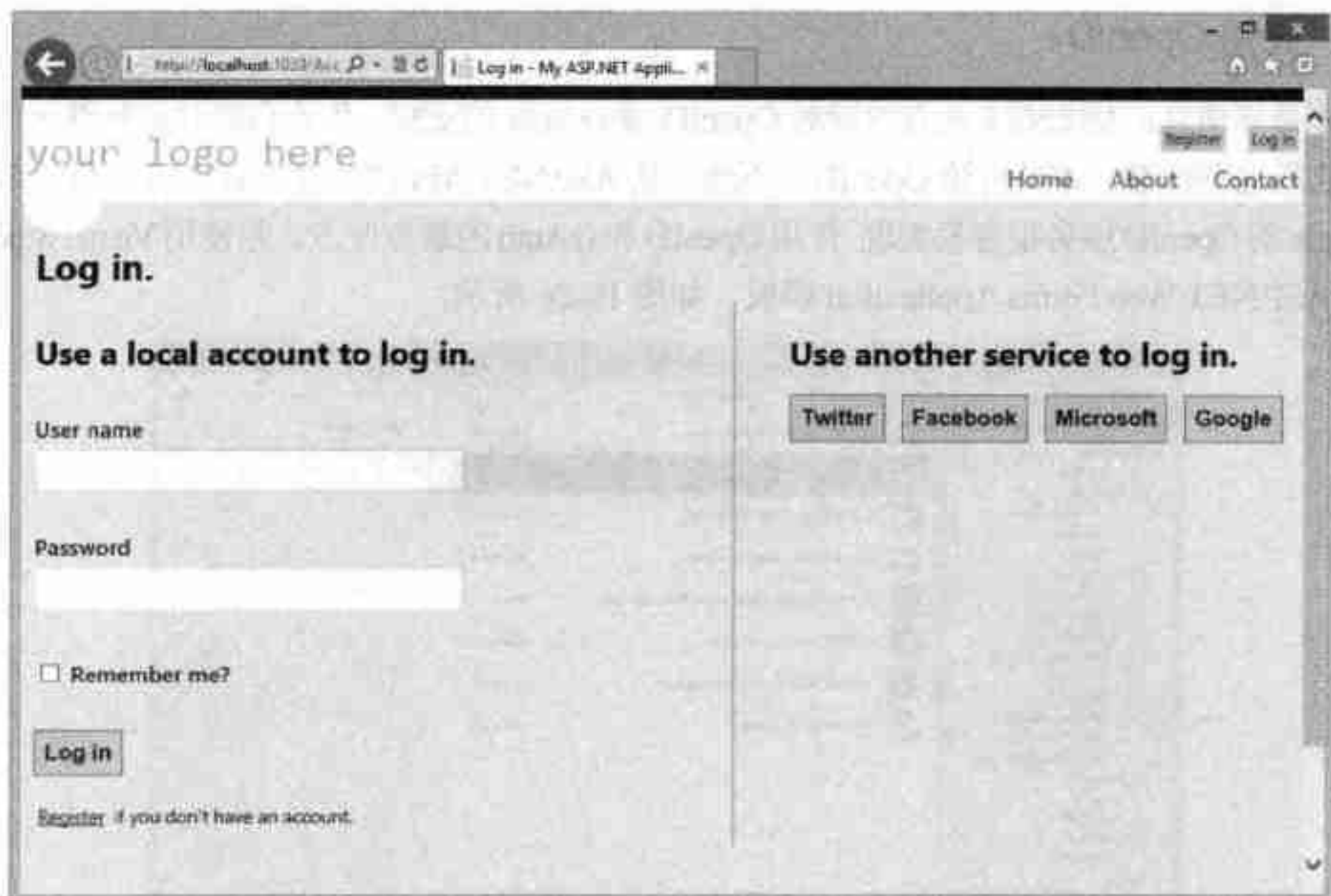


图 19-27



图 19-28

登录到 Google 后, Google 站点(不是 ASP.NET 站点)询问, Localhost 应用程序能否访问你的数据,尤其是你的电子邮件地址,如图 19-29 所示。同意访问后,就重定向回 Google 地址显示的 ASP.NET 应用程序,并要求选择用户名,如图 19-30 所示。

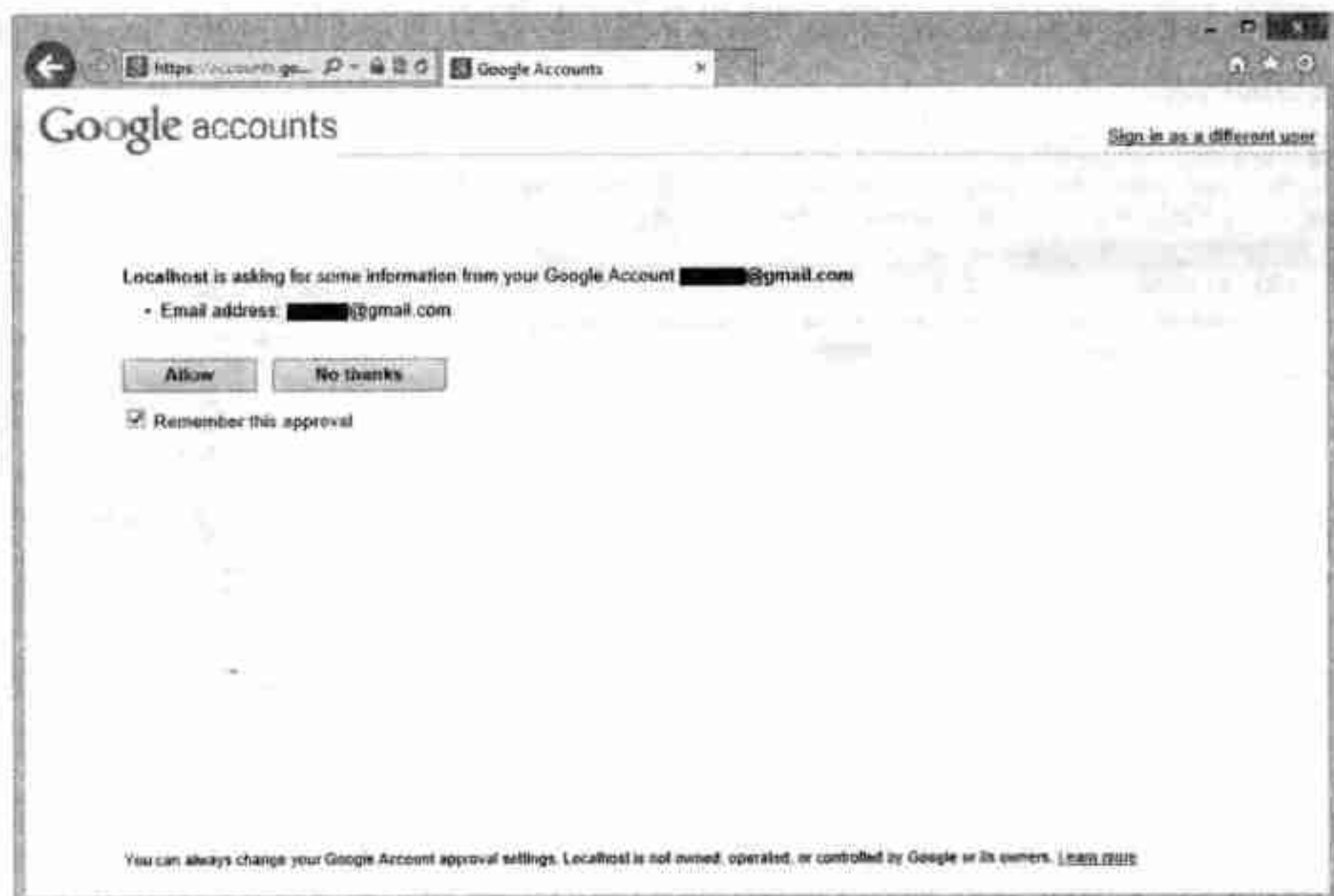


图 19-29

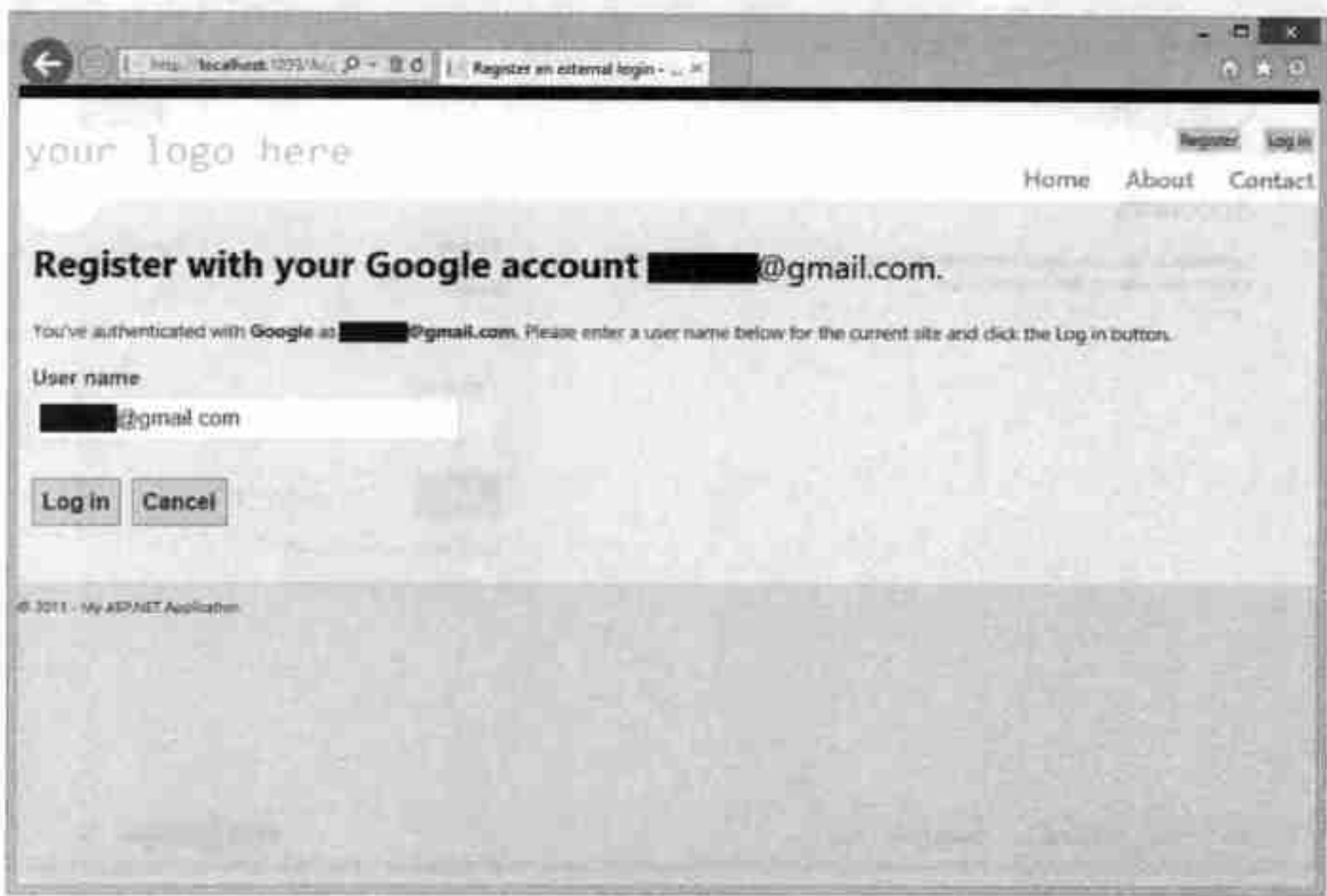


图 19-30

选择用户名后,就看看应用程序默认使用的数据库,其中有两个新表 UsersOpenAuthAccounts 和 UsersOpenAuthData。前者用于存储给什么用户使用 OpenID 还是 OAuth 提供程序,后者把 OpenID/OAuth 数据链接到 ASP.NET 的成员资格系统。图 19-31 显示了用户使用 Google 的 OpenID 服务登录后的结果。

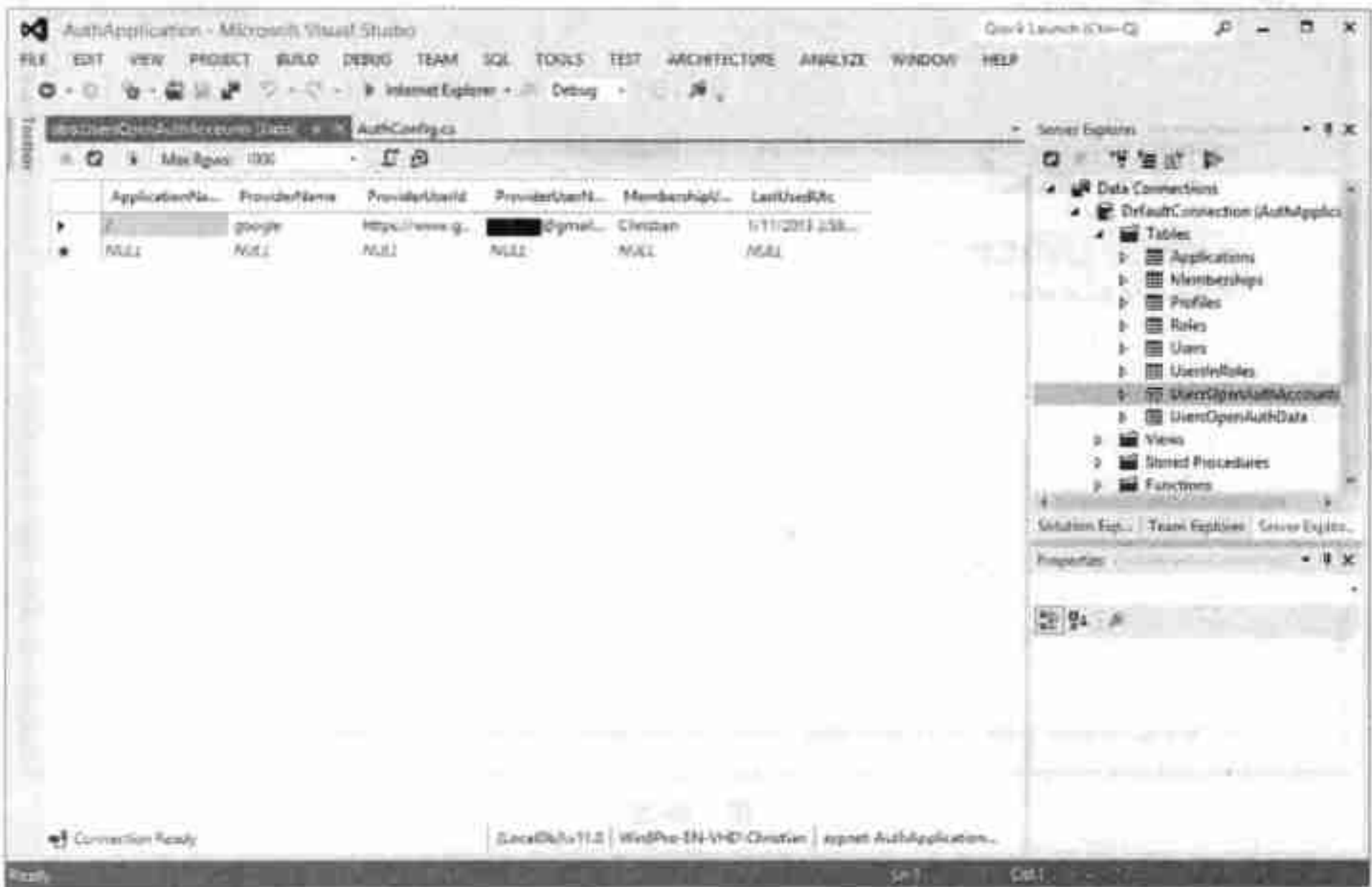


图 19-31

如果用 Visual Studio 附带的模板建立应用程序,就已经有了 Google OpenID 支持。在后台工作的 API 其实很简单,后面会讨论,但这次使用 OAuth 提供程序(不然,从 ASP.NET API 的角度看会带来很大的差异)。

19.7.2 使用 OAuth

读者可能希望使用 Facebook 登录到本章一直使用的 ASP.NET 网站。为此,首先需要用 Facebook 创建一个应用程序,再配置 ASP.NET 站点,与这个应用程序通信。所以如果还没有 Facebook 账户,就注册一个。然后进入 Facebook 开发人员站点(<https://developers.facebook.com/>),看看在访问它之前,是否需要执行额外的注册步骤。在 <https://developers.facebook.com/apps> 上创建一个新的 Facebook 应用程序。选择一个唯一的名字(Facebook 会指出该名字是否可用),如图 19-32 所示。但注意,本章使用的应用程序名显然已被他人采用了。



图 19-32

在所得到的页面的顶部有两条重要的信息应与他人共享(本章的下载代码不包含这些信息):应用程序 ID 和应用程序密钥,把它们写下来以备使用。

现在寻找“Select how your app integrates with Facebook”部分,选择 Website with Facebook Login 选项。需要提供页面的 URL,用户用 Facebook 验证身份后会加载该页面。本章是下载代码中的 Listing19-41.aspx,但你的 URL 可能与此不同。还要注意 ASP.NET 使用的端口号是随意的,所以如果端口改变了,就需要修改 Facebook 应用程序的设置。图 19-33 显示了配置屏幕。



有时很难使用 localhost 作为服务器名称,有些 OAuth 服务甚至禁止使用它。一种简单、有效的解决方法是使用实际指向本地机器的特定域名。localtest.me 服务提供了这个功能,更多信息可以参阅 <http://readme.localtest.me/>。



图 19-33

现在返回 ASP.NET: 如果希望在未使用 Visual Studio 功能完整的模板创建的网站或 Web 应用程序中使用 OpenID 或 OAuth, 就必须先添加身份验证功能。加载 ASP.NET 网站或应用程序时, 启动 Packet Manager Console, 安装两个包:

- DotNetOpenAuth.AspNet (它实现了 OpenID/OAuth 标准)
- Microsoft.AspNet.Membership.OpenAuth (它集成了 OpenID/OAuth 和 ASP.NET 的成员资格提供程序)

程序清单 19-38 显示了需要的命令。

程序清单 19-38 安装 NuGet 包以支持 OpenID/OAuth

```
Install-Package DotNetOpenAuth.AspNet
Install-Package Microsoft.AspNet.Membership.OpenAuth
```

安装后, 就需要建立应用程序或网站, 以激活一个或多个 OpenID 或 OAuth 提供程序。最佳方式是在应用程序启动时激活。创建 Global.asax 文件, 使之包含程序清单 19-39 中的代码。确保输入刚才建立的 Facebook 应用程序的 ID 和密钥。

程序清单 19-39 在 Global.asax 中激活 Facebook OAuth

```
<%@ Application Language="C#" %>
<%@ Import Namespace="Microsoft.AspNet.Membership.OpenAuth" %>

<script runat="server">
    void Application_Start(object sender, EventArgs e)
    {
        OpenAuth.AuthenticationClients.AddFacebook(
            appId: "** use your own **",
            appSecret: "** use your own **");
    }
</script>
```


在继续之前，一定要先简要讨论下面的过程。如果希望用 Facebook 登录，就必须先把用户重定向到 Facebook 验证页面，该页面会把用户重定向到 Facebook 应用程序设置提供的 URL(这里是服务器上的 Listing 19-41.aspx)。当然，不必自己寻找 Facebook URL，前面安装的 NuGet 包完成了大多数工作。程序清单 19-40 显示了新的登录页面(该页面最终必须替代项目中的 Login.aspx，因为 Login.aspx 是站点中未验证身份的用户唯一可以访问的页面)。注意添加一个按钮，单击它就会调用 `OpenAuth.RequestAuthentication` 方法，进行 Facebook 身份验证，再重定向到 Listing 19-41.aspx。

程序清单 19-40 给站点添加 Facebook 登录

```
<%@ Page Language="C#" %>
<%@ Import Namespace="Microsoft.AspNet.Membership.OpenAuth" %>

<script runat="server">

    protected void Button1_Click(object sender, EventArgs e)
    {
        OpenAuth.RequestAuthentication("facebook", "~/Listing 19-41.aspx");
    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Login Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Login ID="Login1" runat="server">
        </asp:Login>
        <asp:Button ID="Button1" Text="Login with Facebook"
            runat="server" OnClick="Button1_Click" />
    </form>
</body>
</html>
```

图 19-34 显示了新的登录屏幕，图 19-35 显示了单击新按钮后的情形：浏览器重定向到 Facebook，用户需要同意应用程序访问他的电子邮件地址。



图 19-34

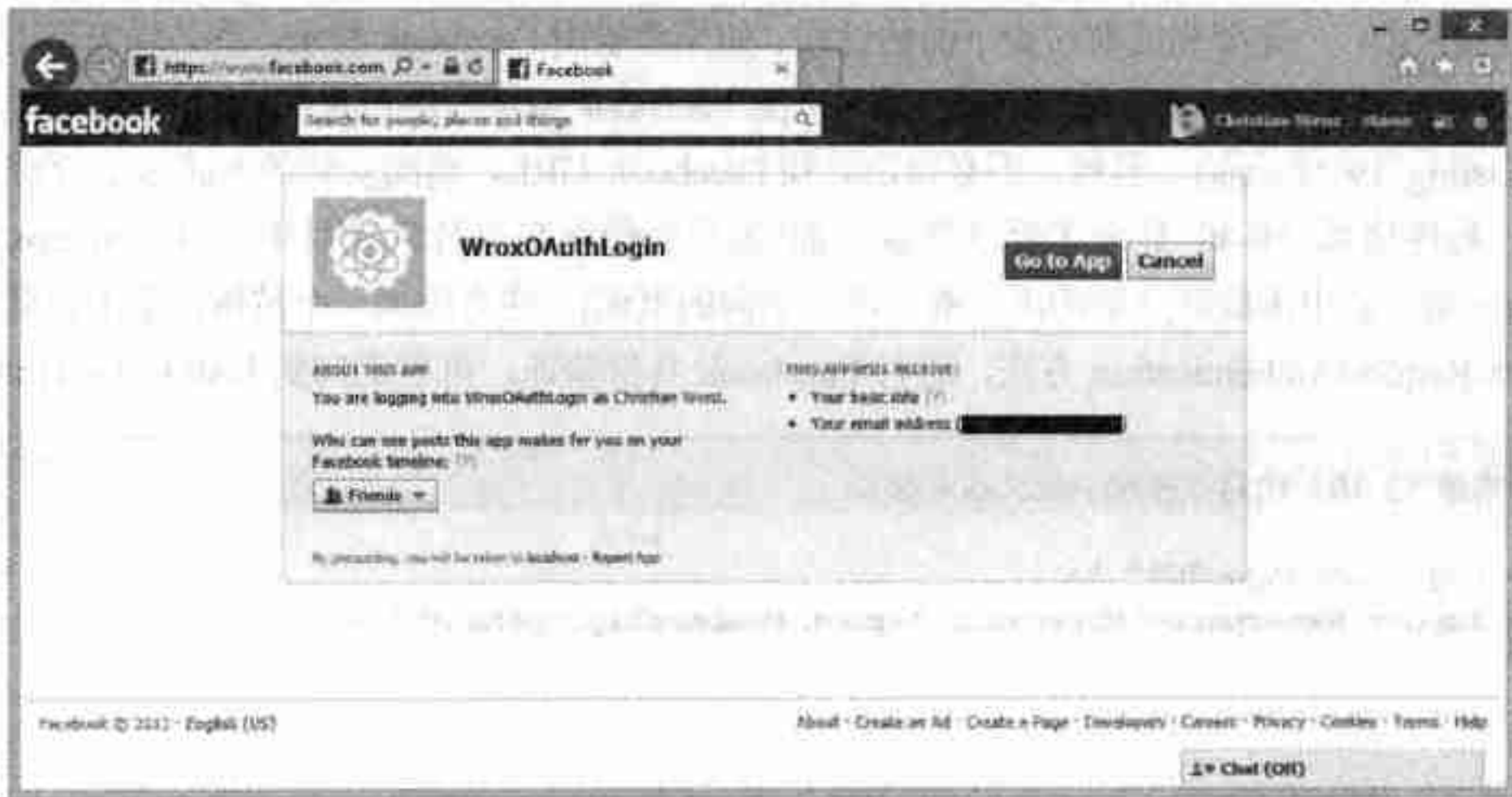


图 19-35

剩下的是实现用户从 Facebook OAuth 验证站点返回时调用的页面。前面安装的 Microsoft.AspNet.Membership.OpenAuth 包提供了许多有趣的功能，但这里最适合的 API 调用是 `OpenAuth.VerifyAuthentication()`，必须给该调用传送(当前)回调页面的 URL 作为参数。这个调用的返回值是 `AuthenticationResult` 类型。其 `IsSuccessful` 属性指出身份验证是否成功，如果成功，`UserName` 属性就包含 OAuth 提供程序中的用户名，这里是运行 Facebook 账户的电子邮件地址。

程序清单 19-41 把这些部分组合在一起，如果验证成功，就显示用户名，否则就重定向回登录页面。图 19-36 显示了成功时的结果。

程序清单 19-41 处理验证结果

```
<%@ Page Language="C#" %>
<%@ Import Namespace="Microsoft.AspNet.Membership.OpenAuth" %>

<script runat="server">
    void Page_Load()
    {
        var result = OpenAuth.VerifyAuthentication("~/Listing%2019-41.aspx");
        if(result.IsSuccessful)
        {
            Label1.Text = HttpUtility.HtmlEncode(result.UserName);
        }
        else
        {
            FormsAuthentication.RedirectToLoginPage();
        }
    }
</script>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
```

```
</head>
<body>
  <form id="form1" runat="server">
    <div>
      Welcome, <asp:Label ID="Label1" runat="server" />
    </div>
  </form>
</body>
</html>
```



图 19-36



本书的一位作者 Pranav Rastogi 在 ASP.NET 对 OpenID 和 OAuth 的支持方面发表了许多博客文章，建议读者阅读这些文章。网址是 <http://blogs.msdn.com/b/webdev/archive/2012/08/19/oauth-openid-support-for-webforms-mvc-and-webpages.aspx>。在文章的末尾有一些到其他相关帖子的链接。另一颇受关注的博客是 <http://blogs.msdn.com/b/webdev/archive/2012/09/12/integrate-openauth-openid-with-your-existing-asp-net-application-using-universal-providers.aspx>，上面集中了 OpenID、OAuth 和通用提供程序。

19.8 本章小结

本章介绍了 ASP.NET 4.5 中两个杰出的功能。成员资格服务和角色管理服务是 ASP.NET 的一部分，它们大大简化了对用户及其角色的管理。

本章讨论了 Membership 和 Roles API，以及利用这些 API 的控件。这些控件和 API 与 ASP.NET 的其他部分一样，也使用相同的数据提供程序模型。本章的例子使用 SQL Server Express Edition 进行后台存储，也可以配置这些系统，使用其他类型的数据存储。

使用 OpenID 和 OAuth 等标准也可以连接外部的验证服务，例如 Facebook、Twitter 和 Google 提供的验证服务。

第20章

安全性

本章要点

- 使用基于 Windows 和基于表单的身份验证
- 管理编程授权
- 通过 IIS 进行保护

使用 ASP.NET 建立的每个页面并不对 Internet 上的每个人都开放。有时，应用程序中的某些页面或部分只能由一组拥有访问权限的用户访问。因此，需要采取本章介绍的安全措施，以保护应用程序中的数据，防止应用程序被误用。

安全性是一个含义非常广泛的术语。毫无疑问的是，在建立应用程序的每一步中，都必须仔细考虑怀有恶意的终端用户可能试图绕过安全措施的方式。必须采取措施，确保没有人能接管应用程序或访问其资源。无论是涉及使用基本服务器控件还是访问数据库，都应考虑采取安全保护措施。

安全措施如何应用到应用程序是一个需要仔细斟酌的过程。例如，Internet 上一个对公众开放的 ASP.NET 页面与只能由选定的个人使用的 ASP.NET 应用程序相比，会有不同的安全要求，因为该 ASP.NET 页面要处理诸如信用卡号或医疗信息等机密信息。

第一步是给手边的任务应用相应的安全措施。我们可以采取许多不同的措施来保护应用程序和资源，因此必须决定采用哪些措施。本章将介绍保护应用程序的一些措施。

注意，本书将通篇讨论安全性。本书还有两章将讨论本章未探讨的 ASP.NET 提供的特定安全性架构。第 18 和 19 章讨论了 ASP.NET 的成员资格和角色管理架构，以及这个版本中的个性化功能。这些主题都是安全性的内容，大大简化了安全应用程序的建立。尽管这些安全性架构是 ASP.NET 最新版本提供的架构，但我们仍然可以像在 ASP.NET 以前版本中那样建立自己的安全措施。本章就将讨论这些内容。

安全性的一个重要方面是如何处理身份验证和对应用程序中资源的访问授权。在开始使用 ASP.NET 中的身份验证和授权功能之前，应首先了解这两个术语的含义：

- 如第 19 章所述，身份验证是确定用户身份的过程。在用户通过身份验证后，开发人员就可以确定该用户是否有权继续操作。如果没有进行身份验证，就不能进行实体授权。

- 授权过程如下：确定验证过的用户是否有权访问应用程序的某个部分、某个点，或者只有权访问应用程序提供的特定数据集。对用户和组进行身份验证和授权后，就可以根据用户的类型和喜好来定制站点。

20.1 应用身份验证措施

ASP.NET 提供了许多不同类型的身份验证措施以供应用程序使用，包括基本身份验证、摘要身份验证、表单身份验证和 Windows 集成验证，还可以开发自己的验证方法。如果没有给资源请求应用验证过程，那么绝对不要授予对资源的访问权限。

不同的身份验证模式是通过设置来建立的，而这些设置可以应用于应用程序的 web.config 文件，或者与应用服务器的 IIS 实例结合使用。

可通过应用服务器上的一系列.config 文件来配置 ASP.NET。它们是基于 XML 的文件，很容易改变 ASP.NET 的操作方式。这是操作所需配置设置的理想方式。以层次结构的方式应用 ASP.NET 配置文件。.NET Framework 提供了服务器级别的配置文件 machine.config，位于 C:\Windows\Microsoft .NET\Framework\v4.0.xxxx\CONFIG 文件夹中。这个文件提供了服务器级别的 ASP.NET 应用程序设置，也就是说，这些设置应用于服务器上的每个 ASP.NET 应用程序。

web.config 文件是另一个基于 XML 的配置文件，位于 Web 应用程序的根目录下。web.config 文件中的设置会重写上一级的 machine.config 文件中的设置。

甚至还可以嵌套 web.config 文件，从而主应用程序的 web.config 文件位于应用程序的根目录下，而其他 web.config 文件位于应用程序的子目录下，如图 20-1 所示。子目录中的 web.config 文件会重写根目录下的 web.config 文件。因此，子目录下的 web.config 文件中的设置会改变应用程序的主 web.config 文件中的设置。

在本章的许多例子中，都使用 web.config 文件在应用程序中应用需要的身份验证和授权机制，还可以使用 IIS 直接把这些设置应用于应用程序。

IIS 是处理所有入站 HTTP 请求的 Web 服务器。必须修改 IIS，才能执行希望的操作。只有当页面有特定的文件扩展名(例如.aspx)时，IIS 才把请求发送给 ASP.NET 引擎。本章后面将介绍如何使用 IIS 7.x 和 8。



图 20-1

20.1.1 <authentication>节点

在应用程序的 web.config 文件中使用<authentication>节点，可以设置 ASP.NET 应用程序需要的身份验证类型：

```

<system.web>
  <authentication mode="Windows|Forms|Passport|None">

  </authentication>
</system.web>

```


<authentication>节点使用 mode 特性设置要使用的身份验证模式，其选项包括 Windows、Forms、Passport 和 None。表 20-1 解释了这些选项。

表 20-1

可 选 项	说 明
Windows	Windows 身份验证与 IIS 身份验证一起使用。IIS 以如下方式进行身份验证：基本验证、摘要验证或集成的 Windows 验证。IIS 身份验证完成后，ASP.NET 就使用已验证的身份授予访问权限。这是默认设置
Forms	使用 HTTP 客户端重定向功能将未通过验证的请求重定向到一个 HTML 表单。用户要提供登录信息，并提交该表单。如果应用程序验证该请求，系统就发送一个表单，该表单包含用于重新获得身份的凭据或密钥
Passport	微软提供的集中式身份验证服务，为成员站点提供登录和核心配置信息服务。微软从 2005 年末开始不再推荐这种身份验证模式
None	不使用任何身份验证模式

我们可以使用两种方法为 ASP.NET 应用程序建立身份验证和授权模型。下面首先介绍 Windows 身份验证模式。

20.1.2 基于 Windows 的身份验证

基于 Windows 的身份验证在 ASP.NET 应用程序所在的 Windows 服务器和客户端计算机之间处理。在基于 Windows 的身份验证模型中，请求直接发送给 IIS，以进行身份验证过程。这种类型的身份验证在内联网环境中非常有用。在该环境下，可以让服务器完全处理身份验证过程，尤其是当用户已登录到网络时，只需获取并利用已有的凭据完成身份验证过程。

IIS 首先从域登录中获得用户的凭据。如果这个过程失败，IIS 就显示一个弹出对话框，用户可以在其中输入或重新输入登录信息。要让 ASP.NET 应用程序使用基于 Windows 的身份验证，首先要创建一些用户和组。

1. 创建用户

使用基于 Windows 的身份验证可以让已提供域登录的特定用户访问应用程序或其中的一部分。由于使用这种类型的身份验证模式，因此 ASP.NET 很容易处理在内联网环境下部署的应用程序。如果用户在本地计算机上登录为域用户，那么在访问该域中的网络计算机时就不需要再次验证身份。

下面的步骤说明了如何创建用户。注意，必须有足够的权限才能在服务器上创建用户。如果被授予该权限，创建用户的步骤就如下所示：

(1) 如果使用 Windows 7 或 8，就选择 Start | Control Panel | System and Security | Administrative Tools | Computer Management 命令。如果使用 Windows Server 2003、2008 或 2012 服务器，就选择 Start | Control Panel | Administrative Tools | Computer Management 命令。这两个操作都会打开 Computer Management 实用程序，该实用程序管理和控制本地 Web 服务器上的资源。可以使用这个实用程序完成许多任务，但这里主要考虑用户的创建。

- (2) 展开 System Tools 节点。
- (3) 展开 Local Users and Groups 节点。
- (4) 选择 Users 文件夹, 结果如图 20-2 所示。

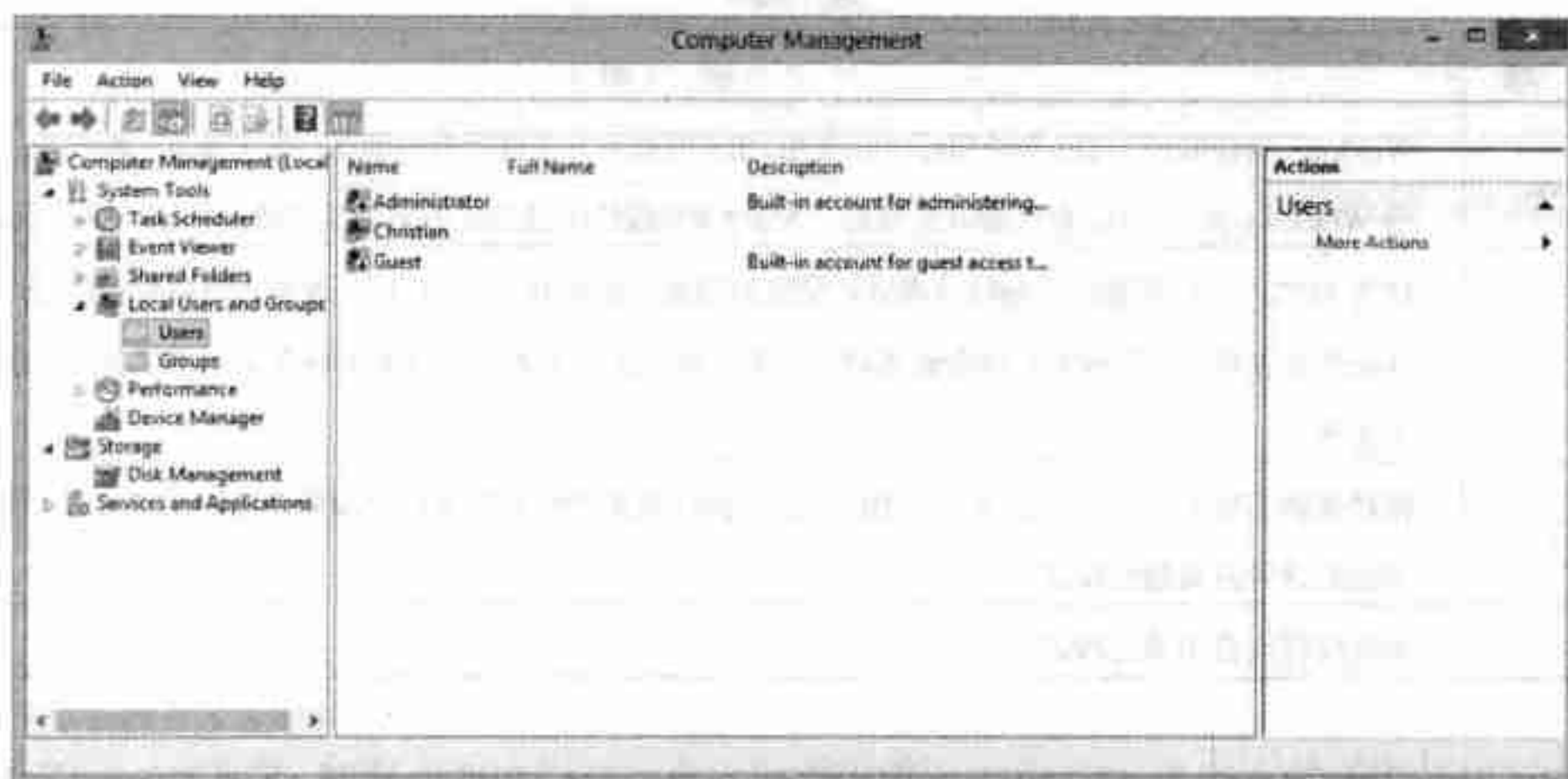


图 20-2

- (5) 右击 Users 文件夹, 选择 New User 命令, 打开 New User 对话框, 如图 20-3 所示。



图 20-3

- (6) 给用户指定名称、密码和描述(说明这是测试用户)。在这个例子中, 把用户命名为 Bubbles。
- (7) 取消选中需要用户在下次登录时修改密码的复选框。
- (8) 单击 Create 按钮, 创建的测试用户会显示在 Computer Management 实用程序的 Users 文件夹中, 如图 20-4 所示。



图 20-4

下面创建页面来使用这个用户。

2. 用户的身份验证和授权

现在创建一个应用程序，让用户可以进入该应用程序。我们使用应用程序的 `web.config` 文件控制哪些用户可以访问站点，哪些用户不可以访问站点。

把程序清单 20-1 中的代码添加到 `web.config` 文件中。

程序清单 20-1 通过 `web.config` 文件拒绝所有用户

```
<system.web>
  <authentication mode="Windows" />
  <authorization>
    <deny users="*" />
  </authorization>
</system.web>
```

在这个例子中，`web.config` 文件通过 `<authentication>` 元素的 `mode` 特性，把应用程序配置为使用基于 Windows 的身份验证。另外，使用 `<authorization>` 元素定义允许访问应用程序的用户或组的特定信息。在本例中，`<deny>` 元素指定拒绝所有用户(即使他们已通过身份验证)访问应用程序。使用 `<allow>` 元素禁止特定用户访问没有任何意义，这个例子只演示 `<deny>` 元素的用法，结果如图 20-5 所示。

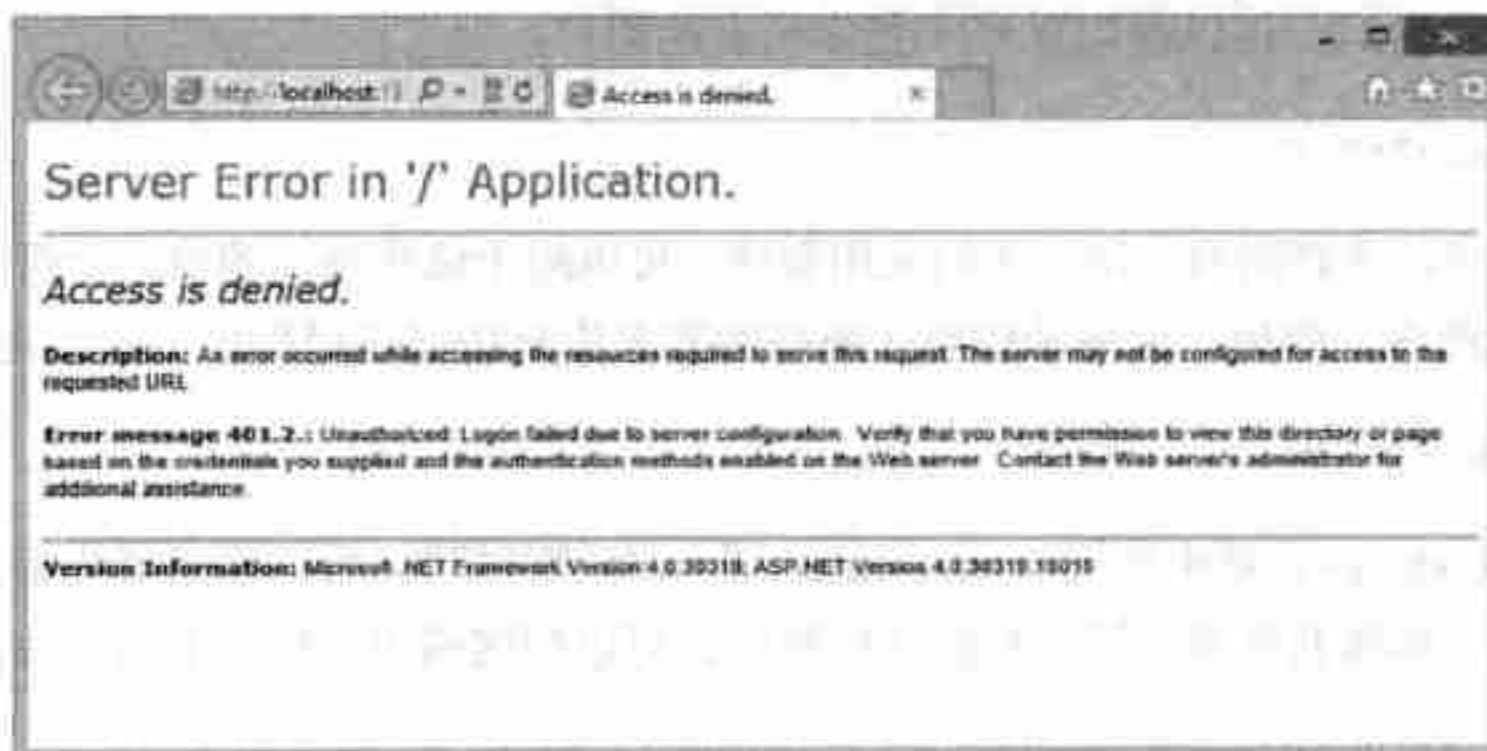


图 20-5

试图访问站点的任何终端用户，无论是已验证的用户还是未验证的用户，都会在浏览器窗口中看到“Access is denied”。我们正是希望所有的用户都不能访问应用程序。

但在大多数情况下，我们都希望允许至少一部分用户访问应用程序。在 web.config 文件中使用 <allow> 元素就可以允许某个用户访问应用程序。下面是其语法：

```
<allow users="Domain\Username" />
```

程序清单 20-2 演示了如何允许用户访问应用程序。

程序清单 20-2 通过 web.config 文件允许单个用户访问应用程序

```
<system.web>
  <authentication mode="Windows" />
  <authorization>
    <allow users="Win8Pro-En\Bubbles"/>
    <deny users="*" />
  </authorization>
</system.web>
```

即使使用 <deny> 元素拒绝所有用户(甚至是已验证的用户)访问应用程序，<allow> 元素中定义的内容也是优先的。在这个例子中，只允许用户 Bubbles 访问应用程序。

现在，如果以 Bubbles 用户身份在客户端计算机上登录，在浏览器中运行页面，就可以访问应用程序。

3. <allow>和<deny>节点详解

<allow>和<deny>节点不仅可以操作特定的用户，还可以操作组。这两个元素支持表 20-2 中列出的特性。

表 20-2

特 性	说 明
users	用于通过域和(或)名称指定用户
roles	用于指定允许或拒绝访问的访问组
verbs	用于指定允许或拒绝访问的 HTTP 传输方法

在使用这些特性时，可以使用星号(*)指定所有的用户：

```
<allow roles="*" />
```

在这个例子中，允许所有的用户访问应用程序。可以用于这些特性的另一个符号是问号(?)，它表示所有的匿名用户。例如，如果希望阻止所有的匿名用户访问应用程序，可使用下面的代码：

```
<deny users="?" />
```

在<allow>或<deny>元素中使用 users、roles 或 verbs 特性时，可以指定多个条目，并且使用逗号隔开这些条目。如果允许多个用户访问应用程序，可以把这些用户放在不同的元素中，如下代码所示：

```
<allow users="MyDomain\User1" />
```

```
<allow users="MyDomain\User2" />
```

也可以使用下面的代码:

```
<allow users="MyDomain\User1, MyDomain\User2" />
```

在定义多个角色和谓词时,使用的语法与上面的代码相同。

4. 组的身份验证和授权

可以允许或拒绝组对应用程序或应用程序资源的访问。服务器可以包含许多不同的组,每个组都可以有任意多个用户。一个用户还可以属于多个组。打开 Computer Management 实用程序,访问在服务器上定义的组列表。单击 Computer Management 实用程序中的 Groups 文件夹,就会显示组的列表,如图 20-6 所示。

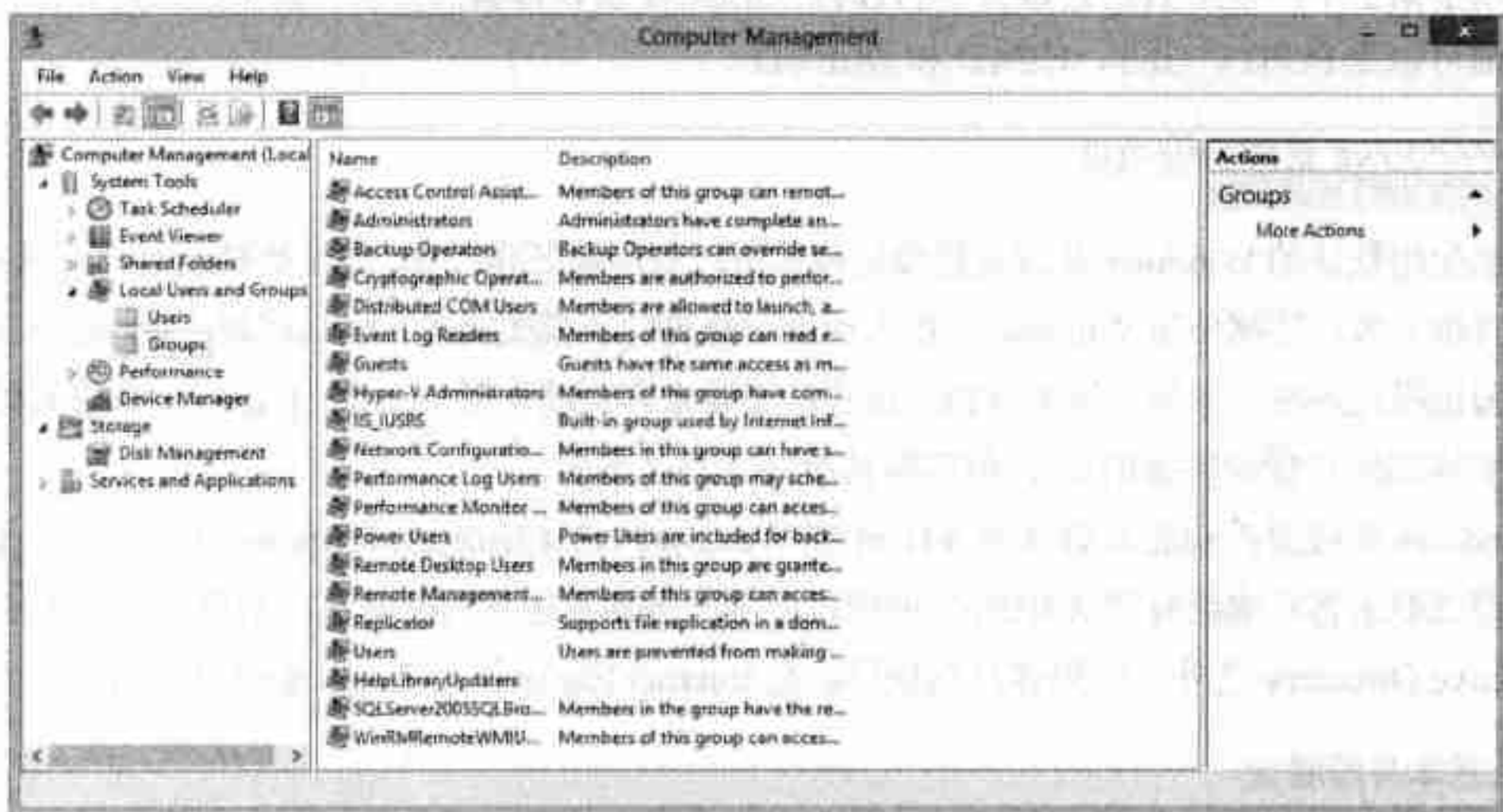


图 20-6

右击 Groups 文件夹,选择 New Group 命令,打开 New Group 对话框,如图 20-7 所示。



图 20-7

要创建组，应指定组的名称和描述；然后单击 Add 按钮，选择要成为组中成员的用户。创建组后，就可以允许组访问应用程序，如下所示：

```
<allow roles="MyGroup" />
```

可以使用<allow>或<deny>元素的 roles 特性来操作刚才创建的组或已有的组。

5. HTTP 传输方法的身份验证和授权

除了身份验证和授权特定的用户或用户组之外，还可以通过特定的 HTTP 传输协议来接受或拒绝请求。可以使用<allow>或<deny>元素的 verbs 特性实现该操作：

```
<deny verbs="GET, DEBUG" />
```

在这个例子中，使用 HTTP GET 或 HTTP DEBUG 协议传输的请求会被拒绝访问站点。Verbs 特性的值可以是 POST、GET、HEAD 和 DEBUG。

6. Windows 集成身份验证

前面使用默认的 Windows 集成身份验证模式进行身份验证和授权。如果处理的是内联网应用程序，并且每个客户端都使用 Windows，那么使用该身份验证模式就没有什么问题。Windows 是支持该身份验证模式的唯一系统。这个身份验证系统还要求客户端使用 Internet Explorer 来直接处理(如果不希望终端用户受到挑战的话)，但实际情况并非总是如此。

Windows 集成身份验证以前称为 NTLM 或 Windows NT Challenge/Response 身份验证。这个身份验证模式要求客户端给驻留 ASP.NET 应用程序的服务器发送其凭据的散列结果，以证明其身份。除了 Active Directory 之外，如果客户端使用的是 Internet Explorer，还可以使用 Kerberos。

7. 基本身份验证

另一个选项是使用基本身份验证，它也要求客户端提供用户名和密码，以验证其身份。基本身份验证的一大优点是，它是 HTTP 规范的一部分，因此大多数浏览器都支持。基本身份验证的缺点是，它把用户名和密码作为明文传送给服务器，因此用户名和密码很容易被窃取。因此，基本身份验证必须与 SSL(安全套接字层)一起使用。

如果使用的是 Windows Vista/Windows Server 2008 (IIS 7.0)、Windows 7/Windows Server 2008 R2 (IIS 7.5)或 Windows 8/Windows Server 2012 (IIS 8.0)，就不容易找到激活基本身份验证模式的选项，而必须首先激活 IIS，才能使用基本身份验证模式。为此，需要选择 Start | Control Panel | Programs | Programs and Features | Turn Windows features on or off 命令。在打开的对话框中，导航到 Internet Information Services 部分并展开，直到看到 World Wide Web Services | Security 节点为止。在其中选中 Basic Authentication 复选框，然后单击 OK 按钮进行安装。这个选项如图 20-8 所示。



图 20-8

安装完之后, 就可以返回 Internet Information Services (IIS) Manager, 为当前的虚拟目录选择 IIS 部分的 Authentication 选项。在其中突出显示 Basic Authentication 选项, 从 Actions 窗格中选择 Enable 选项, 如图 20-9 所示。



图 20-9

如果使用的是 IIS 6(或更早的版本), 为了给应用程序应用基本身份验证, 必须打开 IIS, 再打开 Web 站点的 Properties 对话框。选择 Directory Security 选项卡, 然后单击 Anonymous Access and Authentication Control 框中的 Edit 按钮, 打开 Authentication Methods 对话框。

取消选中底部的 Integrated Windows authentication 复选框, 再选中上面的 Basic authentication 复选框, 如图 20-10 所示。此时会显示警告, 说明这种模式把用户名和密码作为明文传送。

最后单击对话框中的 OK 按钮。现在应用程序使用基本身份验证模式, 而不是 Windows 集成身份验证模式。

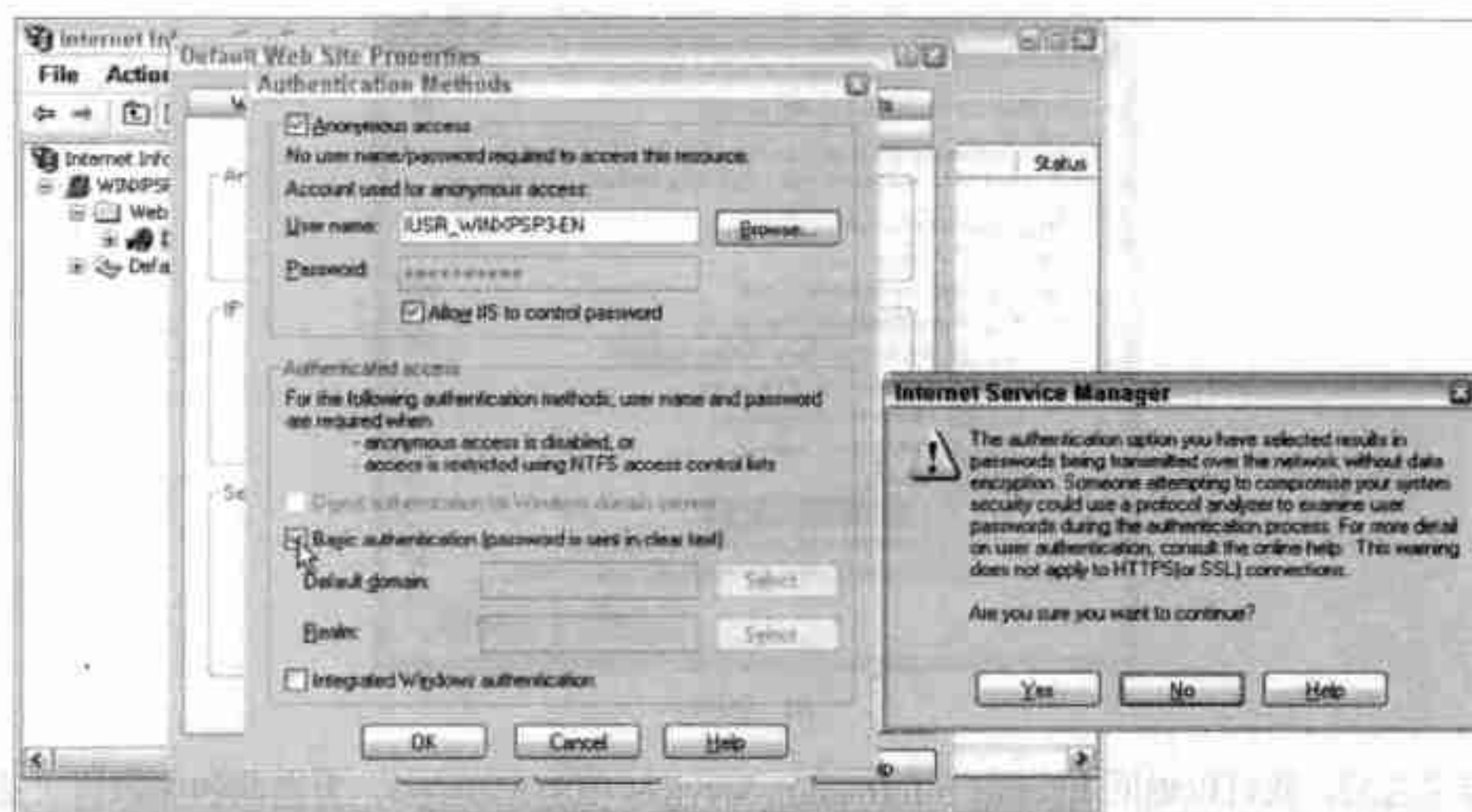


图 20-10

8. 摘要身份验证

摘要身份验证是本节介绍的最后一种身份验证模式。该模式解决了基本身份验证把客户端的凭据作为明文传送的问题。在把客户端的凭据传送给应用服务器之前，摘要身份验证使用算法加密它们。

要使用摘要身份验证，需要有 Windows 域控制器。摘要身份验证的主要问题是，不是所有的平台都支持，并且浏览器必须遵循 HTTP 1.1 规范。但摘要身份验证不仅可与防火墙一起工作，还能与代理服务器兼容。

如果使用的是 IIS 7 或更高版本，就需要像安装基本身份验证那样安装摘要身份验证。完成安装后，就可以找到该选项，并在 IIS Manager 的 Authentication 部分激活这种模式。

如果使用的是 IIS 6 或更早版本，只要在属性对话框中选中 Digest Authentication 复选框，就可以在 Authentication Methods 对话框中为应用程序选择摘要身份验证。

20.1.3 基于表单的身份验证

基于表单的身份验证是验证用户身份以允许其访问整个应用程序或其特定资源的一种流行模式。使用该模式可以把登录表单直接放在应用程序中，从而终端用户只需把用户名和密码输入到浏览器的 HTML 表单中即可。基于表单的身份验证的缺点是：用户名和密码作为明文传送，除非使用 SSL。

很容易在 Web 应用程序中实现基于表单的身份验证。首先修改应用程序的 web.config 文件，如程序清单 20-3 所示。

程序清单 20-3 为基于表单的身份验证修改 web.config 文件

```
<system.web>
  <authentication mode="Forms">
    <forms name="Wrox" loginUrl="Login.aspx" path="/" />
  </authentication>
</system.web>
```



```

</authentication>
<authorization>
  <deny users="?" />
</authorization>
</system.web>

```

这个结构必须应用于 web.config 文件。首先使用前面介绍的<authorization>元素，拒绝所有匿名用户访问应用程序。只有验证用户才能访问应用程序包含的页面。

如果请求者未通过验证，就执行<authentication>元素中定义的内容。Mode 特性的值设置为 Forms，表示 Web 应用程序使用基于表单的身份验证。下一个指定的特性是 loginUrl，指向包含应用程序的登录表单的页面。在这个例子中，该页面指定为 Login.aspx。如果试图访问应用程序的终端用户未通过身份验证，他的请求就会被重定向到 Login.aspx，以便对该用户进行身份验证和授权。在提供有效的凭据后，用户就会返回到他在应用程序中最初发出请求的地方。这里使用的最后一个特性是 path，它指定保存 cookie 的位置，cookie 用于永久保存授权用户的访问令牌。在大多数情况下，该特性的值默认为/。表 20-3 列出了<forms>元素的特性。

表 20-3

特 性	说 明
name	这是赋予 cookie 的名称，cookie 用于在请求之间保存用户。默认值是 ASPXAUTH
loginUrl	在没有找到有效的身份验证 cookie 时指定将请求重定向到的 URL。默认值是 Login.aspx
protection	指定要应用于身份验证 cookie 的保护级别。有如下 4 种设置： <ul style="list-style-type: none"> • All：应用程序使用数据有效性验证和加密机制来保护 cookie。这是默认设置 • None：不加密 cookie • Encryption：加密 cookie，但不对之进行数据有效性验证。以这种方式使用的 cookie 可能会受到纯文本攻击 • Validation：与 Encryption 设置相反，进行数据有效性验证，但不加密 cookie
path	指定应用程序所用 cookie 的路径。在大多数情况下应使用/，这是默认设置
timeout	指定 cookie 过期后的时间(分钟)，默认值是 30
cookieless	指定在进行身份验证和授权的过程中，基于表单的身份验证过程是否使用 cookie
defaultUrl	指定默认的 URL
domain	指定要与表单身份验证 cookie 一起发送的域名
slidingExpiration	指定是否给 cookie 应用可变化的过期时间。如果该特性设置为 True，就对发送给服务器的每个请求重新设置 cookie 的过期时间。默认值是 False
enableCrossAppsRedirect	指定是否允许跨应用程序的重定向
requireSSL	指定在传输身份验证信息时是否需要 SSL 连接

有了 web.config 文件后，就给应用程序创建一个可以访问的页面。程序清单 20-4 显示了一个简单的页面(本章下载代码中的 default.aspx)。

程序清单 20-4 一个简单的页面

```

<%@ Page Language="C#" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>The Application</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            Hello World
        </div>
    </form>
</body>
</html>

```

这个页面在浏览器上写入 Hello World。表单身份验证的强大功能在程序清单 20-5 所示的 Login.aspx 页面中有所体现(本章下载代码中的 Login.aspx)。

程序清单 20-5 Login.aspx 页面

```

<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        if(tbUsername.Text == "Christian" && tbPassword.Text == "Bubbles") {
            FormsAuthentication.RedirectFromLoginPage(tbUsername.Text, true);
        }
        else {
            Response.Write("Invalid credentials");
        }
    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Login Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            Username<br />
            <asp:TextBox ID="tbUsername" runat="server"></asp:TextBox><br />
            <br />
            Password<br />
            <asp:TextBox ID="tbPassword" runat="server"
                TextMode="Password"></asp:TextBox><br />
            <br />
            <asp:Button ID="Button1" OnClick="Button1_Click" runat="server"
                Text="Submit" />
        </div>
    </form>

```

```
</body>
</html>
```

Login.aspx 有两个简单的 TextBox 控件和一个 Button 控件,该页面要求用户提交用户名和密码。Button1_Click 事件使用 FormsAuthentication 类的 RedirectFromLoginPage 方法,这个方法把请求从 Login.aspx 重定向到最初请求的资源。

RedirectFromLoginPage 方法有两个参数。第一个参数是用户名,用于进行 cookie 验证。这个参数并不映射为账户名,而是由 ASP.NET 的 URL 授权功能使用。第二个参数指定是否生成永久的 cookie,如果设置为 True,终端用户在从一个浏览器会话转到下一个会话时就不需要再次登录应用程序。

使用前面构建的 3 个页面,程序清单 20-4 中对 Default.aspx 页面的每个请求都会导致 ASP.NET 检查是否存在正确的身份验证令牌。如果没有找到该令牌,请求就被重定向到指定的登录页面(本例中是 Login.aspx)。查看浏览器中的 URL,你会发现,ASP.NET 使用查询字符串值来确定在用户获得授权后将其返回到什么地方:

```
http://localhost:35089/Security/Login.aspx?ReturnUrl=%2fSecurity%2fDefault.aspx
```

其中的查询字符串 returnUrl 使用最初请求的页面和文件夹的值。

仔细查看程序清单 20-5 中的 Login.aspx 页面,注意要检查两个文本框中的值,确保它们遵循特定的用户名和密码。如果用户名和密码正确,就调用 RedirectFromLoginPage 方法,否则就使用 Response.Write() 语句。在大多数情况下,都不需要在代码中对用户名和密码硬编码。还有许多其他选项用于检查用户名和密码是否来自授权用户,下面介绍其中一些选项。

1. 根据 web.config 文件包含的值进行身份验证

上一个例子不是处理为身份验证提供的用户名和密码的最佳方式。把这些信息直接硬编码到应用程序中并不合适。下面把这些值存储在 web.config 文件中。

在程序清单 20-3 中,web.config 文件的 <forms> 元素还可以有子元素。<credentials> 子元素允许直接在 web.config 文件中指定用户名和密码组合。可以使用两种方式添加这些值。最简单的方法如程序清单 20-6 所示。

程序清单 20-6 修改 web.config 文件以添加用户名和密码值

```
<system.web>
  <authentication mode="Forms">
    <forms name="Wrox" loginUrl="Login.aspx" path="/">
      <credentials passwordFormat="Clear">
        <user name="Christian" password="Bubbles" />
      </credentials>
    </forms>
  </authentication>
  <authorization>
    <deny users="?" />
  </authorization>
</system.web>
```

<credentials> 元素在配置文件中添加了用户名及密码。<credentials> 元素有个特性 passwordFormat,

值可以是 Clear、MD5 和 SHA1。下面描述了这些选项：

- Clear: 密码存储为明文。用户的密码直接与这个值比较, 不需要进一步转换。
- MD5: 使用 MD5 散列摘要存储密码。在验证凭据时, 使用 MD5 算法散列用户密码, 再与这个值进行相等比较。从来不要存储或比较明文密码。这个算法比 SHA1 的性能好, 但不大安全。
- SHA1: 使用 SHA1 散列摘要存储密码。在验证凭据时, 使用 SHA1 算法散列用户密码, 再与这个值进行相等比较。从来不要存储或比较明文密码。这个算法的安全性最高。

在程序清单 20-6 的例子中, 使用了 Clear 设置。这不是最安全的方法, 但可用于演示。<credentials> 元素的一个子元素是 <user>, 在该子元素中, 可以使用特性 name 和 password 为授权用户定义用户名和密码。

接着修改 Login.aspx 页面的 Button1_Click 事件, 如程序清单 20-7 所示。

程序清单 20-7 修改 Login.aspx 页面以使用 web.config 文件

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        if (FormsAuthentication.Authenticate(tbUsername.Text, tbPassword.Text)) {
            FormsAuthentication.RedirectFromLoginPage(tbUsername.Text, true);
        }
        else {
            Response.Write("Invalid credentials");
        }
    }
</script>
```

在这个例子中, 使用 Authenticate 方法让 ASPNET 页面查找存储在 web.config 文件中的凭据, 以进行验证。Authenticate 方法带有两个参数: 要检查的用户名和密码。如果成功找到了凭据, 就调用 RedirectFromLoginPage 方法。



当然, 更好的方法是把用户信息存储在数据库中, 以提高性能和可维护性, 如第 19 章所述。

最好不要像上面的例子那样在 web.config 文件中把用户的密码存储为明文, 而应使用散列功能, 防止终端用户的密码被窃取。为此, 应在配置文件中存储散列的密码, 如程序清单 20-8 所示。

程序清单 20-8 使用加密的密码

```
<forms name="Wrox" loginUrl="Login.aspx" path="/">
    <credentials passwordFormat="SHA1">
        <user name="Christian" password="58356FB4CAC0B801F011B397F9DFF45ADB863892" />
    </credentials>
</forms>
```


使用这种构建方式,甚至开发人员都不知道密码,因为没有使用明文密码。Login.aspx 页面的 Authenticate 方法使用 SHA1 对密码进行散列(因为这是 web.config 文件的<credentials>节点中指定的方法),并对这两个散列进行比较。如果它们匹配,就给用户授权。

在使用 SHA1 或 MD5 时,唯一要修改的是 web.config 文件。不必修改登录页面或应用程序中的其他页面。但要存储散列的密码,应使用 FormsAuthentication.HashPasswordForStoringInConfigFile 方法(它是 .NET Framework 中最长的方法名之一),以如下方式使用该方法:

```
FormsAuthentication.HashPasswordForStoringInConfigFile(TextBox2.Text, "SHA1")
```

2. 根据数据库中的值进行身份验证

获取用户名和密码的另一常用方式是直接从数据库中获取它们。这样可以根据存储在 SQL Server 中的值检查用户输入的凭据,如程序清单 20-9 所示。

程序清单 20-9 在 SQL Server 中检查凭据

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        SqlConnection conn;
        SqlCommand cmd;
        string cmdString = @"SELECT [Password] FROM [AccessTable] WHERE
            ([Username] = @Username) AND ([Password] = @Password)";

        conn = new SqlConnection(@"Data Source=localhost;Initial
            Catalog=Northwind;Persist Security Info=True;User ID=sa");
        cmd = new SqlCommand(cmdString, conn);
        cmd.Parameters.Add("@Username", SqlDbType.VarChar, 50);
        cmd.Parameters["@Username"].Value = tbUsername.Text;
        cmd.Parameters.Add("@Password", SqlDbType.VarChar, 50);
        cmd.Parameters["@Password"].Value = tbPassword.Text;
        conn.Open();
        SqlDataReader myReader;
        myReader = cmd.ExecuteReader(CommandBehavior.CloseConnection);
        if(myReader.Read()) {
            FormsAuthentication.RedirectFromLoginPage(tbUsername.Text, false);
        }
        else {
            Response.Write("Invalid credentials");
        }
        myReader.Close();
    }
</script>
```

除了 Login.aspx 页面之外,其他内容都与前面的例子相同。现在可以根据存储在 SQL Server 中

的数据来验证用户名和密码。在 `Button1_Click` 事件中,建立了与 SQL Server 的连接(为了安全起见,应把连接字符串存储在 `web.config` 文件中)。传送两个参数,即用户在 `tbUsername` 和 `tbPassword` 中的输入。如果返回结果,就调用 `RedirectFromLoginPage` 方法。

3. 使用 Login 控件和表单身份验证

前面介绍了如何结合使用 ASP.NET 表单身份验证和标准的 ASP.NET 服务器控件,如简单的 `TextBox` 和 `Button` 控件。也可以在定制开发的表单身份验证架构中使用 ASP.NET 服务器控件,如 `Login` 服务器控件。这说明 ASP.NET 非常强大——可以组合许多不同的部分,构建出希望的解决方案。

程序清单 20-10 显示了使用 `Login` 服务器控件修改的 `Login.aspx` 页面。

程序清单 20-10 在 Login.aspx 页面上使用 Login 服务器控件

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Login1_Authenticate(object sender, AuthenticateEventArgs e)
    {
        if(Login1.UserName == "Christian" && Login1.Password == "Bubbles") {
            FormsAuthentication.RedirectFromLoginPage(Login1.UserName,
                Login1.RememberMeSet);
        }
        else {
            Response.Write("Invalid credentials");
        }
    }
}
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Login Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Login ID="Login1" runat="server" OnAuthenticate="Login1_Authenticate">
                </asp:Login>
        </div>
    </form>
</body>
</html>
```

因为页面上没有 `Button` 服务器控件,所以使用 `Login` 控件的 `OnAuthenticate` 属性指向身份验证服务器端事件 `Login1_Authenticate`。该事件查找授权信息(但本例中将其值硬编码)。Login 控件的用户名文本框可以通过 `Login1.UserName` 访问,而密码可以通过 `Login1.Password` 访问。`Login1.RememberMeSet` 属性用于指定是否存储用户的身份验证 cookie,这样在下次访问时用户就不必验证身份。

这个例子比使用 `TextBox` 和 `Button` 控件创建自己的登录表单简单得多。可以给 `Login` 控件提供

预定义的外观和操作方式,也可以使用 Login 控件的子控件属性。总之,在 ASPNET 应用程序中使用什么方法取决于我们自己。

4. FormsAuthentication 类

从本章的表单身份验证的各个示例中可以看出,许多任务的实现都取决于 FormsAuthentication 类。因此,下面就介绍这个类。

FormsAuthentication 类有许多方法和属性,可以读取和控制身份验证 cookie 以及其他信息(例如,请求的返回 URL)。表 20-4 列出了 FormsAuthentication 类的一些方法和属性。

表 20-4

方法/属性	说 明
Authenticate	这个方法用于验证存储在配置文件(例如 web.config 文件)中的凭据
Decrypt	返回一个有效的加密身份验证票证实例,从 HTTP cookie 中提取该实例,它是 FormsAuthenticationTicket 类的一个实例
Encrypt	创建一个字符串,其中包含一个有效的加密身份验证票证,该票证可用于 HTTP cookie
FormsCookieName	给当前应用程序返回 cookie 的名称
FormsCookiePath	给当前应用程序返回 cookie 的路径(cookie 的位置)
GetAuthCookie	给指定的用户提供身份验证 cookie
GetRedirectUrl	返回一个 URL,通过登录页面授权用户后,就会将该用户重定向到这个 URL
HashPasswordForStoringInConfigFile	创建所提供的字符串密码的散列。这个方法带两个参数,一个是密码;另一个是要在字符串上实现的散列类型。散列值可以是 SHA1 和 MD5
Initialize	从 web.config 文件中读取配置设置,以及获取应用程序的给定实例中使用的 cookie 和加密密钥,以此来执行 FormsAuthentication 类的初始化
RedirectFromLoginPage	把 HTTP 请求重定向回最初请求的页面。只能在用户获得授权后执行这个操作
RenewTicketIfOld	在 FormsAuthenticationTicket 实例上有条件地更新可变化的过期时间
RequireSSL	指定 cookie 是否应只通过 SSL(HTTPS)传输
SetAuthCookie	创建一个身份验证票证,把它关联到输出响应包含的 cookie
SignOut	删除身份验证票证
SlidingExpiration	提供一个布尔值,表示过期时间是否可变化

20.2 验证特定的文件和文件夹

我们不希望应用程序中的每个页面或资源都需要凭据。例如,有一个公共 Internet 站点,其中的页面即使没有凭据也可以访问,但把管理部分作为应用程序的一部分,因此需要身份验证和授权措施。

URL 授权允许使用 web.config 文件应用需要的设置。使用 URL 授权,可以给特定的文件或文件夹应用任意身份验证措施。程序清单 20-11 是一个锁定单个文件的例子。

程序清单 20-11 给单个文件应用授权要求

```

<configuration>
  <system.web>
    <authentication mode="None" />

    <!-- The rest of your web.config file settings go here -->

  </system.web>

  <location path="AdminPage.aspx">
    <system.web>
      <authentication mode="Windows" />

      <authorization>
        <allow users="Win8Pro-En\Christian" />
        <deny users="*" />
      </authorization>
    </system.web>
  </location>
</configuration>

```

这个 web.config 文件使 Web 应用程序对一般公众开放，同时锁定应用程序中的一个文件，即 AdminPage.aspx 页面。可通过<location>元素完成该操作。<location>元素包含特性 path，它指定在 web.config 文件的<system.web>部分中定义的资源。

在这个例子中，<authentication>和<authorization>元素用于为 AdminPage.aspx 页面提供身份验证和授权信息。这个页面使用 Windows 身份验证，唯一允许访问它的用户是 ReutersServer 域中的 Christian。在 web.config 文件中可以有任意多个<location>部分。

20.3 以编程方式授权

前面的许多身份验证例子都是为应用程序中的某个页面或文件夹提供一般授权，还可以对页面上的一些项提供更具体的授权措施。例如，给某个文档提供一个链接，只有明确声明为 Windows 角色的用户才能使用这个链接，而其他用户只能查看其他内容。还可以给某些用户提供其他注释或信息，而其他用户只能看到信息的压缩版本。无论如何，我们都可以在 ASP.NET 中使用某些对象实现这种基于角色的授权过程。

可以使用 Page 对象的 User 属性，它提供了 IPrincipal 对象的一个实例。User 属性有一个方法和一个属性：

- **Identity**: 这个属性提供了 System.Security.Principal.IIdentity 对象的一个实例，用于获得验证用户的一些具体属性。
- **IsInRole**: 这个方法带有一个参数，即系统角色的字符串表示。它返回一个布尔值，表示用户是否在指定的角色中。

20.3.1 使用 User.Identity 属性

User.Identity 属性可以处理已授权用户的一些上下文信息。在 ASP.NET 应用程序中使用这个属性可以根据对象提供的信息来决定能否访问资源。

使用 User.Identity 属性可以获得用户的名称、身份验证类型，以及该用户是否已通过身份验证。表 20-5 列出了 User.Identity 提供的属性。

表 20-5

属 性	说 明
AuthenticationType	提供当前用户的身份验证类型，值可以是 Basic、NTLM、Forms 和 Passport
IsAuthenticated	返回一个布尔值，指定用户是否已通过身份验证
Name	提供用户的用户名以及用户的域(只有使用 Windows 账户登录时才会有域)

为了演示 User 对象的使用，下面检查用户的登录名。为此，使用如程序清单 20-12 所示的代码。

程序清单 20-12 获取登录用户的用户名

```
string userName;
userName = User.Identity.Name;
```

另一项可以使用 User.Identity 对象完成的任务是通过应用程序的身份验证方法检查用户是否已通过验证，如程序清单 20-13 所示。

程序清单 20-13 检查用户是否已通过身份验证

```
bool authUser;
authUser = User.Identity.IsAuthenticated;
```

这个例子的结果是一个布尔值，表示用户是否已通过身份验证，也可以在 if/else 语句中使用 IsAuthenticated 方法，如程序清单 20-14 所示。

程序清单 20-14 使用 if/else 语句检查身份验证情况

```
if(User.Identity.IsAuthenticated) {
    // Do some actions here for authenticated users
}
else {
    // Do other actions here for unauthenticated users
}
```

还可以使用 User 对象检查用户的身份验证类型，需要借助该对象的 AuthenticationType 属性来完成该操作，如程序清单 20-15 所示。

程序清单 20-15 使用 AuthenticationType 属性

```
string authType;
authType = User.Identity.AuthenticationType;
```

结果可以是 Basic、NTLM 或 Forms。

20.3.2 使用 IsInRole 方法

如果使用的是基于 Windows 的身份验证模式,就可以确保已验证用户在某个 Windows 角色中。例如,如果只为 Computer Management 实用程序的 Subscribers 组中的用户显示某些信息,那么可以使用 User 对象的 IsInRole 方法,如程序清单 20-16 所示。

程序清单 20-16 检查用户是否在某个角色中

```
if (User.IsInRole("Win8Pro-En\\Subscribers")) {  
    // Private information for subscribers  
}  
else {  
    // Public information  
}
```

IsInRole 方法的参数提供了一个字符串值,表示域和组(Windows 角色)。在本例中,指定允许 ReutersServer 域的 Windows 角色 Subscribers 中的任意用户查看信息,而不属于这个角色的用户不能查看这些信息。

另一种方法是指定一些内置组。Windows 包含一系列内置账户,如 Administrator、Guest 和 User。可以通过两种方式来访问这些内置账户,一种方式是直接使用域指定内置账户:

```
User.IsInRole("Win8Pro-En\Administrator")
```

另一种方式是使用 BUILTIN 关键字:

```
User.IsInRole("BUILTIN\Administrator")
```

20.3.3 使用 WindowsIdentity 显示更多信息

前面在处理用户的身份信息时使用了 ASP.NET 中默认的标准 Identity 对象。如果使用基于 Windows 的身份验证模式,那么还可以使用 WindowsIdentity 对象和其他对象。为了访问这些功能丰富的对象,可在应用程序中创建对 System.Security.Principal 对象的引用。

这些对象和前面例子中的 Identity 对象一起使用,可以更容易完成一些任务。例如,如果使用角色, System.Security.Principal 就会提供对 WindowsBuiltInRole 枚举的访问。

程序清单 20-17 是使用 WindowsBuiltInRole 枚举的一个例子。

程序清单 20-17 使用 WindowsBuiltInRole 枚举

```
bool adminUser;  
adminUser = User.IsInRole(WindowsBuiltInRole.Administrator.ToString());
```

没有指定域和角色的字符串值,而是可以使用 WindowsBuiltInRole 枚举来访问应用服务器上的某些角色。在使用这个枚举和其他枚举时, IntelliSense 还会简化我们的选择操作,如图 20-11 所示。

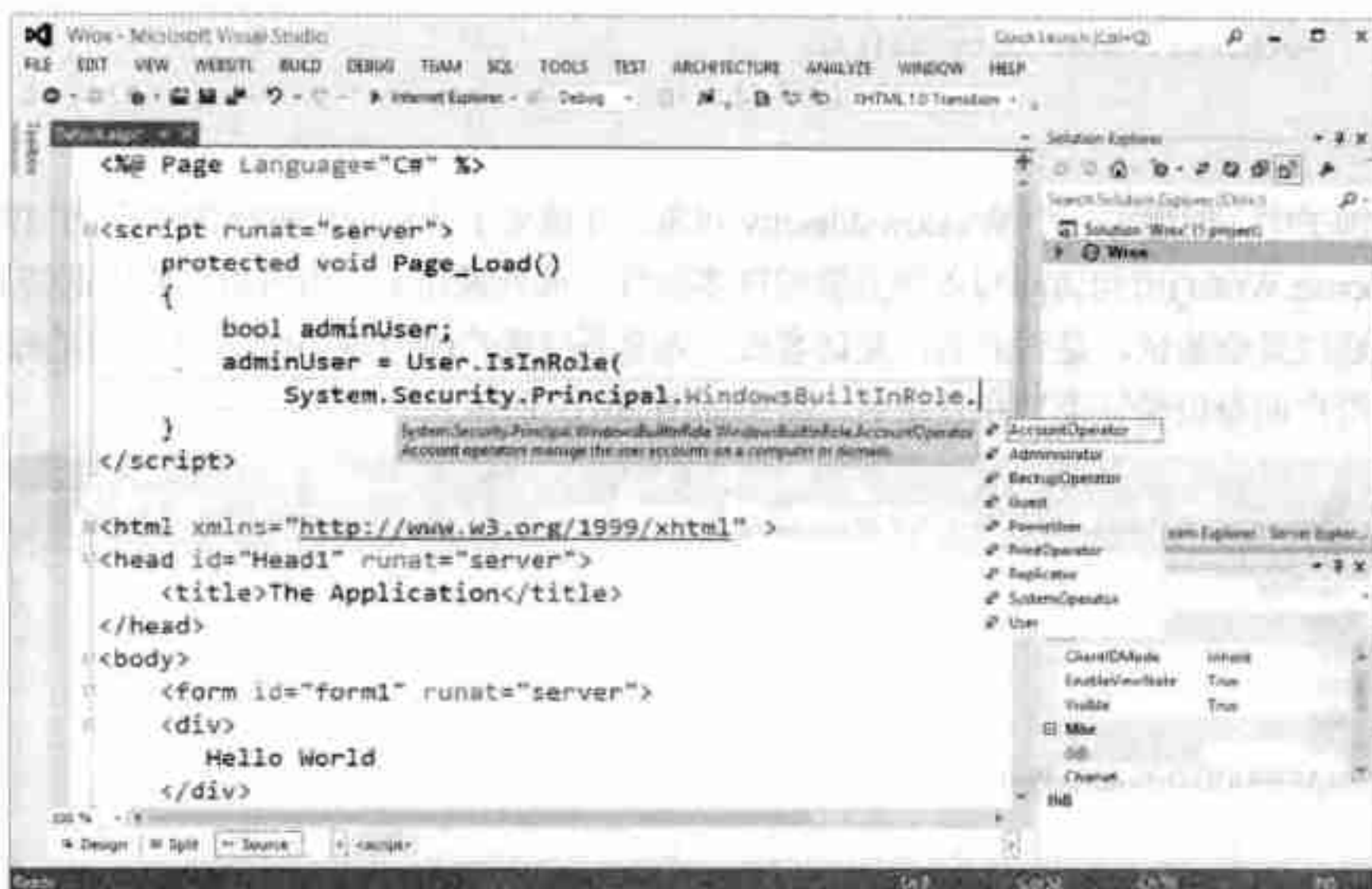


图 20-11

WindowsBuiltInRole 枚举中的角色包括:

- AccountOperator
- Administrator
- BackupOperator
- Guest
- PowerUser
- PrintOperator
- Replicator
- SystemOperator
- User

使用 System.Security.Principal 可以访问 WindowsIdentity 对象, 该对象的功能比默认的 Identity 对象强大得多。程序清单 20-18 列出了可以通过 WindowsIdentity 对象获得的其他信息。

程序清单 20-18 使用 WindowsIdentity 对象

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Security.Principal" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        WindowsIdentity AuthUser = WindowsIdentity.GetCurrent();
        Response.Write(AuthUser.AuthenticationType.ToString() + "<br>" +
            AuthUser.ImpersonationLevel.ToString() + "<br>" +
            AuthUser.IsAnonymous.ToString() + "<br>" +
            AuthUser.IsAuthenticated.ToString() + "<br>" +
            AuthUser.IsGuest.ToString() + "<br>" +
            AuthUser.IsSystem.ToString() + "<br>" +
```

```

        AuthUser.Name.ToString());
    }
</script>

```

在这个例子中，创建了一个 `WindowsIdentity` 对象，并填充了访问应用程序的用户的当前身份。接着使用 `Response.Write()` 语句访问写入浏览器的许多属性。该列表显示了当前用户的凭据信息，例如，用户是否已通过身份验证，是否匿名，是访客账户还是系统账户(例如应用程序池使用的账户)。该列表还给出了用户的身份验证类型和登录名，结果如图 20-12 所示。

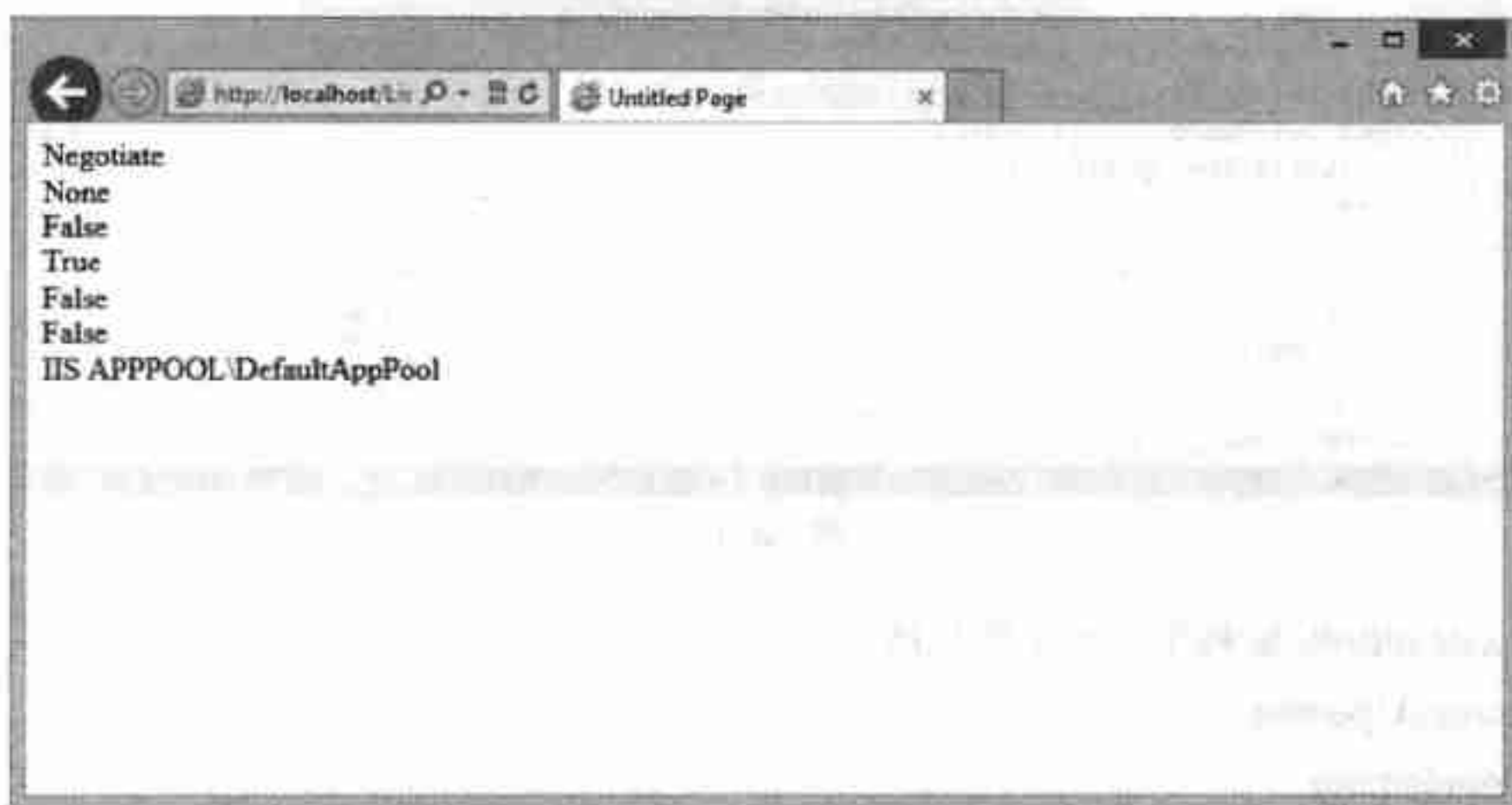


图 20-12

20.4 身份和模拟

默认情况下，ASP.NET 运行在权限有限的账户下。例如，尽管账户可以访问网络，但不能在网络的其他计算机上进行身份验证。在 `machine.config` 文件中进行账户设置：

```

<processModel
  enable="true"
  userName="machine"
  password="AutoGenerate" />

```

这些设置迫使 ASP.NET 运行在系统账户(ASP.NET 或 Network Service)下。这是通过 `userName` 特性指定的，该特性的值是 `machine`。这个特性的值还可以是 `system`。下面是这两个值的详细说明：

- **machine**: 最安全的设置。最好不要修改这个值。这是非常理想的选择，因为它会强制 ASP.NET 账户在权限最少的情况下运行。
- **System**: 强制 ASP.NET 在本地 SYSTEM 账户下运行，它的权限要比访问网络 and 文件大得多。

还可以在 `machine.config` 文件或 `web.config` 文件中使用 `<processModel>` 元素指定账户：

```

<processModel
  enable="true"
  userName="MySpecifiedUser"
  password="MyPassword" />

```

在这个例子中，ASP.NET 在指定的管理员或用户账户下运行，而不是在默认的 ASP.NET 或

Network Service 账户下运行。它继承了这个账户提供的所有权限。应考虑对该文件的这个部分加密, 有关配置文件加密的内容, 详见第 28 章。

使用 web.config 文件中的 <identity> 元素, 无论 ASP.NET 在哪个指定的账户下运行, 都可以改变 ASP.NET 的运行方式。web.config 文件中的 <identity> 元素允许启用模拟功能。模拟功能可以让 ASP.NET 为某个会话使用另一个用户的权限作为进程运行。更具体地说, 就是模拟功能允许 ASP.NET 在向应用程序发出请求的实体账户下运行。要启用模拟功能, 可以使用 <identity> 元素的 impersonate 特性, 如下所示:

```
<configuration>
  <system.web>
    <identity impersonate="true" />
  </system.web>
</configuration>
```

默认情况下, impersonate 特性设置为 false。把这个特性设置为 true, 将确保 ASP.NET 运行在向应用程序发出请求的用户账户下。如果请求者是匿名用户, ASP.NET 就运行在 IUSR_MachineName 账户下。为了演示这一点, 运行程序清单 21-18, 启用模拟功能。此时我们获得的用户名不是 IIS APPPOOL\DefaultAppPool, 而是请求页面的用户的名称 Win8Pro-EN\Christian, 如图 20-13 所示。当然, 这需要用户有访问 ASP.NET 文件的权限。



图 20-13



可能需要使用如下 web.config 设置, 让 Web 服务器接受新的模拟设置:

```
<system.webServer>
  <validation validateIntegratedModeConfiguration="false" />
</system.webServer>
```

还可以使用 web.config 文件中的 <identity> 元素, 使 ASP.NET 在指定的账户下运行:

```
<identity impersonate="true" userName="MySpecifiedUser" password="MyPassword"/>
```

如上所示, 通过 userName 和 password 特性, 可以在指定的账户下运行 ASP.NET 进程。这些值

在 web.config 文件中存储为明文。

查看 web.config 根文件, 可以看到 ASP.NET 在完全可信的级别下运行, 也就是说, 它利用一些高级功能来运行和访问资源。下面是相应设置:

```
<system.web>

  <location allowOverride="true">
    <system.web>
      <securityPolicy>
        <trustLevel name="Full" policyFile="internal"/>
        <trustLevel name="High" policyFile="web_hightrust.config"/>
        <trustLevel name="Medium" policyFile="web_mediumtrust.config"/>
        <trustLevel name="Low" policyFile="web_lowtrust.config"/>
        <trustLevel name="Minimal" policyFile="web_minimaltrust.config"/>
      </securityPolicy>
      <trust level="Full" originUrl="" />
      <fullTrustAssemblies />
      <partialTrustVisibleAssemblies />
    </system.web>
  </location>

</system.web>
```

可以给 ASP.NET 的信任级别指定 5 个设置: Full、High、Medium、Low 和 Minimal。通过 <trust> 元素的 level 特性指定信任级别, 默认设置为 Full。每个信任级别都指向特定的策略配置文件, 在该文件中, 可以找到信任级别设置。Full 设置不包含策略文件, 因为它跳过了所有的代码访问安全检查。

20.5 通过 IIS 进行保护

ASP.NET 和 IIS 一起工作。我们不仅可以在 ASP.NET 中直接应用安全性设置(通过代码或配置文件), 还可以在 IIS 中应用其他安全措施。IIS 允许使用用户和组(详见本章前面的内容)、受限的 IP 地址、文件扩展名等来应用访问方法。通过 IIS 设置的安全性需要一整章的篇幅来讨论, 这里只探讨核心主题。

20.5.1 使用文件扩展名

在 ASP.NET 中可以使用许多类型的文件。使用其扩展名定义这些文件。例如, .aspx 是一般的 ASP.NET 页面, .asmx 是 ASP.NET Web 服务文件的扩展名。这些文件都由 IIS 映射给 ASP.NET DLL, 即 aspnet_isapi.dll。

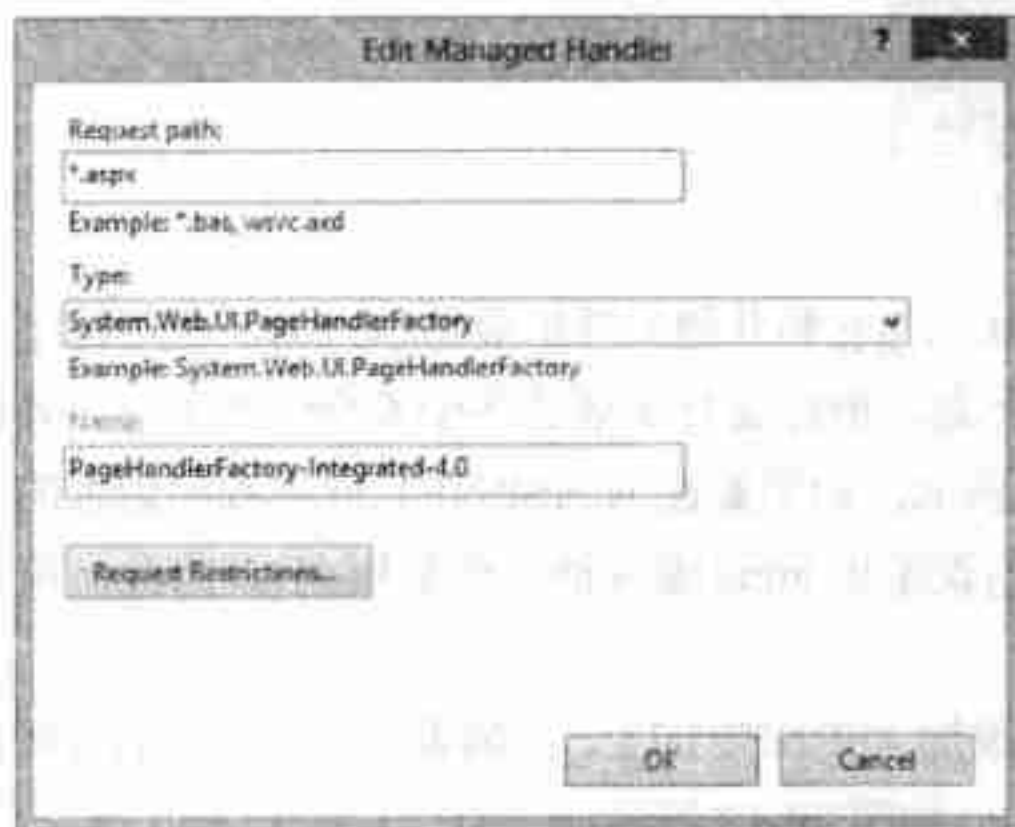
如果使用 Windows 7 或更高版本, 就可以使用 IIS Manager 把文件扩展名映射到处理程序。在这个工具中, 选择 IIS 部分中的 Handler Mappings, 就会打开已提供映射的列表, 如图 20-14 所示。

突出显示第一个 *.aspx 选项, 单击 Edit 按钮, 这个扩展名就映射为 System.Web.UI.PageHandlerFactory 处理程序, 如图 20-15 所示。

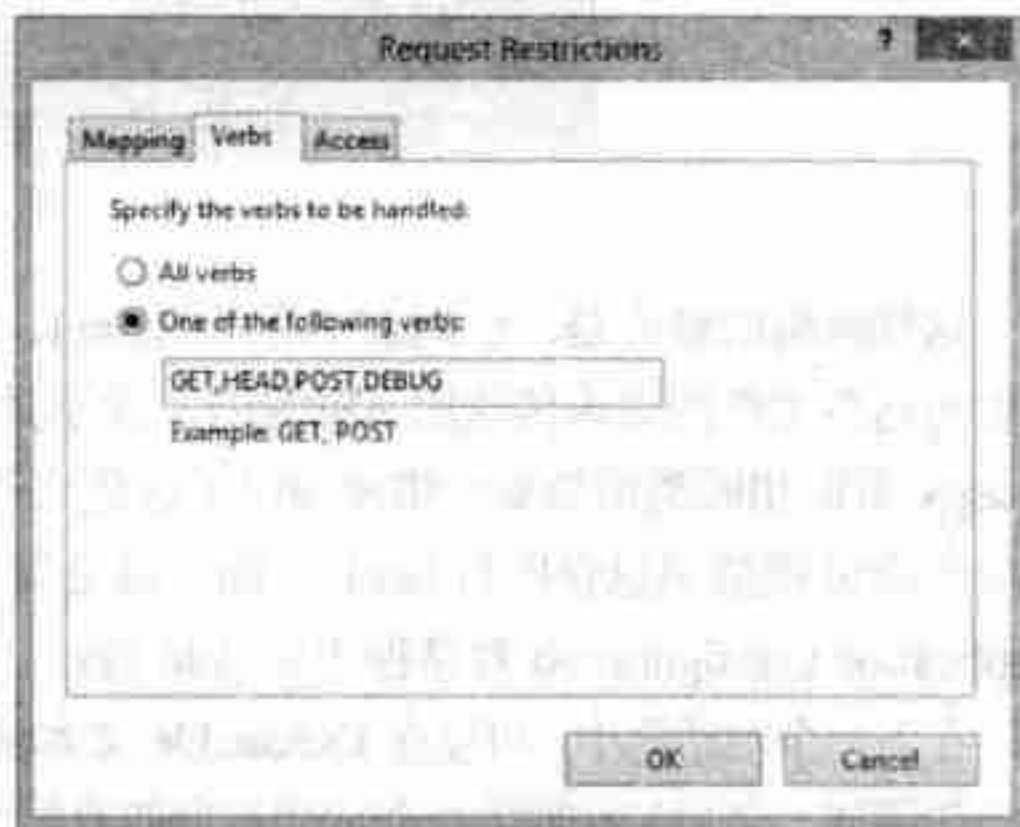
单击 Request Restrictions 按钮, 会打开一个对话框, 该对话框允许选择可以使用的谓词, 如图 20-16 所示。



图 20-14



Year	Percentage (%)
1980	11.5
1985	12.5
1990	13.5
1995	14.5
2000	15.5
2005	16.5
2010	17.5
2015	18.5
2020	19.5



20-16

为了获得与 IIS 6.0 类似的结果,可在 IIS 中打开 Web 应用程序的 Properties 对话框,或者打开 Default Web Site Properties 对话框。对于特定的 Web 应用程序,必须在 Directory 选项卡中进行操作;但如果使用 Default Web Site Properties 对话框,就可以使用 Home Directory 选项卡。在这些选项卡中,单击 Application Settings 框中的 Configuration 按钮。Application Configuration 对话框包含 Mapping 选项卡,在其中可以配置映射。突出显示映射列表中的.aspx 选项,单击 Edit 按钮,图 20-17 显示了结果。

在 Executable 文本框中, 所有的.aspx 页面都映射到 ASP.NET 中的 aspnet_isapi.dll, 还可以指定允许应用程序发出的请求类型。可以允许使用所有的谓词(如 GET 或 POST), 也可以指定用于访问应用程序的谓词。



图 20-17

这些映射的要点是，在列表中看不到.html、.htm、.jpg 和其他文件扩展名(如.txt)。应用程序不会把对这些文件的请求传送给 ASP.NET。这不是大问题，但在运行本章的各个安全性例子时，要把与.aspx 页面相同类型的安全措施应用于这些文件。例如，如果要在为 ASP.NET 应用程序请求的表单身份验证模型中包含所有.html 页面，就必须给列表添加.html(或其他文件扩展名)。为此，单击 Application Configuration 对话框中的 Add 按钮。

在下一个对话框中，可以在 Executable 文本框中添加 ASP.NET DLL，在列表中添加相应的文件扩展名和谓词，之后把映射添加到应用程序的映射表中，如图 20-18 所示。



图 20-18

在处理站点的安全性时，必须记住所有没有包含到默认映射列表中的文件，并把需要的文件添加到同一安全结构中。

20.5.2 使用 IIS 7.x/8 Manager

在 IIS 7、7.5 和 8 中对站点进行相同修改的工具是 Internet Information Services (IIS) Manager，如图 20-19 所示。

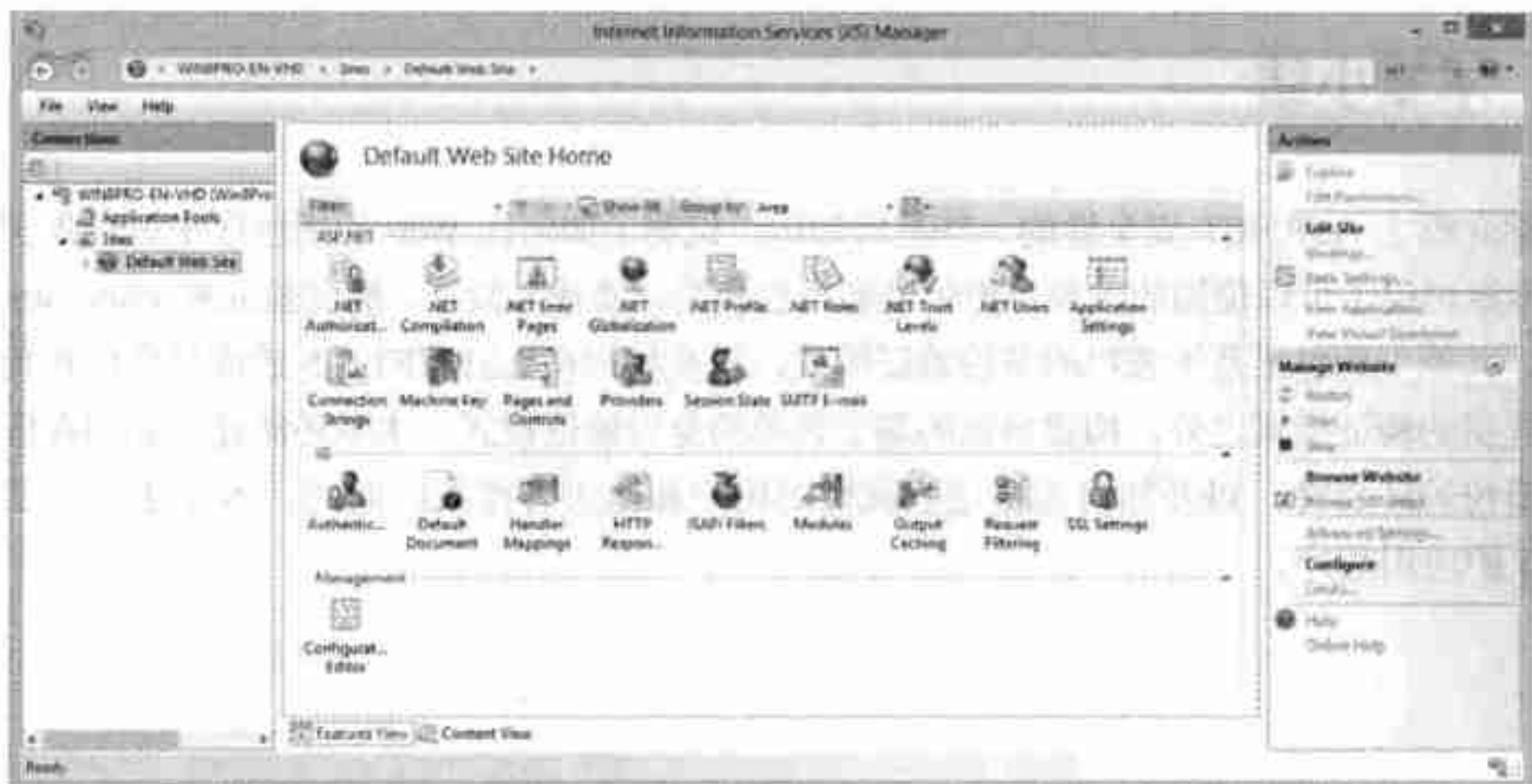


图 20-19

通过这个对话框进行修改后，就可以选择 Actions 窗格中的 Apply Changes 链接，它会说明所做的修改是否已保存。如果已保存，所做的修改就会应用于站点的 web.config 文件。

20.5.3 使用 ASP.NET MMC 管理单元

在 IIS 7 以前的版本中，还没有 IIS Manager，但 ASP.NET MMC 控制台允许使用易用的 GUI 来编辑 web.config 文件和 machine.config 文件，而不是对这些文件的文本进行必要的修改。也可以使用这个对话框修改本书介绍的大多数项。Web 应用程序在 IIS 中运行时，就可以在 ASP.NET 选项卡中使用这个插件，如图 20-20 所示。直接在该对话框中修改时，还要对配置文件进行硬编码式的修改。

单击 ASP.NET 选项卡中的 Edit Configuration 按钮，打开 ASP.NET Configuration Settings 对话框。在该对话框中可以修改表单身份验证模式在 GUI 中的工作方式，而无须直接进入应用程序的 web.config 文件。图 20-21 是在 GUI 中处理表单身份验证模式的例子。



图 20-20

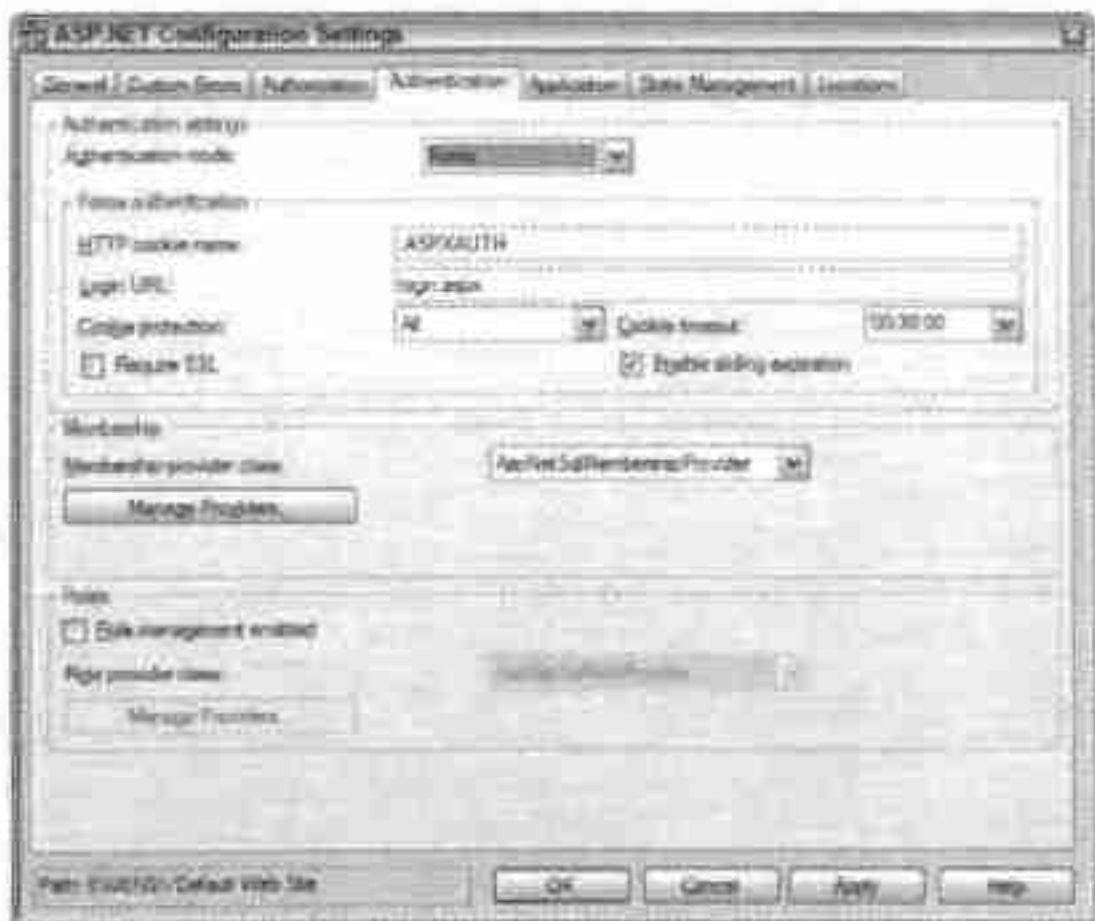


图 20-21

20.6 本章小结

本章介绍了 ASP.NET 安全性的一些基础知识，说明了如何在 Web 应用程序中进行身份验证和授权。本章讨论了可以使用的各种身份验证和授权模型，如基本验证、摘要验证和 Windows 集成身份验证。其他主题包括基于表单的身份验证模式，以及如何在 ASP.NET 4.5 的成员资格和角色管理功能所提供的验证模式之外，构建自己的基于表单的身份验证模式。本章还阐述了如何在应用程序中使用身份验证属性，以及如何根据这些属性对用户和组进行授权。同时，本章还介绍了如何通过 IIS 保护应用程序。

第Ⅵ部分

应用程序状态

- 第 21 章 状态管理
- 第 22 章 高速缓存

第21章

状态管理

本章要点

- 使用 Session 对象
- 用于控制状态的其他选项

HTTP 是无状态协议，不能保存信息。但许多当前的 Web 应用程序必须维护状态，它们需要保存登录的用户、购物车的内容等。

在 Web 流行起来之前，使用标准的客户端/服务器体系结构意味着使用胖客户端和胖服务器。也许桌面应用程序可以与数据库通信，而状态存储在客户端或服务器端的数据库中。一般可以使用客户端上的少量内存和硬盘来管理状态。但是，传统客户端/服务器设计的最重要的方面是，客户端总是与服务器连接起来。人们很容易忘记这一点，但 HTTP 是无状态协议。在大多数情况下，每次给远程服务器发出调用时，都要建立连接，然后删除。HTTP 1.1 的实时技术在 TCP 级别提供了优化。即使有了这些优化，服务器也仍然无法确定后续的连接是否来自同一个客户端。

ASP.NET 的会话管理支持提供了一个简单的 API，在会话期间存储数据；所有的工作都是由该架构完成的，我们不需要担心。本章介绍几个状态管理技术，以便读者选择最适合自己的技术。

21.1 如何选择会话状态

假定有无状态的协议(如 HTTP)，而在服务器端是 ASP.NET，那么应该如何管理 Web 上的状态？图 21-1 列出了管理状态的主要方式。问题很大，但解决方案的范围更大。本章假定不使用 Java applet 或 ActiveX 控件管理状态，因为这些技术不适用于所有的系统。

关于状态管理，要记住一点：状态管理并没有什么正确答案。一些答案肯定比另一些答案好，但管理状态有非常多的方式。回想自己的上一个项目，我们花费多少天才决定在何处管理状态？其难点是真正理解每种方法的优缺点。

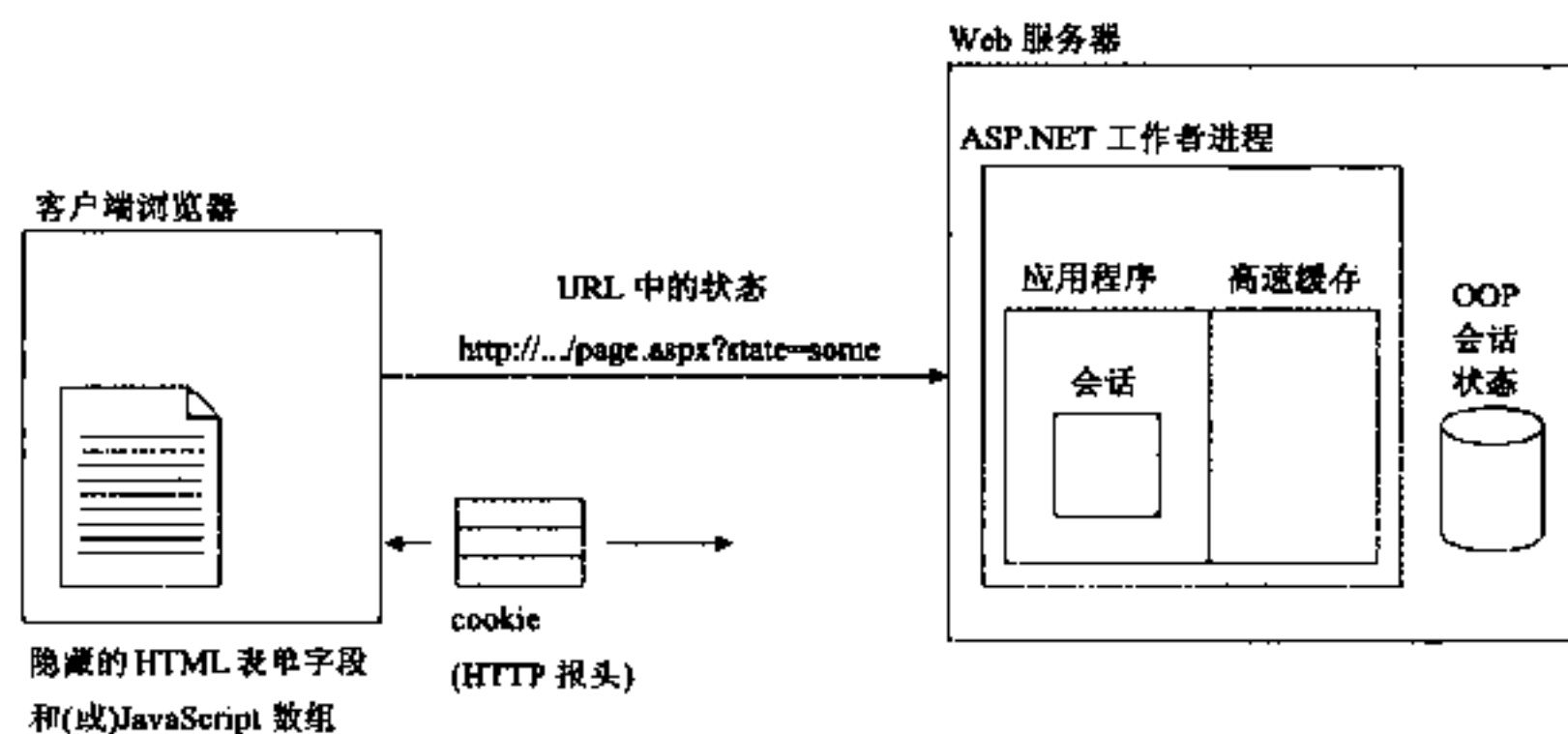


图 21-1

要对一种方法作出有意义的决定，应理解请求的生存期和在该过程的每一阶段如何进行状态管理：

(1) Web 浏览器对服务器上的页面 `http://myserver/myapp/mypage.aspx` 发出 HTTP GET 请求。这个客户端 Web 浏览器以前从来没有访问过该站点。

(2) IIS 和 ASP.NET 应用程序通过返回 `mypage.aspx` 显示的 HTML 作为响应。另外，`mypage.aspx` 还返回带唯一 ID 的 cookie，以此来跟踪这个 Web 浏览器。cookie 实际上是一个稍微抽象的概念。通过把 Set-Cookie HTTP 报头返回给客户端来设置 cookie。然后客户端会在 HTTP 报头中为每个后续的 HTTP 调用返回 cookie 的值。在这个例子中，状态实际上是客户端和服务端之间达成的协议，允许在响应中为每个请求来回传送 cookie。

(3) 返回的 HTML 可以包含隐藏的文本框，如 `<input type="hidden" value="somestate"/>`。这些文本框类似于 cookie，因为如果提交这个页面上的表单，就把它们传送回服务器。cookie 在每个域中设置，而隐藏的表单字段在每个页面上设置。

(4) 在下一个请求中，以前设置的 cookie 会返回给服务器。如果这个请求是把表单提交为 HTTP POST，就返回表单中的所有字段(包括隐藏的字段和未隐藏的字段)。

(5) 以前设置为 cookie 的唯一标识符现在可以在任意类型的服务器端状态机制中用作键。状态可以是内存中的散列表，也可以是 SQL 数据库。

注意客户端和服务端之间用于来回传送信息的协议。该信息可以放在 URL、HTTP 报头中，甚至是在提交的表单中作为输入字段。

在服务器端，可以使用几个选项。应根据可用的内存大小、要存储的数据量和需要访问数据的频率和速度，从这些选项中选择。

表 21-1 和表 21-2 描述了服务器端和客户端的每个选项，并分别列出了它们的优缺点。

表 21-1

服务器端选项	优点	缺点
Application State	很快，在所有的用户间共享	在多个服务器配置中，状态要在每台服务器上存储一次

(续表)

服务器端选项	优点	缺点
Cache Object(在应用程序的范围内)	类似于 Application State 选项, 但通过依赖性(详见第 22 章关于高速缓存的讨论)包含过期设置	在多个服务器配置中, 状态要在每台服务器上存储一次
Session State	3 个选择: 在进程中、在进程外、由 DB 支持。可以配置为无 cookie	可能会被滥用。在对象退出进程时要进行序列化。在进程中需要 Web 服务器。无 cookie 配置更容易受到攻击
Database	状态可以由 Web Farm 中的任意服务器访问	在对象退出进程时要进行序列化和永久保存

在客户端, 每个可用的选项都需要一定的带宽。每个选项都要在客户端和服务器之间来回传送数据。所存储的每个数据字节都要传送两次: 一次是传送给服务器, 另一次是传送回来。

表 21-2

客户端选项	优点	缺点
Cookie	简单	可能会被浏览器拒绝。不适合大量数据的情况, 也不适合敏感数据。要占据每个 HTTP 请求和响应的空间
Hidden Field	对于页面范围内的数据来说非常简单	不适合大量数据的情况, 也不适合敏感数据
ViewState	对于页面范围内的数据来说非常简单	把序列化的对象编码为二进制 Base64 编码数据, 会增加大约 30% 的系统开销。序列化的系统开销较小。该选项不太好用, 尤其是与 DataGrid 一起使用时
ControlState	对于页面范围内用于控件的数据来说非常简单	与 ViewState 类似, 但用于需要 ViewState 的控件(即使开发人员关闭了 ViewState)
QueryString (URL)	非常简单, 如果能让终端用户直接修改 URL, 该选项通常很方便	相对复杂。不能存储大量信息。不适合敏感数据。容易被终端用户修改
HTML5 Web Storage	存储名/值对的简单 API	数据从来不自动发送给服务器, 所以主要用于客户端逻辑

表 21-1 和表 21-2 提供了一些服务器端和客户端选项。ASP.NET 4.x 中对高速缓存的改进详见第 22 章。

21.2 理解 ASP.NET 中的 Session 对象

在传统的 ASP 中, Session 对象保存在 IIS 进程中。用户接收到带唯一键(形式是 GUID)的 cookie。Session 键是存储对象引用的字典中的索引。

在 ASP.NET 的所有版本中, Session 对象都提供了一个进程中(in-process)选项, 还包含了一个进

程外(out-of-process)选项和一个由数据库支持(database-backed)的选项。另外,开发人员还可以使用无 cookie 的会话状态,在该状态下,Session 键显示在 URL 中,而不是作为 cookie 发送。

21.2.1 会话和事件模型

HttpApplication 对象在 HTTP 协议请求的生存期内会引发一系列事件,本节提及所有的事件,但只详细探讨与会话相关的事件:

- BeginRequest
- AuthenticateRequest
- AuthorizeRequest
- ResolveRequestCache
- AcquireRequestState(这个事件表示要获得与这个 HTTP 请求相关的所有会话状态)



在 AcquireRequestState 事件触发后,开发人员就可以获得会话状态。会话状态键对每个用户来说都是唯一的,可以从 cookie 或 URL 中获得。

- PreRequestHandlerExecute
- PostRequestHandlerExecute
- ReleaseRequestState(表示会话状态应存储起来,在此刻使用 web.config 文件中配置的会话状态模块存储会话状态)
- UpdateRequestCache
- EndRequest

在执行应用程序代码时,使用 cookie 中当前的 Session 键填充 Session 对象,或者使用 URL 中的 Session 键填充 Session 对象。如果要在会话开始时进行一些处理,而不是在 AcquireRequestState 事件中处理,就可以为会话状态 HttpModule 的 Start 事件定义事件处理程序:

```
void Session_OnStart() {
    'this fires after session state has been acquired by the SessionStateModule.
}
```



Session 对象包含 Start 和 End 事件,它们都可以根据需求来关联事件处理程序。但是,Session_OnEnd 事件仅能在进程中的 Session State 模式下使用。如果使用进程外的 State Server 模式或 SQL Server 模式,就不会触发这个事件。该事件的处理程序不会监听到 Session 对象的结束。

可以在 Page 对象的子类的任何事件中使用 HttpSessionState 对象。在 ASP.NET 中创建的页面派生自 System.Web.UI.Page,因此可以把会话状态作为集合来访问,因为 System.Web.SessionState.HttpSessionState 实现了 ICollection。

Page 类的公有属性 Session 会从当前的 HttpContext 中自动提取 Session。看起来好像 Session 对象在页面中,但实际上位于 HttpContext 中,页面的公有属性 Session 会提取对会话状态的引用。这

不仅方便了传统的 ASP 程序员，还减少了输入量。

Session 对象可以通过如下方式在页面中引用：

```
Session["SomeSessionState"] = "Here is some data";
```

或者：

```
HttpContext.Current.Session["SomeSessionState"] = "Here is some data";
```

Session 对象实际上位于当前的 HttpContext 中，这一点很重要。这就允许在上下文中访问 Session 对象，而不是在页面中(例如在 HttpHandler 中)访问。

21.2.2 配置会话状态的管理

页面中的所有代码都使用类似字典样式的语法来引用 Session 对象，但 HttpSessionState 对象使用提供程序的模式来选择会话状态的存储方式。通过修改 web.config 文件中的 sessionState 元素，可以在包含的提供程序中进行选择。ASP.NET 包含如下 3 个存储提供程序：

- **进程中的会话状态存储：**在 ASP.NET 内存的高速缓存中存储会话。
- **进程外的会话状态存储：**在 ASP.NET State Server 服务 aspnet_state.exe 中存储会话。
- **Sql 会话状态存储：**在 SQL Server 数据库中存储会话，使用 aspnet_regsql.exe 配置会话。

web.config 文件的 sessionState 元素的格式如下所示：

```
<configuration>
  <system.web>
    <sessionState mode="Off|InProc|StateServer|SQLServer|Custom" ../>
  </system.web>
```

在新 Web 站点的 web.config 文件中设置 sessionState 元素的 mode="InProc" 特性，开始配置会话状态。这是会话状态在 ASP.NET 中最常用的配置，也是最快速的配置。

21.2.3 进程中的会话状态

将配置设置为 InProc 时，就会使用实现了 ICollection 的 ISessionStateItemCollection，将会话数据存储在 HttpRuntime 的内部高速缓存中。会话状态键是 120 位的字符串值，是对象引用的全局字典中的索引。会话状态在进程中时，对象就存储为活动的引用。这是一种非常快速的机制，因为没有进行序列化，对象也没有退出进程。显然，如果对象存储到 In-Process Session 对象中，它们就不会被垃圾收集，因为其引用仍然存在。

另外，因为对象存储在内存中，所以在会话超时前它们会耗尽内存。如果有用户访问站点，单击了一个页面，就可能要在正在进行的会话中存储 40MB 大小的 XmlDocument 对象。如果该用户一直没有返回，这块内存就要被占用 20 分钟左右(这是个可配置的值)，直到会话结束为止。

1. InProc 陷阱

尽管 InProc 会话模型是最快速、最常用的默认模型，但却存在很大的局限。如果再次使用工作者进程或应用程序域，那么所有的会话状态数据都会丢失。另外，ASP.NET 应用程序会由于许多原因而重新启动，例如：

- 修改了 web.config 或 Global.asax 文件，或者改变了文件的修改日期。

- 修改了\bin 或\App_Code 目录中的文件。
- 在 web.config 或 machine.config 文件中设置了 processModel 元素, 指明应用程序应何时重新启动。引起重新启动的条件有内存限制或请求队列的限制。
- 防病毒软件修改了上述文件之一。这在只负责保护文件的防病毒软件中是很常见的情况。

这也就是说, 进程中的会话状态非常适合只需要一台 Web 服务器的小型应用程序, 也适合 IP 负载均衡机制把每个用户返回到最初创建会话的服务器。

如果用户已经有了会话键, 但把它返回到不是创建该会话的计算机, 此时, 目标计算机不知道会话键, 因此创建新的会话。这个新会话的基础是用户提供的会话 ID。会话键可能是相同的, 但目标计算机以前不知道这个会话, 所以没有把数据关联到这个会话键。这个新会话是空的, 可能会得到预料不到的结果。但是, 如果在 web.config 文件中把 regenerateExpiredSessionId 特性设置为 True, 就会创建新的会话 ID 并赋予该用户。

2. Web 园

Web 园(Web gardening)是用于多处理器系统的技术, 在这种系统中, 会启动 ASP.NET 工作者进程的多个实例, 并指定处理器的亲和级别。在有 4 个 CPU 的大型 Web 服务器上, 可以在 1~4 个工作者进程中存储 ASP.NET。处理器的亲和级别表示 ASP.NET 工作者进程与特定 CPU 的紧密关系, 它会“钉”在该 CPU 上。这种技术通常只能在非常大的 Web farm 中使用。

注意, 进程中的会话状态只是进程中的状态。即使 Web 应用程序只包含一台 Web 服务器, 并且所有的 IP 通信都路由到这台服务器, 也不能保证在同一处理器上处理后续每个请求。Web 园必须遵循 Web farm 遵循的许多规则。



如果在多处理器系统中使用 Web 园, 就不能使用进程中的会话状态, 否则就会丢失会话。进程中的会话状态只适合应用程序与应用程序域的比例为 1:1 的情况。

3. 在 Session 对象中存储数据

在下面的简单例子中, Button_Click 事件把文本框的内容添加到带指定键的 Session 对象中。然后用户单击进入该应用程序的另一个页面, 提取 Session 对象中的数据, 并显示在浏览器中。

注意<asp:HyperLink>控件的使用。当然, 这个标记可以硬编码为 HTML, 但这个细小的区别以后会很有用。另外, URL 是相对于这个站点的, 而不是绝对的。查看它有助于理解本章后面的内容。

程序清单 21-1 演示了 Session 对象的使用。该对象与其他 IDictionary 集合一样, 可以存储与任何对象相关的 String 类型的键。

程序清单 21-1 设置会话状态中的值

```
<%@ Page Language="C#" %>

<!DOCTYPE html>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
```

```

    {
        Session["mykey"] = TextBox1.Text;
    }
</script>

<html>
<head id="Head1" runat="server">
    <title>Session State</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
            <asp:Button ID="Button1" runat="server" Text="Store in Session"
                OnClick="Button1_Click" />
            <br />
            <asp:HyperLink ID="HyperLink1" runat="server"
                NavigateUrl="Listing 21-02.aspx">Next Page</asp:HyperLink>
        </div>
    </form>
</body>
</html>

```

程序清单 21-1 中的页面在浏览器中显示,如图 21-2 所示。将 Session 对象作为由字符串键索引的字典进行访问。也可以使用 Add、Count 等方法,但上述语法是最常用的。

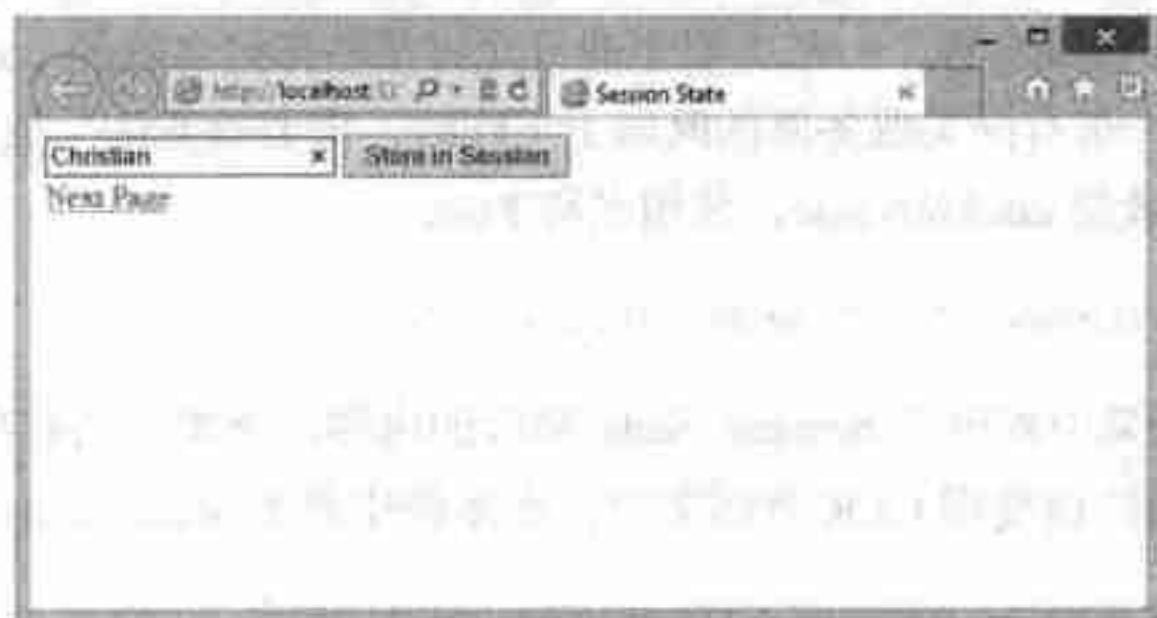


图 21-2

如果激活页面跟踪功能,就可以为开发人员显示页面和 Session 对象的更多信息。为此,在应用程序的 web.config 文件中,给<system.web>元素添加<trace>元素,如下所示:

```
<trace enabled="true" pageOutput="true" />
```

现在激活了跟踪功能,跟踪的结果被直接发送到页面上。有关跟踪和调试的更多信息,请参阅第 29 章。现在只进行这个修改,并刷新浏览器。

在图 21-3 中,显示了激活跟踪功能时返回的跟踪信息的开头和大致中间的部分。会话状态非常符合 ASP.NET 的构造。在跟踪信息的 Request Details 部分可以看到,HTTP POST 的结果不仅包含这个页面,还包含会话 ID——提升到了最高一级的状态。但是,ASP.NET 会话 ID 在默认情况下是 cookie,如图 21-3 底部的 Cookies 集合所示。

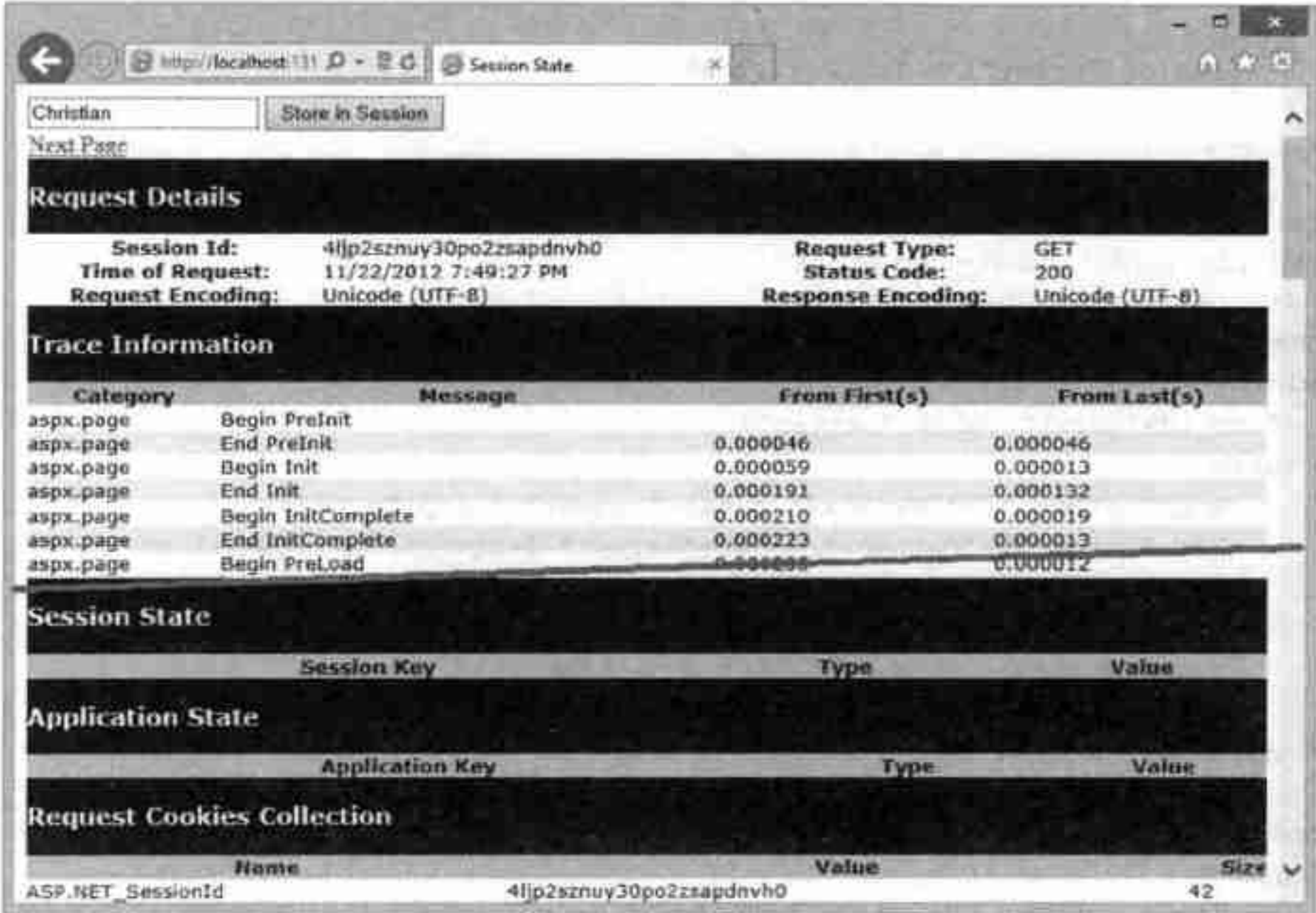



图 21-3

该 cookie 的默认名称是 ASPNET_SessionId，但可以通过 web.config 文件中<sessionState>元素的 cookieName 特性来配置该名称。一些大型企业只允许某些指定名称的 cookie 通过它们的代理，因此在外联网上工作或在带有网关服务器的网络上工作时，需要改变这个值，但这是非常少见的情况。在下面的例子中，改变 cookieName，使用名称 Foo：

```
<sessionState cookieName="Foo" mode="InProc" />
```

图 21-3 中的跟踪结果中列出了 Session State 集合的内容，该集合当前存储了名称 mykey 和值 Hanselman。另外，存储的值使用 CLR 数据类型，在本例中是 System.String。



跟踪结果的 Value 列来自对被包含对象的 ToString 方法的调用。如果在会话中存储自己的对象，就可以重写 ToString 方法，给对象提供更友好的文本表示，使跟踪结果更为有用。

现在添加下一个页面，它从会话中提取这个值。创建一个新的 ASP.NET 页面，添加一个标签，再添加 Page_Load 事件处理程序，如程序清单 21-2 所示。

程序清单 21-2 从会话中提取值

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        string myValue = (string)Session["mykey"];
    }
</script>
```



```

        Label1.Text = myValue;
    }
</script>

<html>
<head id="Head1" runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="Label1" runat="server" />
        </div>
    </form>
</body>
</html>

```

因为会话包含对象引用，所以得到的对象通过 C# 中的强制类型转换机制转换为字符串。

4. 使会话透明

要在 Session 对象中存储和提取特定数据类型的值，通常需要强制转换数据，因为任何内容都会存储为对象。把字符串键用作索引，就可以补偿页面和 Session 对象之间相对脆弱的联系。可以创建专门用于应用程序的会话辅助程序来隐藏这些细节，也可以给基类 Page 添加属性，以更友好的方式把这些对象显示到页面上。因为一般的 Session 对象都可用作 System.Web.UI.Page 的属性，所以可添加派生于 Page 的新类，它有新的属性 MyKey。

首先右击项目，从上下文菜单中选择 Add New Item 命令，创建一个新类，将其命名为 SmartSessionPage，然后单击 OK 按钮。IDE 会告诉我们，它把这个新类放在 App_Code 文件夹中，使其可用于整个应用程序，单击 Yes 按钮。

新的基本页面非常简单，它继承了 System.Web.UI.Page 的所有功能，还包含一个新属性，如程序清单 21-3 所示(代码文件 SmartSessionPage.cs)。

程序清单 21-3 进一步支持会话的基本页面

```

using System;
using System.Web;

public class SmartSessionPage : System.Web.UI.Page
{
    private const string MYKEY = "mykey";
    public string MyKey
    {
        get
        {
            return (string)Session[MYKEY];
        }
        set
        {
            Session[MYKEY] = value;
        }
    }
}

```

```

    }
}

```

现在返回到程序清单 21-1 中的代码，使页面派生于这个新的基类。为此，在隐藏代码文件中修改基类，使其派生于 `SmartSessionPage` 类。程序清单 21-4 说明了隐藏代码文件中的类派生于 `SmartSessionPage` 类，`SmartSessionPage` 类又派生自 `System.Web.UI.Page`。程序清单 21-4 中突出显示了对程序清单 21-1 所做的修改。

程序清单 21-4 派生于新的基本页面

```

<%@ Page Language="C#" Inherits="SmartSessionPage" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        //Session["mykey"] = TextBox1.Text;
        MyKey = TextBox1.Text;
    }
</script>

```

在这段代码中，改变了对 `Session` 对象的访问，使其使用新的公有属性。在修改完程序清单 21-3 后，派生的所有页面都有公有属性 `MyKey`。可以在不进行任何类型转换或与会话键索引相关的情况下使用这个属性。其他属性的添加与会话中的其他对象一样。



在程序清单 21-3 中，私有字符串值的名称与 VB 中的公有属性相冲突，因为它们仅在大小写上有区别。在 C# 中，私有变量 `MYKEY` 与公有属性 `MyKey` 都是可以接受的。在创建用于多种语言的 API 时要注意这个问题。应考虑到 CLS 兼容性。

5. 优化会话性能的高级技术

在默认情况下，所有的页面都可以对会话进行写入访问。因为可以在浏览器客户端上同时请求多个页面(使用框架、同一台计算机上的多个浏览器窗口等)，所以在请求页面的过程中，该页面会在会话上打开读取器/写入器锁定。如果页面在一个会话上有写入锁定，该会话上请求的其他页面就必须等待第一个请求结束。也就是说，只为该会话 ID 锁定会话。这些锁定不影响有其他会话的用户。

为了使使用会话的页面有最佳性能，ASP.NET 允许通过 `@Page` 特性 `EnableSessionState` 明确声明页面需要什么 `Session` 对象。该特性的可选值有 `True`、`False` 和 `ReadOnly`：

- `EnableSessionState="True"`：页面需要对会话的读写访问。有这个会话 ID 的会话在每个请求过程中都会被锁定。
- `EnableSessionState="False"`：页面不需要访问会话。如果代码使用 `Session` 对象，就抛出 `HttpException` 异常，停止页面的执行。
- `EnableSessionState="ReadOnly"`：页面需要对会话的只读访问。在会话上给每个请求加读取器锁定，但可以同时读取其他页面。锁定请求的顺序是很重要的。只要请求写入器锁定，即使在线程获得访问权限之前请求了该锁定，所有后续的读取器锁定请求也会被禁止，无

论当前是否设置这些读取器锁定。ASP.NET 显然可以处理多个请求，但一次只有一个请求能获得会话的写入访问。把会话状态设置为 `ReadOnly`，多个请求就可以同时访问会话。

修改 `@Page` 指令，以反映每个页面的实际需求，从而可以在站点加载时影响性能。给页面添加 `EnableSessionState` 特性，如下所示：

```
<%@ Page Language="C#" EnableSessionState="ReadOnly" %>
```

ASP.NET 使用 `System.Web.SessionState` 名称空间中的标记接口跟踪每个页面的需求。在生成 `Default.aspx` 的部分类时，实现了 `IRequiresSessionState` 接口，而 `Retrieve.aspx` 实现了 `IReadOnlySessionState` 接口。所有的 `HttpRequest` 都由实现了 `IHttpHandler` 的对象处理，而页面由 `PageHandlerFactory` 处理。第 30 章将详细讨论 `HttpHandler`。 `SessionStateModule` 在内部执行如下所示的伪代码：

```
If TypeOf HttpContext.Current.Handler Is IReadOnlySessionState Then
    Return SessionStateStore.GetItem(itemKey)
ElseIf TypeOf HttpContext.Current.Handler Is IRequiresSessionState
    Return SessionStateStore.GetItemExclusive(itemKey)
End If
```

程序员应知道编译时页面的内部情况，ASP.NET 不会在运行期间指出这些情况。在页面中包含 `EnableSessionState` 特性，就可以使 ASP.NET 更高效地执行。注意，ASP.NET 总是作出最保守的决定，除非给它提供更多的信息。



如果想编写不需要会话的页面，可以设置 `EnableSessionState="False"`。这会使 ASP.NET 在需要会话的页面之前安排该页面，提高应用程序的整体可伸缩性。另外，如果应用程序不使用会话，那么在 `web.config` 文件中设置 `Mode="Off"` 就可以减少整个应用程序的系统开销。

21.2.4 进程外的会话状态

进程外的会话状态保存在 `aspnet_state.exe` 进程中，该进程作为 Windows 服务运行。使用 `Services MMC` 管理单元或在命令行上运行如下 `net` 命令，就可以启动 ASP.NET 状态服务：

```
net start aspnet_state
```

在默认情况下，状态服务监听 TCP 端口 42424，但可以在服务的注册键上修改这个端口，如下所示(状态服务在默认情况下是不启动的)：

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\aspnet_state\Parameters\Port
```

把 `web.config` 文件的设置从 `InProc` 改为 `StateServer`。另外，必须在 `stateConnectionString` 特性中包含运行会话状态服务的 IP 地址和端口。在 `Web farm`(多台 Web 服务器的组)中，可以在任意服务器或完全独立的计算机上运行状态服务。在这个例子中，状态服务器运行在本地计算机上，因此 IP 地址是 127.0.0.1。如果在另一台计算机上运行状态服务器，应打开相应的端口，这里是 TCP 端口 42424。


```
<configuration>
  <system.web>
    <sessionState mode="StateServer"
      stateConnectionString="tcpip=127.0.0.1:42424"/>
  </system.web>
</configuration>
```

使用的状态服务总是 ASP.NET 中最近安装的服务。也就是说，如果在一台计算机上运行 ASP.NET 2.0/3.5/4/4.5 和 1.1，那么存储在 Session 对象中的所有 ASP.NET 版本的所有状态都放在 ASP.NET 状态服务的单个实例中，用于处理会话的服务属于 ASP.NET 的最新版本。

应用程序的代码运行在 ASP.NET 工作者进程中(aspnet_wp.exe 或 w3wp.exe)，而状态服务运行在 aspnet_state.exe 进程中，因此存储在会话中的对象不能存储为引用，而必须通过二进制序列化退出工作者进程。



对于世界一流的、可用性非常高的、可伸缩的 Web 站点来说，应考虑使用非 InProc 的会话模型。即使可以通过负载平衡机制确保把会话保存起来，但是仍然要面对应用程序的再次利用问题。进程外状态服务的数据在再次利用应用程序时会保存起来，但在计算机重新启动时会丢失。如果状态存储在完全不同的计算机上，那么当 Web 服务器再次使用和重新启动时，状态仍然存在。

只有使用[Serializable]特性标记的类才能序列化。在 Session 对象中，可以把[Serializable]特性看作许可证，准许类的实例退出工作者进程。如果以前使用 InProc 会话，这就很重要，因为它们没有这个要求，所以如果改变会话状态模式，就可能要修改代码。

更新 App_Code 目录，使其包含新类 Person，如程序清单 21-5 所示。确保使用[Serializable]特性标记它，否则就会出现如图 21-4 所示的错误。

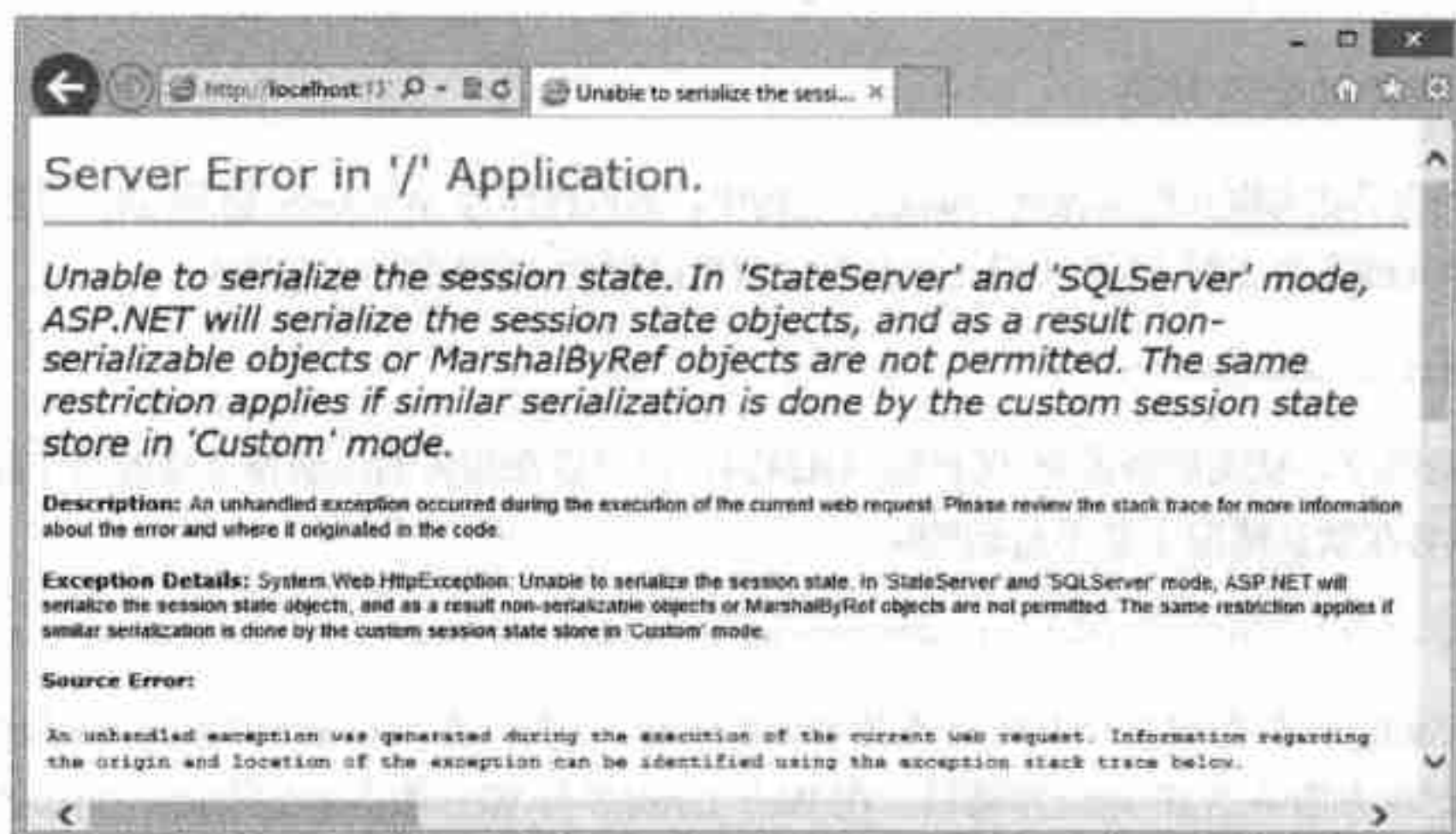


图 21-4

只要把对象标记为[Serializable]，它们就可以退出 ASP.NET 进程。注意，程序清单 21-5 中的对

象(代码文件 Person.cs)被标记为[Serializable]。

程序清单 21-5 可序列化的对象可以在进程外的会话中使用

```
using System;
using System.Web;

[Serializable]
public class Person
{
    public string firstName;
    public string lastName;
    public override string ToString()
    {
        return String.Format("Person Object: {0} {1}", firstName, lastName);
    }
}
```

把程序清单 21-5 中 Person 类的实例放在当前已配置为 StateServer 的 Session 对象中, 因此应在程序清单 21-3 的 Page 基类中添加一个强类型化的属性。在程序清单 21-6 中已添加了这个强类型化的属性(这个类重命名为 SmartSessionPage2, 使之更容易区分)。程序清单 21-6 在代码文件 SmartSessionPage2.cs 中。注意 Get 属性的类型转换, 强类型化的返回值表示这个属性只处理 Person 类型的对象。

程序清单 21-6 给 SmartSessionPage 类添加一个强类型化的属性

```
using System;
using System.Web;

public class SmartSessionPage2 : System.Web.UI.Page
{
    private const string MYPERSON = "myperson";

    public Person MyPerson
    {
        get
        {
            return (Person)Session[MYPERSON];
        }
        set
        {
            Session[MYPERSON] = value;
        }
    }
}
```

现在添加代码, 创建一个新的 Person 实例, 在文本框中填充它的字段, 把该实例放在现在处于进程外的会话状态服务中。之后, 提取 Person 实例, 使用程序清单 21-5 中重载的 ToString 方法将它的值写入浏览器。



Framework 类库中的一些类没有被标记为[Serializable]。如果在自己的对象中使用这种类型的对象，就不会序列化这些对象。例如，在类中包含 DataRow 字段，再把对象添加到状态服务中，就会收到消息 "... is not marked as serializable"，因为 DataRow 包含未序列化的对象。

在程序清单 21-7 中，将 TextBox 的值放在字符串数组中，将前两个字符串放在 Person 实例中。例如，如果输入 "Christian Wenz"，"Christian" 就放在 Person.firstName 中，而 "Wenz" 放在 Person.lastName 中。在后面的程序清单 21-8 中提取输入的值，并显示在浏览器上。

程序清单 21-7 在会话中使用状态服务和基本页面来设置对象

```
<%@ Page Language="C#" Inherits="SmartSessionPage2" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        string[] names = TextBox1.Text.Split(' ');
        Person p = new Person()
        {
            firstName = names[0],
            lastName = names[1]
        };
        MyPerson = p;
    }
</script>

<html>
<head id="Head1" runat="server">
    <title>Session State</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <asp:Button ID="Button1" runat="server" Text="Store in Session"
            OnClick="Button1_Click" />
        <br />
        <asp:HyperLink ID="HyperLink1" runat="server"
            NavigateUrl="Listing 21-08.aspx">Next Page</asp:HyperLink>
    </div>
    </form>
</body>
</html>
```

程序清单 21-8 在会话中使用状态服务和基本页面来提取对象

```
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        Person p = MyPerson;
```



```

        Label1.Text = p.ToString();
    }
}
</script>

<!DOCTYPE html>

<html>
<head id="Head1" runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="Label1" runat="server" />
        </div>
    </form>
</body>
</html>

```

现在打开浏览器,输入姓名,单击按钮,把它存储在会话中。然后通过超链接访问程序清单 21-8,结果如图 21-5 所示。

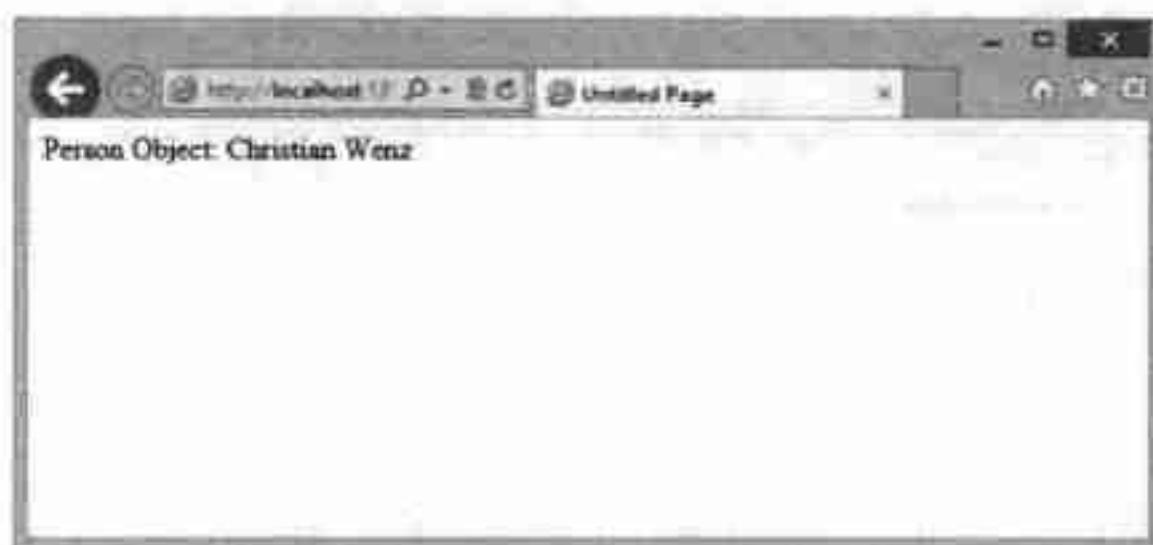


图 21-5

程序清单 21-7 和 21-8 中完整的代码和技术演示了会话管理的一些最佳实践:

- 如果使用非 InProc 会话状态,应把对象标记为[Serializable]。
- 最好在本地会话状态服务器上进行所有的开发。这会在早期找出不能序列化的对象,从而了解 aspnet_state.exe 的性能和内存使用情况,并可以在部署期间选择任意会话选项。
- 使用 Page 基类或辅助对象以及强类型化的属性来简化代码。这可以隐藏对会话键的类型转换,否则就要在代码中引用这些会话键。

这些最佳实践可应用于所有的状态存储方法,包括 SQL 会话状态(稍后会介绍)。在使用进程外的会话状态时,无论是像之前描述的那样来使用,还是使用 SQL 支持的会话状态,为了使应用程序能使用这种类型的状态,状态中存储的对象需要来回地进行序列化和反序列化。一些序列化后移到内存中的对象会非常大。

ASP.NET 4.x 包括一项新的功能——压缩存储在进程外状态中的对象,如下所示:

```

<sessionState
    mode="SqlServer"
    sqlConnectionString="data source=dbserver;Initial Catalog=aspnetstate"
    allowCustomSqlDatabase="true"
    compressionEnabled="true"
/>

```

当使用 `compressionEnabled` 特性允许压缩时, 可以使用 `System.IO.Compression.GzipStream` 类。默认情况下, `compressionEnabled` 特性设置为 `false`。通过进行压缩, 可以注意到用于存储状态的内存大小存在很大差异。

21.2.5 SQL 支持的会话状态

ASP.NET 会话还可以存储在 SQL Server 数据库中。InProc 提供了速度, 而 StateServer 提供了回应/速度的平衡。SQL Server 中存储的会话也有回应功能, 允许根据需要把会话用于大型 Web farm, 从而可以在 IIS 重新启动的过程中保存会话。

可以使用 `aspnet_regsql.exe` 配置 SQL 支持的会话状态。使用这个工具可以添加和删除对许多 ASP.NET 功能的支持, 例如高速缓存依赖性(详见第 22 章)、个性化/成员资格(第 18 和 19 章)以及会话支持。在命令行上运行不带选项的 `aspnet_regsql.exe`, 会打开如图 21-6 所示的 GUI。这个实用程序位于 .NET Framework 的安装目录下, 通常是 `C:\Windows\Microsoft .NET\Framework\v4.0.30319`。



图 21-6

在图 21-6 所示的对话框中, 其文本包含在命令行上运行带 `-?` 选项的 `aspnet_regsql` 的指令。`aspnet_regsql` 有非常多的选项, 因此应使用 `aspnet_regsql -? | more` 这样的形式显示可用的选项。表 21-3 中列出了用于会话状态的选项。

表 21-3

会话状态的选项	说 明
<code>-ssadd</code>	添加对 SQLServer 模式会话状态的支持
<code>-ssremove</code>	删除对 SQLServer 模式会话状态的支持
<code>-sstype t p c</code>	支持的会话状态类型如下: t: 表示临时(temporary)。会话状态数据存储在 tempdb 数据库中。管理会话的存储过程安装在 ASPState 数据库中。如果重新启动 SQL, 数据不会保存(默认) p: 表示永久保存(persisted)。会话状态数据和存储过程都存储在 ASPState 数据库中 c: 表示定制(custom)。会话状态数据和存储过程都存储在定制的数据库中。必须指定数据库名
<code>-d <database></code>	<code>-sstype</code> 选项是 c 时使用的定制数据库名

有3个选项支持会话状态：t、p和c。它们最重要的区别在于，-sstype t选项不在SQL Server重启的过程中永久保存会话状态数据，而-sstype p选项则永久保存会话状态数据。另外，可以使用-c选项指定定制数据库，使用-d database选项指定数据库的名称。

下面的命令行示例给系统配置了SQL会话支持，其中SQL Server位于localhost，sa密码是wrox，在ASPState数据库中永久存储(当然，不应使用sa和脆弱的密码来部署系统，但这样做会简化例子。理想情况下应使用Windows集成身份验证，并提供工作者进程身份来访问ASPState数据库)。如果使用SQL Express，就使用\SQLEXPRESS替代localhost。如果不使用Windows集成身份验证，就需要从Management Studio中明确激活sa账户，运行这个工具，再禁用sa账户以确保安全。

```
C:\>aspnet_regsql -S localhost -U sa -P wrox -ssadd -sstype p
Microsoft (R) ASP.NET SQL Registration Tool version 4.0.30319.17929
Administrative utility to install and uninstall ASP.NET features on a SQL server.
Copyright (C) Microsoft Corporation. All rights reserved.

Start adding session state.
.....
Finished.
```

使用可信的连接(和Windows集成身份验证)时，使用-E选项：

```
C:\>aspnet_regsql -S localhost -E -ssadd -sstype p
```

接着，打开SQL Server Management Studio(或其Express版本)，查看新建的数据库。其中创建了两个表：ASPStateTempApplications和ASPStateTempSessions，并且创建了一系列存储过程，以支持在SQL和内存之间来回移动会话。

如果对SQL Server进行了安全锁定，在执行aspnet_regsql.exe后就会得到错误15501，该错误是在SQL文件InstallSqlState.sql的执行过程中出现的错误。SQL错误号是15501，并且SqlException消息是：

“这个模块被标记为OFF。打开Agent XPs，就可以访问该模块。如果未进行这项工作，msdb.dbo.sp_delete_job中就会出现错误”。

这是一条相当模糊的消息，但aspnet_regsql.exe试图说明，它需要已扩展的存储过程来支持会话状态，但这些存储过程因安全原因未得到支持。我们需要明确支持它们。为此，在SQL Server Management Studio Express中执行下面的命令：

```
USE master
EXECUTE sp_configure 'show advanced options', 1
RECONFIGURE WITH OVERRIDE
GO
EXECUTE sp_configure 'Agent XPs', 1
RECONFIGURE WITH OVERRIDE
GO
EXECUTE sp_configure 'show advanced options', 0
RECONFIGURE WITH OVERRIDE
GO
```

现在修改web.config文件的<sessionState>元素，以使用SQL Server和新的连接字符串：


```
<sessionState mode="SQLServer" sqlConnectionString="data source=127.0.0.1;user id=sa;password=Wrox" />
```

对于可信的连接，连接字符串如下：

```
<sessionState mode="SQLServer"
  sqlConnectionString="data source=127.0.0.1;trusted_connection=yes" />
```

程序清单 21-7 和 21-8 中的会话代码与前面代码的工作方式一样。但是，如果打开 ASPStateTempSessions 表，就会看到已序列化的对象。注意在图 21-7 中，跟踪信息中的会话 ID 在 ASPStateTempSessions 表的一行中显示为主键。

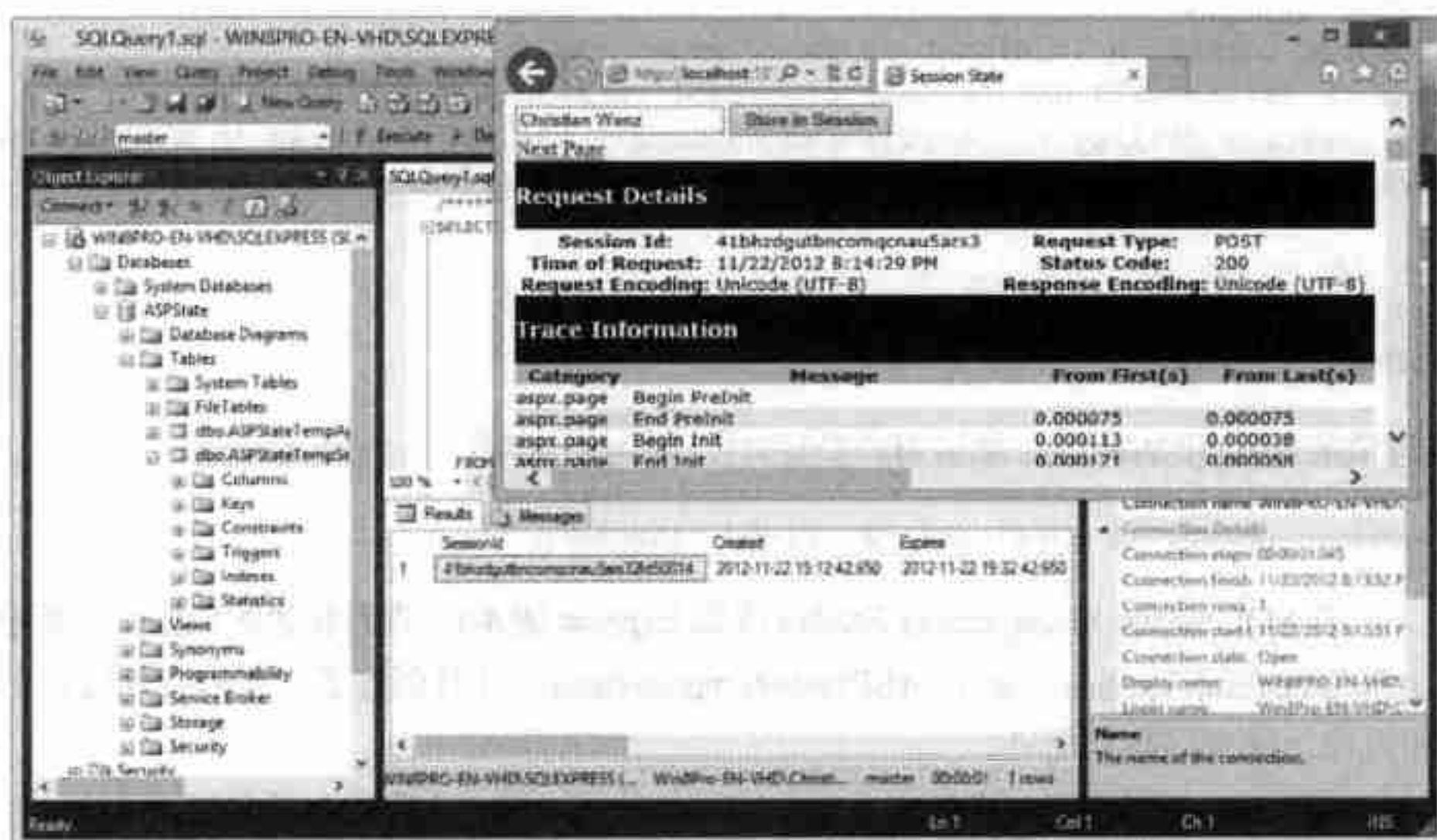


图 21-7

图 21-7 显示了 ASP.NET 跟踪信息的 Request Details 部分的 SessionId，在刚才创建的 ASPState 数据库中，该 SessionId 显示在 ASPStateTempSessions 表的 SessionId 列中。还要注意，ASPStateTempApplications 表跟踪使用同一数据库管理会话的每个 IIS 应用程序。

如果要使用自己的数据库存储会话状态，就使用 aspnet_regsql.exe 的 -d <database> 选项指定数据库名，并在连接字符串中包含 allowCustomSqlDatabase="true" 和数据库名：

```
<sessionState allowCustomSqlDatabase="true" mode="SQLServer"
  sqlConnectionString="data source=127.0.0.1; database=MyCustomASPStateDatabase;" />
```

用户 ID 和密码可以包含在连接字符串中。如果 ASP.NET 工作者进程的身份配置为在 SQL Server 中访问，就可以使用 Windows 集成身份验证。

21.2.6 使用其他提供程序扩展会话状态

ASP.NET 会话状态建立在可扩展、基于提供程序的存储模型之上。通过从 SessionStateStoreProviderBase 中派生，可以实现定制的提供程序，在其他存储机制中存储会话数据。这个扩展功能还可以实现 ISessionIDManager，通过自己的算法生成会话 ID。

首先创建一个继承自 SessionStateStoreProviderBase 的类。会话模块只要派生于 Session-

StateStoreProviderBase, 就会调用会话提供程序的方法。在应用程序的 web.config 文件中注册定制的提供程序, 如下所示:

```
<sessionState mode="Custom" customProvider="WroxProvider">
  <providers>
    <add name="WroxProvider" type="Wrox.WroxStore, WroxSessionSupplier"/>
  </providers>
</sessionState>
```

ASP.NET 首先初始化 SessionStateModule, 然后在定制的实现代码中调用下面这些方法:

- **Initialize:** 这个方法最终继承自 System.Configuration.Provider.ProviderBase, 在构造函数执行后立即被调用。使用这个方法可以给提供程序设置名称, 调用 Initialize 的基本实现代码。
- **SetItemExpireCallback:** 使用这个方法可以注册会话项到期时调用的方法。
- **InitializeRequest:** 这个方法由 SessionStateModule 为每个请求调用。可以在早期为请求获得数据。
- **CreateNewStoreData:** 使用这个方法可以创建 SessionStateStoreData 的一个新实例、保存会话项的数据结构、会话超时值以及其他静态项。

在请求会话项时, ASP.NET 会调用实现代码来提取它。实现下面的方法可以提取会话项:

- **GetItemExclusive:** 这个方法可以从选中的数据库中获得 SessionStateStoreData。我们可以创建 Oracle 提供程序, 在 XML 或其他地方存储数据等。
- **GetItem:** 这个方法可以像 GetItemExclusive 那样获得 SessionStateStoreData, 但不需要独占锁定。这取决于我们选择的数据存储。

下面应该存储项, 调用下面的方法:

- **SetAndReleaseItemExclusive:** 这个方法把 SessionStateStoreData 对象保存在定制数据存储中。



ScaleOut 软件以 StateServer 产品的形式发布了第一个第三方 ASP.NET 状态提供程序, 它填补了 ASP.NET 中单一的 StateServer 和 SQL Server 数据库状态提供程序之间的空白。ScaleOut 软件的 StateServer 是一个进程外的服务, 可运行在 Web farm 的每台计算机上, 确保会话状态以透明、分布的方式存储在 Web farm 中的计算机上。<http://www.scaleoutsoftware.com> 上提供了 StateServer 及其 ASP.NET 会话提供程序的更多信息。

21.2.7 无 cookie 的会话状态

在上面的例子中, ASP.NET 会话状态 ID 存储在 cookie 中。一些设备不支持 cookie, 或者用户可能关闭了浏览器中的 cookie 支持功能。cookie 非常方便, 因为其中的值可以在每个请求和响应之间来回传送。也就是说, 每个 HttpRequest 都包含 cookie 值, 每个 HttpResponse 也都包含 cookie 值。那么, 在每个请求和响应之间来回传送的另一种内容是什么呢? 答案是 URL。

如果在 web.config 文件中包含 cookieless="UseUri", ASP.NET 就不会将 ASP.NET 会话 ID 作为 cookie 回送, 而是修改每个 URL, 使其在请求页面的前面包含会话 ID。

```
<sessionState mode="SQLServer" cookieless="UseUri" sqlConnectionString="data
```



```
source=127.0.0.1;user_id=sa;password=Wrox" />
```

注意，会话 ID 出现在 URL 中，就好比位于实际 Web 站点的虚拟目录和页面之间的目录。进行此修改后，服务器端的用户控件，如程序清单 21-1 中使用的 HyperLink 控件，会自动修改其属性。程序清单 21-1 中的链接可以直接在设计器中硬编码为 HTML，但 ASP.NET 不会修改目标 URL，如图 21-8 所示。

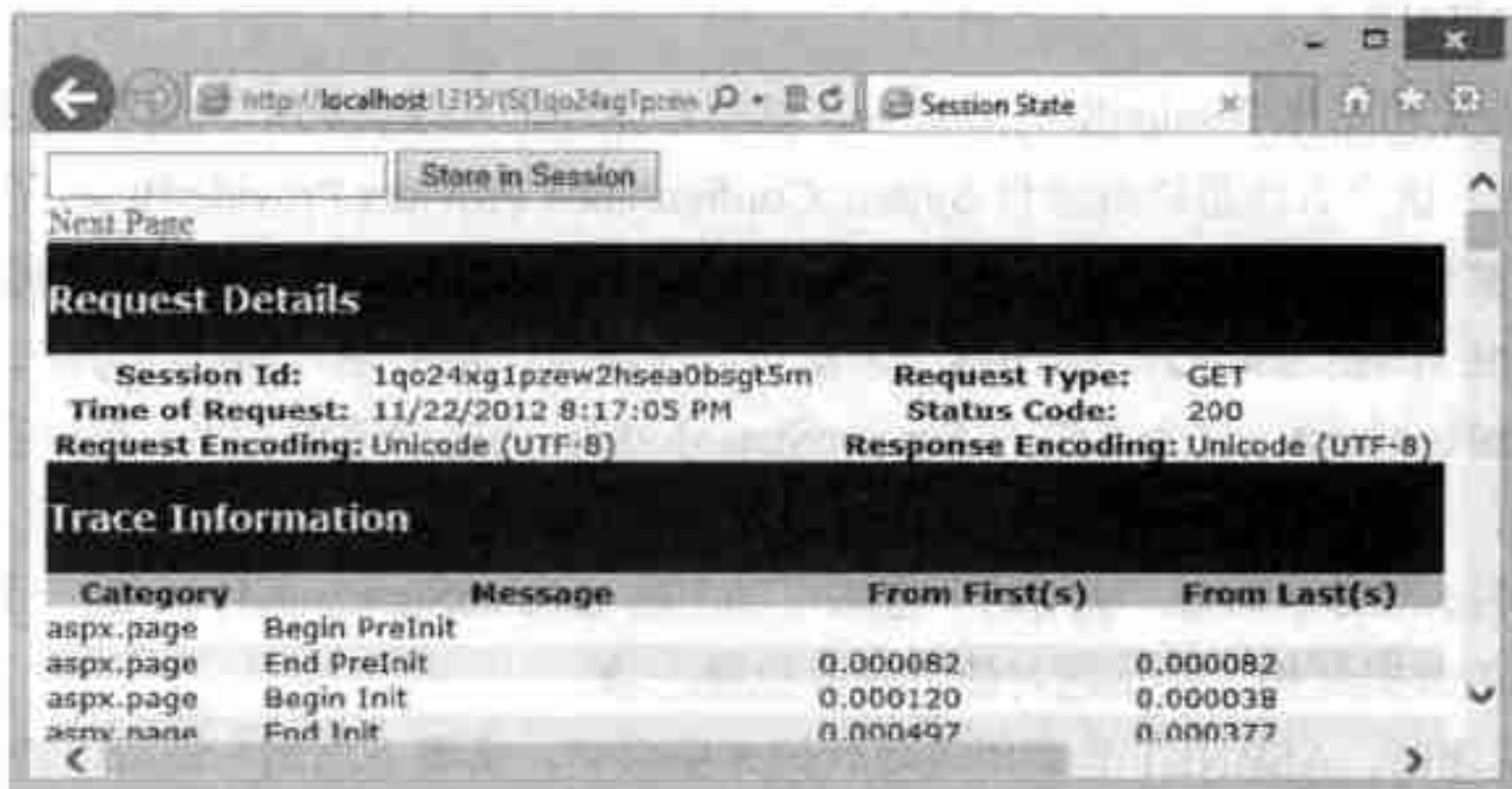


图 21-8

会话 ID 是字符串，其中只包含 URL 中允许具有的 ASCII 字符。从基于 cookie 的会话状态系统迁移到无 cookie 的系统时，需要把会话状态值放在 URL 中。

在图 21-8 中，请求 URL 在括号中包含会话 ID。无 cookie 的会话的缺点是，它们很容易被篡改。当然，使用 HTTP 嗅探器很容易修改 cookie，但 URL 可以由任何人修改。维护会话状态的唯一方式是每个 URL 都以这种方式包含会话 ID。

另外，所有的 URL 都必须是相对路径。注意，会话 ID 看上去似乎是目录。如果调用绝对 URL，如/myapp/retrieve.aspx，会话就会丢失。如果正在服务器端生成 URL，可使用 `HttpResponse.ApplyAppPathModifier` 方法，该方法在嵌入会话 ID 时修改 URL，如下所示：

```
Response.Write(Response.ApplyAppPathModifier("my/file.aspx"));
```

这行代码生成如下 URL：

```
/myapp/(S(avkbnbm14nln5mi5dmfqnu45))/my/file.aspx
```

注意，这行代码不仅把会话信息添加到 URL 中，还会把相对 URL 转换为绝对 URL，使其包含应用程序的虚拟目录。在需要使用 `Response.Redirect` 或手动建立 URL 以从 HTTP 页面重定向到 HTTPS 页面，同时维护无 cookie 的会话状态时，就可以使用这个方法。

21.2.8 选择维护状态的正确方式

熟悉了 ASP.NET 中维护状态的各种选项后，下面是一些实际建议。进程中(InProc)的会话提供程序是最快速的方法，因为内存中的信息是活动的对象引用。这个提供程序保存在 `HttpApplication` 的缓存中，因此对应用程序的再次使用非常敏感。如果使用的是 Windows 2000 Server 或 Windows XP，就由 `aspnet_wp.exe` 进程管理 ASP.NET HTTP 管道。如果运行的是 Windows Server 2008 或 2012，

Windows 7 或 8, w3wp.exe 就是管理运行情况的默认进程。

必须在进程外状态服务的健壮性和进程中提供程序的速度之间找到平衡点。根据笔者的经验,进程外的状态服务一般比进程中的提供程序慢 15%, 因为进程外的状态服务有序列化系统开销和编组。SQL 会话状态比 InProc 慢 25%。当然,每个人的使用情况都不同,不要太在意这些数字。在作出决策前,一定要对应用程序进行可伸缩性测试。



建议所有的开发人员在开发过程中都使用进程外的会话状态,即使这不是应用程序的部署方式。一定要使用进程外的提供程序,才能找出没有 Serializable 特性的定制对象的潜在问题。如果使用进程中的提供程序设计整个站点,而在项目的后期发现需要切换到 SQL 或进程外的提供程序,就不能保证站点按照希望的方式工作。使用进程外的提供程序开发,可以得到最佳效果,并且不会影响最终的部署方法。这是一种非常保险的策略,事先没有什么系统开销。

21.3 Application 对象

Application 对象等价于 ASP.NET 应用程序中的全局变量库。多年以来,全局变量在其他编程环境中都被认为是有害的,ASP.NET 也不例外。应考虑要在 Application 对象中放置什么内容以及原因。使用较灵活的 Cache 对象常常更有效,它有助于控制对象的生存期。有关高速缓存的内容,将在第 22 章讨论。

Application 对象对整台计算机来说不是全局的,而对 HttpApplication 来说是全局的。如果在 Web farm 中运行,那么每台 Web 服务器上的每个 ASP.NET 应用程序都有自己的 Application 对象。ASP.NET 应用程序是多线程的,它接收的请求由多个线程中的代码处理,因此应使用 Application.Lock 和 Application.Unlock 方法管理对 Application 对象的访问。如果代码没有直接调用 Unlock 方法,那么在最初调用 Lock 方法的 HttpRequest 末尾就会隐式地去除锁定。

下面的例子在插入对象之前锁定 Application 对象。其他尝试写入 Application 对象的线程会等待该对象解除锁定。这个例子假定在 Application 对象的 GlobalCount 键中已经存储了一个整数。

```
Application.Lock();
Application["GlobalCount"] = (int)Application["GlobalCount"] + 1;
Application.Unlock();
```

对象引用可以存储在 Application 对象中,这一点与 Session 对象一样,但在检索时它们必须转换回已知的类型,如前面的代码所示。

21.4 查询字符串

URL 或查询字符串是存储用于导航的(不是用于用户的)数据的理想位置。查询字符串是 Web 站点上最容易被攻击的元素,这可能对我们有帮助,也可能有害。例如,如果导航方案在查询字符串

的最后使用页面 ID(如/localhost/mypage.aspx?id=54), 用户就可以在浏览器中使用该 URL, 尝试 id 的每个值。不要盲目地把 id 强制转换为整数, 如果这么做, 就应制定故障规划。最好是在某个人把 URL 改为不合理的值时返回 `Response.StatusCode = 404`。Amazon.com 采用了另一个方法, 即 Smart 404: “抱歉, 没有找到需要的内容, 你的意思是_____?”。

注意, URL 是用户看到的第一项内容, 甚至在看到 HTML 之前。可攻击的 URL 会使站点的可访问性更高。下面哪个 URL 更友好、可攻击性(有正当的理由)更强?

`http://reviews.cnet.com/Philips_42PF9996/4505-6482_7-31081946.html?tag=cnetfd.sd`

或

`http://www.hanselman.com/blog/CategoryView.aspx?category=Movies`

21.5 cookie

想想 cookie 是何时引入的。大多数用户都不知道 cookie 是什么, 但他们都相信 cookie 是罪恶的, 存储了他们的私人信息。回想一下, 似乎私人信息确实是存储在 cookie 中。但实际上并非如此, 在 cookie 中甚至还存储了敏感信息, 如用户 ID 或密码。cookie 应只用于存储不敏感的信息或可以从权威来源中获得的信息。不应信任 cookie, 它们的内容应能进行有效性验证。例如, 如果表单验证 cookie 被篡改, 就注销用户, 抛出异常。如果给已到期的会话传送无效的会话 ID cookie, 就可以指定新的 cookie。

在 cookie 中存储信息(使用 `Response.Cookies`, 读取信息使用 `Request.Cookies`)与在 Session 对象中存储数据是完全不同的:

- cookie 在每个请求中来回传送。也就是说, cookie 要占用每个 HTTP GET 和 HTTP POST 的空间。如果页面上有 10 个 1 像素的 GIF 占位符用于表布局, 那么用户的浏览器就要发送同一 cookie 11 次: 一次用于页面, 10 次用于 GIF 占位符, 即使 GIF 已缓存也是如此。
- cookie 可以被窃取、嗅探或伪造。如果代码使用了 cookie 的值, 就应在代码中做规划, 以防 cookie 被破坏或篡改。
- 如果 cookie 缺失, 应用程序会如何? 如果 cookie 有 4096 字节, 应用程序又会如何? 在设计应用程序时应遵循“异常最少”的规则。如果 cookie 缺失或超过了期望的空间, 应用程序应尝试处理这些情况。
- 在使用 Base64 编码任何较大的内容并将其放在 cookie 中之前要三思而行。如果我们的设计依赖这种技术, 就应考虑使用会话或另一种支持数据存储的技术。

21.6 回送和跨页面回送

ASP.NET 使用回送的概念, 在这个概念中, 要引发一个服务器端的事件, 以告诉开发人员发生了客户端操作。如果在浏览器上单击按钮, Form 集合就回送到服务器上, 现在 ASP.NET 允许开发人员在 `Button1_Click` 和 `TextBox1_Changed` 等事件中编写代码。

但是,这种回送到同一页面的技术并不直观,尤其是在设计用户界面,用于创建向导,告诉用户下一个操作是什么时。

本章讨论状态管理的所有方面。但是,回送和跨页面回送在第3章已详细介绍过,因此本章只在状态管理的领域内探讨它们。回送在 ASP.NET 1.x 中引入,为 Web 开发提供了事件子系统。ASP.NET 1.x 中只有单页面回送,不太方便,开发人员必须在会话中存储回送的小型对象,再重定向到下一个页面,提取出存储的数据。而有了跨页面回送,数据就可以传送到另一个页面,不再需要存储直接传送给另一个页面的少量数据。

ASP.NET 2.0 及更高版本给所有的 Button 控件添加了 PostBackUrl 属性,包括 LinkButton 和 ImageButton 控件。控件在 ASPX 页面中显示时,PostBackUrl 属性是标记的一部分(如下代码所示),也是可在隐藏代码中使用的服务器组件的属性:

```
<asp:Button PostBackUrl="url" ...>
```

单击设置了 PostBackUrl 属性的 Button 控件后,页面不会回送给它自己,而是传送给由该 Button 控件的 PostBackUrl 属性指定的 URL。在发出跨页面的请求时,当前 Page 类的 PreviousPage 属性会保存进行回送的页面的引用。要从 PreviousPage 属性中获得控件引用,可以使用 Controls 属性或 FindControl 方法(需要强制类型转换),或者 @PreviousPageType 特性。

创建一个新页面,如程序清单 21-9 所示。在其中放置一个文本框和一个按钮,把按钮的 PostBackUrl 属性设置为 Listing21-10.aspx。然后创建带标签的 Listing21-10.aspx 页面,双击 HTML 设计器,添加 Page_Load 处理程序。然后,程序清单 21-10 包含检索和处理回送的代码。

程序清单 21-9 跨页面回送: 发送者

```
<%@ Page Language="C#" %>

<!DOCTYPE html>

<script runat="server">

</script>

<html>
<head id="Head1" runat="server">
    <title>Cross-page PostBacks</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
            <asp:Button ID="Button1" runat="server" Text="Button"
                PostBackUrl="~/Listing 21-10.aspx" />
        </div>
    </form>
</body>
</html>
```


程序清单 21-10 跨页面回送: 接收者

```

<%@ Page Language="C#" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        if(PreviousPage != null && PreviousPage.IsCrossPagePostBack)
        {
            TextBox text = PreviousPage.FindControl("TextBox1") as TextBox;
            if(text != null)
            {
                Label1.Text = text.Text;
            }
        }
    }
</script>
<html>
<head id="Head1" runat="server">
    <title>Step 2</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label></div>
        </form>
    </body>
</html>

```

在程序清单 21-9 中, 页面传送给 Listing21-10.aspx, Listing21-10.aspx 再访问 Page.PreviousPage 属性, 并提取引起回送的 Page 类的已填充实例。通过对 FindControl 进行调用和类型转换, 把文本框从前面的页面中提取出来, 并把它值复制到程序清单 21-10 的标签中。

21.7 隐藏字段、ViewState 和 ControlState

隐藏的输入字段, 如<input type="hidden" name="myName">, 在表单 POST 中作为名/值对发送回来, 这一点与其他控件一样, 但它们不显示出来。可以把它们看作隐藏的文本框。图 21-9 在 Visual Studio 设计器中显示了一个 HiddenField 控件及其属性。可以在 ASP.NET 的所有版本中使用隐藏字段。

与 Session 对象一样, ViewState 是键/值对的集合, 但显示为名为 __VIEWSTATE 的隐藏字段, 如下所示:

```

<input type="hidden" name="__VIEWSTATE" value="/AAASSDAS ... Y/1OI=" />

```

放在 ViewState 中的对象都必须标记为 Serializable。ViewState 使用一种特殊的二进制格式化程序 LosFormatter 序列化对象, 其中 Los 表示有限的对象序列化(Limited Object Serialization)。它可以

序列化任意类型的对象，并且对自身进行了优化，以包含字符串、数组和散列表。

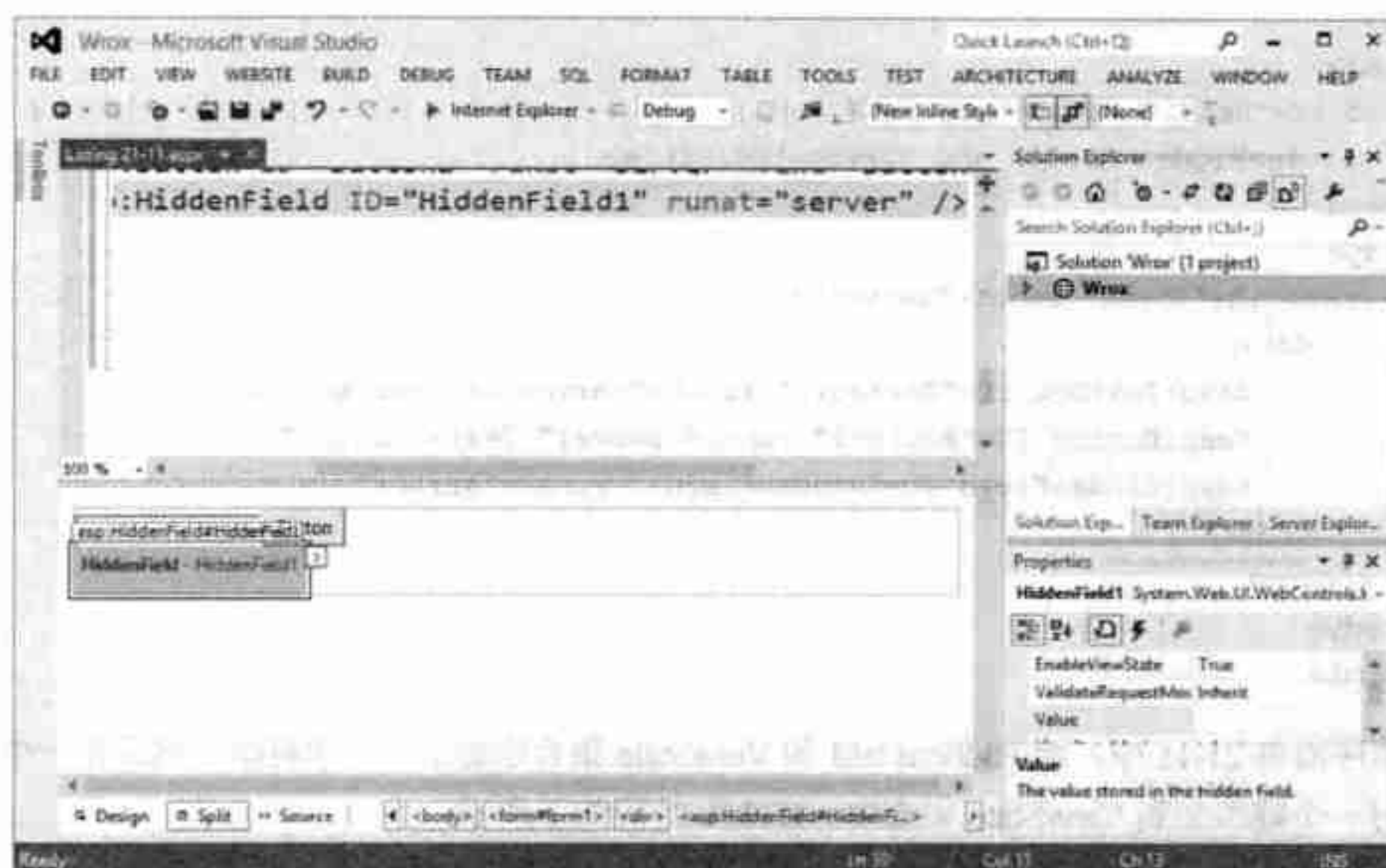


图 21-9

为了说明该格式化程序的工作情况，创建一个新页面，在其中拖放 TextBox、Button 和 HiddenField 控件。双击设计器，创建 Page_Load 事件，添加如程序清单 21-11 所示的代码。这个例子给 HiddenField.Value 属性添加一个字符串，给 ViewState 集合添加一个 Person 实例。这个程序清单演示了如何在客户端的 HTML 文本框中保存 ViewState，它可以包含简单的类型(如字符串)，也可以包含复杂的类型(如 Person)。自 ASP.NET 1.x 引入以来，这种技术一直是保存少量数据并且不需要使用服务器资源的一种强大而简单的方式。

程序清单 21-11 隐藏字段和 ViewState

```
<%@ Page Language="C#" %>
```

```
<script runat="server">
```

```
protected void Page_Load(object sender, EventArgs e)
```

```
{
```

```
if (!Page.IsPostBack)
```

```
{
```

```
HiddenField1.Value = "value 1";
```

```
ViewState["AnotherHiddenValue"] = "value 2";
```

```
Person p = new Person()
```

```
{
```

```
firstName = "Christian",
```

```
lastName = "Wenz"
```

```
};
```

```
ViewState["HiddenPerson"] = p;
```

```
}
```

```
}
```

```
</script>
```

```

<!DOCTYPE html>

<html>
<head id="Head1" runat="server">
    <title>Hidden Fields and ViewState</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
            <asp:Button ID="Button1" runat="server" Text="Button" />
            <asp:HiddenField ID="HiddenField1" runat="server" />
        </div>
    </form>
</body>
</html>

```

在程序清单 21-11 中, 给 HiddenField 和 ViewState 集合添加了一个字符串。然后把一个 Person 实例和另一个键添加到 ViewState 集合中。HTML 片段如下所示:

```

<form method="post" action="Listing 21-11.aspx" id="form1">
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="U77CNOTJ/2LW1BwWh4u
hvq+FGDtavwu+NmR3cqXa9patotBdiioisxrbCrqe3FQqBk/rvwgCnNC2VzdUcXN+bUx3+M0wZloqAMCTrvZ
QqnYyc/KG+YoMT0+4httTnyCfNKYBzDwZtQQhLCOJUzRPlduaS/YYFgze3t+ORW6fnRZ/ai8Pfn4LPS+ejoV
woY7lmkCO2iIHLzWjU0g7lsemPjpjV1lKN+asG8IF2rb5vvGP6RPckWZ1P6e0EMWlWKZcZaaJ9j0toeEcCYZ
SU/GwGvablSPUSV39onQ3a+UzEG0dzmg6jqaw/tQUahBkudZpZ7dbirUqaul642Q3FIjkadibrS9zUQn/KAP
898X+1QvvYgL0IOwpf1G5Jk6Xe6++" />
</div>
    <div>
        <input name="TextBox1" type="text" id="TextBox1" />
        <input type="submit" name="Button1" value="Button" id="Button1" />
        <input type="hidden" name="HiddenField1" id="HiddenField1" value="foo" />
    </div>
</form>

```

注意, ViewState 值只使用有效的 ASCII 字符表示它的所有内容, 因此不要被其迷惑。它很庞大, 看起来很模糊。但是, 它只是一个隐藏的文本框, 会自动回送到服务器上。整个 ViewState 集合可以在 Page_Load 事件中使用。HiddenField 的值存储为纯文本。

ViewState 和 HiddenField 都不能接受敏感数据。

在默认情况下, ViewState 字段和随机化的散列(salted hash)一起发送到客户端, 以避免被篡改。随机化的散列表示, ViewState 的数据在编码前要附加唯一值。Keith Brown 说过, “盐只是炖肉的一种调味品”。这里使用的技术称为 HMAC 或散列的消息验证代码。可以使用 web.config 文件的 <machineKey>元素指定 validationKey, 以及用于保护 ViewState 的算法。该文件的这一部分和 decryptionKey 特性也会影响表单验证 cookie 的加密方式(有关表单验证的内容, 详见第 20 章), 如下所示:


```
<machineKey validationKey="AutoGenerate,IsolateApps"
decryptionKey="AutoGenerate,IsolateApps" validation="SHA1" / >
```

如果在 Web farm 中运行应用程序, <validationKey>和<decryptionKey>就必须手动设置为相同的值。否则, 一台计算机生成的 ViewState 会回送到 Web farm 中有另一个键的计算机上。键最长应有 128 个字符, 以完全随机的方式生成。如果给它们的值添加 IsolateApps, ASP.NET 就会使用应用程序 ID 为每个应用程序生成唯一的加密键。

可以将 validation 特性设置为 SHA1 或 MD5, 以防止被篡改, 也可以通过加密 ViewState 增加一层保护。ASP.NET 提供了一个解密特性, 仅用于为表单验证票证指定加密和解密机制。validation 特性仅用于 ViewState, 现在可以使用 3DES 或 AES 以及存储在 validationKey 特性中的键来加密。

ASP.NET 4.x 还在<pages>配置元素中添加了 ViewStateEncryptionMode 特性, 它有两个值: Auto 和 Always。把该特性设置为 Always, 会强制加密 ViewState; 而把它设置为 Auto, 则仅在控件请求加密时才使用新的方法 Page.RegisterRequiresViewStateEncryption 加密 ViewState。

在 Page_Init 中把 Page.ViewStateUserKey 属性设置为唯一值, 例如用户 ID, 就可以将增加的保护措施应用于 ViewState。必须在 Page_Init 中设置该属性, 因为键应在 ViewState 加载或生成之前提供给 ASP.NET。例如:

```
protected void Page_Init (Object sender, EventArgs e)
{
    if (User.Identity.IsAuthenticated) {
        ViewStateUserKey = User.Identity.Name;
    }
}
```

在优化页面时, ASP.NET 程序员常常在不需要额外状态时禁用许多控件的 ViewState。从而使需要状态信息的控件仍能工作。ASP.NET 现在包含第二个与 ViewState 类似的集合 ControlState。这个字典可以用于存储不应禁用(甚至在使用 ViewState 时也不应禁用)并且规模受限的重要信息。在 ControlState 集合中只应存储对控件的功能至关重要的数据。

ViewState 和 ControlState 虽然不太安全, 但适合于存储少量不属于 cookie 或 Session 对象的数据和状态。如果存储的数据相当少, 并且是页面特定实例的本地数据, 使用 ViewState 就比使用包含许多瞬态数据的 Session 对象好得多。

21.8 为短时间状态存储应用 HttpContext.Current.Items

HttpContext 的 Items 集合是 ASP.NET 保守得最好的秘密之一, 它是 IDictionary 键/值对的对象集合, 在单个 HttpRequest 的生存期内共享。该集合只存在于单个 HttpRequest 中。状态为什么要存储这么短的时间? 有如下原因:

- 在 IHttpModule 和 IHttpHandler 之间共享内容: 如果编写定制的 IHttpModule, 就可以存储用户的信息, 供以后在页面中使用。

- 在同一页面上相同 UserControl 的两个实例之间通信：假定编写一个用于横幅广告的用户控件。该控件的两个实例可以从 HttpContext.Items 中选择广告，以防止在同一页面上显示重复的内容。
- 存储成本很高的调用的结果，防止该调用在页面上出现多次：如果有多个 UserControl，每个 UserControl 都从很昂贵的大型数据库中提取并显示一组数据，这些 UserControl 就可以从 HttpContext.Items 中提取需要的数据。只使用一次数据库。
- HttpRequest 中的各个单元需要处理相同或类似的数据：如果数据的生存期只是请求，就可以考虑使用 HttpContext.Items 作为短期的高速缓存。

Items 集合存储对象，这一点与本章前面使用的许多集合相同。在检索时，需要把这些对象强制转换回特定的类型。

在支持 Web 的数据库访问层中，使用下面的编码模式很容易实现请求的预先高速缓存。注意，这个示例代码是设计模式，其中不包含 MyData 类，仅用于演示：

```
public static MyData GetExpensiveData(int ID)
{
    string key = "data" + ID.ToString();
    MyData d = (MyData) HttpContext.Current.Items[key];
    if (d == null)
    {
        d = new Data();
        //Go to the Database, do whatever...
        HttpContext.Current.Items[key] = d;
    }
    return d;
}
```

这段代码检查当前 HttpContext 的 Items 集合，看看其中是否已有数据。如果没有数据，就从相应的支持数据存储中提取，然后存储在 Items 集合中。在同一 HttpRequest 中对这个函数的后续调用都会得到已高速缓存的对象。

与所有的优化和高速缓存一样，不成熟的优化是所有罪恶的祸根。应仔细考虑对高速缓存的需求和改进情况。不要仅凭感觉就进行高速缓存，而应在确实需要时进行高速缓存。

21.9 本章小结

本章探讨了 ASP.NET 应用程序中管理状态的许多方式。Session 对象及其提供程序提供了多种选择。在以对象引用和序列化对象的方式(从而对应用程序来说，状态的管理就是透明的)管理状态时，每种选择都有自己的优缺点。服务器端的会话状态数据可以在 cookie 或 URL 中存储唯一标识键。cookie 也可以独立使用，存储少量数据，并在访问之间保留这些数据，但 cookie 存储的数据量很少，并且类型较简单。有多个页面时，隐藏字段、ViewState、ControlState、回送和跨页面回送可以管理少量的状态数据。HttpContext.Current.Items 可以存储仅在 HttpRequest 生存期内存在的过渡状态。查询字符串是一种古老的方式，可以存储适合于导航的非私有状态。

ASP.NET 的后续版本使用灵活的会话状态提供程序模块改进了 ASP.NET 1.1 的状态管理选项，增加了用于用户控件的 ControlState，为跨页面回送功能提供了更成熟的编程模型。

第22章

高速缓存

本章要点

- 使用 ASP.NET 中的高速缓存功能
- 使用编程的方式进行高速缓存
- SQL 高速缓存依赖性
- 使用和测试 SQL 高速缓存禁用功能

性能是我们开发的任何应用程序或代码的关键指标之一。浏览器有助于文本和图像的客户端高速缓存，而服务器端的高速缓存对于获得最佳性能至关重要。高速缓存是把频繁使用的数据存储在服务器上以执行后续请求的过程。每次请求对象时，与再次从头开始创建 Web 页面或它们包含的项相比，从内存中提取对象要快得多。高速缓存可以提高应用程序的性能、可伸缩性和可用性。对应用程序的高速缓存方式进行的调整越细致，高速缓存执行起来就越好。

本章主要讨论高速缓存，包括 ASP.NET 提供的 SQL 高速缓存禁用功能。本章将详细介绍高速缓存的这个独特方面。在使用 SQL 高速缓存禁用功能时，如果 SQL Server 的结果集有变化，输出高速缓存就会自动修改。这可以确保终端用户总是看到最新的结果集，所显示的数据永远不会过期。在介绍 SQL 高速缓存禁用功能之后，本章还将研究其他性能改进方法，讨论 Post-Cache Substitution 功能，它可以高速缓存整个页面，同时动态替换特定的内容。最后，本章还将介绍让开发人员创建定制依赖性的功能。

22.1 高速缓存

在 ASP.NET 中，开发人员可以通过几种方式处理高速缓存。一种方式是可以使用输出高速缓存机制，高速缓存整个 HTTP 响应(整个 Web 页面)。另外两种方式是部分页面高速缓存和数据高速缓存。下面就描述这些方式。

22.1.1 输出高速缓存

输出高速缓存是把动态生成的页面内容保存在服务器的内存或磁盘中以供以后检索的方式。这类高速缓存保存以后要显示的内容，这样下次请求这些内容时，就不需要再次生成它们。高速缓存页面后，就可以在对服务器进行后续的请求时使用。要应用输出高速缓存，应在.aspx 页面的顶部插入一条 OutputCache 页面指令，如下所示：

```
<%@ OutputCache Duration="60" VaryByParam="None" %>
```

Duration 特性定义了页面在高速缓存中的存储时间(秒)。VaryByParam 特性确定高速缓存页面输出的版本。根据所需要的 HTTP-POST 或 HTTP-GET 响应可以生成不同的响应。除了 OutputCache 指令的特性之外，ASP.NET 还包含 VaryByHeader、VaryByCustom、VaryByControl 和 Location 特性。另外，Shared 特性也会影响用户控件。

ASP.NET 中的高速缓存实现为 HttpModule，它通过 ASP.NET 工作者进程监听所有的 HttpRequest。OutputCacheModule 监听应用程序的 ResolveRequestCache 和 UpdateRequestCache 事件，处理高速缓存的请求和失败，返回高速缓存的 HTML，并且可以根据需要绕过页面处理程序。

1. VaryByParam 特性

VaryByParam 特性可以指定哪些 QueryString 参数会高速缓存页面的新版本：

```
<%@ OutputCache Duration="90" VaryByParam="pageId;subPageId" %>
```

例如，页面 navigation.aspx 在 QueryString 中包含导航信息，如 pageId 和 subPageId，上面的 OutputCache 指令就为 pageId 和 subPageId 的每个不同的值高速缓存页面。在这个例子中，最好使用等式表示页面的数量：

```
cacheItems = (num of pageIds) * (num of subPageIds)
```

其中，cacheItems 是存储在高速缓存中的 HTML 页面数量。页面仅在请求并传送给 OutputCacheModule 后才高速缓存。只有在至少访问每个组合一次后，才使用高速缓存的最大内存量。尽管这些都是潜在的最大值，但创建等式，表示系统中潜在的最大值是一次很重要的练习。

如果要根据 QueryString 参数中的区别高速缓存页面的新版本，应使用 VaryByParam="*"，如下所示：

```
<%@ OutputCache Duration="90" VaryByParam="*" %>
```

使用 VaryBy 特性时，一定要执行计算。例如，可以添加 VaryByHeader，根据浏览器报告的 User-Agent HTTP 报头，高速缓存页面的另一个版本：

```
<%@ OutputCache Duration="90" VaryByParam="*" VaryByHeader="User-Agent" %>
```

User-Agent 标识用户的浏览器类型。ASP.NET 可以自动生成给定页面的不同显示结果，用于特定的浏览器。因此在许多情况下，在高速缓存中保存这些显示结果是有意义的。Firefox 用户拥有的 HTML 与 IIS 用户略有不同，因此不应给所有的用户发送完全相同的 HTML，用于以后的显示。存在数十个 User-Agent 字符串，因为它们可以识别多种浏览器类型。OutputCache 指令可以根据服务器负载生成成千上万个不同的高速缓存页面版本。此时应根据动态重新创建页面的成本，计算高

速缓存的成本。



高速缓存总是可以获得最佳性能，通过测试可以验证这个假设。不要使用 `VaryByParam="*"`，仅凭感觉就高速缓存。常见的原则是开始时高速缓存尽可能少的数据，以后再根据需要添加更多的高速缓存数据。服务器内存是有限的资源，因此在一些情况下，应配置使用磁盘高速缓存。一定要在有限的资源 and 安全性之间找到平衡点，不要把敏感数据放在磁盘上。

2. VaryByControl

`VaryByControl` 是从复杂的用户控件中获得较大性能改善的一种非常简单的方式，用户控件用于显示不经常改变的大量 HTML。例如，假定用户控件显示一个 `ComboBox`，该 `ComboBox` 显示所有国家的名称。这些名称都是从一个数据库中提取的，并且显示在组合框中，如下所示：

```
<%@ OutputCache Duration="2592000" VaryByControl="comboBoxOfCountries" %>
```

国家的名称肯定不会经常改变，因此 `Duration` 可以设置为月(以秒计)。高速缓存用户控件的显示结果，使用该控件的页面就会有较大的性能改善，同时页面本身仍然是动态的。

3. VaryByCustom

`VaryBy` 特性有很强大的功能，但有时还需要更大的灵活性。如果要从前面的导航示例中提取 `OutputCache` 指令，通过存储在 `cookie` 中的值进行高速缓存，就可以添加 `VaryByCustom`。将 `VaryByCustom` 的值传送给添加到 `Global.asax.cs` 中的 `GetVaryByCustomString` 方法。在每次请求页面时都会调用这个方法，它会返回一个值。

为返回的每个唯一值高速缓存页面的不同版本。例如，假定用户有一个 `cookie`，名为 `Language`，它有 3 个值：`en`、`es` 和 `fr`。无论用户的浏览器报告使用什么语言，都允许用户指定他们首选的语言。`Language` 还有第 4 个值，但它可能不存在。因此，下面例子中的 `OutputCache` 指令高速缓存了页面的许多版本：

```
cacheItems = (num of pageIds) * (num of subPageIds) * (4 possible Language values)
```

假定 `pageId` 有 10 个值，每个 `pageId` 有 5 个 `subPageId` 值，而 `Language` 有 4 个值。这个导航页面就要高速缓存 200 个不同的版本。这个数字远远超出了实际的高速缓存页面数，但应认识到高速缓存的强大功能。

下面的 `OutputCache` 指令包含 `pageId` 和 `subPageId` 作为 `VaryByParam` 的值，将 `VaryByCustom` 的值 `prefs` 传送给程序清单 22-1 中的 `GetVaryByCustomString` 回调函数：

```
<%@ OutputCache Duration="90" VaryByParam="pageId;subPageId" VaryByCustom="prefs"%>
```

ASP.NET 中的高速缓存是 CPU 和内存的协调结果：与存储页面的 200 个版本相比，创建这个页面是很困难的。如果该页面只有 5KB 的 HTML，与成千上万次数据库访问相比，高速缓存只需占用几兆字节的内存空间。大多数页面在循环中至少要访问一次数据库，因此对高速缓存的页面的

每个请求都会避免对数据库的访问。如果需要的数据库服务器和许可较少，高效地使用高速缓存可以大大节省成本。

程序清单 22-1(代码文件 Global.asax)中的代码返回存储在 Language cookie 中的值。GetVaryByCustomString 方法的 arg 参数包含 VaryByCustom 中指定的“prefs”字符串。

程序清单 22-1 HttpApplication 中的回调方法 GetVaryByCustomString

```
public override string GetVaryByCustomString(HttpContext context, string arg)
{
    if(arg.ToLower() == "prefs")
    {
        HttpCookie cookie = context.Request.Cookies["Language"];
        if(cookie != null)
        {
            return cookie.Value;
        }
    }
    return base.GetVaryByCustomString(context, arg);
}
```

程序清单 22-1 中的 GetVaryByCustomString 方法由 Global.asax.cs 中的 HttpApplication 使用，每个使用 VaryByCustom OutputCache 指令的页面都调用该方法。如果应用程序有许多使用 VaryByCustom 的页面，就可以创建一条 switch 语句和一系列辅助函数，从用户的 HttpContext 中提取需要的信息，并为高速缓存键生成唯一值。

4. 扩展<outputCache>

自从 ASP.NET 4 开始，现在可以扩展 OutputCache 指令的工作方式并使用该方式替代自己的定制方式来进行高速缓存。这意味着可以把 OutputCache 指令连接到高速缓存方式的任意类型，包括分布式高速缓存、云高速缓存、磁盘、XML 或其他我们可以想到的任何类型。

为了完成这项任务，需要创建作为类的定制输出高速缓存提供程序，并且需要从新的 System.Web.Caching.OutputCacheProvider 类派生该类。为此，必须重写 Add、Get、Remove 和 Set 方法，以实现输出高速缓存的定制版本。

定制实现后，下一步就是在 machine.config 或 web.config 配置文件中配置。在配置文件中需要对<outputCache>元素进行一些修改，以允许应用定制的高速缓存扩展。

<outputCache>元素位于配置文件的<caching>部分，其中包括新的<providers>子元素：

```
<caching>
  <outputCache>
    <providers>
    </providers>
  </outputCache>
</caching>
```

可以在新的<providers>元素中嵌套<add>元素，以对通过派生 OutputCacheProvider 类建立的新输出高速缓存功能进行适当引用：


```

<aching>
  <outputCache defaultProvider="AspNetInternalProvider">
    <providers>
      <add name="myDistributedCacheExtension"
        type="Wrox.OutputCacheExtension.DistributedCacheProvider,
          DistributedCacheProvider" />
    </providers>
  </outputCache>
</aching>

```

设置好这个新的<add>元素后,就可以使用新的扩展输出高速缓存。这里还要注意,<outputCache>元素中新增了 defaultProvider 特性。在本例中,该特性使用配置文件中的默认设置,即 AspNetInternalProvider。这意味着在默认情况下,输出高速缓存和往常一样工作,并把高速缓存存储到运行程序的计算机的内存中。

设置了自己的输出高速缓存提供程序后,可以通过页面的 OutputCache 指令指向这个提供程序,如下所示:

```

<%@ OutputCache Duration="90" VaryByParam="*"
  providerName="myDistributedCacheExtension" %>

```

如果提供程序的名称没有定义,就使用配置文件中 defaultProvider 特性定义的提供程序。

22.1.2 部分页面(UserControl)的高速缓存

与输出高速缓存类似,部分页面的高速缓存可以只高速缓存 Web 页面的特定块。例如,可以只高速缓存页面的中间部分。可使用用户控件的高速缓存实现部分页面的高速缓存,因此可以建立使用许多用户控件的 ASPNET 页面,再把输出高速缓存应用于选中的用户控件。这实际上只高速缓存了部分页面,页面的其他部分不进行高速缓存。这是一项非常优秀的功能,如果使用正确,就可以使页面更好地工作。这需要事先进行模块化设计,把页面的组件分到用于构建用户控件的逻辑单元中。

一般把用户控件放在多个页面上,以最大程度地重用常见功能。但是,在使用@OutputCache 指令的默认特性高速缓存用户控件(.ascx 文件)时,它们会在每个页面上高速缓存。也就是说,即使 pageA.aspx 上的用户控件与放在 pageB.aspx 上的用户控件输出的 HTML 相同,它们的输出也会高速缓存两次。使用 Shared="true",用户控件的输出就可以在多个页面和大量使用共享用户控件的站点上共享:

```

<%@ OutputCache Duration="300" VaryByParam="*" Shared="true" %>

```

节省的内存大小非常惊人,因为只高速缓存了用户控件的一份副本,而没有高速缓存每个页面的副本。与所有的优化操作一样,要测试输出的正确性和内存的使用情况。



如果 ASCX 用户控件使用 OutputCache 指令,那么该用户控件只能用于第一个请求。如果用户控件从 OutputCache 中提取 HTML,该控件在 ASPX 页面中就不存在,而是创建一个 PartialCachingControl,用作该控件的代理。

如果用户控件是从 `OutputCache` 中重新构建的，ASPX 页面中需要用户控件持续可用的代码就会失败。因此在使用控件之前，一定要检查这类高速缓存。下面的代码演示了访问高速缓存的用户控件时需要的逻辑：

```
protected void Page_Load()
{
    if (PossiblyCachedUserControl != null)
    {
        // Place code manipulating PossiblyCachedUserControl here.
    }
}
```

22.1.3 Post-Cache Substitution

输出高速缓存一般是一种要么全部都有、要么什么都没有的方式。整个页面的输出都高速缓存起来，供以后使用。但是我们常常需要利用输出高速缓存的优点，同时需要保留页面上的一些动态内容。高速缓存了页面，但不能输出动态的“Welcome, Scott!”，这是很让人难为情的结果。

自 2.0 版本以来，ASP.NET 添加了 `Post-Cache Substitution` 功能，以修改要显示的页面。把控件添加到页面上，用作占位符。在高速缓存的内容返回后，该占位符会调用指定的方法。该方法可返回任意字符串输出，但注意不应滥用这个功能。如果 `Post-Cache Substitution` 代码调用了一个昂贵的存储过程，就很容易丧失期望的任何性能改进。

`Post-Cache Substitution` 是很容易使用的一个功能，它使用两种方式控制替换：

- 调用 `Response.WriteSubstitution` 方法，并给它传送对期望的替换方法回调的引用。
- 在页面的指定位置添加一个 `<asp:Substitution>` 控件，把它的 `methodName` 属性设置为回调方法的名称。

为了试用这个功能，创建一个带 `Default.aspx` 的新 Web 站点，把一个标签控件和一个替换控件拖放到设计界面上。程序清单 22-2 中的代码更新该标签，以显示当前时间，但页面会立即高速缓存，以后的请求都会返回高速缓存的值。把替换控件的 `methodName` 属性设置为 `GetUpdatedTime`，表示从高速缓存中提取页面后调用的静态方法名。

回调函数必须是静态的，因为所显示的页面此时不存在(该页面的实例不存在)。由于没有可使用的页面实例，因此这个方法只能在其作用域内工作。但当前的 `HttpContext` 被传送给该方法，因此可以访问请求、响应和会话。这个方法返回的字符串被插入到响应中替换控件的位置。

程序清单 22-2 使用替换控件

```
<%@ Page Language="C#" %>
<%@ OutputCache Duration="30" VaryByParam="None" %>

<script runat="server">
    public static string GetUpdatedTime(HttpContext context)
    {
        return DateTime.Now.ToLongTimeString() + " by " +
            context.User.Identity.Name;
    }
}
```



```

    }
    protected void Page_Load(object sender, EventArgs e)
    {
        Label1.Text = DateTime.Now.ToLongTimeString();
    }
</script>

<!DOCTYPE html>

<html>
<head>
    <title>Substitution Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
            <br />
            <asp:Substitution ID="Substitution1" runat="server"
                MethodName="GetUpdatedTime" />
            <br />
        </div>
    </form>
</body>
</html>

```

程序清单 22-2 中的 ASPX 页面有一个标签和一个 Post-Cache Substitution 控件。该控件用作占位符，在从高速缓存中返回页面后，新内容就插入到该控件所在的位置。第一次访问页面时，只更新标签，因为还没有返回高速缓存的内容。但第二次访问页面时，从高速缓存中提取整个页面，不调用页面的处理程序，因此不触发任何页面级的事件。然而，在高速缓存模块完成其任务后会调用 `GetUpdatedTime` 方法。图 22-1 中的第一行是高速缓存的内容，第二行是动态创建的内容。



图 22-1



使用 `context.User.Identity.Name` 确定当前用户的名称。为此，需要把应用程序配置为使用 Windows 集成验证，并禁用匿名验证，如图 22-2 所示。



图 22-2

22.1.4 HttpCachePolicy 和客户端高速缓存

高速缓存并不只是在服务器端的内存中存储数据。优秀的高速缓存策略还应包含浏览器和客户端高速缓存，使用 `Cache-Control` HTTP 报头控制这些高速缓存。HTTP 报头是指示浏览器如何处理请求的提示和指令。

一些人推荐使用 HTML 标记 `<META>` 控制高速缓存的行为。注意浏览器和路由器都不会执行这些指令。使用 HTTP 报头控制高速缓存会比较成功。

HTTP 报头在 HTTP 消息主体的外部传送，因此查看它们的方式有很多种。可以激活跟踪功能(详见第 29 章)，在跟踪结果中查看报头。还可以使用其他软件，例如浏览器的 Web 开发插件(通常用 F12 功能键激活)，或者 `getfirebug.com` 中流行的 Firefox 扩展程序 Firebug。



有关 HTTP 报头和控制高速缓存的背景知识，可参阅文档 RFC 2616: Hypertext Transfer Protocol-HTTP/1.1，可以从 World Wide Web Consortium 站点 www.w3c.org 上获得该文档。还可以查看 Fiddler(www.fiddlertool.com/fiddler)。商用工具 HttpWatch(www.httpwatch.com)也添加了不少功能。

为了建立高速缓存，可以创建一个文件，把当前时间写入 Load 事件中。现在，查看 ASP.NET 使用的默认 HTTP 报头。注意，HTTP 报头 `Cache-Control: private` 告诉路由器和其他中介设备：这个响应只用于当前用户(私有)。

`HttpCachePolicy` 类为管理客户端状态提供了一个对象模型，禁止添加 HTTP 报头。在 `Page_Load` 中添加程序清单 22-3 所示的代码，会影响响应的报头和浏览器的高速缓存操作。这个程序清单告诉浏览器不在内存中高速缓存这个响应，也不把它存储到磁盘上。它还让响应立即过期。

程序清单 22-3 使用 HTTP 报头强制浏览器不在客户端高速缓存

```
protected void Page_Load(object sender, EventArgs e)
```

```

{
    Response.Cache.SetCacheability(HttpCacheability.NoCache);
    Response.Cache.SetNoStore();
    Response.Cache.SetExpires(DateTime.MinValue);

    Response.Write(DateTime.Now.ToLongTimeString());
}

```

比较程序清单 22-3 在运行前(图 22-3 顶部)和运行后(图 22-3 底部)的结果。将两个新的 HTTP 报头插入客户端的浏览器中,并将 Cache-Control 报头改为“no-cache, no-store”。输出高速缓存的 HttpModule 不修改这些 HTTP 报头,因此给浏览器发送“no-cache, no-store”也会建议 HttpModule 把响应记录为高速缓存缺失。图 22-3 显示了这种差别。

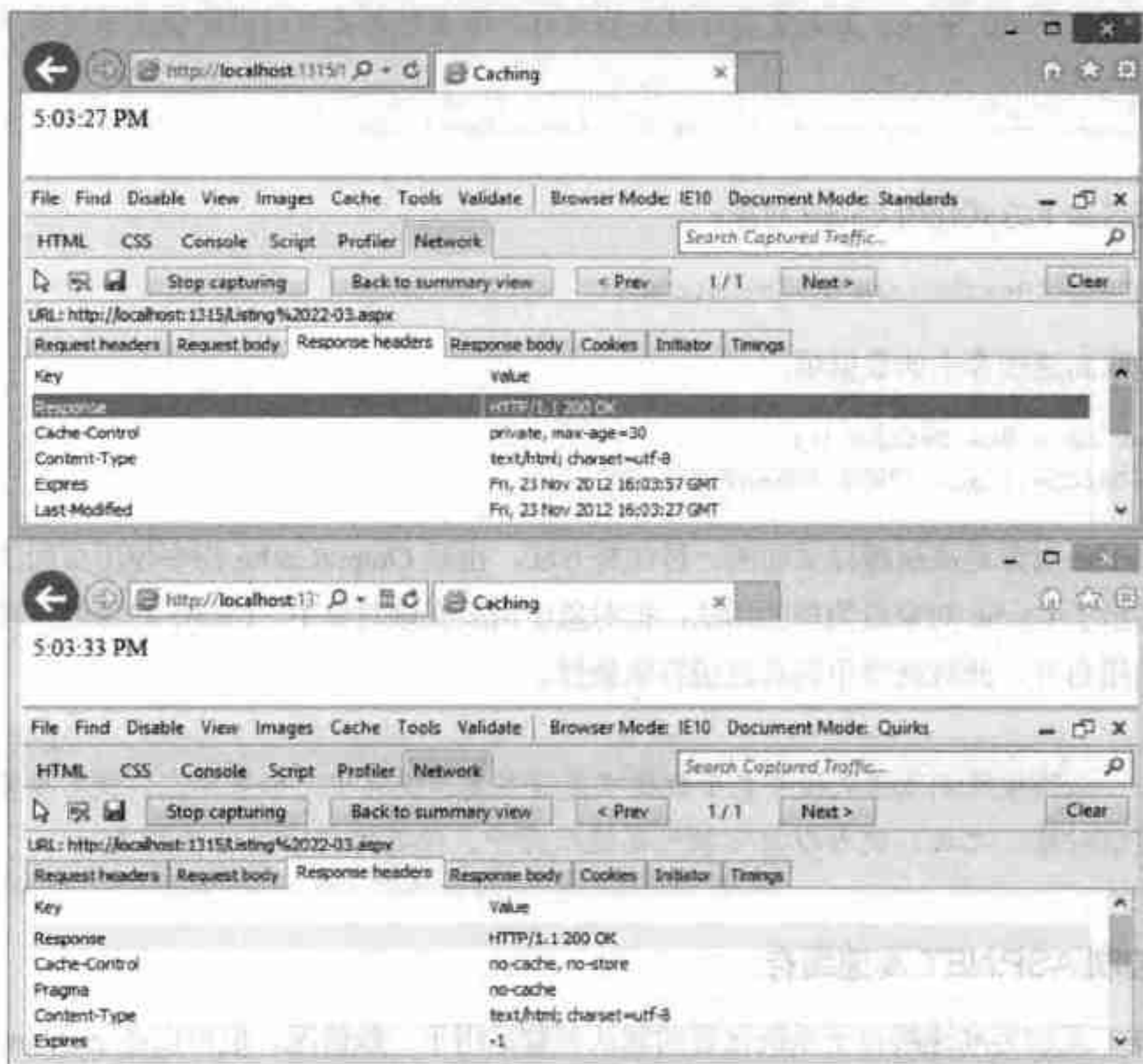


图 22-3

如果 ASP.NET 应用程序包含相当多的静态页面或对时间不敏感的页面,就应考虑使用客户端的高速缓存策略。最好利用磁盘空间和用户客户端计算机的内存,而不是占用服务器的有限资源。

22.2 使用编程方式进行高速缓存

输出高速缓存是声明性的工作。可以使用 OutputCache 指令标记用户控件和页面,极大地改变站点的操作。声明性高速缓存控制着 HTML 标记的生存期,但 ASP.NET 还包含对高速缓存对象的非常重要的编程支持。

22.2.1 使用 Cache 对象高速缓存数据

高速缓存的另一个方法是使用 Cache 对象高速缓存特定的数据项，供某个页面或某组页面以后使用。Cache 对象可以存储许多内容，其范围从简单的名/值对到数据集和整个.aspx 页面这样的复杂对象。



尽管 Cache 对象与会话状态非常类似，但该对象由某个 Web 服务器中存储该应用程序的应用程序域的所有用户共享。因此，如果把某个数据项放在高速缓存中，所有的用户就都能访问该对象。这与服务器场(Server farm)的情况不同，因为不能保证用户下一次访问哪台服务器，即使只涉及一台服务器，也可能有多个运行这个应用程序的应用程序域。另外，如果需要释放某些内存，那么服务器可以随时使高速缓存的数据项失效。

可以通过如下方式使用 Cache 对象：

```
Cache["WhatINeedToStore"] = myDataSet;
```

可以提取高速缓存中的数据项：

```
DataSet ds = new DataSet();  
ds = (DataSet)Cache["WhatINeedToStore"];
```

使用 Cache 对象是高速缓存页面的一种优秀方法，也是 OutputCache 指令使用页面的方式。上面的代码显示了 Cache 对象最简单的用法：把对象引用放在该对象中。Cache 对象的一项重要功能是可以禁用自身，此时就要用到高速缓存依赖性。



必须按照测试模式检查某个数据项是否在高速缓存中，如果不在，就需要重新创建该对象。之后，就可以把它插入高速缓存中，使其可用于下一个请求。

22.2.2 控制 ASP.NET 高速缓存

ASP.NET 最初为高速缓存子系统设置的默认参数适用于一般情况，但可以在 machine.config 或 web.config 文件中配置这些参数。可以修改这些选项，例如在系统内存压力较大时，可以禁止高速缓存的数据项到期，或者完全关闭数据项到期功能。可以设置在高速缓存刷新其数据项之前应用程序的最大私有字节数：

```
<system.web>  
  <cache disableMemoryCollection="false"  
    disableExpiration="false" privateBytesLimit="0"  
    percentagePhysicalMemoryUsedLimit="90"  
    privateBytesPollTime="00:02:00" />  
  ...snip...
```




最好不要修改默认值，除非完成了应用程序的正式配置，并理解了它是如何利用高速缓存的。有关本节的更多内容，可查阅 MSDN: [http://msdn.microsoft.com/en-us/library/vstudio/ms228248\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/vstudio/ms228248(v=vs.100).aspx)。

22.2.3 高速缓存依赖性

使用 Cache 对象，可以根据几种不同的依赖性来存储和禁用高速缓存中的数据项。在 ASP.NET 1.0/1.1 中，只有如下依赖性：

- 基于文件的依赖性
- 基于键的依赖性
- 基于时间的依赖性

在使用 Cache 对象给高速缓存插入数据项时，使用 Insert 方法设置依赖性，如下所示：

```
Cache.Insert("DSN", connectionString,
    new CacheDependency(Server.MapPath("~/myconfig.xml")))
```

在引用的数据项发生变化时使用依赖性，就从内存中删除了该数据项的高速缓存。

ASP.NET 自从 2.0 版本以来对高速缓存依赖性进行了改进，添加了 AggregateCacheDependency 类、可扩展的 CacheDependency 类，还可以创建定制的 CacheDependency 类。

1. AggregateCacheDependency 类

AggregateCacheDependency 类类似于 CacheDependency 类，但它可以使用不同类型的依赖性，建立与高速缓存中数据项的关联。例如，如果 XML 文件中有一个高速缓存的数据项，并且从 SQL Server 数据库表中获得了信息，就可以创建 AggregateCacheDependency，在其中为每个子依赖性插入一个 CacheDependency 对象。为此，要调用 Cache.Insert 方法，添加 AggregateCacheDependency 实例，如下所示：

```
var agg = new AggregateCacheDependency();
agg.Insert(new CacheDependency(Server.MapPath("~/myconfig.xml")));
agg.Insert(new SqlCacheDependency("Northwind", "Customers"));
Cache.Insert("DSN", connectionString, agg);
```

注意，AggregateCacheDependency 要和不同类型的 CacheDependency 类一起使用。如果只想将一个高速缓存的数据项与多个文件建立关联，那么可以使用 CacheDependency 的重载版本，如下所示：

```
Cache.Insert("DSN", yourObject,
    new System.Web.Caching.CacheDependency(
        new string[]
        {
            Server.MapPath("~/file1.xml"),
            Server.MapPath("~/file2.xml")
        }
    )
);
```

AggregateCacheDependency 类支持扩展以前的密封类 CacheDependency，可以使用这个新功能创建定制的 CacheDependency。

2. 非密封的 CacheDependency 类

ASP.NET 自从 2.0 版本以来，高速缓存的一项重大变化是重新修订了 CacheDependency 类，它现在是不密封的(即可以重写)。现在可以创建继承于 CacheDependency 的类，建立更精细的依赖性，而不同于过去的 Time、Key 或 File 依赖性。

在创建自己的高速缓存依赖性时，可以为 Web 服务数据、仅用于午夜的依赖性和文件中的文本字符串改动等添加过程。我们创建的依赖性没有任何限制。CacheDependency 类的非密封性允许指定何时需要禁用高速缓存中的数据项。

有了 CacheDependency 类的非密封性，ASP.NET 小组建立了新的 SQL Server 高速缓存依赖性 SqlCacheDependency。在高速缓存因底层 SQL Server 的表发生变化而无效时，我们会立即在 ASP.NET 应用程序中了解这一点。

CacheDependency 类现在不是密封的，因此可以派生出定制的高速缓存依赖性。

3. 创建定制的高速缓存依赖性

ASP.NET 支持基于时间、基于文件和基于 SQL 的 CacheDependency。那么，为什么还要编写定制的 CacheDependency？原因如下：

- 在 Active Directory 查询的结果中使高速缓存失效
- 在 MSMQ 或 MQSeries 消息到来时使高速缓存失效
- 创建专用于 Oracle 的 CacheDependency
- 使用 XML Web 服务报告的数据使高速缓存失效
- 在 Stock Price 服务中使用新数据更新高速缓存

CacheDependency 类增加了 3 个成员和 1 个重载构造函数，供开发人员使用：

- GetUniqueID：当被重写时，可以给调用者返回定制的 CacheDependency 的唯一标识符。
- DependencyDispose：用于释放定制的 CacheDependency 类使用的资源。在创建定制的 CacheDependency 时，需要实现这个方法。
- NotifyDependencyChanged：根据定制的高速缓存依赖性实例，使高速缓存过期。
- 公有构造函数 CacheDependency。

程序清单 22-4(本章下载代码中的代码文件 RssCacheDependency.cs)创建了新类 RssCacheDependency，如果 RSS(Rich Site Summary，丰富站点摘要)XML 文档发生变化，就使高速缓存键失效。

程序清单 22-4 创建 RssCacheDependency 类

```
using System;
using System.Web;
using System.Threading;
using System.Web.Caching;
using System.Xml;

public class RssCacheDependency : CacheDependency
{

```

```

    Timer backgroundThread;
    int howOften = 900;
    XmlDocument RSS;
    string RSSUrl;

    public RssCacheDependency(string URL, int polling)
    {
        howOften = polling;
        RSSUrl = URL;
        RSS = RetrieveRSS(RSSUrl);

        if(backgroundThread == null)
        {
            backgroundThread = new Timer(
                new TimerCallback(CheckDependencyCallback),
                this, (howOften * 1000), (howOften * 1000));
        }
    }

    public XmlDocument RetrieveRSS(string URL)
    {
        XmlDocument retVal = new XmlDocument();
        retVal.Load(URL);
        return retVal;
    }

    public void CheckDependencyCallback(object sender)
    {
        RssCacheDependency CacheDepends = sender as RssCacheDependency;
        XmlDocument NewRSS = RetrieveRSS(RSSUrl);
        if(NewRSS.OuterXml != RSS.OuterXml)
        {
            CacheDepends.NotifyDependencyChanged(CacheDepends, EventArgs.Empty);
        }
    }

    override protected void DependencyDispose()
    {
        backgroundThread = null;
        base.DependencyDispose();
    }

    public XmlDocument Document
    {
        get
        {
            return RSS;
        }
    }
}

```

为了使用新代码，打开一个新的 Web 站点，把 RssCacheDependency 类放在 App_Code 文件夹中。创建一个新的页面文件，在页面中拖放两个文本框、一个标签和一个按钮。运行这个 Web 站点，输入某博客的 RSS URL(如 MSDN 种子 <http://sxp.microsoft.com/feeds/3.0/msdnnews/msdnnews>)，单击

其中的按钮。程序就会把 URL 用作键，检查 Cache 对象。如果包含 RSS 的 XmlDocument 在高速缓存中不存在，就创建新的 RssCacheDependency，将其超时时间设置为 10 分钟(600 秒)。然后高速缓存 XmlDocument，以后的 10 分钟内对这个页面的请求都会从高速缓存中提取 RSS XmlDocument。

程序清单 22-4 中新的 RssCacheDependency 类如程序清单 22-5 所示。首先创建 RssCacheDependency，再把它传送给 Cache.Insert 调用。Cache 对象处理生存期，并调用 RssCacheDependency 实例的方法。

程序清单 22-5 使用 RssCacheDependency 类

```
<script runat="server">
    void Button1_Click(object sender, System.EventArgs e)
    {
        string RSSUrl = TextBox1.Text;
        Label2.Text = "Loaded From Cache";
        if(Cache[TextBox1.Text] == null)
        {
            Label2.Text = "Loaded Fresh";
            RssCacheDependency itDepends = new RssCacheDependency(RSSUrl, 30);
            Cache.Insert(RSSUrl, itDepends.Document, itDepends);
        }
        TextBox2.Text = ((System.Xml.XmlDocument)Cache[TextBox1.Text]).OuterXml;
    }
</script>
<!DOCTYPE html>

<html>
<head id="Head1" runat="server">
    <title>Custom Cache Dependency Example</title>
</head>
<body>
    <form id="Form1" runat="server"> RSS URL:
        <asp:TextBox ID="TextBox1" runat="server"/>
        <asp:Button ID="Button1" onclick="Button1_Click" runat="server"
            Text="Get RSS" />
        Cached:<asp:Label ID="Label2" runat="server"></asp:Label><br />
        RSS:<br />
        <asp:TextBox ID="TextBox2" runat="server" TextMode="MultiLine"
            Width="800px" Height="300px"></asp:TextBox>
    </form>
</body>
</html>
```

RssCacheDependency 类创建一个 Timer 后台线程来检查 RSS 摘要中的改动。如果 RssCacheDependency 检测到改动，就使用 NotifyDependencyChanged 事件通知高速缓存子系统。清除高速缓存中带有该键的值，下一个页面视图必须从指定的摘要中重新加载请求的 RSS。

22.2.4 .NET 4.x 中新的对象高速缓存选项

到目前为止，从对 System.Web.Caching.Cache 对象的了解中可以看出该对象功能强大，它甚至允许创建定制的高速缓存。但是，Cache 对象的扩展性和作用已经改变。

System.Runtime.Caching.dll 推动了此改动，这是因为它对 System.Web 版本的内容进行了重组，

并将其重新创建到了新的名称空间 `System.Runtime.Caching` 中。

这个改动并不完全是为了 ASP.NET 开发人员，还为了其他的应用程序类型，例如 Windows 窗体应用程序、WPF 应用程序等。`System.Web.Caching.Cache` 对象非常有用，以至于其他的应用程序开发人员为了使用该对象而把 `System.Web` 名称空间引入到他们的项目中。为了避免 Windows 窗体开发人员只是为了使用提供的 `Cache` 对象就将 `System.Web.dll` 引入到项目中，需要从 `System.Runtime.Caching` 名称空间中提取并扩展 `Cache` 对象。

ASP.NET 开发人员仍然可以像在 ASP.NET 之前版本中那样使用 `System.Web.Caching.Cache` 对象。但要注意的是，随着 .NET Framework 的发展，.NET 小组将把重心放在 `System.Runtime.Caching` 名称空间上，而不是放在 `System.Web.Caching` 名称空间上，记住这一点很重要。这意味着，随着时间的推移，在 `System.Runtime.Caching` 版本中增加的改进部分很可能不会在 `System.Web.Caching` 名称空间中出现。但是，这并不代表需要将内容全部移到新的 `System.Runtime.Caching` 名称空间以确保遵循微软的策略路线，因为 .NET 内部会同时管理这两个高速缓存。

本节通过使用 `System.Runtime.Caching` 名称空间的高速缓存的例子进行介绍。在这个例子中，ASP.NET 页面仅使用一个 `Label` 控件来显示存储在 XML 文件中的用户姓名。第一步是在 `App_Data` 文件夹中创建一个 XML 文件，命名为 `Username.xml`。这个简单的文件如程序清单 22-6 所示(代码文件 `Username.xml`)。

程序清单 22-6 Username.xml 文件的内容

```
<?xml version="1.0" encoding="utf-8" ?>
<usernames>
  <user>Christian Wenz</user>
</usernames>
```

把这个 XML 文件放在驱动器的根目录下，利用 `Default.aspx` 页面的代码来使用该文件中的用户姓名，并把该姓名显示到页面的一个 `Label` 控件上。`Default.aspx` 页面的代码如程序清单 22-7 所示。

程序清单 22-7 使用 `System.Runtime.Caching` 名称空间

```
<%@ Page Language="C#" %>
using System.Runtime.Caching;

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        ObjectCache cache = MemoryCache.Default;

        string usernameFromXml = cache["userFromXml"] as string;

        if(usernameFromXml == null)
        {
            List<string> userFilePath = new List<string>();
            userFilePath.Add(Server.MapPath("~/App_Data/Username.xml"));

            CacheItemPolicy policy = new CacheItemPolicy();
            policy.ChangeMonitors.Add(new HostFileChangeMonitor(userFilePath));

            XDocument xdoc = XDocument.Load(Server.MapPath(
```

```

        "~/App_Data/Username.xml"));
var query = from u in xdoc.Elements("usernames")
            select u.Value;

usernameFromXml = query.First().ToString();

cache.Set("userFromXml", usernameFromXml, policy);
}

Label1.Text = usernameFromXml;
}
</script>
<!DOCTYPE html>

<html>
<head runat="server">
    <title>Using System.Runtime.Caching</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="Label1" runat="server"></asp:Label>
        </div>
    </form>
</body>
</html>

```

程序清单 22-7 中的例子使用了 `System.Runtime.Caching` 中新的高速缓存。需要在 ASP.NET 项目中引用该名称空间，如图 22-4 所示。

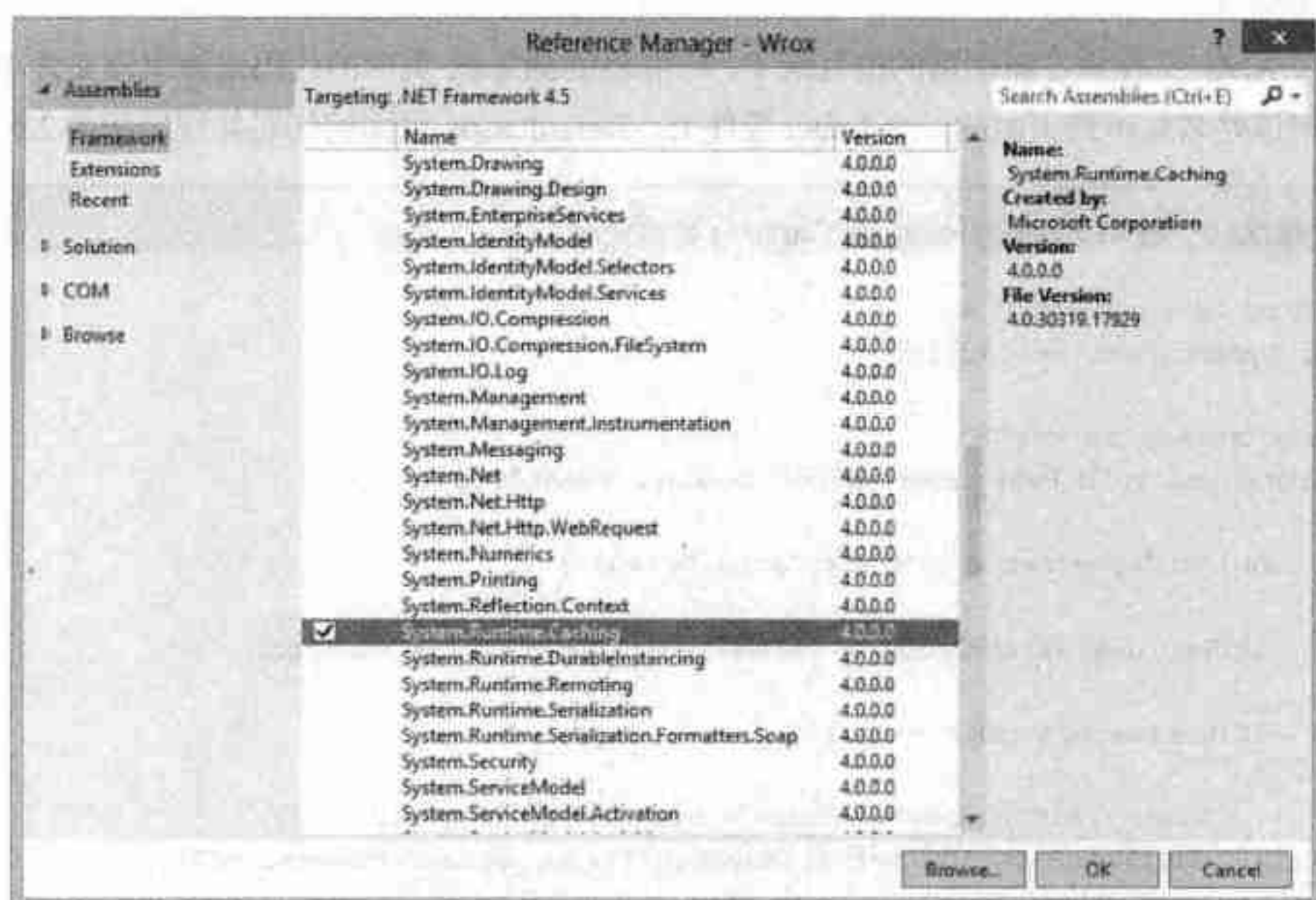


图 22-4

首先，创建高速缓存对象的默认实例：


```
ObjectCache cache = MemoryCache.Default;
```

然后，像使用传统的 ASP.NET 高速缓存对象那样来使用这个高速缓存：

```
string usernameFromXml = cache["userFromXml"] as string;
```

为了启用高速缓存，需要创建一个对象，该对象定义了处理哪种类型的高速缓存。可以创建自定义的实现，也可以使用 .NET 4.x 提供的默认实现：

```
CacheItemPolicy policy = new CacheItemPolicy();
policy.ChangeMonitors.Add(new HostFileChangeMonitor(userFilePath));
```

HostFileChangeMonitor 是用于查看目录、文件路径和监控程序改动情况的一种方法。例如，当 XML 文件发生改变时，就会触发高速缓存的禁用功能。ChangeMonitor 对象的其他实现包括 FileChangeMonitor 和 SqlChangeMonitor。

在这个例子中，注意第一次运行时，文本 Christian Wenz 被加载到高速缓存中，并且出现在 Label1 控件中。继续运行应用程序，然后返回 XML 文件并改变其值，你就会注意到在页面刷新时禁用了高速缓存。

22.3 使用 SQL Server 高速缓存依赖性

要利用 ASP.NET 中的 SQL Server 高速缓存依赖性，必须使 SQL Server 数据库支持该功能。为此，可使用 C:\Windows\Microsoft .NETFramework\v4.0.xxxxx\下的 aspnet_regsql.exe 工具。该工具会对 SQL Server 进行必要的修改，以便开始使用 SQL 高速缓存禁用功能。

使用 SQL Server 高速缓存依赖性应遵循下面的步骤：

- (1) 使数据库支持 SQL Server 高速缓存依赖性。
- (2) 使表支持 SQL Server 高速缓存依赖性。
- (3) 在 ASP.NET 应用程序的 web.config 文件中包含 SQL 连接字符串。
- (4) 以如下方式之一利用 SQL Server 高速缓存依赖性：
 - 在代码中以编程的方式创建 SqlCacheDependency 对象
 - 给 OutputCache 指令添加 SqlDependency 特性
 - 通过 Response.AddCacheDependency 给 Response 对象添加 SqlCacheDependency 实例

本节将介绍所有需要的步骤和可用的操作。

首先，需要获得 aspnet_regsql.exe 工具。打开 Visual Studio Command Prompt，输入下面的命令：

```
aspnet_regsql.exe -?
```

这行代码会输出这个命令行工具的帮助命令列表，如表 22-1 所示。

表 22-1

SQL Server 高速缓存依赖性选项	说 明
-d <database>	用于 SQL Server 高速缓存依赖性的数据库名。可以使用连接字符串和 -c 选项指定数据库(必选)
-ed	允许数据库启用 SQL Server 高速缓存依赖性

(续表)

SQL Server 高速缓存依赖性选项	说 明
-dd	禁止数据库启用 SQL Server 高速缓存依赖性
-et	允许表启用 SQL Server 高速缓存依赖性。需要-t 选项
-dt	禁止表启用 SQL Server 高速缓存依赖性。需要-t 选项
-t <table>	支持或禁止 SQL Server 高速缓存依赖性的表名。需要-et 或-dt 选项
-lt	列出所有启用了 SQL Server 高速缓存依赖性的表

下面说明如何使用其中一些命令。



本章示例使用 Northwind 数据库，它不在本章的下载代码中，必须单独检索和安装。在 www.wrox.com/go/SQLServer2012DataSets 上可以下载 Northwind 数据库。第 24 章包含检索这个示例数据库的更多信息。当然，该章介绍的技术也可用于其他数据库。

22.3.1 使数据库支持 SQL Server 高速缓存禁用功能

要在 SQL Server 数据库中使用 SQL Server 高速缓存禁用功能，有两个步骤。第一步是使相应的数据库支持该功能，第二步是使要使用的表支持该功能。必须执行这两步，才能完成此任务。如果要使数据库支持 SQL Server 高速缓存禁用功能，并且在 SQL Server 实例所在的计算机上工作，就可以使用下面的构造代码。如果 SQL Server 实例位于另一台计算机上，就应把这个例子中的 localhost 改为远程计算机的名称。

```
aspnet_regsql.exe -S localhost -U username -P password -d Northwind -ed
```

在本地计算机上使用 SQLEXPRESS 示例的语法如下(这里使用了 Windows 集成验证):

```
aspnet_regsql.exe -S .\SQLEXPRESS -E -d Northwind -ed
```

生成的结果如下所示:

```
Enabling the database for SQL cache dependency.
.....
Finished.
```

在这条命令中，仅使 Northwind 数据库(SQL Server 附带的示例数据库)支持 SQL Server 高速缓存禁用功能。使用-S 指定计算机的名称，使用-U 指定用户名，使用-d 指定数据库，最重要的是，激活 SQL Server 高速缓存禁用功能的命令是-ed。

使数据库支持 SQL Server 高速缓存禁用功能后，就可以使 Northwind 数据库中的一个或多个表支持该功能。

22.3.2 使表支持 SQL Server 高速缓存禁用功能

使用下面的命令可以使一个或多个表支持 SQL Server 高速缓存禁用功能:

```
aspnet_regsql.exe -S localhost -U username -P password -d Northwind -t Customers -et
aspnet_regsql.exe -S localhost -U username -P password -d Northwind -t Products -et
```

这条命令与使数据库支持该功能的命令没有什么区别,只是包含额外的-t Customers 条目,以及使用-et 使表支持该功能,而不是使用-ed 使数据库支持该功能。这里,Customers 是要支持该功能的表名。

下面使 Customers 和 Products 表支持该功能。对每个表运行一次该命令。使表支持该功能后,会得到如下响应:

```
Enabling the table for SQL cache dependency.
```

```
.....
```

```
Finished.
```

使表支持 SQL Server 高速缓存禁用功能后,就可以使用该功能。但在开始使用之前,首先介绍支持这些功能对 SQL Server 造成的影响。

22.3.3 对 SQL Server 的影响

Northwind 数据库、Customers 和 Products 表都已支持 SQL Server 高速缓存禁用功能,下面查看 SQL Server 中有什么变化。如果打开 SQL Server Management Studio(或其 Express 版本),就会在 Northwind 数据库中看到新表 AspNet_SqlCacheTablesForChangeNotification(相当长的表名),屏幕如图 22-5 所示。

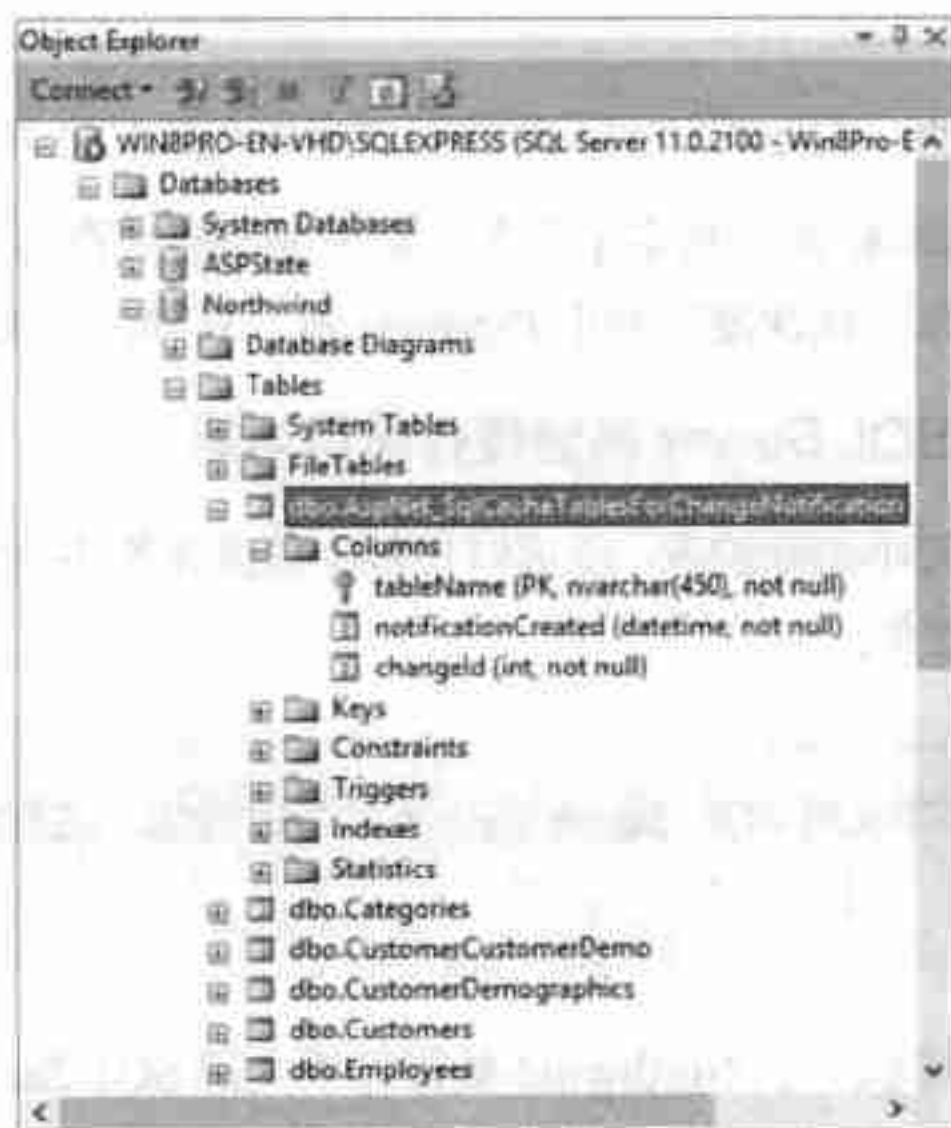


图 22-5

在右边的窗格中,表的列表的顶部是 AspNet_SqlCacheTablesForChangeNotification 表。ASP.NET 使用这个表监控哪些表有变化,并对这些变化发出通知。查看其内容,就会发现这个表其实非常简单,如图 22-5 所示。

- **tableName:** 这一列显示对同一数据库中包含的表名的字符串引用。这里列出的表都支持 SQL Server 高速缓存禁用功能。

- notificationCreated: 这一列显示表支持 SQL Server 高速缓存禁用功能的时间和日期。
- changeId: 这一列用于告诉 ASP.NET 表中的变化。ASP.NET 监控这一列的变化, 并根据该列的值把这些变化存储在内存中, 或者进行新的数据库查询。

22.3.4 查看支持 SQL Server 高速缓存禁用功能的表

使用 aspnet_regsql.exe 工具, 只需使用一条简单的命令就可以看出数据库中的哪些表支持 SQL Server 高速缓存禁用功能。如果使用前面的例子, 那么 Northwind 数据库的 Customers 和 Products 表都支持 SQL Server 高速缓存禁用功能。要获得支持该功能的表的列表, 可以使用下面的命令:

```
aspnet_regsql.exe -S localhost -U username -P password -d Northwind -lt
```

-lt 命令生成支持 SQL Server 高速缓存禁用功能的表的列表。输入这条命令就会生成下面的结果:

```
Listing all tables enabled for SQL cache dependency:
Customers
Products
```

22.3.5 使表不支持 SQL Server 高速缓存禁用功能

掌握如何使 SQL Server 数据库支持 SQL Server 高速缓存禁用功能后, 下面查看如何从监控该过程的指定表中删除这项功能。要从 SQL Server 高速缓存禁用过程中删除表, 可使用 -dt 命令。

在上面的例子中使用 -lt 命令, 显示 Customers 和 Products 表支持该功能。现在使用下面的命令从该过程中删除 Products 表:

```
aspnet_regsql.exe -S localhost -U username -P password -d Northwind -t Products -dt
```

这里只是使用 -t 命令后跟 -dt 命令指定了表名。禁止表高速缓存的命令会再次列出支持 SQL Server 高速缓存禁用功能的表。这次没有列出 Products 表, 只列出了支持该功能的 Customers 表。

22.3.6 使数据库不支持 SQL Server 高速缓存禁用功能

不仅可以选择要从该过程中删除的表, 还可以使整个数据库都不支持 SQL Server 高速缓存禁用功能。为此, 可以使用 -dd 命令。



使整个数据库都不支持 SQL Server 高速缓存禁用功能, 这意味着数据库中的所有表都不会支持该功能。

下面的例子显示了使计算机上的 Northwind 数据库不支持 SQL Server 高速缓存禁用功能:

```
C:\>aspnet_regsql -S localhost -U username -P password -d Northwind -dd
```

```
Disabling the database for SQL cache dependency.
```

```
.....
Finished.
```

为了确保表不再支持 SQL Server 高速缓存禁用功能, 如果使用 -lt 命令试图列出支持 SQL Server 高速缓存禁用功能的表, 就会得到如下错误:

```
C:\>aspnet_regsql -S localhost -U username -P password -d Northwind -lt
An error has happened. Details of the exception:
The database 'Northwind' is not enabled for SQL cache notification, please
use the System.Web.Caching.SqlCacheDependencyAdmin.EnableNotifications method,
or the command line tool aspnet_regsql. To use the tool, please run
'aspnet_regsql.exe -?' for more information.
```

在 SQL Server Management Studio 中打开 Northwind 数据库, 可以发现已经从数据库中删除 AspNet_SqlCacheTablesForChangeNotification 表。

22.3.7 SQL Server 高速缓存禁用功能

SQL Server 支持另一更精细的通知功能, 不需要轮询。直接通知改动是 SQL Server 的内置功能, 通过 ADO.NET SqlCommand 展现, 如程序清单 22-8 所示:

程序清单 22-8 使用 SQL Server 高速缓存禁用功能

```
<%@ Page Language="C#" %>
using System.Data.SqlClient

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        Response.Write("Page created: " + DateTime.Now.ToLongTimeString());
        string connStr = ConfigurationManager.ConnectionStrings[
            "AppConnectionString1"].ConnectionString;
        SqlDependency.Start(connStr);
        SqlConnection connection = new SqlConnection(connStr);
        SqlCommand command = new SqlCommand("Select * FROM Customers", connection);
        SqlCacheDependency depends = new SqlCacheDependency(command);

        connection.Open();
        GridView1.DataSource = command.ExecuteReader();
        GridView1.DataBind();

        connection.Close();

        Response.AddCacheDependency(depends);
    }
</script>

<!DOCTYPE html>

<html>
<head runat="server">
    <title></title>
</head>
```

```
<body>
  <form id="form1" runat="server">
    <asp:GridView ID="GridView1" runat="server" Height="400px" Width="400px">
    </asp:GridView>
  </form>
</body>
</html>
```

SQL Server 支持高速缓存的编程和声明技术。在 `OutputCache` 指令中使用 `"CommandNotification"` 字符串就可以给页面建立基于通知的高速缓存。可以编程指定 SQL Server 高速缓存选项，或者声明该选项，但不能同时使用这两种方式。注意，必须首先调用 `System.Data.SqlClient.SqlDependency.Start`，传送连接字符串以启动 SQL Server 通知引擎：

```
<%@ OutputCache Duration="3600" VaryByParam="none"
  SqlDependency="CommandNotification"%>
```

如果在 ASP.NET 页面上使用 `SqlDataSource` 控件：

```
<asp:SqlDataSource EnableCaching="true" SqlCacheDependency="CommandNotification"
  CacheDuration="2600" />
```

那么在 SQL Server 中的数据发生变化时，SQL Server 和 ADO.NET 会自动禁用在 Web 服务器上高速缓存的数据。

22.4 配置 ASP.NET 应用程序

使数据库和其中的几个表支持 SQL Server 高速缓存禁用功能后，就要为 SQL Server 高速缓存禁用功能配置应用程序。

要配置应用程序以使用 SQL Server 高速缓存禁用功能，首先应修改 `web.config` 文件。在 `web.config` 文件中，指定要使用 Northwind 数据库，并让 ASP.NET 连接它。

程序清单 22-9(代码文件 `web.config`)说明了如何修改 `web.config` 文件，以使用 SQL Server 高速缓存禁用功能。如果使用 SQL Server 2005 或更高版本的通知功能，就不需要 `pollTime` 特性，因为可以使用数据库事件代替以前版本中的轮询模型。

程序清单 22-9 配置 `web.config` 文件

```
<configuration>

  <connectionStrings>
    <add name="AppConnectionString1" connectionString="Data Source=localhost;
      User ID=username;Password=password;Database=Northwind;
      Persist Security Info=False"
      providerName="System.Data.SqlClient" />
  </connectionStrings>

  <system.web>
```



```

    <キャッシング>
      <sqlCacheDependency enabled="true">
        <databases>
          <add name="Northwind" connectionStringName="AppConnectionString1"
            pollTime="500" />
        </databases>
      </sqlCacheDependency>
    </キャッシング>

  </system.web>
</configuration>

```

在这个程序清单中,首先使用 web.config 文件中的 <connectionStrings> 元素建立了指向 Northwind 数据库的连接字符串。注意连接字符串的名称,因为以后要在 SQL Server 高速缓存禁用功能的配置设置中使用。

使用 <キャッシング> 元素配置 SQL Server 高速缓存禁用功能,这个元素必须嵌套在 <system.web> 元素中。我们使用的是 SQL Server 高速缓存依赖性,因此必须使用 <sqlCacheDependency> 子节点。使用 enabled="true" 可以激活整个过程。启用了该特性后,就可以使用 <databases> 部分。使用嵌套在 <databases> 节点中的 <add> 元素引用 Northwind 数据库。表 22-1 描述了 <add> 元素的所有特性。

表 22-2

特 性	说 明
name	为 SQL Server 数据库提供标识符
connectionStringName	指定连接的名称。前面例子中的连接字符串是 AppConnectionString1, 因此 connectionStringName 特性也使用这个值
pollTime	指定 SQL Server 轮询的时间间隔。默认为 5 秒或 500 毫秒(如程序清单 22-9 所示)。SQL Server 2005、2008 和 2012 通知功能不需要这个特性

web.config 文件配置正确后,就可以在页面上使用 SQL Server 高速缓存禁用功能。ASP.NET 会把 SQL Server 请求放在与 AspNet_SqlCacheTablesForChangeNotification 表完全不同的线程上,以查看 changeld 号是否递增。如果该数字有变化,ASP.NET 就知道对底层的 SQL Server 表进行了修改,应检索新的结果集。当确定是否应进行 SQL Server 调用时,对 AspNet_SqlCacheTablesForChangeNotification 表的请求会得到结果。启用了 SQL Server 高速缓存禁用功能后,这些操作执行得非常快,我们几乎察觉不到区别。

22.5 测试 SQL Server 高速缓存禁用功能

配置好 web.config 文件后,下一步是把这些新功能应用到页面。程序清单 22-10 是使用 SQL Server 高速缓存禁用功能的页面示例。

程序清单 22-10 使用 SQL Server 高速缓存禁用功能的 ASP.NET 页面

```

<%@ Page Language="C#" %>
<%@ OutputCache Duration="30" VaryByParam="none"

```

```

    SqlDependency="Northwind:Customers"%>

<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e)
    {
        Label1.Text = "Page created at " + DateTime.Now.ToShortTimeString();
    }
</script>

<!DOCTYPE html>

<html>
<head id="Head1" runat="server">
    <title>Sql Cache Invalidation</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Label ID="Label1" runat="server"></asp:Label><br />
        <br />
        <asp:GridView ID="GridView1" runat="server" DataSourceID="SqlDataSource1">
        </asp:GridView>
        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
            SelectCommand="Select * From Customers"
            ConnectionString="<%%$ ConnectionStrings:AppConnectionString1 %>"
            ProviderName="<%%$ ConnectionStrings:AppConnectionString1.providername %>"
        </asp:SqlDataSource>
    </form>
</body>
</html>

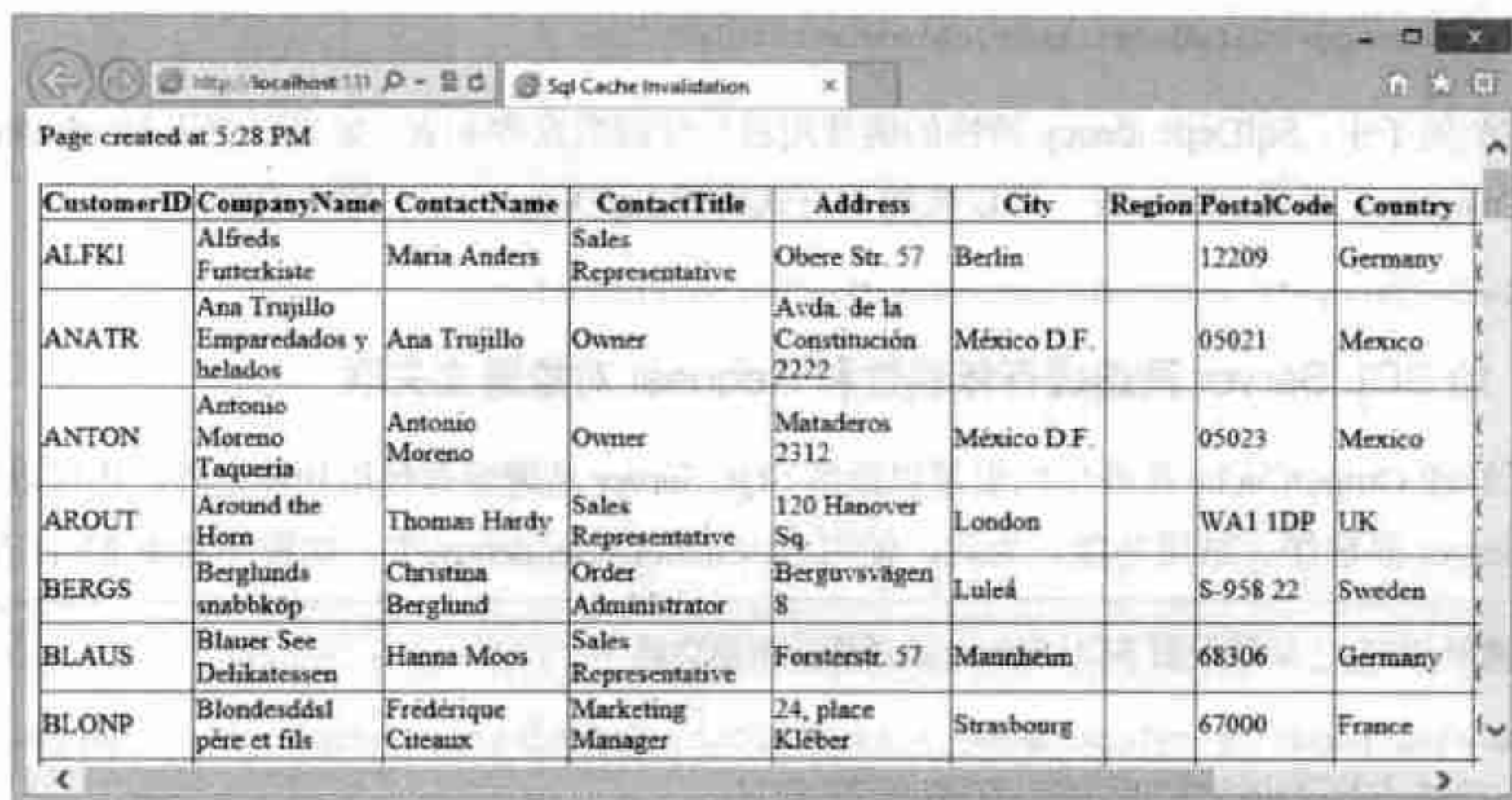
```

这个页面最重要的部分是在文件顶部指定的 `OutputCache` 页面指令。`OutputCache` 指令使用 `Duration` 特性指定页面输出在高速缓存中保存多长时间。接下来是 `VaryByParam` 特性。新增加的特性是 `SqlDependency`，它可以让页面使用 SQL Server 高速缓存禁用功能。下面的代码显示了 `SqlDependency` 特性值的格式：

```
SqlDependency="database:table"
```

`Northwind:Customers` 值指定让 Northwind 数据库的 `Customers` 表支持 SQL Server 高速缓存禁用功能。`OutputCache` 指令的 `Duration` 特性表示，这个页面的输出要在高速缓存中存储很长时间，但如果对 `Customers` 表中的数据进行了修改，就禁用这个高速缓存。

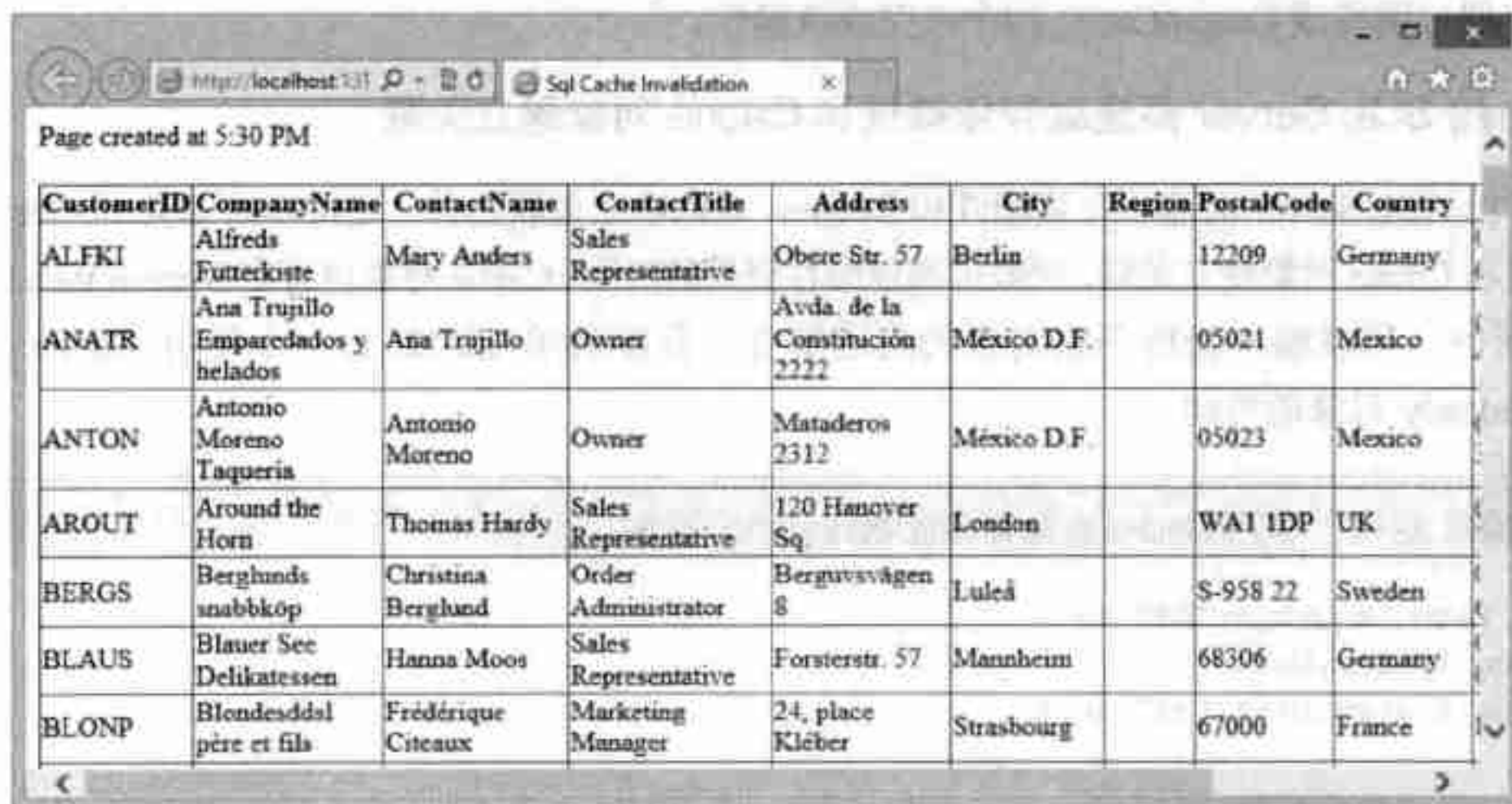
对 Northwind 数据库的 `Customers` 表中的任何单元格进行修改之后，都会禁用高速缓存，并从结果中生成新的高速缓存，其中包含新的 SQL Server 数据库请求。图 22-6 是页面第一次运行时生成的输出。



CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico
ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		68306	Germany
BLONP	Blondesdél père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg		67000	France

图 22-6

图 22-6 中显示了 CustomerID 为 ALFKI 的顾客信息。对于这一条目，在 SQL Server 中把 ContactName 的值从 Maria Anders 改为 Mary Anders。在禁用 SQL Server 高速缓存之前，这个改动对输出高速缓存没有任何影响。高速缓存中最初的页面输出仍然存在，在页面的 OutputCache 指令指定的时间期限内，终端用户仍然会看到 Maria Anders 条目。而由于使用了 SQL Server 高速缓存禁用功能，在表中的底层信息改变之后，输出高速缓存就会被禁用，并提取出新的结果集，进行高速缓存。执行修改后，结果如图 22-7 所示。



CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country
ALFKI	Alfreds Futterkiste	Mary Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico
ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		68306	Germany
BLONP	Blondesdél père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg		67000	France

图 22-7

注意，页面顶部包含了更新时间，以表示这个页面的显示时间。

22.5.1 给页面添加多个表

前面的例子说明了如何在 ASP.NET 页面的一个表中使用 SQL Server 高速缓存禁用功能。如果页面使用两个或更多个表，该怎么办？

要添加多个表，可以使用 OutputCache 指令，如下所示：


```
SqlDependency="database:table;database:table"
```

在这个例子中, SqlDependency 特性的值使用冒号分隔数据库和表。如果要使用 Northwind 数据库中的 Customers 和 Products 表, 可以按如下方式构建 SqlDependency 特性的值:

```
SqlDependency="Northwind:Customers;Northwind:Products"
```

22.5.2 给 SQL Server 高速缓存依赖性和 Request 对象建立关联

除了修改 OutputCache 指令中的设置以激活 SQL Server 高速缓存禁用功能之外, 还可以编程设置 SQL Server 高速缓存禁用功能。为此, 使用 SqlCacheDependency 类, 如程序清单 22-11 所示。

程序清单 22-11 编程设置 SQL Server 高速缓存禁用功能

```
SqlCacheDependency myDependency = new SqlCacheDependency("Northwind", "Customers");
Response.AddCacheDependency(myDependency);
Response.Cache.SetValidUntilExpires(true);
Response.Cache.SetExpires(DateTime.Now.AddMinutes(60));
Response.Cache.SetCacheability(HttpCacheability.Public);
```

首先创建 SqlCacheDependency 对象, 同时指定数据库和表的值。SqlCacheDependency 类带如下参数:

```
SqlCacheDependency(System.Data.SqlClient.SqlCommand sqlCmd)
```

有了 SqlCacheDependency 类之后, 就给 Cache 对象添加依赖性, 并且设置 Cache 对象的一些属性。可以通过编程或 OutputCache 指令来完成该操作。

22.5.3 给 SQL Server 高速缓存依赖性和 Cache 对象建立关联

除了给 SQL Server 高速缓存依赖性和 Request 对象建立关联之外, 还可以给 SQL Server 高速缓存依赖性和 Cache 对象建立关联, 把数据高速缓存更长的时间。Cache 对象包含在 System.Web.Caching 名称空间中, 可以编程处理任意对象的高速缓存。程序清单 22-12 是一个使用 Cache 对象和 SqlDependency 对象的页面。

程序清单 22-12 使用 Cache 对象和 SqlDependency 对象

```
<%@ Page Language="C#" %>
using System.Data
using System.Data.SqlClient

<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e)
    {
        DataSet myCustomers;
        myCustomers = (DataSet)Cache["firmCustomers"];

        if(myCustomers == null)
        {
            SqlConnection conn = new SqlConnection(
                ConfigurationManager.ConnectionStrings[
                    "AppConnectionString1"].ConnectionString);
```

```

        SqlDataAdapter da = new SqlDataAdapter("Select * from Customers", conn);

        myCustomers = new DataSet();
        da.Fill(myCustomers);

        SqlCacheDependency myDependency = new SqlCacheDependency(
            "Northwind", "Customers");
        Cache.Insert("firmCustomers", myCustomers, myDependency);

        Label1.Text = "Produced from database.";
    }
    else
    {
        Label1.Text = "Produced from Cache object.";
    }

    GridView1.DataSource = myCustomers;
    GridView1.DataBind();
}
</script>

<html>
<head runat="server">
    <title>Sql Cache Invalidation</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="Label1" runat="server"></asp:Label><br />
            <br />
            <asp:GridView ID="GridView1" runat="server"></asp:GridView>
        </div>
    </form>
</body>
</html>

```

在这个例子中，SqlCacheDependency 类被关联到 Northwind 数据库的 Customers 表，这一点与前面一样。但是，这次使用 Cache 对象插入提取的数据集和对 SqlCacheDependency 对象的引用。Cache 类的 Insert 方法构建如下：

```

Cache.Insert(String key, Object value,
    System.Web.Caching.CacheDependency dependencies)

```

还可以使用下面的构造代码插入有关依赖性的更多信息：

```

Cache.Insert(String key, Object value,
    System.Web.Caching.CacheDependency dependencies
    Date absoluteExpiration, System.TimeSpan slidingExpiration)

```

最后：

```

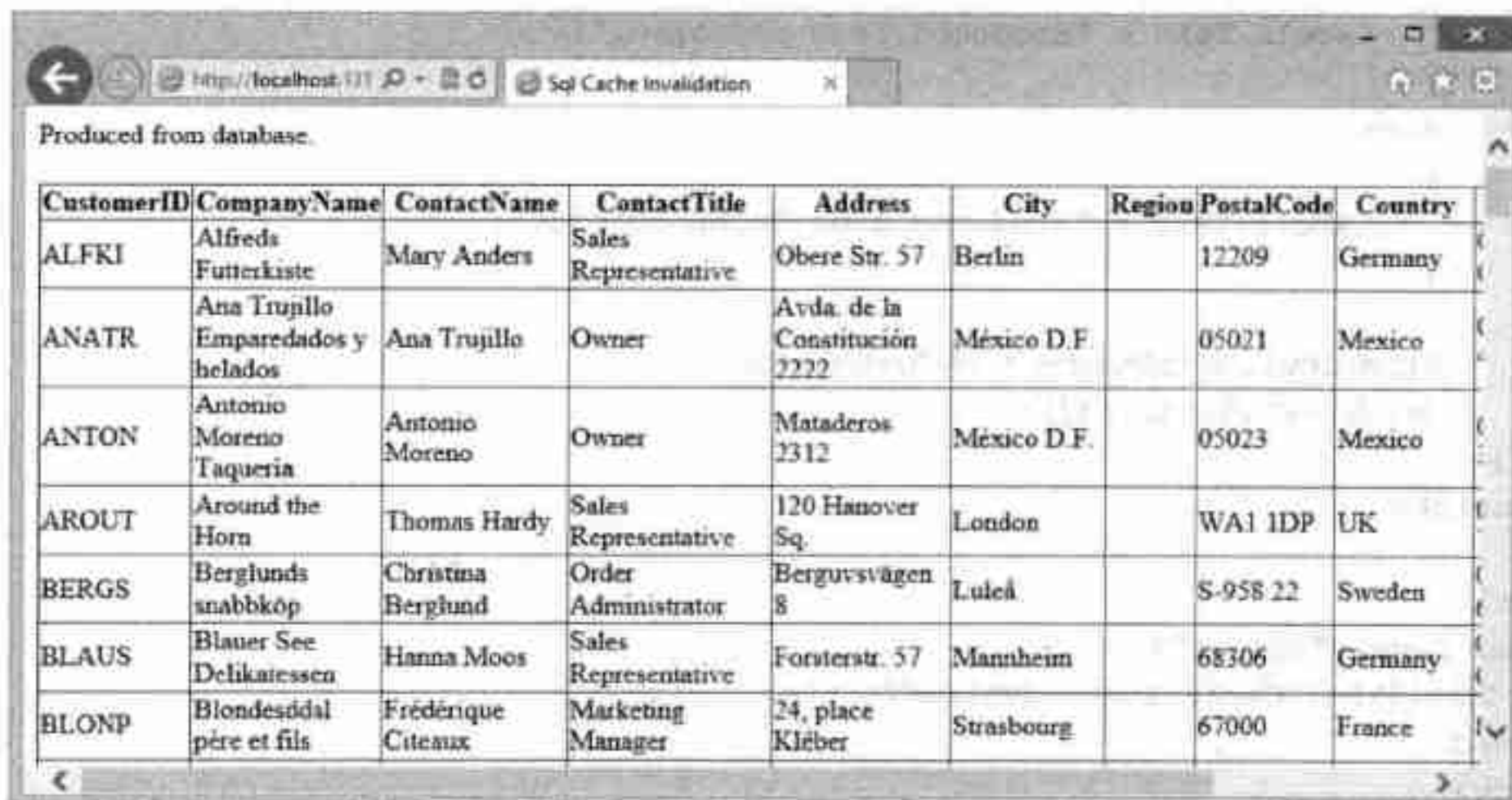
Cache.Insert(String key, Object value,
    System.Web.Caching.CacheDependency dependencies,
    Date absoluteExpiration, System.TimeSpan slidingExpiration,
    System.Web.Caching.CacheItemPriority priority,

```

```
System.Web.Caching.CacheItemRemovedCallback onRemoveCallback)
```

创建了 SQL Server 高速缓存依赖性后, 就执行与前面相同的轮询工作。如果 Customers 表中的数据发生了变化, SqlCacheDependency 类就禁用存储在高速缓存中的内容。在发出下一个请求时, Cache("firmCustomers")就为空, 并且向 SQL Server 发出新的请求。Cache 对象使用生成的新结果重新填充高速缓存。

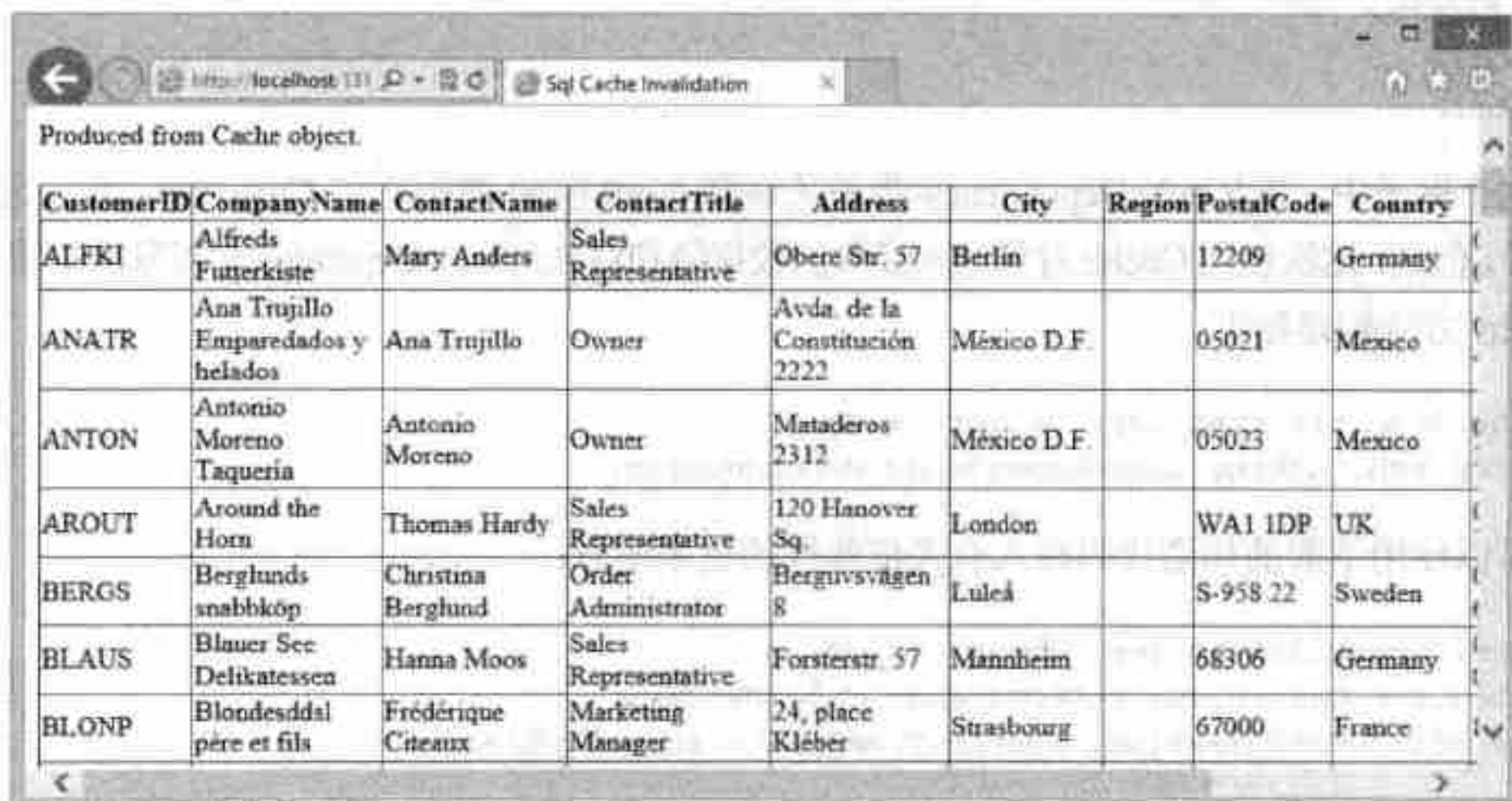
第一次调用程序清单 22-8 中的 ASP.NET 页面时, 生成的结果如图 22-8 所示。



CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country
ALFKI	Alfreds Futterkiste	Mary Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		68306	Germany
BLONP	Blondesddal père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg		67000	France

图 22-8

这是第一次生成页面, 因此高速缓存中没有任何内容。Cache 对象以及与 SQL Server 高速缓存依赖性的关联放在结果集中。图 22-9 中显示了第二个请求的结果。注意, HTML 表是相同的, 因为是从相同的 DataSet 中生成的, 但页面的第一行有变化, 说明这个输出是从高速缓存中生成的。



CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country
ALFKI	Alfreds Futterkiste	Mary Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		68306	Germany
BLONP	Blondesddal père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg		67000	France

图 22-9

在第二个请求中, 数据集已经包含在高速缓存中; 因此, 它是可以检索的。不需要再次进入 SQL Server, 获得完整的结果。但是, 如果在 SQL Server 中修改了信息, Cache 对象就不会返回任何内

容，而是检索新的结果集。

22.6 本章小结

ASP.NET 为高速缓存提供了几个内置的机制。大多数功能都可以使用特性获得，所以不需要编码。重要的是，理解不同类型的高速缓存，包括部分页面的高速缓存和 Post-Cache Substitution。

在处理数据库时，SQL Server 高速缓存禁用功能是 ASP.NET 中一项十分优秀的功能，当被监控的表中的底层数据发生变化时，它可以禁用存储在高速缓存中的数据项。Post-Cache Substitution 在 ASP.NET 技术中填补了一处重要的空白，可以把动态内容和高性能的 Web 站点高速缓存功能结合起来。

在监控数据库发生的改动时，可以简单地在 web.config 文件中配置这些过程，或者直接在代码中以编程的方式使用高速缓存禁用功能。这些改动是可行的，因为 CacheDependency 对象是不密封的。现在可以继承这个对象，创建自己的高速缓存依赖性。SQL Server 高速缓存禁用过程就是这个功能的第一个示例。

第Ⅶ部分

客户端开发

- 第 23 章 ASP.NET AJAX
- 第 24 章 AJAX 控件工具集
- 第 25 章 jQuery
- 第 26 章 实时通信
- 第 27 章 开发移动网站

第 23 章

ASP.NET AJAX

本章要点

- 使用 ASP.NET AJAX 开发应用程序
- 使用 ASP.NET AJAX 的服务器端控件

在 Web 应用程序开发中, AJAX 表示创建利用 XMLHttpRequest 对象的应用程序的能力。XMLHttpRequest 对象负责发出 HTTP 请求。

在 JavaScript 中可以创建和包含 XMLHttpRequest 对象, 另外大多数高级浏览器都支持这个对象的使用, 由此诞生了 AJAX 模型。在 Google 发布许多基于 AJAX 的著名应用程序(例如 Google Maps)之后, AJAX 应用程序变得越来越流行。这些应用程序体现出了 AJAX 的价值, 并被世界上的许多人所关注。XMLHttpRequest 最初的发明者是微软, 它把这个功能实现为用于 IE 的 ActiveX 组件, 所有其他浏览器都把它迁移到自己的浏览器中。目前, IE 把 XMLHttpRequest 支持为内部的 JavaScript 对象(其他浏览器也是这样)。

不久之后, 微软发布了一个新工具集的测试版, 该工具集可以让开发人员在 Web 应用程序中集成 AJAX 功能。这个工具集的代码名称是 Atlas, 后来重命名为 ASP.NET AJAX, 它极大地简化了在应用程序中使用 AJAX 功能的过程。

如果使用的是 ASP.NET 3.5/4/4.5, 就不需要另外安装 ASP.NET AJAX 工具集, 因为已经安装好该组件。

23.1 理解对 AJAX 的需求

ASP.NET AJAX 使 Web 应用程序比以前运行更流畅, 支持 AJAX 的应用程序响应灵敏, 给终端用户提供了即时反馈, 并沿着开发人员所提供的工作流前进。

23.1.1 AJAX 出现之前的请求/响应过程

AJAX 对 Web 应用程序有什么作用呢? 首先查看没有使用 AJAX 的 Web 页面可以做什么。图

23-1 是 Web 应用程序典型的请求和响应活动。

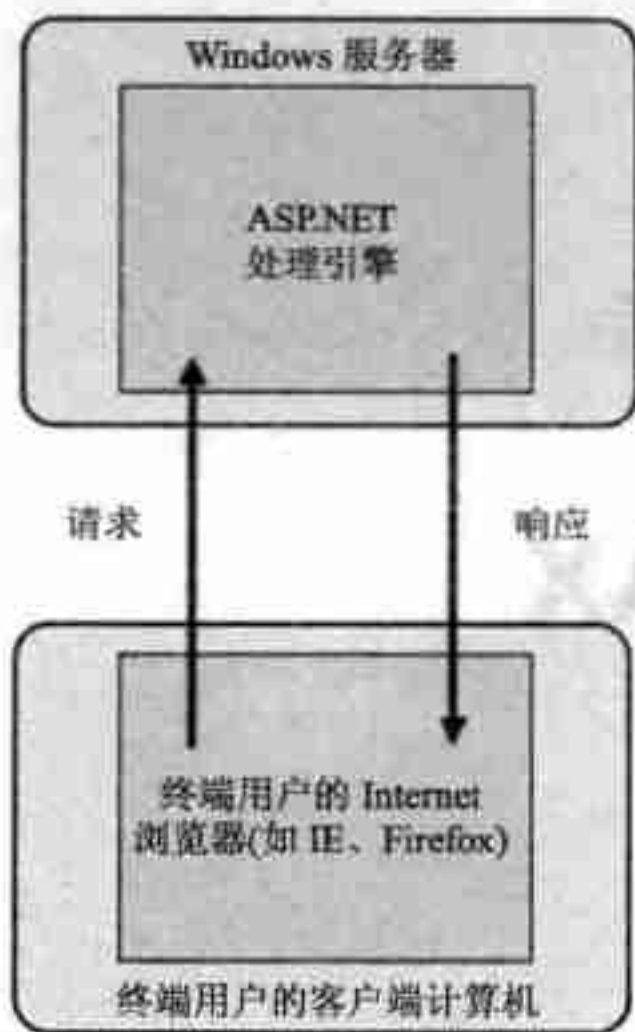


图 23-1

在图 23-1 中，终端用户从浏览器上向存储在 Web 服务器上的应用程序发出请求。服务器处理该请求，ASP.NET 显示一个页面，然后把该页面作为响应发送给请求者。终端用户接收到响应后，响应就显示在终端用户的浏览器上。

此时，应用程序实例中会发生许多事件，就好像应用程序实例位于终端用户的浏览器上一样，这些事件会再次触发完整的请求和响应过程。例如，终端用户单击了单选按钮、复选框、按钮、日历或其他对象，就刷新整个 Web 页面，或者提供新页面。

23.1.2 AJAX 改变了请求/响应过程

与前方讨论的过程相反，支持 AJAX 的 Web 页面在客户端上包含一个 JavaScript 库，该库负责向 Web 服务器发出请求。在发送请求并得到部分页面的响应时，这个 JavaScript 库就会调用 Web 服务器，在使用脚本时也是如此；接着客户端上的库更新这部分页面，但没有更新整个页面。整个页面的更新需要给浏览器发送许多代码，每次都要处理所有这些代码。如果只处理部分页面，终端用户就会觉得页面比较“流畅”，页面似乎响应速度更快。更新部分页面所需的代码量比较少，可产生终端用户期望的响应速度。图 23-2 显示了这个请求/响应过程的示意图。

如图 23-2 所示，首先在最初的请求和响应中发送整个页面。之后，使用客户端脚本库完成页面所需的部分更新。这个库可以进行异步页面请求，更新需要更新的部分页面。这么做的一大优点是更新操作所需传送的数据最少。在只对页面有很少的改动时，更新部分页面比再次调用整个页面更好。

AJAX 依赖于几个技术。第一个是 XMLHttpRequest 对象，这个对象允许浏览器与后台服务器通信，通过 MSXML ActiveX 组件可以在微软环境下使用该对象(自 Internet Explorer 5 版本之后)。当然，另一个重要的组件是 JavaScript。这个技术允许客户端开始与后台服务通信，封装一条消息，之后发送给任意服务器端的服务。AJAX 的另一个方面是支持 DOM(Document Object Model)。在接收到来自服务器的异步响应时，这些技术会改变页面。最后一部分是从客户端传送到服务器上的数据，采用 XML 格式或更重要的 JSON 格式完成该传送。

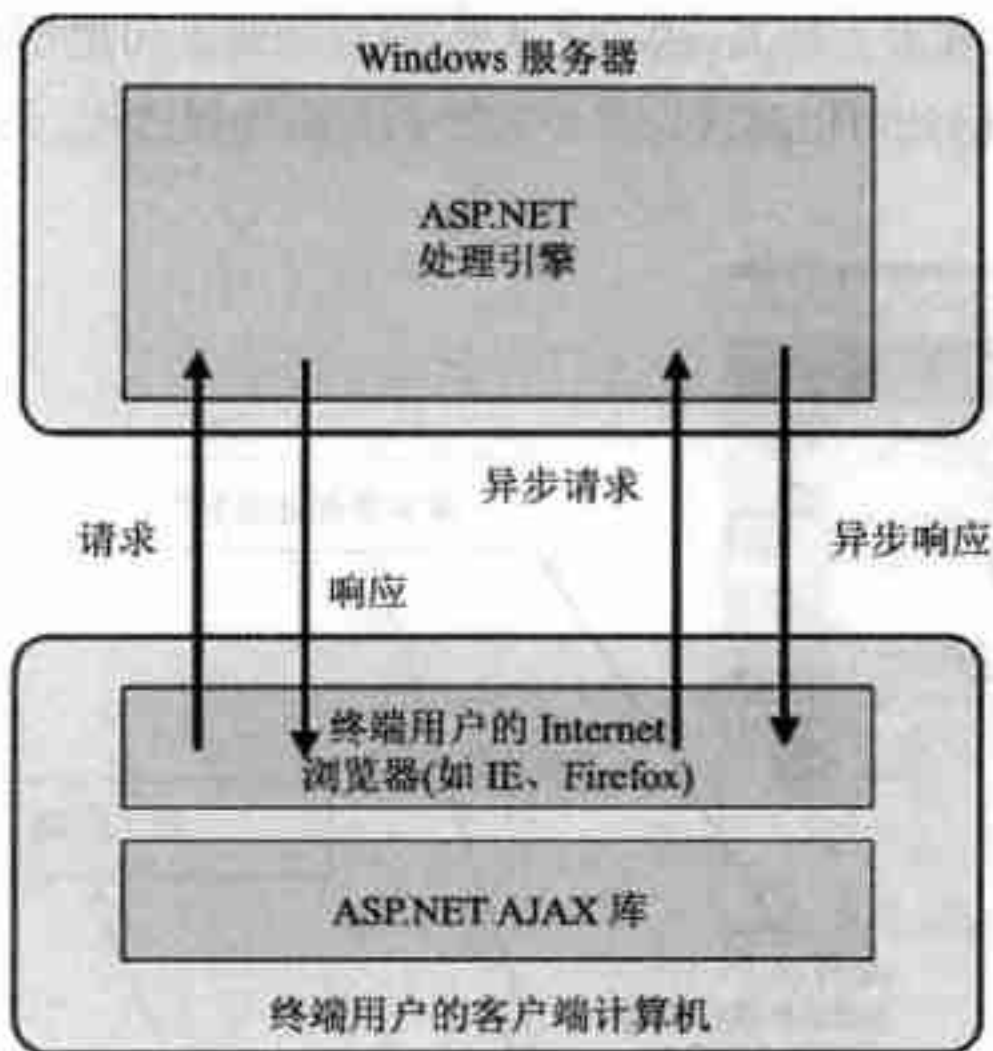


图 23-2

对 XMLHttpRequest 对象的支持使客户端脚本库中的 JavaScript 函数可以调用服务器端的事件。如前所述，典型的 HTTP 请求一般由浏览器发出。浏览器也负责处理来自服务器的响应，在发出响应后，再在浏览器中重新生成整个 Web 页面。该过程如图 23-3 所示。



图 23-3

如果使用 JavaScript 库中的 XMLHttpRequest 对象，就不能使用浏览器启动对整个页面的请求，

而要使用客户端的脚本引擎(基本上是 JavaScript 函数)来启动请求和接收响应。由于不是发出请求和响应来处理整个 Web 页面,因此可以跳过许多不需要的页面处理过程。这就是 AJAX Web 请求的本质,如图 23-4 所示。



图 23-4

AJAX 打开了方便之门。微软也提供了必要的脚本引擎,能自动完成需要的通信工作,以支持 AJAX 类型的功能。

23.2 ASP.NET AJAX 和 Visual Studio 2012

ASP.NET AJAX 不仅是 Visual Studio 2012 IDE 的一部分,它还被内置到 .NET Framework 4.5 中。这就说明,如果使用的是 ASP.NET 4.5,那么使用 ASP.NET AJAX 不需要执行任何安装。



如果使用的是 ASP.NET 3.5 以前的版本,就需要访问 www.asp.net/ajax,以获得使用 AJAX 所需的组件。

总之,微软全面集成了整个 ASP.NET AJAX 技术,因此很容易使用 Visual Studio 及其可视化设计器来处理支持 AJAX 的页面,甚至对应用程序进行全面调试。使用 Visual Studio 2012,还可以调试页面中使用的 JavaScript。

另外,注意微软在 ASP.NET AJAX 的跨平台兼容方面也花费了很多精力。在 .NET Framework 4.5

中创建的支持 AJAX 的应用程序可以在所有主流浏览器(如 Firefox 和 Opera)上运行。

23.2.1 客户端技术

ASP.NET AJAX 有两个部分：第一部分是客户端架构和一系列完全位于客户端的服务，另一部分是服务器端架构。注意，ASP.NET AJAX 的客户端与服务器端就异步请求进行通信。

为此，微软提供了一个客户端脚本库，这是一个 JavaScript 库，负责必要的通信。客户端脚本库如图 23-5 所示。

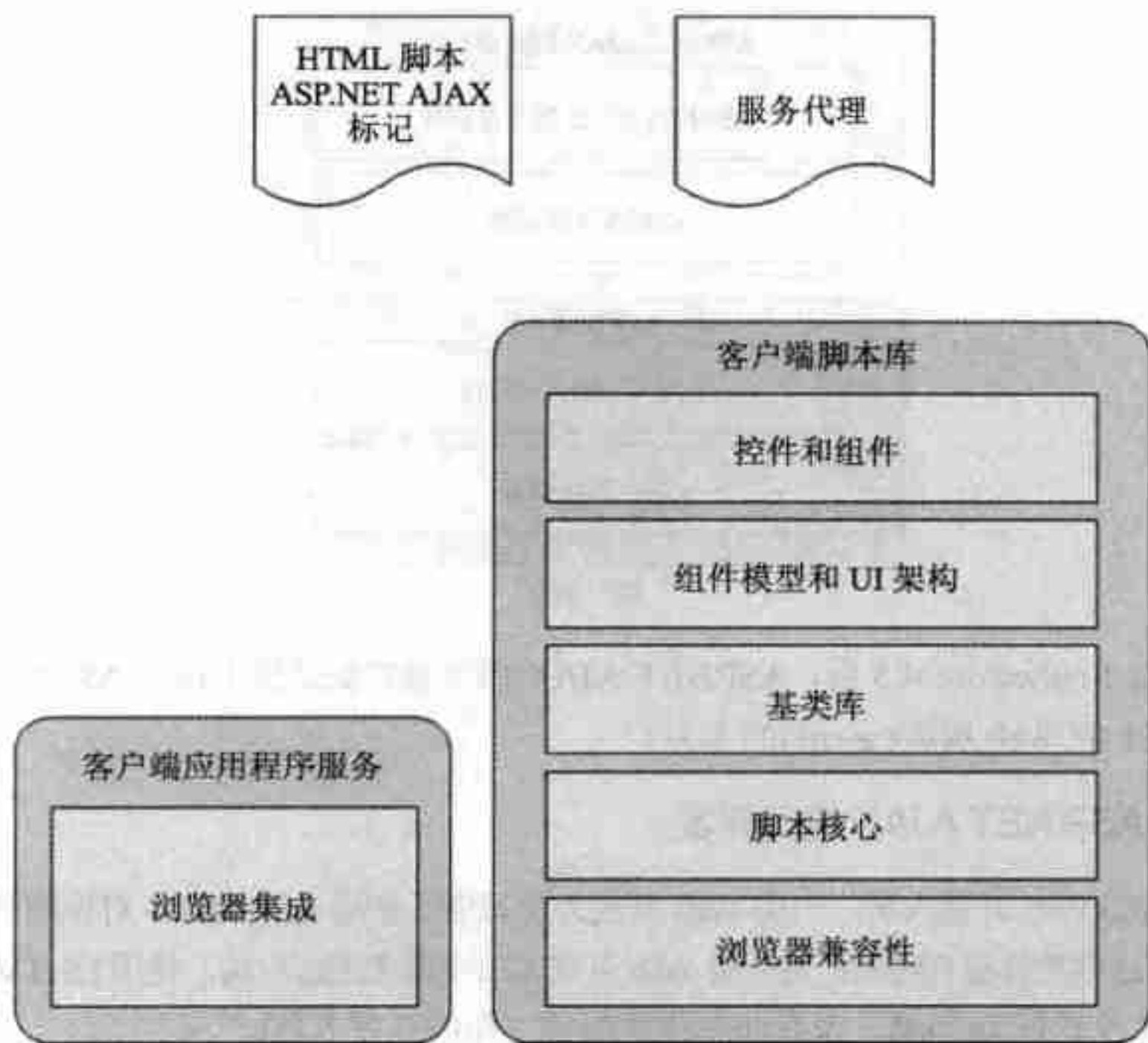


图 23-5

客户端脚本库提供了面向对象的 Javascript 界面，它与 .NET Framework 的各个方面都很一致。因为浏览器兼容组件是内置的，所以在这一层上完成的所有工作(或在大多数情况下)让 ASP.NET AJAX 执行的工作都可以在许多不同的浏览器上完成。另外，有一些组件支持丰富的 UI 基础结构，该结构完成了我们自己建立 UI 时需要大量时间的许多工作。

ASP.NET AJAX 提供的客户端技术的一个有趣之处是它们完全独立于 ASP.NET。实际上，任何开发人员都可以免费下载微软的 AJAX 库(从 asp.net/ajax 上下载)，把它和其他 Web 技术结合使用，例如 PHP(php.net)和 Java Server Pages (JSP)。因此，整个 Web 在有了 ASP.NET AJAX 提供的服务器端技术之后才更加完整。

即使微软现在包含了 JavaScript 库 jQuery(参见第 25 章)，ASP.NET AJAX 也仍有一些独特、方便的功能。

23.2.2 服务器端技术

ASP.NET 开发人员很可能在 ASP.NET AJAX 的服务器端方面花费大部分时间，而 ASP.NET

AJAX 主要是客户端与服务器端之间的通信。可以在 ASP.NET AJAX 的服务器端执行许多任务。

服务器端架构知道如何处理客户端请求(例如使响应具有正确的格式),还负责在 JavaScript 对象和服务器端代码中使用的.NET 对象之间编组(marshaling)对象。图 23-6 是 ASP.NET AJAX 提供的服务器端架构。



图 23-6

安装了 .NET Framework 4.5 后, ASP.NET AJAX 服务器扩展就位于核心 ASP.NET 架构、WCF 和基于 ASP.NET 的 Web 服务(.asmx)的上方。

23.2.3 使用 ASP.NET AJAX 进行开发

有两种类型的 Web 开发人员:一类 Web 开发人员习惯于使用 ASP.NET,对使用服务器端控件和在服务器端处理这些控件很有经验;另一类 Web 开发人员主要考虑客户端,使用 DHTML 和 JavaScript 处理和控制页面及其行为(当然,也有能完成这两类工作的开发人员)。

因此一定要意识到,ASP.NET AJAX 是为这两种开发人员设计的。如果更多是工作在 ASP.NET AJAX 的服务器端,就可以使用新的 ScriptManager 和 UpdatePanel 控件为当前的 ASP.NET 应用程序提供 AJAX 支持,而不用做什么工作。可以使用我们熟悉的 ASP.NET 编程模型来完成所有这些工作。



ScriptManager 和 UpdatePanel 控件详见本章后面的内容。

然而,也可以直接使用客户端脚本库,对客户端计算机上发生的事件进行更多控制。本章接下来将创建一个使用 AJAX 的简单 Web 应用程序。

23.3 创建 ASP.NET AJAX 应用程序

理解了 AJAX 的原理后,接下来创建一个利用这个新架构的基本示例。首先使用 New Web Site 对话框创建一个新的空的 ASP.NET Web Site 应用程序,把项目命名为任意名称,如图 23-7 所示。



图 23-7

应用程序创建好后，它就是一个标准的 Web Site 项目。如果使用 ASP.NET 3.5 创建该应用程序，就要注意 web.config 文件中一些新增的设置。此时，在 web.config 文件的顶部注册了一些新的配置部分来处理 AJAX。而在使用 ASP.NET 4.5 的情况下，这个原来位于 web.config 文件中的配置部分现在可以在 machine.config 文件中找到。machine.config 文件的这个配置部分如程序清单 23-1 所示。

程序清单 23-1 machine.config 文件中的<configSections>元素

```
<?xml version="1.0"?>

<configuration>
  <configSections>
    <sectionGroup name="system.web.extensions"
type="System.Web.Configuration.SystemWebExtensionsSectionGroup, System.Web.Extensions,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35">
      <sectionGroup name="scripting"
type="System.Web.Configuration.ScriptingSectionGroup, System.Web.Extensions,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35">
        <section name="scriptResourceHandler"
type="System.Web.Configuration.ScriptingScriptResourceHandlerSection,
System.Web.Extensions, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
requirePermission="false" allowDefinition="MachineToApplication"/>
        <sectionGroup name="webServices"
type="System.Web.Configuration.ScriptingWebServicesSectionGroup, System.Web.Extensions,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35">
          <section name="jsonSerialization"
type="System.Web.Configuration.ScriptingJsonSerializationSection, System.Web.Extensions,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
requirePermission="false" allowDefinition="Everywhere"/>
          <section name="profileService"
```



```

type="System.Web.Configuration.ScriptingProfileServiceSection, System.Web.Extensions,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
requirePermission="false" allowDefinition="MachineToApplication"/>
    <section name="authenticationService"
type="System.Web.Configuration.ScriptingAuthenticationServiceSection,
System.Web.Extensions, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
requirePermission="false" allowDefinition="MachineToApplication"/>
    <section name="roleService"
type="System.Web.Configuration.ScriptingRoleServiceSection, System.Web.Extensions,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
requirePermission="false" allowDefinition="MachineToApplication"/>
    </sectionGroup>
  </sectionGroup>
</sectionGroup>
</configSections>

<!-- Configuration removed for clarity -->

</configuration>

```

下一步是建立一个还没有使用 AJAX 的简单 ASP.NET 页面。

23.3.1 建立没有使用 AJAX 的简单 ASP.NET 页面

建立一个还没有使用 ASP.NET 4.5 提供的 AJAX 功能的简单页面，该页面只需一个 Label 控件和一个 Button 服务器控件。该页面的代码如程序清单 23-2 所示。

程序清单 23-2 一个没有使用 AJAX 的简单 ASP.NET 4.5 页面

```

<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = DateTime.Now.ToString();
    }
</script>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>My Normal ASP.NET Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="Label1" runat="server"></asp:Label>
            <br />
            <br />
            <asp:Button ID="Button1" runat="server" Text="Click to get machine time"
                onclick="Button1_Click" />
        </div>
    </form>
</body>

```



```
</html>
```

在浏览器中打开这个页面时，可以看到它只包含一个按钮。单击该按钮，页面上的 Label 控件就会填充来自服务器的时间。在单击按钮之前，页面的代码如程序清单 23-3 所示。

程序清单 23-3 没有使用 AJAX 的页面的输出

```
<!DOCTYPE html>
<html>
<head><title>
    My Normal ASP.NET Page
</title></head>
<body>
    <form method="post" action="Listing 23-02.aspx" id="form1">
    <div class="aspNetHidden">
    <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="pVEIMzKuHfHM8A10duc6kbGylGmyb0H8CjA0LI9dKlVRyylcUJXhQRd/p3cvsGTRor7wtWsUZGgu8uEWGn
FcX3/tLVfqhgqN8aAP4Jxb9M0=" />
    </div>

    <div class="aspNetHidden">

        <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
value="/QH9EPcAvi4a4BLcl82HKfMOBKUpXA+2dZkl9KS33lgGgRBza5bYURjDLLkyCwA6OKzqthVHfjDFy9aW9i
l4V4EYWRlN974paiCVSfbfEa8jprjN5bAl2MbXFMtIVBZu" />
    </div>
    <div>
        <span id="Label1"></span>
        <br />
        <br />
        <input type="submit" name="Button1" value="Click to get machine time" id="Button1" />
    </div>
    </form>
</body>
</html>
```

这段代码非常简单。此处有几个 ViewState 和一个传送回 Default.aspx 页面的普通表单。终端用户单击页面上的按钮时，会执行服务器回传的完整过程，重新处理整个页面，并返回给客户端的浏览器。对页面的唯一修改是：使用一个值填充元素，但在此处返回了整个页面。

23.3.2 建立使用 AJAX 的简单 ASP.NET 页面

下一步是以程序清单 23-2 中的页面为基础，添加 AJAX 功能。对于这个例子，需要添加一些控件。要添加的两个控件是一般的 ASP.NET 服务器控件：另一个 Label 控件和另一个 Button 服务器控件。除了这些控件之外，还要添加一些 ASP.NET AJAX 控件。

在 Visual Studio 2012 的工具箱中，新的部分 AJAX Extensions 如图 23-8 所示。

在 AJAX Extensions 部分，把一个 ScriptManager 服务器控件添加到页面的顶部，并在 UpdatePanel 控件中包含第二个 Label 控件和 Button 控件。UpdatePanel 控件是模板服务器控件，允许在其中包含任意多项(与其他 ASP.NET 模板服务器控件一



图 23-8

样)。建立好的页面如图 23-9 所示。



图 23-9

这个页面的代码如程序清单 23-4 所示。

程序清单 23-4 一个简单的 ASP.NET AJAX 页面

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = DateTime.Now.ToString();
    }

    protected void Button2_Click(object sender, EventArgs e)
    {
        Label2.Text = DateTime.Now.ToString();
    }
</script>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>My ASP.NET AJAX Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
            <asp:Label ID="Label1" runat="server"></asp:Label>
            <br />
            <br />
            <asp:Button ID="Button1" runat="server" Text="Click to get machine time">

```

```

        onclick="Button1_Click" />
<br />
<br />
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <ContentTemplate>
        <asp:Label ID="Label2" runat="server" Text=""></asp:Label>
        <br />
        <br />
        <asp:Button ID="Button2" runat="server"
            Text="Click to get machine time using Ajax"
            onclick="Button2_Click" />
    </ContentTemplate>
</asp:UpdatePanel>
</div>
</form>
</body>
</html>

```

在浏览器中打开这个页面，会看到它包含两个按钮。第一个按钮会回送完整的页面，更新 Label1 服务器控件上的当前时间。单击第二个按钮会进行 AJAX 异步回送，更新 Label2 服务器控件上的当前服务器时间。单击第二个按钮时，Label1 控件上显示的时间不会改变，因为它在 UpdatePanel 控件的外面。最终结果如图 23-10 所示。

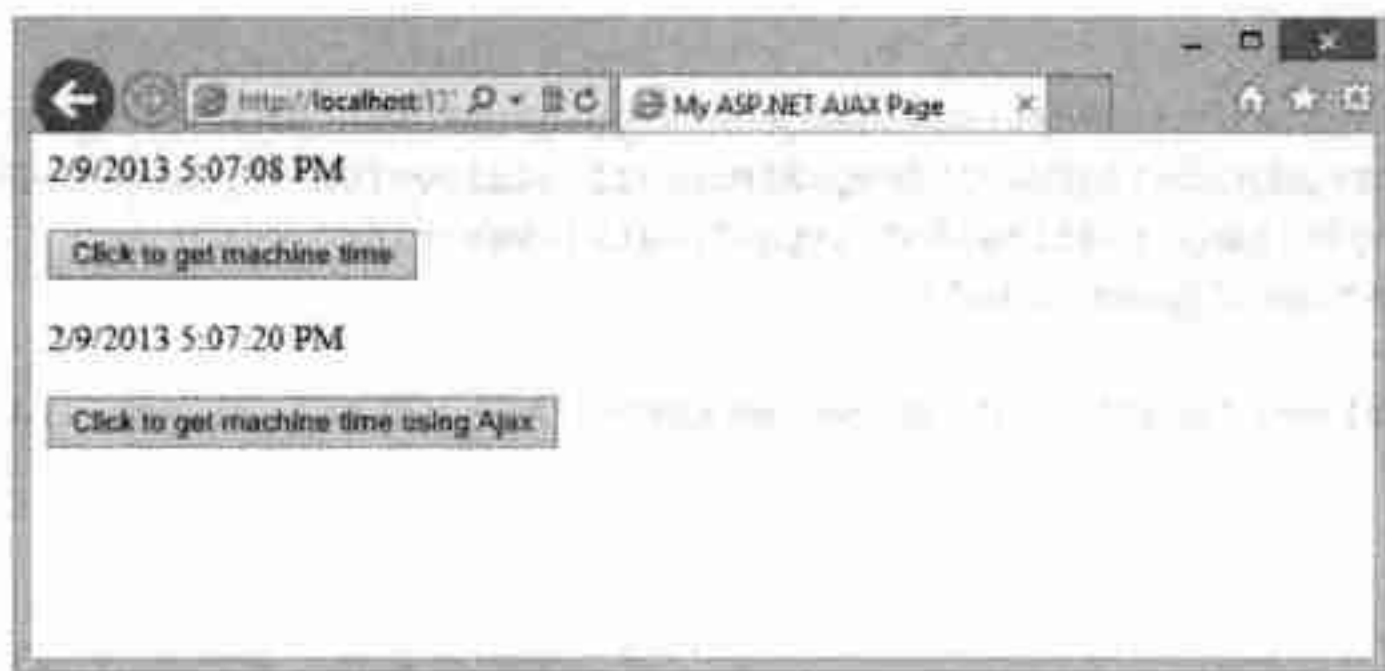


图 23-10

第一次查看程序清单 23-4 中的页面时，该页面的代码与没有使用 AJAX 的页面完全不同。程序清单 23-5 列出了页面的输出。

程序清单 23-5 使用 AJAX 的页面的输出

```

<!DOCTYPE html>
<html>
<head><title>
    My ASP.NET AJAX Page
</title></head>
<body>
    <form method="post" action="Listing 23-04.aspx" id="form1">
    <div class="aspNetHidden">
    <input type="hidden" name="__EVENTTARGET" id="__EVENTTARGET" value="" />
    <input type="hidden" name="__EVENTARGUMENT" id="__EVENTARGUMENT" value="" />
    <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"

```



```

value="WWtnNoYlMaNb9VaXbk6Zu8WKDrp6d2htzRxkrRGD7jPvrZP2WPLX0FA7YUBF4pYZRlQogz6EK+PuFjiYhw
MDKAFkqBFCct1EC2uWNZzT+xI=" />
</div>

<script type="text/javascript">
//
var theForm = document.forms['form1'];
if(!theForm) {
    theForm = document.form1;
}
function __doPostBack(eventTarget, eventArgument) {
    if(!theForm.onsubmit || (theForm.onsubmit() != false)) {
        theForm.__EVENTTARGET.value = eventTarget;
        theForm.__EVENTARGUMENT.value = eventArgument;
        theForm.submit();
    }
}
//]]&gt;
&lt;/script&gt;

&lt;script
src="/WebResource.axd?d=pynGkmcFUVl3HelQd6_TZOfsGAlirZhbNCK-_Bbdm5qc7-I-DHw9-fyqR7RqDfDX
C3Wksp3fzYjv78lBXTZtA2&amp;t=634776625276709129" type="text/javascript"&gt;&lt;/script&gt;

&lt;script
src="/ScriptResource.axd?d=D9drwtSJ4hBA6O8UhT6CQhrg7DRKGxEuvpwpedj7vopaTC0bITXA9Dx80QJ3ec
SfbXl4t2bNWufm4ED2tyMQPnb6IIgfUv2Zl7WvQn25Nc6r2Il_j0ZlDvnIupcl5FHAgUL6JsUsSt_CcKQNorkPXUg
aBAhSYXnodIyqbWtUy3Yl&amp;t=6119e399" type="text/javascript"&gt;&lt;/script&gt;
&lt;script type="text/javascript"&gt;
//<![CDATA[
if(typeof(Sys) === 'undefined') throw new Error('ASP.NET Ajax client-side framework failed
to load.');</pre>
</div>
<div data-bbox="81 564 916 608" data-label="Text">
<pre>
//]]&gt;
&lt;/script&gt;

&lt;script
src="/ScriptResource.axd?d=JnUc-DEDOM5KzzVKtsLltbPwQqzhEDldD8elz4bygtpKPKhJiqq0b_09X56JcI
Gvkszfuglvdyvz3Efx8suOaifmIwTH_7frKsQOuAjd6Fm3WaKZGEdw75nIgO80ypkfh2PCNd7QinNN1Xyd7uh6dRO
IOZ6HqaAT27wpkGD-lfYDELdKBmimdCxNgTRLQi6M0&amp;t=6119e399"
type="text/javascript"&gt;&lt;/script&gt;
&lt;div class="aspNetHidden"&gt;

    &lt;input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
value="RGMilvtxaurgSA6tbndg3zXfgDOK9D+pWkEkUg4X/kJbGksDfFNL5ML2hEEetD+00phBDkOCzf97bxcGQc
og/Bk/jsv574AGkbqT26VM8KsxUvIFRSJWeA3b6vs/4gGwQMU4fPn2S1Rlv5kfVEwOdQ==" /&gt;
&lt;/div&gt;
&lt;div&gt;
    &lt;script type="text/javascript"&gt;
//<![CDATA[
Sys.WebForms.PageRequestManager._initialize('ScriptManager1', 'form1',
['tUpdatePanel1','UpdatePanel1'], [], [], 90, '');
//]]&gt;
&lt;/script&gt;

    &lt;span id="Label1"&gt;&lt;/span&gt;
</pre>
</div>
<div data-bbox="48 952 84 966" data-label="Page-Footer">788</div>
```

```

        <br />
        <br />
        <input type="submit" name="Button1" value="Click to get machine time" id="Button1" />
        <br />
        <br />
        <div id="UpdatePanell1">

            <span id="Label2"></span>
            <br />
            <br />
            <input type="submit" name="Button2" value="Click to get machine time using
Ajax" id="Button2" />

        </div>
    </div>
</form>
</body>
</html>

```

现在,如果单击 Button1,进行整个页面的回送,就会在响应中得到这段代码,即使是只希望更新页面的一小部分!但是,如果单击 Button2,就会发送如程序清单 23-6 所示的请求。

程序清单 23-6 ASP.NET AJAX 页面的异步请求

```

POST /Listing%2023-04.aspx HTTP/1.1
Accept: */*
X-Requested-With: XMLHttpRequest
X-MicrosoftAjax: Delta=true
Cache-Control: no-cache
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Referer: http://localhost:1315/Listing%2023-04.aspx
Accept-Language: en-US,en;q=0.8,de-DE;q=0.5,de;q=0.3
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0)
Host: localhost:1315
Content-Length: 457
DNT: 1
Connection: Keep-Alive

ScriptManager1=UpdatePanell1%7CButton2&__EVENTTARGET=&__EVENTARGUMENT=&__VIEWSTATE=AGU
aegmKl6ukZ0igyXSeqIUWlYM%2F56cB9CvQ%2BKmbx6wzbEtmJZ3b5wehOayVnxSlfia6%2BEH9ZF98axT8I7yvNN
WGS5qzl%2BlxWxTFNqFTb%2FSb0lYWeXwW3lq%2BnqvTOxz0l2aGjtZov9oKuWW2qFU%2F%2B7J%2B4e2HDAkJTKd
lH6dlVAc%3D&__EVENTVALIDATION=sKpjfJEGzCQ6ObwXMOilu%2BSAz2Of34luW4OCIOfTPbUgYxWzz9G6nJLAh
lr3NzY7Fa4vcU0p4Ft783W%2BP5KzZGu4HkiUROxRRltrx3hE%2FtkFy%2B%2B9%2Bx0lmRQsnaDK%2F1jYNW52e4
38i4wQ5SyqQAadWA%3D%3D&__ASYNCPPOST=true&Button2=Click%20to%20get%20machine%20time%20using
%20AJAX

```

这个异步请求的响应如程序清单 23-7 所示。

程序清单 23-7 ASP.NET AJAX 页面的异步响应

```

HTTP/1.1 200 OK
Cache-Control: private

```

```
Content-Type: text/plain; charset=utf-8
Server: Microsoft-IIS/8.0
X-AspNet-Version: 4.0.30319
X-SourceFiles:
=?UTF-8?B?QzpcVXNlcnNcQ2hyaXN0aWwFuXERvY3VtZW50c1xWaXNlYWwgU3RlZGlvIDIwMTJcV2ViU2l0ZXNcV3J
veFwMaXN0aW5nIDE4LTA0LmFzcHg=?=
X-Powered-By: ASP.NET
Date: Fri, 07 Dec 2012 21:09:56 GMT

1|#||4|240|updatePanel|UpdatePanel1|
    <span id="Label2">12/7/2012 10:09:56 PM</span>
    <br />
    <br />
    <input type="submit" name="Button2" value="Click to get machine time using
Ajax" id="Button2" />

|172|hiddenField|__VIEWSTATE|zhb9R4rHclav1lCCq/gAoOeF7XPxtQaoiv64pBVoJEhaVevAbRJq6e9rDJe5
fC0thgx05lLs55qg42/6uEUys1ld1hlzM0t0JmLv55CIhsyD5meWk6kxHkYO4X0jxdIVj8YbphKp+CB485z/cPAzZ
k1LkQs7J96ZtzLGfg1lcv4=|152|hiddenField|__EVENTVALIDATION|sCJzmPa+sVbkZ2zvON/kvEn6/7qjGZK
pheMuCF34am8ZaaHtdk+Zdepz+uf97h+RZWWAv5/is+Mwig6DEPEOOI+oAwOMqfPwnUOF1Dt3G5L/6KwE3EPyK1XL
md9Xm/29BKYhMORGxwO3lH2/wuBRXQ==|0|asyncPostBackControlIDs||0|postBackControlIDs||26|up
datePanelIDs||tUpdatePanel1,UpdatePanel1|0|childUpdatePanelIDs||25|panelsToRefreshIDs||U
pdatePanel1,UpdatePanel1|2|asyncPostBackTimeout||90|18|formAction||Listing
23-04.aspx|20|pageTitle||My ASP.NET AJAX Page|
```

从程序清单 23-7 中可以看出，这个响应比整个 Web 页面小很多。事实上，响应的主要部分只是包含在 UpdatePanel 服务器控件中的代码。输出结果底部的项处理页面的 ViewState(因为它现在有改动)和其他一些页面改动。

23.4 ASP.NET AJAX 的服务器控件

查看 Visual Studio 2012 工具箱中的 AJAX Extensions 部分，可以注意到其中没有多少控件。这些控件主要用于给 ASP.NET 应用程序添加 AJAX 支持，它们是激活控件。如果要查找利用 AJAX 模型的更特殊的服务器控件，可以查看 ASP.NET AJAX 控件工具集，这是单独下载的工具集，详见第 24 章。

ASP.NET 4.5 中的 ASP.NET AJAX 服务器控件如表 23-1 所示。

表 23-1

ASP.NET AJAX 服务器控件	说 明
ScriptManager	这个组件控件管理消息的编组，为需要部分更新的页面提供支持 AJAX 的服务器。每个 ASP.NET 页面都需要一个 ScriptManager 控件。注意，一个页面上只能有一个 ScriptManager 控件
ScriptManagerProxy	这个组件控件用作内容页面的 ScriptManager 控件。ScriptManagerProxy 控件位于内容页面(或子页面)上，与位于母版页上的 ScriptManager 控件协同工作
Timer	这个控件以指定的时间间隔执行客户端事件，允许页面的指定部分在这些时间间隔内更新或刷新

(续表)

ASP.NET AJAX 服务器控件	说 明
UpdatePanel	这个容器控件允许定义页面的某些区域支持使用 ScriptManager, 之后这些区域就可以回送部分页面, 在正常的 ASP.NET 页面回送过程之外更新它们自身
UpdateProgress	这个控件允许给终端用户显示一个可视化元素, 说明部分页面回送操作正在更新页面的某个部分。这是长时间运行 AJAX 更新的理想控件

接下来将介绍这些控件, 以及如何在 ASP.NET 页面中使用它们。

23.4.1 ScriptManager 控件

在 ASP.NET AJAX 领域中, 最重要的控件是 ScriptManager 服务器控件, 它负责处理页面, 允许进行部分页面的显示。每个要使用 ASP.NET 4.5 提供的 AJAX 功能的页面都需要使用一个 ScriptManager 控件。有了 ScriptManager 控件和 UpdatePanel 控件, 支持 AJAX 功能的 ASP.NET 应用程序就可以在页面上添加两个服务器控件以执行操作。

ScriptManager 控件负责管理在页面上使用的 JavaScript 库, 并在服务器和客户端之间来回编组消息, 以完成部分页面的显示过程。可以使用 SOAP 或 JSON, 通过 ScriptManager 控件完成消息的编组。

如果仅在 ASP.NET 页面上放置了 ScriptManager 控件, 它就会负责加载 ASP.NET AJAX 需要的 JavaScript 库。页面如程序清单 23-8 所示。

程序清单 23-8 只包含 ScriptManager 控件的 ASP.NET 页面

```
<%@ Page Language="C#" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>The ScriptManager Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
        </div>
    </form>
</body>
</html>
```

从程序清单 23-8 中可以看出, 这个控件类似于其他 ASP.NET 控件, 只需要 ID 和 runat 属性即可工作。这段 ASP.NET 代码的页面输出如程序清单 23-9 所示。

程序清单 23-9 ScriptManager 控件的页面输出

```
<!DOCTYPE html>
<html>
<head><title>
```

```

        The ScriptManager Control
    </title></head>
    <body>
        <form method="post" action="Listing 23-08.aspx" id="form1">
        <div class="aspNetHidden">
            <input type="hidden" name="__EVENTTARGET" id="__EVENTTARGET" value="" />
            <input type="hidden" name="__EVENTARGUMENT" id="__EVENTARGUMENT" value="" />
            <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="zpV5crkAwVuhffQapeLOpcyY7E3sDEmepTrY/bsvgCn/drFkzuIEIbGMZSnm8fkgNjzDGQ29rapFR3M/2j
UcMfMh6z29ldYhyQUzLL0wLQA=" />
        </div>

        <script type="text/javascript">
        //
        var theForm = document.forms['form1'];
        if(!theForm) {
            theForm = document.form1;
        }
        function __doPostBack(eventTarget, eventArgument) {
            if(!theForm.onsubmit || (theForm.onsubmit() != false)) {
                theForm.__EVENTTARGET.value = eventTarget;
                theForm.__EVENTARGUMENT.value = eventArgument;
                theForm.submit();
            }
        }
        //]]&gt;
        &lt;/script&gt;

        &lt;script
src="/WebResource.axd?d=pynGkmcFUV13HelQd6_TZOfsGAlIiRzhbNCK-_Bbdm5qc7-I-DHw9-fyqR7RqDfDX
C3Wksp3fzYjv78lBXTZtA2&amp;amp;t=634776625276709129" type="text/javascript"&gt;&lt;/script&gt;

        &lt;script
src="/ScriptResource.axd?d=D9drwtSJ4hBA608Uht6CQhrg7DRKGxEuvpwpedj7vopaTC0bITXA9Dx80QJ3ec
SfbXi4t2bNWufm4ED2tyMQPnb6IIgfUv2Z17WvQn25Nc6r2I1_j0Z1DvnIupcl5FHAguL6JsUsSt_CcKQNorkPXUg
aBAhSYXnodIyqbwtUy3Yl&amp;amp;t=6119e399" type="text/javascript"&gt;&lt;/script&gt;
        &lt;script type="text/javascript"&gt;
        //<![CDATA[
        if (typeof(Sys) === 'undefined') throw new Error('ASP.NET Ajax client-side framework failed
to load. ');
        //]]&gt;
        &lt;/script&gt;

        &lt;script
src="/ScriptResource.axd?d=JnUc-DEDOM5KzzVKtsL1tbPwOqzhEDldD8elz4bygtpKPKhJiqq0b_09X56JcI
GvkszfugIvdyyvz3Efx8suOaifmIwTH_7frKsQOuAjd6Fm3WaKZGEdw75nIg080ypkfh2PCNd7QinNN1Xyd7uh6dRO
IOZ6HqaAT27wpgGD-lfYDELDBKMimdCxNgTRLQi6M0&amp;amp;t=6119e399"
type="text/javascript"&gt;&lt;/script&gt;
        &lt;div&gt;
            &lt;script type="text/javascript"&gt;
            //<![CDATA[
            Sys.WebForms.PageRequestManager._initialize('ScriptManager1', 'form1', [], [], [], 90, '');
            //]]&gt;
            &lt;/script&gt;
</pre>
</div>
<div data-bbox="57 951 93 965" data-label="Page-Footer">
<p>792</p>
</div>
```

```

        </div>
    </form>
</body>
</html>

```

该页面的输出显示,有许多 JavaScript 库与页面一起加载。还要注意,脚本源代码是动态注册的,可通过 ScriptResource.axd 处理程序提供的 HTTP 处理程序获得。



如果要查看 JavaScript 库的内容,可以在浏览器的地址栏中输入 src 特性的 URL,并下载所引用的 JavaScript 文件。屏幕会提示保存 ScriptResource.axd 文件,可以重命名该文件,使用 .txt 或 .js 扩展名。

ScriptManager 控件的有趣之处在于它通过执行额外的步骤压缩脚本来处理发送给客户端的脚本。

23.4.2 ScriptManagerProxy 控件

ScriptManagerProxy 控件在本书的第 16 章已经介绍过,因为这个控件专门用于母版页和用户控件。与 23.4.1 节中介绍的 ScriptManager 控件一样,在要使用 ASP.NET AJAX 的每个页面上都需要一个 ScriptManager 控件。但是如前所述,问题在于若使用母版页,该怎么办? 需要把 ScriptManager 控件放在母版页上吗? 它如何处理使用了母版页的内容页面?

从 Add New Item 对话框中创建新的母版页时,只是选择了 Master Page 选项和修改了一些代码以处理 AJAX,如程序清单 23-10 所示。

程序清单 23-10 AJAX 母版页

```

<%@ Master Language="C#" %>

<script runat="server">

</script>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>Untitled Page</title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server" />
            <asp:ContentPlaceHolder id="ContentPlaceHolder1"
                runat="server">

            </asp:ContentPlaceHolder>
        </div>
    </form>

```



```
</body>
</html>
```

这段代码说明，页面上的确有一个 `ScriptManager` 控件，这个页面将被添加到使用该母版页的每个内容页面上。不必对内容页面做任何特殊的操作，就可以使用母版页提供的 ASP.NET AJAX 功能。可以像创建其他内容页面那样创建这个内容页面。

但是，如果要以任何方式修改母版页上的 `ScriptManager` 控件，就必须在内容页面上添加一个 `ScriptManagerProxy` 控件，如程序清单 23-11 所示。

程序清单 23-11 在内容页面上添加 `ScriptManagerProxy` 控件

```
<%@ Page Language="C#" MasterPageFile="~/AJAXMaster.master" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1"
runat="Server">
<asp:ScriptManagerProxy ID="ScriptManagerProxy1" runat="server">
    <Scripts>
        <asp:ScriptReference Path="myOtherScript.js" />
    </Scripts>
</asp:ScriptManagerProxy>
</asp:Content>
```

在这个例子中，内容页面在母版页的 `ScriptManager` 控件上插入了一个该内容页面的脚本引用。如果在内容页面上使用 `ScriptManagerProxy` 控件，但母版页上没有 `ScriptManager` 控件，就会出错。

23.4.3 Timer 控件

在 ASP.NET 页面上处理异步回送时，一项常见的任务是希望这些异步回送以特定的时间间隔发生。为此，可以使用工具箱的 AJAX Extensions 部分的 `Timer` 控件。下面的简单例子演示了这个控件如何工作，该例把一些时间戳放在页面上，并把回送设置为以指定的时间间隔进行，如程序清单 23-12 所示。

程序清单 23-12 使用 `Timer` 控件

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            Label1.Text = DateTime.Now.ToString();
        }
    }

    protected void Timer1_Tick(object sender, EventArgs e)
    {
        Label1.Text = DateTime.Now.ToString();
    }
}
```

```

</script>

<!DOCTYPE html>
<html>
<head id="Head1" runat="server">
    <title>Timer Example</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server" />
            <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                <ContentTemplate>
                    <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
                    <asp:Timer ID="Timer1" runat="server" OnTick="Timer1_Tick"
                        Interval="10000">
                    </asp:Timer>
                </ContentTemplate>
            </asp:UpdatePanel>
        </div>
    </form>
</body>
</html>

```

在这个例子中，页面上只有 3 个控件。第 1 个是 ScriptManager 控件，其后是 Label 和 Timer 控件。这个页面第一次加载时，通过调用 Page_Load 事件处理程序使用 DateTime 值填充 Label 控件。在第一次加载时给 Label 控件填充 DateTime 值后，Timer 控件就负责修改这个值。

Timer 控件的 OnTick 属性可以完成这项任务，它指向到达 Interval 属性指定的时间间隔时触发的函数。

Interval 属性设置为 10 000，即 10 000 毫秒(1000 毫秒等于 1 秒)。这就表示，每隔 10 秒进行一次异步回送，调用一次 Timer1_Tick 函数。

运行这个页面，会看到页面上的时间每隔 10 秒变化一次。

23.4.4 UpdatePanel 控件

UpdatePanel 服务器控件是 AJAX 特有的控件，在处理 AJAX 时经常使用 UpdatePanel 控件。这个控件保存回送模型，允许执行部分页面的显示。

UpdatePanel 控件是一个容器控件，这表示它没有相关的 UI 项。它是触发部分页面回送并且仅更新指定的部分页面的方式。

1. <ContentTemplate>元素

有两种方式可以处理页面上引发异步页面回送的控件。第一种方式最简单，如程序清单 23-13 所示。

程序清单 23-13 把触发器放在 UpdatePanel 控件内部

```

<%@ Page Language="C#" %>

<script runat="server">

```

```

        protected void Button1_Click(object sender, EventArgs e)
        {
            Label1.Text = "This button was clicked on " + DateTime.Now.ToString();
        }
    }
</script>

<!DOCTYPE html>
<html>
<head id="Head1" runat="server">
    <title>UpdatePanel Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
            <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                <ContentTemplate>
                    <asp:Label ID="Label1" runat="server"></asp:Label>
                    <br />
                    <br />
                    <asp:Button ID="Button1" runat="server"
                        Text="Click to initiate async request"
                        OnClick="Button1_Click" />
                </ContentTemplate>
            </asp:UpdatePanel>
        </div>
    </form>
</body>
</html>

```

在这个例子中, Label 和 Button 服务器控件都包含在 UpdatePanel 服务器控件中。<asp:UpdatePanel> 元素包含两个子元素: <ContentTemplate> 和 <Triggers>。需要在异步页面回送中改变的内容都应包含在 UpdatePanel 控件的 <ContentTemplate> 部分。

在默认情况下, 包含在 <ContentTemplate> 部分的任何类型的控件触发器(一般会触发页面回送)都会触发异步页面回送。这就表示, 在程序清单 23-13 中, 页面上的按钮会触发异步页面回送, 而不是完整页面回送。每次单击该按钮, 都会改变显示在 Label 控件上的时间。

2. <Triggers>元素

程序清单 23-13 演示了这个模型存在的如下严重问题: 发生异步回送时, 不仅回送 Label 控件的日期/时间值, 还回送页面上按钮的所有代码。

```

1|#||4|282|updatePanel|UpdatePanel1|
    <span id="Label1">This button was clicked on 12/7/2012 10:15:13 PM</span>
    <br />
    <br />
    <input type="submit" name="Button1" value="Click to initiate async
request" id="Button1" />

```



```
|216|hiddenField|__VIEWSTATE|9SRD65c4+HKABtyXj1cwTSdqRx5djmaW4UM/pnsbaV6BW9Yp3Jt0dMEUO49q
M6bpcgVY7ZGaizibE3WuKeC39UPCgR3ivwhh+47AYvrBWPoKY7YpNww7zr7mo5HoMqgYJeUnJIP1XQml2/gFiO18C
mEbk/9u0t156Bjjw/rEWOYJdNSXHBxsWJgGe2cA+JZstvkNnOaJtGBj1OHfkkhqCg==|128|hiddenField|__EVE
NTVALIDATION|rcyhqzLnEyzijEQwniWhKYVo9K47PWymKxRO+QMdU2BwNSArt0P8hu33J7bhc+80rIB2IUeWXR3
YlsPZyDoFl2BoHyJ7gCCmjzhPUyUrSH4c0xR3c9EjVqL2Bl2QYvK|0|asyncPostBackControlIDs|||0|postBa
ckControlIDs|||26|updatePanelIDs||tUpdatePanel1,UpdatePanel1|0|childUpdatePanelIDs|||25|p
anelsToRefreshIDs||UpdatePanel1,UpdatePanel1|2|asyncPostBackTimeout||90|18|formAction||Li
sting 23-13.aspx|19|pageTitle||UpdatePanel Control|
```

这段通过异步回送返回给客户端的代码显示, 包含在 UpdatePanel 控件中的所有部分都再次发出。如果能只包含页面更新的部分, 就可以减小页面尺寸。如果把按钮放在 UpdatePanel 控件中 <ContentTemplate> 部分的外面, 就必须在该控件中包含 <Triggers> 部分。

必须包含 <Triggers> 部分的原因是, 要通过异步回送来改变的内容都包含在 <ContentTemplate> 部分, 但是必须关联一个页面事件以引发回送。这就是使用 UpdatePanel 控件的 <Triggers> 部分的方式。使用该部分可以指定引发异步页面回送的各种触发器。在 UpdatePanel 控件中使用 <Triggers> 元素, 可以重写程序清单 23-13, 如程序清单 23-14 所示。

程序清单 23-14 使用触发器引发异步页面回送

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = "This button was clicked on " + DateTime.Now.ToString();
    }
</script>

<!DOCTYPE html>
<html>
<head id="Head1" runat="server">
    <title>UpdatePanel</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
            <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                <ContentTemplate>
                    <asp:Label ID="Label1" runat="server"></asp:Label>
                </ContentTemplate>
                <Triggers>
                    <asp:AsyncPostBackTrigger ControlID="Button1" EventName="Click" />
                </Triggers>
            </asp:UpdatePanel>
        </div>
    </form>
</body>
```

```

        <asp:Button ID="Button1" runat="server"
            Text="Click to initiate async request"
            OnClick="Button1_Click" />
    </div>
</form>
</body>
</html>

```

在这个例子中，Button 控件和 HTML 元素在 UpdatePanel 控件的<ContentTemplate>部分之外，因此在每次异步页面回送时不发送回客户端。唯一包含在<ContentTemplate>部分的项是页面上需要通过回送改变的项——Label 控件。把它们关联在一起的就是<Triggers>部分。

<Triggers>部分可以包含两个控件：AsyncPostBackTrigger 和 PostBackTrigger。这个例子使用了 AsyncPostBackTrigger 控件。PostBackTrigger 控件会执行完整页面回送，而 AsyncPostBackTrigger 控件仅执行异步页面回送(正如该控件的名称所示)。

从使用了 AsyncPostBackTrigger 元素的程序清单 23-14 中可以看出，只使用了两个属性就把 Button 控件和异步回送的触发器关联在一起：ControlID 和 EventName。要用作异步页面回送触发器的控件放在 ControlID 属性中(使用控件的 ID 属性指定控件名)。EventName 属性的值是在 ControlID 属性中指定的控件的事件名，在客户端的异步请求中调用该事件。在这个例子中，调用了 Button 控件的单击事件，这个事件改变了位于 UpdatePanel 控件的<ContentTemplate>部分的控件值。

运行这个页面，单击按钮，会向客户端返回一个较小的异步响应：

```

1|#|4|113|updatePanel|UpdatePanel1|
    <span id="Label1">This button was clicked on 12/7/2012 10:16:00 PM</span>
1216|hiddenField|__VIEWSTATE|exYy/K/aPNML3q+zrqHMWbTkzeycwcLnou
HX4mfKnCfPJggjY8k7//1iUwxhyUEg2eJU/QLD4Cgr4E/B/V7oss0Rcn2z05s17I/0oK2mw0MKM0/OQ3F2Nt
qeCR5rwiC9FiqbVx8G8V52uW0CLlpdnLarj9pi/NSyYpeeVmEAXapYQklsDPvcSKpHmjhIWihfJ0QmSvO6Yv7LUSk
4OFETg==|128|hiddenField|__EVENTVALIDATION|MdQFKILTdqEbZgfpPoEQWHNM/A1icFRG9iE9Qy+b6wB4LQ
ZQXWfe9OK1Y6nNzLoFtTFX/hOdRHxv1Fci3EZNZSfpap+bu5DHqP+wDA6qL85imfWkquq8youly59ztNKU|15|asy
ncPostBackControlIDs||Button1,Button1|0|postBackControlIDs||26|updatePanelIDs||tUpdatePa
nell,UpdatePanel1|0|childUpdatePanelIDs||25|panelsToRefreshIDs||UpdatePanel1,UpdatePanel
1|2|asyncPostBackTimeout||90|18|formAction||Listing_23-14.aspx|11|pageTitle||UpdatePanel1

```

该响应不比前面的例子小多少，但这是因为本例使用的页面比较小(页面越大，改进就越明显)。有很多内容的页面，响应的规模可能小很多，这取决于使用 UpdatePanel 控件构建页面的方式。

3. 使用 Visual Studio 2012 建立触发器

如果在建立 ASP.NET 页面时喜欢使用 Visual Studio 的设计界面，就会发现 Visual Studio 设计界面为建立 ASP.NET AJAX 页面提供了很好的支持，包括在 UpdatePanel 控件中创建触发器。为了说明这一点，把一个 UpdatePanel 服务器控件放在页面上，在 Visual Studio 的 Properties 窗口中查看该控件。列表中的 Triggers 项的旁边有一个按钮，可以用于修改与它关联的项，如图 23-11 所示。

单击 Properties 窗口中的这个按钮，会打开 UpdatePanelTrigger Collection Editor 对话框，如图 23-12 所示。这个编辑器允许用户添加任意数量的触发器，并且非常方便地把它们关联到控件和控件事件。

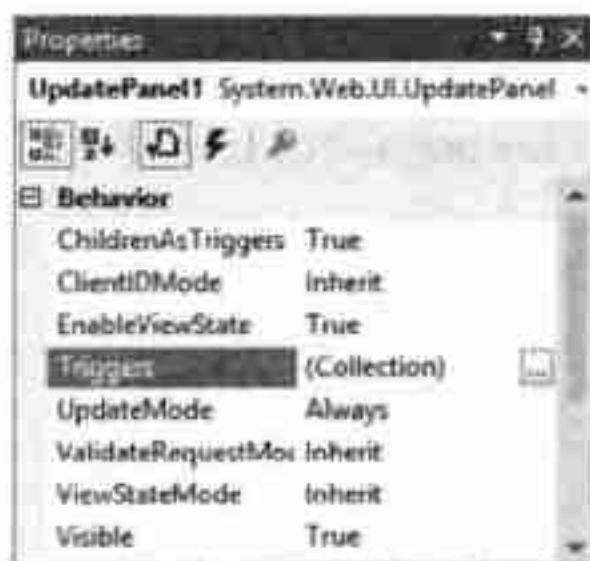


图 23-11

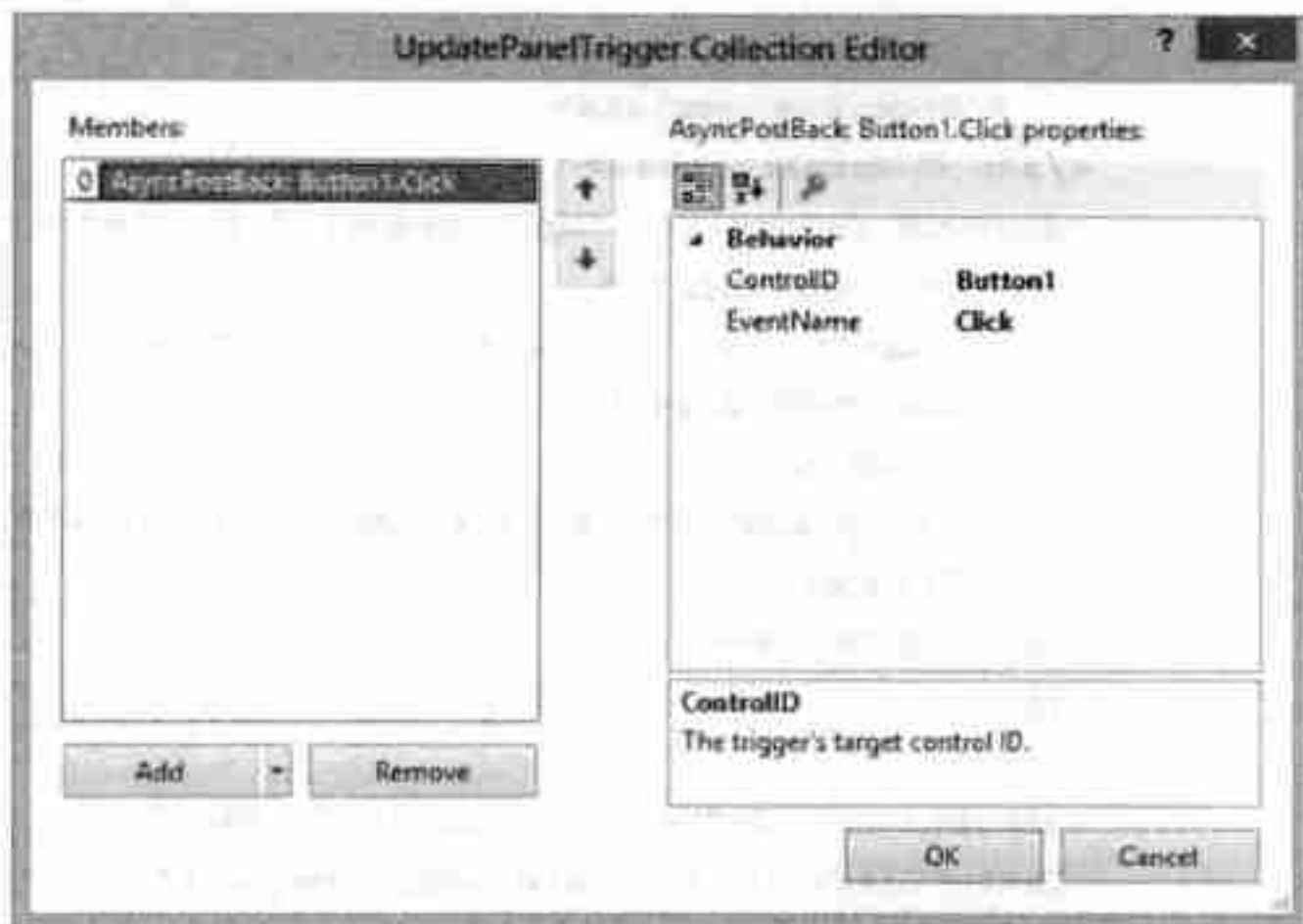


图 23-12

单击 OK 按钮，把触发器添加到 UpdatePanel 控件的<Triggers>部分。

23.4.5 UpdateProgress 控件

Visual Studio 2012 工具箱的 AJAX Extensions 部分的最后一个服务器控件是 UpdateProgress。一些异步回送需要执行一段时间，这是因为响应比较大或获得结果以发送回客户端所需的计算时间较长。UpdateProgress 控件为客户端提供了一个可视化的指示器，用于显示工作完成的情况，并会很快得到结果(而不是显示为页面锁定)。

程序清单 23-15 是 UpdateProgress 控件的文本实现方式。

程序清单 23-15 使用 UpdateProgress 控件向客户端显示文本消息

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        System.Threading.Thread.Sleep(10000);
        Label1.Text = "This button was clicked on " + DateTime.Now.ToString();
    }
</script>

<!DOCTYPE html>
<html>
<head id="Head1" runat="server">
    <title>UpdatePanel</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
            <asp:UpdateProgress ID="UpdateProgress1" runat="server">
                <ProgressTemplate>
```



```

        An update is occurring...
    </ProgressTemplate>
</asp:UpdateProgress>
<asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Conditional">
    <ContentTemplate>
        <asp:Label ID="Label1" runat="server"></asp:Label>
    </ContentTemplate>
    <Triggers>
        <asp:AsyncPostBackTrigger ControlID="Button1" EventName="Click" />
    </Triggers>
</asp:UpdatePanel>
<br />
<br />
<asp:Button ID="Button1" runat="server"
    Text="Click to initiate async request"
    OnClick="Button1_Click" />
</div>
</form>
</body>
</html>

```

为了给响应添加一些延迟(以模拟长时间运行的计算机进程),此处调用了 `Thread.Sleep` 方法。这里,在页面中希望显示更新消息的部分添加了一个 `UpdateProgress` 控件。在这个例子中,将 `UpdateProgress` 控件添加到 `UpdatePanel` 服务器控件的上面。这个控件没有放在 `UpdatePanel` 控件的内部,而是放在该控件的外面。但是,与 `UpdatePanel` 控件一样, `UpdateProgress` 控件也是模板控件。

`UpdateProgress` 控件只有子元素 `<ProgressTemplate>`。在触发 `UpdateProgress` 控件时,放在控件的这个部分中的内容就会显示出来。在这个例子中,仅把一些文本放在控件的这个部分中。运行这个页面,会得到如图 23-13 所示的更新。



图 23-13

在这个例子中,文本会立即显示出来,并且在异步回送完成之前不会消失。放在 `<ProgressTemplate>` 部分的代码包含在页面中,但通过 CSS 关闭显示:

```

<div id="UpdateProgress1" style="display:none;">
    An update is occurring...
</div>

```

1. 控制消息显示的时间

现在只要单击按钮, UpdateProgress 就会立即显示出来。但是, 一些过程不会执行那么长时间, 我们也并不总是希望进度通知从客户端消失。UpdateProgress 控件的 DisplayAfter 属性允许控制进度更新消息的显示时间。DisplayAfter 属性的用法如程序清单 23-16 所示。

程序清单 23-16 使用 DisplayAfter 属性

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server" DisplayAfter="5000">
  <ProgressTemplate>
    An update is occurring...
  </ProgressTemplate>
</asp:UpdateProgress>
```

DisplayAfter 属性的值是数字, 表示 UpdateProgress 控件在显示包含在<ProgressTemplate>部分中的内容之前的等待时间(毫秒)。程序清单 23-16 中的代码指定, <ProgressTemplate>部分中的文本在 5000 毫秒(5 秒)后显示。

2. 在<ProcessTemplate>部分添加图像

前面的例子使用了包含文本的 UpdateProgress 控件, 但是还可以把任意内容放在这个模板控件中。例如, 可以把旋转图像放在该控件中, 向终端用户说明正在处理请求。旋转图像的使用如程序清单 23-17 所示。

程序清单 23-17 在<ProcessTemplate>部分中使用旋转图像

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server"
  DisplayAfter="5000">
  <ProgressTemplate>
    <asp:Image ID="Image1" runat="server"
      ImageUrl="~/spinningwheel.gif" />
  </ProgressTemplate>
</asp:UpdateProgress>
```

与前面的文本例子一样, 用于图像的代码放在客户端的页面实例中, 只是通过 CSS 关闭显示:

```
<div id="UpdateProgress1" style="display:none;">
  
</div>
```

23.5 使用多个 UpdatePanel 控件

本章前面介绍了如何使用单个 UpdatePanel 控件, 但是还可以在一个页面上使用多个 UpdatePanel 控件。这样就可以控制页面上指定区域的输出。

使用多个 UpdatePanel 控件的例子如程序清单 23-18 所示。

程序清单 23-18 使用多个 UpdatePanel 控件

```
<%@ Page Language="C#" %>
```

```

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = "Label1 was populated on " + DateTime.Now;
        Label2.Text = "Label2 was populated on " + DateTime.Now;
    }
</script>

<!DOCTYPE html>
<html>
<head id="Head1" runat="server">
    <title>Multiple UpdatePanel Controls</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
            <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                <ContentTemplate>
                    <asp:Label ID="Label1" runat="server"></asp:Label>
                </ContentTemplate>
                <Triggers>
                    <asp:AsyncPostBackTrigger ControlID="Button1" EventName="Click" />
                </Triggers>
            </asp:UpdatePanel>
            <asp:UpdatePanel ID="UpdatePanel2" runat="server">
                <ContentTemplate>
                    <asp:Label ID="Label2" runat="server"></asp:Label>
                </ContentTemplate>
            </asp:UpdatePanel>
            <br />
            <br />
            <asp:Button ID="Button1" runat="server"
                Text="Click to initiate async request"
                OnClick="Button1_Click" />
        </div>
    </form>
</body>
</html>

```

这是一个有趣的页面，其上有两个 UpdatePanel 控件：UpdatePanel1 和 UpdatePanel2。它们都包含一个 Label 控件，这两个 Label 控件都可以从服务器响应中提取日期/时间值。

UpdatePanel1 控件有一个关联的触发器：页面上的 Button 控件。单击这个按钮时，Button1_Click() 事件会触发，并执行其操作。如果运行这个页面，就会根据 Button1_Click() 事件来更新这两个 UpdatePanel 控件，如图 23-14 所示。

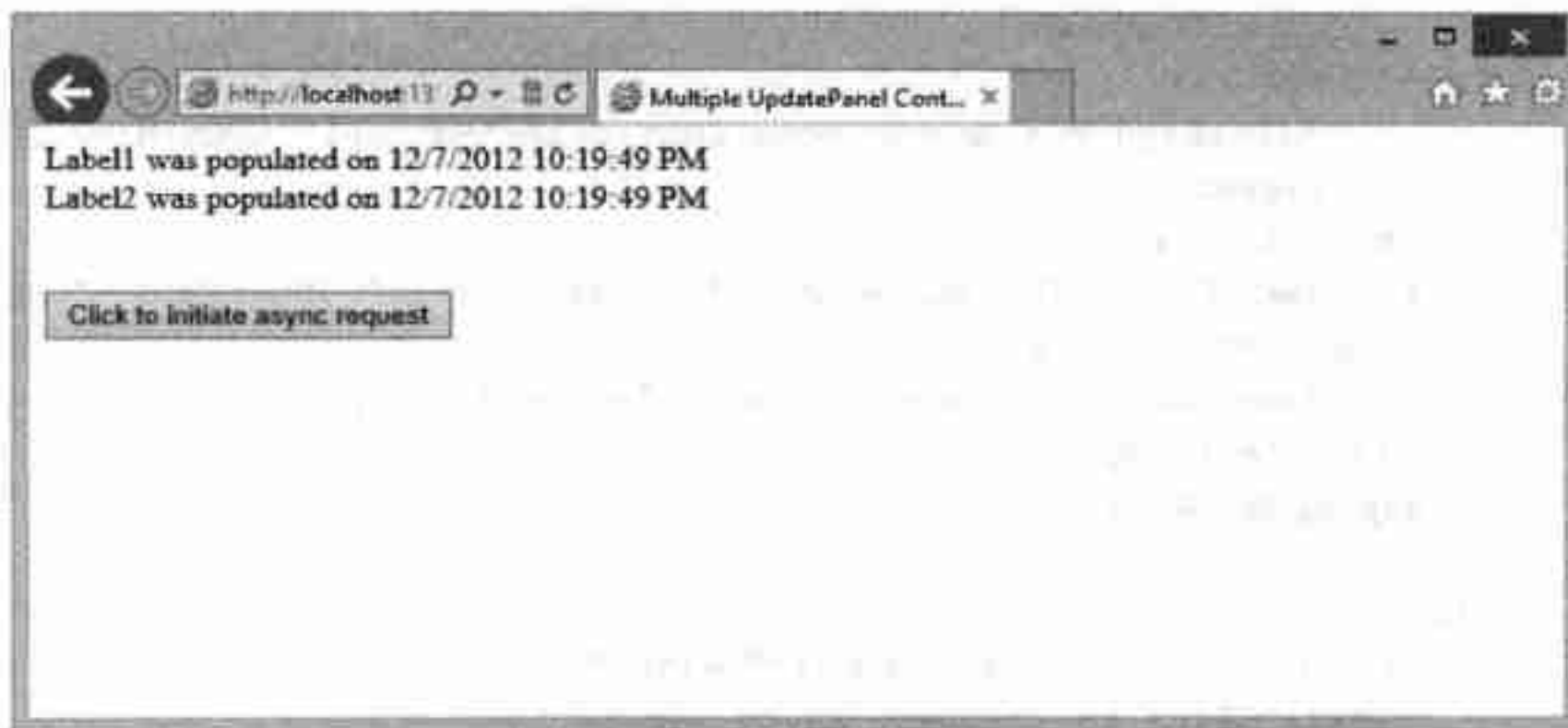


图 23-14

使用 `Button1_Click()` 事件更新两个 `UpdatePanel` 部分，因为在默认情况下，单个页面上的所有 `UpdatePanel` 控件都在每个异步回送发生时更新。这就表示，`Button1` 按钮控件引发的回送也会引发 `UpdatePanel2` 控件的回送。

通过 `UpdatePanel` 控件的 `UpdateMode` 属性可以控制这种行为。`UpdateMode` 属性有两个枚举值：`Always` 和 `Conditional`。如果不设置这个属性，它就会使用 `Always` 值，表示每个 `UpdatePanel` 控件总是在每次异步请求时更新。

另一种选择是把该属性设置为 `Conditional`，表示 `UpdatePanel` 控件仅在满足一个触发条件时更新。例如，修改页面上的 `UpdatePanel` 控件，它们现在使用 `UpdateMode` 属性的 `Conditional` 值，如程序清单 23-19 所示。

程序清单 23-19 使用多个 `UpdatePanel` 控件

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = "Label1 was populated on " + DateTime.Now;
        Label2.Text = "Label2 was populated on " + DateTime.Now;
    }
</script>

<!DOCTYPE html>
<html>
<head id="Head1" runat="server">
    <title>Multiple UpdatePanel Controls</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
            <asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Conditional">
                <ContentTemplate>
                    <asp:Label ID="Label1" runat="server"></asp:Label>
```

```

</ContentTemplate>
<Triggers>
  <asp:AsyncPostBackTrigger ControlID="Button1" EventName="Click" />
</Triggers>
</asp:UpdatePanel>
<asp:UpdatePanel ID="UpdatePanel2" runat="server" UpdateMode="Conditional">
  <ContentTemplate>
    <asp:Label ID="Label2" runat="server"></asp:Label>
  </ContentTemplate>
</asp:UpdatePanel>
<br />
<br />
<asp:Button ID="Button1" runat="server"
  Text="Click to initiate async request"
  OnClick="Button1_Click" />
</div>
</form>
</body>
</html>

```

两个 UpdatePanel 控件都把 UpdateMode 属性设置为 Conditional。运行这个页面时，结果如图 23-15 所示。

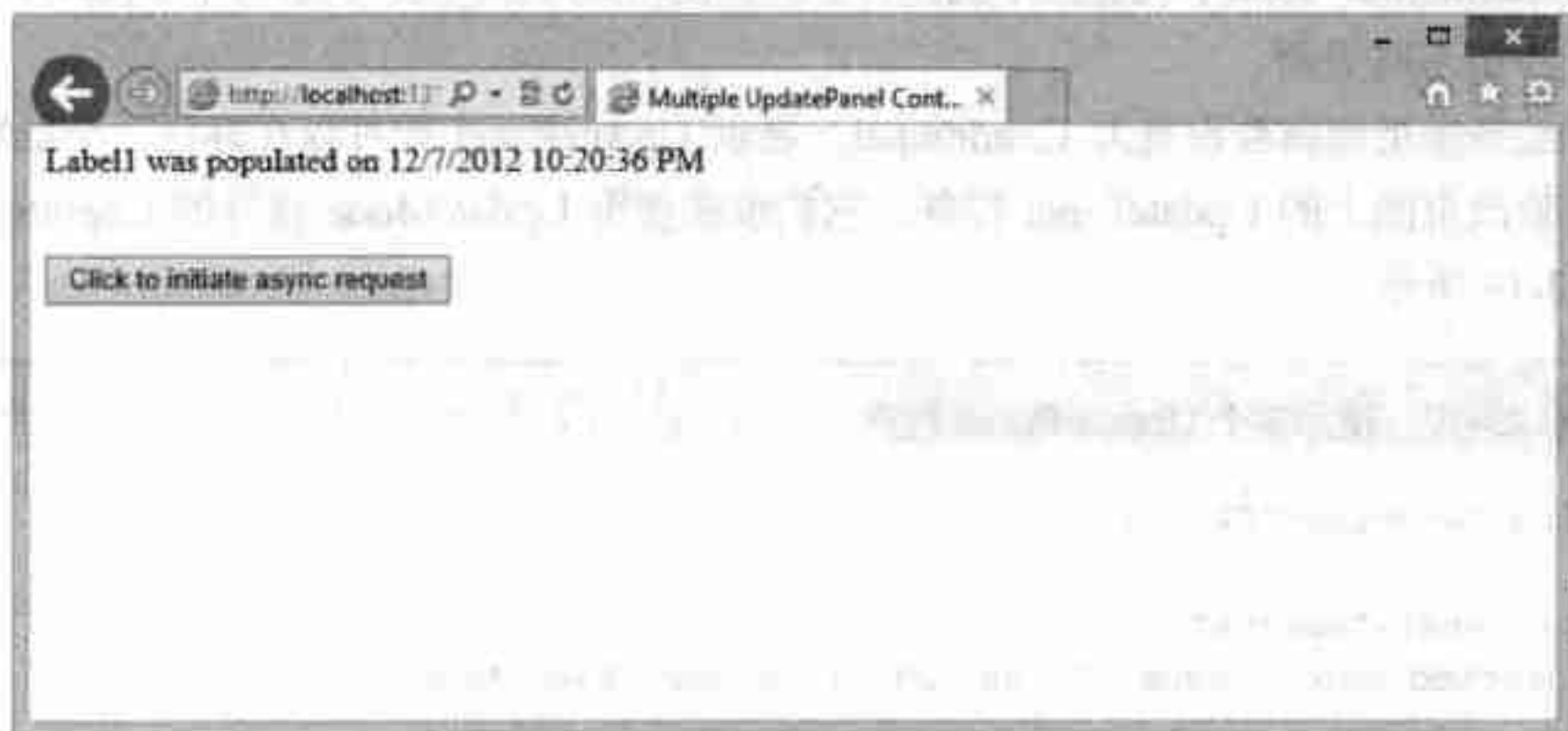


图 23-15

在这个例子中，即使 Button1_Click() 事件试图改变 Label1 和 Label2 的值，也只有右边的 Label 控件 Label1 由异步请求更新，原因是 UpdatePanel2 控件没有满足条件的触发器。

23.6 使用页面历史记录

使用 AJAX 页面存在的一个问题是，终端用户单击浏览器的 Back 按钮时，会删除在当前页面和服务器之间发生的异步请求以及在这些请求之间管理的所有状态。

如果在页面中使用了 AJAX，那么在单击浏览器中的 Back 按钮时，就会转到支持 AJAX 页面之前的页面，而不管在此之前是否已发出了异步页面请求。

另外，如果导航的第一个页面是一系列异步页面请求，那么终端用户不能单击页面上的 Back 按钮，即使当前页面与导航过程中所提供的页面完全不同也是如此。

有了 ASP.NET AJAX 后,开发人员就可以处理应用程序的后退历史。它不像激活对象那样简单,而是需要进行一些编码才能得到我们想要的结果。我们需要执行一个过程,告诉 ASP.NET 页面我们要跟踪什么状态,因为终端用户可以通过浏览器的 Back 和 Forward 按钮来实现导航。

为了说明这一点,创建一个使用 AJAX 的简单页面,这个页面如程序清单 23-20 所示。

程序清单 23-20 创建一个基本的 ASP.NET AJAX 页面

```
<%@ Page Language="C#" %>

<script runat="server">

    protected void Button1_Click(object sender, EventArgs e)
    {
        PopulateFields(TextBox1.Text);
    }

    private void PopulateFields(string InputName)
    {
        if(InputName == null)
        {
            Label1.Text = "Hello there. What is your name?";
        }
        else
        {
            Label1.Text = "Hello there " + HttpUtility.HtmlEncode(InputName);
        }
    }

</script>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>Ajax Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
            <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                <ContentTemplate>
                    <asp:Label ID="Label1" runat="server"
                        Text="Hello there. What is your name?"></asp:Label><br />
                    <br />
                    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
                    <asp:Button ID="Button1" runat="server" Text="Submit Name"
                        OnClick="Button1_Click" />
                </ContentTemplate>
            </asp:UpdatePanel>
        </div>
    </form>
</body>
</html>
```



```

    </form>
</body>
</html>

```

这是一个标准的 ASP.NET AJAX 页面，其中没有什么难懂的地方。在这个例子中，可以使用 AJAX 把姓名输入文本框，之后将这个姓名放在页面的 Label 控件中。如果输入多个姓名，就好像在使用多个页面一样，那么浏览器的导航按钮(Back 和 Forward 按钮)就不会启用，也不能返回在文本框中输入的上一个姓名。浏览器的书签也不能工作，因为它仅存储页面的 URL(永远不会改变)，不存储其状态。

但是，终端用户期望能返回上一个姓名，因此这个新功能将后退历史功能添加到了这类页面中。

如果使用 ASP.NET 3.5 SP1 或更高版本，并查看页面上的 ScriptManager 控件，就会注意到该控件的新属性 EnableHistory，它默认设置为 False。但对于这个例子，应把这个属性设置为 True。

在 ASP.NET AJAX 页面中使用后退历史功能时，不仅要设置 EnableHistory 属性进行上述改动，还需要告诉 ASP.NET 如何记录上一个页面以及当用户导航到上一个页面时要做什么(如果用户单击了几次 Back 按钮，就应记录前几个页面)。

为此，应指定在页面上工作时要记录的历史点和要使用的索引。程序清单 23-21 显示了一个完整的页面示例，该页面会记录这个状态，以便用户使用浏览器的 Back 和 Forward 按钮。

程序清单 23-21 添加历史功能

```

<%@ Page Language="C#" %>

<script runat="server">

    protected void Button1_Click(object sender, EventArgs e)
    {
        PopulateFields(TextBox1.Text);
    }

    private void PopulateFields(string InputName)
    {
        if(InputName == null)
        {
            Label1.Text = "Hello there. What is your name?";
        }
        else
        {
            Label1.Text = "Hello there " + InputName;
        }

        if(ScriptManager1.IsInAsyncPostBack && !ScriptManager1.IsNavigating)
        {
            ScriptManager1.AddHistoryPoint("myIndexPoint", InputName,
                string.Format("Entering name: {0}", InputName));
        }
        else
        {
            TextBox1.Text = InputName;
            Page.Title = string.Format("Entering name: {0}", InputName);
        }
    }

```

```

    }

    protected void ScriptManager1_Navigate(object sender, HistoryEventArgs e)
    {
        PopulateFields(e.State["myIndexPoint"]);
    }

</script>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>Ajax Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server" EnableHistory="True"
                OnNavigate="ScriptManager1_Navigate">
            </asp:ScriptManager>
            <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                <ContentTemplate>
                    <asp:Label ID="Label1" runat="server"
                        Text="Hello there. What is your name?"></asp:Label><br />
                    <br />
                    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
                    <asp:Button ID="Button1" runat="server" Text="Submit Name"
                        OnClick="Button1_Click" />
                </ContentTemplate>
            </asp:UpdatePanel>
        </div>
    </form>
</body>
</html>

```

现在查看单击页面上的按钮时发生的事件，可以发现一些修改用于处理历史点的注册：

```

if (ScriptManager1.IsInAsyncPostBack && !ScriptManager1.IsNavigating)
{
    ScriptManager1.AddHistoryPoint("myIndexPoint", InputName,
        string.Format("Entering name: {0}", InputName));
}
else
{
    TextBox1.Text = InputName;
    Page.Title = string.Format("Entering name: {0}", InputName);
}

```

首先，检查页面上的 **ScriptManager** 控件是否正在执行异步回送，或者是否没有在进行导航。只要满足这两种情况中的任意一种，就使用 **AddHistoryPoint** 调用注册历史点。调用 **AddHistoryPoint** 方法是为了添加键/值对，以定义希望页面记录的索引状态。在这个例子中，键是 **MyIndexPoint** 字符串，值是页面上的 **TextBox** 服务器控件提供的内容。输入参数列表中的最后一个选项是页面使用的标题。这个页面标题会显示在浏览器的页面选项卡(假定有页面选项卡)中，以及 **Back** 和 **Forward**

按钮的选项列表的导航项列表中。

除了使用键/值对调用 `AddHistoryPoint` 方法之外，还可以使用 `NameValueCollection` 对象传送全部的键/值对。

因此，在这个例子中使用了 `myIndexPoint` 键和 `TextBox1` 服务器控件中的值，在页面标题中也使用了这个姓名。

除了上述操作之外，还需要为 `ScriptManager` 创建 `Navigate` 方法，指定在使用其中一个按钮时应执行什么操作。我们将使用这个方法提供当前使用的项的索引：

```
protected void ScriptManager1_Navigate(object sender, HistoryEventArgs e)
{
    PopulateFields(e.State["myIndexPoint"]);
}
```

在这段代码中，`HistoryEventArgs` 允许用户访问已注册的项。

现在运行这个页面，连续输入几个姓名，并单击页面上的按钮，在 **Back** 按钮的选项列表中会提供数据项的历史。还可以导航回这一项(如果需要，还可以前进)，返回以前使用的页面。**Back** 按钮的选项列表如图 23-16 所示。

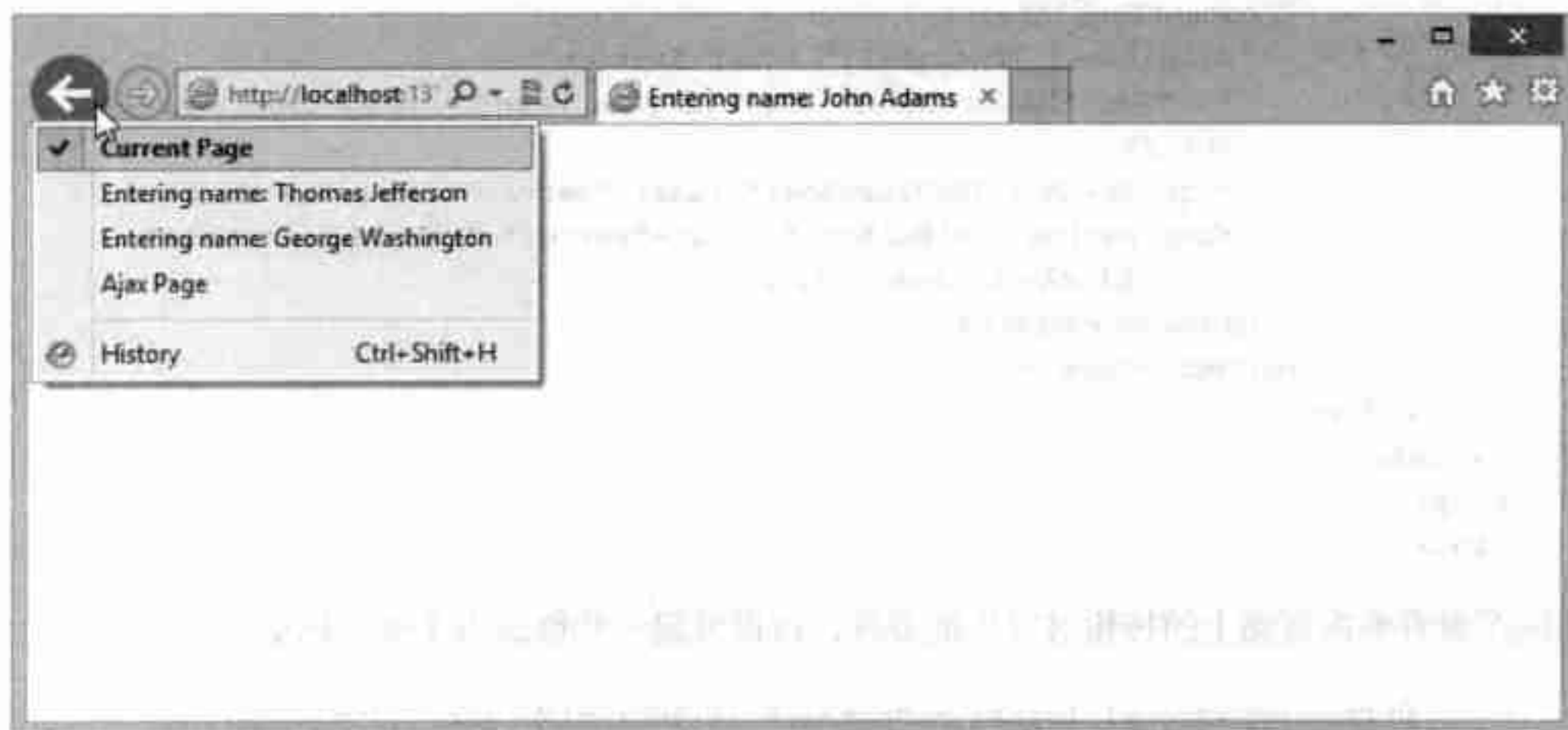


图 23-16

在页面的 URL 中，可以看到如何存储状态。在本例中，可以得到如下 URL：

```
http://localhost:1315/Listing%2023-21.aspx#&&wqMi/gEg0K2EhgXXhZ/
AnMrZ5icHgZTHIkB2OLoKnWV/+5utJPEUUtMfgp9c3dYDB89CnH5S1UPmDYfcy2Q9V91S5tAKv9ulfKHf2h30dkw=
```

注意，在 URL 中加密索引点。这是默认操作，但与 ASP.NET 中的大多数操作一样，也可以改变这个默认操作。为此，可以在 `ScriptManager` 控件中把 `EnableSecureHistoryState` 属性设置为 `False`：

```
<asp:ScriptManager ID="ScriptManager1" runat="server" EnableHistory="True"
    OnNavigate="ScriptManager1_Navigate" EnableSecureHistoryState="False">
```

之后，URL 应如下所示：

```
http://localhost:1315/Listing%2023-21.aspx#&&myIndexPoint=John+Adams
```




有一个 ASP.NET JavaScript API 可以仅使用客户端代码就完成这个任务。HTML5 也引入了一个历史记录 API，但只有最新的浏览器才支持它。

23.7 脚本合并

默认情况下，ASP.NET AJAX 页面有时会为当前查看的页面下载许多不同的脚本。单独下载这些脚本时，页面的整体性能会降低。

造成这种情况的原因是浏览器需要对页面上的每个脚本执行一次请求，这意味着获得全部脚本的时间要多于获得一个较大脚本的时间。另外，分别调用脚本意味着，当前传送的页面的总负载大于调用一个大脚本。

ASP.NET 4.5 可以把多个脚本合并为单个请求和响应。该功能称为脚本合并(script combining)，详见第 28 章。

可以给 ASP.NET 指定要合并的脚本，这需要使用页面上已有的 ScriptManager 控件。技巧是指定需要合并的脚本，因为哪些脚本需要合并并不总是很明显。

为此，在 Internet 上有一个 CodePlex 项目，它提供了一个服务器控件，可以将该控件放在页面上以帮助指定哪些脚本需要合并。这个服务器控件就是 ScriptReferenceProfiler，它提供了页面需要的脚本列表。可以在下述网址上找到这个控件：

<http://aspnet.codeplex.com/releases/view/13356>

这个项目近几年都没有更新，单击如图 23-17 所示的一个链接时，会得到一条错误消息，但除此之外，该项目仍非常便于给 ASP.NET AJAX 的脚本合并指定需要的脚本。

这个项目是一个简单的.dll 文件。在 Visual Studio 的工具箱上右击，从弹出的菜单中选择 Choose Items 命令，就会打开一个对话框，在这个对话框中可以把新控件添加到工具箱中。单击 Browse 按钮，找到刚才下载的.dll 文件。选择该文件后，单击 OK 按钮，就会把它添加到控件集合中。之后，就可以创建一个示例页面，用于合并脚本以获得更佳的性能。这个新页面如程序清单 23-22 所示。

程序清单 23-22 使用 ScriptReferenceProfiler 控件

```
<%@ Page Language="C#" %>

<%@ Register Assembly="ScriptReferenceProfiler"
    Namespace="ScriptReferenceProfiler" TagPrefix="ccl" %>
<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="asp" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = "Hello " + TextBox1.Text + "<br />"
            + "Today is " + TextBox2.Text;
    }
}
```

```

</script>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>Script Combining</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
            <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                <ContentTemplate>
                    What is your name?<br />
                    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
                    <br />
                    <br />
                    What is today's date?<br />
                    <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
                    <asp:CalendarExtender ID="TextBox2_CalendarExtender" runat="server"
                        Enabled="True" TargetControlID="TextBox2">
                    </asp:CalendarExtender>
                    <br />
                    <br />
                    <asp:Button ID="Button1" runat="server" Text="Submit"
                        OnClick="Button1_Click" />
                    <br />
                    <br />
                    <asp:Label ID="Label1" runat="server"></asp:Label>
                </ContentTemplate>
            </asp:UpdatePanel>
            <ccl:ScriptReferenceProfiler ID="ScriptReferenceProfiler1" runat="server" />
        </div>
    </form>
</body>
</html>

```

这是一个简单的页面，它使用 AJAX 以根据终端用户在页面中的输入来更新页面上的 Label 服务器控件。页面上的两个文本框控件要求终端用户输入姓名和当前日期。添加 ScriptReferenceProfiler 控件会在页面的顶部添加两个 @Register 页面指令。

运行这个页面，结果如图 23-17 所示。



图 23-17

在这个页面中，提供了页面上需要调用的脚本列表。现在可以使用页面上的 ScriptManager 控件来指定自己感兴趣的脚本，使用 ASP.NET 的脚本合并功能加载这些脚本。

为此，在浏览器中复制页面中提供的配置脚本，并把这些文本粘贴到 <CompositeScript> 部分的 ScriptManager 控件中，如程序清单 23-23 所示。

程序清单 23-23 使用 ScriptManager 服务器控件合并脚本

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="asp" %>

<script runat="server">
...
</script>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>Script Combining</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
                <CompositeScript>
                    <Scripts>
                        <asp:ScriptReference name="MicrosoftAjax.js"/>
                        <asp:ScriptReference name="MicrosoftAjaxWebForms.js"/>
                        <asp:ScriptReference name="Common.Common.js"
assembly="AjaxControlToolkit, Version=4.1.60919.0, Culture=neutral,
PublicKeyToken=28f01b0e84b6d53e"/>
                        <asp:ScriptReference name="Common.DateTime.js"
assembly="AjaxControlToolkit, Version=4.1.60919.0, Culture=neutral,
PublicKeyToken=28f01b0e84b6d53e"/>
```



```

        <asp:ScriptReference name="Compat.Timer.Timer.js"
assembly="AjaxControlToolkit, Version=4.1.60919.0, Culture=neutral,
PublicKeyToken=28f01b0e84b6d53e"/>
        <asp:ScriptReference name="Animation.Animations.js"
assembly="AjaxControlToolkit, Version=4.1.60919.0, Culture=neutral,
PublicKeyToken=28f01b0e84b6d53e"/>
        <asp:ScriptReference name="ExtenderBase.BaseScripts.js"
assembly="AjaxControlToolkit, Version=4.1.60919.0, Culture=neutral,
PublicKeyToken=28f01b0e84b6d53e"/>
        <asp:ScriptReference name="Animation.AnimationBehavior.js"
assembly="AjaxControlToolkit, Version=4.1.60919.0, Culture=neutral,
PublicKeyToken=28f01b0e84b6d53e"/>
        <asp:ScriptReference name="PopupExtender.PopupBehavior.js"
assembly="AjaxControlToolkit, Version=4.1.60919.0, Culture=neutral,
PublicKeyToken=28f01b0e84b6d53e"/>
        <asp:ScriptReference name="Common.Threading.js"
assembly="AjaxControlToolkit, Version=4.1.60919.0, Culture=neutral,
PublicKeyToken=28f01b0e84b6d53e"/>
        <asp:ScriptReference name="Calendar.CalendarBehavior.js"
assembly="AjaxControlToolkit, Version=4.1.60919.0, Culture=neutral,
PublicKeyToken=28f01b0e84b6d53e"/>
    </Scripts>
</CompositeScript>
</asp:ScriptManager>
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <ContentTemplate>
        What is your name?<br />
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <br /><br />
        What is today's date?<br />
        <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
        <asp:CalendarExtender ID="TextBox2_CalendarExtender" runat="server"
            Enabled="True" TargetControlID="TextBox2">
        </asp:CalendarExtender>
        <br /><br />
        <asp:Button ID="Button1" runat="server" Text="Submit"
            OnClick="Button1_Click" />
        <br /><br />
        <asp:Label ID="Label1" runat="server"></asp:Label>
    </ContentTemplate>
</asp:UpdatePanel>
</div>
</form>
</body>
</html>

```

在<CompositeScript>部分定义所有这些脚本后,就指定要合并这些脚本,并一起发送给页面,因此提高了 ASP.NET 应用程序的整体性能。

ASP.NET AJAX 的未来

微软平台已融合了 JavaScript 库 jQuery(详见第 25 章),所以 ASP.NET AJAX 可能不再增加新功能,但它仍得到了微软的支持,因为它是 ASP.NET 的一部分。只要 ASP.NET 获得支持,使用 ASP.NET AJAX 功能的代码就会获得支持,但是,微软不大可能对这些代码或功能进行根本性的升级。尤其是对于旧的应用程序,不需要废弃 ASP.NET AJAX,但对于新的应用程序,建议至少评估一下 jQuery 是否符合自己的要求。

23.8 本章小结

AJAX 从根本上改变了 Web 应用程序开发的方式。ASP.NET AJAX 是十分杰出的架构，支持 ASP.NET 站点的多个 AJAX 功能，包括部分页面的更新。我们不再需要为每个请求删除整个页面，而后重建，而是可以在终端用户请求页面时重建页面的某些部分。但注意，微软将来仍会支持 ASP.NET AJAX，但不大可能给它增加新功能。

本章介绍了 Visual Studio 2012 默认安装中的 ASP.NET AJAX 的核心基础知识。除此之外，ASP.NET AJAX 还有许多内容，其中之一是 ASP.NET AJAX 控件工具集，这是下一章讨论的主题。

第24章

AJAX 控件工具集

本章要点

- 安装 AJAX 控件工具集
- 在 Web 页面上添加交互功能和动画效果

第 23 章介绍了 ASP.NET AJAX 应用程序。ASP.NET Web 窗体非常依赖这种模块化代码的方式，但为什么用于 ASP.NET AJAX 的服务器控件很少？原因是微软把它们当作开放源代码的项目，而不是把它们融合到 Visual Studio 2012 中。

微软及其社区中的开发人员开发了一系列支持 AJAX 的、可以在 ASP.NET 应用程序中使用的服务器控件。这些控件最初统称为 ASP.NET AJAX 控件工具集。后来微软给这个工具集指定了新名称，使它听起来不像是 ASP.NET 的一部分，它现在称为 AJAX 控件工具集，位于 CodePlex 中，网址是 <http://ajaxcontroltoolkit.codeplex.com/>。AJAX 控件工具集的下载页面如图 24-1 所示。

如前一章所述，ASP.NET AJAX 是构建富 Web 应用程序的基础，它充分利用了浏览器的功能，但是它不具备能真正模糊 Web 应用程序和桌面应用程序之间界限的富 UI 元素。ASP.NET AJAX 包括几个功能强大的 ASP.NET 控件，这些控件可以方便地在已有的应用程序中添加 AJAX 功能，或者把更好的用户体验内置于新的应用程序中。然而，AJAX 控件工具集提供了一些多功能的 ASP.NET AJAX 控件，这些控件使 AJAX Web 应用程序真正达到了实用阶段。此工具集可以方便地改善应用程序的用户界面，使其大大超出用户对传统 Web 应用程序的期望。

这个工具集是一个共享资源的项目，其中的代码来自微软和其他地方的开发人员。使用 ASP.NET AJAX 的大多数开发人员也应该下载这个工具集，以获得它包含的一套额外的控件。在本章前面提及下载 AJAX 控件工具集时，可以下载包含这些控件和扩展程序的已编译 DLL 文件，或者下载源文件和项目文件，自己进行编译。无论采用哪种方式，请确保把此 DLL 文件添加到 Visual Studio 的工具箱中，详细内容稍后介绍。

在这个工具集中，包含了一些具有 AJAX 功能的新控件，以及很多控件扩展程序。可以将扩展程序附加到另一个控件上以扩展这个控件的功能。因为这些控件覆盖了应用程序开发的许多领域，所以本章按照字母顺序介绍它们，控件名是很容易理解的，因此在以后使用本章作为参考时，非常容易定位

到所需的信息。



图 24-1

另外需要注意的是，这个工具集项目还在发展，还会有开发人员继续对此进行改进。本章内容只是更新到撰写本书为止，但是此工具集会定期地添加新的内容。第 25 章将介绍一个真正的开源项目，它提供了类似的功能，但完全独立于微软技术。AJAX 控件工具集与 ASP.NET 紧密相关，因此很容易使用，但不要遗漏下一章的内容，该章介绍了一个维护得很好的 JavaScript 库 jQuery。

24.1 下载和安装 AJAX 控件工具集

由于 ASP.NET AJAX 控件工具集不是 Visual Studio 2012 默认安装的一部分，因此必须自己安装它。AJAX 控件工具集的下载站点提供了两个选项。

CodePlex 站点为 .NET 3.5、4.0 和 4.5 提供了 AJAX 控件工具集版本。本章介绍用于 Visual Studio 2012 和 .NET Framework 4.5 的控件工具集。也可以完全忽略该项目的首页，使用 NuGet 把 AJAX 控件工具集添加到项目中：

```
Install-Package AjaxControlToolkit
```

如果希望在安装之前了解会给项目添加什么内容，建议手工安装。要开始安装，首先从刚才提到的 <http://ajaxcontroltoolkit.codeplex.com/> 站点下载 .zip 文件，并解压缩到计算机上的指定位置。安装步骤如下：

(1) 把控件安装到 Visual Studio 中。把控件添加到 Visual Studio 2012 的工具箱中非常简单。右击工具箱，在弹出的菜单中选择 Add Tab 命令。给新选项卡命名，对于这个例子，将选项卡命名为 Ajax Controls。

(2) 工具箱中有了新选项卡后，右击该选项卡，从弹出的菜单中选择 Choose Items 命令，如图

24-2 所示。之后就会打开 Choose Toolbox Items 对话框。

(3) 在下载的文件中查找 AjaxControlToolkit.dll 文件。找到这个 DLL 文件后, 单击 Open 按钮, Visual Studio 会询问是否要从 Internet 上安装, 如图 24-3 所示。如果同意, Choose Toolbox Items 对话框中便会显示包含在这个 DLL 文件中的控件。这些控件在对话框中突出显示, 并且已经选中, 如图 24-4 所示。

(4) 单击 OK 按钮, ASP.NET AJAX 控件工具集的控件便会添加到 Visual Studio 工具箱中, 结果如图 24-5 所示。

可以在 ASP.NET 应用程序中使用这些添加到工具箱中的 40 多个控件和扩展程序。

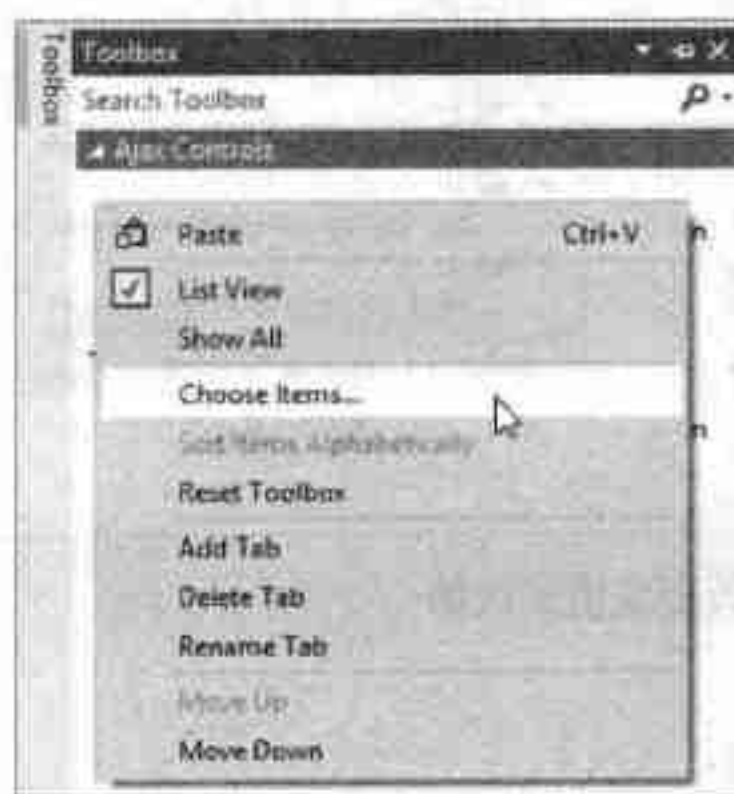


图 24-2



图 24-3

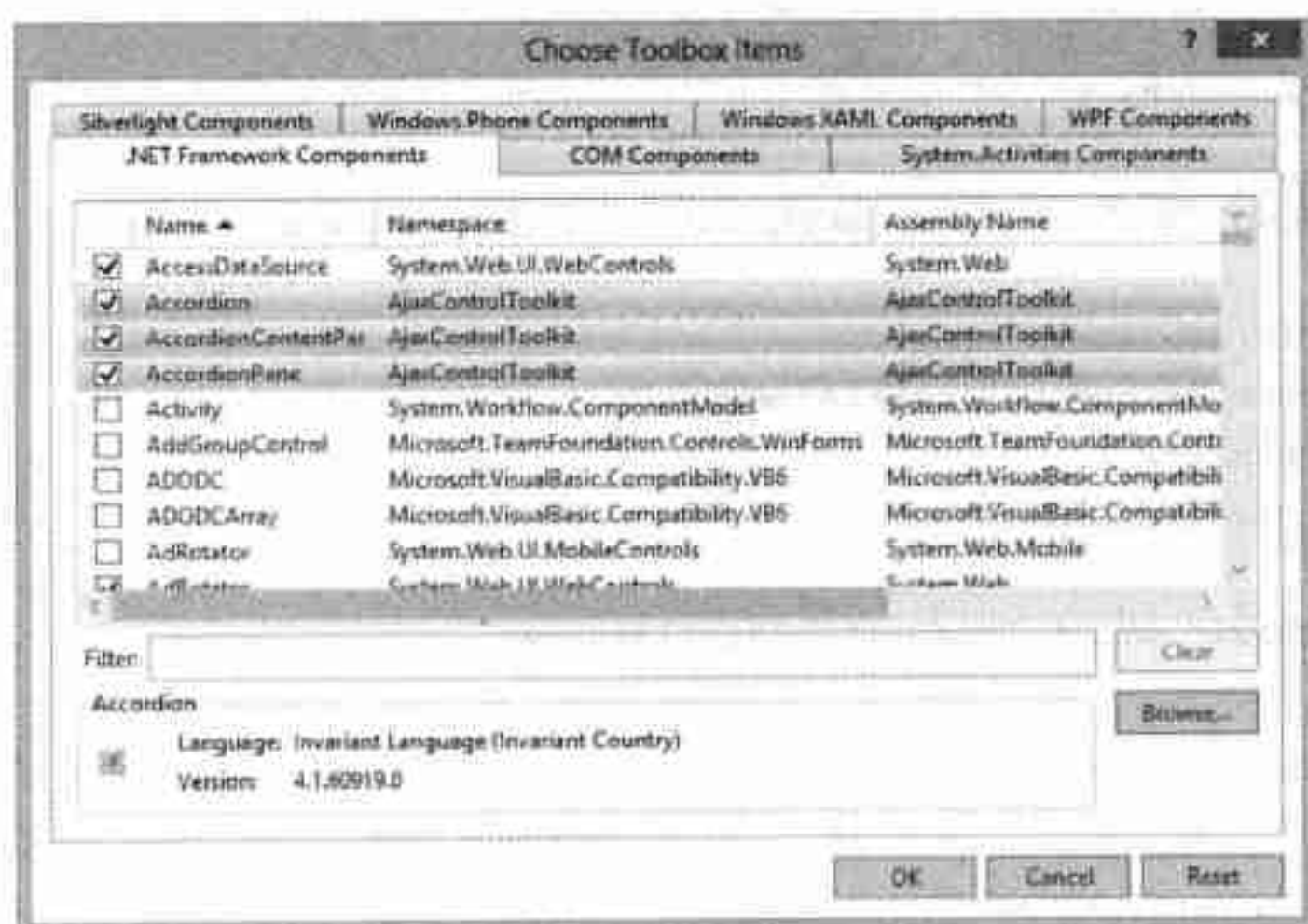


图 24-4



图 24-5

24.2 ASP.NET AJAX 控件

AJAX 控件工具集中有很多控件和扩展程序,如前所述,目前有 40 多个控件和扩展程序。本节介绍这些项,以及如何在 ASP.NET 应用程序中使用它们。

在页面上添加 ASP.NET AJAX 服务器控件时,要注意许多本地化为各种语言的 DLL 都已添加到解决方案的 Bin 文件夹中。所有的资源文件都被组织到该文件夹中的各种语言文件夹内,如图 24-6 所示。

除了将本地化的 DLL 添加到项目中之外,也将 ASP.NET AJAX 控件添加到 ASP.NET 中,添加方式与其他定制服务器控件相同。程序清单 24-1 显示了添加一个 ASP.NET AJAX 控件后的 ASP.NET 页面。

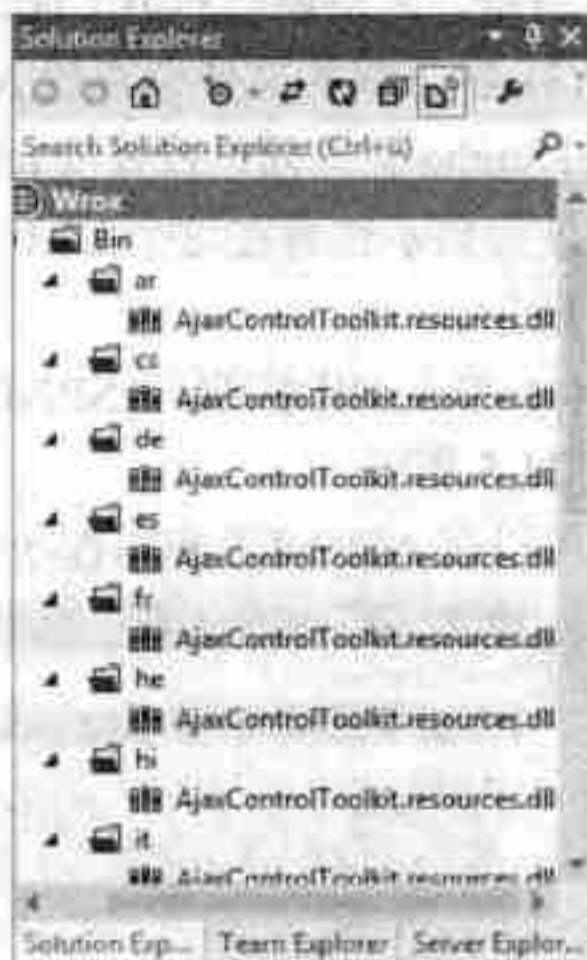


图 24-6

程序清单 24-1 添加一个 ASP.NET AJAX 控件后, ASP.NET 页面发生的改动

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeFile="Default.aspx.cs" Inherits="_Default" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>

<html>
<head runat="server">
    <title>First Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server" />

            <asp:AlwaysVisibleControlExtender
                ID="AlwaysVisibleControlExtender1" runat="server"
                TargetControlID="TextBox1">
            </asp:AlwaysVisibleControlExtender>

            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        </div>
    </form>
</body>
</html>
```

在这个例子中, 使用 `@Register` 指令将 ASP.NET AJAX 控件注册到页面上。这个指令指向 `AjaxControlToolkit` 程序集, 允许所有使用这个程序集的控件引用 `asp` 标记前缀。Visual Studio 2012 默认把引用添加到 `web.config` 文件中, 但为了使代码尽可能自包含, 本章使用 `@Register` 指令把这些引用放在顶部。

AJAX 控件工具集提供的另一个有趣的控件是 `ToolkitScriptManager` (需要手工添加, Visual Studio 不会自动添加)。这个控件派生于 `ScriptManager` 控件, 并在默认情况下扩展以处理脚本合并问题, 使页面比以前更加完善。本章的所有示例都用到了 `ScriptManager` 控件的这一新版本。

24.2.1 AJAX 控件工具集的扩展程序

首先讨论的是 ASP.NET AJAX 控件工具集中的扩展程序, 扩展程序基本上是协助或扩展其他控件的控件。例如, 可以把 ASP.NET 验证控件 (在本书的第 6 章已作介绍) 看作扩展程序控件。在页面上添加一个 `RequiredFieldValidator` 服务器控件, 把它与一个 `TextBox` 控件关联起来, 这就扩展了 `TextBox` 控件, 改变了它的行为。`TextBox` 控件一般只接受文本。现在, 如果没有在该控件中输入数据, 该控件就会引发一个事件返回给 `RequiredFieldValidator` 控件, 其客户端行为由 JavaScript 控制。

AJAX 控件工具集的扩展程序可以很好地完成这项任务。这些控件使用客户端上的额外 JavaScript 以及一些服务器端通信, 扩展了 ASP.NET 服务器控件的行为。

1. AlwaysVisibleControlExtender 控件

在浏览器中展示信息时, 我们也许想保留一些信息, 使它们固定在用户的视图中。屏幕空间是很有限的, 有时需要某些信息总是显示出来, 而不需要用户滚动页面。`AlwaysVisibleControlExtender` 控件可以扩展任何 ASP.NET 控件以满足这个要求。首先, 我们需要使用 `AlwaysVisibleControlExtender` 控件指定 ASP.NET 控件的位置, 当用户滚动页面以浏览其他信息时, 指定的控件总是会保留在当前视图中。当用户滚动屏幕或调整窗口大小时, 这个控件会自动移动, 而且会停留在浏览器窗口可浏览部分中相同的相对位置。

`AlwaysVisibleControlExtender` 控件只包含 6 个属性, 使用方法如程序清单 24-2 所示。

程序清单 24-2 使用 `AlwaysVisibleControlExtender` 控件

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Response.Write("The page has been submitted!");
    }
</script>

<!DOCTYPE html>
<html>
<head runat="server">
```

```

        <title>AlwaysVisibleControlExtender</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server" />
            <asp:AlwaysVisibleControlExtender
                ID="AlwaysVisibleControlExtender1"
                runat="server" TargetControlID="Panel1"
                HorizontalOffset="10"
                HorizontalSide="Right" VerticalOffset="10">
            </asp:AlwaysVisibleControlExtender>
            Form Element :
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
            <br />
            Form Element :
            <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
            <br />

            <!-- Excessive code removed for clarity -->

            Form Element :
            <asp:TextBox ID="TextBox29" runat="server"></asp:TextBox>
            <br />
            Form Element :
            <asp:TextBox ID="TextBox30" runat="server"></asp:TextBox>
            <br />
            <br />
            <asp:Panel ID="Panel1" runat="server">
                <asp:Button ID="Button1" runat="server" Text="Submit"
                    OnClick="Button1_Click" />
                <asp:Button ID="Button2" runat="server" Text="Clear" />
            </asp:Panel>
        </div>
    </form>
</body>
</html>

```

这段代码显示了一个很长的表单，要求终端用户在浏览器上滚动该页面。AlwaysVisibleControlExtender 控件位于这个页面上，要显示该控件，还需要在页面上放置一个 ScriptManager 控件(这与所有 ASP.NET AJAX 控件的要求相同)。

AlwaysVisibleControlExtender1 控件使用 TargetControlID 属性扩展了 Panel1 控件。在这个例子中，TargetControlID 属性的值指向 Panel1 控件。Panel1 控件包含表单的 Submit 按钮。程序清单 24-2 的执行结果如图 24-7 所示。

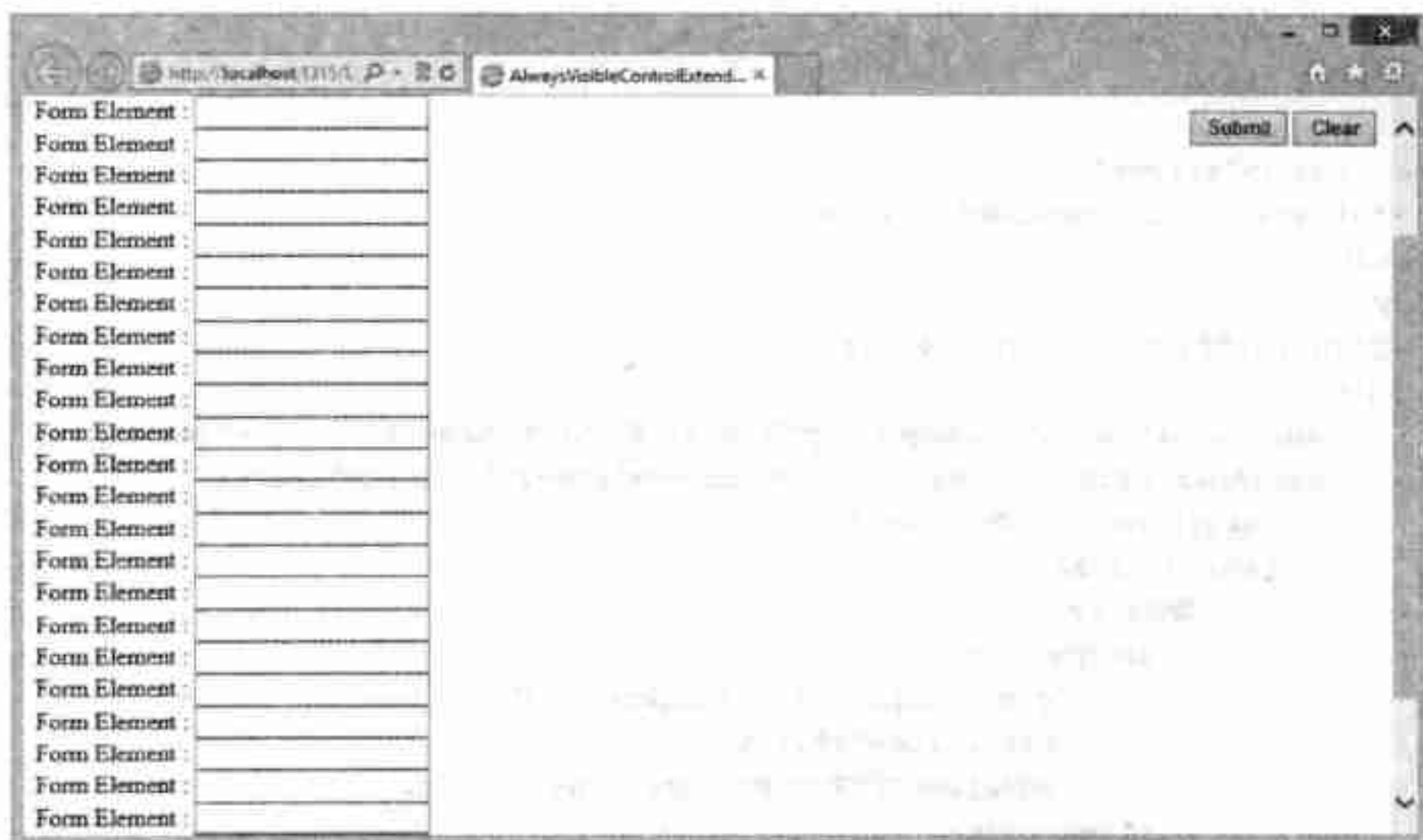


图 24-7

可通过几个控件属性的组合来控制页面上 Submit 按钮和 Clear 按钮的位置。首先，页面上的位置由 HorizontalSide 属性(其值可以是 Center、Left 和 Right)和 VerticalSide 属性(其值可以是 Bottom、Middle 和 Top)确定。接着，使用 HorizontalOffset 和 VerticalOffset 属性在控件的周围添加内边距，在这个例子中，这两个属性都设置为 10 像素。

2. AnimationExtender 控件

AnimationExtender 服务器控件提供了许多功能，它可以为页面上的控件编写流畅的动画。使用这个控件可以做许多工作(远多于本章中介绍的工作)。

这个控件可以根据指定终端用户的触发器(例如按钮单击)让元素在页面上移动。在编写动画时可以使用如下事件：

- OnClick
- OnHoverOver
- OnHoverOut
- OnLoad
- OnMouseOver
- OnMouseOut

创建动画不像许多人认为的那样简单，因为 Visual Studio 对这方面提供的支持很少(例如向导或 IntelliSense)。为了创建第一个动画，程序清单 24-3 说明了如何根据终端用户的操作使元素在页面上淡入淡出。

程序清单 24-3 使用 AnimationExtender 控件淡化背景色

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>
```

```

<!DOCTYPE html>
<html>
<head runat="server">
    <title>AnimationExtender</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server" />
            <asp:AnimationExtender ID="AnimationExtender1" runat="server"
                TargetControlID="Panel1">
                <Animations>
                    <OnClick>
                        <Sequence>
                            <Color PropertyKey="background"
                                StartValue="#999966"
                                EndValue="#FFFFFF" Duration="5.0" />
                        </Sequence>
                    </OnClick>
                </Animations>
            </asp:AnimationExtender>
            <asp:Panel ID="Panel1" runat="server" BorderColor="Black"
                BorderWidth="3px" Font-Bold="True" Width="600px">
                Lorem ipsum dolor sit amet, consectetur adipiscing elit.
                Donec accumsan lorem. Ut consectetur tempus metus.
                Aenean tincidunt venenatis tellus. Suspendisse molestie
                cursus ipsum. Curabitur ut lectus. Nulla ac dolor nec elit
                convallis vulputate. Nullam pharetra pulvinar nunc. Duis
                orci. Phasellus a tortor at nunc mattis congue.
                Vestibulum porta tellus eu orci. Suspendisse quis massa.
                Maecenas varius, erat non ullamcorper nonummy, mauris erat
                eleifend odio, ut gravida nisl neque a ipsum. Vivamus
                facilisis. Cras viverra. Curabitur ut augue eget dolor
                semper posuere. Aenean at magna eu eros tempor
                pharetra. Aenean mauris.
            </asp:Panel>
        </div>
    </form>
</body>
</html>

```

在这个例子中，打开程序清单 24-3 所示的页面时会看到已使用了一个 `AnimationExtender` 控件，该控件位于 `Panel1` 控件的外面。使用 `TargetControlID` 属性建立这个连接。

如前所述，在输入包含在 `AnimationExtender` 控件中的代码时，不能使用 `IntelliSense`。因此，需要为要创建的动画查看文档。对于前面的例子，使用 `<OnClick>` 元素定义了一系列单击控件时需要引发的事件。对于这个例子，仅在 `<Sequence>` 元素中定义了如下动画——将颜色变为元素的背景色。其中 `<Color>` 元素指定，CSS 属性 `StartValue` 开始时应是 `#999966` 颜色，并且在 5 秒内(使用 `Duration` 属性定义)要完全变为 `#FFFFFF` 颜色。

打开这个页面，单击 `Panel` 元素，就会看到颜色在 5 秒内从指定的起始颜色变成最终颜色。

3. AutoCompleteExtender 控件

终端用户在文本框中输入搜索关键字后, AutoCompleteExtender 控件可以帮助他们找到需要的信息。目前许多搜索站点上都有这个功能, 一旦终端用户开始在文本框中输入字符, 就会从数据存储中获得与目前所输入内容匹配的结果, 这表现为一个直接位于正在输入搜索关键字的文本框下方的下拉列表。

为了完成类似的任务, 可以创建一个仅包含 ScriptManager 控件、AutoCompleteExtender 控件和 TextBox 控件的新页面。该页面的 ASP.NET 部分如程序清单 24-4 所示(本章下载代码中的 AutoCompleteExtender.aspx)。

程序清单 24-4 ASP.NET 页面

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeFile="AutoComplete.aspx.cs" Inherits="AutoComplete" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>AutoComplete</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>
            <asp:AutoCompleteExtender ID="AutoCompleteExtender1"
                runat="server" TargetControlID="TextBox1"
                ServiceMethod="GetCompletionList" UseContextKey="True">
            </asp:AutoCompleteExtender>
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        </div>
    </form>
</body>
</html>
```

与其他 ASP.NET AJAX 控件一样, 也可以使用 TargetControlID 属性扩展 TextBox 控件。第一次把这些控件添加到页面上时, 并没有在 AutoCompleteExtender 控件中定义 ServiceMethod 属性。使用 Visual Studio 2012 可以在设计界面上让该架构建立一个服务方法, 把扩展程序控件与这个方法关联起来。展开 TextBox 控件的智能标记后, 从弹出的菜单中选择 Add AutoComplete page method 选项, 如图 24-8 所示。

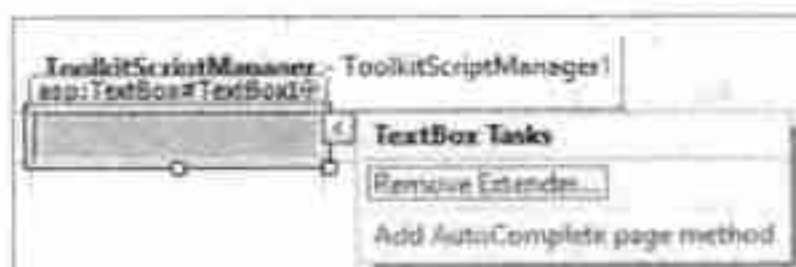


图 24-8

这个操作会在页面的隐藏代码中创建一个服务方法。程序清单 24-5(本章下载代码中的 AutoCompleteExtender.aspx.cs)列出了完成这个方法以从 Northwind 数据库中提取公司名称所需的步骤。



有关下载和使用 Northwind 数据库及 Visual Studio 2012 的指令, 可以访问 <http://msdn.microsoft.com/en-us/library/8b6y4c7s.aspx>。本章使用的 Northwind 数据库可以从 www.wrox.com/go/SQLServer2012DataSets 下载。

程序清单 24-5 为自动完成功能建立服务方法的隐藏代码

```
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
public partial class AutoComplete : System.Web.UI.Page
{
    [System.Web.Services.WebMethodAttribute(),
    System.Web.Script.Services.ScriptMethodAttribute()]
    public static string[] GetCompletionList(string prefixText,
        int count, string contextKey)
    {
        SqlConnection conn;
        SqlCommand cmd;
        string cmdString =
            "Select CompanyName from Customers WHERE CompanyName LIKE " +
            "@prefixText";
        conn = new
            SqlConnection(@"Data Source=. \SQLEXPRESS;
            AttachDbFilename=|DataDirectory|\NORTHWND.MDF;
            Integrated Security=True;User Instance=True");
        // Put this string on one line in your code
        cmd = new SqlCommand(cmdString, conn);
        cmd.Parameters.AddWithValue("@prefixText", prefixText + "%");
        conn.Open();

        SqlDataReader myReader;
        List<string> returnData = new List<string>();

        myReader = cmd.ExecuteReader(CommandBehavior.CloseConnection);

        while (myReader.Read())
        {
            returnData.Add(myReader["CompanyName"].ToString());
        }

        return returnData.ToArray();
    }
}
```

运行这个页面, 在文本框中输入字符 ant, 就会调用 GetCompletionList 方法, 并且传送这些字

符。可以通过 `prefixText` 参数提取这些字符(也可以使用 `count` 参数, 该参数的默认值为 10)。使用 `prefixText` 值调用 Northwind 数据库, 该值会返回给 `TextBox1` 控件。最后, 得到一个下拉列表, 其中的项的前 3 个字符匹配在文本框中输入的字符, 如图 24-9 所示。

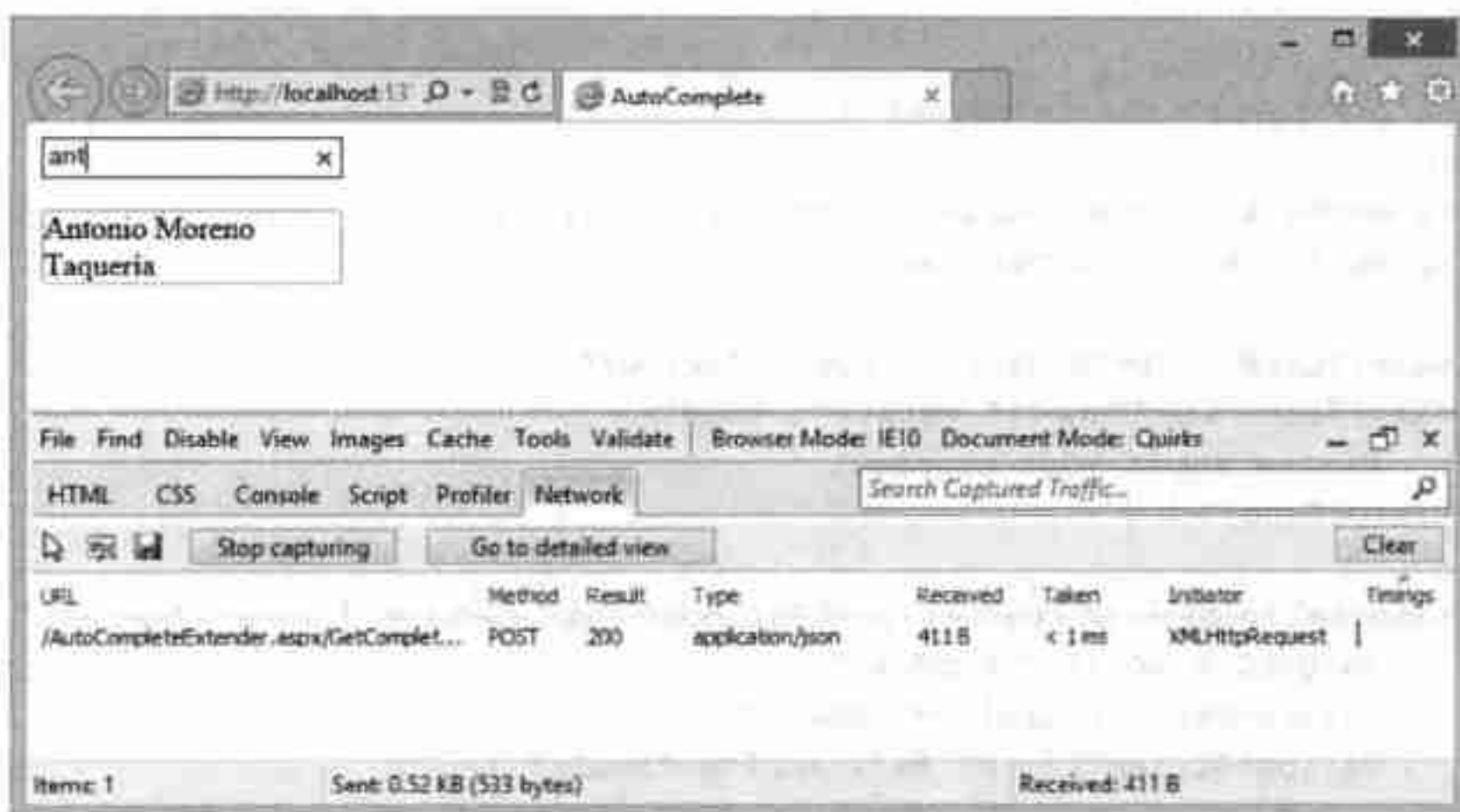


图 24-9

第一次生成结果后, 最好高速缓存它们。通过 `EnableCaching` 属性(该属性的默认值为 `true`)控制高速缓存。还可以修改自动完成的下拉列表的样式, 指定显示多少元素, 以及设置其他方面。比较重要的一点是不一定要调用控件所在页面上的方法, 如本章中的示例所示, 而是可以调用另一个页面上的服务器端方法或 Web 方法。

4. BalloonPopupExtender 控件

这是开发人员强烈呼吁要在 AJAX 控件工具集中添加的一个控件, 它具备给元素添加类似气球 UI 的功能。最后, 在 AJAX 控件工具集于 2011 年 11 月发布的版本中, AJAX 控件工具集团队添加了这个控件。

`BalloonPopupExtender` 控件给 `TargetControlID` 中引用的元素添加了一个“气球”。气球的实际内容位于页面的一个随意控件中。该扩展程序的 `BalloonPopupControlID` 属性包含该控件的 ID。

该扩展程序支持几个内置的样式(`BalloonStyle` 属性)和大小(`BalloonSize` 属性), 还可以定制样式。程序清单 24-6 是为 `TextBox` 控件提供云朵外观的气球的一个例子。

程序清单 24-6 为 TextBox 控件添加气球

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="asp" %>

<!DOCTYPE html>

<script runat="server">

</script>
```

```

<!DOCTYPE html>
<html>
<head runat="server">
    <title>BalloonPopupExtender</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>

            <asp:TextBox ID="TextBox1" runat="server" />
            <asp:Panel ID="Panel1" runat="server">
                Please enter your country.
            </asp:Panel>

            <asp:BalloonPopupExtender ID="BalloonPopupExtender1" runat="server"
                TargetControlID="TextBox1"
                BalloonPopupControlID="Panel1"
                BalloonStyle="Cloud" BalloonSize="Small" />

        </div>
    </form>
</body>
</html>

```

单击文本框，就会显示气球，如图 24-10 所示。

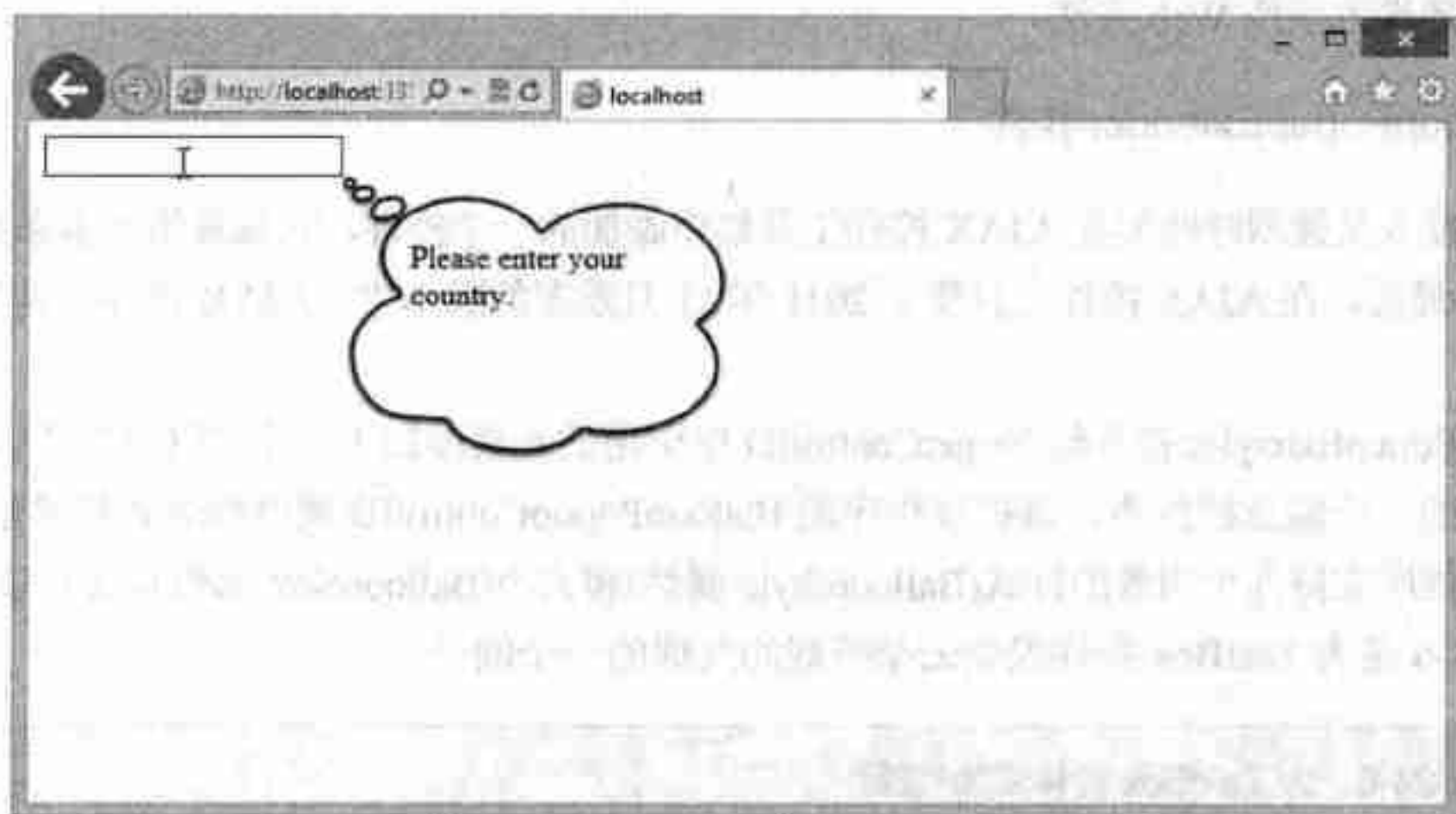


图 24-10

5. CalendarExtender 控件

选择日期是许多应用程序的常见要求，也是阻碍提交表单的最常见原因之一。终端用户常常因为确定表单需要的日期格式而使应用程序运行变慢。CalendarExtender 控件可以使终端用户更容易从表单中选择日期。CalendarExtender 控件能关联一个文本框，弹出一个日历，供用户选择日期。在默认情况下，在文本框获得焦点时就会显示日历。但是，如果把 PopupButtonID 属性设置为另一个控件的 ID，在该控件获得焦点时就会显示日历。

在表单中选择日期的最快捷方式是使用日历，用户可以浏览日历，找到需要的日期，接着把日历日期转换为文本框中的文本日期格式。通过几个关键属性可以很容易地使用 CalendarExtender 控件：TargetControlID 属性指向接收所选日期的文本框，Format 属性指定文本框中日期的字符串格式。CalendarExtender 控件提供了这类操作所需的所有客户端代码。程序清单 24-7 是在 TextBox 控件中提供日历控件的一个例子。

程序清单 24-7 在 TextBox 控件中使用日历控件

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>CalendarExtender</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>
            <asp:CalendarExtender ID="CalendarExtender1" runat="server"
                TargetControlID="TextBox1">
            </asp:CalendarExtender>
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        </div>
    </form>
</body>
</html>
```

运行这个页面，结果是在页面上仅显示一个文本框，该文本框看起来与其他文本框没有任何区别。但是，当终端用户单击文本框的内部时，在其下方就会显示一个日历，如图 24-10 所示。接着，当终端用户从日历中选择日期时，日期就作为文本显示在文本框中，如图 24-11 所示。



图 24-11

这个控件的属性有 FirstDayOfWeek 和 PopupPosition(PopupPosition 属性的值可以是 BottomLeft、

BottomRight、TopLeft 和 TopRight)。也可以改变日历在客户端的启动方式。一些站点在文本框的旁边提供了日历按钮，只有当终端用户单击该按钮时才会弹出日历选项。如果这就是要在页面上进行的操作，就应使用 PopupButtonID 属性，它必须指向所使用的图像或按钮的 ID。

6. CollapsiblePanelExtender 控件

CollapsiblePanelExtender 服务器控件可以把一个控件折叠到另一个控件中。使用两个 Panel 服务器控件时，可以提供一种很好的方式来控制 ASP.NET 页面的区域问题。

CollapsiblePanelExtender 控件类似于 Accordion 控件(本章后面将介绍)，但它不指向多个内容区域。ASP.NET 面板控件会根据用户与给定控件之间的交互操作在视图中显示或隐藏。这就允许隐藏用户不需要总是看到的信息。在单击 ExpandControlID 时，TargetControlID 就会显示；而在单击 CollapseControlID 时，TargetControlID 就会隐藏。另外，如果 AutoCollapse 和 AutoExpand 属性设置为 True，就会根据用户是否把光标停放在控件上来显示或隐藏 TargetControlID。

程序清单 24-8 演示了 CollapsiblePanelExtender 控件的用法，它在折叠面板时把面板大小设置为 0，而在展开面板时把面板大小设置为 300 像素。另一个面板用作展开和折叠面板的选择器。另外还包含一个标签，它被指定为 TextLabelID。该 Label 控件的值根据当前的状态在 ExpandedText 和 CollapsedText 之间改变。

程序清单 24-8 将 CollapsiblePanelExtender 控件用于两个 Panel 控件

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head id="Head1" runat="server">
    <title>CollapsiblePanelExtender</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>
            <asp:Panel ID="Panel1" runat="server" BackColor="#000066"
                ForeColor="White">
                <asp:Label ID="Label2" runat="server"
                    Text="This is my title"></asp:Label>
                <asp:Label ID="Label1" runat="server"></asp:Label>
            </asp:Panel>
            <asp:Panel ID="Panel2" runat="server" Style="overflow: hidden;" Height="0">
                Lorem ipsum dolor sit amet, consectetur adipiscing elit.
                Donec accumsan lorem. Ut consectetur tempus metus.
                Aenean tincidunt venenatis tellus. Suspendisse molestie
                cursus ipsum. Curabitur ut lectus. Nulla ac dolor nec elit
                convallis vulputate. Nullam pharetra pulvinar nunc. Duis
                orci. Phasellus a tortor at nunc mattis congue. Vestibulum
                porta tellus eu orci. Suspendisse quis massa.
                Maecenas varius, erat non ullamcorper nonummy, mauris erat
```



```

        eleifend odio, ut gravida nisl neque a ipsum. Vivamus
        facilisis. Cras viverra. Curabitur ut augue eget dolor
        semper posuere. Aenean at magna eu eros tempor pharetra.
        Aenean mauris.
    </asp:Panel>
    <asp:CollapsiblePanelExtender ID="CollapsiblePanelExtender1"
        runat="server"
        TargetControlID="Panel2" Collapsed="true"
        ExpandControlID="Panel1"
        CollapseControlID="Panel1"
        CollapsedSize="1"
        ExpandedSize="300" CollapsedText="[Click to expand]"
        ExpandedText="[Click to collapse]" TextLabelID="Label1"
        SuppressPostBack="true">
    </asp:CollapsiblePanelExtender>
</div>
</form>
</body>

</html>

```

在这个例子中，第一次打开页面时只能看到 Panel1 控件的内容，即标题面板。在默认情况下，通常会看到两个控件，但因为 Panel2 控件的 Collapsed 属性设置为 True，所以只能看到 Panel1 控件。单击 Panel1 控件，就会显示 Panel2 控件的内容。实际上，内容会从 Panel1 控件滑出。把这两个控件关联在一起以执行这个操作是通过使用 CollapsiblePanelExtender 控件完成的，该控件的 TargetControlID 属性被赋予第二个 Panel 控件 Panel2，因为 Panel2 控件需要在页面上展开。ExpandControlID 属性用于指定启动该展开操作的控件。

展开 Panel2 控件后，当终端用户单击该控件时，其中的内容就会消失，改为显示 Panel1 控件的内容。实现该操作需要把 CollapseControlID 属性设置为 Panel2。

CollapsiblePanelExtender 控件有许多属性，可以细调展开和折叠的方式。例如，还可以把 Label1 控件设置为这个过程的启动器，甚至根据 Panel2 控件是展开还是折叠来改变 Label 控件的文本。

7. ColorPickerExtender 控件

从终端用户获取的困难数据点之一是颜色。如果只使用文本，那么这个特殊的数据点很难定义。如果对颜色有很多种选择，那么终端用户如何定义蓝色的阴影呢？为此，使用 ColorPickerExtender 可快速、方便地对一些诸如 TextBox 这样的控件进行扩展，从而可以轻易地解决这个问题。程序清单 24-9 显示了实现这项任务的方法。

程序清单 24-9 使用 ColorPickerExtender 控件选择颜色

```

<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">

```



```

<title>ColorPickerExtender</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
      </asp:ToolkitScriptManager>
      <br />
      Pick your favorite color:<br />
      <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
      <asp:ColorPickerExtender ID="ColorPickerExtender1"
        runat="server" TargetControlID="TextBox1">
      </asp:ColorPickerExtender>
    </div>
  </form>
</body>
</html>

```

当打开这个页面时，页面上只有一个 TextBox 服务器控件。在这个 TextBox 控件上单击，将弹出颜色选择器，如图 24-12 所示。

之后，终端用户可以在颜色选项上滚动，当选中其中一种颜色时，选择器会自动消失，并会在 TextBox 控件中以十六进制代码的形式显示出用户的选择结果，如图 24-13 所示。

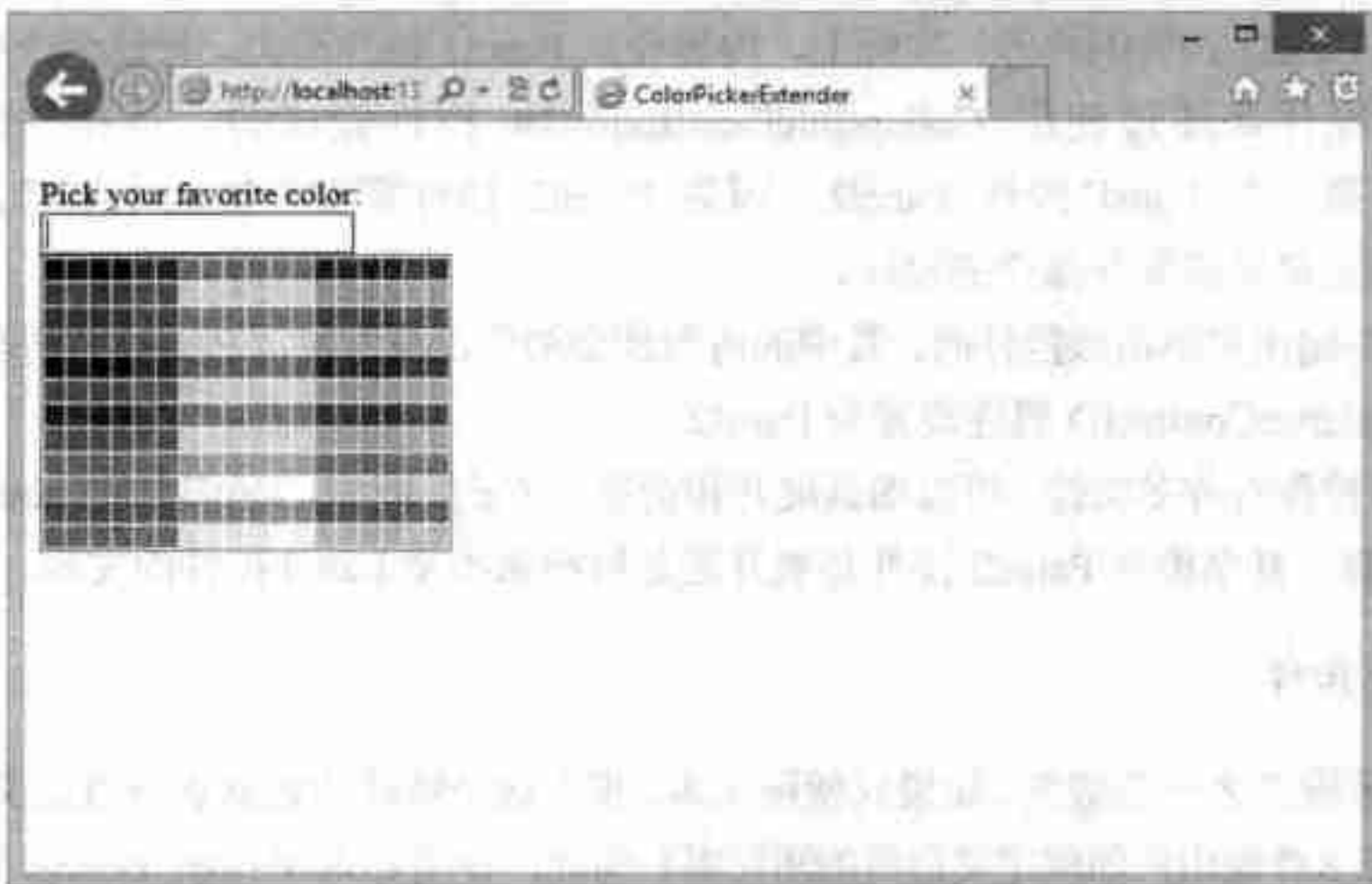


图 24-12

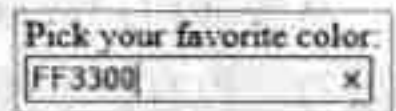


图 24-13

8. ConfirmButtonExtender 控件和 ModalPopupExtender 控件

在允许终端用户通过浏览器应用程序删除数据之前，通常都要终端用户确认该操作。ConfirmButtonExtender 控件允许给终端用户提出此确认问题，并再次确认他们希望执行的操作。程序清单 24-10 说明了如何使用这个控件。

程序清单 24-10 使用 ConfirmButtonExtender 控件再次确认用户的操作

```

<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"

```

```

    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>ConfirmButtonExtender</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>
            <asp:ConfirmButtonExtender ID="ConfirmButtonExtender1"
                runat="server" TargetControlID="Button1"
                ConfirmText="Are you sure you wanted to click this button?">
            </asp:ConfirmButtonExtender>
            <asp:Button ID="Button1" runat="server" Text="Button" />
        </div>
    </form>
</body>
</html>

```

在这个例子中, `ConfirmButtonExtender` 控件扩展了 `Button1` 服务器控件, 使用 `ConfirmText` 属性定义的文本添加了一个确认对话框, 这个对话框如图 24-14 所示。

如果终端用户在这个对话框中单击 `OK` 按钮, 页面就按正常方式执行, 就好像该对话框从来没有打开过一样。但是, 如果用户单击 `Cancel` 按钮, 默认情况下该对话框就会消失, 并且不提交表单(就好像根本没有单击按钮一样)。此时, 也可以捕获单击的 `Cancel` 按钮, 使用 `OnClientClick` 事件并给它指定一个客户端 `JavaScript` 函数的值, 以此来执行客户端操作。

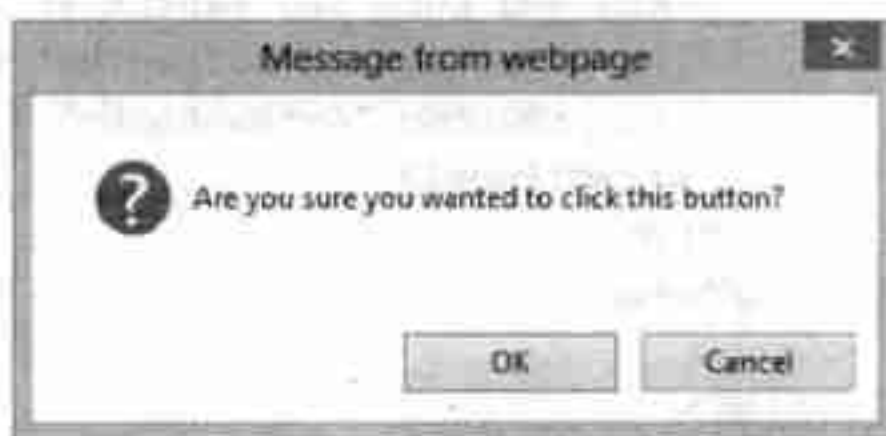


图 24-14

除了使用浏览器的模态对话框之外, 还可以创建自己的对话框并用作确认窗体。为此, 需要使用 `ModalPopupExtender` 服务器控件。 `ModalPopupExtender` 控件指向用于确认的另一个控件。

`ModalPopupExtender` 控件禁止用户与底层页面交互, 除非用户完成了一个模态对话框。它非常类似于 `HoverMenuExtender` 控件, 只是用户必须使用 `PopupControlID` 指定的控件, 之后才能继续执行。该控件的属性有 `OkControlID` 和 `CancelControlID`, 以及根据用户的选择来运行的 `OnOkScript` 和 `OnCancelScript` 属性。程序清单 24-11 说明了如何使用这个控件。

程序清单 24-11 使用 `ModalPopupExtender` 控件创建自己的确认窗体

```

<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>

```

```

<head runat="server">
    <title>ConfirmButtonExtender</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>
            <asp:ConfirmButtonExtender ID="ConfirmButtonExtender1"
                runat="server" TargetControlID="Button1"
                DisplayModalPopupID="ModalPopupExtender1">
            </asp:ConfirmButtonExtender>
            <asp:ModalPopupExtender ID="ModalPopupExtender1" runat="server"
                CancelControlID="ButtonNo" OkControlID="ButtonYes"
                PopupControlID="Panel1"
                TargetControlID="Button1">
            </asp:ModalPopupExtender>
            <asp:Button ID="Button1" runat="server" Text="Button" />
            <asp:Panel ID="Panel1" runat="server"
                style="display:none; background-color:White; width:200;
                border-width:2px; border-color:Black; border-style:solid;
                padding:20px;">
                Are you sure you wanted to click this button?<br />
                <asp:Button ID="ButtonYes" runat="server" Text="Yes" />
                <asp:Button ID="ButtonNo" runat="server" Text="No" />
            </asp:Panel>
        </div>
    </form>
</body>
</html>

```

在这个例子中，ConfirmButtonExtender 控件仍然指向页面上的 Button1 服务器控件，这表示单击该按钮时，ConfirmButtonExtender 控件会执行操作。这里没有使用 ConfirmText 属性，而是使用了 DisplayModalPopupID 属性。此时，该属性指向 ModalPopupExtender1 控件，即另一个扩展程序控件。

ModalPopupExtender 控件使用 PopupControlID 属性引用页面上的 Panel1 控件。这个 Panel 控件的内容用于按钮单击的确认。为此，ModalPopupExtender 控件必须给 OkControlID 和 CancelControlID 属性指定值。在这个例子中，这两个属性指向包含在 Panel 控件中的两个 Button 控件。运行这个页面，结果如图 24-15 所示。

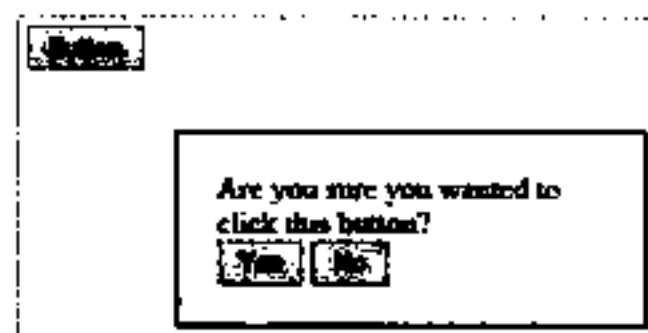


图 24-15

9. DragPanelExtender 控件

DragPanelExtender 控件用于定义终端用户可以在其上移动元素的页面区域。实际上，终端用户可以在浏览器页面的任意位置拖放元素。

为了使用该功能，必须执行几个操作。首先在页面上创建一块足够大的<div>区域，可以将元素拖动到该区域中。之后，需要指定把什么对象用作拖动句柄，并指定另一个跟随拖动句柄移动的控件。在程序清单 24-12 所示的例子中，把 Label 控件用作拖动句柄，而 Panel2 控件是要在屏幕上拖

动的内容。

程序清单 24-12 在页面上拖动 Panel 控件

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>DragPanel control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>
            <div style="height: 600px;">
                <asp:DragPanelExtender ID="DragPanelExtender1"
                    runat="server"
                    DragHandleID="Label1" TargetControlID="Panel1">
                </asp:DragPanelExtender>
                <asp:Panel ID="Panel1" runat="server" Width="450px">
                    <asp:Label ID="Label1" runat="server"
                        Text="Drag this Label control to move the control"
                        BackColor="DarkBlue" ForeColor="White"></asp:Label>
                    <asp:Panel ID="Panel2" runat="server" Width="450px">
                        Lorem ipsum dolor sit amet, consectetur adipiscing elit.
                        Donec accumsan lorem. Ut consectetur tempus metus.
                        Aenean tincidunt venenatis tellus. Suspendisse molestie
                        cursus ipsum. Curabitur ut lectus. Nulla ac dolor nec elit
                        convallis vulputate. Nullam pharetra pulvinar nunc. Duis
                        orci. Phasellus a tortor at nunc mattis congue.
                        Vestibulum porta tellus eu orci. Suspendisse quis massa.
                        Maecenas varius, erat non ullamcorper nonummy, mauris erat
                        eleifend odio, ut gravida nisl neque a ipsum. Vivamus
                        facilisis. Cras viverra. Curabitur
                        ut augue eget dolor semper posuere. Aenean at magna eu eros
                        tempor pharetra. Aenean mauris.
                    </asp:Panel>
                </asp:Panel>
            </div>
        </div>
    </form>
</body>
</html>
```

这个例子创建了一个高度为 600 像素的<div>元素。在这块定义的区域，该例使用 DragPanel-Extender 控件的 TargetControlID 属性把 Panel1 控件指定为目标。

在 Panel1 控件中有另外两个服务器控件：Label 控件和另一个 Panel 控件。该例使用 DragPanel-

Extender 控件的 DragHandleID 属性把 Label 控件指定为拖动句柄。有了这段代码后,就可以在浏览器窗口中拖动 Panel1 控件。如图 24-16 显示,将 Label 控件用作拖动句柄,在 Panel 控件上拖动。

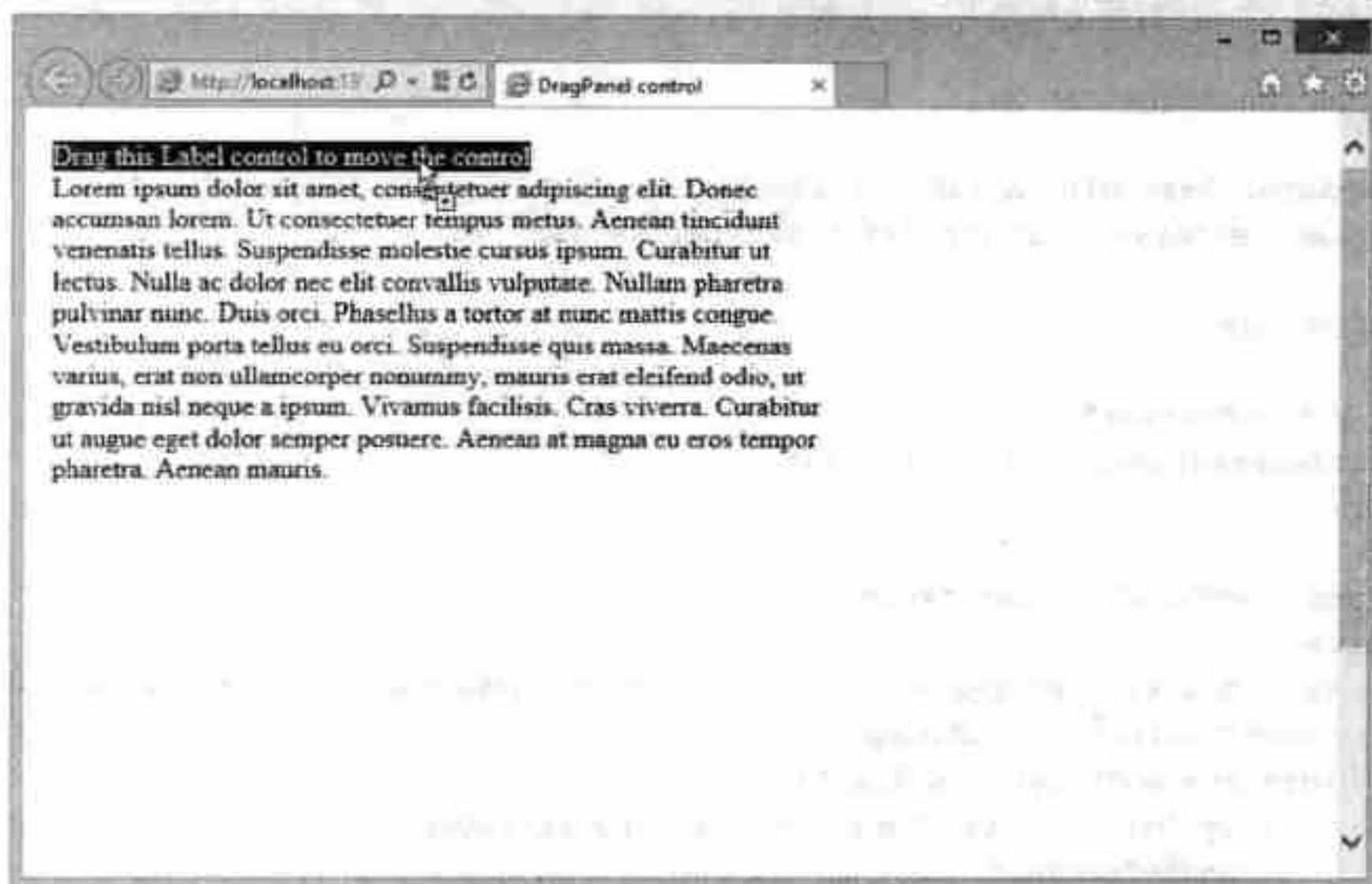


图 24-16

10. DropDownExtender 控件

DropDownExtender 控件允许用户提取任何控件,并在它的下面提供下拉选项列表以供选择。该控件提供了与一般下拉列表控件不同的架构,因为它允许进行非常广泛的定制。程序清单 24-13 说明了如何把图像用作下拉选项列表的启动器。

程序清单 24-13 将 Image 控件用作下拉选项列表的启动器

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        Image1.ImageUrl = "Images/ToolkitLogo.jpg";
    }

    protected void Option_Click(object sender, EventArgs e)
    {
        Image1.ImageUrl = "Images/" + ((LinkButton)sender).Text + ".jpg";
    }
</script>

<!DOCTYPE html>
<html>
<head runat="server">
```

```

<title>DropDownExtender Control</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
      </asp:ToolkitScriptManager>
      <asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
          <asp:DropDownExtender ID="DropDownExtender1" runat="server"
            DropDownControlID="Panel1" TargetControlID="Image1">
          </asp:DropDownExtender>
          <asp:Image ID="Image1" runat="server">
          </asp:Image>
          <asp:Panel ID="Panel1" runat="server" Height="50px"
            Width="125px">
            <asp:LinkButton ID="Option1" runat="server"
              OnClick="Option_Click">ToolkitLogo</asp:LinkButton>
            <asp:LinkButton ID="Option2" runat="server"
              OnClick="Option_Click">ToolkitLogo1</asp:LinkButton>
            <asp:LinkButton ID="Option3" runat="server"
              OnClick="Option_Click">ToolkitLogo2</asp:LinkButton>
          </asp:Panel>
        </ContentTemplate>
      </asp:UpdatePanel>
    </div>
  </form>
</body>
</html>

```

在这个例子中，将 DropDownExtender 控件与 Image 控件关联起来，在 Page_Load 事件中，Image 控件会显示一幅指定的图像。DropDownExtender 控件需要指定两个特殊的属性。第一个是 TargetControlID 属性，它定义了成为下拉列表的启动器的控件。第二个是 DropDownControlID 属性，它定义了页面上用作下拉项的元素，这些下拉项显示在控件的下面。在这个例子中，DropDownControlID 属性指向带 3 个 LinkButton 控件的 Panel 控件。

每个 LinkButton 控件都表示显示在页面上的一幅图像。选择其中一个选项，就会通过 Option_Click 方法改变要选择的图像。运行这个页面，结果如图 24-17 所示。

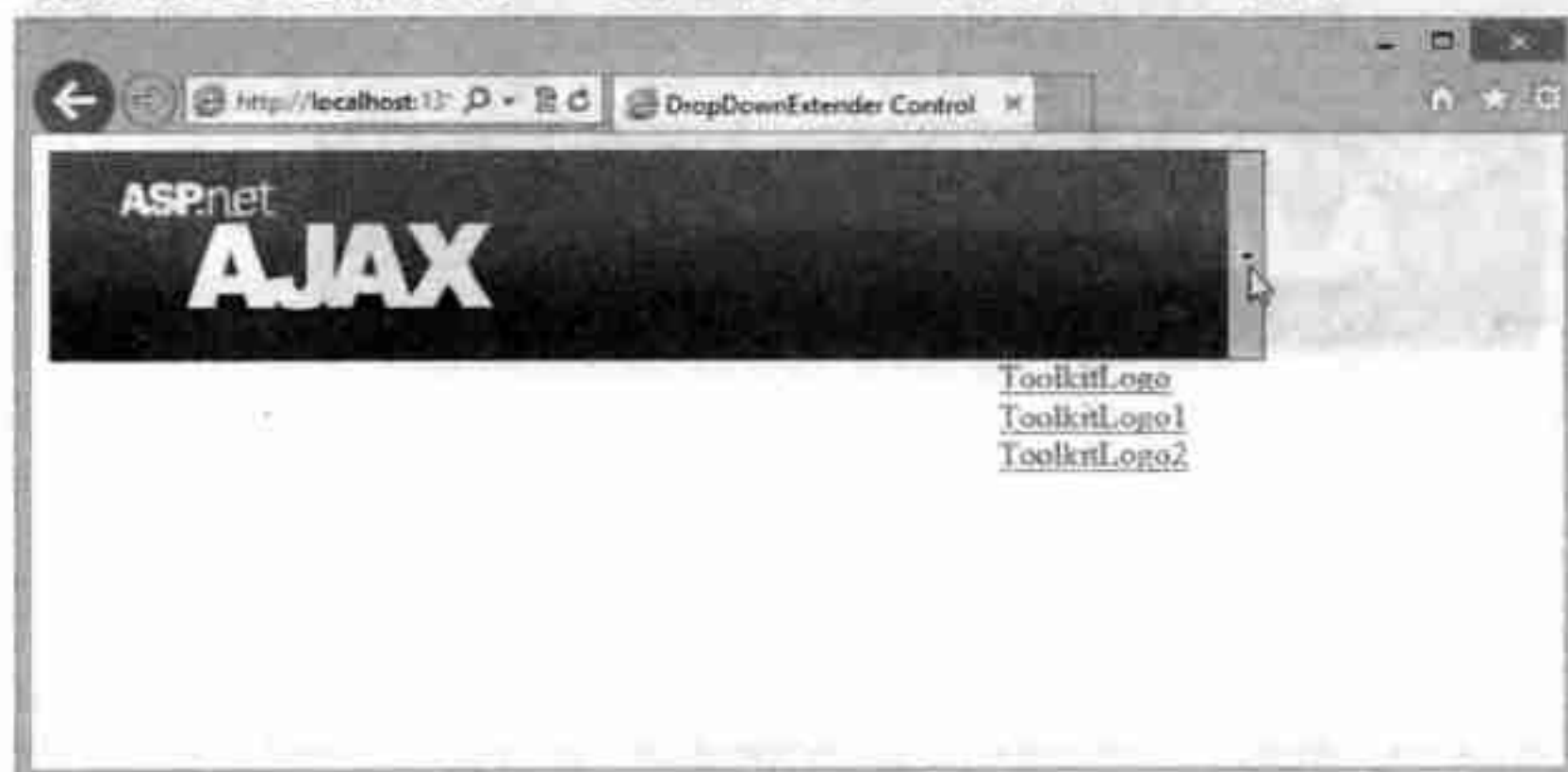


图 24-17

11. DropShadowExtender 控件

DropShadowExtender 控件可以在页面的 ASP.NET 面板或图像上添加 DropShadow 效果。首先设置 TargetControlID 属性,再控制 Width 和 Opacity,最后控制是否应为圆角(Rounded 属性)。如果面板可以移动或重置大小,那么还可以把 TrackPosition 属性设置为 True,表示可运行 JavaScript 来跟踪面板,根据需要更新 DropShadow 效果。

首先可以想到使用该控件的地方就是图像,如程序清单 24-14 所示,也可以把它用于任意控件。

程序清单 24-14 将 DropShadowExtender 控件用于 Image 控件

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>DropShadowExtender Control</title>
</head>
<body>
    <form id="form1" runat="server"> <div>
        <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
        </asp:ToolkitScriptManager>
        <asp:DropShadowExtender ID="DropShadowExtender1" runat="server"
            TargetControlID="Image1">
        </asp:DropShadowExtender>
        <asp:Image ID="Image1" runat="server"
            ImageUrl="Images/ToolkitLogo.jpg" />
    </div>
    </form>
</body>
</html>
```

在这个例子中,使用了 DropShadowExtender 控件,把它的 TargetControlID 属性设置为 Image1。之后,相应的图像就会显示在浏览器上,如图 24-18 所示。

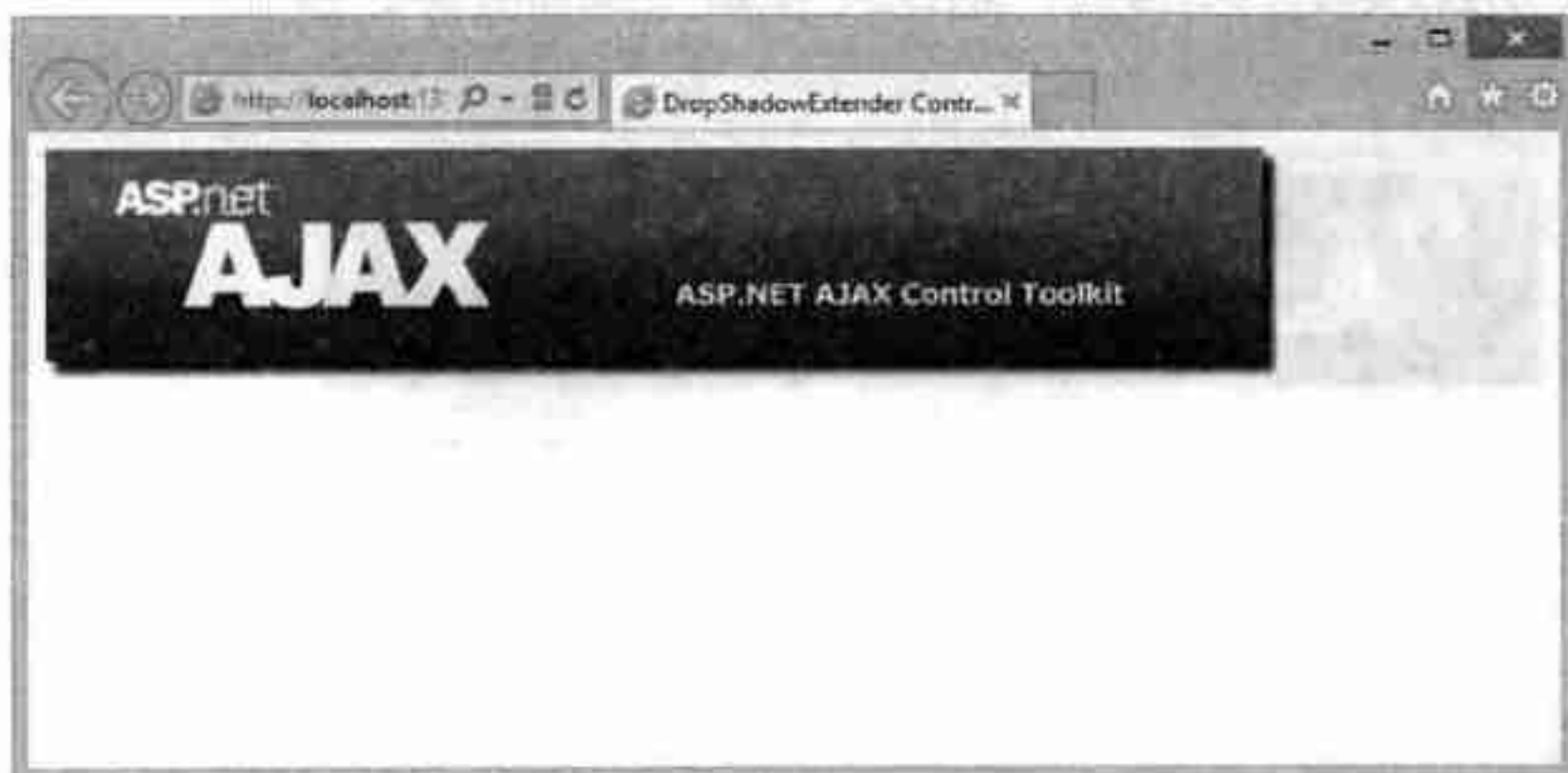


图 24-18

如前所述,除了图像之外,还可以把 DropShadowExtender 控件用于其他任意控件。程序清单 24-15 说明了如何把它用于 Panel 控件。

程序清单 24-15 将 DropShadowExtender 控件用于 Panel 控件

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>DropShadowExtender Control</title>
</head>
<body>
    <form id="form1" runat="server"> <div>
        <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
        </asp:ToolkitScriptManager>
        <asp:DropShadowExtender ID="DropShadowExtender1" runat="server"
            TargetControlID="Panel1" Rounded="True">
        </asp:DropShadowExtender>
        <asp:Panel ID="Panel1" runat="server" BackColor="Orange"
            Width="300" HorizontalAlign="Center">
            <asp:Login ID="Login1" runat="server">
            </asp:Login>
        </asp:Panel>
    </div>
</form>
</body>
</html>
```



如果出现错误——WebForms UnobtrusiveValidationMode 需要 ScriptResourceMapping 才能执行 jQuery, 就在 web.config 文件中使用如下设置:

```
<appSettings>
    <add key="ValidationSettings:UnobtrusiveValidationMode" value="None" />
</appSettings>
```

在这个例子中,使用 DropShadowExtender 控件扩展了包含 Login 控件的 Panel 控件。结果非常类似于前面的 Image 控件的结果。但是,这里的 DropShadowExtender 控件还把 Rounded 属性设置为 True(该属性默认设置为 False),结果如图 24-19 所示。

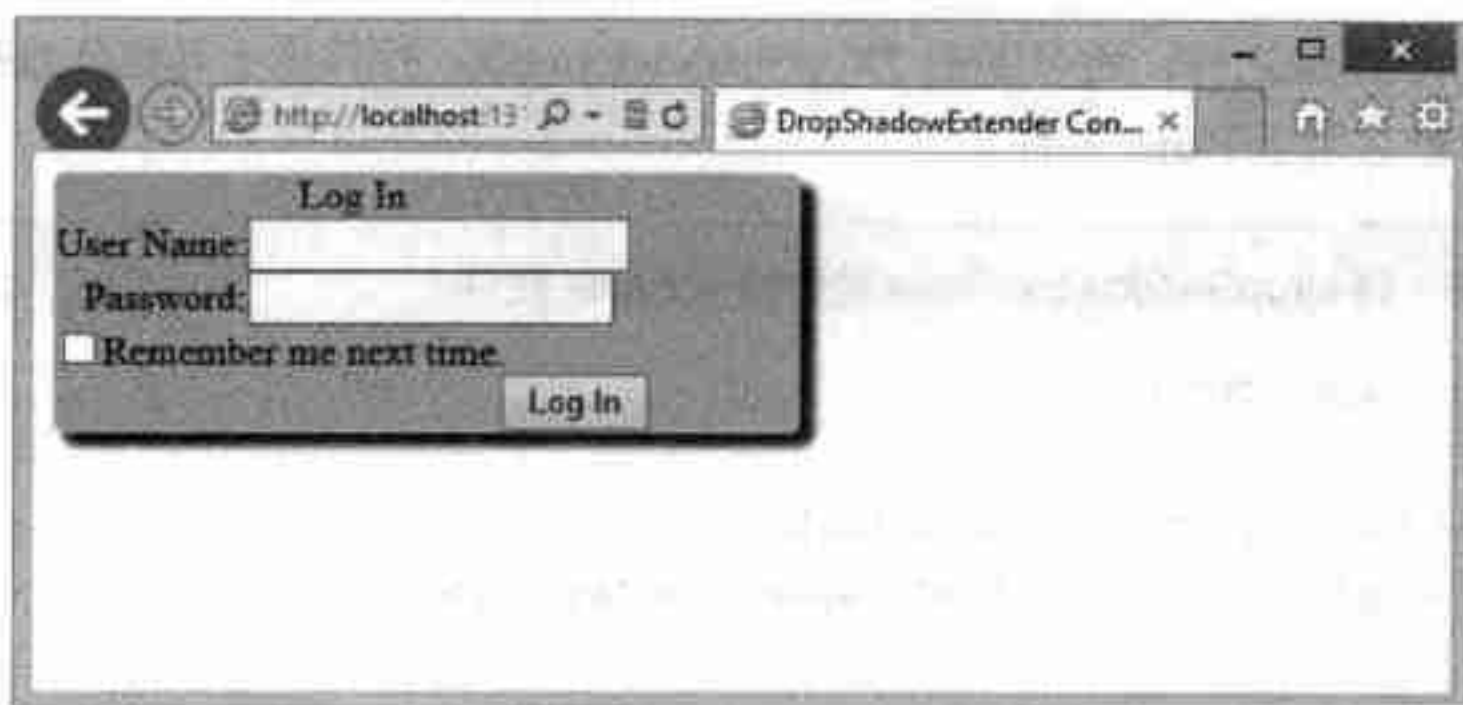


图 24-19

如图 24-19 所示, 不仅对阴影的边界进行了圆角操作, 还对整个 Panel 控件进行了圆角操作。可以使用的其他样式属性有 `Opacity` 和 `Radius`, 其中 `Opacity` 属性控制阴影的不透明度, `Radius` 属性控制边界圆角时的半径, 只有 `Rounded` 属性设置为 `True` 时才有效。在默认情况下, `Opacity` 属性设置为 1, 表示 100% 可见。要设置 50% 的不透明度, 必须把 `Opacity` 属性设置为 0.5。

12. DynamicPopulateExtender 控件

`DynamicPopulateExtender` 控件可以给 Panel 控件发送动态的 HTML 输出。为此, 需要使用控件或事件触发对服务器的回调, 以此获得 HTML 输出。之后把 HTML 输出放在 Panel 控件上, 从而在客户端上执行动态改动。

与 `AutoCompleteExtender` 控件一样, 这个控件也需要服务器端事件, 把某些数据异步返回到客户端。程序清单 24-16 列出了在 .aspx 页面上使用这个控件所需的代码(本章下载代码中的 `DynamicPopulateExtender.aspx`)。

程序清单 24-16 使用 DynamicPopulateExtender 控件填充 Panel 控件

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="DynamicPopulateExtender.aspx.cs"
    Inherits="DynamicPopulateExtender" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head id="Head1" runat="server">
    <title>DynamicPopulateExtender Control</title>
    <script type="text/javascript">
        function updateGrid(value) {
            var behavior = $find('DynamicPopulateExtender1');
            if(behavior) {
                behavior.populate(value);
            }
        }
    </script>
</head>
<body>
    <form id="form1" runat="server">
        <div>
```



```

<asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server" />
<asp:DynamicPopulateExtender ID="DynamicPopulateExtender1" runat="server"
    TargetControlID="Panell" ServiceMethod="GetDynamicContent">
</asp:DynamicPopulateExtender>
<div onclick="updateGrid(0);">
<asp:LinkButton ID="LinkButton1" runat="server"
    OnClientClick="return false;">Customers</asp:LinkButton></div>
<div onclick="updateGrid(1);">
<asp:LinkButton ID="LinkButton2" runat="server"
    OnClientClick="return false;">Employees</asp:LinkButton></div>
<div onclick="updateGrid(2);">
<asp:LinkButton ID="LinkButton3" runat="server"
    OnClientClick="return false;">Products</asp:LinkButton></div>
<asp:Panel ID="Panell" runat="server">
</asp:Panel>
</div>
</form>
</body>
</html>

```

这个.aspx 页面做了许多工作。首先, 客户端的 JavaScript 函数 updateGrid 调用了页面上的 DynamicPopulateExtender 控件。页面上还有 3 个 LinkButton 服务器控件, 每个控件都放在一个<div>元素中, <div>元素调用 updateGrid 函数, 并给该函数传送一个值。因为我们希望<div>元素的 onclick 事件通过单击来触发, 而不是通过 LinkButton 控件的单击事件来触发, 所以每个 LinkButton 控件都包含 OnClientClick 属性, 但是不执行任何操作。可使用“return false;”语句实现该功能。

页面上的 DynamicPopulateExtender 控件把 Panell 控件作为容器, 提取在异步请求时来自服务器的 HTML。DynamicPopulateExtender 控件知道使用 ServiceMethod 特性从哪里获取 HTML。这个特性的值调用 GetDynamicContent 方法, 该方法位于页面的隐藏代码文件中。

有了.aspx 页面后, 下一步就是创建隐藏代码页面。这个页面包含 DynamicPopulateExtender 控件调用的服务器端方法, 如程序清单 24-17 所示(本章下载代码中的 DynamicPopulateExtender.aspx.cs)。

程序清单 24-17 DynamicPopulateExtender.aspx 页面的隐藏代码页面

```

using System.Data;
using System.Data.SqlClient;
using System.IO;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class DynamicPopulateExtender : System.Web.UI.Page
{
    [System.Web.Services.WebMethodAttribute(),
    System.Web.Script.Services.ScriptMethodAttribute()]
    public static string GetDynamicContent(string contextKey)
    {
        SqlConnection conn;
        SqlCommand cmd;
        string cmdString = "Select * from Customers";

        switch(contextKey)

```

```
{
    case ("1"):
        cmdString = "Select * from Employees";
        break;
    case ("2"):
        cmdString = "Select * from Products";
        break;
}

conn = new
    SqlConnection(@"Data Source=.\SQLEXPRESS;
        AttachDbFilename=|DataDirectory|\NORTHWND.MDF;
        Integrated Security=True;User Instance=True");
    // Put this string on one line in your code
cmd = new SqlCommand(cmdString, conn);
conn.Open();

SqlDataReader myReader;
myReader = cmd.ExecuteReader(CommandBehavior.CloseConnection);

DataTable dt = new DataTable();
dt.Load(myReader);
myReader.Close();

GridView myGrid = new GridView();
myGrid.ID = "GridView1";
myGrid.DataSource = dt;
myGrid.DataBind();

StringWriter sw = new StringWriter();
HtmlTextWriter htw = new HtmlTextWriter(sw);

myGrid.RenderControl(htw);
htw.Close();

return sw.ToString();
}
}
```

这段代码是 `DynamicPopulateExtender.aspx` 页面的隐藏代码页面，其中包含一个可以异步调用的方法。`GetDynamicContent` 方法带有参数 `contextKey`，这是一个字符串值，可以用于确定终端用户单击了什么链接。

根据用户作出的选择，使用特定的命令字符串填充 `DataTable` 对象。之后，把 `DataTable` 对象用作显示的编程控件 `GridView` 的数据源，并将该对象作为字符串返回给客户端。客户端提取这个大字符串，使用其中的文本填充页面上的 `Panel1` 控件。单击其中一个链接的结果如图 24-20 所示。

ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
1	Chai	1	1	10 boxes x 20 bags	18.0000	39	0	10	
2	Chang	1	1	24 - 12 oz bottles	19.0000	17	40	25	
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10.0000	13	70	25	
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22.0000	53	0	0	
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.3500	0	0	0	
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25.0000	120	0	25	
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	30.0000	15	0	10	
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	40.0000	6	0	0	
9	Mishi Kobe Niku	4	8	18 - 500 g pkgs.	97.0000	29	0	0	
10	Ikan	4	8	12 - 200 ml jars	31.0000	11	0	0	
11	Queso Caboties	5	4	1 kg pkg	21.0000	22	30	30	
12	Queso Manchego La Pastora	5	4	10 - 500 g pkgs.	38.0000	36	0	0	
13	Konbu	6	8	2 kg box	6.0000	24	0	5	
14	Tofu	6	7	40 - 100 g pkgs.	23.2500	35	0	0	
15	Genen Shoyu	6	2	24 - 250 ml bottles	15.5000	39	0	5	
16	Pavlova	7	3	32 - 500 g boxes	17.4500	29	0	10	
17	Alice Mutton	7	6	20 - 1 kg tins	39.0000	0	0	0	
18	Carnarvon Tigers	7	8	16 kg pkg	62.5000	42	0	0	
19	Teatime Chocolate Biscuits	8	3	10 boxes x 12 pieces	9.2000	25	0	5	
20	Sir Rodney's Marmalade	8	3	30 gift boxes	81.0000	40	0	0	
21	Sir Rodney's Scones	8	3	24 pkgs. x 4 pieces	10.0000	1	40	5	
22	Gustaf's Knäckebröd	9	3	24 - 500 g pkgs.	21.0000	104	0	25	
23	Tunnbröd	9	5	12 - 250 g pkgs.	9.0000	61	0	25	
24	Cheese Puffs	10	1	12 - 3.66 kg pkgs.	4.0000	30	0	0	

图 24-20

13. FilteredTextBoxExtender 控件

FilteredTextBoxExtender 控件最初由本书的作者提供, 使用 TextBox 控件指定终端用户可以在控件中输入的字符类型。例如, 如果希望终端用户只能在文本框中输入数字, 就可以把 FilteredTextBoxExtender 关联到 TextBox 控件, 指定该输入限制。程序清单 24-18 是一个例子。

程序清单 24-18 限制文本框只能使用数字

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>FilteredTextBoxExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>
            <asp:FilteredTextBoxExtender ID="FilteredTextBoxExtender1" runat="server"
                TargetControlID="TextBox1" FilterType="Numbers">
            </asp:FilteredTextBoxExtender>
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        </div>
    </form>
</body>
</html>
```

在这个例子中, 使用 TargetControlID 属性把 FilteredTextBoxExtender 控件关联到 TextBox1 控件。

FilteredTextBoxExtender 控件有一个 FilterType 属性, 它的值可以是 Custom、LowercaseLetters、Numbers 和 UppercaseLetters。

在这个例子中, FilterType 属性的值是 Numbers, 表示只能把数字输入到文本框中。如果终端用户试图输入其他类型的信息, 就不会发生任何事情——对终端用户而言, 就好像按键不起作用一样。

FilteredTextBoxExtender 控件的其他两个属性是 FilterMode 和 InvalidChars。使用这两个属性的例子如下:

```
<asp:FilteredTextBoxExtender ID="FilteredTextBoxExtender1" runat="server"
    TargetControlID="TextBox1" InvalidChars="*" FilterMode="InvalidChars">
</asp:FilteredTextBoxExtender>
```

FilterMode 属性的默认值是 ValidChars。当 FilterMode 属性设置为 ValidChars 时, 该控件就使用 FilterType 属性, 只允许输入该属性定义的类型。而当 FilterMode 属性设置为 InvalidChars 时, 就可以使用 InvalidChars 属性指定无效字符(多个字符放在一起, 中间没有空格或其他字符)。

14. HoverMenuExtender 控件

当终端用户把光标停放在另一个控件上时, HoverMenuExtender 控件允许把隐藏的控件显示在屏幕上。这表示可以建立漂亮的工具提示, 或者在终端用户把光标停放在应用程序的某个地方时提供额外的功能。

一个例子是修改 ListView 控件, 当终端用户把光标停放在产品名称上时, 该行数据的 Edit 按钮就显示在屏幕上。ListView 控件中给<ItemTemplate>添加扩展程序的完整代码如程序清单 24-19 所示。

程序清单 24-19 在 ListView 控件的<ItemTemplate>部分添加停放按钮

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head id="Head1" runat="server">
    <title>HoverMenuExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>
            <asp:ListView ID="ListView1" runat="server" DataSourceID="SqlDataSource1">
                <LayoutTemplate>
                    <table>
                        <tr>
                            <th></th>
                            <th>ProductID</th>
                            <th>ProductName</th>
                            <th>SupplierID</th>
                            <th>CategoryID</th>
                            <th>QuantityPerUnit</th>
                        </tr>
```

```

        <asp:PlaceHolder ID="itemPlaceholder" runat="server" />
    </table>
</LayoutTemplate>
<ItemTemplate>
    <tr style="background-color: #DCDCDC; color: #000000;">
        <td>
            <asp:HoverMenuExtender ID="HoverMenuExtender1" runat="server"
                TargetControlID="ProductNameLabel" PopupControlID="Panel1"
                PopDelay="25" OffsetX="-50">
            </asp:HoverMenuExtender>
            <asp:Panel ID="Panel1" runat="server" Height="50px"
                Width="125px">
                <asp:Button ID="EditButton" runat="server"
                    CommandName="Edit" Text="Edit" />
            </asp:Panel>
        </td>
        <td>
            <asp:Label ID="ProductIDLabel" runat="server"
                Text='<%= Eval("ProductID") %>' />
        </td>
        <td>
            <asp:Label ID="ProductNameLabel" runat="server"
                Text='<%= Eval("ProductName") %>' />
        </td>
        <td>
            <asp:Label ID="SuppliedIDLabel" runat="server"
                Text='<%= Eval("SupplierID") %>' />
        </td>
        <td>
            <asp:Label ID="CategoryIDLabel" runat="server"
                Text='<%= Eval("CategoryID") %>' />
        </td>
        <td>
            <asp:Label ID="QuantityPerUnitLabel" runat="server"
                Text='<%= Eval("QuantityPerUnit") %>' />
        </td>
    </tr>
</ItemTemplate>
</asp:ListView>
<asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\NORTHWND.MDF;Integrated
Security=True;User Instance=True" ProviderName="System.Data.SqlClient"
SelectCommand="SELECT * FROM [Products]"></asp:SqlDataSource>
</div>
</form>
</body>
</html>

```

这里把 HoverMenuExtender 控件关联到 ID 为 ProductNameLabel 的 Label 控件, 该 Label 控件显示在 ListView 控件的每一行中。这需要使用 TargetControlID 属性, 而在用户把光标停放在目标控件上时, 使用 PopupControlID 属性指定动态显示的控件。

HoverMenuExtender 控件有几个属性, 可用于控制弹出内容的样式和行为。这个例子使用了 PopDelay 属性, 它提供了一种弹出内容延迟出现的方式(单位为毫秒)。OffsetX 和 OffsetY 属性根据

目标控件指定弹出内容的位置。在这个例子中,将偏移量设置为 - 50 像素。该操作的结果如图 24-21 所示。



ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit
1	Chai	1	1	10 boxes x 20 bags
2	Chang	1	1	24 - 12 oz bottles
3	Aniseed Syrup	1	2	12 - 550 ml bottles
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars
5	Chef Anton's Gumbo Mix	2	2	36 boxes
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs
10	Ikura	4	8	12 - 200 ml jars
11	Queso Cabrales	5	4	1 kg pkg
12	Queso Manchego La Pastora	5	4	10 - 500 g pkgs
13	Konbu	6	8	2 kg box
14	Tofu	6	7	40 - 100 g pkgs
15	Genen Shoyu	6	2	24 - 250 ml bottles
16	Pavlova	7	3	32 - 500 g boxes
17	Alice Mutton	7	6	20 - 1 kg tins
18	Carnarvon Tigers	7	8	16 kg pkg
19	Teatime Chocolate Biscuits	8	3	10 boxes x 12 pieces

图 24-21

15. ListSearchExtender 控件

ListSearchExtender 控件扩展了 ListBox 或 DropDownList 控件,但在 Opera 和 Safari 等浏览器中并不总是能得到最佳效果。这个扩展程序可以提供搜索这些控件中的大型集合的功能,从而终端用户不需要搜索集合以找出他们需要的项。

在使用时,这个扩展程序会为控件上方的搜索区域添加搜索文本,显示终端用户输入的字符。程序清单 24-20 显示了这个扩展程序的用法。

程序清单 24-20 使用 ListSearchExtender 控件扩展 ListBox 控件

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>ListSearchExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>
            <asp:ListSearchExtender ID="ListSearchExtender1" runat="server"
                TargetControlID="ListBox1">
            </asp:ListSearchExtender>
            <asp:ListBox ID="ListBox1" runat="server" Width="150">
```



```

        <asp:ListItem>Aardvark</asp:ListItem>
        <asp:ListItem>Bee</asp:ListItem>
        <asp:ListItem>Camel</asp:ListItem>
        <asp:ListItem>Dog</asp:ListItem>
        <asp:ListItem>Elephant</asp:ListItem>
    </asp:ListBox>
</div>
</form>
</body>
</html>

```

在这个例子中,只使用了 ListSearchExtender 控件的 TargetControlID 属性,把它扩展的控件关联起来。运行这个页面的结果如图 24-22 所示。

终端用户在开始输入时会在控件下方看到输入的文本,如图 24-23 所示。



图 24-22

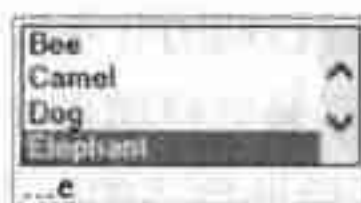


图 24-23

使用 PromptCssClass、PromptPosition 和 PromptText 属性可以定制显示在控件顶部的文本。在默认情况下, PromptPosition 属性设置为 Top(另一个值是 Bottom), PromptText 属性的值是 Type to search。

16. MaskedEditExtender 控件和 MaskedEditValidator 控件

MaskedEditExtender 控件类似于 FilteredTextBoxExtender 控件,因为它也限制终端用户在 TextBox 控件中输入特定的文本。但是,这个控件使该过程更进了一步,因为它在文本框中为终端用户提供了一个模板,如果终端用户不遵循该模板,就不能继续输入。还可以在该控件中使用 MaskedEditValidator 控件,给终端用户提供有效性警告。

程序清单 24-21 是使用这两个控件的一个例子。

程序清单 24-21 使用 MaskedEditExtender 控件和 MaskedEditValidator 控件

```

<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>MaskedEditExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>
            <asp:MaskedEditExtender ID="MaskedEditExtender1" runat="server"
                TargetControlID="TextBox1" MaskType="Number" Mask="999">
            </asp:MaskedEditExtender>

```

```

    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    <asp:MaskedEditValidator ID="MaskedEditValidator1"
        runat="server" ControlExtender="MaskedEditExtender1"
        ControlToValidate="TextBox1" IsValidEmpty="False"
        EmptyValueMessage="A three digit number is required!"
        Display="Dynamic"></asp:MaskedEditValidator>
</div>
</form>
</body>
</html>

```

在这个例子中,MaskedEditExtender 控件使用 TargetControlID 属性把自己关联到 TextBox1 控件。MaskType 属性提供了文本框上的蒙版或过滤器的类型,其值可以是:

- None: 表示不进行有效性检查。
- Date: 表示验证日期的有效性。
- DateTime: 表示验证日期和时间的有效性。
- Number: 表示验证数字的有效性。
- Time: 表示验证时间的有效性。

程序清单 24-21 使用了 Number 值,接着指定需要提取数字的蒙版或模板。使用 Mask 属性实现该操作。在这个例子中,Mask 属性设置为 999,表示数字的长度只能是 3 位。

使用 999 作为 Mask 属性的值,表示终端用户在文本框中输入值时,文本框中会显示 3 条下划线。图 24-24 显示了用于输入项的模板。

如果将 Mask 属性改为 99,999.99,如下所示:

```

<asp:MaskedEditExtender ID="MaskedEditExtender1" runat="server"
    TargetControlID="TextBox1" MaskType="Number" Mask="99,999.99">
</asp:MaskedEditExtender>

```

便会出现文本框的输入模板,如图 24-25 所示。

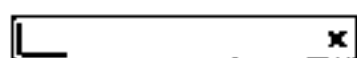


图 24-24

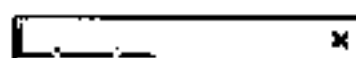


图 24-25

从图 24-25 中可以看出,模板中出现了逗号和句号。在终端用户输入时,不需要再次输入这些字符。光标会移到数字中需要输入的下一部分。

从 Mask 属性值可以看出,使用 9 表示数字。在使用其他 MaskType 值时,还需要了解其他蒙版字符,如下所示:

- 9: 仅输入数字
- L: 仅输入字母
- \$: 仅输入字母或空格
- C: 仅输入定制字符(区分大小写)
- A: 仅输入字母或定制字符
- N: 仅输入数字或定制字符
- ? : 可以输入任意字符

除了对字符的限制之外,模板中使用的分隔符如下:

- /是日期分隔符

- :是时间分隔符
- .是小数分隔符
- ,是千位分隔符
- \是转义字符
- {是重复蒙版的起始分隔符
- }是重复蒙版的结束分隔符

使用其中的一些项，很容易修改 MaskedEditExtender 控件以处理 DateTime 值：

```
<asp:MaskedEditExtender ID="MaskedEditExtender1" runat="server"
    TargetControlID="TextBox1" MaskType="DateTime" Mask="99/99/9999 99:99:99">
</asp:MaskedEditExtender>
```

在文本框中创建的模板如图 24-26 所示。

MaskedEditExtender 控件有许多属性用来控制和处理文本框的行为和样式。MaskedEditExtender 控件还可以与 MaskedEditValidator 控件一起使用，提供文本框控件的有效性检查。

在前面的例子中，通过 MaskedEditValidator 控件的一个实例完成有效性检查。

```
<asp:MaskedEditValidator ID="MaskedEditValidator1" runat="server"
    ControlExtender="MaskedEditExtender1" ControlToValidate="TextBox1"
    IsValidEmpty="False" EmptyValueMessage="A three digit number is required!"
    Display="Dynamic"></asp:MaskedEditValidator>
```

这个控件使用 ControlExtender 属性把自己和 MaskedEditExtender 控件关联起来，并且使用 ControlToValidate 属性检查表单上的特定控件。在默认情况下，IsValidEmpty 属性设置为 True，但改为 False 则表示终端用户必须在文本框中输入某个值，才能通过有效性检查，并且不会出现 EmptyValueMessage 属性指定的错误消息。

触发 MaskedEditValidator 控件会显示如图 24-27 所示的消息。一定要注意，可以使用许多方式设置控件的样式，生成希望的有效性消息显示。



图 24-26

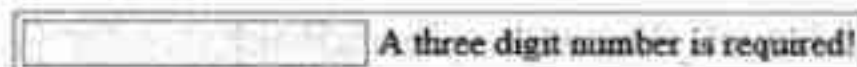


图 24-27

17. MutuallyExclusiveCheckBoxExtender 控件

我们常常要提供一组复选框，把它们用作单选按钮。也就是说，在有一组复选框时，我们只希望终端用户从所提供的项列表中选择一项。

使用 MutuallyExclusiveCheckBoxExtender 控件就可以完成这个操作。程序清单 24-22 说明了如何完成这项任务。

程序清单 24-22 将 MutuallyExclusiveCheckBoxExtender 控件用于复选框

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>
```



```

<!DOCTYPE html>
<html>
<head runat="server">
    <title>MutuallyExclusiveCheckBoxExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>
            <asp:MutuallyExclusiveCheckBoxExtender
                ID="MutuallyExclusiveCheckBoxExtender1" runat="server"
                TargetControlID="CheckBox1" Key="MyCheckboxes" />
            <asp:CheckBox ID="CheckBox1" runat="server" Text="Blue" />
            <br />
            <asp:MutuallyExclusiveCheckBoxExtender
                ID="MutuallyExclusiveCheckBoxExtender2" runat="server"
                TargetControlID="CheckBox2" Key="MyCheckboxes" />
            <asp:CheckBox ID="CheckBox2" runat="server" Text="Brown" />
            <br />
            <asp:MutuallyExclusiveCheckBoxExtender
                ID="MutuallyExclusiveCheckBoxExtender3" runat="server"
                TargetControlID="CheckBox3" Key="MyCheckboxes" />
            <asp:CheckBox ID="CheckBox3" runat="server" Text="Green" />
            <br />
            <asp:MutuallyExclusiveCheckBoxExtender
                ID="MutuallyExclusiveCheckBoxExtender4" runat="server"
                TargetControlID="CheckBox4" Key="MyCheckboxes" />
            <asp:CheckBox ID="CheckBox4" runat="server" Text="Orange" />
            <br />
        </div>
    </form>
</body>
</html>

```

不能把 **MutuallyExclusiveCheckBoxExtender** 控件和 **CheckBoxList** 控件关联起来, 因此, 需要使用 **CheckBox** 控件布置每个复选框, 如上面的代码所示。页面上的每个 **CheckBox** 控件都要有一个 **MutuallyExclusiveCheckBoxExtender** 控件。

使用 **Key** 属性建立一个 **CheckBox** 控件组。组中的所有复选框都需要有相同的 **Key** 值。在程序清单 24-22 所示的例子中, 所有复选框的 **Key** 值都是 **MyCheckboxes**。

运行这个页面, 会得到一个包含 4 个复选框的列表。选中其中一个复选框时, 会出现复选标记。选中另一个复选框时, 就会取消选中第一个选中的复选框。甚至可以取消在该组中的所有选择, 即在复选框组中不选中任何复选框。

18. NumericUpDownExtender 控件

NumericUpDownExtender 控件可以把上/下指示器放在 **TextBox** 控件的旁边, 以便终端用户更方便地控制选择(类似于 HTML5 的 `<input type="number">` 元素)。

程序清单 24-23 是使用该控件的一个简单例子。

程序清单 24-23 使用 NumericUpDownExtender 控件

```

<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>NumericUpDownExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>
            <asp:NumericUpDownExtender ID="NumericUpDownExtender1"
                runat="server" TargetControlID="TextBox1" Width="150"
                Maximum="10" Minimum="1">
            </asp:NumericUpDownExtender>
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        </div>
    </form>
</body>
</html>

```

这里的 NumericUpDownExtender 控件扩展了页面上的 TextBox 控件。使用 NumericUpDownExtender 控件时, 必须使用 Width 属性指定该控件的宽度。否则, 就只能看到上下箭头键, 而看不到文本框区域。在这个例子中, Width 属性设置为 150 像素。Maximum 和 Minimum 属性提供了上下指示器使用的范围。

将 Maximum 属性设置为 10, Minimum 属性设置为 1, 因此该控件的范围是 1 到 10。运行这个页面的结果如图 24-28 所示。

除了使用数字(如程序清单 24-23 所示)之外, 还可以使用文本, 如程序清单 24-24 所示。

程序清单 24-24 对 NumericUpDownExtender 控件使用字符而不是数字

```

<asp:NumericUpDownExtender ID="NumericUpDownExtender1" runat="server"
    TargetControlID="TextBox1" Width="150"
    RefValues="Blue;Brown;Green;Orange;Black;White">
</asp:NumericUpDownExtender>

```

在这个例子中, 在 RefValues 属性中定义单词(以分号隔开), 结果如图 24-29 所示。

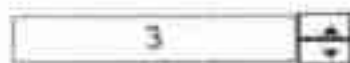


图 24-28

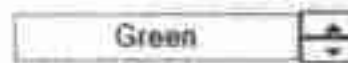


图 24-29

19. PagingBulletedListExtender 控件

PagingBulletedListExtender 控件可以使用很长的项目符号列表, 并对列表按字母顺序分页。例如, 程序清单 24-25 就对 Northwind 数据库中的 Customers 表使用了这个控件。

程序清单 24-25 对 Northwind 数据库中的项目符号列表分页

```

<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="ajaxToolkit" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>PagingBulletedListExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <ajaxToolkit:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </ajaxToolkit:ToolkitScriptManager>
            <ajaxToolkit:PagingBulletedListExtender
                ID="PagingBulletedListExtender1"
                runat="server" TargetControlID="BulletedList1">
            </ajaxToolkit:PagingBulletedListExtender>
            <asp:SqlDataSource ID="SqlDataSource1" runat="server"
                ConnectionString="Data Source=.\SQLEXPRESS;
                AttachDbFilename=|DataDirectory|\NORTHWND.MDF;
                Integrated Security=True;User Instance=True"
                ProviderName="System.Data.SqlClient"
                SelectCommand="SELECT [CompanyName] FROM [Customers]">
            </asp:SqlDataSource>
            <asp:BulletedList ID="BulletedList1" runat="server"
                DataSourceID="SqlDataSource1" DataTextField="CompanyName"
                DataValueField="CompanyName">
            </asp:BulletedList>
        </div>
    </form>
</body>
</html>

```

这段代码从 Northwind 数据库的 Customers 表中提取了所有的 CompanyName 值, 并把这些值绑定到页面的 BulletList 控件。运行这个页面的结果如图 24-30 所示。

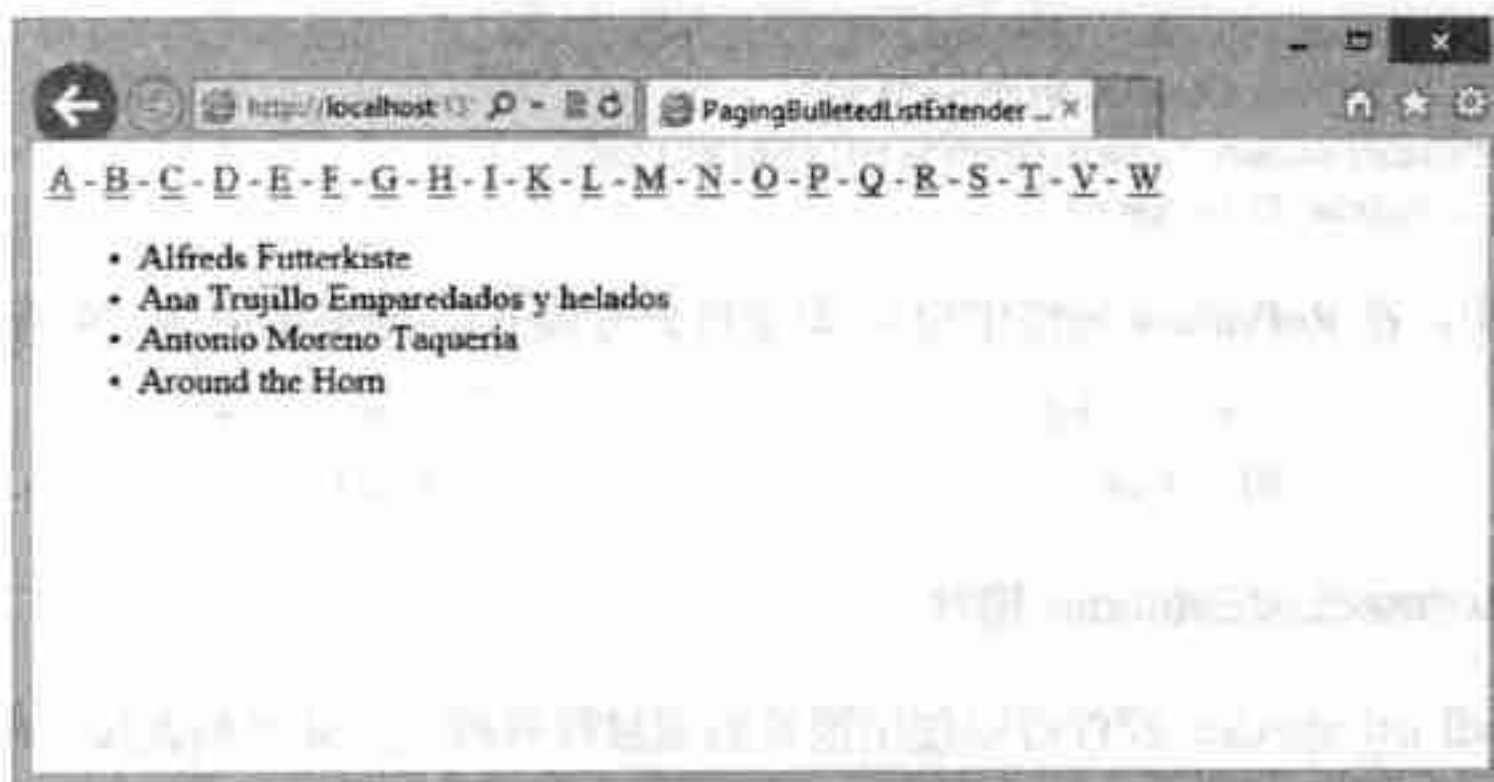


图 24-30

从图 24-30 中可以看出，在客户端是按字母顺序组织分页的，只有字母显示在字母链表中。单击任意一个字母，都会列出项目符号列表中以该字母开头的项。

20. PopupControlExtender 控件

PopupControlExtender 控件可以为页面上的任意控件创建弹出窗口。例如，可以创建一个弹出窗口，它在 TextBox 控件中包含一个 Calendar 控件，以模拟前面的 CalendarExtender 控件，如程序清单 24-26 所示。

程序清单 24-26 使用 PopupControlExtender 控件创建 CalendarExtender 控件

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<script runat="server">
    protected void Calendar1_SelectionChanged(object sender, EventArgs e)
    {
        PopupControlExtender1.Commit(Calendar1.SelectedDate.ToShortDateString());
    }
</script>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>PopupControlExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>
            <asp:PopupControlExtender ID="PopupControlExtender1"
                runat="server" TargetControlID="TextBox1"
                PopupControlID="UpdatePanel1" OffsetY="25">
            </asp:PopupControlExtender>
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
            <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                <ContentTemplate>
                    <asp:Calendar ID="Calendar1" runat="server"
                        BackColor="White" BorderColor="White"
                        BorderWidth="1px" Font-Names="Verdana"
                        Font-Size="9pt" ForeColor="Black" Height="190px"
                        NextPrevFormat="FullMonth" Width="350px"
                        OnSelectionChanged="Calendar1_SelectionChanged">
                        <SelectedDayStyle BackColor="#333399"
                            ForeColor="White" />
                        <TodayDayStyle BackColor="#CCCCCC" />
                        <OtherMonthDayStyle ForeColor="#999999" />
                        <NextPrevStyle Font-Bold="True" Font-Size="8pt"
                            ForeColor="#333333" VerticalAlign="Bottom" />
                        <DayHeaderStyle Font-Bold="True" Font-Size="8pt" />
                    </asp:Calendar>
                </ContentTemplate>
            </asp:UpdatePanel>
        </div>
    </form>
</body>
</html>
```

```

        <TitleStyle BackColor="White" BorderColor="Black"
            BorderWidth="4px" Font-Bold="True"
            Font-Size="12pt" ForeColor="#333399" />
    </asp:Calendar>
</ContentTemplate>
</asp:UpdatePanel>
</div>
</form>
</body>
</html>

```

运行这个页面，页面上只会显示一个文本框。单击文本框内部，会弹出日历，可以从中选择日期以填充文本框，如图 24-31 所示。

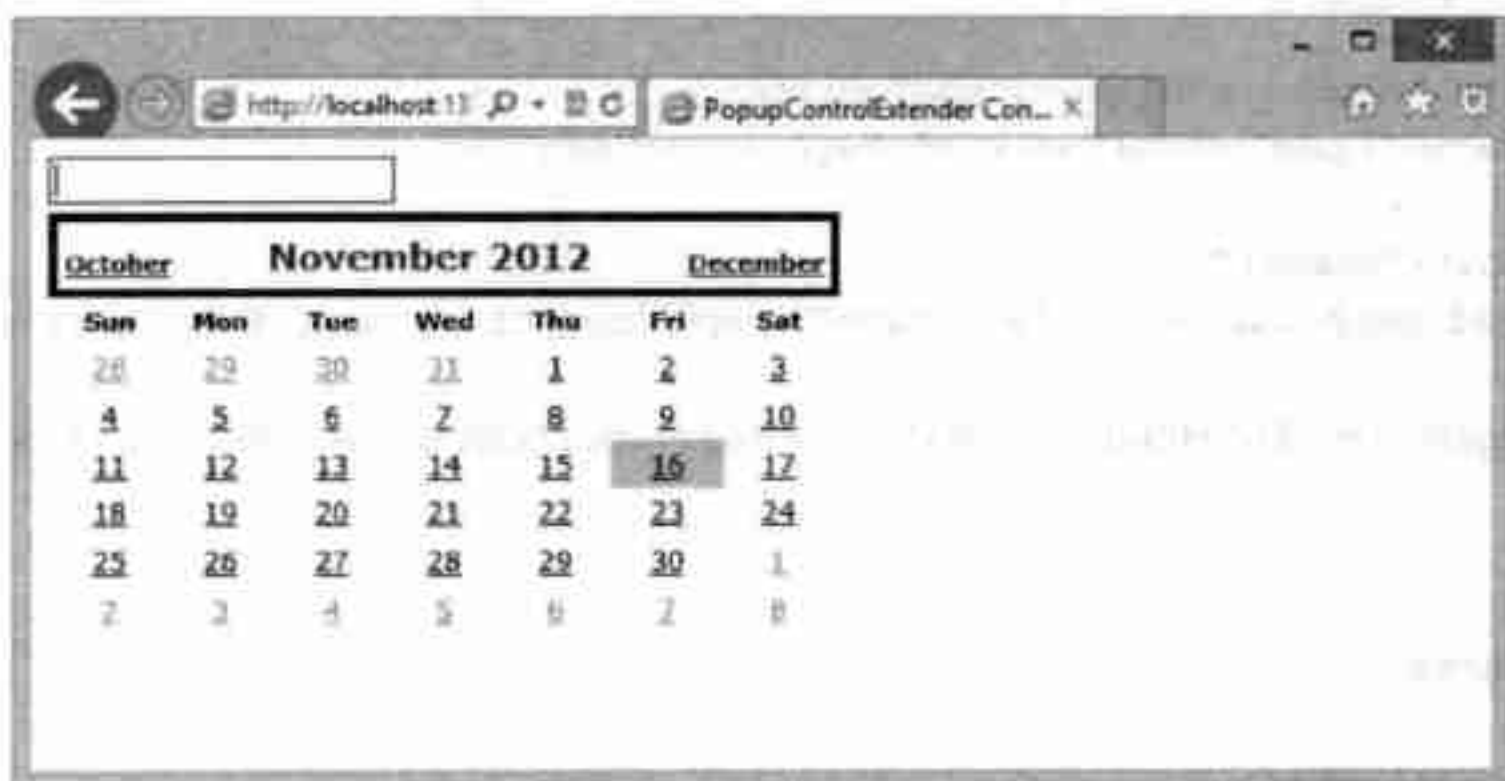


图 24-31

可以把弹出控件放在 ASP.NET AJAX 的 UpdatePanel 控件中，把弹出控件的值传送给目标控件 (TextBox1 控件)，因此要使用 Commit 方法：

```
PopupControlExtender1.Commit(Calendar1.SelectedDate.ToShortDateString())
```

21. ResizableControlExtender 控件

在许多情况下，可能需要限制元素最初显示时的尺寸，但允许用户增大或缩小元素。ResizableControlExtender 控件就可以完成这项任务。把 ResizableControlExtender 控件放在页面上，使用 TargetControlID 控件使之指向 ASP.NET Panel 控件。

ResizableControlExtender 控件可以提取 Panel 控件，让终端用户抓取句柄并改变元素的大小。因此，放在 Panel 控件中的任意对象可以根据终端用户拉伸该对象的方式改变尺寸。为此，必须为终端用户创建句柄，用于拉伸或收缩元素。

使用 HandleCssClass 属性可以指定句柄外观的样式信息，用户可以使用句柄调整面板大小。ResizableCssClass 属性是改变面板时显示的样式信息。

这个控件还提供了事件 OnClientResizeBegin、OnClientResizing 和 OnClientResize，可以给它们关联一些代码来响应面板的调整大小操作。这些事件可以改变文本的大小，如果增大面板，就检索更多的数据；如果面板缩小，就隐藏元素。程序清单 24-27 使用 ResizableControlExtender 控件处理页面上的内联 CSS 信息。这个例子说明了如何将 ResizableControlExtender 控件用于图像。

程序清单 24-27 将 ResizableControlExtender 控件用于图像

```

<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>ResizableControlExtender Control</title>
    <style type="text/css">
        .handle
        {
            width:10px;
            height:10px;
            background-color:Black;
            cursor: se-resize;
        }
        .resizable
        {
            border-style:solid;
            border-width:2px;
            border-color:Black;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>
            <asp:ResizableControlExtender ID="ResizableControlExtender1" runat="server"
                TargetControlID="Panel1" HandleCssClass="handle"
                ResizableCssClass="resizable">
            </asp:ResizableControlExtender>
            <asp:Panel ID="Panel1" runat="server" Width="300" Height="225">
                <asp:Image ID="Image1" runat="server"
                    ImageUrl="Images/ToolkitLogo.jpg"
                    style="width:100%; height:100%"/>
            </asp:Panel>
        </div>
    </form>
</body>
</html>

```

在这个例子中，ResizableControlExtender 控件根据 CSS 为终端用户创建句柄，以调整 Panel 控件的大小。TargetControlID 属性指向要调整大小的控件。

在 ResizableControlExtender 控件中有两个 CSS 引用。第一个 CSS 引用在处理控件时，就好像位于屏幕上，并且不与终端用户交互。这实际上是向终端用户说明，可以调整元素的大小。这需要使使用 HandleCssClass 属性，这个属性的值指向同一文件中的 CSS 类 handle。第二个 CSS 引用在处理控件时，就好像被单击并按住不放(此时终端用户没有执行单击操作)。这需要用到 ResizableCssClass 属性，

这个属性的值指向 CSS 类 `resizable`。

编译并运行代码，应生成如图 24-32 所示的页面。



图 24-32

在顶部的屏幕截图中可以看出没有与终端用户交互时图像的外观。在这个例子中，图像右下角有黑色的方框(由 CSS 定义)。底部的屏幕截图显示了终端用户抓取句柄并开始改变图像的形状时该图像的外观。

22. RoundedCornersExtender 控件

`RoundedCornersExtender` 控件可以给页面上的元素添加圆角。与 `ResizableControlExtender` 控件一样，也可以把需要处理的元素放在 `Panel` 控件中。程序清单 24-28 对包含 `Login` 服务器控件的 `Panel` 控件进行圆角操作。

程序清单 24-28 对包含 `Login` 服务器控件的 `Panel` 控件进行圆角操作

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>RoundedCornersExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
```

```

<div>
  <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
  </asp:ToolkitScriptManager>
  <asp:RoundedCornersExtender ID="RoundedCornersExtender1"
    runat="server" TargetControlID="Panel1">
  </asp:RoundedCornersExtender>
  <asp:Panel ID="Panel1" runat="server" Width="250px"
    HorizontalAlign="Center" BackColor="Orange">
    <asp:Login ID="Login1" runat="server">
    </asp:Login>
  </asp:Panel>
</div>
</form>
</body>
</html>

```

这里的 `RoundedCornersExtender` 控件通过 `TargetControlID` 属性指向 `Panel` 控件。把这个 `Panel` 控件的背景色设为橘色，可以看出它的各个角已变成圆角。运行这段代码的结果如图 24-33 所示。

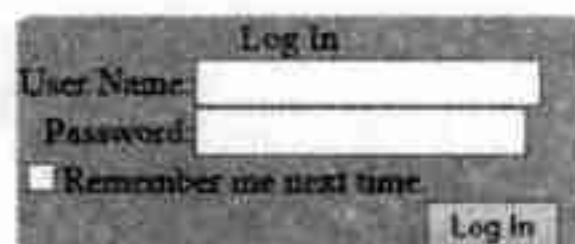


图 24-33

使用 `RoundedCornersExtender` 控件的 `Radius` 属性可以控制圆角的度数，这个属性默认设置为 5。也可以使用 `Corners` 属性指定要进行圆角操作的角。`Corners` 属性的值可以是 `All`、`Bottom`、`BottomLeft`、`BottomRight`、`Left`、`None`、`Right`、`Top`、`TopLeft` 和 `TopRight`。

23. SliderExtender 控件和 MultiHandleSliderExtender 控件

`SliderExtender` 控件可以拉伸 `TextBox` 控件，使其看上去完全不像原来的外观。AJAX 控件工具集中的这个控件可以创建真正的滑块控件，允许终端用户使用鼠标选择一定范围的数字，而不是输入数字。程序清单 24-29 显示了使用滑块的一个简单例子。

程序清单 24-29 使用 `SliderExtender` 控件

```

<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
  Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
  <title>SliderExtender Control</title>
</head>
<body>
  <form id="form1" runat="server">
  <div>
    <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
    </asp:ToolkitScriptManager>
    <asp:SliderExtender ID="SliderExtender1" runat="server" TargetControlID="TextBox1">
    </asp:SliderExtender>
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
  </div>

```

```

    </form>
</body>
</html>

```

这段代码把 SliderExtender 控件关联到一个典型的 TextBox 控件, 生成如图 24-34 所示的结果。

这种方法很不错, 但终端用户很难确定他们选择了什么数字。因此, 最好给终端用户提供指示器。在页面上添加 Label 控件 Label1, 把 SliderExtender 控件改为包含 BoundControlID 属性, 就可以满足上述要求。进行修改后的代码如下:

```

<asp:SliderExtender ID="SliderExtender1" runat="server" TargetControlID="TextBox1"
    BoundControlID="Label1">
</asp:SliderExtender>

```

这个小改动会生成如图 24-35 所示的结果(页面上有相应的 Label 控件)。

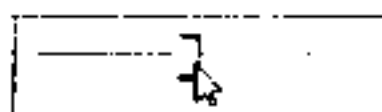


图 24-34

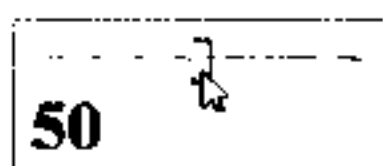


图 24-35

当终端用户在滑块上移动句柄时, 就可以看到所选的数字。SliderExtender 控件的一些属性如下:

- **Decimal:** 可以指定结果使用的小数位。小数位越多, 终端用户选择准确数字的可能性就越小。
- **HandleCssClass:** 用于设计句柄的 CSS 类。
- **HandleImageUrl:** 用于表示句柄的图像文件。
- **Length:** 滑块的长度(以像素为单位), 默认值是 150。
- **Maximum:** 滑块可表示的最大数字, 默认值是 100。
- **Minimum:** 滑块可表示的最小数字, 默认值是 0。
- **Orientation:** 滑块的方向, 值是 Horizontal 和 Vertical。默认值是 Horizontal。
- **RailCssClass:** 用于设计滑块上的横杆的 CSS 类。
- **ToolTipText:** 终端用户把光标停放在滑块上时显示的工具提示。在文本中使用 0, 可以向终端用户显示滑块的当前位置。

MultiHandleSliderExtender 控件的原理基本上一样, 但这个特殊的扩展程序允许终端用户在处理时使用多个句柄。如果需要使用滑块来选择范围或最大值/最小值时, 该控件就会更有用。

24. SlideShowExtender 控件

SlideShowExtender 控件可以把幻灯片放映图像放在浏览器中。SlideShow 控件允许终端用户移到前面或后面的图像, 或者把图像作为幻灯片播放, 并且每幅图像之间有定义好的等待时间。程序清单 24-30 是创建幻灯片放映的例子(本章下载代码中的 SlideShowExtender.aspx)。

程序清单 24-30 使用 3 幅图像创建幻灯片放映

```

<%@ Page Language="C#" AutoEventWireup="true"
    CodeFile="SlideShowExtender.aspx.cs" Inherits="SlideShowExtender" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit" TagPrefix="asp" %>

```



```

<!DOCTYPE html>
<html>
<head id="Head1" runat="server">
    <title>SlideShowExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>
            <asp:Panel ID="Panel1" runat="server" Width="300px"
                HorizontalAlign="Center">
                <asp:SlideShowExtender ID="SlideShowExtender1" runat="server"
                    ImageTitleLabelID="LabelTitle" TargetControlID="Image1"
                    UseContextKey="True" NextButtonID="ButtonNext"
                    PlayButtonID="ButtonPlay"
                    PreviousButtonID="ButtonPrevious"
                    SlideShowServiceMethod="GetSlides"
                    ImageDescriptionLabelID="LabelDescription">
                </asp:SlideShowExtender>
                <asp:Label ID="LabelTitle" runat="server" Text="Label"
                    Font-Bold="True"></asp:Label><br /><br />
                <asp:Image ID="Image1" runat="server"
                    ImageUrl="Images/ToolkitLogo.jpg" /><br />
                <asp:Label ID="LabelDescription" runat="server"
                    Text="Label"></asp:Label><br /><br />
                <asp:Button ID="ButtonPrevious" runat="server" Text="Previous" />
                <asp:Button ID="ButtonNext" runat="server" Text="Next" />
                <asp:Button ID="ButtonPlay" runat="server" />
            </asp:Panel>
        </div>
    </form>
</body>
</html>

```

SlideShowExtender 控件有许多属性。使用 ImageTitleLabelID 和 ImageDescriptionLabelID 属性可以指定定义图像标题和描述的位置。除此之外,这个页面还包含 3 个 Button 控件,一个用作 Previous 按钮,另一个用作 Next 按钮,最后一个用作 Play 按钮。但是一定要注意,单击 Play 按钮时(启动幻灯片放映),该按钮会变成 Stop 按钮。

SlideShowServiceMethod 属性很重要,因为它指向一个服务器端方法,该方法返回幻灯片放映中的图像。在这个例子中,指向的方法是 GetSlides,如程序清单 24-31 所示(本章下载代码中的 SlideShowExtender.aspx.cs)。

程序清单 24-31 GetSlides 方法的实现代码

```

public partial class SlideShowExtender : System.Web.UI.Page
{
    [System.Web.Services.WebMethodAttribute(),
    System.Web.Script.Services.ScriptMethodAttribute()]
    public static AjaxControlToolkit.Slide[]
        GetSlides(string contextKey)
    {

```

```

return new AjaxControlToolkit.Slide[] {
    new AjaxControlToolkit.Slide("Images/ToolkitLogo.jpg",
        "The Logo", "This is the Ajax Control Toolkit Logo."),
    new AjaxControlToolkit.Slide("Images/ToolkitLogo1.jpg",
        "The 2nd Logo", "This is the modified Ajax Control Toolkit Logo."),
    new AjaxControlToolkit.Slide("Images/ToolkitLogo2.jpg",
        "The 3rd Logo", "This is another modified Ajax Control Toolkit Logo.") };
}
}

```

有了这些隐藏代码，SlideShowExtender 控件就有了针对照片调用的服务器端方法。这个方法是 GetSlides，它返回一个 Slide 对象数组，该数组包含对象的位置(路径)、标题和描述。运行这个页面，会得到如图 24-36 所示的结果。

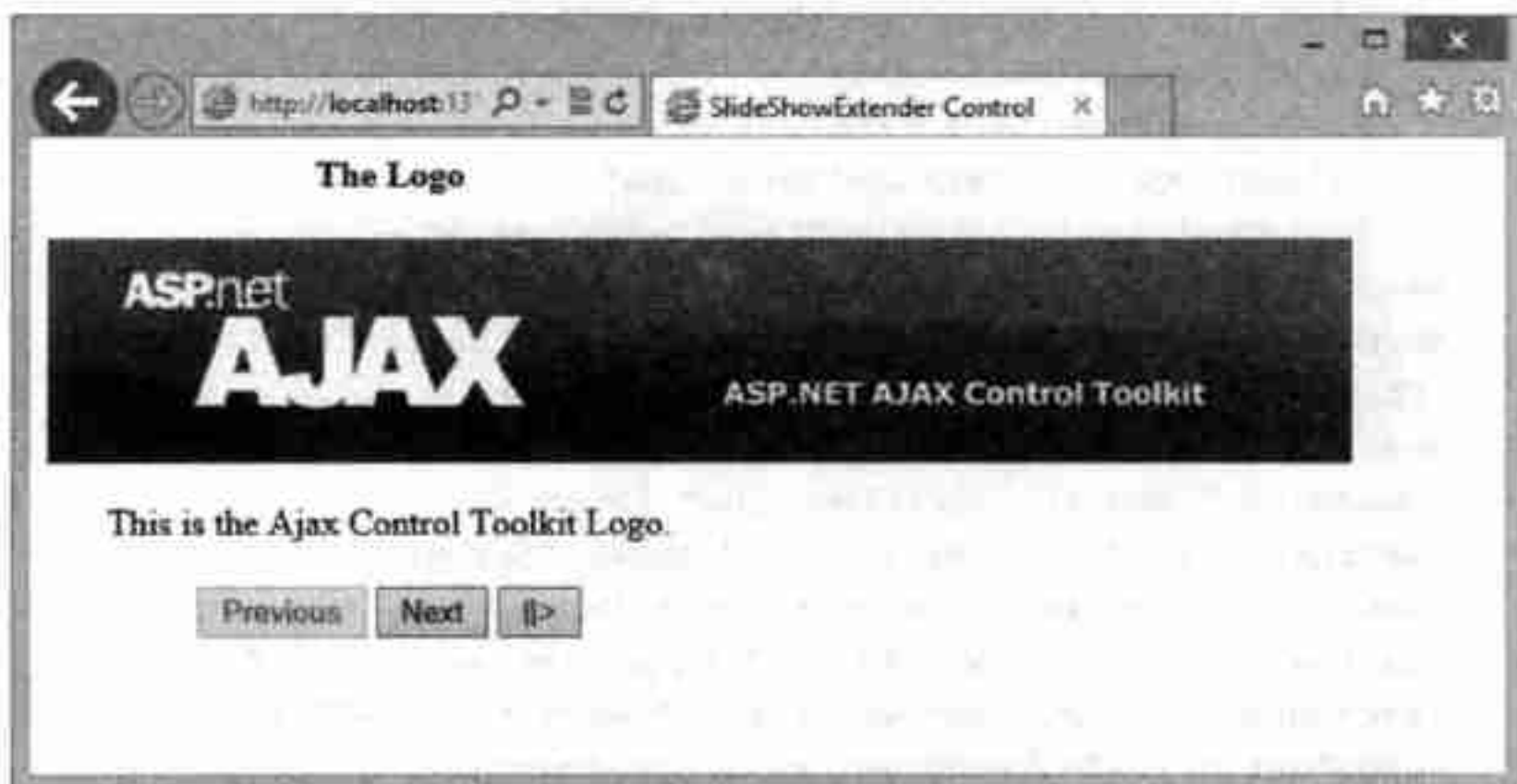


图 24-36

单击页面上的 Play 按钮，会按顺序播放图像，直到播放完所有图像为止。图像不会重复播放，除非把 SlideShowExtender 控件的 Loop 属性设置为 True(该属性默认设置为 False)。

另一个需要注意的重要属性是 PlayInterval。这个属性的值是整数，表示浏览器改为显示图像系列中的下一张照片所花费的毫秒数。该属性默认设置为 3000 毫秒。

25. TextBoxWatermarkExtender 控件

TextBoxWatermarkExtender 控件可以为终端用户把使用说明放在控件中，使终端用户更好地理解控件的用途。该使用说明可以是文本或图像(使用 CSS)。程序清单 24-32 是对 TextBox 服务器控件的使用说明。

程序清单 24-32 将 TextBoxWatermarkExtender 控件用于 TextBox 控件

```

<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">

```

```

<title>TextBoxWatermarkExtender Control</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
      </asp:ToolkitScriptManager>
      <asp:TextBoxWatermarkExtender ID="TextBoxWatermarkExtender1"
        runat="server" WatermarkText="Enter in something here!"
        TargetControlID="TextBox1">
      </asp:TextBoxWatermarkExtender>
      <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    </div>
  </form>
</body>
</html>

```

在这个例子中，将 `TextBoxWatermarkExtender` 控件关联到一个简单的 `TextBox` 控件，并且使用 `WatermarkText` 属性提供要显示在该 `TextBox` 控件中的文本。图 24-37 是这段代码的执行结果。

Enter in something here!

图 24-37

图 24-37 中的文本是纯文本，并没有在 `TextBox` 控件中设置任何样式。当终端用户单击 `TextBox` 控件的内部时，该文本就会消失，光标将位于文本框的开头。

要给用作水印的内容应用某种样式，可以使用 `WatermarkCssClass` 属性。修改代码以包含某种样式，如程序清单 24-33 所示。

程序清单 24-33 给水印应用样式

```

<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
  Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
  <title>TextBoxWatermarkExtender Control</title>
  <style type="text/css">
    .watermark
    {
      width:150px;
      font:Verdana;
      font-style:italic;
      color:GrayText;
    }
  </style>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
      </asp:ToolkitScriptManager>

```



```

    <asp:TextBoxWatermarkExtender ID="TextBoxWatermarkExtender1"
        runat="server" WatermarkText="Enter in something here!"
        TargetControlID="TextBox1"
        WatermarkCssClass="watermark">
    </asp:TextBoxWatermarkExtender>
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
</div>
</form>
</body>
</html>

```

这一次使用了 `WatermarkCssClass` 属性,它指向页面上的内联 CSS 类 `watermark`。运行这个页面,可以看到应用的样式,如图 24-38 所示。



图 24-38

26. ToggleButtonExtender 控件

`ToggleButtonExtender` 控件处理 `CheckBox` 控件,允许用户使用自己的图像替代 `CheckBox` 控件常用的标准复选框图像。使用 `ToggleButtonExtender` 控件可以指定图像的选中、未选中或禁用状态。程序清单 24-34 是使用这个控件的一个例子。

程序清单 24-34 使用 `ToggleButtonExtender` 控件

```

<%@ Page Language="C#" %>
<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>ToggleButtonExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
        </asp:ToolkitScriptManager>
        <asp:MutuallyExclusiveCheckBoxExtender
            ID="MutuallyExclusiveCheckBoxExtender1" runat="server"
            Key="MyCheckBoxes" TargetControlID="CheckBox1">
        </asp:MutuallyExclusiveCheckBoxExtender>
        <asp:MutuallyExclusiveCheckBoxExtender
            ID="MutuallyExclusiveCheckBoxExtender2" runat="server"
            Key="MyCheckBoxes" TargetControlID="CheckBox2">
        </asp:MutuallyExclusiveCheckBoxExtender>
        <asp:ToggleButtonExtender ID="ToggleButtonExtender1"
            runat="server" TargetControlID="CheckBox1"
            UncheckedImageUrl="Images/Unchecked.gif"
            CheckedImageUrl="Images/Checked.gif"
            CheckedImageAlternateText="Checked"
            UncheckedImageAlternateText="Not Checked" ImageWidth="25"
            ImageHeight="25">
        </asp:ToggleButtonExtender>
        <asp:CheckBox ID="CheckBox1" runat="server">

```

```

        Text="&nbsp;Option One" />
<asp:ToggleButtonExtender ID="ToggleButtonExtender2"
    runat="server" TargetControlID="CheckBox2"
    UncheckedImageUrl="Images/Unchecked.gif"
    CheckedImageUrl="Images/Checked.gif"
    CheckedImageAlternateText="Checked"
    UncheckedImageAlternateText="Not Checked" ImageWidth="25"
    ImageHeight="25">
</asp:ToggleButtonExtender>
<asp:CheckBox ID="CheckBox2" runat="server"
    Text="&nbsp;Option Two" />
</div>
</form>
</body>
</html>

```

这个页面包含两个 `CheckBox` 控件。每个复选框都关联了一个 `ToggleButtonExtender` 控件，并使用 `MutuallyExclusiveCheckBoxExtender` 控件将这两个复选框关联在一起。`ToggleButtonExtender` 控件使用 `CheckedImageUrl` 和 `UncheckedImageUrl` 属性指定要使用的对应图像。另外，如果终端用户的浏览器禁用图像，就使用 `CheckedImageAlternateText` 和 `UncheckedImageAlternateText` 属性提供的文本来替代图像。还必须为 `ImageWidth` 和 `ImageHeight` 属性指定值，页面才能运行。

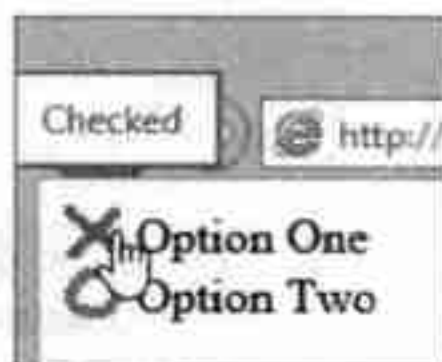


图 24-39

运行这个页面，结果如图 24-39 所示。

27. UpdatePanelAnimationExtender 控件

在刷新内容时给 `UpdatePanel` 控件设置动画是一种常见的需求。`UpdatePanelAnimationExtender` 控件允许使用工具集中的许多动画功能，并在更新指定的 `UpdatePanel` 控件或更新完成时自动播放动画。

`UpdatePanelAnimationExtender` 控件可以为两个特定的事件把动画应用于 `Panel` 控件。第一个事件是 `OnUpdating`，第二个事件是 `OnUpdated`。然后可以使用 AJAX 控件工具集提供的动画架构，根据这两个事件改变页面的样式。程序清单 24-35 显示了一个例子，在终端用户单击页面上 `UpdatePanel` 控件包含的 `Calendar` 控件中的某个特定日期时，使用 `OnUpdated` 事件。

程序清单 24-35 为 OnUpdated 事件使用动画

```

<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<script runat="server">
    protected void Calendar1_SelectionChanged(object sender, EventArgs e)
    {
        Label1.Text = "The date selected is " +
            Calendar1.SelectedDate.ToLongDateString();
    }
</script>

```



```

<!DOCTYPE html>
<html>
<head runat="server">
    <title>UpdatePanelAnimationExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>
            <asp:UpdatePanelAnimationExtender
                ID="UpdatePanelAnimationExtender1"
                runat="server" TargetControlID="UpdatePanel1">
                <Animations>
                    <OnUpdated>
                        <Sequence>
                            <Color PropertyKey="background"
                                StartValue="#999966"
                                EndValue="#FFFFFF" Duration="5.0" />
                        </Sequence>
                    </OnUpdated>
                </Animations>
            </asp:UpdatePanelAnimationExtender>
            <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                <ContentTemplate>
                    <asp:Label ID="Label1" runat="server"></asp:Label>
                    <br />
                    <asp:Calendar ID="Calendar1" runat="server"
                        onselectionchanged="Calendar1_SelectionChanged">
                    </asp:Calendar>
                </ContentTemplate>
            </asp:UpdatePanel>
        </div>
    </form>
</body>
</html>

```

有了这段代码，在单击 Calendar 控件中的某个日期时，包含日历的 UpdatePanel 控件就会根据动画中的设定，在 5 秒的时间内改变其背景色。定义的动画可以很复杂，而制作华丽的动画也超出了本章的讨论范围。

28. ValidatorCalloutExtender 控件

最后一个控件是 ValidatorCalloutExtender，这个控件可以为使用表单的终端用户添加更容易引起注意的有效性验证消息。这个控件不是与要验证的控件关联起来，而是与验证控件关联起来。程序清单 24-36 中就把 ValidatorCalloutExtender 控件与 RegularExpressionValidator 控件关联起来。

程序清单 24-36 使用 ValidatorCalloutExtender 控件创建验证结果

```

<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"

```



```

Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
  <title>ValidatorCalloutExtender Control</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
      </asp:ToolkitScriptManager>
      <asp:ValidatorCalloutExtender ID="ValidatorCalloutExtender1"
        runat="server" TargetControlID="RegularExpressionValidator1">
      </asp:ValidatorCalloutExtender>
      Email Address: &nbsp;
      <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
      <asp:RegularExpressionValidator
        ID="RegularExpressionValidator1" runat="server"
        ErrorMessage="You must enter an email address" Display="None"
        ControlToValidate="TextBox1"
        ValidationExpression=
          "\w+([-+.']\w+)*@\w+([-.\w+)*\.\w+([-.\w+)*">
      </asp:RegularExpressionValidator><br />
      <asp:Button ID="Button1" runat="server" Text="Submit" />
    </div>
  </form>
</body>
</html>

```

这个页面包含用于表单的一个文本框、一个 Submit 按钮和一个 RegularExpressionValidator 控件。按正常方式创建 RegularExpressionValidator 控件，但使用了 Display 属性，并把它设置为 None。我们不喜欢一般的 ASP.NET 验证控件也显示消息，因为这会与 ValidatorCalloutExtender 控件显示的消息冲突。尽管 Display 属性设置为 None，但仍然使用 ErrorMessage 属性提供错误消息。运行这个页面的结果如图 24-40 所示。

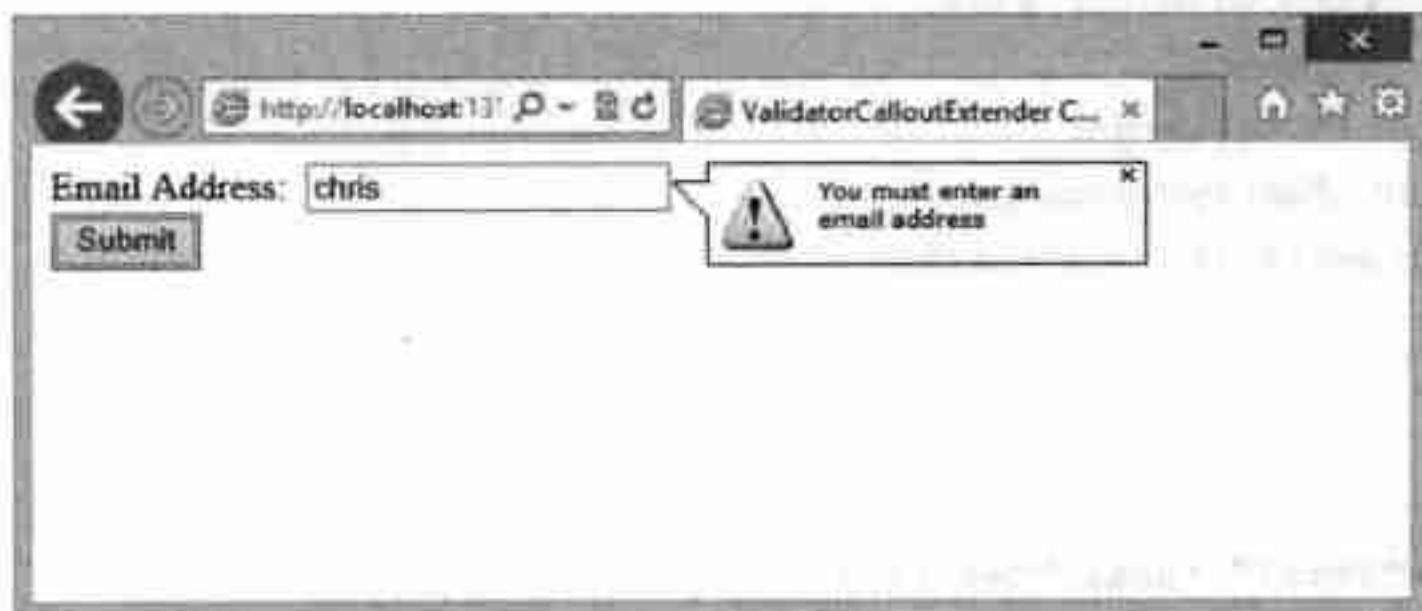


图 24-40

24.2.2 AJAX 控件工具集中的服务器控件

下一组 ASP.NET AJAX 控件并不总是扩展其他 ASP.NET 控件，而是扩展它们自己。下面将详细介绍这些控件。

1. Accordion 控件

Accordion 控件用于指定一组窗格，类似于 Microsoft Outlook 中的导航菜单。每个窗格都由标题模板和内容模板组成。所有窗格的标题模板都是可见的，但只有一个内容模板是可见的。用户通过单击标题来指定显示哪个窗格。以前的活动窗格的内容会隐藏起来，而新选择的窗格的内容会显示出来。

在活动窗格之间切换时，Accordion 控件可以提供淡入淡出切换效果。这需把 FadeTransitions 属性设置为 True，再设置 TransitionDuration 和 FramesPerSecond 属性的值。这两个属性的默认值分别是 250 毫秒和 40 帧/秒。

SelectedIndex 属性可以通过声明方式和编程方式控制显示哪个窗格。其他重要的属性有 AutoSize 和 Height。AutoSize 属性默认设置为 None，表示 Accordion 控件的大小根据活动窗格的大小而改变。屏幕上的其他内容会缩小，以适应变化的尺寸。但把 AutoSize 属性设置为 Limit 时，控件的大小会由 Height 值决定。如果内容大于可用的空间，活动窗格就会显示滚动栏。AutoSize 属性的另一个值是 Fill，意味着如果此窗格的内容不够多，就会扩大此窗格，达到足以满足 Height 属性所定义的大小。程序清单 24-37 中的 Accordion 控件用于两个窗格。

程序清单 24-37 带两个 AccordionPane 控件的 Accordion 控件

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>Accordion Control</title>
    <style type="text/css">
        .titlebar
        {
            background-color:Blue;
            color:White;
            font-size:large;
            font-family:Verdana;
            border:solid 3px Black;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>
            <asp:Accordion ID="Accordion1" runat="server" HeaderCssClass="titlebar"
                HeaderSelectedCssClass="titlebar"
                FadeTransitions="true"
                TransitionDuration="333">
```

```

FramesPerSecond="30">
  <Panels>
    <asp:AccordionPanel runat="server" ID="AccordionPanel1">
      <Header>
        This is the first pane
      </Header>
      <Content>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit.
        Donec accumsan lorem. Ut consectetur tempus metus.
        Aenean tincidunt venenatis tellus. Suspendisse molestie
        cursus ipsum. Curabitur ut lectus. Nulla ac dolor nec elit
        convallis vulputate. Nullam pharetra pulvinar nunc. Duis
        orci. Phasellus a tortor at nunc mattis congue.
        Vestibulum porta tellus eu orci. Suspendisse quis massa.
        Maecenas varius, erat non ullamcorper nonummy, mauris erat
        eleifend odio, ut gravida nisl neque a ipsum. Vivamus
        facilisis. Cras viverra. Curabitur
        ut augue eget dolor semper posuere. Aenean at magna eu eros
        tempor pharetra. Aenean mauris.
      </Content>
    </asp:AccordionPanel>
    <asp:AccordionPanel runat="server" ID="AccordionPanel2">
      <Header>
        This is the second pane
      </Header>
      <Content>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit.
        Donec accumsan lorem. Ut consectetur tempus metus.
        Aenean tincidunt venenatis tellus. Suspendisse molestie
        cursus ipsum. Curabitur ut lectus. Nulla ac dolor nec elit
        convallis vulputate. Nullam pharetra pulvinar nunc. Duis
        orci. Phasellus a tortor at nunc mattis congue.
        Vestibulum porta tellus eu orci. Suspendisse quis massa.
        Maecenas varius, erat non ullamcorper nonummy, mauris erat
        eleifend odio, ut gravida nisl neque a ipsum. Vivamus
        facilisis. Cras viverra. Curabitur
        ut augue eget dolor semper posuere. Aenean at magna eu eros
        tempor pharetra. Aenean mauris.
      </Content>
    </asp:AccordionPanel>
  </Panels>
</asp:Accordion>
</div>
</form>
</body>
</html>

```

titlebar 类是在文档中定义的单个 CSS 类，用作 HeaderCssClass 和 HeaderSelectedCssClass 属性的值。这里的 Accordion 控件包含两个 AccordionPanel 控件。AccordionPanel 控件的子元素是<Header>和<Content>。放在<Header>部分的项位于可单击的窗格标题上，而包含在<Content>部分的项是隐藏

的，在选择相关的标题时才会显示出来。

注意在切换窗格时还有切换效果。运行这个页面的结果如图 24-41 所示。

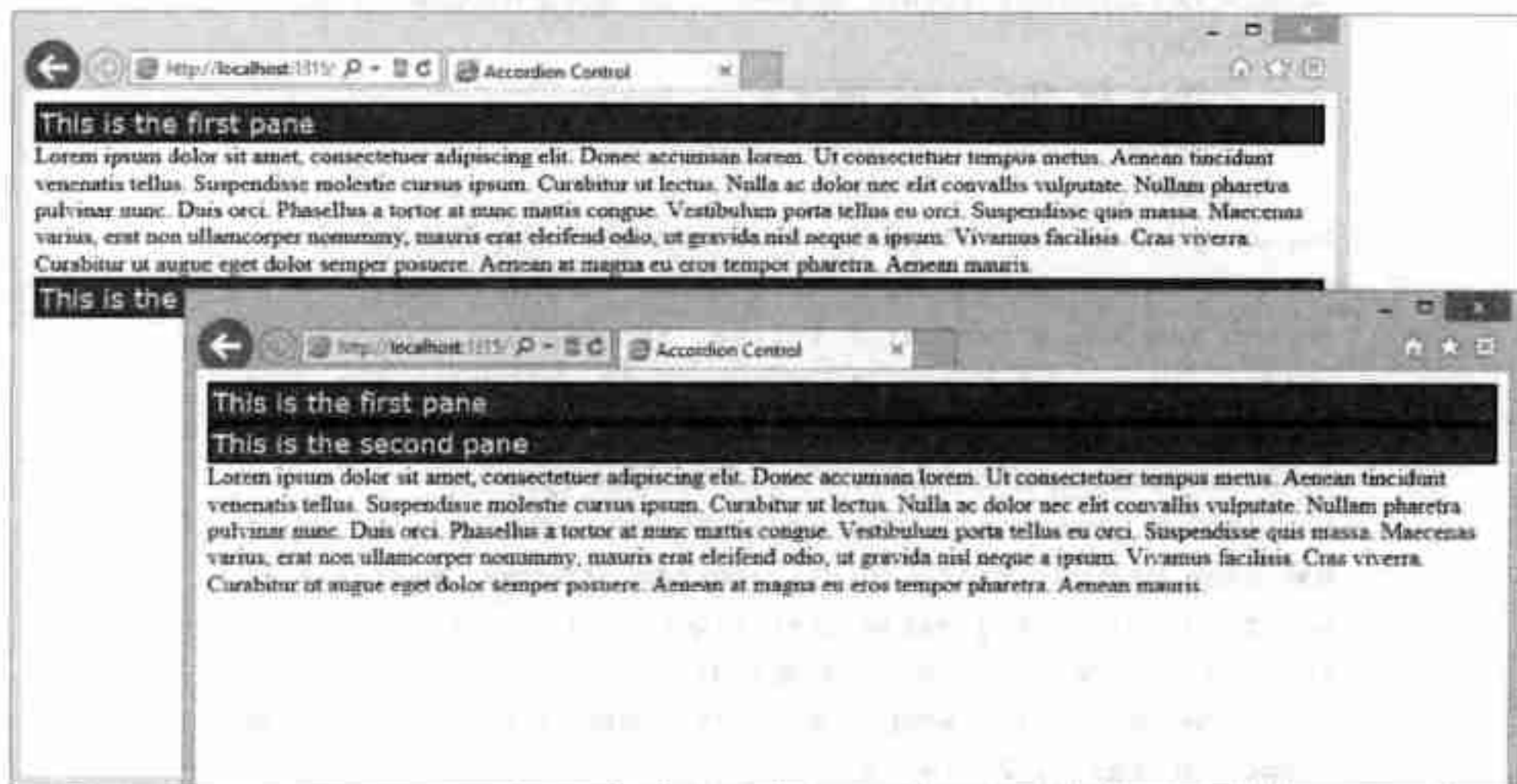


图 24-41

图 24-41 显示了选择每个窗格的屏幕截图。一些比较重要的属性如下：

- **AutoSize**: 定义控件如何处理尺寸的变化。其值可以是 None、Fill 和 Limit，默认值为 None。使用默认值时，控件下面的项会移动，为控件的伸展提供空间。值 Fill 要使用 Height 属性，控件会填充到需要的高度。这表示一些窗格必须变大，以包含更多的空间；而一些窗格要变小，并添加一个滚动栏，以适应高度受限的有限空间。值 Limit 也要使用 Height 属性，只是控件不会变得比这个高度值还大。窗格可能比指定的高度小。
- **TransitionDuration**: 切换到另一个窗格所需的毫秒数。
- **FramesPerSecond**: 用于切换到另一个窗格的每秒帧数。
- **RequiredOpenedPane**: 指定至少一个窗格需要总是打开。这个属性的默认值是 True。值 False 表示所有的窗格都可以折叠。

最后，DataSource、DataSourceID 和 DataMember 属性可以在代码中绑定这个控件。

2. CascadingDropDown 控件

DropDownList 控件中可用的选项可能基于在另一个 DropDownList 控件中的选择。CascadingDropDown 控件很容易在应用程序中实现这个功能。可以把 TargetControlID 属性的值设置为 DropDownList 控件，并通过回调服务器来填充 DropDownList 控件。还可以指定类别来对 DropDownList 控件分类。

在填充 DropDownList 控件之前，会显示 PromptText 属性的值。而且，在调用服务器时会显示 LoadingText 属性的值。可以将 ServicePath 属性的值设置为在不同的 Web 服务中调用 ServiceMethod，或把 ServiceMethod 名称设置为页面上的静态 ScriptMethod，如程序清单 24-38 所示。

这个例子中的第一个 DropDownList 控件允许用户选择州，在这个例子中只包含 Missouri 和 Oregon 州。选择了某个州后，就根据用户在第一个 DropDownList 控件中选择的值填充第二个 DropDownList 控件。为了指定一个 DropDownList 控件依赖于另一个 DropDownList 控件的值，应设

置 CascadingDropDown 控件的 ParentControlID 属性。

程序清单 24-38 使用 CascadingDropDown 控件

```
using System.Web.Services;
using AjaxControlToolkit;
<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<script runat="server" language="C#">

    [WebMethod]
    [System.Web.Script.Services.ScriptMethod]
    public static CascadingDropDownNameValue[]
        GetStates(string knownCategoryValues, string category)
    {
        return new[] {
            new CascadingDropDownNameValue("Missouri", "Missouri"),
            new CascadingDropDownNameValue("Oregon", "Oregon") };
    }

    [WebMethod]
    [System.Web.Script.Services.ScriptMethod]
    public static CascadingDropDownNameValue[]
        GetCounties(string knownCategoryValues, string category)
    {
        if(knownCategoryValues.Contains("Missouri"))
        {
            return new[] {
                new CascadingDropDownNameValue("St. Charles", "St. Charles"),
                new CascadingDropDownNameValue("St. Louis", "St. Louis"),
                new CascadingDropDownNameValue("Jefferson", "Jefferson"),
                new CascadingDropDownNameValue("Warren", "Warren"),
                new CascadingDropDownNameValue("Franklin", "Franklin") };
        }
        if(knownCategoryValues.Contains("Oregon"))
        {
            return new[] {
                new CascadingDropDownNameValue("Baker", "Baker"),
                new CascadingDropDownNameValue("Benton", "Benton"),
                new CascadingDropDownNameValue("Clackamas", "Clackamas"),
                new CascadingDropDownNameValue("Clatsop", "Clatsop"),
                new CascadingDropDownNameValue("Columbia", "Columbia") };
        }
        return null;
    }

</script>

<!DOCTYPE html>
<html>
<head id="Head1" runat="server">
    <title>CascadingDropDown</title>
</head>
<body>
```



```
<form id="form1" runat="server">
  <asp:ToolkitScriptManager runat="server" ID="scriptManager" />
  <div>
    <asp:DropDownList runat="server" ID="ddl1" Width="200" />
    <br />
    <asp:DropDownList runat="server" ID="ddl2" Width="200" />
    <br />
    <asp:CascadingDropDown runat="server" ID="cddl"
      TargetControlID="ddl1"
      PromptText="Select a State"
      Category="state" LoadingText="[Loading States]"
      ServiceMethod="GetStates" />
    <asp:CascadingDropDown runat="server" ID="cddl2"
      TargetControlID="ddl2"
      ParentControlID="ddl1"
      PromptText="Select County" Category="county"
      LoadingText="[Loading Counties]"
      ServiceMethod="GetCounties" />
  </div>
</form>
</body>
</html>
```

3. NoBot 控件

NoBot 控件用于确定实体如何与表单交互，帮助确保是真正的人在使用表单，而不是一些自动化代码在操作应用程序。

NoBot 控件的使用如程序清单 24-39 所示(本章下载代码中的 NoBot.aspx)。

程序清单 24-39 使用 NoBot 控件限制登录表单

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="NoBot.aspx.cs" Inherits="NoBot" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
  <head id="Head1" runat="server">
    <title>NoBot Control</title>
  </head>
  <body>
    <form id="form1" runat="server">
      <div>
        <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
        </asp:ToolkitScriptManager>
        <asp:NoBot ID="NoBot1" runat="server" CutoffMaximumInstances="3"
          CutoffWindowSeconds="15" ResponseMinimumDelaySeconds="10"
          OnGenerateChallengeAndResponse="NoBot1_GenerateChallengeAndResponse" />
        <asp:Login ID="Login1" runat="server">
        </asp:Login>
        <asp:Label ID="Label1" runat="server"></asp:Label>
      </div>
    </form>
  </body>
```



```
</html>
```

NoBot 控件有 3 个重要属性,在控制表单的提交方式时应注意这些属性。这些属性分别是 CutoffMaximumInstances、CutoffWindowSeconds 和 ResponseMinimumDelaySeconds。

CutoffMaximumInstances 是在 CutoffWindowSeconds 属性指定的时间内(秒数)允许终端用户尝试提交表单的次数。ResponseMinimumDelaySeconds 属性定义了终端用户必须提交表单的最少秒数。如果知道所使用的表单需要花费一些时间,就应把这个属性设置为某个阻止人们提交表单的时间值(即使是 5 秒)。

OnGenerateChallengeAndResponse 属性可以定义处理复杂问题的服务器端方法,并根据该复杂问题提供响应。在程序清单 24-39 中使用了这个属性,并向用户回送表单提交的状态。

这个页面的隐藏代码如程序清单 24-40 所示(本章下载代码中的 NoBot.aspx.cs)。

程序清单 24-40 NoBot 控件的 OnGenerateChallengeAndResponse 属性的隐藏代码页面

```
using System;
using AjaxControlToolkit;

public partial class NoBot : System.Web.UI.Page
{
    protected void NoBot1_GenerateChallengeAndResponse(object sender,
        AjaxControlToolkit.NoBotEventArgs e)
    {
        NoBotState state;
        NoBot1.IsValid(out state);

        Label1.Text = state.ToString();
    }
}
```

运行这个页面,尝试在 10 秒钟内提交表单,这会使提交无效。另外,尝试在 15 秒钟内超过 3 次提交表单,也会使提交无效。

4. PasswordStrength 控件

PasswordStrength 控件可以检查 TextBox 控件中密码的内容,并验证保密强度。此外还给终端用户提供一条消息,告知保密强度是否合理。PasswordStrength 控件的一个简单例子如程序清单 24-41 所示。

程序清单 24-41 使用 PasswordStrength 控件和 TextBox 控件

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>Password Strength Control</title>
</head>
```

```

<body>
  <form id="form1" runat="server">
    <div>
      <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
      </asp:ToolkitScriptManager>
      <asp>PasswordStrength ID="PasswordStrength1" runat="server"
        TargetControlID="TextBox1">
      </asp>PasswordStrength>
      <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    </div>
  </form>
</body>
</html>

```

这个简单的页面生成了一个文本框，当终端用户开始在文本框中输入时，页面会通知用户密码的保密强度如何，如图 24-42 所示。

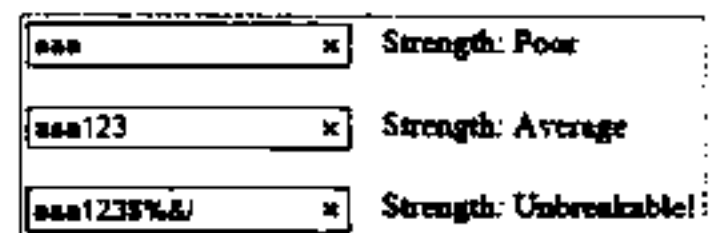


图 24-42

这里可以使用的重要属性有 `MinimumLowerCaseCharacters`、`MinimumNumericCharacters`、`MinimumSymbolCharacters`、`MinimumUpperCaseCharacters` 和 `PreferredPasswordLength`。

5. Rating 控件

`Rating` 控件允许终端用户查看和设置等级(如星级)。可以控制等级数、已填充等级的外观、空等级的外观等。程序清单 24-42 中的页面显示了一个 5 星级系统，终端用户可以自己设置等级。

程序清单 24-42 终端用户可以操作的 `Rating` 控件

```

<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
  Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
  <title>Rating Control</title>
  <style type="text/css">
    .ratingStar {
      font-size: 0pt;
      width: 13px;
      height: 12px;
      margin: 0px;
      padding: 0px;
      cursor: pointer;
      display: block;
      background-repeat: no-repeat;
    }

    .filledRatingStar {
      background-image: url(Images/FilledStar.png);
    }
  </style>
</head>
<body>
  <asp:Rating ID="Rating1" runat="server">
    <asp:RatingStar ID="RatingStar1" runat="server">
    </asp:RatingStar>
  </asp:Rating>
</body>
</html>

```

```

        .emptyRatingStar {
            background-image: url(Images/EmptyStar.png);
        }

        .savedRatingStar {
            background-image: url(Images/SavedStar.png);
        }
    }
</style>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>
            <asp:Rating ID="Rating1" runat="server"
                StarCssClass="ratingStar"
                WaitingStarCssClass="savedRatingStar"
                FilledStarCssClass="filledRatingStar"
                EmptyStarCssClass="emptyRatingStar">
            </asp:Rating>
        </div>
    </form>
</body>
</html>

```

此处, Rating 控件使用许多 CSS 类定义它在各种状态下的外观和行为。除了 CSS 类属性(StarCssClass、WaitingStarCssClass、FilledStarCssClass 和 EmptyCssClass)之外,还可以指定等级对齐、等级项数(默认为 5)、宽度、当前等级等。程序清单 24-42 中的代码生成了如图 24-43 所示的结果。



图 24-43

6. TabContainer 控件

TabContainer 控件和 TabPanel 控件可以显示我们熟悉的选项卡式用户界面。它们会在窗格的顶部显示一组选项卡,并且窗格中显示的是活动选项卡的内容。用户选择另一个选项卡时,内容就会改变。选项卡是控制有许多内容的页面的一种优秀方法。TabContainer 控件可以包含一个或多个 TabPanel 控件,提供一系列选项卡,一次显示一个选项卡的内容。

TabContainer 控件允许关联服务器事件 ActiveTabChanged,如果活动选项卡改变,就会在回送过程中触发这个事件。还可以使用 OnClientActiveTabChanged 事件,在用户选择另一个选项卡时,在浏览器中触发 JavaScript 事件。ScrollBars 属性可以指定滚动栏是否应为 Horizontal、Vertical、Both、None 或 Auto(在设置为 Auto 时,由控件决定如何显示滚动栏)。

TabPanel 控件给选项卡标题提供了<HeadTemplate>元素,给选项卡的内容提供了<ContentTemplate>元素。可以先使用<HeadTemplate>元素,再指定 HeaderText 属性。该控件还有一个事件 OnClientClick,在选择选项卡时触发该事件。选项卡的一个有趣功能是可以把 Enabled 属性设置为 False,在浏览器中使用 JavaScript 以编程方式禁用选项卡。

程序清单 24-43 是带 3 个 TabPanel 控件的 TabContainer 控件的例子。

程序清单 24-43 在 TabContainer 控件中显示 3 个选项卡

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>TabContainer Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
            </asp:ToolkitScriptManager>
            <asp:TabContainer ID="TabContainer1" runat="server"
                Height="300px">
                <asp:TabPanel runat="server">
                    <HeaderTemplate>Tab 1</HeaderTemplate>
                    <ContentTemplate>
                        Here is some tab one content.
                    </ContentTemplate>
                </asp:TabPanel>
                <asp:TabPanel runat="server">
                    <HeaderTemplate>Tab 2</HeaderTemplate>
                    <ContentTemplate>
                        Here is some tab two content.
                    </ContentTemplate>
                </asp:TabPanel>
                <asp:TabPanel runat="server">
                    <HeaderTemplate>Tab 3</HeaderTemplate>
                    <ContentTemplate>
                        Here is some tab three content.
                    </ContentTemplate>
                </asp:TabPanel>
            </asp:TabContainer>
        </div>
    </form>
</body>
</html>
```

这个简单页面的结果如图 24-44 所示。



图 24-44

24.3 本章小结

利用 AJAX 控件工具集很容易给 Web 应用程序添加丰富的动画和交互操作功能。除了可以使用 UpdatePanel 控件异步更新页面内容之外,还可以使用 UpdatePanelAnimation 控件为用户显示后台处理工作正在进行的动画效果。AJAX 控件工具集有助于模糊桌面应用程序和 Web 应用程序之间的界限。模态对话框和弹出对话框增强了 Web 应用程序的功能,使其超出了用户对使用浏览器工作的期望。

可以看出, AJAX 控件工具集有许多控件,这是微软及其社区努力的成果, AJAX 控件的列表在以后还会逐步增大。

本章介绍了 AJAX 控件工具集中的许多控件,以及如何在 ASP.NET 应用程序中使用它们。应经常访问微软 AJAX 页面和这些控件的 AJAX 控件工具集站点,以利用最新的控件。

第 25 章

jQuery

本章要点

- 下载并包含 jQuery
- 访问并操纵文档对象模型(DOM)
- 处理事件
- 对服务器触发 AJAX 调用
- 使用 UI 组件

开源 JavaScript 库 jQuery 是目前现代 JavaScript 开发的事实标准。根据 <http://trends.builtwith.com/javascript> 上的一份统计(2013 年 1 月), 在使用著名 JavaScript 库的前 1000000 个网站中, 超过 40% 选择了 jQuery。所以微软也包含 jQuery 就没有什么可惊讶的。

以前, 微软开发有自己的 JavaScript 项目: Microsoft Ajax Library。它目前仍是 ASP.NET Web Forms 的一部分(如果将 ScriptManager 控件添加到页面上, 就会自动加载 AJAX 库, 如第 23 章所述), 但从来都无法吸引非微软开发人员。因此, 微软决定采用市场的领头羊 jQuery, Visual Studio 2010 及以后版本附带的许多 Web 模板都带有 jQuery。

jQuery 是一个非常强大的库, 使无数常见的 JavaScript 任务很容易完成。jQuery 需要用整本书的篇幅来介绍, 这方面的畅销书有《jQuery 高级编程》。显然, 本章只能介绍 jQuery 的皮毛, 引导读者使用 JavaScript 库完成最常见的任务。更深入的讨论可查阅前面提及的图书或 jQuery 站点的富文档 <http://api.jquery.com/>。



本章的代码是用 jQuery 1.9 编写和测试的, 但应可以运行在 jQuery 1.7 之后的所有版本中(其中一些程序代码可以运行在更早的版本上)。

25.1 jQuery 简介

jQuery 首页是 <http://jquery.com/> (如图 25-1 所示), 在 2013 年 1 月进行了重新设计。jQuery 主要有三个特点:

- 轻型: 如果使用 HTTP 的 Gzip 进行压缩, 整个库约 32KB。
- 与 CSS3 兼容: 甚至在不支持 CSS3 的浏览器中(例如 IE6 以前的版本), 也支持大多数 CSS3 标准。
- 跨浏览器: jQuery 在最常见的浏览器中进行了全面测试, 包括 Internet Explorer、Firefox、Chrome、Opera 和 Safari。



图 25-1

要在 ASP.NET 应用程序中使用 jQuery, 需要把这个库添加到当前的网站或项目中。如果希望手工下载该库, 只需在 jQuery 首页上点击下载链接。但代码较难辨别, 如图 25-2 所示。原因是代码进行了优化, 以提高性能。其中删除了空白, 重命名了标识符, 使其更短。

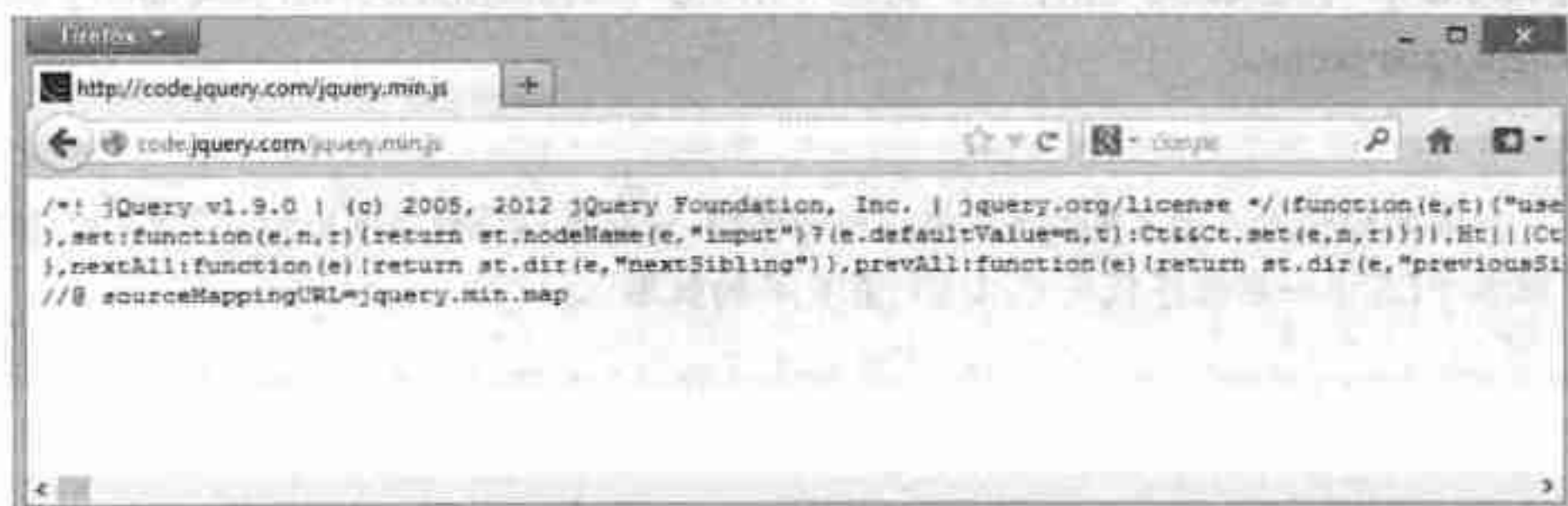


图 25-2

如果单击 Download Un-minified Copy 链接, 就可以下载可读性更好、添加了注释的代码版本。图 25-3 显示了该文件中的一部分 JavaScript。

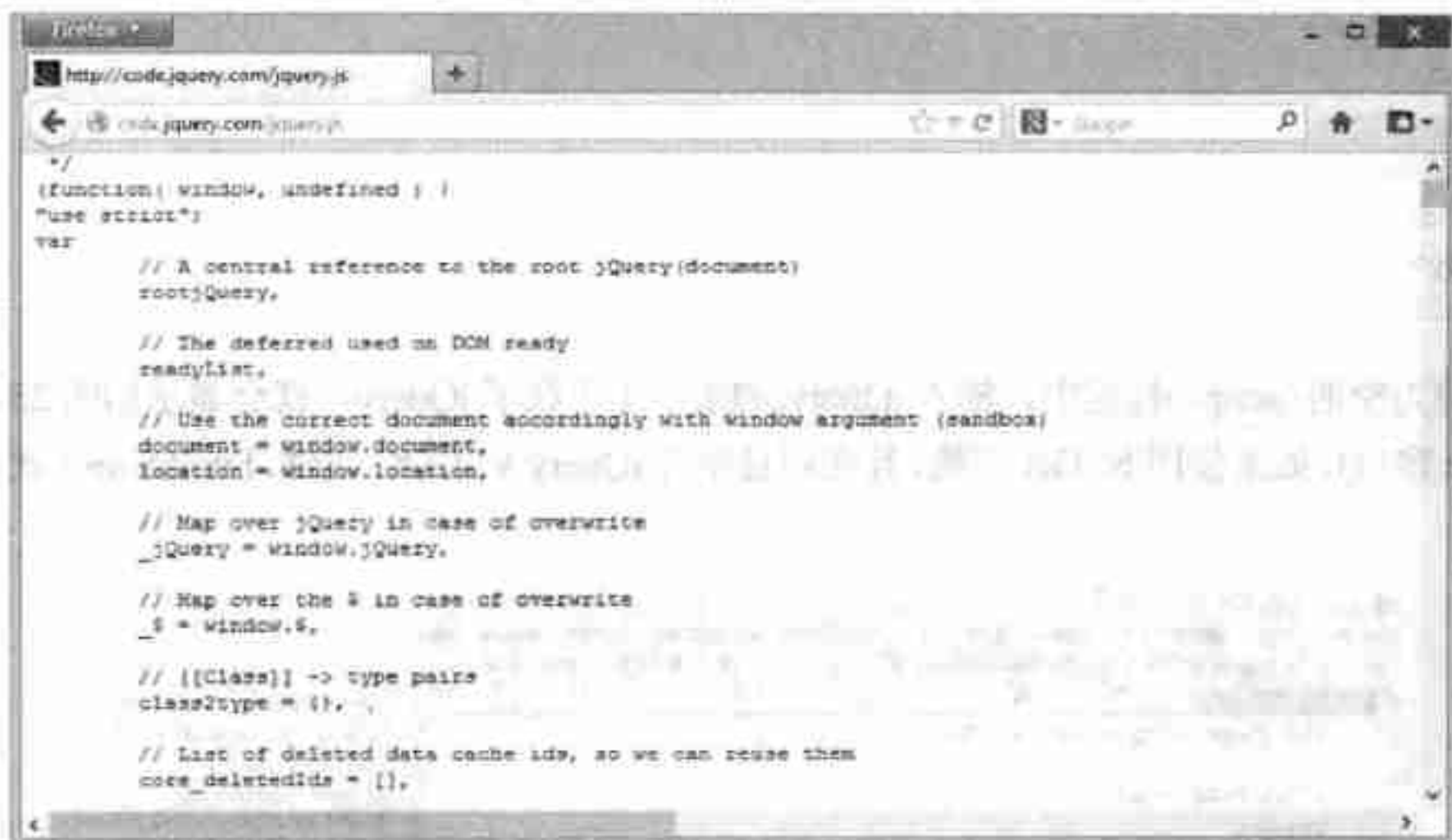


图 25-3

选择自己需要的文件版本(小版本的文件通常足以应付开发, 也应部署这个版本), 把它添加到站点或 Web 项目中。

要在项目中添加最新版本的 jQuery, 一种更方便的方式是使用 NuGet 包管理器。在 Package Manager Console 中, 执行程序清单 25-1 中的命令。

程序清单 25-1 通过 NuGet 安装 jQuery

```
Install-Package jQuery
```

过一会儿, 网站就将包含一个新的 Script 文件夹, 其中至少包含如下文件(假定 jQuery 1.9.0 是最新版本——读者尝试安装时, jQuery 的最新版本可能与此不同):

- jquery-1.9.0.js: 可读性更好、添加了注释的 jQuery 版本。
- jquery-1.9.0.min.js: jQuery 的压缩版本。
- jquery-1.9.0-vsdoc.js: Visual Studio 中 IntelliSense 的提示。

根据安装 jQuery 的方式和 jQuery 的当前版本, 文件的路径和名称可能与本章使用的有所不同。确保采用自己系统的文件名。

上述列表中的第三个文件比较有趣, 因为它可以使 jQuery 的使用更方便。为了演示这一点, 看看程序清单 25-2, 其中 JavaScript 库 jQuery 已被添加到带有<script>标记的页面上。

程序清单 25-2 加载 jQuery

```
<%@ Page Language="C#" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>jQuery</title>
```



```
<script type="text/javascript" src="Scripts/jquery.js"></script>
<script type="text/javascript">

</script>
</head>
<body>
    <div>
    </div>
</body>
</html>
```

在当前为空的<script>标记中, 输入jQuery(如果手工下载了jQuery, 就会显示如图 25-4 所示的 IntelliSense 窗口)。如果使用 NuGet 下载, 且在项目中有 jQuery VSDOC 文件, IntelliSense 就如图 25-5 所示。

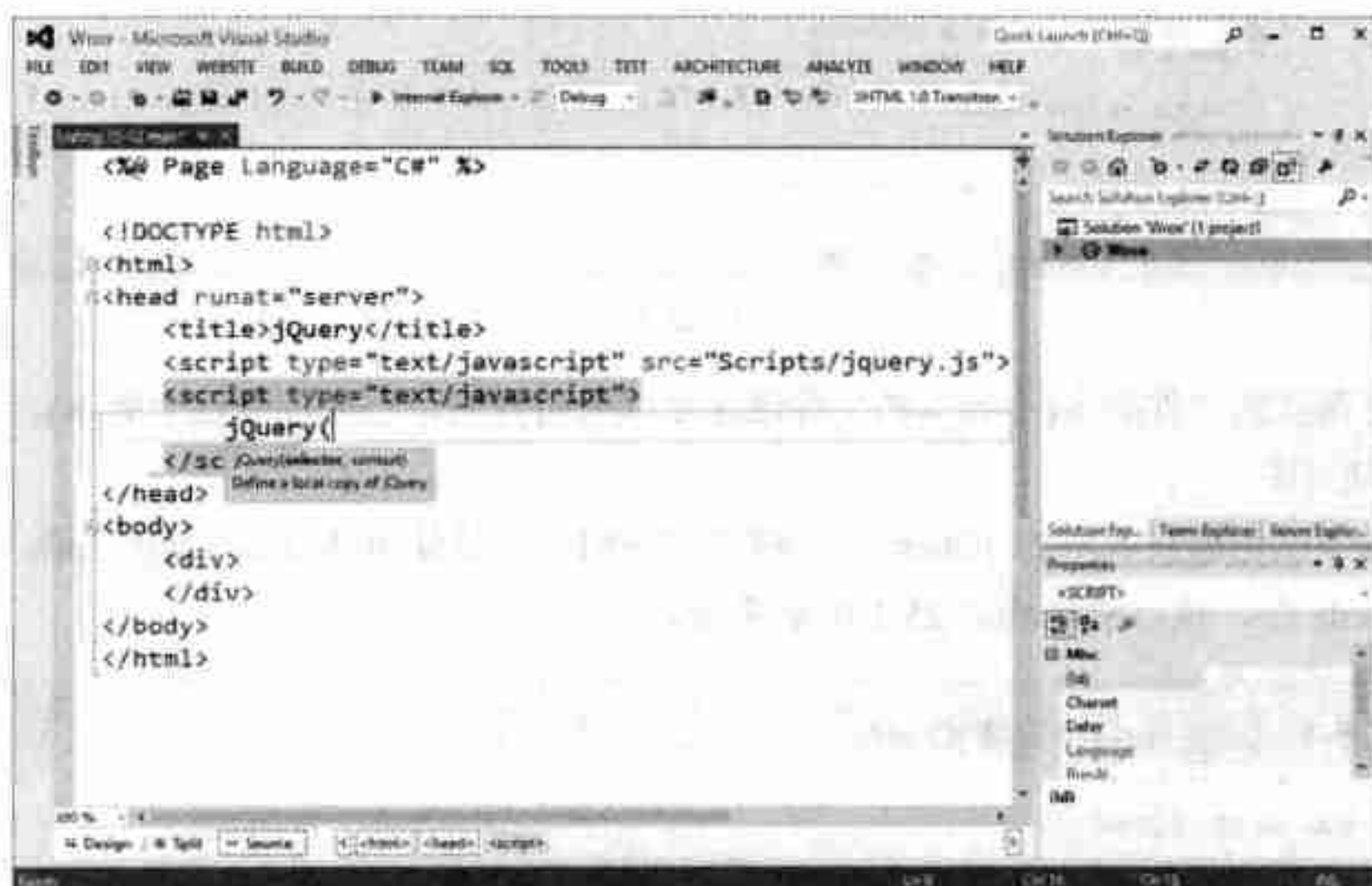


图 25-4

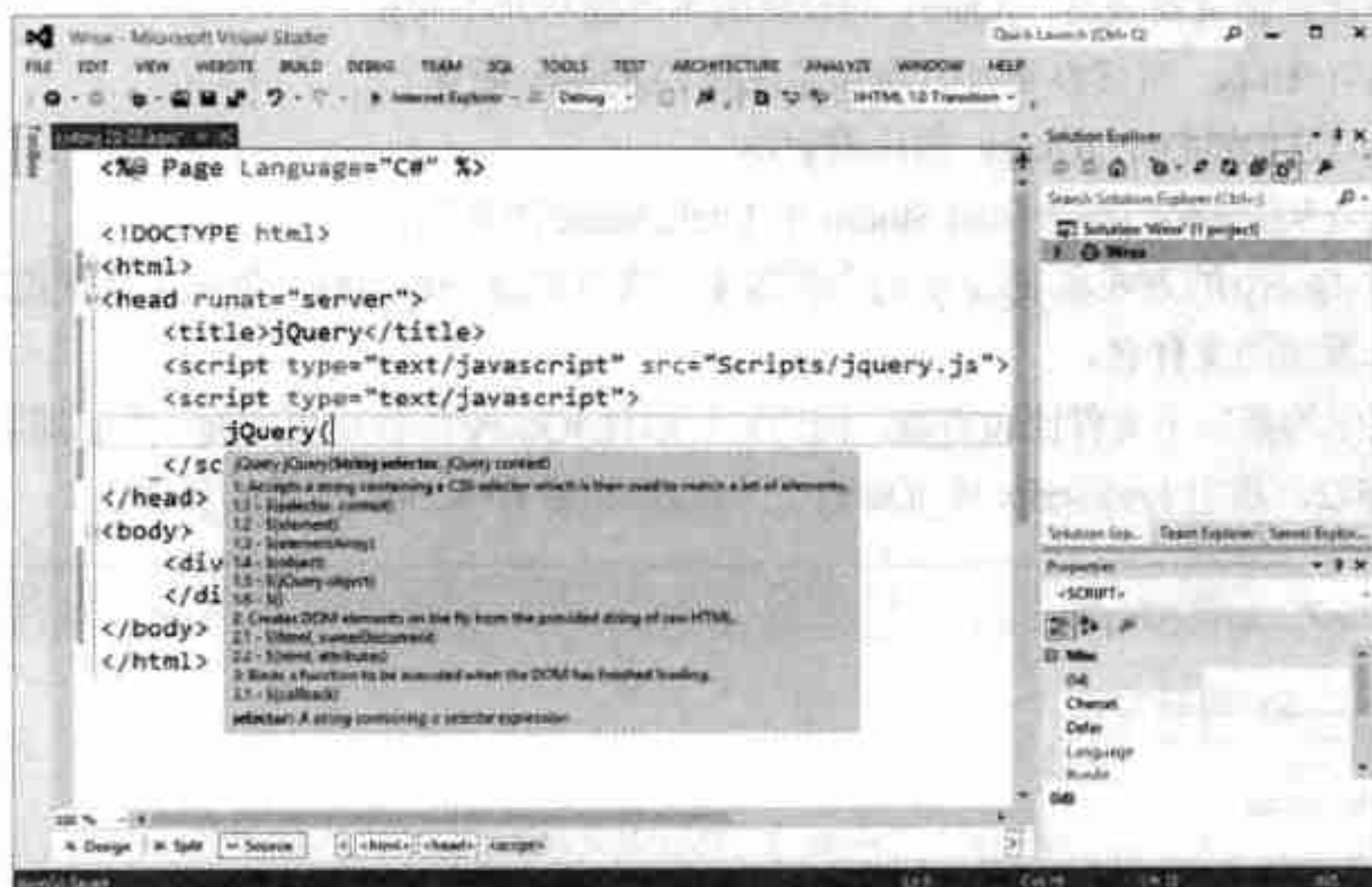


图 25-5

IntelliSense 不同的原因是, VSDOC 文件包含额外的类型提示和注释, 如图 25-5 所示。但即使手工下载了 jQuery, 也不要失望: 因为 VSDOC 文件可以单独下载。微软的内容交付网络(CDN)通常在发布了 jQuery 新版本后不久, 就带有 VSDOC 版本。http://www.asp.net/ajaxlibrary/cdn.ashx#jQuery_Releases_on_the_CDN_0 上显示了最新列表。

在深入了解如何使用 jQuery 之前, 本节先介绍大多数支持 jQuery 的站点中的一种常用模式。程序清单 25-3 说明了通常的设置: 首先, 加载 jQuery, 如果使用与此不同的版本, 那么可能要修改版本号。在下一个<script>块中, 开始使用 jQuery 代码。第一行有点奇怪:

```
$(function() {
```

该行以美元符号开头——这是在使用 PHP 吗? 不是, 在 JavaScript 中, 可以把\$用作标识符。因为不会有代码调用\$()函数, 所以 JavaScript 库使用\$()是不会有名称冲突的。

程序清单 25-3 使用 jQuery 的常见模式

```
<%@ Page Language="C#" %>

<!DOCTYPE html>
<html>
<head runat="server">
  <title>jQuery</title>
  <script type="text/javascript" src="Scripts/jquery.js"></script>
  <script type="text/javascript">
    $(function() {
      // DOM is ready
    });
  </script>
</head>
<body>
  <div>
  </div>
</body>
</html>
```

其实, \$()是 jQuery 的核心函数, 有几个作用。如果觉得这使代码难以读懂, 可以使用 jQuery 替代\$, 它们是等价的。



用 jQuery 编写的大多数代码都使用短格式。

如果查看图 25-5 中\$()/jQuery()的 IntelliSense 输出, 就会看到该函数的几个重载版本。第 3 个选项是 Binds a function to be executed when the DOM has finished loading(加载完 DOM 时, 绑定要执行的函数)。在 AJAX 应用程序中, 这是非常重要的。在大多数情况下, 我们都使用文档对象模型(DOM)访问页面上的元素。但这必须在完全加载 DOM 后才可行, 而完全加载 DOM 常常在 HTML 文档从服务器完全传递到浏览器后才完成。

所以, 如果把函数作为参数提供给\$()——JavaScript 不是一种很吸引人的语言——这个函数就在 DOM 准备好后执行。在大多数情况下, 这就是我们需要的。因此, 本章使用这种模式。



一段时间以前, 达到该效果(在 DOM 准备好后执行代码)的更常见(但比较繁琐)方式如下:

```
$(document).ready(function() {
    //...
});
```

目前这已是旧代码, 但它们仍有效。

25.2 选择元素

使用 JavaScript 库时常见的动作顺序如下: 等待一个事件(例如 DOM 加载完、用户单击按钮等); 然后选择页面上的一个元素; 最后, 读写该 HTML 元素。如果我们已知这些事件是如何工作的(DOM 已加载完), 就可以选择元素。幸好, jQuery 团队没有为这个任务开发全新的语法, 而是重用已建好的、用于相同目的的标准: CSS 选择器。

此 W3C 标准是 CSS 的一部分, 描述了页面上的特定元素如何选择。在 CSS 中, 选择过程的目的是指定元素的样式或位置。在 jQuery 中, 可以对这些元素进行任何处理。



该规范的当前版本(<http://www.w3.org/TR/css3-selectors/>)包含标准的完整列表, jQuery 支持其中的大多数标准。实际上, jQuery 非常聪明: 在旧浏览器上, JavaScript 库有自己的选择器引擎 Sizzle, 它也可以用作独立的项目(<http://sizzlejs.com/>)。这个选择器引擎解析 CSS 选择器, 返回一组元素。

大多数现代浏览器都内置了 JavaScript API, 以进行 CSS 选择。如果遇到这样的客户端, jQuery 就会把复杂的解析和选择任务委托给浏览器。

与 jQuery 一样, CSS 是非常复杂的主题。但是, 在大多数情况下, 只需使用几个常见的 CSS 选择器。CSS 元素选择器的三种常见类型是:

- 标记选择器: 例如, 选择器 `p` 匹配页面上所有的 `<p>` 元素, `a` 匹配 `<p>` 元素中所有的 `<a>` 元素。注意得到的 HTML 很重要, 而不是所用的 ASP.NET Web 控件。例如, 选择器 `Calendar` 就是无效的。
- 类选择器: 它们有前缀 `.`。例如, 选择器 `.myClass` 会选择具有 HTML 属性 `class="myClass"` 的所有元素。也可以混合匹配: `a.myClass` 会匹配带有类 `myClass` 的所有 `<a>` 标记, 而不匹配 `<p>` 标记。使用 ASP.NET 时, Web 控件的 `CssClass` 属性在得到的 HTML 中会变成 `class` 特性。
- ID 选择器: 如果 HTML 元素有 ID, 例如 `id="myID"`, `#myID` 选择器就匹配这个元素。



使用 ASP.NET Web Forms 时,这种方式通常很危险,因为 ASP.NET 默认会重写一些 ID。使用 ClientIDMode="Static"可能有帮助,或者利用控件的 ClientID 属性动态确定客户端 ID。但在大多数情况下,组合使用标记和类选择器是最佳方式。

如果调用 \$() 函数,并把 CSS 选择器用作参数,就会返回页面上匹配这个选择器的元素列表。准确地说,会得到一个 jQuery 对象,它表示所选的元素。区别是 jQuery 对象可以有方法,所以可以进一步处理这些选中的元素。本章的后面会使用这个功能。

现在只使用 jQuery 对象的一个方法:使用 size() 返回元素个数。程序清单 25-4 显示了一个简单的页面,其中包含几个元素。接着 JavaScript 代码选中几个元素,使用 JavaScript 的 alert() 方法输出结果,如图 25-6 所示。

程序清单 25-4 用 jQuery 选择元素

```
<%@ Page Language="C#" %>

<!DOCTYPE html>
<html>
<head runat="server">
  <title>jQuery</title>
  <script type="text/javascript" src="Scripts/jquery.js"></script>
  <script type="text/javascript">
    $(function() {
      var divs = $("div").size();
      var spans = $("span").size();
      alert(divs + " <div> elements\n" +
        spans + " <span> elements");
    });
  </script>
</head>
<body>
  <form runat="server">
    <div>
      <span>First label</span>
    </div>
    <asp:Panel ID="Panell" runat="server">
      <asp:Label ID="Labell" runat="server"
        Text="Second label" />
    </asp:Panel>
  </form>
</body>
</html>
```



图 25-6

图 25-6 中的结果可能令人惊讶。但是，`<asp:Panel>` 默认显示为 `<div>` 元素，`<asp:Label>` 变成 `` 元素。再看看得到的 HTML 标记，会发现隐藏的 `ViewState` 字段包含在另一个 `<div>` 元素中，所以结果是正确的。注意，起作用的是 HTML，而不是 ASP.NET 标记。

25.3 修改元素

我们很少计算选中了多少个元素；但通常知道获得了多少个元素。重要的是之后如何处理它们。jQuery 提供了完整的 CRUD(Create、Read、Update、Delete) 功能，但在大多数情况下，我们仅读写元素，或者给页面添加新元素。

25.3.1 修改内容

jQuery 提供了 4 个方法，根据选中元素的类型，从元素中读取信息：

- `attr("特性名")`：选中元素的 HTML 特性值
- `html()`：选中元素的内部 HTML 内容
- `text()`：选中元素的内部文本内容(去掉 HTML)
- `val()`：大多数窗体元素的值，对应于元素的 HTML 特性值

也可以修改这些值。为此提供了下面的 4 个方法：

- `attr("特性名", "特性值")`
- `html("新的 HTML 内容")`
- `text("新的文本内容")`
- `val("新值")`

可以看出，getter 和 setter 方法是相同的——只有参数的个数有助于 jQuery 确定应执行什么操作。从 API 的角度来看，这初看起来有点怪异，但有助于使方法列表短小简洁。

程序清单 25-5 演示了 `html()` 方法的两个作用：一旦 DOM 准备好，就提取选中元素的当前 HTML 内容，接着添加当前的时间。结果如图 25-7 所示。

程序清单 25-5 用 jQuery 读写 HTML 内容

```

<%@ Page Language="C#" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>jQuery</title>
    <script type="text/javascript" src="Scripts/jquery.js"></script>
    <script type="text/javascript">
        $(function() {
            var oldvalue = $("div").html();
            $("div").html(oldvalue + " " + new Date().toLocaleTimeString());
        });
    </script>
</head>
<body>
    <div>
        <i>jQuery</i>
    </div>
</body>
</html>

```

如果使用 `html()`(或 `text()`、`val()`或 `attr()`)作为 `setter` 方法,那么方法调用的返回值就仍旧是表示选中元素的 jQuery 对象。因此,可以使用链接功能——把方法调用链接起来。Cynics 宣称,大多数 jQuery 代码其实只是一行 JavaScript 代码,但非常长。

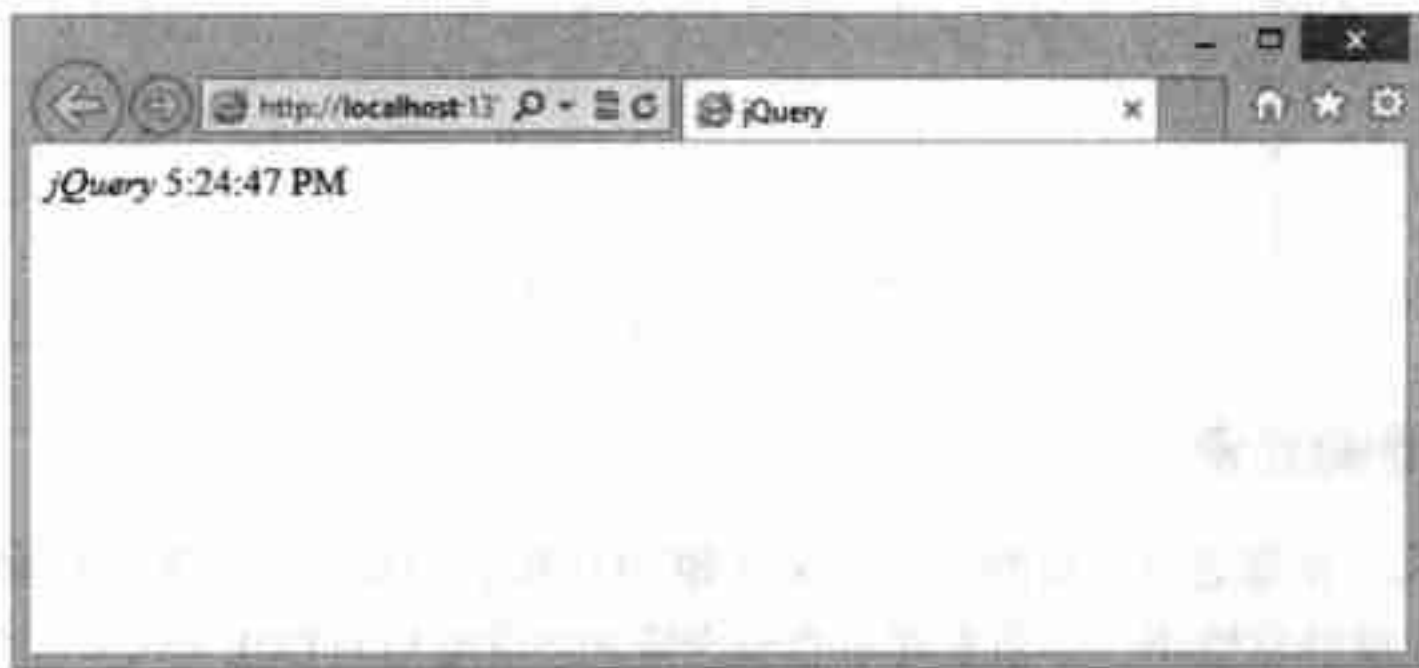


图 25-7

为了演示链接功能,看看下面的 jQuery 方法 `css()`,它可以用于修改元素。顾名思义,可以使用它获取或设置 CSS 属性。这个方法有三种用法:

- `css("property")`: 读取一个 CSS 属性
- `css("property", "value")`: 写入一个 CSS 属性
- `css({"property1": "value1", "property2": "value2", ...})`: 写入多个 CSS 属性

程序清单 25-6 以程序清单 25-5 为基础,给 jQuery 以前修改的文本添加了背景色和前景色。对 `css()`的调用被链接到对 `html()`的调用之后。图 25-8 是该程序清单在浏览器中的结果。

程序清单 25-6 用 jQuery 修改 CSS 样式

```

<%@ Page Language="C#" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>jQuery</title>
    <script type="text/javascript" src="Scripts/jquery.js"></script>
    <script type="text/javascript">
        $(function() {
            var oldvalue = $("div").html();
            $("div").html(oldvalue + " " + new Date().toLocaleTimeString())
                .css("color", "orange")
                .css("background-color", "black");
        });
    </script>
</head>
<body>
    <div>
        <i>jQuery</i>
    </div>
</body>
</html>

```

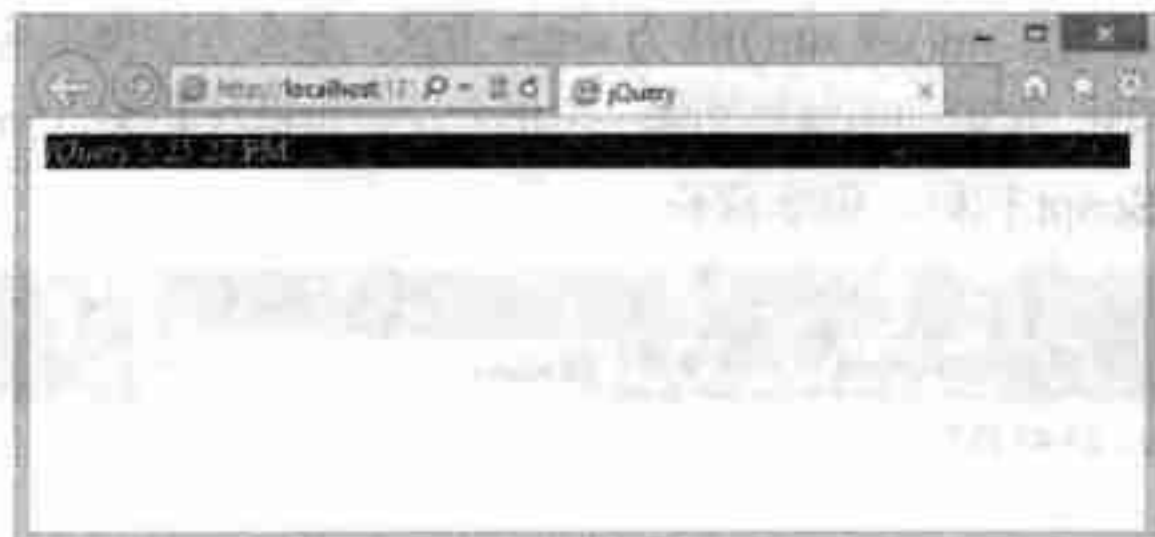


图 25-8

25.3.2 添加和删除元素

刚才介绍的技术非常适合在页面上显示文本和 HTML。但是，当希望给页面添加几个 HTML 元素时，这种方式就非常繁琐：必须非常小心地给特殊字符编写 HTML 标记，才能使用 `html()` 方法添加所有需要的内容。

但是，jQuery 有一个非常强大的 API，用于在 HTML 页面中添加和删除元素。从添加元素开始。这可以使用 `html()` 方法，但接着就需要考虑转义。然而，在 jQuery 中还有另一个 API。使用它需要几步。首先需要创建一个新元素。这里也可以使用 `$()` 函数。如果第一个参数是带尖括号的 HTML 元素(例如 `$("<a>")`)，例如，函数调用的返回值就是表示新 HTML 元素的 jQuery 对象。接着就可以使用 `html()`、`css()` 和其他几个前面讲到的方法进一步修改这个元素。最后，如果希望给页面添加新元素，首先需要选择宿主，这可以是占位符，如 `<div>` 元素或 `<body>` 元素。接着使用 `append()` 方法把新 HTML 元素添加到选中的元素中。

程序清单 25-7 创建了新的 `<a>` 元素，接着设置其 `href` 特性，使之成为正常的链接。最后，把新链接添加到页面的 `<body>` 元素中。

程序清单 25-7 用 append()方法添加元素

```

<%@ Page Language="C#" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>jQuery</title>
    <script type="text/javascript" src="Scripts/jquery.js"></script>
    <script type="text/javascript">
        $(function() {
            $("body").append(
                $("<a>").attr("href", "http://jquery.com/").html("jQuery")
            );
        });
    </script>
</head>
<body>
</body>
</html>

```

在程序清单 25-8 中, 使用了类似的方式。这次使用新建元素的 appendTo()方法。参数再次选择了一个 CSS 选择器, 但这次选择已有的元素, 把新的 HTML 元素追加到该元素中。

程序清单 25-8 用 appendTo()方法添加元素

```

<%@ Page Language="C#" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>jQuery</title>
    <script type="text/javascript" src="Scripts/jquery.js"></script>
    <script type="text/javascript">
        $(function() {
            $("<a>").attr("href", "http://jquery.com/")
                .html("jQuery")
                .appendTo("body");
        });
    </script>
</head>
<body>
</body>
</html>

```

程序清单 25-7 和 25-8 的结果相似, 如图 25-9 所示。

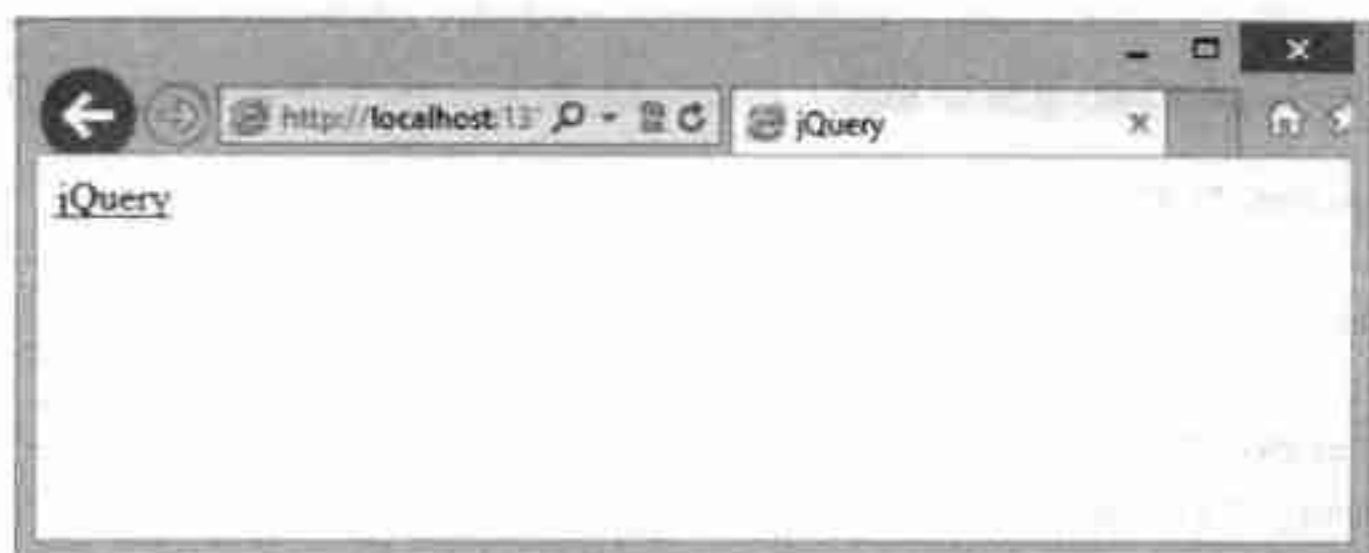


图 25-9

如果希望删除元素，就很容易完成。一种不好的方式是选择周围元素，再执行 `html("")`。更简单的方式是选择要删除的元素，再调用 `remove()` 方法，如程序清单 25-9 所示。

程序清单 25-9 用 `remove()` 方法删除元素

```
$(function() {  
    $("p.myClass").remove();  
});
```

25.4 事件处理

前面所有的代码都在 DOM 完全加载后执行，这是很重要的一步，因为 DOM 的可用性对大多数 JavaScript 应用程序而言是至关重要的。但是，还有一些事件，例如用户单击按钮或按下某个键。

对于大多数事件，jQuery 提供了特定的方法。下面列出了最常见的方法：

- `blur()`：元素失去焦点。
- `change()`：改变窗体元素的内容。
- `click()`：单击元素。
- `dblclick()`：双击元素。
- `focus()`：元素获得焦点。
- `submit()`：提交窗体。

这些方法都是 jQuery 对象方法。首先使用 `$()` 选择一个或多个元素，再订阅这些元素的事件。对于上述方法的参数，可以提供 JavaScript 函数。元素接收给定的事件时，就执行这个函数。

程序清单 25-10 在用户单击按钮时，给页面添加文本。注意事件处理必须在 DOM 加载完后进行。图 25-10 显示了结果。

程序清单 25-10 用特定的方法处理事件

```
<%@ Page Language="C#" %>  
  
<!DOCTYPE html>  
<html>  
  <head runat="server">  
    <title>jQuery</title>  
    <script type="text/javascript" src="Scripts/jquery.js"></script>  
    <script type="text/javascript">
```



```

$(function() {
    $("#button1").click(
        function() {
            $("#label1").html("clicked at: " +
                new Date().toLocaleTimeString());
        }
    );
});
</script>
</head>
<body>
    <form runat="server">
        <span id="label1"></span>
        <input type="button" id="button1" value="Click!" />
    </form>
</body>
</html>

```

并不是所有的 DOM 事件都由这些特定的 jQuery 方法来表示,例如移动设备上的触摸操作。为此, jQuery 提供了通用的事件处理设置方法 `on()`。只要提供事件名和事件处理函数,就完成了设置。程序清单 25-11 是程序清单 25-10 的一部分,它使用了 `on()` 方法。

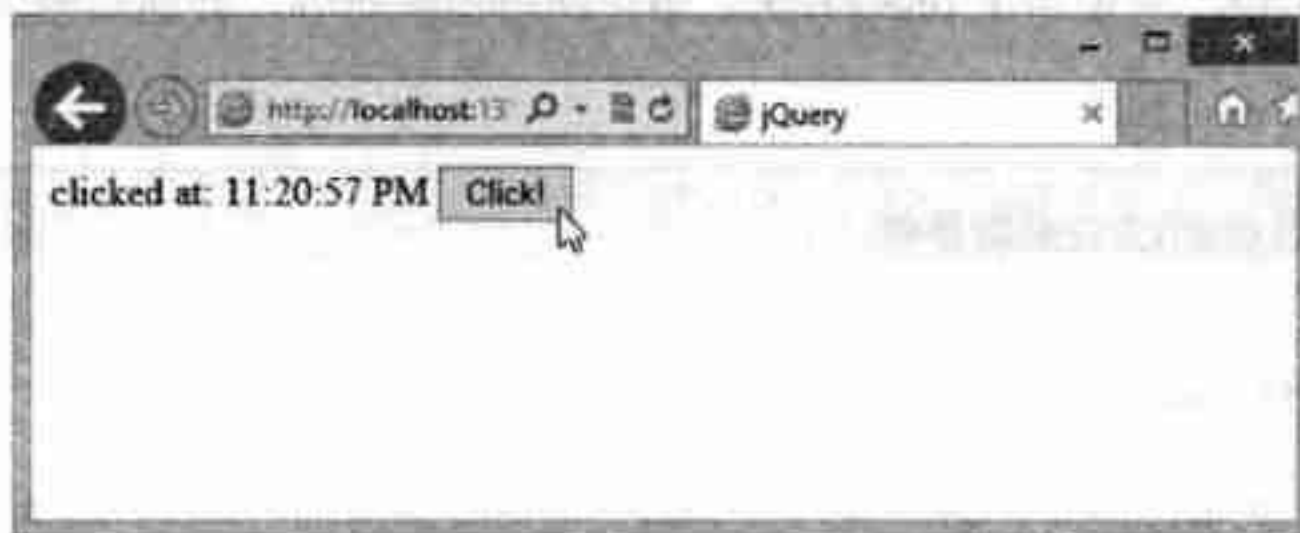


图 25-10

程序清单 25-11 用 `on()` 方法处理事件

```

<%@ Page Language="C#" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>jQuery</title>
    <script type="text/javascript" src="Scripts/jquery.js"></script>
    <script type="text/javascript">
        $(function() {
            $("#button1").on(
                "click",
                function() {
                    $("#label1").html("clicked at: " +
                        new Date().toLocaleTimeString());
                }
            );
        });
    </script>
</head>

```

```

<body>
  <form runat="server">
    <span id="label1"></span>
    <input type="button" id="button1" value="Click!" />
  </form>
</body>
</html>

```

使用 bind()、live()和 delegate()方法

on()方法(及删除事件处理程序的对应方法 off())在 jQuery 1.7 中引入。此外还包含其他功能,例如给页面上以后添加的 HTML 元素添加事件处理程序。JavaScript 库的以前版本用其他方法建立事件处理程序,通常是 bind(),以及用于以后绑定事件处理程序的 delegate()和 live()。使用 on()比使用这些方法更好,但旧代码中仍有这些方法。jQuery 文档详细说明了如何把旧代码迁移到新的 on()语法,参见 <http://api.jquery.com/bind/>、<http://api.jquery.com/delegate> 和 <http://api.jquery.com/live/>。

在一些情况下,事件处理程序只触发一次,例如提交订单(并不是所有的网站都考虑了这一点,毫无耐性地单击两次很不好)。对于这种情形,可以使用 one()方法。事件处理程序只执行一次,接着就删除了。

看看程序清单 25-12,它使用了 one()方法。第一次单击按钮时,会显示当前时间。以后的所有单击都不显示结果。

程序清单 25-12 用 one()方法处理事件

```

$(function() {
  $("#button1").one(
    "click",
    function() {
      $("#label1").html("clicked at: " + new Date().toLocaleTimeString());
    }
  );
});

```

25.5 AJAX

jQuery 的最后一个重要成分是 AJAX,它把 HTTP 请求发送给服务器。为了便于测试这个功能,首先创建一个 ASHX 处理程序(参见程序清单 25-13),用于接收 GET 和 POST 请求,再根据输入返回一些数据。实际上,这个处理程序有三个操作:

- 浏览器发送一条简单的 GET 请求。接着处理程序把当前时间返回为字符串。
- 浏览器发送一条带有 URL 信息 json=true 的 GET 请求,接着处理程序把当前时间返回为 JSON 格式的字符串(放在双引号中)。
- 浏览器发送一条 POST 请求。接着处理程序返回文本,说明这是 POST 请求。

程序清单 25-13 AJAX 调用的通用处理程序

```
<%@ WebHandler Language="C#" Class="Listing_25_13" %>
```

```

using System;
using System.Web;
using System.Web.Script.Serialization;

public class Listing_25_13 : IHttpHandler {

    public void ProcessRequest(HttpContext context)
    {
        switch(context.Request.RequestType)
        {
            case "GET":
                if(context.Request["json"] == null ||
                    context.Request["json"] != "true")
                {
                    context.Response.ContentType = "text/plain";
                    context.Response.Write(DateTime.Now.ToLongTimeString());
                }
                else
                {
                    var json = new JavaScriptSerializer().Serialize(
                        DateTime.Now.ToLongTimeString());
                    context.Response.ContentType = "application/json";
                    context.Response.Write(json);
                }
                break;
            case "POST":
                context.Response.ContentType = "text/plain";
                context.Response.Write("POST at " +
                    DateTime.Now.ToLongTimeString());
                break;
        }
    }

    public bool IsReusable
    {
        get
        {
            return false;
        }
    }
}

```

jQuery 提供了许多发出 AJAX 请求的方式。首先介绍简单的 GET 请求。`$.get()` 方法(注意 `$` 和 `get!` 之间的句点)提示浏览器, 给 URL(第一个参数)发送 HTTP GET 请求。服务器发送回数据后, 就调用回调函数(第二个参数), 并把来自服务器的数据自动传递为参数。参见程序清单 25-14。

程序清单 25-14 发送简单的 GET 请求

```

<%@ Page Language="C#" %>
<!DOCTYPE html>
<html>
<head runat="server">
    <title>jQuery</title>

```



```

<script type="text/javascript" src="Scripts/jquery.js"></script>
<script type="text/javascript">
    $(function() {
        $.get("Listing%2025-13.ashx", function(result) {
            $("div").text(result);
        });
    });
</script>
</head>
<body>
    <div></div>
</body>
</html>

```

如果使用了浏览器插件，例如 Internet Explorer F12 Developer Tools 或 Firebug for Firefox，就可以观察正在发出的 HTTP 请求和发送回浏览器的数据(在这些数据显示之前)，如图 25-11 所示。

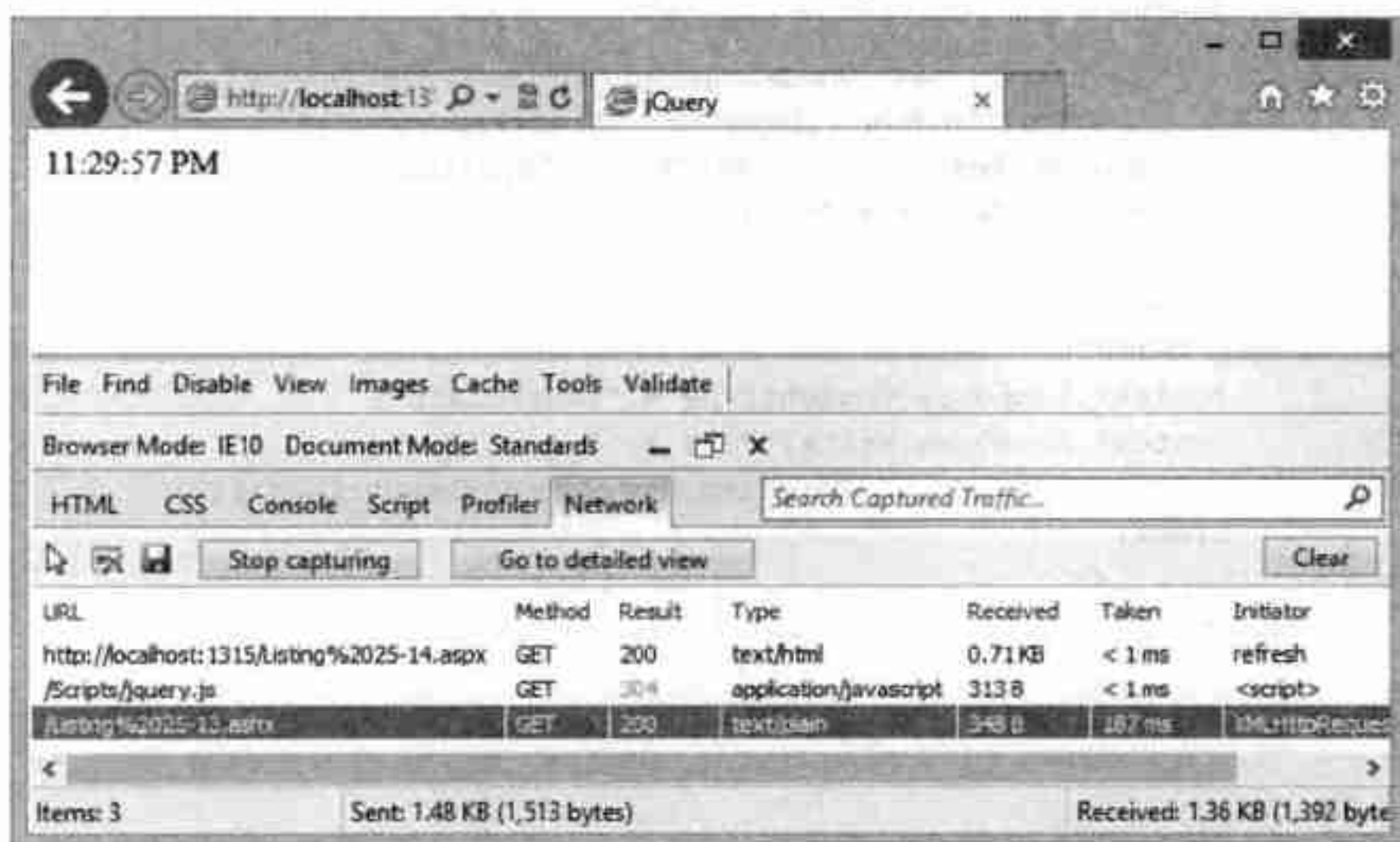


图 25-11



Same Origin Policy 是一项 JavaScript 保护功能，它规定，只有当请求目标(所请求的 URL)与包含 JavaScript 代码(发出 AJAX 请求)的 HTML 页面有相同的源(域、端口、协议)时，才能发出 HTTP 请求。现代浏览器提供了绕过此限制的机制，但为了向后兼容性，应尝试给 AJAX 调用目标和 HTML 页面使用相同的源。为了使本节的代码示例工作，最好使用包含 HTML 文件和服务(ASHX、ASMX)的网站。

如果希望同时发送数据和 GET 请求，就可以将之提供为第二个参数；于是，回调函数就变成第三个参数。注意最好提供一个对象，jQuery 会正确处理它的 URL 编码。程序清单 25-15 发送了一条 GET 请求和一些额外的数据。

程序清单 25-15 发送 GET 请求和数据

```

<%@ Page Language="C#" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>jQuery</title>
    <script type="text/javascript" src="Scripts/jquery.js"></script>
    <script type="text/javascript">
        $(function() {
            $.get("Listing%2025-13.ashx",
                { a: 1, b: 2 },
                function(result) {
                    $("div").text(result);
                });
        });
    </script>
</head>
<body>
    <div></div>
</body>
</html>

```

到目前为止，GET 请求返回的数据都是简单的字符串。对于更复杂的数据，事实标准是使用 JSON(<http://json.org/>)。当然，可以从服务器把 JSON 加载为字符串，再编写用户代码，把它转换为 JavaScript 对象。例如使用 JavaScript 内置的 `JSON.parse()` 方法。但是，更方便的是使用 jQuery 的 `$.getJSON()` 方法，它会发出 HTTP 请求，并解析 JSON 结果，如程序清单 25-16 所示。



`$.get()` 方法也支持 JSON 解析。默认情况下，jQuery 很聪明：如果看起来像是返回了 JSON，就把数据解析为 JSON。如果希望明确指定返回哪些数据，就可以提供数据类型(例如 "json")作为方法调用的最后一个参数。

程序清单 25-16 发出 GET 请求，解析返回的 JSON 数据

```

<%@ Page Language="C#" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>jQuery</title>
    <script type="text/javascript" src="Scripts/jquery.js"></script>
    <script type="text/javascript">
        $(function() {
            $.getJSON("Listing%2025-13.ashx",

```

```

        { json: "true" },
        function(result) {
            $("div").text(result);
        });
    });
</script>
</head>
<body>
    <div></div>
</body>
</html>

```

发送 POST 请求看起来非常相似，但在底层，浏览器需要发送额外的 HTTP 报头，这样服务器才能解析输入数据。当然，如果使用 `$.post()` 方法，jQuery 就可以处理这些。工作方式与 `$.get()` 相同，但使用 POST，而不是 GET。程序清单 25-17 给服务器发送 POST 请求，并显示结果。图 25-12 证明的确使用了 POST。

程序清单 25-17 发送 POST 请求

```

<%@ Page Language="C#" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>jQuery</title>
    <script type="text/javascript" src="Scripts/jquery.js"></script>
    <script type="text/javascript">
        $(function() {
            $.post("Listing%2025-13.ashx",
                { a: 1, b: 2 },
                function(result) {
                    $("div").text(result);
                });
        });
    </script>
</head>
<body>
    <div></div>
</body>
</html>

```

`$.get()`、`$.getJSON()` 和 `$.post()` 其实是速记方法，它们在内部仅把工作委托给 jQuery 通用的 AJAX 方法 `$.ajax()`。这个方法需要两个参数：发送 HTTP 请求的目的地 URL，以及包含额外选项的散列表。额外选项有要使用的 HTTP 谓词、回调函数等。在 jQuery 1.9.0 中，有 33 个不同的选项，所以完整列表超出了本章的范围。但表 25-1 列出了最重要的选项。

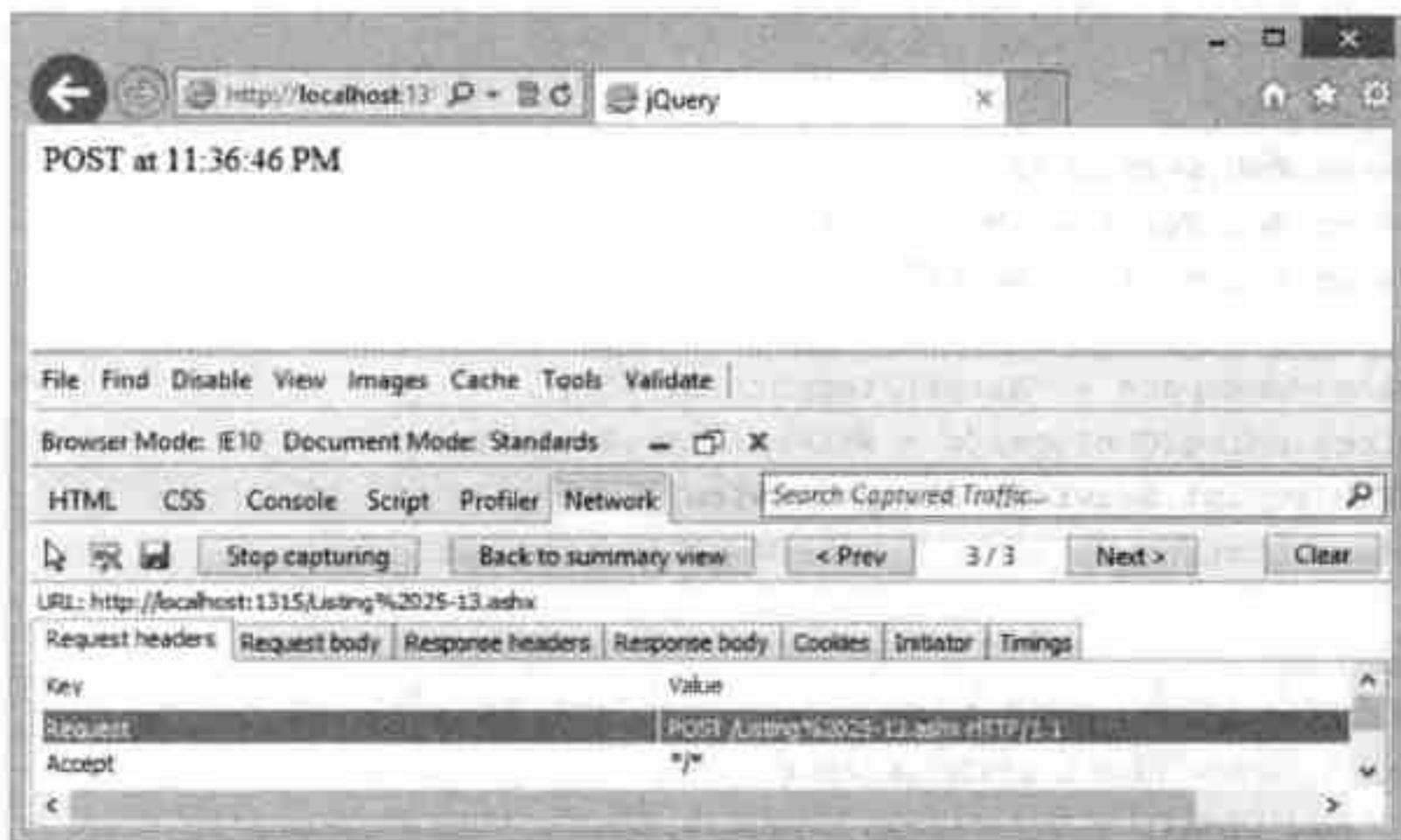


图 25-12

表 25-1

选 项	说 明
contentType	要发送给服务器的 HTTP 报头的内容类型
data	要发送给服务器的数据
dataType	期望从服务器返回的数据的格式(例如"json"或"xml")
error	HTTP 请求失败时使用的处理函数
success	HTTP 请求成功时使用的处理函数
timeout	超时值(毫秒)。发送 HTTP 请求后,若过了这个时间,没有从服务器返回结果,就执行错误处理函数
type	请求类型,例如 GET 或 POST
url	发送请求的目的地 URL

有了这些,就可以执行更复杂的 HTTP 请求。例如,如果使用的是 ASMX Web 服务,从 AJAX 调用该服务的 HTTP 请求(参见第 23 章)就必须非常特别:必须设置额外的报头,并且返回的数据还有特定的格式。

程序清单 25-18 实现了一个简单的 ASMX 服务。注意如何使用 `System.Web.Script.Services.ScriptService` 特性使之可通过 JavaScript 调用。ASMX 服务创建了一系列“Link”对象。每个对象都包含链接 URL 和链接文本。最后,返回该列表。ASMX 服务会自动把这些数据转换为 JSON,并返回给浏览器。

接着,程序清单 25-19 通过 `$.ajax()` 调用这个服务,解析 JSON 对象(其实封装在另一个 JSON 对象中,其“d”属性值包含实际返回的数据)。然后给每个链接创建列表项(HTML 元素 ``),并追加到页面上。图 25-13 显示了浏览器中的结果。

程序清单 25-18 通过 AJAX 使用的 ASMX 服务

```
<%@ WebService Language="C#" Class="Listing_25_18" %>
```

```

using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Collections.Generic;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.Web.Script.Services.ScriptService]
public class Listing_25_18 : System.Web.Services.WebService {

    [WebMethod]
    public List<Link> getLinks() {
        var links = new List<Link>() {
            new Link() { Url="http://jquery.com/", Text="jQuery"},
            new Link() { Url="http://jqueryui.com/", Text="jQuery UI"},
            new Link() { Url="http://juiceui.com/", Text="Juice UI"}
        };
        return links;
    }

}

public class Link
{
    public string Url { get; set; }
    public string Text { get; set; }
}

```

程序清单 25-19 发送 POST 请求

```

<%@ Page Language="C#" %>

<!DOCTYPE html>
<html>
<head runat="server">
    <title>jQuery</title>
    <script type="text/javascript" src="Scripts/jquery.js"></script>
    <script type="text/javascript">
        $(function() {
            $.ajax({
                url: "Listing%2025-18.aspx/getLinks",
                contentType: "application/json",
                dataType: "json",
                type: "POST",
                success: function(result) {
                    var links = result.d;
                    $.each(links, function(key, value) {
                        $("<li>").append(

```

```

        $("<a>").attr("href", value.Url).text(value.Text)
    ).appendTo("ul");
    });
}
});
});
</script>
</head>
<body>
    <ul></ul>
</body>
</html>

```

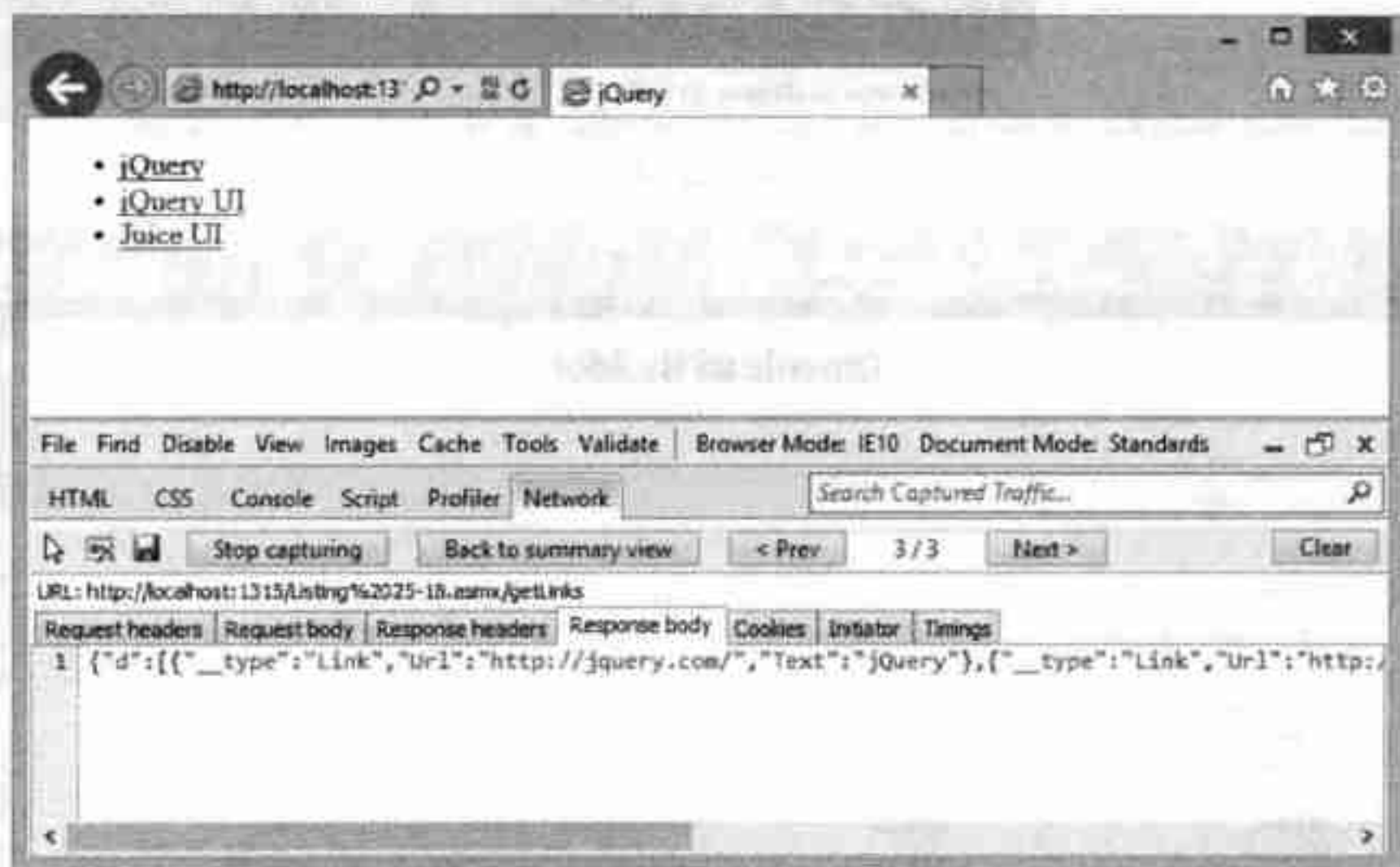


图 25-13

25.6 jQuery UI

前面只使用了 jQuery 的 JavaScript 功能，但仍必须给行为和用户界面编码。然而，jQuery 有一个姐妹项目，名为 jQuery UI，其首页是 <http://jqueryui.com/>，它把自己描述为“用户界面的交互操作、效果、小工具和主题的辅助集合”。的确，jQuery UI 的功能集并不大，但质量非常高。jQuery UI 与 jQuery 的关系类似于 ASP.NET Web 控件与 ASP.NET 的关系：后者是架构，而前者建立在该架构的基础上，提供了 UI 控件。本节简要介绍其中的一个小工具，说明使用该库的通用方法。

在 jQuery UI 首页的右上角(如图 25-14 所示)，可以下载完整的 jQuery UI 包(通常推荐 Stable 版本)，也可以单击 Custom Download，再选择需要的组件(如图 25-15 所示)。为了简单起见，这个例子使用完整的包。也可以使用包管理器来安装 jQuery UI。程序清单 25-20 安装了完整的包。



图 25-14



图 25-15

程序清单 25-20 使用 NuGet 安装 jQuery UI

```
Install-Package jQuery.UI.Combined
```



jQuery UI NuGet 包也会安装 jQuery。新的 jQuery 版本会很快发布，jQuery UI 包就可能引用过时的 jQuery 版本，所以安装可能失败。此时，建议手工安装。

根据安装 jQuery 的方式，本节后面使用的一些路径可能不同，所以确保把代码调整为使用自己的本地系统。这里假定 jQuery 和 jQuery UI JavaScript 文件位于 Scripts 文件夹，jQuery UI 发布的 css 文件夹放在网站的 Content 文件夹中。

jQuery UI 的用法很简单，因为该工具已被完全集成到 jQuery API 中。基本上，先加载 jQuery

库，再加载 jQuery UI 库，以及包附带的 jQuery CSS 文件。

程序清单 25-21 使用 jQuery 的日期选择器控件进行了演示。为此，要把一个简单的文本框放在页面上，再使用 `$()` 选择这个文本框。返回的 jQuery 对象支持 `datepicker()` 方法，我们执行这个方法。如图 25-16 所示，一旦单击文本框，就会显示日期选择器。

程序清单 25-21 使用 jQuery UI 的日期选择器

```
<%@ Page Language="C#" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <link href="Content/css/smoothness/jquery-ui-1.10.0.custom.css" rel="stylesheet" />
    <script type="text/javascript" src="Scripts/jquery.js"></script>
    <script type="text/javascript" src="Scripts/jquery-ui-1.10.0.custom.js"></script>
    <script type="text/javascript">
        $(function() {
            $(".dp").datepicker();
        });
    </script>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <input type="text" class="dp" />
        </div>
    </form>
</body>
</html>
```

如前所述，jQuery UI 尝试获得与 ASP.NET Web 控件类似的效果。有一种方式可以合并这两个技术。Juice UI 项目为每个 jQuery UI 元素创建 Web 控件。该项目的首页是 <http://juiceui.com/> (图 25-17)，安装通过 NuGet 完成(见程序清单 25-22)。



图 25-16

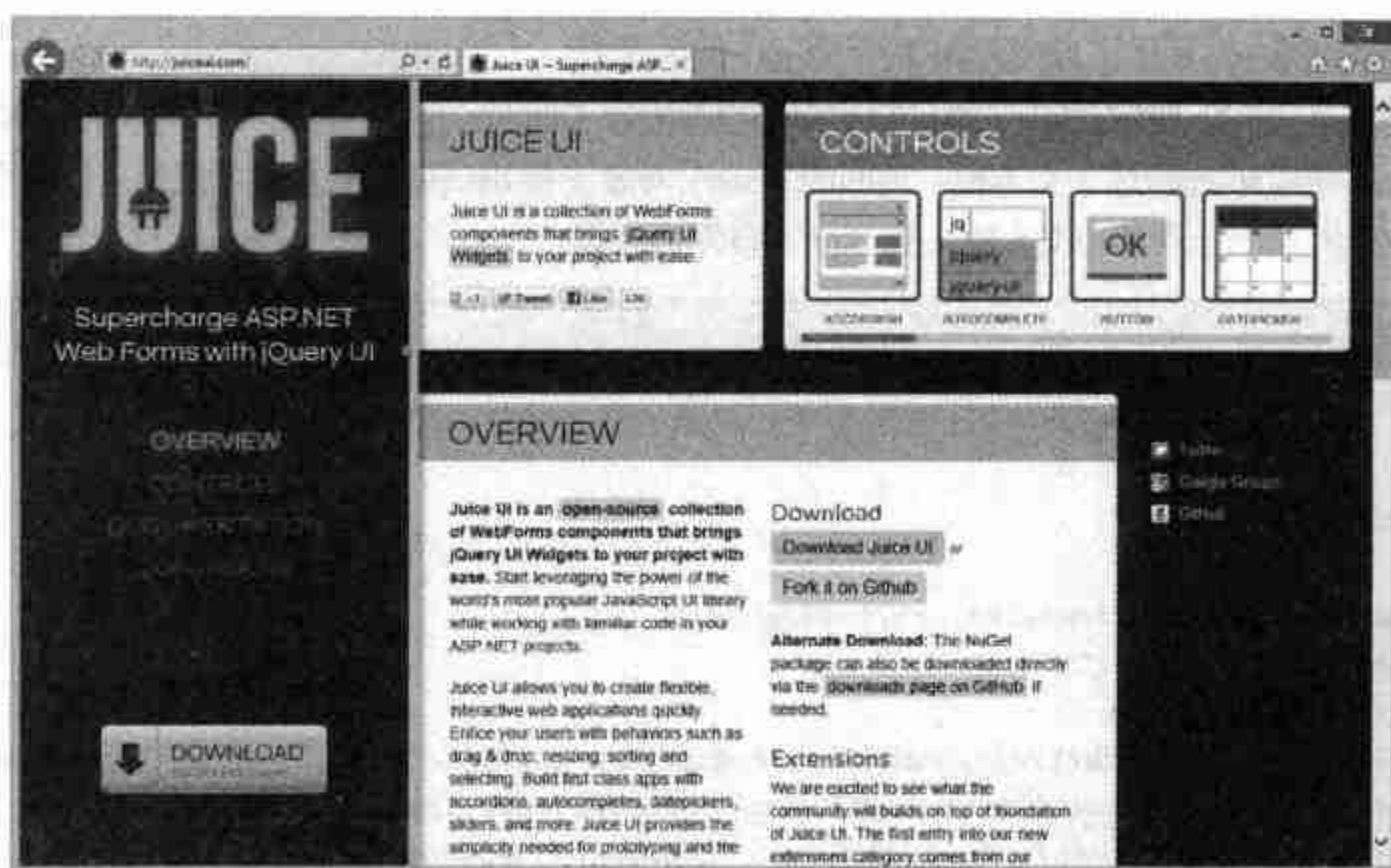


图 25-17

程序清单 25-22 安装 Juice UI

```
Install-Package JuiceUI
```

之后，很容易把程序清单 25-21 迁移到 Juice UI: <Juice:Datepicker>控件表示日期选择器工具。与 AJAX 控件工具集的 API(参见第 24 章)相同，使用 TargetControlID 属性引用接收日期选择器的文本框。程序清单 25-23 列出了(短)代码。

程序清单 25-23 使用 jQuery UI 日期选择器和 Juice UI

```
<asp:TextBox ID="TextBox1" runat="server" />

<Juice:Datepicker TargetControlID="TextBox1" runat="server" />
```

25.7 本章小结

本章讨论了如何把 jQuery 添加到网站或 Web 应用程序中，接着学习了大多数 AJAX 应用程序的通用方法：选择元素，修改它们，监听事件，发出 AJAX 请求。jQuery UI 是 jQuery 的工具库，利用第三方的 Juice UI 项目，可把 jQuery UI 用作 Web 控件。

jQuery 目前是排行第一的 JavaScript 库，原因是，它紧凑、简明、易于使用、非常强大。微软 AJAX 库与 Visual Studio 进行了很好的集成(包括已提供的 VSDOC 文件)，没有人能忽视它。也许有一天，它也会被其他软件替代，但目前第 23 章介绍的控件仍在大量使用它。

第26章

实时通信

本章要点

- 理解实时的 Web 技术
- 实现 HTML5 WebSockets
- 理解 SignalR

过去 10 年, WWW 已经相当成熟了。以前, 大多数 Web 应用程序只包含静态页面, 现在则包含更多的动态内容。过去几年来, WWW 迁移到 Web 2.0, 这样用户就可以交互操作和合作。所有这些合作和交互操作在 Facebook、Twitter 等社交网络中都是实时的。例如, 有人在 Facebook 或微博中发信时, 我们会立即得到通知, 这样就可以与他实时交互操作。

实时 Web 是一组技术和实践方式, 发信者一发出信息, 用户就能接收到信息, 而无须定期检查更新。实时应用程序的例子有社交应用程序, 比如 Facebook 或 Twitter; 多用户游戏, 比如 Scrabble 或 Tic-Tac-Toe; 商务合作, 比如一张绘图板上的多用户合作; 新闻、天气、财务更新应用程序, 比如证券报价机。本章介绍如何编写 ASP.NET 应用程序, 通过降低客户端与服务器之间的延迟, 提高实时用户的体验。

26.1 传统的实时通信选项

本章首先介绍在 .NET 中编写实时应用程序的流行选项。在这种应用程序中, 最常见的需求是在客户端和服务端之间维护永久的连接, 这样服务器一接收到数据, 这些数据就能从服务器推向客户端。考虑 IRC(Internet Relay Chat)的情况, 你希望连接在整个聊天过程中都应是打开的。在聊天过程中, 有时不传输任何数据, 因为用户可能是空闲的。此时, 仍希望服务器和客户端之间的连接是打开的。

尽管 HTTP 使用 Keep-Alive 标记能很好地维护永久的连接, 但如果没有传输数据, 就不能维护长时间打开的永久连接。浏览器和服务端都会断开连接; 如果有新数据进入, 就必须重建连接。如果重建过程增加延时时间, 用户就会感觉系统的反应很迟缓。大多数浏览器仅允许从浏览器到服务

器打开两个连接。当其中一个连接用于进行实时通信,需要下载其他资源时,例如 JavaScript、CSS 文件等,就会减慢网站的速度。HTTP 是请求/响应协议,其中,客户端向服务器发出请求,服务器把响应发送回客户端。如果客户端不发出请求,服务器就无法把响应发送给客户端。图 26-1 是这个协议的简单示意图。

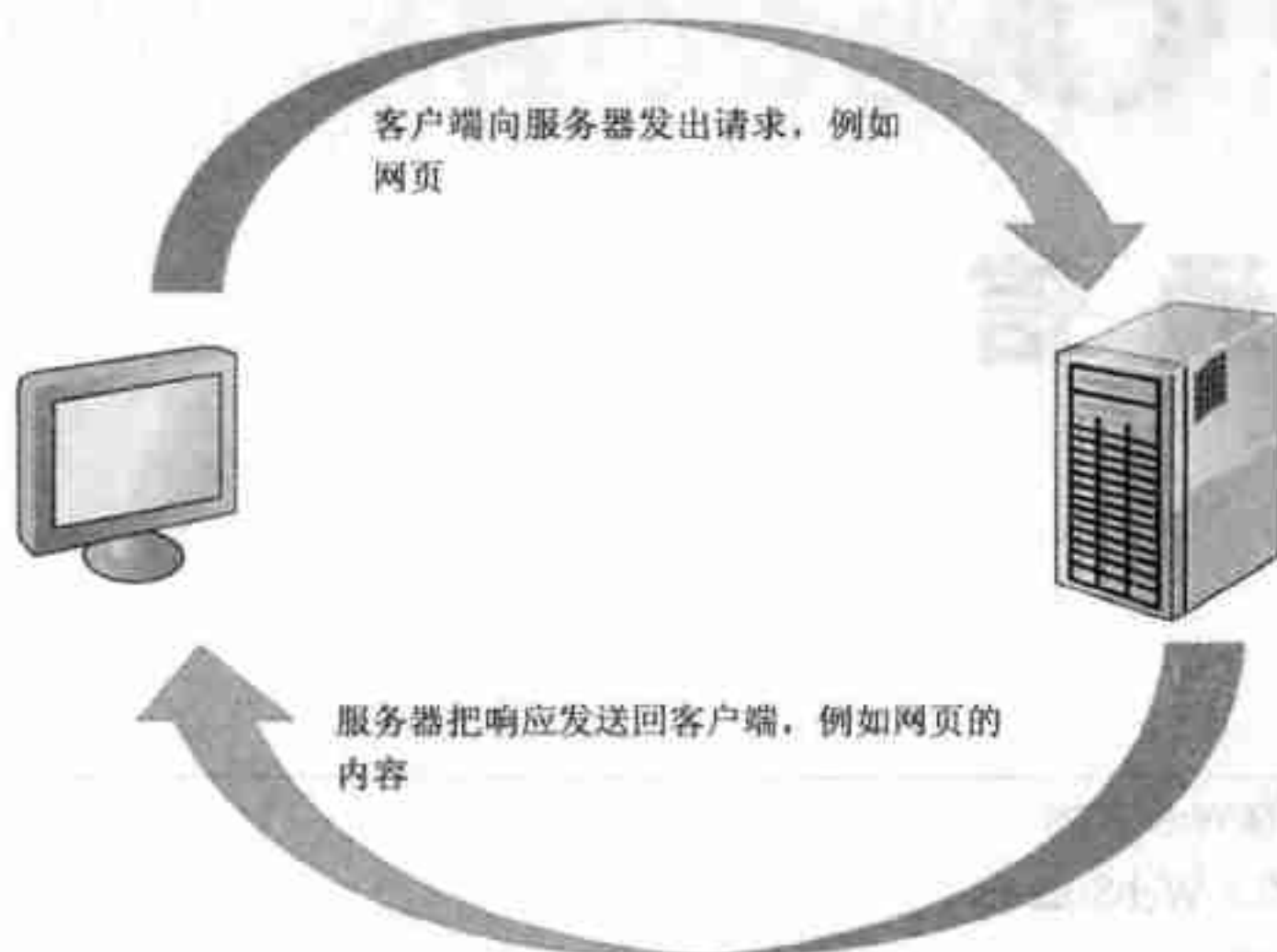


图 26-1

本节介绍在客户端和服务端之间建立长时间打开的连接的一些方式,使服务器可以把数据传递给客户端,而无须客户端请求它们。获得实时功能的另一种方式是客户端明确地从服务器获取数据,但应用程序的性能较差。

26.1.1 使用 Comet

Comet 是 Web 应用程序模型,其中客户端和服务端可以有长时间打开的 HTTP 连接,服务器可以把数据传递给浏览器,而无须浏览器请求数据。因为浏览器和代理程序在设计时没有考虑服务器事件,所以人们开发了几个技术,每个技术都有不同的优缺点。最大的障碍是 HTTP 1.1 规范,它指出,浏览器与 Web 服务器之间的同时连接不应超过两个。下面是实现 Comet 的两种方法,它们依赖所有浏览器都提供的功能,例如 JavaScript,而不是插入模型。

1. 流

实现 Comet 流的应用程序会在服务器和客户端之间打开永久的连接,并监听事件。每次服务器发送新事件时,在浏览器中处理这些事件。流可以用隐藏的 iframe 或使用 XMLHttpRequest API 来实现。

应用程序可以有隐藏的 iframe 元素。每次服务器发送新消息时,消息以大块的形式发送为 iframe 元素。事件发生时,iframe 会填充包含 JavaScript 的脚本标记,这些 JavaScript 需要在浏览器中执行。因为所有的浏览器都按顺序处理页面,所以每个脚本标记就在接收到时处理。

2. 带 Long Polling 的 AJAX

Long Polling 是一种技术,其中,浏览器向服务器投票,检查服务器是否有希望发送的新数据。

浏览器向服务器发出 AJAX 样式的请求, 该请求会一直打开, 直到服务器给客户端发送数据, 作为响应为止。客户端一获得响应, 就会立即给服务器发送另一个请求, 使连接继续打开。这样, 服务器上一发生事件, 服务器就可以发送新数据。

26.1.2 Polling

Polling 是一种方法, 其中客户端向服务器发出请求, 服务器会立即响应, 而无论是否有数据。接着客户端等待一段时间(几秒钟), 再给服务器发出另一个请求。如果服务器有新数据, 就发送它们, 否则, 就给客户端发送回空响应。图 26-2 显示了 Polling 的工作方式。

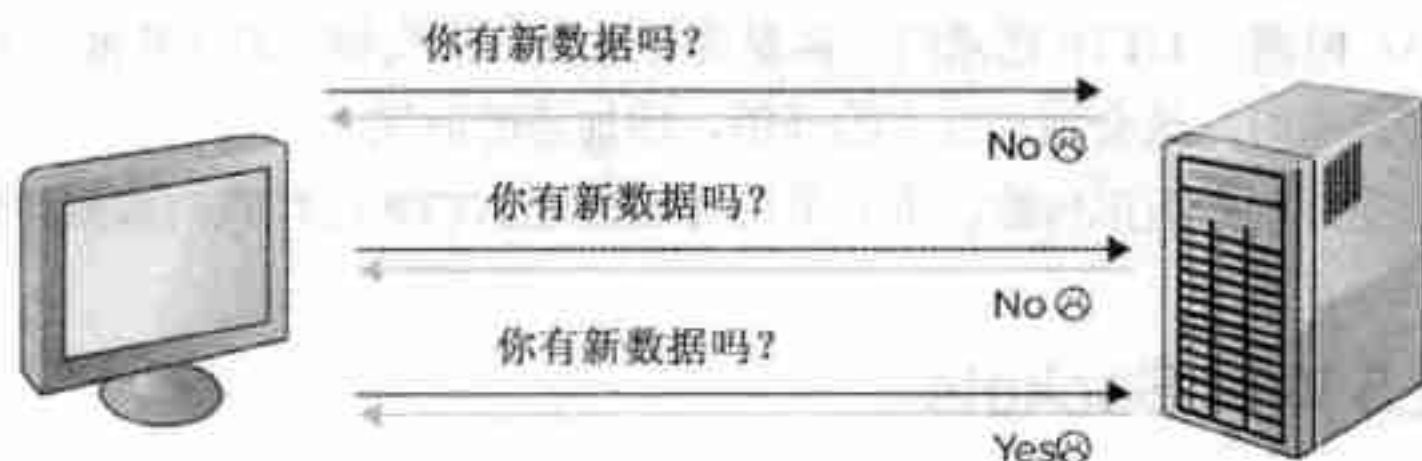


图 26-2

26.1.3 服务器发送的事件

服务器发送的事件(Server-Sent Event, SSE)是一种技术, 用于把推出通知从服务器发送给客户端。推送通知以 DOM 事件的形式发送, 这样客户端就可以通过名为 Event Source 的 JavaScript API 来进行处理。不需要双向数据传输(聊天程序就需要双向数据传输)时, 可以使用 SSE, 因为服务器只需要发送数据。例如, 如果建立证券报价机, SSE 就是不错的选择。图 26-3 是服务器发送事件的简单工作流。



图 26-3

目前这些方法都有许多实现方案, 开发人员可以使用它们建立实时 Web 应用程序。下一节介绍这些方法的缺点, 其中的许多缺点是把 WebSockets 建立为协议的动力。

26.1.4 现有方法的缺点

上述所有推送技术都尝试在客户端和服务器之间维护长时间打开的永久连接, 这样服务器就可以把数据推向客户端。

Comet 流实现起来相当简单, 但很难在不同的浏览器之间工作。并不是所有的浏览器都支持所有的流方法, 所以开发人员需要实现复杂的流传输, 以支持所有的浏览器, 且需要根据浏览器进行

切换。无法实现可靠的错误处理，也无法跟踪连接的状态，因为连接和数据传输是通过 HTML 标记处理的。

使用 AJAX 的 Long Polling 比流好一些，因为它实现了可靠的错误处理和连接管理，但它必须在服务器上使连接一直处于打开状态，直到有数据要发送为止。这种方法的服务器实现比较困难，因为请求是挂起的，直到发送了响应为止。这意味着，要一直使用服务器线程，直到有数据要发送为止。

传统的 Polling 技术使客户端和服务端实现方案非常简单。但是，每次客户端给服务器发出请求时，都有创建 HTTP 请求和发送空响应的额外开销。如果考虑聊天程序中要交换的数据的类型，消息的尺寸会相当小。但建立 HTTP 请求时，需要选择适当的报头和请求的其他元数据。每次把消息从服务器发送给客户端时，这会增加巨大的开销，增加延时时间。

SSE 也不适于支持双向数据传输。下一节将介绍通过 HTTP 进行双向数据传输的新标准。

26.2 HTML5 WebSockets

刚才简要介绍了实现实时通信的一些传统模式以及使用它们的优缺点。

除了本章前面提及的问题之外，所有已有的传统实时通信方法都通过 HTTP 传输数据，这意味着有数据传输的开销，因为每个请求都需要 HTTP 报头。假定编写一个聊天程序来发送聊天数据，则每次发送消息(消息通常很小，也许只有几个字节)时，因为消息是通过 HTTP 发送的，所以需要添加发送 HTTP 报头的开销。这会给所传输的消息增加开销。

还有打开多个底层 TCP 连接的额外开销。服务器要使用 TCP 连接发送消息，再为每个从不同客户端发送来的消息使用 TCP 连接。

HTML5 WebSockets 解决了这些问题，提高了实时应用程序的性能。下面介绍 WebSockets 协议，学习如何在 ASP.NET 中使用。

26.2.1 WebSockets 的概念

WebSockets 是规范，允许在客户端和服务端之间进行双向通信。通信管道是单个 TCP 连接。WebSockets 在 Web 浏览器和 Web 服务器上实现。WebSockets 只使用 HTTP 升级客户端和服务端之间的连接，使数据可以按照 WebSockets 协议进行交换。这个协议可以进行实时交互操作，提供了一种标准化的方式，允许服务器给客户端发送数据，而无须客户端请求数据。

WebSockets 的一种关键实现方案是所有的通信都在端口 80 或 443 上进行。这特别重要，因为许多环境都禁止在端口 80 上进行非标准的 HTTP 连接。WebSockets 以这种方式实现，所以该技术可以运行在现有环境下，而没有其他 HTTP 连接的问题。

WebSockets 需要在浏览器客户端和服务端上实现。所有现代的桌面浏览器都支持 WebSockets 协议。如果在 Windows 8 上使用 ASP.NET 4.5，就拥有了服务器端的 WebSockets 支持。在介绍 WebSockets 协议之前，应先了解现有的传输层。图 26-4 显示了不同的 OSI 层。

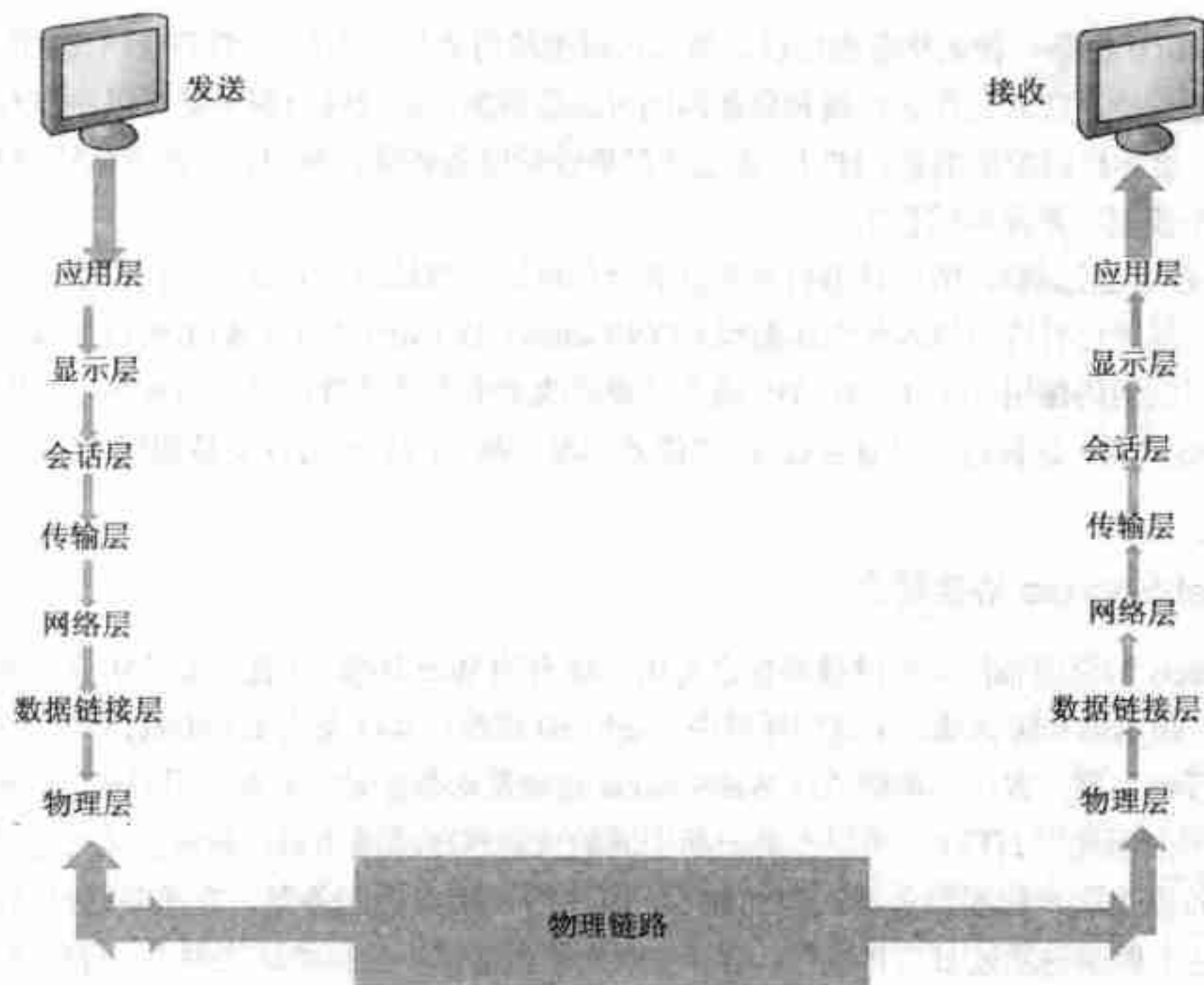


图 26-4

26.2.2 TCP/IP

TCP(Transmission Control Protocol)/IP(Internet Protocol)是用于互联网和类似网络的互联协议集。它们是第一组定义的联网协议。互联网随着时间不断地演变，其上的通信形式也在不断地演变。TCP/IP 以 OSI(Open System Interconnection, 开放系统互联)为基础建模，OSI 定义了构成 TCP/IP 的许多层。把 TCP/IP 分解为层有助于它随时间演变，根据硬件和技术的变化而调整，并允许开发人员工作在不同的层，在该协议的顶部建立协议堆栈或应用程序。

在 OSI 模型的底部是物理层，它通过有线媒介访问控制等协议实现。这一层提供了物理连接。在 OSI 层中向上移动，就是网络层，它由 IP 实现。网络层允许 TCP/IP 与不同的物理层相互操作，把包从一台机器路由到另一台机器。

26.2.3 TCP/HTTP

TCP 和 UDP(User Datagram Protocol)是传输层中最著名的传输协议。这些协议有端口的概念，它们在发送消息时，使用端口对消息进行多路传输和取消多路传输。UDP 在私有网络应用程序中广泛使用，但在 Web 上使用得不多。而 TCP 因为 HTTP 而广泛应用于 Web。

TCP 在计算机之间提供了可靠的数据传输，能管理复杂的情形，例如数据包的定序、从入站数据包中正确构建消息、错误检测、流控制、超时等。应用程序开发人员可以进行无缝的双向数据传输，而无需管理低级复杂性。

HTTP 协议为可靠的、面向连接的请求-响应协议带来了支持。客户端协议给服务器发送请求时，就打开 TCP 端口，通过 TCP 端口进行请求/响应。如果客户端发送另一个请求，就通过另一个 TCP 端口进行请求/响应。HTTP 协议中的这种简单性使 HTTP 协议非常普遍，所以该协议在 Web 上得到

广泛使用。HTTP 也是一种无状态的协议，所以可以缩放得更好。但是，HTTP 没有利用 TCP 中的双向数据传输支持。TCP 允许客户端和服务端同时发送数据。这意味着服务器可以在没有客户端请求的情况下，给客户端发送消息。HTTP 的这种简单性使服务器缓存响应，直到通信管道畅通为止。这说明，这个模型不是高伸缩性的。

Web 的使用已经成熟，所以目前有更丰富的合作机制，包括即时共享、消息传输和许多其他实时交互操作。尽管已有许多技术使用压缩或 CDN(Content Delivery Network)来缩短加载应用程序的响应时间，但它们仍使用 HTTP。HTTP 缺乏重要的改进来支持这种实时应用程序，使用户获得畅快的实时 Web 体验，这促进了 WebSockets 协议的实现，WebSockets 协议支持通过 TCP 进行双向数据传输。

26.2.4 WebSockets 协议简介

WebSockets 协议面临的主要挑战是使之可用于现有的 Web 环境。这意味着该协议必须使用现有的代理程序、路由器和防火墙。在这些环境中，端口 80 或端口 443 是为 HTTP/HTTPS 连接打开的，并可以交换任何数据。客户端希望通过 WebSockets 连接发送数据时，需要调用服务器，使用 HTTP 进行连接。因为仍使用 HTTP，所以不必更新中间的代理程序或路由器，就可以进行连接。

该协议首先是客户端和服务端之间的握手，其中客户端告诉服务器，它希望通过 WebSockets 发送数据。这个握手也通过 HTTP 进行，这样中间的路由器就不会拒绝这次握手。程序清单 26-1 列出了客户端要发送给服务器的报头，指定要建立 WebSockets 连接。

程序清单 26-1 WebSockets 握手

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

这些报头与任何正常的 HTTP 报头没有任何区别。这次握手是正常的 HTTP GET 请求。客户端在 Host 报头中包含主机名，这样客户端和服务端就可以验证，它们是否同意使用哪台主机。Upgrade 和 Connection 报头也在 HTTP 中指定，由客户端用于升级要使用的协议。Upgrade 报头是 WebSockets 协议所需要的。它指定，客户端希望把连接升级到 WebSockets 协议。图 26-5 显示了如何把 HTTP 请求升级到 WebSockets 连接。

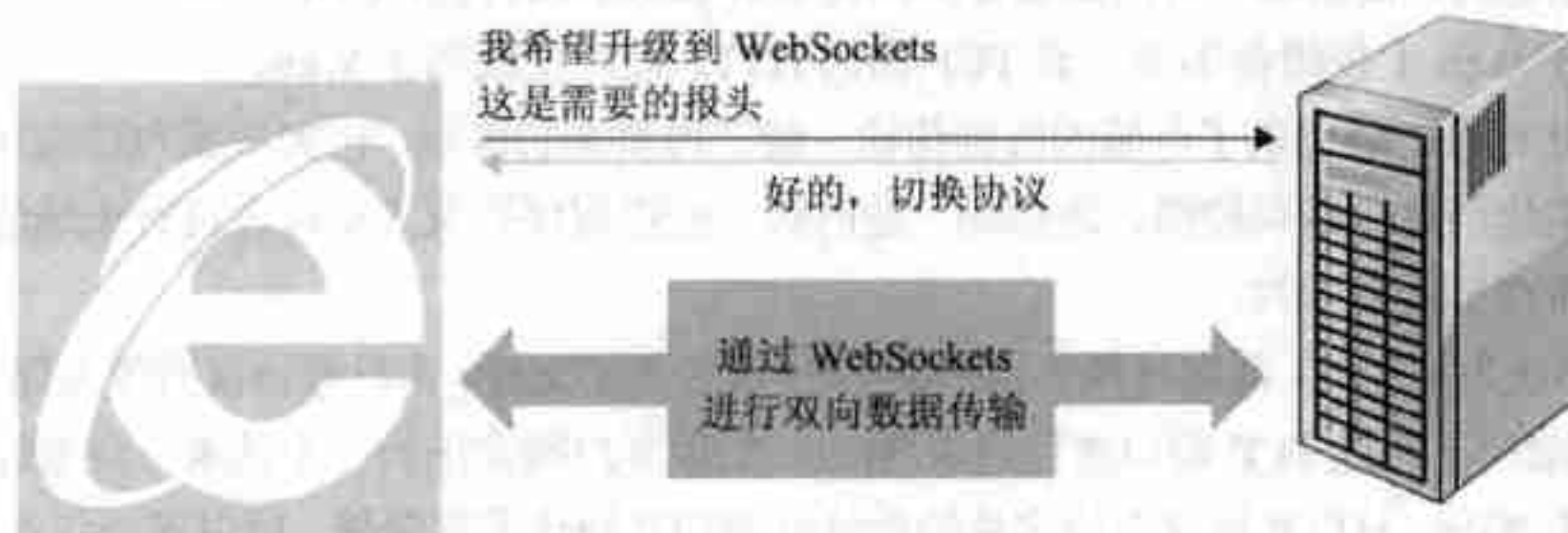


图 26-5

Origin 报头字段通过在浏览器中使用 WebSockets API 的脚本, 禁止 WebSockets 的未授权使用。如果服务器不希望接受来自这台主机的入站连接, 就可以拒绝连接, 并发送回错误码。

Sec-WebSocket-Version 报头告诉服务器, 客户端支持哪个版本的 WebSockets 协议。如果在 Windows 8 中建立应用程序, 版本号就应设置为 13。如果服务器支持 WebSockets, 但不支持这个版本的协议, 服务器就可以中止这个请求。这很重要, 因为随着 WebSockets 规范的发展, 不同的客户端(例如 IE、Chrome、Firefox 等)和服务器会支持该规范的不同版本。客户端和服务端可以在自己方便的时间升级到该规范的最新版本。在理想情况下, 所有的客户端和服务端都使用最新版本。但是, 不同的公司实现软件所需的时间是不同的, 所以客户端和服务端支持的版本常常并不相同。这个报头可以确保, 客户端和服务端验证它们分别支持协议的哪个版本, 在版本不兼容时, 拒绝 WebSockets 请求。

Sec-WebSocket-Key 报头禁止服务器接受来自非 WebSockets 客户端的连接。这可以防止攻击者使用 XMLHttpRequest 或窗体提交, 给 WebSockets 服务器发送小信息包来欺骗服务器。这个报头向客户端和服务端证明, 正在建立的是合法的连接。其值必须是随机选择的——最好是加密的随机 16 字节数, 在安全领域中称为 nonce(乱数), 它是这个报头值的 64 位编码。

其他报头字段用于在 WebSockets 协议中选择选项。在这个版本中, 常见的可用选项是子协议选择器(Sec-WebSocket-Protocol), 它是客户端支持的一系列扩展(Sec-WebSocket-Extensions)。Sec-WebSocket-Protocol 请求报头字段可以表示服务器接受哪个子协议(应用程序级的协议, 在 WebSockets 协议之上)。服务器选择一个可接受的协议, 或者不选择任何协议, 并在握手过程中响应该值, 表示它已选择该子协议。

客户端发送握手的信号, 服务器理解了 WebSockets 后, 服务器就把该协议切换为 WebSockets。服务器给客户端发送 HTTP 101 响应, 表示 WebSockets 协议的握手已完成, 连接现在升级为使用 WebSockets 协议。程序清单 26-2 显示了服务器发送回客户端的响应报头。

程序清单 26-2 WebSockets 信号交换的响应

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

这些也是有效的 HTTP 报头, 因此上述是常规的 HTTP 响应。客户端还需要验证响应, 确保响应来自接受握手的服务器。为此, 客户端使用 Sec-WebSocket-Accept 报头。服务器获得 Sec-WebSocket-Key 值时, 不会解码它, 而是把该字符串和著名的 GUID(全球唯一标识符)连接起来, 用 SHA1 散列合并的结果, 生成 64 位编码值, 然后放在这个报头中, 发送回客户端。客户端可以执行相同的过程, 验证其结果。如果验证成功, 客户端就可以接受握手的响应, 建立一个 WebSockets 连接。

客户端和服务端现在可以通过这个 WebSockets 连接收发数据了。下一节介绍如何进行数据传输。

26.2.5 WebSockets 数据传输

WebSockets 协议用于为服务器和客户端之间的双向通信提供高可伸缩性和低系统开销。升级了

连接后，通信就通过 TCP 进行。该协议在 TCP 的顶部提供了一种小信息包架构机制，但对信息包的大小没有限制。TCP 是基于流的，而 WebSockets 是基于消息的数据传输机制。这意味着，数据通过 WebSockets 传输时，会传输为序列帧。帧协议只需要额外的几字节帧开销，因为传输是在 TCP 上进行的，所以具备在 TCP 上传输的可靠性和顺序传输特性。帧有两种类型：控制帧和数据帧。控制帧用于交流 WebSockets 的状态，而数据帧用于传输数据。

每个帧都有 opcode，表示帧的类型和负载的大小。负载包含客户端和服务器相互发送的实际数据。WebSockets 支持帧的碎片化。这表示，消息可以分解为长度不同的帧。这给信息包提供了方便，无须处理复杂的大小限制。如果发送者和接收者不知道所收发数据的确切长度，WebSockets 就把消息分解为帧。接着，每一帧都可以指定它有多少数据，是否是消息的最后一帧，从而表示整个消息是否已传输完毕。数据帧可以带有二进制或文本数据，这由帧的 opcode 值来表示。图 26-6 是 WebSockets 的典型数据帧。

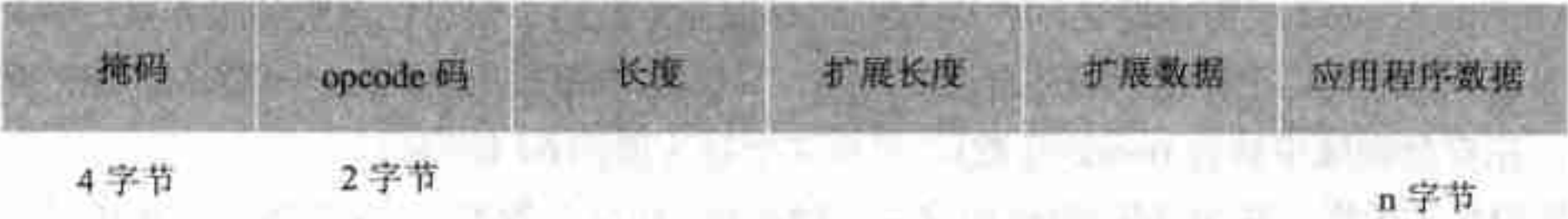


图 26-6

控制帧主要用于关闭连接，也可以用于 ping 客户端或服务器，确保端点仍是活动的，TCP 连接一直处于打开状态。如果连接的一端发送了 ping 命令，另一端就可以发送 pong 命令，确认连接是活动的。对于过分热情的代理程序或防火墙，这可以使连接一直处于打开状态。如果连接处于空闲状态，代理程序或防火墙会断开连接。客户端或服务器可以指定合适的 opcode 来构建控制帧。

数据帧的一项重要应用是遮蔽(masking)。遮蔽是一种简单的算法，其中，负载与包含在帧中的键执行 XOR 操作。遮蔽键必须是不能预测的，以防止恶意应用程序的作者选择出现在线路上的字节。

26.2.6 WebSockets API

WebSockets 协议由 IETF(Internet Engineering Task Force)标准组织进行了标准化。这意味着，任何人都可以遵循这个标准，在自己的客户端或服务器上实现它。

所有常用的浏览器，例如 Internet Explorer、Chrome、FireFox、Safari、Opera 等，都添加了对 WebSockets API 的支持。

表 26-1 列出的一些常见 JavaScript 事件监听器可用于在 Web 浏览器的 WebSockets 连接上执行基本操作。

表 26-1

事件监听器	说 明
onopen	打开 WebSockets 连接时，触发该事件
onclose	关闭 WebSockets 连接时，触发该事件
onmessage	客户端接收服务器通过 WebSockets 连接发送来的数据时，触发该事件
send	该方法用于把数据从客户端发送到服务器
close	该方法关闭服务器和客户端之间的 WebSockets 连接

26.2.7 ASP.NET 4.5 中的 WebSockets

前面学习了 WebSockets 协议，本节介绍如何在 ASP.NET 中使用 WebSockets 建立应用程序。

因为 WebSockets 协议需要由客户端和服务端实现，所以要考虑它们两个。在 Web 服务器上，有一个新的模块，名为 WebSockets，它在 Windows Server 2012 的 IIS 8 中引入。该模块实现了 WebSockets 协议，负责处理数据传输。

图 26-7 演示了如何在 IIS 8 中启用 WebSockets 协议。必须在 Windows 中启用 IIS Web Server 角色。之后，就需要在 IIS Application Development Features 区域选择 WebSocket Protocol 选项。

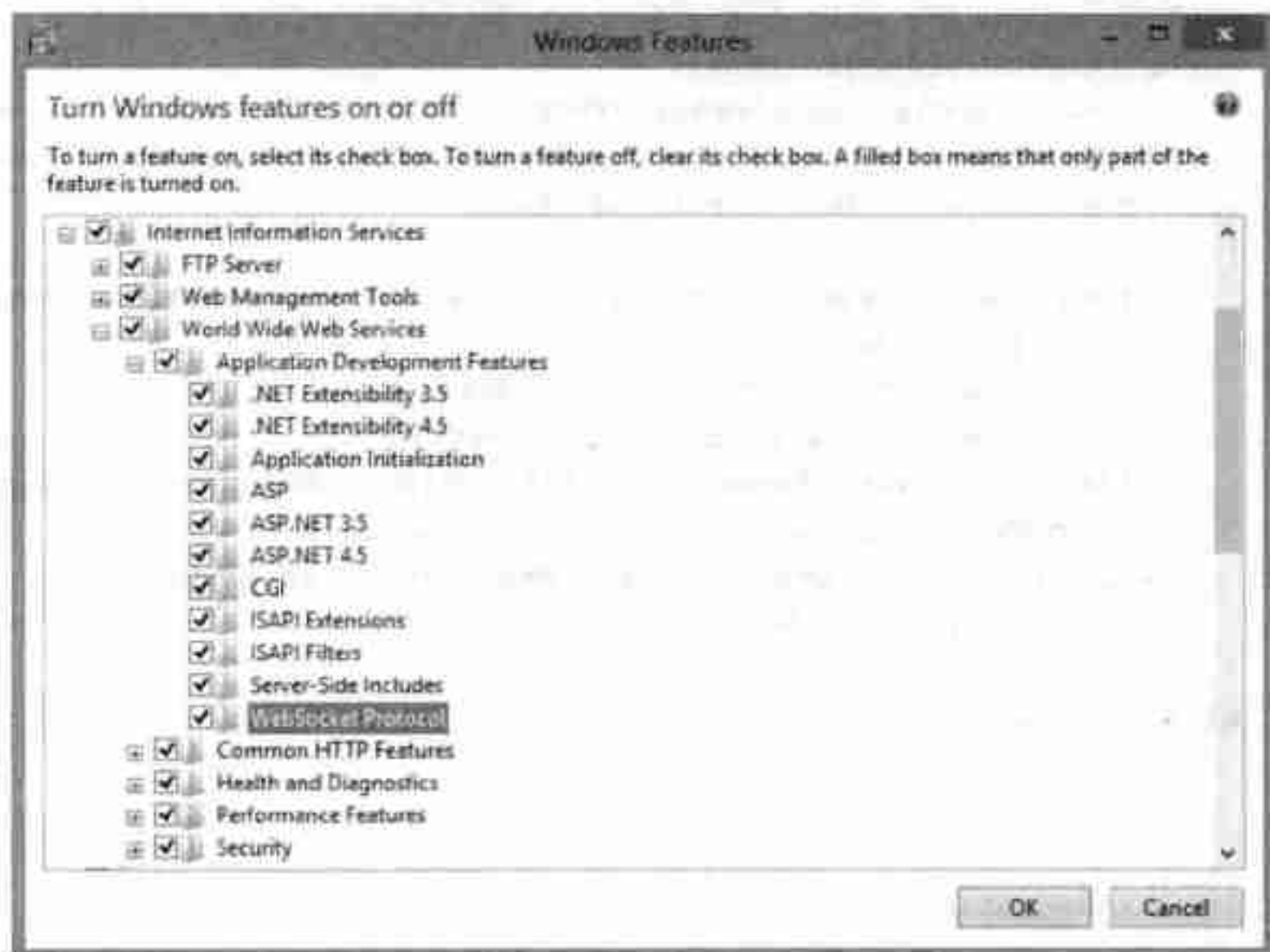


图 26-7

ASP.NET 提供了很好的 API，应用程序开发人员可以使用它处理 WebSockets 请求。这些 API 允许开发人员接受 WebSockets 连接，通过该连接进行通信。接着编写 ASP.NET HTTP 处理程序，这种处理程序可以与任何 WebSockets 请求交互操作，还可以收发数据。

客户端发送 WebSockets 握手信号时，ASP.NET 会检查服务器是否支持 WebSockets。如果满足所有的协议要求，就升级请求。成功升级服务器和客户端之间的连接后，ASP.NET 和 IIS 就接收通过该连接入站的数据，并通过该连接发送数据。

应用程序需要发送数据时，模块就根据 WebSockets 规范构建合适的帧，并通过 WebSockets 连接发送这些帧。模块还查看来自客户端的入站帧，从这些帧中构建消息，这样应用程序就可以接收消息了。

1. 服务器端的 WebSockets 响应处理程序

程序清单 26-3 是一个简单的 WebSockets 响应处理程序，它接收一个 WebSockets 请求，从客户端接收消息，再把该消息传送回客户端。

程序清单 26-3 ASP.NET 中典型的 WebSockets 响应处理程序

```
public class Handler : IHttpHandler {

    public async Task MyWebSocket(AspNetWebSocketContext context)
    {
        WebSocket socket = context.WebSocket;
        while(true)
        {
            ArraySegment<byte> buffer = new ArraySegment<byte>(new byte[1024]);
            // Asynchronously wait for a message to arrive from a client
            WebSocketReceiveResult result =
                await socket.ReceiveAsync(buffer, CancellationToken.None);
            // If the socket is still open, echo the message back to the client
            if(socket.State == WebSocketState.Open)
            {
                string userMessage = Encoding.UTF8.GetString(buffer.Array, 0,
                    result.Count);
                userMessage = "You sent: " + userMessage + " at " +
                    DateTime.Now.ToLongTimeString();
                buffer = new ArraySegment<byte>(Encoding.UTF8.GetBytes(userMessage));
                // Asynchronously send a message to the client
                await socket.SendAsync(buffer, WebSocketMessageType.Text,
                    true, CancellationToken.None);
            }
            else { break; }
        }
    }

    public bool IsReusable
    {
        get { throw new NotImplementedException(); }
    }

    public void ProcessRequest(HttpContext context)
    {
        if(context.IsWebSocketRequest)
        {
            context.AcceptWebSocketRequest(MyWebSocket);
        }
    }
}
```

这个响应处理程序接收到请求时，就先在 `ProcessRequest` 函数中检查该请求是否是 WebSockets 请求。如果是，响应处理程序就在 `ProcessRequest` 调用的 `MyWebSocket` 任务中读取通过 WebSockets 连接发送的消息。响应处理程序使用 WebSockets API 收发消息。

2. 客户端的 WebSockets 代码

学习了 ASP.NET 处理程序中的 WebSockets 服务器端实现代码后，就该看看如何在 Web 页面中创建 WebSockets 连接了。

要连接端点，只需创建一个新的 WebSockets 实例，通过 URL 提供新对象，该 URL 表示要连接

的端点。



ws://和 wss://前缀一般分别用于表示 WebSockets 连接和安全的 WebSockets 连接。

建立好连接后，就可以使用 `send` 和 `onmessage` 事件监听器通过该连接收发消息了。使用表 26-1 中的其他事件监听器可以关闭连接，执行 WebSockets API 支持的其他函数。

程序清单 26-4 中的典型 Web 页面在 JavaScript 中使用 WebSockets API 实现方案连接 WebSockets 响应处理程序。

程序清单 26-4 典型的 WebSockets JavaScript 客户端代码

```
<%@ Page AutoEventWireup="true" %>
<!doctype html>
<html>
<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<title>Web Socket Test</title>
<script type="text/javascript">
    var socket;
    function initializeWebSocket() {
        var host = "ws://localhost:33610 /CS/Handler.ashx";
        try {
            socket = new WebSocket(host);

            socket.onopen = function (msg) {
                var s = 'Socket open';
                document.getElementById("serverStatus").innerHTML = s;
            };

            socket.onmessage = function (msg) {
                var serverData = document.getElementById("serverData");
                var newElem = document.createElement("p");
                newElem.appendChild(document.createTextNode(msg.data));
                serverData.insertBefore(newElem, serverData.firstChild);
            };

            socket.onclose = function (msg) {
                var s = 'Socket closed';
                document.getElementById("serverStatus").innerHTML = s;
            };
        }
        catch(ex) { alert(ex); }
    }

    function send() {
        var e = document.getElementById("msgText");
        socket.send(e.value);
    }
}
```

```

    }
    initializeWebSocket();
</script>
</head>
<body>
<h1>Web Socket Echo Demo</h1>
<p id="serverStatus"></p>
<p>
    This text will be sent on the socket:<br />
    <input id="msgText" type="text" size="30">
    <input type="button" value="Send" onclick="send()">
</p>
<div id="serverData"></div>
</body>
</html>

```

浏览这个页面时，会执行 JavaScript 代码。在 JavaScript 中，页面首先尝试向程序清单 26-3 中的响应处理程序发出 WebSockets 请求。单击页面上的 Send 按钮时，页面会通过 WebSockets 连接把数据发送给响应处理程序。响应处理程序把响应发送回浏览器。浏览器接收到数据后，就调用 JavaScript 函数 onmessage()，在页面上显示接收到的数据。图 26-8 显示了这个示例的响应。



图 26-8

26.2.8 使用 WebSockets 的优点

HTML5 WebSockets 不仅改进了 HTTP 通信，还使实时应用程序更强大。它大大降低了负载开销，缩短了延时时间，所以 Web 领域的大多数著名公司都立即改为在 Web 应用程序中使用 WebSockets。

WebSockets 为 HTTP 提供内置的支持，以编写实时应用程序。把连接从 HTTP 升级为 WebSockets，通过 WebSockets 管道传输数据，能大大减小负载，这是因为每次消息传输都不再需要发送 HTTP 报头了。另外，因为 WebSockets 通信在 TCP 连接上实现，所以不再需要在服务器上打开多个端口，收发多个客户端的数据。应用程序在实现实时场景时，会非常简单，因为 WebSockets API 的易用性，更便于维护从客户端到服务器的连接，跟踪来自服务器的回应。

WebSockets 协议的握手在 HTTP 上进行。这表示，它可以用在已经使用 HTTP/HTTPS 的环境中。确保能方便地采用 WebSockets，是因为这些端口和所有的中间介质都可以通过 HTTP 传输数据。

WebSockets 是安全的。这个协议的设计人员面对的一项挑战是安全性。WebSockets 有 URI 规

范 wss，它使用与 HTTPS 和 TLS/SSL 一样的加密方式。WebSockets 在握手时，客户端和服务端都使用 Sec-WebSocket-Protocol 报头验证连接。在把数据传输为帧时使用遮蔽技术，以防止对网络基础设施的攻击。

26.3 SignalR

WebSockets 为支持可伸缩性很高的实时应用程序迈出了一大步。有了 WebSockets 协议，就可以在 HTTP 上进行双向数据传输。因为 WebSockets 是现代标准，最近才引入为 HTML5 的一部分，所以还未得到广泛采用。本节为不能采用 WebSockets 的人介绍一种更现实的替代方案——SignalR，它集中了两者的优点。

WebSockets 是现代标准，有其优点和缺点。尽管现代标准为更好、更强大的未来铺平了道路，但该标准要得到广泛采用还需要一定的时间。在服务器端，只有 Windows 8 或 Windows Server 2012 在 .NET Framework 4.5 中支持 WebSockets，这限制了 WebSockets 应用程序的使用范围。

如果使用 WebSockets 编写应用程序，旧平台上的用户就不能使用该应用程序，除非为他们提供一个备用方案。这个备用方案可能涉及默认采用 Comet 或其他推送技术。应用程序需要两个实现方案会令人畏缩，并最终将变成维护梦魇。

开发 WebSockets 的整体理念是解决通过 HTTP 双向传输数据的基本问题。如前所述，HTTP 是一种简单的协议，最适合于请求-响应情形。随着人们用于通信的方式越来越丰富，Web 也变得越来越多样化，HTTP 已变成瓶颈。开发 WebSockets 协议的主要动力就是解决这个问题。WebSockets 规范的一部分还关注 WebSockets API，它为创建 WebSockets 连接提供了非常基本的支持——通过该连接收发数据。这些 API 没有为编写实时应用程序提供任何更高级的编程构造，例如管理多个连接、错误恢复等。

最近 10 年来，Web 访问的最大变化是 Web 的访问方式。桌面浏览器不再是人们浏览 Web 的唯一方式。带有不同窗体元素的设备呈爆炸式增长，从智能手机到平板电脑，应有尽有。除了浏览器之外，这些窗体元素揭示了人们可用于与应用程序交互操作的各种形式。在此变化中，用户使用触摸屏与应用程序交互操作，而不是传统的鼠标和键盘。

这些变化给应用程序开发人员带来了挑战。他们编写的实时应用程序必须能工作在不同的环境中和设备上。解决这些问题需要一个库，且便于开发人员编写应用程序，而不必担心环境和设备的底层细节。这就是 SignalR 诞生的原因，它使实时应用程序的开发更简单、更有趣。

26.3.1 什么是 SignalR?

SignalR 是建立在 ASP.NET 上的库，非常便于把实时功能添加到 Web 应用程序中。在 WebSockets 可用时，SignalR 就在后台使用 WebSockets，WebSockets 在不可用时，SignalR 就采用其他技术。这样，应用程序代码总是一致的。

SignalR 还提供了一个非常简单的高级 API，以编写实时应用程序，并给连接的管理提供有效的关联。它还支持全新类型的、需要在服务器上频繁更新的应用程序(例如实时游戏)。图 26-9 显示了 SignalR 的高级功能。

SignalR 支持不同的环境和设备。它会确定客户端和服务端支持什么，选择建立实时连接的最佳

方式。如果客户端和服务端都支持 WebSockets，就使用 WebSockets，否则就使用下一个最佳选项。这表示，应用程序会继续在不同的 Windows 版本中无缝地工作在不同的浏览器上，无须修改应用程序代码。SignalR 的另一优点是提供了 API，所以可以为移动设备和平板电脑建立实时应用程序，这样不同设备的客户端连接就可以使用相同的服务器端代码。

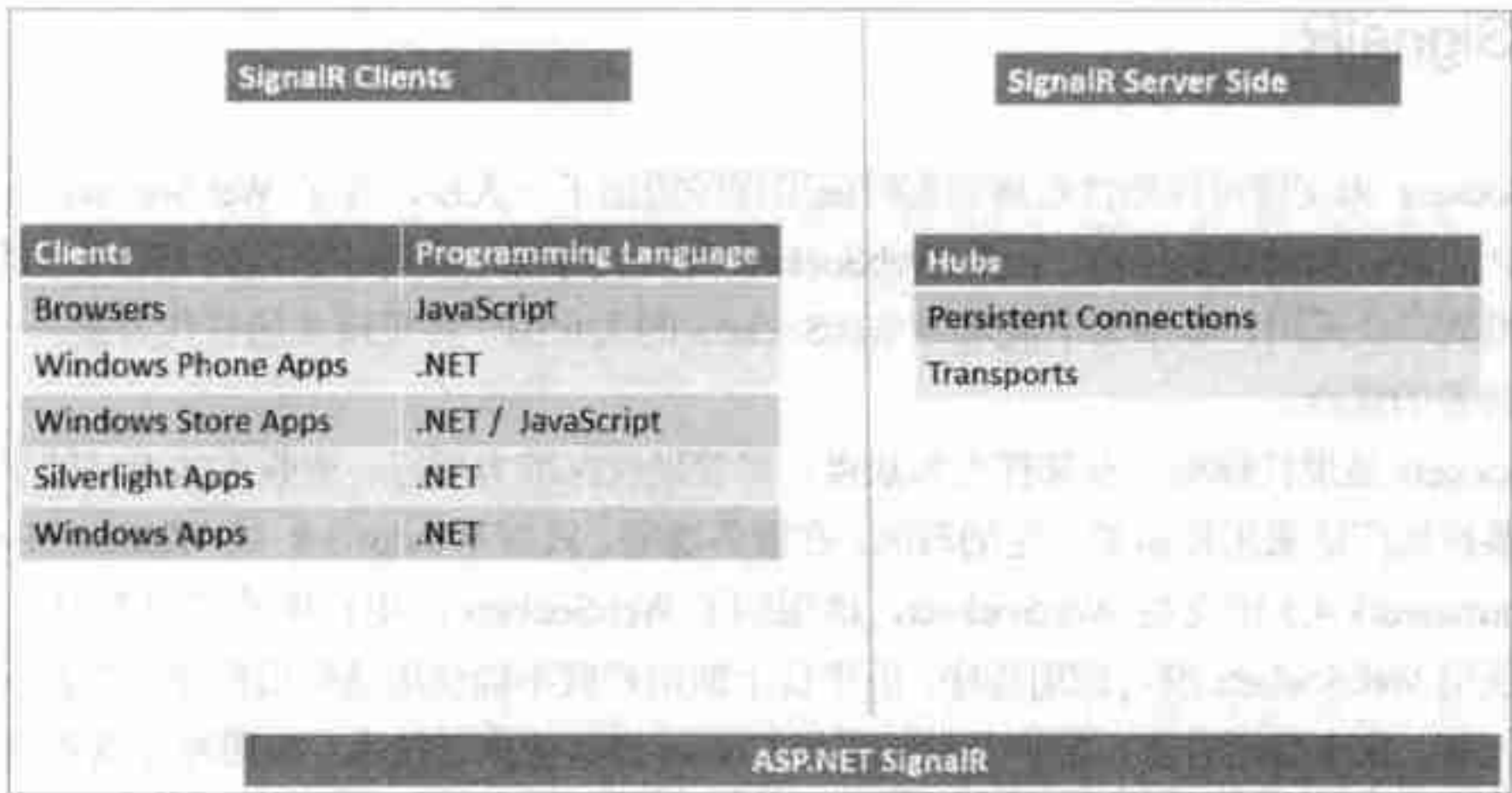


图 26-9

26.3.2 ASP.NET 中的服务器端 SignalR

在 ASP.NET 应用程序中使用 SignalR 的方式与使用 WebSockets 类似。其中，由服务器端部分负责接收来自客户端的连接，并在所有客户端之间传输数据。SignalR 和 WebSockets API 的区别是，SignalR 提供了管理连接的额外功能，确保所有的客户端都可以无缝地交换数据。程序清单 26-3 说明了 HTTP 处理程序如何处理 WebSockets 连接，服务器端点是集线器，客户端可以调用它。它还管理所有的连接，收发各个客户端的数据。程序清单 26-5 是可用于聊天程序的集线器示例。

与 WebSockets 响应处理程序不同，集线器不需要任何代码来接收连接。开发人员看不到建立连接和选择最佳协议的底层细节，这样他们就可以只考虑应用程序代码，而不用理会协议。

程序清单 26-5 典型的 ASP.NET SignalR 聊天集线器

```
public class CSChat : Hub
{
    public void Send(string message)
    {
        // Call the addMessage method on all clients
        Clients.All.addMessage(message);
    }
}
```

SignalR 执行了服务器到客户端的 RPC(在客户端调用服务器端.NET 代码中的函数)。例如，在程序清单 26-5 中，addMessage 是 Web 页面上的 JavaScript 函数，如程序清单 26-6 所示，该函数在服务器端的集线器中调用。

集线器在服务器端提供了许多高级功能，包括安全级别。它们可以检查客户端是否通过了验证，是否授权与服务器通信。集线器可以用各种方式对所有已连接的客户端，或者发送了消息的客户端，

或者特定的客户端广播消息。集线器还提供了事件，在客户端连接、断开连接或重新连接时触发这些事件。这些功能便于跟踪客户端，传输消息。

26.3.3 ASP.NET 中的客户端 SignalR

在应用程序中使用 SignalR 的另一方是客户端。SignalR 可以用于不同类型的客户端。本节介绍如何在浏览器可访问的 Web 页面中使用 SignalR。这个过程类似于程序清单 26-4，它说明了如何在 JavaScript 中使用 WebSockets API 建立与程序清单 26-3 中响应处理程序的 WebSockets 连接。

运行应用程序时，SignalR 会在客户端生成一个代理程序，这样客户端就可以通过该代理程序调用服务器上的函数。如程序清单 26-6 所示，客户端调用服务器上定义的 send 函数，该函数见程序清单 26-5。

程序清单 26-6 在 Web 页面上使用了一个 SignalR JavaScript 库。

程序清单 26-6 典型的 ASP.NET SignalR JavaScript 客户端代码

```
<!DOCTYPE html>
<script src="Scripts/jquery-1.6.4.js"></script>
<script src="Scripts/jquery.signalR-1.0.1.js" type="text/javascript"></script>
<script src="<%= ResolveClientUrl("~/signalr/hubs") %>"
    type="text/javascript"></script>
<script type="text/javascript">
    $(function() {
        // Proxy created on the fly
        var chat = $.connection.csChat;

        // Declare a function on the chat hub so the server can invoke it
        chat.client.addMessage = function(message) {
            $('#messages').append('<li>' + message + '</li>');
        };

        // Start the connection
        $.connection.hub.start().done(function () {
            $("#broadcast").click(function() {
                // Call the chat method on the server
                chat.server.send($('#msg').val());
            });
        });
    });
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>
<input type="text" id="msg" />
<input type="button" id="broadcast" value="send" />

<ul id="messages">
</ul>
```



```
</div>
</form>
</body>
</html>
```

如程序清单 26-6 所示，要建立与集线器的连接，应调用 SignalR 生成的代理程序/signalr/hubs。连接启动后，就可以通过该连接收发数据了。图 26-10 是以程序清单 26-5 和 26-6 中的代码为基础的聊天示例程序。在应用程序中，要实现基本的聊天功能，只须编写前两个程序清单中的代码。运行应用程序，就可以使用不同的浏览器浏览到这个应用程序。从浏览器发送文本消息时，SignalR 会自动把该消息发送给所有其他已连接的客户端。

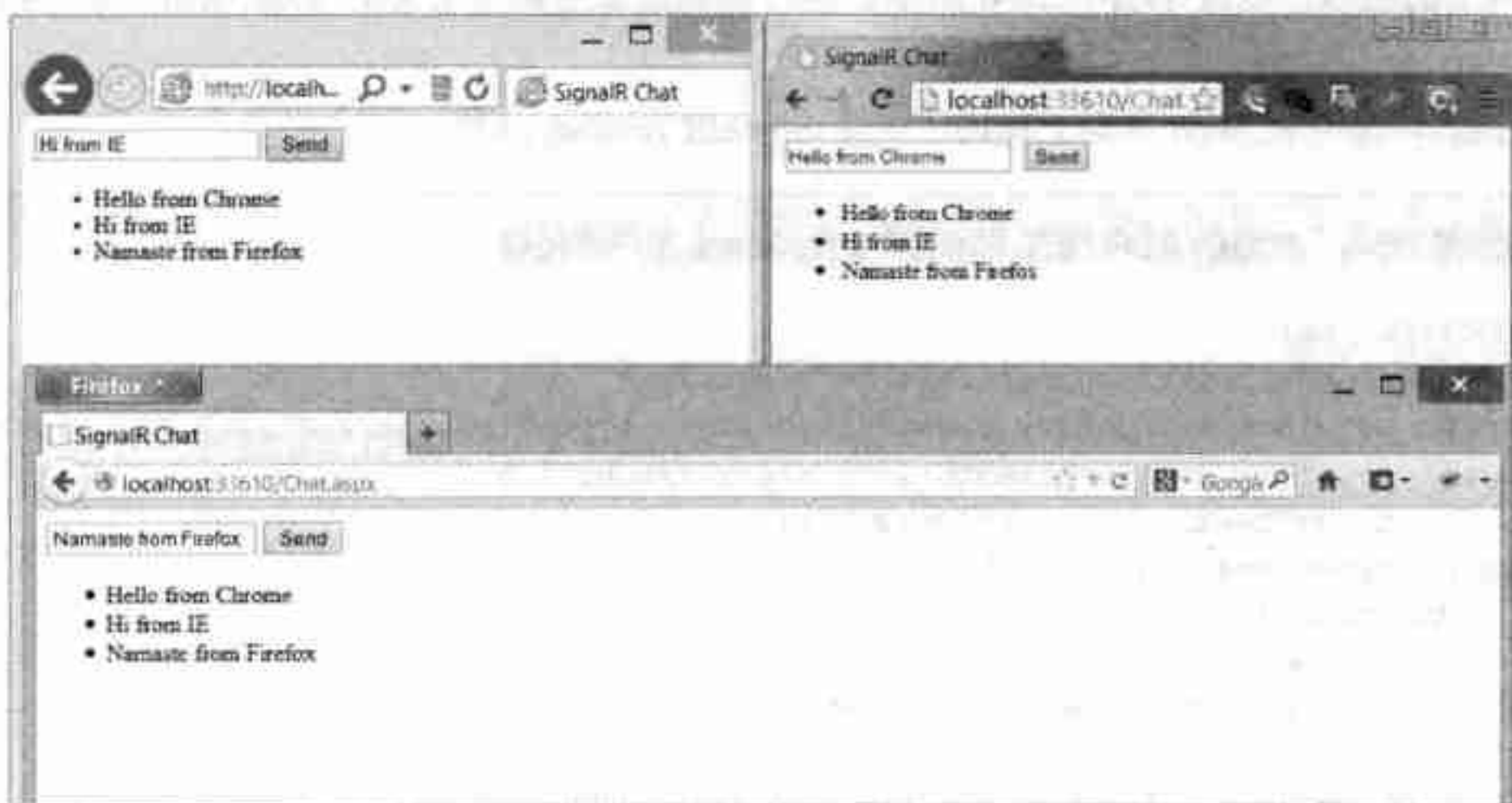


图 26-10

使用 SignalR 的最大优点是“编写一次，就可以在任何地方运行”。开发人员只须使用 SignalR 编写应用程序，就能根据用户使用的平台，选择最佳传输方式进行通信。

SignalR 建立在 ASP.NET 的基础上，所以可以利用所有熟悉的 ASP.NET 概念，例如路由、验证和授权，建立 SignalR 应用程序时也可以使用它们。

26.4 本章小结

Web 在不同类型的交互操作和用户通信模型方面有很大的进步。用户越来越被更社会化的应用程序所吸引，所以实时通信的需求也增长迅猛。社会化交互操作方式的使用，由于提供了实时协作环境，提高了用户的期望，因而迫使 W3C 考虑升级常用通信协议的创新方式，让它们支持实时通信。

本章介绍了支持实时应用程序的已有的各种方式，这些旧方法的一些缺点使它们在现代 Web 环境中的表现欠佳。为了解决这些问题，人们把 WebSockets 协议引入为 HTML5 规范的一部分。WebSockets 为确保实时应用程序的良好执行和伸缩带来了巨大的突破。

最后，本章简要介绍了 SignalR，SignalR 更便于实现“编写一次，就可以在任何地方运行”，因而大大提升了编写实时应用程序的开发体验。

第 27 章

开发移动网站

本章要点

- 响应式 Web 设计的概念
- 检测移动设备
- ASP.NET 的移动功能
- 理解 ASP.NET MVC 4 的移动功能
- 测试移动站点的显示效果

目前，成功的网站意味着移动设备可以访问。可浏览互联网的智能手机数继续在增长，使用它们上网的人数也在增长。我们不能安全地假定某个网站不能用移动设备浏览。对于许多公司而言，有个移动网站是增加收益的源泉。甚至对于不依赖移动流量来获得收入的公司，它们的客户也开始期望能在移动设备上浏览站点了。随着移动设备的使用在增长，移动 Web 设计也比以前更重要。

本章介绍便于适应移动设备的技术。虽然 ASP.NET 提供了有用的工具，但并不是所有的解决方案都依赖某种特定的技术。

无线标记语言的起源

尽管移动设计在近年来有很大的变化，但开发移动网站的概念并不是新的。其实，Wireless Markup Language (无线标记语言，WML) 是 1998 年开发的，它是一种基于 XML 的语言，是专门为开发移动设备的网站而设计的。微软早在 .NET Framework 2 版本就为 ASP.NET 创建了一组移动控件。System.Web.UI.MobileControls 名称空间中的控件会根据浏览器生成 WML、CHTML 或 HTML。该名称空间在 .NET Framework 4 版本中已被标记为废除，因为有了更现代的移动 Web 开发标准。Visual Studio 2010 和 ASP.NET MVC 引入了一系列工具，使移动网站的开发容易了许多。Visual Studio 2012 和 ASP.NET MVC 的最新版本继续改进移动 Web 开发的工具，使得更容易建立移动站点。

27.1 移动 Web 设计的挑战

尽管开发移动网站的工具和技术变了，但挑战没有变化，这些挑战是：

- 屏幕大小：这是最明显的挑战。使用较小的屏幕常常意味着，布置网站的选项在移动设备上完全不同。可能需要把桌面浏览器上的多个水平栏改为一个堆叠的栏，以适应较小的屏幕。也可能需要铺平复杂的水平导航按钮，使它们垂直显示。
- 带宽：这常常被忽视。使用桌面浏览器时，用户可能有可用性很高、带宽很大的互联网连接。移动设备的连接带宽一般较小，有时还可能断开连接。更不用说，移动设备上的带宽通常较贵。在给移动设备提供内容时，重要的是有效利用带宽。在建立富桌面网站时，可能需要依赖许多大型客户端脚本库和大型 CSS 文件，而在设计小型移动浏览器时，这可能并不可行。
- 图片大小：移动网站不仅要考虑客户端脚本库和 CSS 文件，还必须考虑图片的大小，这不仅仅因为会增加带宽，还因为图片的物理大小。理想情况下，用于移动设备的图片应在像素大小和文件大小上都显著小于桌面浏览器。
- 用户的交互操作：与移动设备的交互操作显著不同于与桌面设备的交互操作。移动设备使用触摸和手势，而桌面设备使用鼠标和键盘。在桌面屏幕上用鼠标选择元素，大大不同于在移动屏幕上触摸元素。触摸的准确率要低得多，元素也必须较大，用户才能选中。在移动设备上进行键盘输入会困难得多，因为键很小，但这在不同的设备上会有所不同。一些设备可以使用虚拟键盘，而其他设备有物理键盘。

移动 Web 开发人员面临的挑战还不止于此，但上述是为移动设备开发网站时的主要挑战。克服这些困难常常需要经验，这包括成功和失败的经验。Visual Studio 2012、.NET Framework 4.5 和 ASP.NET MVC 最新版本中的工具肯定有助于简化移动网站的开发。

27.2 响应式设计和适应式设计

为移动浏览器设计站点不再是例外，而是规则。大多数成功的网站现在都明确支持移动浏览器。与大多数成为主流的技术一样，围绕移动网站的术语也是混乱不堪的。也许读者听说过响应式设计、渐进式增强和适应式设计等术语。它们的区别是什么？

- 适应式设计一般包含了允许 Web 页面适应设备大小和方向的各种技术。没有所谓适应式设计的单一技术。适应式设计通过联合使用多个技术来实现。
- 渐进式增强是指首先设计最小功能集，再根据客户端对它们的支持，添加功能或样式的实践活动。
- 响应式设计是一种适应式设计，合并了根据显示网站的浏览器来拉伸和重新安排网站的技术。另外还允许站点响应浏览器在大小和方向上的实时变化。

这些术语常常联合使用。响应式设计示例的技术也可能是渐进式增强的示例，反之亦然。例如响应式设计中使用的两个技术是：包括 viewport 元标记和利用 CSS 媒介查询。利用 CSS 媒介查询的技术也可以在渐进式增强中使用。

27.2.1 修改视口

浏览器显示网站时，网站是在视口中显示的。视口只是一块可用于查看网站的区域。使用移动浏览器浏览网站时，默认情况下，浏览器会假定用户在桌面上浏览网站。浏览器假定用户希望查看整个网站，而不是网站的一角。因此它把视口的宽度设置为远大于物理屏幕的宽度，使视口把整个站点显示在小屏幕上。为了演示这一点，下面创建一个新的 ASP.NET Empty Web Application 项目 HTMLMobile，再创建一个新的页面。右击该项目，选择 Add | New Item，接着选择 Html Page，命名为 default.html。default.html 页面的内容就显示在程序清单 27-1 中(本章下载代码中的 defaultbasic.html 文件)。

程序清单 27-1 default.html 页面

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Responsive Design Example</title>
  <style>
    header{ background-color: #808080; }
    footer{
      background-color: #808080;
      text-align: left;
      height: 50px;
      clear: both;
    }
    nav{ text-align:left; }
    nav a{
      padding-left: 10px;
      padding-right: 10px;
      color: black;
    }
    #content{
      padding-left: 10px;
      border-right: 1px solid black;
      width: 45%;
      float: left;
    }
    #news{
      padding-left: 10px;
      width: 45%;
      float: left;
    }
  </style>
</head>
<body>
  <header>
    <nav>
      <a href="#">Home</a>
      <a href="#">Page 1</a>
      <a href="#">Page 2</a>
      <a href="#">About</a>
    </nav>
  </header>
```

```

<div id="content">
  <h1>Welcome!</h1>
  Welcome to responsive design example.
</div>
<div id="news">
  <h2>News</h2>
  Breaking news!
</div>
<footer>
  Copyright &copy; Examples By U
</footer>
</body>
</html>

```

使用移动浏览器浏览页面，很容易看出视口的作用。图 27-1 显示了在移动浏览器中查看的 default.html 页面。

可以看出，浏览器在显示页面时，就好像是把页面显示在一个更大的屏幕上，再把页面缩小，使页面能显示在移动浏览器的屏幕上。屏幕上的许多元素都太小了，无法阅读，更不能用于交互操作。为了解决这个问题，需要告诉浏览器使用与设备屏幕相同宽度的视口。为此，应在 `<head>` 标记之间添加 `viewport` 元标记。程序清单 27-2(本章下载代码中的 defaultviewport.html 文件)显示了添加 `viewport` 元标记后 default.html 文件的一部分。



图 27-1

程序清单 27-2 带有 viewport 元标记的 default 页面

```

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Responsive Design Example</title>
  <meta name="viewport" content="width=device-width" />
  <style>
    header
    {
      background-color: #808080;
    }
  </style>

```

图 27-2 是添加 `viewport` 元标记后显示在移动浏览器中的页面。

很容易看出，`viewport` 元标记完成了这项工作。现在站点内容的可读性更高。但现在有个新问题。尽管内容不再很小，但页面的布局并不适合小屏幕。页面上的元素开始重叠了。站点的比例是正确的，但布局需要适应屏幕的大小。



图 27-2

27.2.2 使用 CSS 媒介查询

强制浏览器根据屏幕的大小缩放其内容只是第一步。但在大多数情况下，为大屏幕设计的布局可能不适合小屏幕。如果页面的布局设计为使用 CSS，就可以使用不同的 CSS 改变布局。问题是如何告诉浏览器为不同大小的屏幕使用不同的 CSS。答案是使用 CSS 媒介查询。

CSS 媒介查询可以使 CSS 规则面向特定的显示特征。用 CSS 规则指定媒介类型，如屏幕和打印，这一功能自从 HTML4 和 CSS2 就是可用的。但是，媒介查询扩展了该功能，使之包含其他条件。媒介查询包含媒介类型和 0 个或多个表达式，以检查特定媒介功能的条件。可以使用的媒介功能有宽度、高度和颜色。面向不同的屏幕大小时，最常见的条件是宽度。尽管可以使用其他条件，但常用的条件是 `min-width` 和 `max-width`。

使用 CSS 媒介查询可以实现渐进式增强，但并不总是能采用渐进式增强技术。有时，尤其是给已有网站添加移动功能时，起点就已经设置好了。但在这个示例中，可以修改站点的布局，使其默认面向最小功能集，即移动浏览器。接着使用 CSS 媒介查询，添加额外的样式规则，提供另一个布局，以适应不同的屏幕大小。程序清单 27-3 在本章下载代码的 `defaultqueries.html` 文件中，它演示了进行修改后，样式块默认面向移动浏览器，再逐步面向更大的屏幕。

程序清单 27-3 添加 CSS 媒介查询后的样式块

```
<style>
  header{ background-color: #808080; }
  footer{
    background-color: #808080;
    text-align: center;
    height: 50px;
  }
  nav{ text-align:center; }
  nav a{
    display: block;
    padding-left: 10px;
    padding-right: 10px;
    color: black;
  }
  #content{ padding-left: 10px; }
  #news{ padding-left: 10px; }
  @media screen and (min-width: 601px)
  {
    footer{ clear: both; }
    nav a{ display: inline; }
    #content{
      border-right: 1px solid black;
      width: 45%;
      float: left;
    }
    #news{
      width: 45%;
      float: left;
    }
  }
  @media screen and (min-width: 901px)
  {
    footer{ text-align: left; }
    nav{ text-align: left; }
  }
</style>
```


对于上述代码，第一组样式规则应用为面向移动浏览器的基本规则集。注意 nav 和 footer 元素改为指定要居中的文本，锚定元素的显示值是 block。content 和 news 元素也删除了其浮点值，因此垂直堆叠它们，而不是水平堆叠。垂直显示的元素常常比水平显示更适合移动浏览器，因为水平空间是有限的。

媒介查询定义了一个样式块，只有当媒介类型是 screen，且屏幕的最小宽度是 601 像素时，才应用这个样式块。601 像素到 900 像素的屏幕宽度适合许多平板设备。注意这个块中的元素规则替代了许多布局样式，这些布局样式已从面向最小功能集(移动浏览器)的设备中删除：

```
@media screen and (min-width: 601px)
```

另一个媒介查询也定义了一个样式规则块，只有当媒介类型是 screen，且屏幕的最小宽度是 901 像素时，才应用这个样式块。因为许多平板设备使用的屏幕宽度至多为 900 像素，所以这个块中定义的规则适合更大的屏幕，例如桌面。应用下面的媒介查询时，只需调整菜单项和页脚文本。尽管这看起来很简单，但却说明了渐进式增强的多个级别。

```
@media screen and (min-width: 901px)
```

图 27-3 是移动浏览器中显示的页面，而图 27-4 是桌面浏览器中显示的页面。



图 27-3

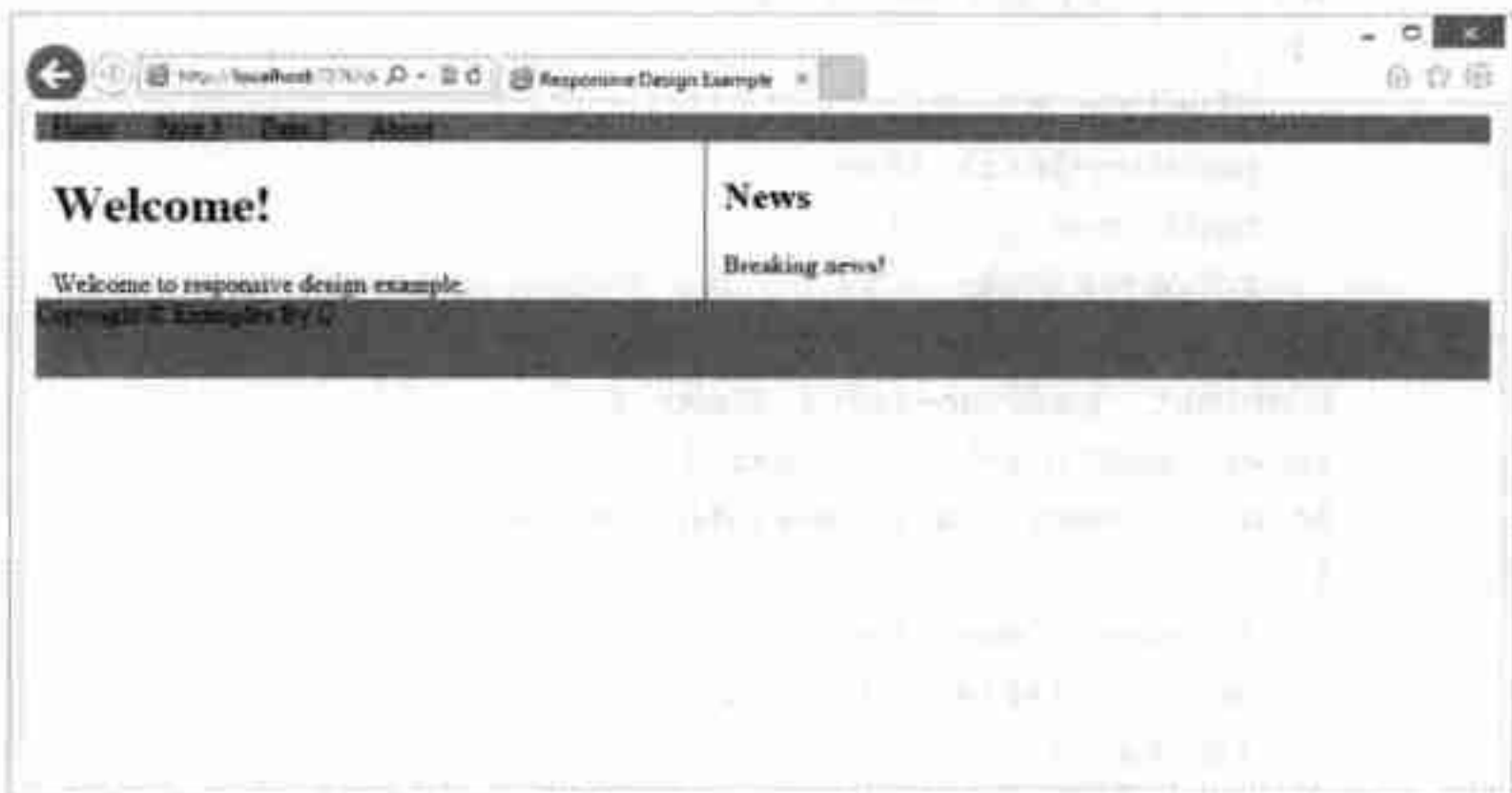


图 27-4

27.3 ASP.NET 移动应用程序

使 ASP.NET 应用程序面向移动设备涉及客户端和服务端上的修改。选择只修改其中一端，无法提供完整的解决方案。在移动 Web 开发中，这是个有许多争议的领域。一些人认为，无论是在桌面还是移动设备中查看，应用程序中的内容都应相同。此时，用户在移动设备中查看内容时，唯一要修改的仅仅是内容的布局，以及用于显示内容的元素样式。这是响应式 Web 设计方法。其他人则认为，移动用户可能不需要看到相同的内容，至少不完全相同。此时，在桌面和移动设备上查看内容的区别就更大。

只对客户端进行修改,例如针对响应式设计进行修改,应用程序可以更好地缩放,以适应移动设备屏幕的大小。然后,应用程序可以响应方向的改变。可以提供更适合设备屏幕大小的布局和样式选项。但是,试图以单个页面适应所有可能的设备,意味着要包含不必要的标记、CSS 和 JavaScript,因而增加了需要的带宽。

对服务器进行修改涉及给移动设备提供专用的页面。这可能导致某些效果和代码的重复,但也允许完全控制用户在桌面浏览器和移动浏览器中看到的内容。采用以服务器为中心的方法,就必须把常见的逻辑和功能封装到库中,这个库可以在页面的两个不同版本之间共享。

如果从头开始阅读本章,就很熟悉可以在客户端代码中进行的修改,以创建对移动设备更友好的页面。本节介绍建立移动应用程序的以服务器为中心的方法。

首先创建一个新的 ASP.NET Web Forms 应用程序,命名为 WebFormsMobile。

27.3.1 检测移动浏览器和设备

第一个要克服的困难是确定如何检测查看应用程序所用的浏览器的特性。如果要根据浏览器的信息在服务器上作出决定,了解这些信息就是必需的。幸好,ASP.NET 提供了 `Request.Browser` 对象,它可以检索所用浏览器的特性信息。尽管 `Request.Browser` 对象有许多有趣、有益的属性,但专门用于移动浏览器的属性有 3 个:

- `MobileDeviceManufacturer`: 返回设备厂商的名称。
- `MobileDeviceModel`: 返回设备的型号。
- `IsMobileDevice`: 尽管面向某些移动厂商和型号可能很有趣,但这种方法并不可行,因为有太多不同的移动设备。更有用的属性是 `IsMobileDevice`,它返回的值只表示请求是否来自于移动设备。

27.3.2 处理移动母版页

了解了如何检测请求页面的浏览器的功能后,就可以把移动设备专用的页面传递给移动浏览器。只要使用移动设备专用的母版页,就可以处理移动页面和标准页面之间的差别。要在应用程序的根目录中创建新的母版页,应右击项目,选择 `Add | New item`,再从 `Add New item` 对话框中选择 `Master Page`,命名为 `MobileSite.Master`。下面创建标准母版页文件的简化版本,用于移动设备。程序清单 27-4 显示了 `MobileSite.Master` 文件的 C# 代码。这些代码在本章下载代码的 `WebFormsMobileCS` 项目根目录下的 `MobileSite.Master` 文件中。

程序清单 27-4 MobileSite.Master 文件

```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="MobileSite.master.cs"
    Inherits="WebFormMobile.MobileSite" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title></title>
    <asp:ContentPlaceHolder ID="head" runat="server">
    </asp:ContentPlaceHolder>
```



```

<meta name="viewport" content="width=device-width" />
</head>
<body>
  <form id="form1" runat="server">
    <div>
      This is a mobile browser!

      <asp:ContentPlaceholder ID="FeaturedContent" runat="server">
      </asp:ContentPlaceholder>

      <asp:ContentPlaceholder ID="MainContent" runat="server">
      </asp:ContentPlaceholder>
    </div>
  </form>
</body>
</html>

```

创建了母版页的简化移动版本后，就要命令在移动浏览器中显示的所有页面使用 MobileSite.Master 母版页。为此，重写每个页面的 Page_PreInit 处理程序，并修改每个页面要使用的母版页。首先，当用户在移动浏览器中查看项目中的主页面时，把主页面改为使用 MobileSite.Master。程序清单 27-5 显示了重写 Page_PreInit 处理程序后主页面的隐藏代码，如果主页面在移动设备上查看，主页面就改变母版页。这个程序清单的代码在本章下载代码的 WebFormsMobileCS 项目根目录下的 Default.aspx.cs 文件中。

程序清单 27-5 重写 Page_PreInit 处理程序后主页面的隐藏代码

```

using System;
using System.Web.UI;

namespace WebFormMobile
{
  public partial class _Default : Page
  {
    protected void Page_PreInit(object sender, EventArgs e)
    {
      if (Request.Browser.IsMobileDevice)
        MasterPageFile = "~/MobileSite.Master";
    }

    protected void Page_Load(object sender, EventArgs e)
    {
    }
  }
}

```

执行应用程序，在移动浏览器中查看，会得到如图 27-5 所示的结果。



图 27-5

27.3.3 创建移动 Web 窗体

仅修改母版页可能还不够。例如，需要更多地修改提供给移动浏览器的内容。此时，可以提供一组平行的页面，它们是专门为显示在移动浏览器中而设计的。尽管这样会产生一些重复的代码，但也提供了一组只有移动浏览器可以使用的专用页面。对于应用程序中可用于移动浏览器的每个页面，都建立专用的移动版本。为此，应在应用程序的根目录下创建文件夹 Mobile，接着在这个文件夹中创建新 Web 窗体 Default.aspx。这个 Web 窗体是根文件夹中标准的默认 Web 窗体的移动专用版本。为了使示例更便于管理，该 Web 窗体不应包含任何样式或脚本，而只包含一些简单的内容。新的 Default.aspx 文件的 C# 版本如程序清单 27-6 所示(本章下载代码中的/Mobile/Default.aspx 文件)。

程序清单 27-6 移动专用的 Default.aspx

```
<%@ Page Language="C#" MasterPageFile="~/MobileSite.Master" AutoEventWireup="true"
    CodeBehind="Default.aspx.cs" Inherits="WebFormMobile.Mobile.Default" %>

<asp:Content runat="server" ID="FeaturedContent"
    ContentPlaceHolderID="FeaturedContent">
    <p>
        Welcome to the Mobile version of the site!
    </p>
</asp:Content>

<asp:Content runat="server" ID="BodyContent" ContentPlaceHolderID="MainContent">
    <p>
        This is a mobile version of the site!
    </p>
</asp:Content>
```

如果在创建的 Mobile 文件夹中复制所有标准的 Web 窗体，包括相关的文件夹，重定向到移动浏览器就会容易得多。只需把标准 Web 窗体的第一个请求重定向到 Mobile 文件夹中对应的 Web 窗体即可，后续的请求会利用 Mobile 文件夹中的相对路径，不需要重定向。另外，如果希望允许用户在移动浏览器中切换到应用程序的标准桌面版本，只需提供返回 Web 窗体标准版本的链接。相对路径会使后续的请求都指向标准 Web 窗体文件夹结构。

最初把移动浏览器重定向到 Mobile 文件夹结构可以在浏览会话的开始进行。如果应用程序使用的是 C# 语言，就在 Global.asax.cs 文件中添加 Session_Start 处理程序。程序清单 27-7 显示了 Session_Start 处理程序的代码。这个程序清单的代码在本章下载代码的 WebFormsMobileCS 项目根目录下的 Global.asax.cs 文件中。

程序清单 27-7 Global.asax 文件中的 Session_Start 处理程序

```
void Session_Start(object sender, EventArgs e)
{
    HttpRequest httpRequest = HttpContext.Current.Request;
    if (httpRequest.Browser.IsMobileDevice)
    {
    }
```

```

string path = httpRequest.Url.PathAndQuery;
bool isOnMobilePage = path.StartsWith("/Mobile/",
    StringComparison.OrdinalIgnoreCase);
if(!isOnMobilePage)
{
    string redirectTo = "~/Mobile/";
    HttpContext.Current.Response.Redirect(redirectTo);
}
}
}

```

如果执行应用程序,浏览到站点的根目录,就会看到 Mobile 文件夹中新的主页面,而不是标准的主页面。图 27-6 是在移动浏览器中显示的移动专用的主页面。



图 27-6

27.3.4 ASP.NET Web 窗体中的 FriendlyURLs

ASP.NET Web 应用程序中的路由提供了一种方式,可以给用户提供更易记忆和理解的 URL。无论怎样创建路由、管理路由表,都需要花一定的时间来理解。ASP.NET 的新功能 FriendlyUrls 使路由更容易,URL 更简洁。

FriendlyUrls 包含在 Visual Studio 2012.2 更新包中,但可以作为 NuGet 包用于 C# 项目。NuGet 包的名称是 Microsoft.AspNet.FriendlyUrls。要在 WebFormsMobileCS 项目中安装 NuGet 包,可以使用在 Visual Studio 2012 中默认安装的 NuGet 包管理器。在 Visual Studio 菜单中单击 Tools | Library Package Manager | Package Manager Console,打开 Package Manager Console。在控制台提示符中,运行下面的命令:

```
Install-Package Microsoft.AspNet.FriendlyUrls
```

安装 NuGet 包 FriendlyUrls 时,RouteConfig.cs 文件会添加到 Web 应用程序根目录下的 App_Start 文件夹中。必须在 Web 应用程序根目录下的 Global.asax 文件中给 Application_Start 处理程序添加一行代码。在 Application_Start 处理程序中添加如下代码:

```
RouteConfig.RegisterRoutes(RouteTable.Routes);
```

现在,如果执行应用程序,就可以浏览到任意.aspx 页面,而无须指定扩展名。例如,为了浏览到 Contact.aspx 页面,只需在 URL 中指定 Contact,不需要扩展名,FriendlyUrls 就会查找 Contact.aspx 文件。图 27-7 是只在 URL 中指定 Contact,在移动浏览器中显示的 Contact.aspx 页面。

使用 FriendlyUrlSettings 对象,也可以告诉 ASP.NET 自动从 URL 中去除页面的扩展名。处理 WebFormsMobileCS 等已有的项目时,你不希望检查项目中的每个 Web 窗体,修改每个链接和 URL,以去除.aspx 扩展名。此时可以使用 FriendlyUrls 完成这项工作。打开 App_Start 文件夹中的 RouteConfig.cs 文件。这个文件是 NuGet 包 FriendlyUrls 添加的。把



图 27-7


```
routes.EnableFriendlyUrls();
```

改为

```
FriendlyUrlSettings settings = new FriendlyUrlSettings();
settings.AutoRedirectMode = RedirectMode.Permanent;
routes.EnableFriendlyUrls(settings);
```

完成此修改后, 执行应用程序, 浏览到 Web 窗体 Contact.aspx, ASP.NET 会把对/Contact.aspx 的 GET 请求改为到/Contact 的 301 重定向。

27.4 ASP.NET MVC 4 移动应用程序

ASP.NET MVC 架构用于建立基于标准的、便于缩放和测试的 Web 应用程序, 该架构使用构建得很成功的 Model-View-Controller 设计模式。在 ASP.NET MVC 3 中, 引入了一组功能, 以便于移动 Web 开发。即使有了这些功能, 也很难在这么多不同的移动浏览器和平台上提供一致的开发方式。ASP.NET MVC 4 改进、增强了最初在 ASP.NET MVC 3 中引入的功能。这些新增功能旨在更便于在更多的设备和浏览器中提供一致的移动开发方式。

27.4.1 ASP.NET MVC 4 中的适应式显示功能

在 ASP.NET MVC 4 中, Visual Studio 项目模板对 ASP.NET MVC 3 进行了许多修改, 最重要的是整体可视化设计。即使打算使用新模板提供的设计, 较有趣的变化也是包含提供适应式显示功能的元素。使用 viewport 元标记和 CSS 媒介查询允许这些新模板提供页面的更好缩放效果和适应小屏幕的已修改布局。图 27-8 是移动浏览器中的 ASP.NET MVC 3 默认模板, 图 27-9 是移动浏览器中的 ASP.NET MVC 4 默认模板。忽略模板设计的区别, 两个模板中的缩放效果区别也很明显。ASP.NET MVC 3 模板在显示页面时, 尝试先显示在桌面浏览器中, 再缩放到 Windows Phone 屏幕的大小。ASP.NET MVC 4 模板则使用适应式显示功能, 使页面用更适合屏幕大小的比例来显示。从这两幅图中看不出来的是, ASP.NET MVC 4 模板还修改了页面布局, 以更好地在小屏幕上显示页面内容。



图 27-8



图 27-9

为了查看这些新模板提供的适应式显示元素,可以创建一个 ASP.NET MVC 项目。首先在 Visual Studio 中创建新项目 MvcMobile。选择 .NET Framework 4.5 作为目标框架,并选择 ASP.NET MVC 4 Web Application 项目类型。由于创建新 ASP.NET MVC Web Application 时的可用模板很多,所以这是个两步骤的过程。选择 ASP.NET MVC 4 Web Application 不会立即创建 Web 应用程序,而必须选择要使用的 ASP.NET MVC 4 应用程序模板的类型。选择了 ASP.NET MVC 4 Web Application 后,会显示模板选择对话框。选择 Internet Application 模板。图 27-10 显示了做出正确选择的 New ASP.NET MVC 4 Project 对话框。

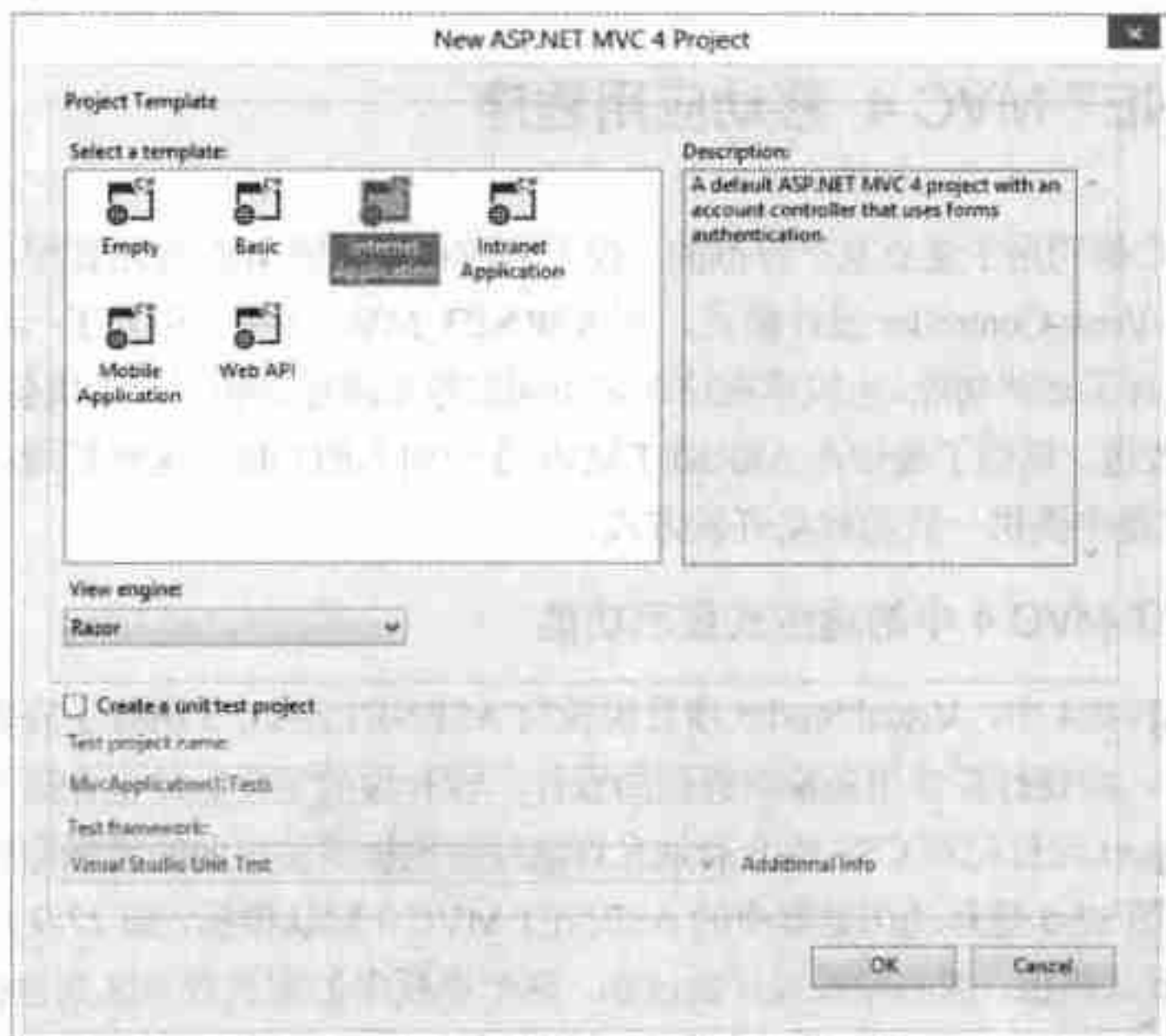


图 27-10

1. 使用 viewport 元标记

如果已学习本章前面的响应式设计,就很熟悉 viewport 元标记,它告诉浏览器,在显示 Web 页面时,应考虑浏览器显示区域的物理大小。图 27-10 中显示的 Internet Application、Intranet Application 和 Mobile Application 模板包含默认的 Layout 文件,Layout 文件已经包含 viewport 元标记。程序清单 27-8 显示了该布局文件的部分 C# 代码,使用 Internet Application 模板时,该布局文件会包含在项目中。布局文件位于项目根目录的 Views\Shared 文件夹下。这个程序清单中的代码在本章下载代码的 _Layout.cshtml 文件中。

程序清单 27-8 ASP.NET MVC 4 应用程序中的默认布局文件

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>@ViewBag.Title - My ASP.NET MVC Application</title>
    <link href="~/favicon.ico" rel="shortcut icon" type="image/x-icon" />
    <meta name="viewport" content="width=device-width" />
```

```

    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
<body>
    <header>
        <div class="content-wrapper">
            <div class="float-left">
                <p class="site-title">
                    @Html.ActionLink("your logo here", "Index" "Home")
                </p>
            </div>

```

页面开头部分的 **viewport** 元标记告诉浏览器，使用设备屏幕的宽度作为页面布局的宽度：

```
<meta name="viewport" content="width=device-width" />
```

2. 使用 CSS 媒介查询

CSS 媒介查询使用媒介功能，例如宽度、高度和颜色，来限制样式表的使用范围。图 27-10 中显示的 Internet Application、Intranet Application 和 Mobile Application 模板也包含默认的 CSS 文件 Site.css，该 CSS 文件包含许多默认的样式规则，还包含另一组通过 CSS 媒介查询来过滤或限制的样式规则。程序清单 27-9(本章下载代码中的\Content\Site.css)显示了 Site.css 文件包含 CSS 媒介查询的部分代码。

程序清单 27-9 Site.css 文件中的 CSS 媒介查询

```

/*****
 *   Mobile Styles   *
 *****/
@media only screen and (max-width: 850px) {

    /* header
    -----*/
    header .float-left,
    header .float-right {
        float: none;
    }

    /* logo */
    header .site-title {
        margin: 10px;
        text-align: center;
    }

```

该例使用 CSS 媒介查询提供了一组样式规则，以根据浏览器屏幕的宽度改变页面的布局。媒介查询指出，不使用 CSS 文件的 Mobile Styles 部分包含的样式规则，除非媒介类型是 **only screen**，且宽度小于等于 850 像素：

```
@media only screen and (max-width: 850px)
```

27.4.2 创建移动专用的视图

页面在不同的条件(包括小屏幕)下查看时, 适应式显示功能允许修改页面上应用了 CSS 样式的元素的比例和布局。但是, 在移动浏览器中显示页面时, 页面可能需要做更大的修改。为此, 应为非移动浏览器提供一个视图, 为移动浏览器提供另一个专用视图。在 ASP.NET MVC 4 推出之前, 必须使用浏览器嗅探功能和 ViewEngine 重写功能, 才能确定哪个视图用于什么浏览器。

ASP.NET MVC 4 提供了一项新功能, 允许使用约定-配置方法来重写用于移动设备的视图。该功能可用于所有的视图, 包括 Layouts。要提供视图的移动专用版本, 只需使用命名约定[viewname].mobile.cshtml 或[viewname].mobile.vbhtml, 创建一个文件即可。ASP.NET MVC 4 处理来自移动浏览器的请求时, 首先会查找遵循移动命名约定的文件。如果没有找到, 就使用标准视图。

要建立移动视图, 可以在 Views/Home 文件夹中创建一个新视图, 步骤是右击 Views/Home 文件夹, 选择 Add | View, 把新视图命名为 Index.mobile。现在可以在该视图中创建内容。页面由移动浏览器请求时, 就只显示这些内容。Index.mobile 视图显示在程序清单 27-10 中(本章下载代码中的 Views/Home/Index.mobile.cshtml 文件)。

程序清单 27-10 Index 移动视图

```
@{
    ViewBag.Title = "Index Mobile";
}

<h2>Index Mobile View</h2>
This page is being viewed in a mobile browser
```

运行应用程序, 在移动浏览器中查看时, 就会显示 Index 视图的移动版本中的内容; 而在桌面浏览器中查看应用程序, 会显示 Index 标准视图中的内容。图 27-11 显示了桌面浏览器中的 Index 视图, 图 27-12 显示了移动浏览器中的 Index 视图。两个浏览器都指向相同的 URL, 但显示的内容是不同的。

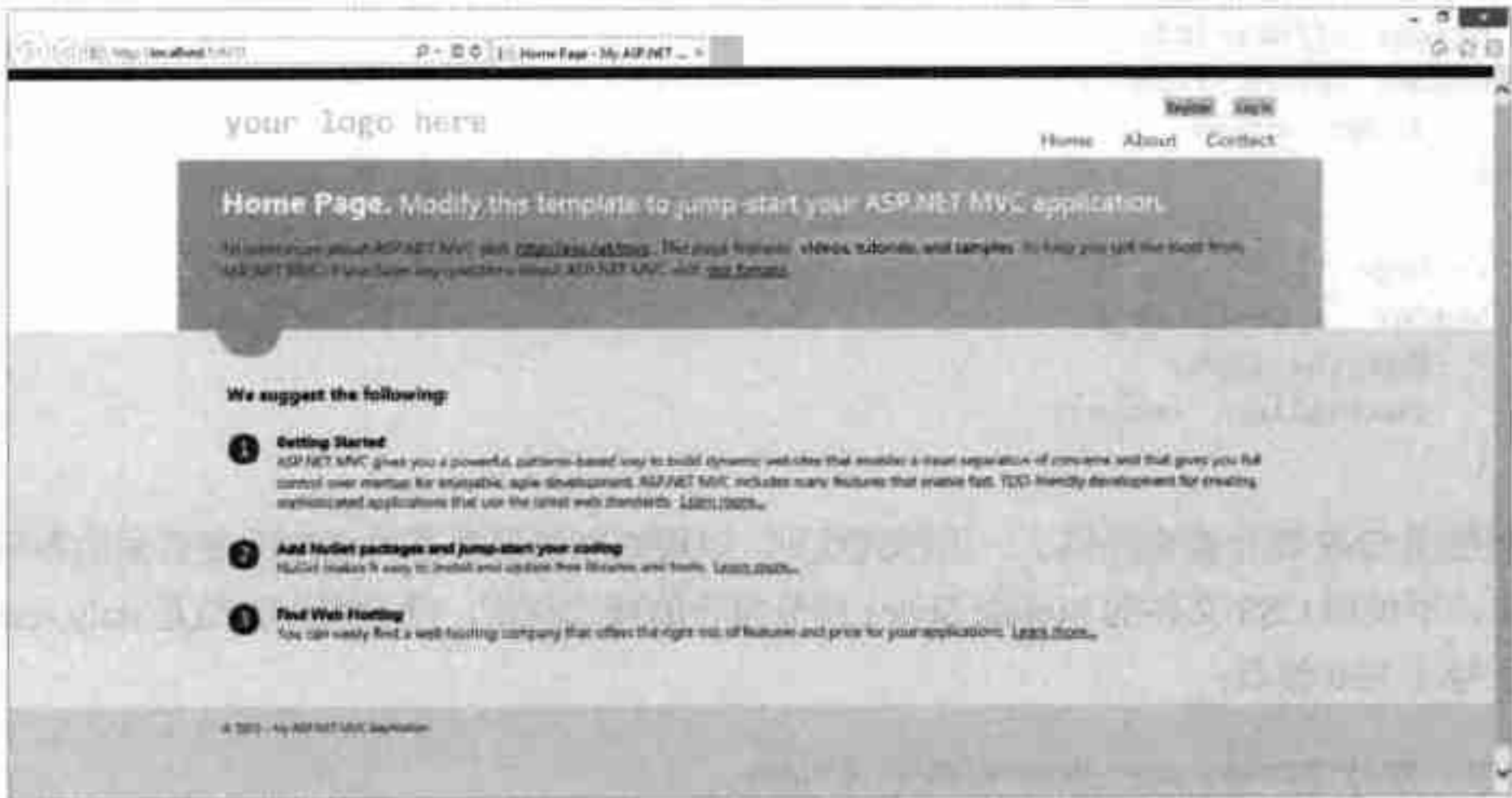


图 27-11



图 27-12

27.4.3 提供显示模式

使用移动专用视图，就可以提供专用的视图和布局，在移动浏览器发出请求时使用。但也可能需要另一层定制，例如为特定的浏览器或设备提供特定视图。例如，如果有人使用 iPhone 或 iPad 查看站点，就需要提供一个视图，其样式设置得更像是该设备的本地应用程序。

如果已经为 Web 建立了应用程序，就可能需要编写样式或脚本，使应用程序可以在特定的浏览器中工作。要知道，只利用一个页面面向多个特定的浏览器或平台是非常痛苦的。ASP.NET MVC 4 引入了一项新功能——显示模式，它很容易面向特定的浏览器或平台。与 ASP.NET MVC 4 中的许多新功能一样，它也使用约定-配置方法。

为了创建新的显示模式，需要给视图使用特定的命名约定，为特定的触发条件使用浏览器嗅探功能。为了演示这一点，要创建 iPhone 浏览站点时显示的特定视图。为此，在 Views\Home 文件夹中创建一个新视图，把它命名为 Index.iphone。接着在视图中创建内容，只有页面由 iPhone 请求时，才显示这些内容。Index.iphone 视图列在程序清单 27-11 中(本章下载代码的 Views\Home\Index.iphone.cshtml 文件)。

程序清单 27-11 用于 iPhone 的 Index 视图

```
@{
    ViewBag.Title = "Index iPhone";
}

<h2>Index iPhone View</h2>
This page is being viewed on an iPhone
```

现在，需要给 iPhone 后缀注册显示模式。显示模式只需注册一次，因此一般在 Global.asax 的 Application_Start 事件处理程序中注册。在 Global.asax 文件中，包含对 System.Web.WebPages 名称空间的引用：

```
using System.Web.WebPages;
```

现在需要使用 `DisplayModeProvider` 注册显示模式。对于 iPhone，需要指定显示模式的条件是：用户代理包含单词 iPhone。程序清单 27-12 是添加显示模式注册后的 `Global.asax` 文件。这些代码在本章下载代码的 `MvcMobileCS` 项目根目录下的 `Global.asax.cs` 文件中。

程序清单 27-12 `Global.asax` 文件显示了显示模式的注册

```
using System;
using System.Web.Http;
using System.Web.Mvc;
using System.Web.Optimization;
using System.Web.Routing;
using System.Web.WebPages;

namespace MvcMobile
{
    // Note: For instructions on enabling IIS6 or IIS7 classic mode,
    // visit http://go.microsoft.com/?LinkId=9394801

    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();

            WebApiConfig.Register(GlobalConfiguration.Configuration);
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
            AuthConfig.RegisterAuth();

            DisplayModeProvider.Instance.Modes.Insert(0, new
                DefaultDisplayMode("iphone")
            {
                ContextCondition = (context =>
                    context.GetOverriddenUserAgent().IndexOf
                        ("iPhone", StringComparison.OrdinalIgnoreCase) >= 0)
            });
        }
    }
}
```

新的显示模式被插入到 `Modes` 集合的顶部：

```
DisplayModeProvider.Instance.Modes.Insert
```

新的显示模式也有名称，这里叫做 `iphone`：

```
new DefaultDisplayMode("iphone")
```

显示模式的名称用作视图命名约定的后缀 `Index.iphone`。这里可以使用任意名称，只是要确保显示模式的名称匹配视图名称中使用的后缀。

最后，指定调用显示模式必须满足的条件。这里的条件是，在用户代理字符串中必须有单词 `iPhone`：

```
context.GetOverriddenUserAgent()  
    .IndexOf("iPhone", StringComparison.OrdinalIgnoreCase) >= 0)
```



应使用 `GetOverriddenUserAgent` 方法获得用户代理字符串, `GetOverriddenUserAgent` 方法返回重写的用户代理值, 如果没有指定重写, 就返回实际的用户代理字符串。重写用户代理字符串的技术常常用于为用户提供如下功能: 即使用户使用移动设备, 也能查看页面的桌面视图。

现在用 iPhone 查看页面, 就会看到 iPhone 专用视图指定的 iPhone 页面版本。图 27-13 显示了 iPhone 中的 Index 视图。



图 27-13

27.4.4 包含 jQuery Mobile 和 ViewSwitcher

一旦开始开发专用于移动浏览器的视图, 就可能希望在这些视图中包含移动专用的功能和客户库。常见的移动 JavaScript 库是 jQuery Mobile。jQuery Mobile 是开源库, 可基于 jQuery Core 为移动设备建立 UI。尽管 ASP.NET MVC 4 包含的许多功能都有助于开发移动 Web 应用程序, 但 ASP.NET MVC 项目模板默认不包含 jQuery Mobile (Mobile Application 项目模板例外)。

如果希望在项目中安装 jQuery Mobile, 就可以使用 NuGet 安装。有个 NuGet 包可以在 ASP.NET MVC 项目中安装 jQuery Mobile。NuGet 包管理器默认在 Visual Studio 2012 中安装。在 Visual Studio 菜单中单击 Tools | Library Package Manager | Package Manager Console, 打开 Package Manager Console, 在控制台提示符中, 运行下面的命令:

```
Install-Package jQuery.Mobile.MVC
```

NuGet 包 jQuery.Mobile.MVC 给项目添加了几个文件, 包括 jQuery Mobile 需要的 CSS 和

JavaScript 文件。此外还添加了移动专用的布局文件 `_Layout.Mobile.cshtml` 此移动专用的布局文件包含了一些引用,把添加的 CSS 和 JavaScript 文件作为一组来使用。要创建这些引用,需要在 `Global.asax` 文件中包含一行代码。NuGet 包还包含 `readme` 文件,其中提供了这行代码。必须把这行代码添加到 `Application_Start` 处理程序中。程序清单 27-13 列出了添加这行代码(粗体显示)后的 `Application_Start` 处理程序。这个程序清单的代码在本章下载代码的 `MvcMobileCS` 项目根目录下的 `Global.asax.cs` 文件中。

程序清单 27-13 `Application_Start` 处理程序带有必要的 jQuery Mobile 库

```
protected void Application_Start()
{
    AreaRegistration.RegisterAllAreas();

    WebApiConfig.Register(GlobalConfiguration.Configuration);
    FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
    RouteConfig.RegisterRoutes(RouteTable.Routes);
    BundleConfig.RegisterBundles(BundleTable.Bundles);
    AuthConfig.RegisterAuth();

    DisplayModeProvider.Instance.Modes.Insert(0, new
        DefaultDisplayMode("iphone")
    {
        ContextCondition = (context =>
            context.GetOverriddenUserAgent().IndexOf
                ("iPone", StringComparison.OrdinalIgnoreCase) >= 0)
    });

    BundleMobileConfig.RegisterBundles(BundleTable.Bundles);
}
```

除了支持 jQuery Mobile 的文件之外, NuGet 包还添加了控制器和 `ViewSwitcher` 的部分视图。`ViewSwitcher` 允许用户在移动浏览器视图和标准桌面浏览器视图之间切换。其他移动网站也提供了一种方式,在移动设备上查看站点的桌面版本。这就是 `ViewSwitcher` 的作用。`ViewSwitcher` 部分视图包含在移动布局文件中:

```
@Html.Partial("_ViewSwitcher")
```

`ViewSwitcher` 部分视图显示了一个链接,允许用户切换到应用程序的桌面视图。如果希望用户能切换回移动视图,就可以把这个部分视图添加到标准布局文件中。程序清单 27-14(代码文件 `Views\Shared_ViewSwitcher.cshtml`)显示了 `ViewSwitcher` 部分视图。

程序清单 27-14 `ViewSwitcher` 部分视图

```
@if (Request.Browser.IsMobileDevice && Request.HttpMethod == "GET")
{
    <div class="view-switcher ui-bar-a">
        @if (ViewContext.HttpContext.GetOverriddenBrowser().IsMobileDevice)
        {
            @: Displaying mobile view
            @Html.ActionLink("Desktop view", "SwitchView", "ViewSwitcher", new {
```

```

        mobile = false, returnUrl = Request.Url.PathAndQuery }, new { rel =
        "external" })
    }
    else
    {
        @: Displaying desktop view
        @Html.ActionLink("Mobile view", "SwitchView", "ViewSwitcher",
            new { mobile= true,
                returnUrl = Request.Url.PathAndQuery },
            new { rel = "external"
                })
    }
</div>
}

```

只有当请求来自移动浏览器时，ViewSwitcher 才是可见的：

```
@if (Request.Browser.IsMobileDevice && Request.HttpMethod == "GET")
```

如果请求来自移动浏览器，就调用 GetOverriddenBrowser 方法，确定浏览器类型是否重写了。根据响应，显示链接，使视图在移动视图和标准视图之间切换：

```
@if (ViewContext.HttpContext.GetOverriddenBrowser().IsMobileDevice)
```

ActionLink 引用了 ViewSwitcher 控制器的 SwitchView 动作，ViewSwitcher 控制器是由 NuGet 包 jQuery.Mobile.MVC 添加的。程序清单 27-15(本章下载代码中的 Controllers\ViewSwitcherController.cs 文件)显示了 ViewSwitcher 控制器。

程序清单 27-15 ViewSwitcher 控制器

```

using System.Web.Mvc;
using System.Web.WebPages;

namespace MvcMobile.Controllers
{
    public class ViewSwitcherController : Controller
    {
        public RedirectResult SwitchView(bool mobile, string returnUrl) {
            if (Request.Browser.IsMobileDevice == mobile)
                HttpContext.ClearOverriddenBrowser();
            else
                HttpContext.SetOverriddenBrowser(mobile ? BrowserOverride.Mobile :
                    BrowserOverride.Desktop);

            return Redirect(returnUrl);
        }
    }
}

```

图 27-14 是添加 NuGet 包 jQuery.Mobile.MVC 后显示在移动浏览器中的应用程序。



图 27-14

27.4.5 使用 Mobile Application 项目模板

前面介绍的所有 ASP.NET MVC 4 功能都以某种方式支持桌面和移动浏览器。如果希望建立纯移动 Web 应用程序，该怎么办？对提供桌面浏览体验不感兴趣，该怎么办？ASP.NET MVC 4 提供的 Mobile Application 项目模板是直接面向移动设备的，该项目模板在 New ASP.NET MVC 4 Project 对话框中，如图 27-10 所示。

用 Mobile Application 项目模板创建项目时，注意项目的总体结构与其他项目模板相同，视图、控制器和模型仍是相同的。但如果查看视图的内容，就会发现它们是专门为移动浏览器建立的。jQuery Mobile 是默认包含的，在所有视图都实现了。所有的默认页面元素都是专门为移动设备设计的，指定了 jQuery Mobile 使用的 data-特性。图 27-15 是使用 Mobile Application 项目模板建立的、在移动浏览器中显示的一款应用程序。



图 27-15

27.5 测试移动应用程序

缩小浏览器窗口的尺寸，在桌面上浏览移动网站，可以测试移动网站的一些方面。但要全面测试移动网站，就需要在移动设备上查看它。除非有要测试的每种设备，否则就需要利用其他方式测试移动站点。不使用实际设备进行测试的两种常用方式是，利用允许改变 HTTP 报头信息的浏览器和使用模拟器。

浏览器，例如 Mozilla Firefox 和 Google Chrome，有几个可用的扩展，能用于修改请求中的 HTTP 报头信息。在浏览器请求中修改此信息，可以让 Web 服务器相信，请求来自移动设备。

测试移动站点的另一种方式是给要测试的特定设备使用模拟器。安装 Windows Phone SDK, 就可以安装 Windows Phone 模拟器。编写本书时, Windows Phone 8 SDK 需要 Hyper-V 和 Windows 8 Pro 的 64 位版本。即使没有支持运行 Windows Phone 8 模拟器的硬件, 也仍可以安装 Windows Phone SDK 7.1。尽管不是该模拟器的最新版本, 但仍能在运行 Windows Phone 7 或 7.5 的 IE9 上测试站点。

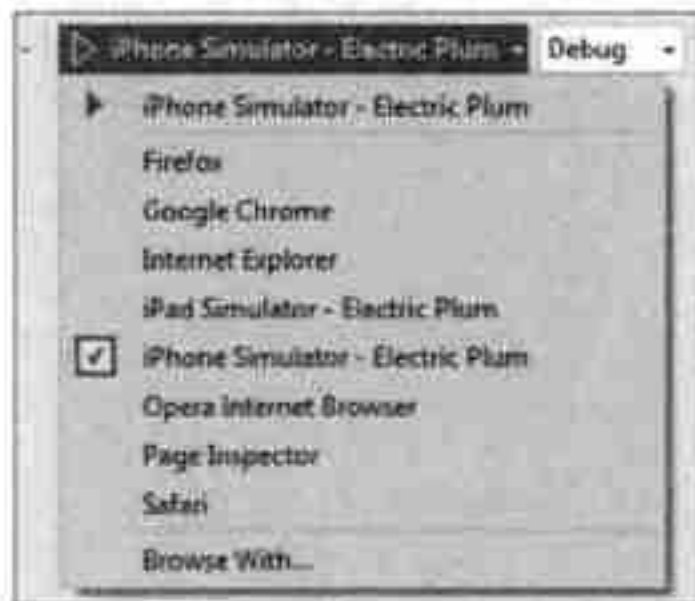


图 27-16

用于测试移动网站的另外两个常用设备是 Apple 的 iPhone 和 iPad 设备。Electric Plum 提供了 iPhone 和 iPad 模拟器, 作为 Electric Mobile Studio 的一部分。Electric Plum 与 Microsoft 合作, 提供 iPhone 和 iPad 模拟器作为 WebMatrix 2 的扩展, WebMatrix 是 Microsoft 开发的一款免费的轻型 Web 开发工具。这些模拟器也可用作 Visual Studio 2012 扩展。在 Visual Studio 2012 中安装 Electric Mobile Studio 扩展, iPhone 和 iPad 模拟器就可以在 Visual Studio 的 Browse With 选项中使用。图 27-16 是在安装了 Electric Mobile Studio 扩展后显示的 Browse With 选项。

27.6 本章小结

开发人员自从 20 世纪 90 年代后期就开始建立移动网站了。但随着移动设备的流行和广泛使用, 站点与移动设备兼容的要求也越来越高。

尽管开发人员建立移动网站的时间很长, 但挑战是相同的。建立移动网站需要不同的设计决策和考虑。可用的选项很多, 从单个网站的适应式设计, 到平行网站, 到直接面向移动设备的页面, 不一而足。

像 Visual Studio 2012 这样的工具, 以及一些可用于 ASP.NET MVC 最新版本的功能, 使移动站点的开发容易了许多。测试这些移动站点也因为使用浏览器扩展、模拟器而变得简单了。

我们不再有任何借口, 不建立可以在桌面和移动设备上查看的网站了。

第Ⅵ部分

应用程序的配置与开发

- 第 28 章 配置
- 第 29 章 调试和错误处理技术
- 第 30 章 模块和处理程序
- 第 31 章 异步通信
- 第 32 章 国际化应用程序的建立
- 第 33 章 打包和部署 ASP.NET 应用程序

第 28 章

配 置

本章要点

- 介绍 ASP.NET 配置文件
- 理解 ASP.NET 配置设置
- 加密配置文件的各个部分
- 研究 ASP.NET 配置 API
- 存储和检索敏感信息
- 在构建过程中更新配置设置
- 打包、压缩脚本和 CSS 文件

ASP.NET 使用一种基于 XML 的、灵活、可访问的、易于使用的配置系统。XML 配置文件允许管理员(在 Web 应用程序构建和部署后,负责管理这些应用程序的人)直接使用配置文件,或者使用 GUI 工具与配置文件交互操作,以方便地配置 ASP.NET 应用程序。附录 D 将详细介绍各种基于 GUI 的工具,本章则讨论如何直接使用 XML 配置文件改变 ASP.NET 应用程序的操作方式。

下面首先概述 ASP.NET 中的配置。

28.1 配置概述

ASP.NET 的配置以分层的方式存储在两个主要的、基于 XML 的文件中。XML 用于描述 ASP.NET 应用程序的各个方面的属性和操作。

ASP.NET 配置系统支持如下两种配置文件:

- `machine.config`: 服务器或计算机范围的配置文件。
- `web.config`: 应用程序配置文件。

配置文件是基于 XML 的,因此描述配置的元素都是区分大小写的。而且,ASP.NET 配置系统遵循驼峰式大小写命名约定。例如,在程序清单 28-1 的会话状态配置示例中可以看到,处理会话状态的 XML 元素表示为 `<sessionState>`。

程序清单 28-1 会话状态配置

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
<system.web>
  <sessionState
    mode="InProc"
    stateConnectionString="tcpip=127.0.0.1:42424"
    stateNetworkTimeout="10"
    sqlConnectionString="data source=127.0.0.1; user id=sa; password=P@55word"
    cookieless="false"
    timeout="20" />
  </system.web>
</configuration>
```

使用 XML 配置文件代替二进制元数据库的优点如下:

- 配置信息是可读的, 还可以使用纯文本编辑器(如 Notepad)修改, 但最好使用 Visual Studio 2010 或其他支持 XML 的编辑器修改。与二进制元数据库不同, 基于 XML 的配置文件很容易从一台服务器复制到另一台服务器, 这一点与复制简单文件一样。这个功能非常适合用于 Web 场(farm)。
- 在配置文件中修改一些设置后, ASP.NET 会自动检测到这些改动, 并把它们应用于正在运行的 ASP.NET 应用程序。ASP.NET 创建 ASP.NET 应用程序的新实例, 并把终端用户的所有未来请求发送到这个新的应用程序实例, 以此来完成这项工作。
- 在将对配置的修改应用于 ASP.NET 应用程序时, 不需要管理员停止并再次启动服务器。这些修改对终端用户是完全透明的。
- ASP.NET 配置系统是可扩展的。
- 与应用程序相关的信息很容易存储和检索。
- 可以加密存储在 ASP.NET 配置系统中的敏感信息, 防止他人盗取。

28.1.1 服务器配置文件

每台 ASP.NET 服务器的安装都包含一系列配置文件, 如 machine.config 文件, 该文件随 .NET Framework 一起默认安装。该文件和其他服务器特定的配置文件都位于 C:\Windows\Microsoft .NET \Framework\v4.0.30319\CONFIG 文件夹中, 这些文件表示在服务器上安装的所有 ASP.NET Web 应用程序使用的默认设置。

服务器范围内的配置文件有:

- legacy.web_hightrust.config
- legacy.web_hightrust.config.default
- legacy.web_lowtrust.config
- legacy.web_lowtrust.config.default
- legacy.web_mediumtrust.config
- legacy.web_mediumtrust.config.default
- legacy.web_minimaltrust.config
- legacy.web_minimaltrust.config.default

- machine.config
- machine.config.comments
- machine.config.default
- web.config
- web.config.comments
- web.config.default
- web_hightrust.config
- web_hightrust.config.default
- web_lowtrust.config
- web_lowtrust.config.default
- web_mediumtrust.config
- web_mediumtrust.config.default
- web_minimaltrust.config
- web_minimaltrust.config.default

系统范围内的配置文件 `machine.config` 用于给计算机上的所有应用程序配置公共的 .NET Framework 设置。一般来说,最好不要编辑或操作 `machine.config` 文件,除非知道自己正在做什么。修改这个文件会影响计算机上的所有应用程序(Windows 应用程序、Web 应用程序等)。

除了 `machine.config` 文件之外, .NET Framework 安装程序还安装了另外两个文件: `machine.config.default` 和 `machine.config.comments`。`machine.config.default` 文件用作 `machine.config` 文件的备份。如果要改回 `machine.config` 文件的出厂设置,只需把 `machine.config.default` 文件中的设置复制到 `machine.config` 文件中即可。

`machine.config.comments` 文件包含对每个配置部分的描述和对最常用的值的明确设置。`machine.config.default` 和 `machine.config.comments` 文件都不由 .NET Framework 运行时使用,安装它们是为了能够改回出厂时的默认设置和默认值。



.NET Framework 支持并行执行模式,因此,如果在服务器上安装了 .NET Framework 的多个版本,就会有多个 `machine.config` 文件。例如,如果在服务器上运行 .NET Framework 1.0、1.1、2.0 和 4,那么每个 .NET Framework 版本都会有自己的 `machine.config` 文件。也就是说,服务器上有 4 个 `machine.config` 文件。注意, .NET Framework 3 和 3.5 建立在 .NET Framework 2.0 基础之上(包含额外的 DLL,这些 DLL 有时称为扩展程序)。因此, .NET Framework 3 和 3.5 使用与 .NET Framework 2.0 相同的 `machine.config` 文件。要注意的是: .NET Framework 4 是全新的 CLR,因而不像 .NET Framework 3 和 3.5 那样依赖 .NET Framework 2.0。 .NET Framework 4 有自己的 `machine.config` 文件。 .NET Framework 4.5 也不同,它是 .NET Framework 4 的改进版。 .NET Framework 4 最初的库由 .NET Framework 4.5 替代。

`machine.config` 文件所在的 CONFIG 文件夹中还有一个根级别的 `web.config` 文件。在修改服务器上的设置时,应总是在这个根级别的 `web.config` 文件中进行修改,而不是修改 `machine.config` 文件。

machine.config 文件包括 machine.config.default 和 machine.config.comments 文件，而 web.config 文件也包括相应的 web.config.default 和 web.config.comments 文件。

ASP.NET Web 应用程序默认运行在完全信任设置下。在根级别的 web.config 文件中查看 <securityPolicy> 和 <trust> 部分，即可看到这个设置，如程序清单 28-2 所示。

程序清单 28-2 根级 web.config 文件的信任级别

```
<configuration>
  <location allowOverride="true">
    <system.web>
      <securityPolicy>
        <trustLevel name="Full" policyFile="internal" />
        <trustLevel name="High" policyFile="web_hightrust.config" />
        <trustLevel name="Medium" policyFile="web_mediumtrust.config" />
        <trustLevel name="Low" policyFile="web_lowtrust.config" />
        <trustLevel name="Minimal" policyFile="web_minimaltrust.config" />
      </securityPolicy>
      <trust level="Full" originUrl="" />
    </system.web>
  </location>
</configuration>
```

在特定的信任级别定义其他策略文件，这些级别确定了 ASP.NET 的代码访问安全性 (Code-Access Security, CAS)。要改变 ASP.NET 应用程序在服务器上运行的信任级别，只需在 web.config 文件的文档或应用程序实例中修改 <trust> 元素即可。例如，使用程序清单 28-3 中的代码，就可以改为中等信任级别。

程序清单 28-3 改为中等信任级别

```
<configuration>
  <location allowOverride="false">
    <system.web>
      <securityPolicy>
        <trustLevel name="Full" policyFile="internal" />
        <trustLevel name="High" policyFile="web_hightrust.config" />
        <trustLevel name="Medium" policyFile="web_mediumtrust.config" />
        <trustLevel name="Low" policyFile="web_lowtrust.config" />
        <trustLevel name="Minimal" policyFile="web_minimaltrust.config" />
      </securityPolicy>
      <trust level="Medium" originUrl="" />
    </system.web>
  </location>
</configuration>
```

在本例中，代码不仅要求使用 web_mediumtrust.config 文件，而且将 allowOverride 特性设置为 false，强制服务器上的所有 ASP.NET 应用程序都使用该信任级别。各个应用程序实例不能在它们本地的 web.config 文件中重写该设置，因为该设置是在根级别的 web.config 文件中指定的。

查看各个信任级别配置文件，如 web_mediumtrust.config 文件，注意它们通过代码操作定义了我们能执行的操作。例如，web_hightrust.config 文件允许在服务器的任意地方进行 FileIO 访问，如程

序清单 28-4 所示。

程序清单 28-4 web_hightrust.config 文件对 FileIO CAS 的定义

```
<IPermission
  class="FileIOPermission"
  version="1"
  Unrestricted="true"
/>
```

但是,在中等信任级别的 web.config 文件(web_mediumtrust.config 文件)中,只允许 ASP.NET 在应用程序目录中进行这些 FileIO 操作,其定义如程序清单 28-5 所示。

程序清单 28-5 web_mediumtrust.config 文件中的 FileIO 限制

```
<IPermission
  class="FileIOPermission"
  version="1"
  Read="\$AppDir\$"
  Write="\$AppDir\$"
  Append="\$AppDir\$"
  PathDiscovery="\$AppDir\$"
/>
```

最好查看 ASP.NET 应用程序可以运行在什么信任级别下,并修改<trust>部分,以支持 CAS 的相应级别。

28.1.2 应用程序配置文件

与 machine.config 文件不同,每个 ASP.NET 应用程序都把自己的配置设置存储在 web.config 文件中。如果将 Web 应用程序放在多个子文件夹下,每个子文件夹就有自己的 web.config 文件,它们继承或重写了父文件夹的文件设置。

要使用这些新设置更新场中的服务器,只需把这个 web.config 文件复制到相应的应用程序目录下。ASP.NET 会完成其他工作,不需要重新启动服务器,也不需要在本机访问服务器,应用程序会继续正常工作,但现在使用的是配置文件中应用的新设置。

28.1.3 应用配置设置

ASP.NET 运行时为给定的 Web 请求应用配置设置时, machine.config 文件(以及 web.config 文件中的配置信息)会合并为单个单元,然后这些信息会应用于给定的应用程序。配置设置继承于父 web.config 文件或 machine.config 文件, machine.config 文件是根配置文件或“祖先”。图 28-1 是一个应用 ASP.NET Web 应用程序的配置设置的示例。

每个 Web 应用程序的配置都是唯一的,但设置可以继承于父应用程序。例如,Web 站点根目录下的 web.config 文件把会话超时定义为 10 分钟,该设置会重写从 machine.config 文件或根 web.config 文件继承的默认 ASP.NET 设置。子目录或子文件夹中的 web.config 文件可以重写或继承这些设置(例如 10 分钟的会话超时设置)。

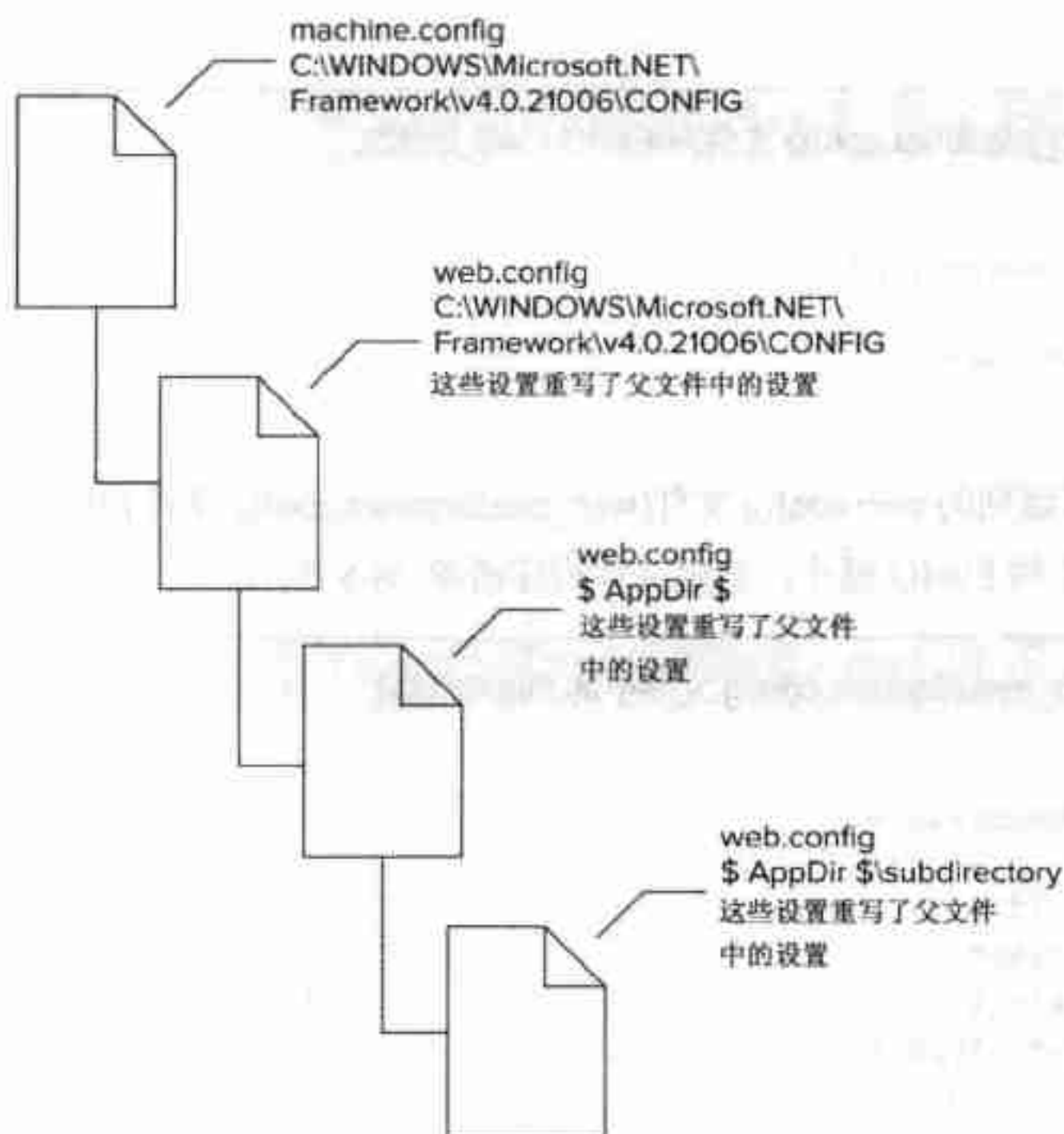


图 28-1



虚拟目录的配置设置独立于物理目录结构，除非虚拟目录的组织方式是专门指定的，否则就会导致配置问题。

注意，在大多数情况下，使用程序清单 28-3 中的 `allowOverride="false"` 机制，就可以禁止这些继承或重写规则。

28.1.4 检测配置文件发生的改动

ASP.NET 会自动检测配置文件 `machine.config` 或 `web.config` 发生的改动，通过监听操作系统提供的文件改动通知事件实现该操作。

在启动 ASP.NET 应用程序时，会读取配置设置，并将其存储到 ASP.NET 高速缓存中。然后把文件依赖放在 `machine.config` 和 `web.config` 配置文件的高速缓存条目中。在 `machine.config` 文件中检测到配置文件更新时，ASP.NET 就会创建新的应用程序域，以服务新的请求。旧的应用程序域在处理完已有的请求后就会被删除。

28.1.5 配置文件的格式

`machine.config` 文件和 `web.config` 文件之间的主要区别是文件名。除此之外，它们的模式是相同的。配置文件分为几组。配置文件中的根级别 XML 元素命名为 `<configuration>`。一般的伪 `web.config`

文件都有控制 ASP.NET 的部分，如程序清单 28-6 所示。

程序清单 28-6 伪 web.config 文件

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <configSections>
    <section name="[sectionSettings]" type="[Class]"/>
    <sectionGroup name="[sectionGroup]">
      <section name="[sectionSettings]" type="[Class]"/>
    </sectionGroup>
  </configSections>
</configuration>
```



方括号中的值在实际的配置文件中是唯一的。

XML 配置文件中的根元素总是<configuration>。每个部分处理程序和设置都可以封装在<sectionGroup>部分中。<sectionGroup>部分在配置文件中提供组织功能，允许把配置组织为唯一的组。例如，<system.web>部分组用于标识配置文件中专门用于 ASP.NET 的区域。

<configSections>部分是分组与每个配置部分相关的配置部分处理程序的机制。要创建自己的部分处理程序，就必须在<configSections>部分中声明它们。<httpModules>部分有设置为 System.Web.Caching.HttpModulesSection 的配置处理程序，而<sessionState>部分有设置为 System.Web.SessionState.SessionStateSection 类的配置处理程序，如程序清单 28-7 所示。

程序清单 28-7 machine.config 文件中的 HTTP 模块配置设置

```
<configSections>
  <sectionGroup>
    <section name="httpModules"
      type="System.Web.Configuration.HttpModulesSection,
      System.Web, Version=4.0.0.0, Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a"/>
    </sectionGroup>
  </configSections>
```

28.2 公共配置设置

ASP.NET 应用程序依赖于几个公共配置设置。这些设置在 web.config 和 machine.config 文件中都有。本节就介绍这些公共配置设置。

28.2.1 连接字符串

在 ASP.NET 的早期版本中，所有的连接字符串信息都存储在<appSettings>部分中。ASP.NET 的最新版本包含<connectionStrings>部分，可以存储所有的连接字符串信息。尽管将连接字符串存储在

<appSettings>元素中仍然可以正常工作，但存在如下问题：

- 连接字符串存储在<appSettings>部分中时，支持数据的控件，如 SqlCacheDependency 或 MembershipProvider，就找不到这些信息。
- 使用加密算法保护连接字符串是个难题。
- 最后，该功能不仅应用于 ASP.NET，还应用于所有的.NET 应用程序类型，包括 Windows 窗体应用、Web 服务等。

由于连接字符串信息独立于<appSettings>部分进行存储，因此可以使用强类型化的集合方法 ConnectionStrings 检索它们。程序清单 28-8 给出了存储连接字符串的例子。

程序清单 28-8 存储连接字符串

```
<configuration>
  <connectionStrings>
    <add
      name="ExampleConnection"
      connectionString="server=401kServer;database=401kDB;
      uid=WebUser;pwd=P@$$word9" />
    </connectionStrings>
  </configuration>
```

程序清单 28-9 演示了如何在代码中检索连接字符串 ExampleConnection。

程序清单 28-9 检索连接字符串

```
protected void Page_Load(Object sender, EventArgs e)
{
    . . .
    SqlConnection dbConnection = new
        SqlConnection(ConfigurationManager.ConnectionStrings["ExampleApplication"]
            .ConnectionString);
    . . .
}
```

这类构造的功能很强大。不是将连接字符串硬编码到 ASP.NET 应用程序的每个页面中，而是可以集中存储连接字符串的实例(例如，在 web.config 文件中存储)。如果要修改这个连接字符串，只需要在一个地方修改，而不必在多处修改。

28.2.2 配置会话状态

因为基于 Web 的应用程序遵循无状态的 HTTP 协议，所以必须把与应用程序有关的或与用户有关的状态永久存储。Session 对象是永久存储与用户有关的信息的公共存储。会话存储实现为散列表，按照键/值对组合的方式存储数据。

ASP.NET 可以在 InProc、StateServer、SqlServer 和 Custom 中存储会话状态数据。Custom 设置允许开发人员更多地控制会话状态在永久存储中的存储方式。例如，ASP.NET 不允许把会话数据存储在非微软数据库(如 Oracle)或 NoSQL 数据库(如 MongoDB)中。如果要在这些数据库或定制存储(如 XML 文件)中存储会话数据，就需要编写定制的提供程序类(请参见稍后的“定制状态存储”部分和第 21 章，以了解 ASP.NET 4.5 中的会话状态功能)。

可以使用<sessionState>元素配置会话信息，如程序清单 28-10 所示。

程序清单 28-10 配置会话信息

```
<sessionState
  mode="StateServer"
  cookieless="false"
  timeout="20"
  stateConnectionString="tcpip=ExampleSessionStore:42424"
  stateNetworkTimeout="60"
  sqlConnectionString=""
/>
```

下面的列表描述了上述代码中<sessionState>元素的一些特性：

- **mode**: 指定是否应保存会话信息。mode 设置有 5 个选项：Off、InProc、StateServer、SqlServer 和 Custom，默认选项是 InProc。
- **cookieless**: 指定是否支持 HTTP 无 cookie 会话键管理。
- **timeout**: 指定会话的生存时间。timeout 值是可以变化的，对于每个请求，超时周期都会重新设置为当前时间加上超时值。例如，如果 timeout 值是 20 分钟，在 10:10 AM 收到请求，就会在 10:30 AM 出现超时。
- **stateConnectionString**: mode 特性设置为 StateServer 时，这个设置指定 TCP/IP 地址和端口，用于与提供状态管理的 Windows 服务通信。
- **stateNetworkTimeout**: 指定试图在进程外的会话存储(如 StateServer)中存储状态的超时值(秒)。
- **sqlConnectionString**: mode 特性设置为 SqlServer 时，这个设置用于连接 SQL Server 数据库，以存储和检索会话数据。

支持 Web 场

多台 Web 服务器组合为一组来工作，这个组称为 Web 场。可以把 ASP.NET 应用程序部署到 Web 场内的多台服务器上，ASP.NET 就支持这种部署。但是，会话数据需要保存在进程外的会话存储中，如 StateServer 或 SqlServer。

状态服务器

StateServer 和 SqlServer 都支持进程外会话状态。但是，StateServer 在 Windows 服务中存储所有的会话信息，其中 Windows 服务在内存中存储会话数据。使用这个选项，如果 Web 场内驻留会话状态服务的服务器失败，那么访问 Web 站点的所有 ASP.NET 客户端都会失败，并且无法恢复会话数据。

如果使用 Windows 7，就选择 Start | Control Panel | System and Security | Administrative Tools | Services 命令，打开 Services 对话框，从中可以配置会话状态服务；如果使用 Windows 8，就在 Search 功能区的 Settings 选项卡中选择 View Local Services，如图 28-2 所示。



图 28-2

另外，还可以使用命令行提示符，输入 `net start` 命令来启动会话状态服务，如下所示：

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\> net start aspnet_state
```



ASP.NET 的所有兼容版本都共享单个状态服务实例，也就是使用 ASP.NET 的最高版本安装的服务。例如，如果在已有 ASP.NET 2.0 和 1.1 的服务器上又安装了 ASP.NET 4.5，ASP.NET 4.5 安装就会替代 ASP.NET 2.0 的状态服务器实例。ASP.NET 4.5 服务可以用于 ASP.NET 以前所有的兼容版本。另外，还必须用提升的权限运行命令提示符，以使用 `net start` 命令。

SQL Server

选择 SQLServer 选项时，会话数据就会存储在 SQL Server 数据库中。即使 SQL Server 失败，内置的 SQL Server 恢复功能也仍然可以恢复所有的会话数据。把 ASP.NET 配置为支持 SQL Server 的会话状态服务，只需配置 Windows 服务即可。唯一的区别是，要运行 ASP.NET 附带的 T-SQL 脚本 `InstallSqlState.sql`。卸载 ASP.NET SQL Server 支持的 T-SQL 脚本 `UninstallSqlState.sql` 也包含在内。安装和卸载脚本都位于 Framework 文件夹中。使用 SQLServer 选项的示例如程序清单 28-11 所示。

程序清单 28-11 给会话状态服务使用 SQLServer 选项

```
<configuration>
  <system.web>
    <sessionState
      mode="SQLServer"
      sqlConnectionString="data source=ExampleSessionServer;
      user id=ExampleWebUser;password=P@55word"
      cookieless="false"
      timeout="20"
    />
```



```
</system.web>
</configuration>
```

ASP.NET 通过存储过程访问存储在 SQL Server 中的会话数据。在默认情况下,所有的会话数据都存储在 Temp DB 数据库中。但是,可以修改存储过程,把这些数据存储在功能全面的数据库表中,而不是存储在 Temp DB 中。



即使基于 SQL Server 的会话状态服务给会话状态提供了可伸缩的用法,也有可能失败。这是因为 SQL Server 会话状态为同一 ASP.NET 进程中的所有应用程序使用相同的 SQL Server 数据库。这个问题在 ASP.NET 2.0 中就已经得到了解决,我们可以为每个应用程序配置不同的数据库。现在可以使用 aspnet_regsql.exe 实用程序为所有的应用程序配置基于 SQL Server 的会话状态。但是,如果需要旧式的 .NET Framework 解决方案,那么可以在 <http://support.microsoft.com/kb/836680> 上查找。

连接字符串以强类型化模式存储,因此可以在配置文件的其他部分引用连接字符串信息。例如,当配置将要存储在 SQL Server 中的会话状态时,可以在 <connectionStrings> 部分指定连接字符串,然后在 <sessionState> 元素中指定连接字符串的名称,如程序清单 28-12 所示。

程序清单 28-12 配置带连接字符串的会话状态

```
<configuration>
  <connectionStrings>
    <add name = "ExampleSqlSessionState"
      connectionString = "data source=ExampleSessionServer;
      user id=ExampleWebUser;password=P@55word" />
  </connectionStrings>
  <system.web>
    <sessionState
      mode="SQLServer"
      sqlConnectionString="ExampleSqlSessionState"
      cookieless="false"
      timeout="20"
    />
  </system.web>
</configuration>
```

定制状态存储

ASP.NET 4.5 中的会话状态基于一种可插入的体系结构,该体系结构包含继承于 SessionStateStoreProviderBase 类的不同提供程序。如果要创建自己的定制提供程序或使用第三方的提供程序,就必须把 mode 特性设置为 Custom。

可以指定继承于 SessionStateStoreProviderBase 类的定制提供程序程序集,如程序清单 28-13 所示。

程序清单 28-13 使用自己的会话状态提供程序

```
<configuration>
  <system.web>
    <sessionState mode="Custom" customProvider="CustomStateProvider">
```



```

    <providers>
      <add name="CustomStateProvider"
        type="CustomStateProviderAssembly,
        CustomStateProviderNamespace.CustomStateProvider"/>
    </providers>
  </sessionState>
</system.web>
</configuration>

```

在上面的例子中，把会话状态模式配置为 Custom，因为已经把提供程序名称指定为 CustomStateProvider。接着，添加<providers>元素，包含提供程序的类型、名称空间和类名。



有关提供程序模型和定制提供程序的内容，详见第 14 和 15 章。

28.2.3 编译配置

ASP.NET 支持 ASP.NET 页面、Web 服务、HttpHandler、ASP.NET 应用程序文件(如 Global.asax 文件)、源文件等文件的动态编译。在 ASP.NET 应用程序第一次请求这些文件时，会按要求自动编译它们。

对动态编译过的文件进行修改，会使所有相关的资源自动无效并重新进行编译。这个系统允许开发人员快速开发应用程序，并且进程开销最小，因为开发人员单击 Save 按钮后，对代码的修改会立即在应用程序中发挥作用。

可以使用 web.config 或 machine.config 文件中的<compilation>部分来配置 ASP.NET 编译设置。ASP.NET 引擎在需要时编译 ASP.NET 页面，并在代码高速缓存中保存生成的代码。在执行 ASP.NET 页面时使用这些高速缓存的代码。程序清单 28-14 演示了<compilation>部分的语法。

程序清单 28-14 <compilation>部分

```

<!-- compilation Attributes -->
<compilation
  tempDirectory="" [String]
  debug="false" [true|false]
  strict="false" [true|false]
  explicit="true" [true|false]
  batch="true" [true|false]
  optimizeCompilations="false" [true|false]
  urlLinePragmas="false" [true|false]
  batchTimeout="900" [in Seconds][number]
  maxBatchSize="1000" [number]
  maxBatchGeneratedFileSize="1000" [number]
  numRecompilesBeforeAppRestart="15" [number]
  defaultLanguage="vb" [String]
  targetFramework="" [String]
  assemblyPostProcessorType="" [String]
>
  <assemblies>
    <add assembly="" [String, Required, Collection Key] />
  </assemblies>

```

```

<buildproviders>
  <add extension="" [String, Required, Collection Key]
    type="" [String, Required] />
</buildproviders>
<folderLevelBuildProviders>
  <add name="" [String, Required, Collection Key]
    type="" [String, Required] />
</folderLevelBuildProviders>
<expressionBuilders>
  <add expressionPrefix="" [String, Required, Collection Key]
    type="" [String, Required] />
</expressionBuilders>
<codeSubDirectories>
  <add directoryName="" [String, Required, Collection Key] />
</codeSubDirectories>
</compilation>

```

下面详细描述这些<compilation>特性:

- **batch**: 指定是否支持批量编译, 默认值为 **true**。
- **maxBatchSize**: 指定可以成批编译的最大页面/类数, 默认值为 1000。
- **maxBatchGeneratedFileSize**: 指定程序集批量编译的最大输出尺寸, 默认值为 1000KB。
- **batchTimeout**: 指定用于批量编译的时间(分钟)。如果已经超过这个时间, 但编译未完成, 就抛出异常。默认值为 15 分钟。
- **optimizeCompilations**: 指定是动态编译整个站点, 还是仅编译修改的项。当设置为 **False**(默认值)时, 会在顶层文件发生变化时重新编译整个站点; 当设置为 **True** 时, 仅重新编译修改的文件。
- **debug**: 指定是编译程序集还是调试程序集, 默认值为 **false**。
- **defaultLanguage**: 指定在动态编译文件中使用的默认编程语言, 如 C#。使用<compiler>子元素定义语言名称。
- **explicit**: 指定是否明确显示 Visual Basic 代码编译选项, 默认值为 **true**。
- **numRecompilesBeforeAppRestart**: 指定在应用程序重新启动前动态重新编译的资源量。
- **strict**: 指定 Visual Basic 严格编译选项的设置。
- **urlLinePragmas**: 指示编译器是否应使用 URL, 而不是实际路径(默认行为)。
- **tempDirectory**: 指定在编译过程中用于存储临时文件的目录。默认情况下, ASP.NET 会在 [WinNT\Windows]\Microsoft .NET\Framework\[version]\Temporary ASP.NET Files 文件夹中创建临时文件。
- **assemblies**: 指定 ASP.NET 编译过程中使用的程序集。
- **codeSubDirectories**: 指定子目录的有序集合, 这些子目录包含了在运行时编译的文件; 添加 codeSubDirectories 部分可创建多个程序集。
- **buildproviders**: 指定已建立的提供程序集合, 用于编译定制的资源文件。
- **folderLevelBuildProviders**: 指定已建立的提供程序集合, 用于在指定文件夹中编译定制的资源文件。
- **expressionBuilders**: 指定要在编译过程中使用的资源字符串的集合。

28.2.4 定制错误

ASP.NET 应用程序失败时，ASP.NET 页面可以显示默认的错误页面，其中包含源代码和错误所在的行号。但是，这种方法有几个问题：

- 源代码和错误消息对缺乏经验的终端用户来说没有什么意义。
- 如果把源代码和错误消息显示给黑客，就可能遭到进一步的破坏。

显示太多的错误信息会提供重要的实现细节，但我们常常不希望公布这些实现细节。图 28-3 显示了一个例子。

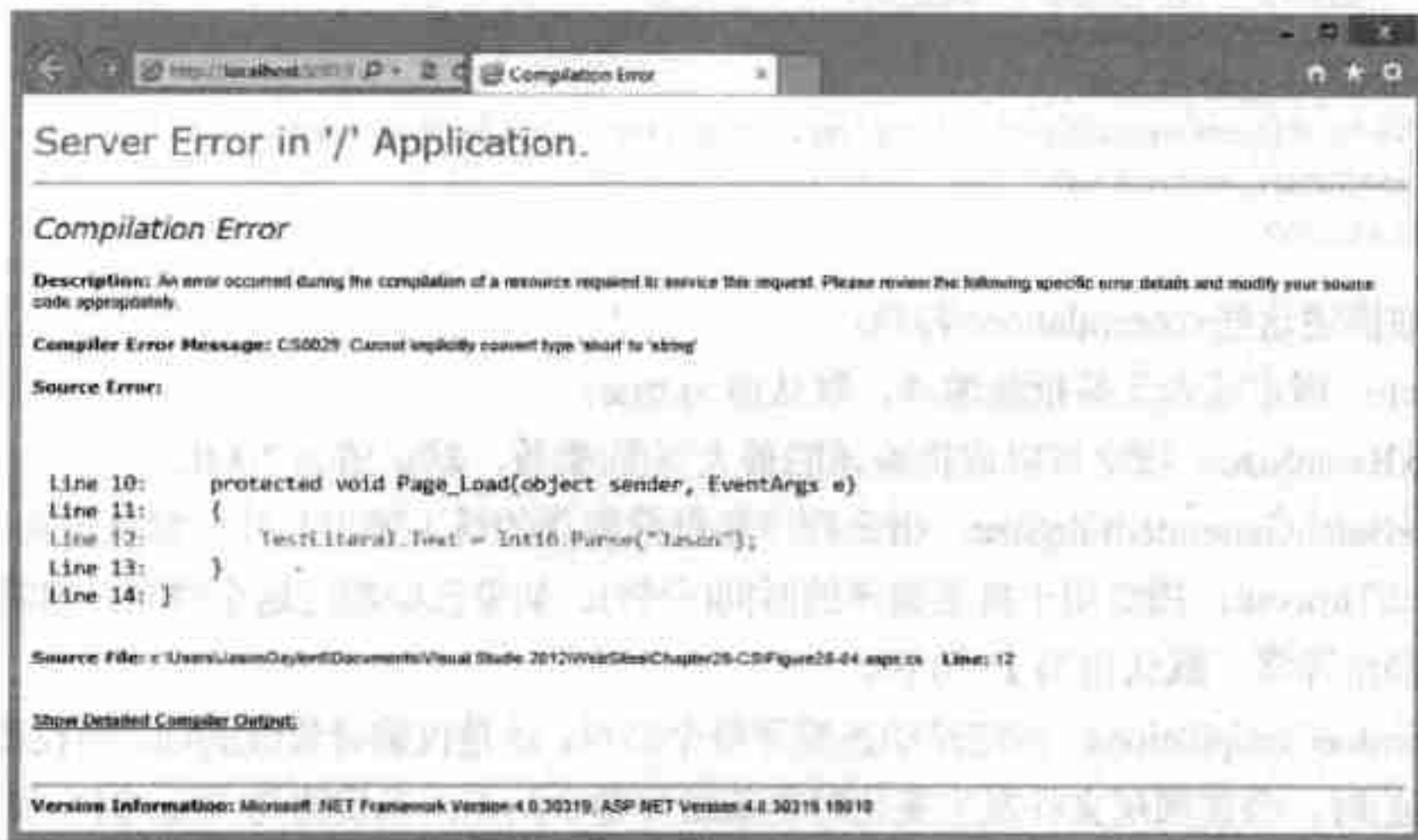


图 28-3

但是，ASP.NET 提供了非常好的基础结构，可以避免显示这种错误信息。<customErrors>部分可以在 ASP.NET 应用程序中定义定制的错误消息，语法如下：

```
<customErrors defaultRedirect="[url]" mode="[on/off/remote]">
  <error statusCode="[statusCode]" redirect="[url]" />
</customErrors>
```

- defaultRedirect: 指定在发生错误时将客户端浏览器重定向到的默认 URL，这是可选设置。
- mode: 指定定制错误的状态是启用、禁用或只显示给远程计算机。值可以是 On、Off 和 RemoteOnly。On 表示启用定制错误，Off 表示禁用定制错误，RemoteOnly 表示定制错误只显示给远程客户端。
- customErrors: 支持用于定义定制错误的多个<error>子元素。每个<error>子元素都可以包含 statusCode 特性和 URL。

28.2.5 身份验证

第 20 章详细介绍了身份验证过程。本节将复习该过程中与配置相关的内容。身份验证过程将验证用户的身份，并在服务器和请求之间建立身份。HTTP 是无状态的协议，因此身份验证信息要存储在客户端或服务器的某个地方，ASP.NET 支持这两种存储方式。

可以把服务器端的信息存储在 Session 对象中，而在客户端中则有许多选项：

- cookie
- ViewState
- URL
- 隐藏字段

ASP.NET 支持如下身份验证方法:

- Windows 身份验证
- 表单身份验证



在 ASP.NET 4 之前, 还有第三个提供程序: passport 验证提供程序。但这个提供程序目前标记为废弃, 不再是有效的模式类型。

如果要禁用身份验证, 可以使用 `mode="None"` 设置:

```
<authentication mode="None" />
```

1. Windows 身份验证

ASP.NET 依赖于 IIS 的基础结构来实现 Windows 身份验证, Windows 身份验证可以使用 Windows Challenge/Response 语义验证请求。当 Web 服务器接收到请求时, 会首先拒绝对请求的访问(这是难点)。这会使浏览器弹出窗口以收集凭据。请求使用 Windows 凭据的散列值来响应, 然后服务器就可以进行验证。

要实现 Windows 身份验证, 应使用 IIS 配置相应的 Web 站点或虚拟目录。接着使用 `<authentication>` 元素为 Web 应用程序或虚拟目录标记 Windows 身份验证, 程序清单 28-16 演示了这个过程。

程序清单 28-15 把身份验证方法设置为 Windows 身份验证

```
<configuration>
  <system.web>
    <authentication mode="Windows" />
  </system.web>
</configuration>
```



只能为计算机、站点或应用程序声明 `<authentication>` 元素。如果在子目录的配置文件或页面上声明该元素, 就会产生分析错误消息。

2. 表单身份验证

表单身份验证是一种广泛应用的身份验证机制。可以使用 `<authentication>` 部分和 `<forms>` 子部分来配置表单身份验证。在配置文件中处理表单身份验证的 `<authentication>` 部分具有如程序清单 28-16

所示的结构。

程序清单 28-16 处理表单身份验证的<authentication>部分

```
<configuration>
  <system.web>
    <authentication mode="Forms">
      <forms
        name=".ASPXAUTH" [String]
        loginUrl="login.aspx" [String]
        defaultUrl="default.aspx" [String]
        protection="All" [All|None|Encryption|Validation]
        timeout="30" [in Minutes][number]
        path="/" [String]
        requireSSL="false" [true|false]
        slidingExpiration="true" [true|false]
        enableCrossAppRedirects="false" [true|false]
        cookieless="UseDeviceProfile"
          [UseUri|UseCookies|AutoDetect|UseDeviceProfile]
        domain="" [String]
        ticketCompatibilityMode="Framework20" [Framework20|Framework40]>
      <credentials passwordFormat="SHA1" [Clear|SHA1|MD5]>
        <user name="" [String, Required, CollectionKey]
          password="" [String, Required] />
      </credentials>
    </forms>
  </authentication>
</system.web>
</configuration>
```

各特性如下所述：

- name: 指定 HTTP 身份验证票证的名称，默认值为 ASPXAUTH。
- loginUrl: 指定一个 URL，如果当前请求没有有效的验证票证，就把该请求重定向到这个 URL 上。
- protection: 指定用于保护 cookie 数据的方法，有效值是 All、None、Encryption 和 Validation。
 - Encryption: 指定在配置文件中使用 TripleDES 或 DES 加密算法加密的 cookie 的内容。但是，不对 cookie 进行数据有效性验证。
 - Validation: 指定 cookie 的内容不加密，但验证 cookie 数据在传输过程中是否未修改。
 - All: 指定使用数据有效性验证和加密两种方式保护 cookie 的内容。根据 <machineKey> 元素使用配置的数据有效性验证算法，并使用 TripleDES 加密算法进行加密。这是默认值，表示最高保护级别。
 - None: 指定不对 cookie 应用保护机制。不存储敏感信息和不使用 cookie 存储个性化信息的 Web 应用程序可以使用这个选项。指定 None 值时，将禁用加密和数据有效性验证。
- timeout: 指定 cookie 的有效期(分钟)。timeout 特性是可以改变的，表示从接收到上一个请求开始后到过期的时间。默认值为 30 分钟。
- path: 指定用于 cookie 的路径，默认值为 /，以避免路径的大小写不匹配，因为浏览器在返回 cookie 时是严格区分大小写的。

- **requireSSL**: 指定在安全的 HTTPS 连接中是否启用表单身份验证。
- **slidingExpiration**: 指定有效的 cookie 在使用时是否应定期更新。该特性设置为 False 时, 票证只在发布时期内有效, 而且用户即使在活动的会话中也必须重新验证身份。
- **cookieless**: 指定是否支持无 cookie 身份验证。值可以是 UseCookies、UseUri、Auto 和 UseDeviceProfile, 默认值为 UseDeviceProfile。
- **defaultUrl**: 指定验证身份后控制重定向的登录控件使用的默认 URL。
- **enableCrossAppRedirects**: 当该特性设置为 True 时, 允许重定向到不在当前应用程序中的 URL。
- **domain**: 指定在身份验证 cookie 中附加的域名字符串。在域的多个站点中共享同一身份验证 cookie 时, 这个特性非常有用。
- **ticketCompatibilityMode**: 默认情况下, Framework20 设置使用本地时间为票证设定有效日期, 而 Framework40 设置使用的是 UTC。



推荐对 loginUrl 使用 SSL URL (https://), 以保护凭据不被盗取。

28.2.6 匿名身份

许多应用程序类型都需要处理匿名用户的功能, 电子商务 Web 应用程序尤其需要这个功能。电子商务站点必须支持匿名用户和验证用户。在匿名用户浏览站点并把商品添加到购物车中时, Web 应用程序需要一种唯一标识这些用户的方式。例如, 在 Amazon.com、BN.com 等繁忙的电子商务 Web 站点上, 并没有匿名用户的概念, 而是给每个用户赋予唯一的身份。

在 ASP.NET 的早期版本中, 并不支持开发人员对用户进行这种身份标识。大多数开发人员都使用 SessionID 唯一地标识用户。但是, 这种方法存在一些固有的缺陷。后来, ASP.NET 通过在配置文件中使用 <anonymousIdentification> 部分添加了匿名身份支持。程序清单 28-17 列出了 <anonymousIdentification> 配置部分的设置。

程序清单 28-17 在配置文件中使用匿名身份标识

```
<configuration>
  <system.web>
    <anonymousIdentification
      enabled="false" [true|false]
      cookieName=".ASPXANONYMOUS" [String]
      cookieTimeout="100000" [in Minutes][number]
      cookiePath="/" [String]
      cookieRequireSSL="false" [true|false]
      cookieSlidingExpiration="true" [true|false]
      cookieProtection="Validation" [None|Validation|Encryption|All]
      cookieless="UseCookies"
        [UseUri|UseCookies|AutoDetect|UseDeviceProfile]
      domain="" [String]
```



```

    />
  </system.web>
</configuration>

```

`<anonymousIdentification>`配置部分的 `enabled` 特性指定是否启用 ASP.NET 的匿名访问功能。其他特性对应于程序清单 28-16 中的 `<authentication>` 部分。在使用匿名身份标识时,终端用户会在他们的环境中禁用 cookie。在终端用户没有启用 cookie 时,将用户的身份存储在浏览器的 URL 字符串中。

28.2.7 授权

授权过程验证用户是否有权访问他试图请求的资源。ASP.NET 支持文件和 URL 授权。可以使用配置文件中的 `<authorization>` 部分控制应用程序指示的授权过程。`<authorization>` 部分如下面的程序清单 28-18 所示,可以包含子部分,用于给用户、包含在系统指定角色中的一组用户或以某种方式(如 HTTP GET 请求)发送给服务器的请求授予或拒绝访问权限。还可以使用 `<location>` 部分给应用程序中特定的文件夹或文件授予特定的权限。

程序清单 28-18 配置文件中的授权功能

```

<authorization>
  <allow users="" roles="" verbs="" />
  <deny users="" roles="" verbs="" />
</authorization>

```

1. URL 授权

URL 授权是 `URLAuthorizationModule`(继承于 `HttpModule`)提供的一项服务,用于控制对 .aspx 文件等资源的访问。如果要给某些用户或角色授予或拒绝对 ASP.NET 应用程序中某些部分的访问权限,URL 授权就很有用。

例如,ASP.NET 应用程序的管理部分只允许管理员访问,而拒绝其他人访问。使用 URL 授权可以很容易做到这一点。可以根据用户、角色、HTTP 谓词(如 HTTP GET 请求或 HTTP POST 请求)等配置 URL 授权。

可以在 `web.config` 文件中使用 `<allow>` 和 `<deny>` 特性配置 URL 授权。例如,程序清单 28-19 演示了如何给用户 Jason 授予访问应用程序的权限,而拒绝 Sales 组和 Marketing 组访问。

程序清单 28-19 在 `<authorization>` 部分允许或拒绝实体

```

<system.web>
  <authorization>
    <allow users="Jason" />
    <deny roles="Sales, Marketing" />
  </authorization>
</system.web>

```

`<allow>` 和 `<deny>` 元素支持 `users`、`roles` 和 `verbs` 值。从上面的代码示例中可以看出,可以添加多个用户和组,它们之间以逗号分隔。

`URLAuthorizationModule` 支持两个特殊的字符:星号(*)和问号(?)。星号(*)表示所有的用户(匿

名用户和注册用户), 而问号(?)只表示匿名用户。程序清单 28-20 中的代码示例拒绝所有匿名用户的访问, 并把访问权限授予 Admin 角色中的任何用户。

程序清单 28-20 拒绝匿名用户

```
<system.web>
  <authorization>
    <allow roles="Admin" />
    <deny users="?" />
  </authorization>
</system.web>
```

还可以给用户或组授予或拒绝对某些 HTTP 方法的访问权限。在下面的程序清单 28-21 中, 拒绝 Admin 角色中的用户访问 HTTP GET 方法, 并且拒绝所有用户访问 HTTP POST 方法。

程序清单 28-21 按照谓词拒绝用户和角色

```
<system.web>
  <authorization>
    <deny verbs="GET" roles="Admin" />
    <deny verbs="POST" users="*" />
  </authorization>
</system.web>
```

2. 文件授权

通过在配置文件中构造<authorization>部分, 就可以使用<location>元素把指定的权限应用于某个特定的文件或目录。例如, 应用程序有根目录 Home, 其中嵌套了子目录 Documents。假定要把 Documents 子目录的访问权限仅授予 Admin 角色中的用户, 如程序清单 28-22 所示。

程序清单 28-22 把 Documents 子目录的访问权限授予 Admin 角色

```
<configuration>
  <location path="Documents">
    <system.web>
      <authorization>
        <allow roles="Admin" />
        <deny users="*" />
      </authorization>
    </system.web>
  </location>
</configuration>
```



ASP.NET 应用程序不验证 path 特性中指定的路径。如果给定的路径无效, ASP.NET 就不会应用安全设置。

还可以为单个文件设置安全性, 如程序清单 28-23 所示。

程序清单 28-23 把某个文件的访问权限授予 Admin 角色

```
<configuration>
  <location path="Documents/Default.aspx">
    <system.web>
      <authorization>
        <allow roles="Admin" />
        <deny users="*" />
      </authorization>
    </system.web>
  </location>
</configuration>
```

28.2.8 锁定配置设置

ASP.NET 的配置系统非常灵活，可以把配置信息应用于特定的应用程序或文件夹。我们有时还希望限制服务器上某个应用程序可以控制的配置选项，例如改变 ASP.NET 会话信息的存储方式。使用<location>元素的 allowOverride、allowDefinition 以及 path 特性就可以实现这种锁定过程。

在程序清单 28-24 中，machine.config 文件中的<location>部分指定了路径"Default Web Site/ExampleApplication"，允许所有的应用程序使用 allowOverride 特性重写跟踪设置。

程序清单 28-24 允许在低级的配置文件中重写<trace>部分

```
<configuration>
  <location path="Default Web Site/ExampleApplication" allowOverride="true">
    <trace enabled="false"/>
  </location>
</configuration>
```

可以重写跟踪设置是因为 allowOverride 特性设置为 true。在 ExampleApplication 的 web.config 文件中重写跟踪设置，启用本地的<trace>元素，就可以重写程序清单 28-24 中的设置。

但是，如果在 machine.config 文件的<location>部分将特性设置为 allowOverride="false"，ExampleApplication 的 web.config 文件就不能重写跟踪设置。

28.2.9 ASP.NET 页面配置

部署完 ASP.NET 应用程序后，可以使用配置文件的<pages>部分控制每个 ASP.NET 页面的一些默认操作。这些操作包括是否在页面发送输出之前缓冲它们，是否为整个应用程序启用会话状态等。使用<pages>部分的例子如程序清单 28-25 所示。

程序清单 28-25 配置<pages>部分

```
<configuration>
  <system.web>
    <pages
      buffer="true" [true]|false]
      enableSessionState="true" [False|ReadOnly|True]
      enableViewState="true" [true]|false]
      enableViewStateMac="true" [true]|false]
      enableEventValidation="true" [true]|false]
```



```

    smartNavigation="false" [true|false]
    autoEventWireup="true" [true|false]
    maintainScrollPositionOnPostBack="false" [true|false]
    pageBaseType="System.Web.UI.Page" [String]
    userControlBaseType="System.Web.UI.UserControl" [String]
    pageParserFilterType="" [String]
    validateRequest="true" [true|false]
    masterPageFile="" [String]
    theme="" [String]
    styleSheetTheme="" [String]
    maxPageStateFieldLength="-1" [number]
    compilationMode="Always" [Auto|Never|Always]
    viewStateEncryptionMode="Auto" [Auto|Always|Never]
    asyncTimeout="45" [in Seconds][number]
    renderAllHiddenFieldsAtTopOfForm="true" [true|false]
    clientIDMode="Predictable" [Inherit|AutoID|Predictable|Static]
    controlRenderingCompatibilityVersion="4.0" [Version] >
    <namespaces autoImportVBNamespace="true" [true|false] >
        <add namespace="" [String, Required, CollectionKey] />
    </namespaces>
    <controls />
    <tagMapping />
    <ignoreDeviceFilters />
</pages>
</system.web>
</configuration>

```

下面的列表详细描述了一些 ASP.NET 页面配置信息元素:

- **buffer**: 指定在发送给客户端之前是否必须在服务器上缓冲这些请求。
- **enableSessionState**: 指定是否为当前 ASP.NET 应用程序启用会话状态。值可以是 true、false 和 readonly。readonly 值表示应用程序只能读取会话值,但不能修改它们。
- **enableViewState**: 指定是否为所有的控件启用 ViewState。如果应用程序不使用 ViewState,就可以在应用程序的 web.config 文件中把值设置为 false。
- **autoEventWireup**: 指定 ASP.NET 是否自动启动常见的页面事件,如 Load 或 Error。
- **smartNavigation**: 智能导航把 IE 用作客户端浏览器,以防止将页面回送给自身时重新绘制页面。使用智能导航,会通过客户端的 IFRAME 发送请求,IE 仅重新绘制页面上有改动的部分。这个选项默认设置为 false。启用该选项时,它只能用于 IE 浏览器,而其他浏览器都采用标准操作。
- **maintainScrollPositionOnPostBack**: 指定在回送之后是否让用户返回到页面上的相同位置。如果将该特性的值设置为 False(默认设置),用户将返回到页面顶部。
- **masterPageFile**: 标识当前 ASP.NET 应用程序的母版页。如果要把母版页应用于页面的子集(例如包含在应用程序的某个文件夹中的页面),就可以使用 web.config 文件中的 <location>元素:

```

<configuration>
    <location path="ExampleApplicationAdmin">
        <system.web>
            <pages
                masterPageFile="~/ExampleApplicationAdminMasterPage.master"

```

```

        />
    </system.web>
</location>
</configuration>

```

- **theme:** 指定用于页面的主题的名称。
- **styleSheetTheme:** 定义声明控件之后要使用的主题。
- **maxPageStateFieldLength:** 如果设置为正值, ASP.NET 就会将 ViewState 分成多个块, 这些块的大小将小于定义的大小。默认值为 -1, 这意味着所有的 ViewState 将分成一块。
- **pageBaseType:** 指定当前 ASP.NET 应用程序中所有 ASP.NET 页面的基类。这个选项默认设置为 System.Web.UI.Page。但是, 如果希望所有的 ASP.NET 页面都继承于其他基类, 就可以通过这个设置改变默认值。
- **userControlBaseType:** 指定当前 ASP.NET 应用程序中所有 ASP.NET 用户控件的基类, 默认为 System.Web.UI.UserControl。可以使用这个元素重写默认选项。
- **validateRequest:** 指定 ASP.NET 是否验证所有具有潜在危险(例如跨站点脚本攻击和脚本注入攻击)的入站请求。这个功能会自动检查请求中的所有参数, 确保它们的内容不包含 HTML 元素, 为防止跨站点脚本攻击和脚本注入攻击提供很好的保护。有关这个设置的更多信息, 可参阅 <http://www.asp.net/faq/RequestValidation.aspx>。
- **namespaces:** 导入可以在预处理过程中包含的一组程序集。
- **compilationMode:** 指定 ASP.NET 如何编译当前的 Web 应用程序, 值可以是 Never、Always 和 Auto。设置 compilationMode="Never" 时, 表示页面从来不编译。如果页面中的结构需要编译, 就会产生错误。设置 compilationMode="Always" 时, 表示总是编译页面。设置 compilationMode="Auto" 时, 即使页面可以编译, ASP.NET 也不会编译。
- **viewStateEncryptionMode:** 指定是否加密 ViewState。
- **asyncTimeout:** 指定在异步操作的过程中页面应等待异步处理程序完成的秒数。
- **clientIDMode:** 指定给页面上的服务器控件创建 ClientID 值所用的算法。默认设置是 AutoID, 对于服务器控件, 默认值是 Inherit。
- **controlRenderingCompatibilityVersion:** 指定控件显示 HTML 时的 ASP.NET 版本。

28.2.10 包含文件

ASP.NET 在 machine.config 和 web.config 文件中都支持包含文件。配置内容包含在多个地方或位于<location>元素中时, 包含文件就是封装这些内容的绝佳方式。

使用<pages>部分的 configSource 特性, 配置文件中的任意部分都可以包含另一个文件中的内容, 该特性的值表示包含文件的虚拟相对文件名。程序清单 28-26 是一个例子。

程序清单 28-26 给 web.config 文件添加其他内容

```

<configuration>
  <system.web>
    <pages configSource="SystemWeb.config" />
  </system.web>
</configuration>

```


配置包含文件可以包含应用于单个部分的信息，单个包含文件不能包含多个配置部分或某个配置部分的一部分。如果设置了 `configSource` 特性，源文件中的部分元素就不应包含其他特性或子元素。

然而，包含文件并不是完整的配置文件，而应只包含 `<pages>` 部分，如程序清单 28-27 所示。

程序清单 28-27 SystemWeb.config 文件

```
<pages authentication mode="Forms" />
```

`configSource` 特性不能嵌套。包含文件不能使用 `configSource` 特性嵌套其他文件。



ASP.NET 配置文件发生改变时，应用程序会在运行时重新启动。在配置文件中使用外部包含文件时，配置将在不重新启动应用程序的情况下重新加载。

28.2.11 配置 ASP.NET 运行时设置

一般的配置设置能指定给定的 ASP.NET 资源(如页面)在超时之前可以执行多长时间。其他设置指定请求的最大尺寸(以 KB 为单位)，或者指定是否在重定向时使用完全限定的 URL。可以使用配置文件中的 `<httpRuntime>` 部分来指定这些设置。`<httpRuntime>` 元素应用于文件夹级别的 ASP.NET 应用程序。程序清单 28-28 列出了 `<httpRuntime>` 部分中使用的默认值。

程序清单 28-28 <httpRuntime> 部分

```
<configuration>
  <system.web>
    <httpRuntime
      useFullyQualifiedRedirectUrl="false"
      enable="true"
      executionTimeout="90"
      maxRequestLength="4096"
      requestLengthDiskThreshold="512"
      appRequestQueueLimit="5000"
      minFreeThreads="8"
      minLocalRequestFreeThreads="4"
      enableKernelOutputCache="true" />
  </system.web>
</configuration>
```

1. 启用和禁用 ASP.NET 应用程序

`enable` 特性指定是否启用当前的 ASP.NET 应用程序。将该特性设置为 `false` 时，就禁用当前的 ASP.NET 应用程序，试图连接这个站点的所有客户端都会收到 HTTP 404—File Not Found 异常。应只在计算机或应用程序级别设置这个值。如果在其他级别设置这个值(如子文件夹级别)，就会忽略该值。这是一项很强大的功能，它允许管理员关闭应用程序，而无须启动或停止 IIS。该特性的默认值是 `true`。



除了这个设置之外,还可以在应用程序的根目录下放置 App_Offline.htm 文件,使应用程序快速脱机。这个.htm 文件不需要包含具体内容(不会带来任何区别)。但如果这个文件的大小没有超过 512 字节,那么对应用程序的所有请求都会获得 Page Not Found 错误。

2. 完全限定的重定向 URL

useFullyQualifiedRedirectUrl 特性指定客户端重定向是否应包含完全限定的 URL。在给移动设备编程时,一些设备需要指定完全限定的 URL。该特性的默认值是 false。

3. 请求超时

executionTimeout 设置指定 ASP.NET 请求的超时选项。这个特性的值是在 ASP.NET 把请求设定为超时之前资源可以执行的时间(秒)。默认设置为 110 秒。如果某个 ASP.NET 页面或 Web 服务执行的时间超过了 110 秒,就可以在配置文件中延长该时限。

4. 请求的最大尺寸

maxRequestLength 特性指定 ASP.NET 运行时可以接受的最大文件尺寸。例如,如果 ASP.NET 应用程序需要处理大型文件,那么最好修改这个设置,默认值是 4096KB(4MB)。

目前,Web 应用程序很容易受到攻击,这些攻击包括脚本注入攻击和拒绝服务(Denial of Service, DoS)攻击。DoS 是一种使用大文件的请求轰炸 Web 服务器的攻击。这些大量的请求最终会使 Web 服务器崩溃。利用 maxRequestLength 特性可以设置请求的大小限制,防止 DoS 攻击。

5. 缓冲上传

ASP.NET 4.5 包含设置 requestLengthDiskThreshold。这个设置允许管理员配置文件上传的缓冲行为,而不会影响编程模型。管理员可以配置阈值,小于阈值的请求会缓冲到内存中。请求超过该限制后,就会透明地缓冲到磁盘上,由用于使用这些数据的机制利用。这个设置的有效值在 1 到 Int32.MaxValue 之间,单位是 KB。

启用文件缓冲功能时,文件会上传到 codegen 文件夹。codegen 文件夹的默认路径是:

```
[WinNT\Windows]\Microsoft.NET\Framework\[version]\Temporary ASP.NET Files\[ApplicationName]
```

在 codegen 文件夹 Uploads 的子文件夹中使用随机名称缓冲文件。可以使用<compilation>部分的 tempDirectory 特性为每个应用程序配置 codegen 文件夹的位置。

6. 线程管理

ASP.NET 运行时可以使用线程池中的空闲线程来处理请求。MinFreeThreads 特性指定 ASP.NET 保证线程池中可用的线程数,默认值是 8。对于需要额外线程来完成处理的复杂应用程序来说,这个特性可以确保有可用的线程,应用程序在等待空闲线程来完成工作时不会被阻塞。minLocalRequestFreeThreads 特性控制用于处理本地请求的空闲线程数。该特性的默认为 4。

7. 应用程序的队列长度

appRequestQueueLimit 特性指定 ASP.NET 为当前 ASP.NET 应用程序排队的最大请求数。在没有足够的空闲线程处理请求时, ASP.NET 会给这些请求排队。minFreeThreads 特性指定 ASP.NET 应用程序应保持的空闲线程数, 这个设置会影响存储在队列中的请求数。



在排队的请求数超过 appRequestQueueLimit 设置的限制时, 会拒绝所有的进站请求, 并向浏览器显示 HTTP 503 - Server Too Busy 错误。

8. 输出高速缓存

enableKernalOutputCache 特性指定在 IIS 核心级别(Http.sys)是否启用输出高速缓存。目前, 这个设置只能应用于 Web 服务器的 IIS 6 或更高版本。

28.2.12 配置 ASP.NET 工作者进程

IIS 接收到 ASP.NET 页面的请求时, 会把请求传送给非托管的 DLL, 即 aspnet_isapi.dll。aspnet_isapi.dll 进一步把请求传送给另一个工作者进程。在 IIS 5 中, 该工作者进程是 aspnet_wp.exe, 它会运行所有的 ASP.NET 应用程序。而在 IIS 6 或更高的版本中, 所有的 ASP.NET 应用程序都由 w3wp.exe 进程运行。可以使用 machine.config 文件中的 <processModel> 部分配置 ASP.NET 工作者进程。



前面介绍的所有配置部分都由托管代码读取, 而 <processModel> 部分由非托管 DLL aspnet_isapi.dll 读取。因为配置信息由非托管 DLL 读取, 所以只有重新启动 IIS 后, 改变的处理模型信息才会应用于所有的 ASP.NET 应用程序。

程序清单 28-29 演示了 <processModel> 部分的默认格式。

程序清单 28-29 <processModel> 元素的结构

```
<processModel
  enable="true|false"
  timeout="hrs:mins:secs|Infinite"
  idleTimeout="hrs:mins:secs|Infinite"
  shutdownTimeout="hrs:mins:secs|Infinite"
  requestLimit="num|Infinite"
  requestQueueLimit="num|Infinite"
  restartQueueLimit="num|Infinite"
  memoryLimit="percent"
  cpuMask="num"
  webGarden="true|false"
  userName="username"
  password="password"
  logLevel="All|None|Errors"
  clientConnectedCheck="hrs:mins:secs|Infinite"
```



```

responseDeadlockInterval="hrs:mins:secs|Infinite"
responseRestartDeadlockInterval="hrs:mins:secs|Infinite"
comAuthenticationLevel="Default|None|Connect|Call|
Pkt|PktIntegrity|PktPrivacy"
comImpersonationLevel="Default|Anonymous|Identify|
Impersonate|Delegate"
maxWorkerThreads="num"
maxIoThreads="num"
autoConfig="true|false"
minWorkerThreads="num"
minIoThreads="num"
serverErrorMessageFile=""
pingFrequency="hrs:mins:secs|Infinite"
pingTimeout="hrs:mins:secs|Infinite"
maxAppDomains="number"
/>

```

下面详细介绍这些特性：

- **enable**：指定是否启用进程模型。将该特性设置为 **false** 时，ASP.NET 应用程序运行在 IIS 的进程模型下。



ASP.NET 运行在 IIS 6 或更高版本的本机模式下时，使用 IIS 6 或更高版本的进程模型，忽略配置文件中的 `<processModel>` 部分，但仍然应用 **autoConfig** 和 **requestQueueLimit** 特性。

- **timeout**：指定在创建新的工作者进程以代替当前的工作者进程之前，当前工作者进程的存活时间。如果应用程序在运行几个星期后，性能因为内存泄漏而开始略有下降，就可以使用这个值。我们不需要手动启动和停止进程，ASP.NET 可以自动重启。默认值是 **Infinite**。
- **idleTimeout**：指定工作者进程在停止前应等待的时间。可以使用 **idleTimeout** 选项自动停止 ASP.NET 工作者进程，默认值是 **Infinite**。也可以使用 **HH:MM:SS** 格式把这个值设置为某个时间。
- **shutdownTimeout**：指定在 ASP.NET 调用 **Kill** 命令之前让工作者进程自己停止的时间。**Kill** 是一个低级命令，能强制退出进程。该特性的默认值是 5 秒。
- **requestLimit**：指定 ASP.NET 工作者进程在处理一定数量的请求之后应何时进行再循环，默认值是 **Infinite**。
- **requestQueueLimit**：指定当超过排队的请求数时 ASP.NET 再循环工作者进程。默认设置是 5000。
- **memoryLimit**：指定在发现工作者进程误操作或泄漏内存之前允许其使用的物理内存大小，默认值是可用物理内存的 60%。
- **username** 和 **password**：默认使用 ASP.NET 身份执行所有的 ASP.NET 应用程序。如果希望使用另一个账户运行 ASP.NET 应用程序，就可以使用这两个特性提供用户名和密码对。
- **logLevel**：指定 ASP.NET 工作者进程记录事件的方式。默认设置是只记录错误。还可以指定 **None** 以禁用日志记录功能，或者指定 **All** 以记录所有信息。所有的日志项都写入 Windows 应用程序事件日志。

- **clientConnectedCheck**: clientConnectedCheck 设置可以在开始执行工作前检查客户端是否在指定的时间内处于连接状态, 默认值是 5 秒。
- **responseDeadlockInterval**: 指定死锁检查发生的频率。当请求在排队并且在这个时间段内没有发送响应时, 就表示出现死锁。死锁发生后, 进程要重新启动。默认值是 3 分钟。
- **responseRestartDeadlockInterval**: 指定当 CLR 运行时检测出发生死锁时, 在重新启动进程之前 CLR 运行时应等待的时间。默认值是 9 分钟。
- **comAuthenticationLevel**: 控制 DCOM 安全性的验证级别, 默认设置为 Connect, 其他值有 Default、None、Call、Pkt、PktIntegrity 和 PktPrivacy。
- **comImpersonationLevel**: 控制 COM 安全性的验证级别, 默认设置为 Impersonate, 其他值有 Default、Anonymous、Identify 和 Delegate。
- **webGarden**: 指定是否启用 Web Garden 模式。默认设置是 false。Web Garden 允许把多个 ASP.NET 工作者进程放在一台服务器上, 从而为应用程序提供更好的硬件扩展性。只有多处理器的服务器才支持 Web Garden 模式。
- **cpuMask**: 指定在 webGarden="true" 时, 哪些处理器应与 ASP.NET 工作者进程相近。cpuMask 是一个 16 进制值, 其默认值是所有的处理器, 显示为 0xFFFFFFFF。
- **maxWorkerThreads**: 指定 ASP.NET 工作者进程线程池中的最大线程数, 默认值为 20。
- **maxIoThreads**: 指定 ASP.NET 工作者进程中的最大 I/O 线程数, 默认值为 20。
- **autoConfig**: 指定是否配置 ASP.NET 应用程序的性能设置。
- **minWorkerThreads**: 指定 ASP.NET 工作者进程线程池中的最小线程数。默认值是 1。
- **minIoThreads**: 指定 ASP.NET 工作者进程中的最小 I/O 线程数。默认值是 1。
- **serverErrorMessageFile**: 指定用于错误消息而不是默认的“服务器不可用”消息的内容页面。
- **pingFrequency**: 指定一个时间段, 在这个时间段内 ISAPI ping 工作者进程以确定工作者进程是否在运行。
- **pingTimeout**: 指定一个时间段, 在该时间段内如果工作者进程没有响应, 就进行重启。
- **maxAppDomains**: 为进程设置绝对最大的应用程序域数量。

使用 .NET Framework 的多个版本运行多个 Web 站点

在与 ASP.NET 工作者进程相同的环境下, 在给定的 Web 服务器上可以驻留多个 Web 站点。每一个这样的站点都可以绑定到 .NET Framework 的特定版本。一般使用 aspnet_regiis.exe 实用程序实现该操作。.NET Framework 的每个版本都带有 aspnet_regiis.exe 实用程序。

这个实用程序有多个选项, 使用 -s 选项可以为给定的 Web 站点安装 .NET Framework 运行时的当前版本。程序清单 28-30 演示了如何为 Web 站点 ExampleApplication 安装 .NET Framework 2.0 版本。

程序清单 28-30 为 Web 站点 ExampleApplication 安装 .NET Framework 2.0 版本

```
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727>
aspnet_regiis -s W3SVC/1ROOT/ExampleApplication
```


28.2.13 存储与应用程序相关的设置

每个 Web 应用程序都必须存储一些与应用程序相关的设置，供其运行时使用。web.config 文件的<appSettings>部分可以为 ASP.NET 应用程序定义定制应用程序设置。该部分可以包含多个<add>子元素，语法如下：

```
<appSettings>
  <add key="[key]" value="[value]"/>
</appSettings>
```

<add>子元素支持如下两个特性：

- key: 指定 appSettings 散列表中的键值。
- value: 指定 appSettings 散列表中的值。

程序清单 28-31 演示了如何存储与应用程序相关的连接字符串。key 值设置为 ApplicationInstanceID，value 值设置为 ASP.NET 应用程序实例和运行应用程序的服务器名称。

程序清单 28-31 应用程序实例信息

```
<appSettings>
  <add key="ApplicationInstanceID" value="InstanceOnServerOprta"/>
</appSettings>
```

28.2.14 对配置文件编程

现在，ASP.NET 包含的 API(ASP.NET Management Object)可以操作 machine.config 和 web.config 文件中的配置信息设置。ASP.NET Management Object 提供了强类型化的编程模型，除了具有 .NET Web Application Server 管理方面的功能外，还可以控制 ASP.NET Web 配置的创建和维护。使用 ASP.NET Management Object，可以操作存储在本地或远程计算机上的配置文件中的配置信息。这些配置信息可以用于完成常见的管理任务，或者编写安装脚本。

所有的 ASP.NET Management Object 都存储在 System.Configuration 和 System.Web.Configuration 名称空间中。使用 WebConfigurationManager 类可以访问配置。System.Configuration.Configuration 类表示 machine.config 文件和 web.config 文件中配置设置的合并视图。System.Configuration 和 System.Web.Configuration 名称空间中有多个类可以用于访问配置文件中的所有设置。System.Configuration 和 System.Web.Configuration 名称空间之间的主要区别是，System.Configuration 名称空间包含应用于所有 .NET 应用程序的所有类，而 System.Web.Configuration 名称空间包含只能用于 ASP.NET Web 应用程序的类。表 28-1 列出了 System.Configuration 名称空间中的重要类及其作用。

表 28-1

类 名	作 用
Configuration	可以操作存储在本地或远程计算机上的配置
ConfigurationElementCollection	可以枚举存储在配置文件中的子元素
AppSettingsSection	可以操作配置文件中的<appSettings>部分
ConnectionStringSettings	可以操作配置文件中的<connectionStrings>部分

(续表)

类 名	作 用
ProtectedConfigurationSection	可以操作配置文件中的<protectedConfiguration>部分
ProtectedDataSection	可以操作配置文件中的<protectedData>部分

表 28-2 列出了 System.Web.Configuration 名称空间中的一些类及其作用。

表 28-2

类 名	作 用
AuthenticationSection	可以操作配置文件中的<authentication>部分
AuthorizationSection	可以操作配置文件中的<authorization>部分
CompilationSection	可以操作配置文件中的<compilation>部分
CustomErrorsSection	可以操作配置文件中的<customErrors>部分
FormsAuthenticationConfiguration	可以操作配置文件中的<forms>部分
GlobalizationSection	可以操作配置文件中的<globalization>部分
HttpHandlersSection	可以操作配置文件中的<httpHandlers>部分
HttpModulesSection	可以操作配置文件中的<httpModules>部分
HttpRuntimeSection	可以操作配置文件中的<httpRuntime>部分
MachineKeySection	可以操作配置文件中的<machineKey>部分
MembershipSection	可以操作配置文件中的<membership>部分
PagesSection	可以操作配置文件中的<pages>部分
ProcessModelSection	可以操作配置文件中的<processModel>部分
WebPartsSection	可以操作配置文件中的<webParts>部分

可以基于简单的面向对象体系结构来实现所有的配置类，面向对象体系结构有存储所有数据的实体类和包含添加、删除、枚举等方法的集合类。可以使用简单的连接字符串枚举以开始配置文件的编程，如下所述。

1. 枚举连接字符串

在 Web 应用程序中，可以存储多个连接字符串，其中一些连接字符串由系统使用，另外一些连接字符串与应用程序相关。下面编写一个非常简单的 ASP.NET 应用程序，枚举存储在 web.config 文件中的所有连接字符串，如程序清单 28-32 所示。

程序清单 28-32 web.config 文件

```
<?xml version="1.0" ?>
<configuration>
  <appSettings>
    <add key="symbolServer" value="192.168.1.1" />
  </appSettings>
  <connectionStrings>
```



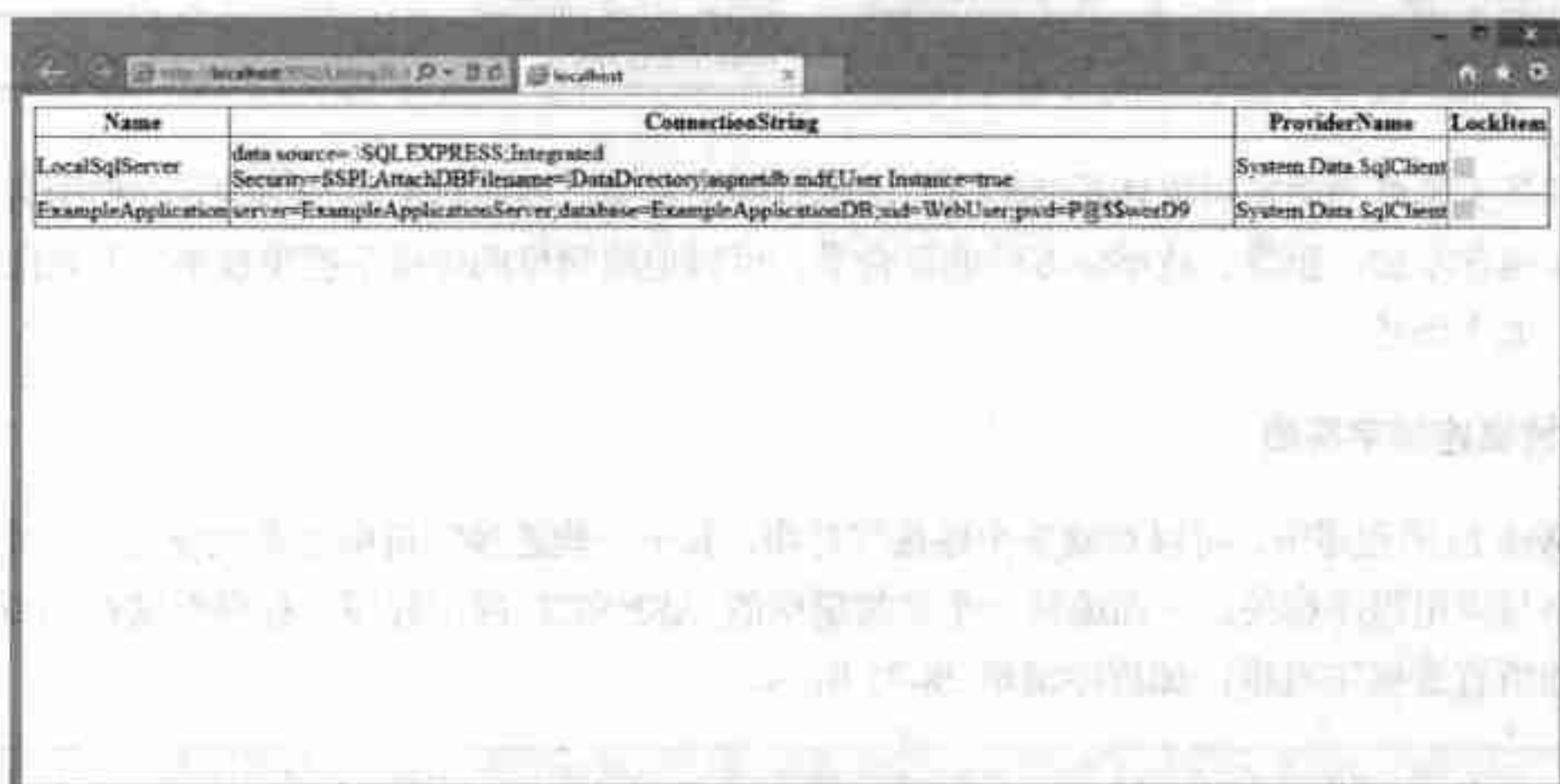
```
<add name="ExampleApplication"
  connectionString="server=ExampleApplicationServer;
  database=ExampleApplicationDB;uid=WebUser;pwd=P@$$word9"
  providerName="System.Data.SqlClient" />
</connectionStrings>
<system.web>
  <compilation debug="false" targetFramework="4.5" />
  <authentication mode="None" />
</system.web>
</configuration>
```

如程序清单 28-32 所示，有个应用程序设置指向符号服务器，并且有个连接字符串存储在 web.config 文件中。使用 System.Web.Configuration.WebConfigurationManager 类的 ConnectionStrings 集合读取连接字符串，如程序清单 28-33 所示。

程序清单 28-33 把 ConnectionStrings 集合属性绑定到 GridView 控件

```
protected void Page_Load(object sender, EventArgs e)
{
  GridView1.DataSource =
    System.Web.Configuration.WebConfigurationManager.ConnectionStrings;
  GridView1.DataBind();
}
```

如程序清单 28-33 所示，把 WebConfigurationManager 类的 ConnectionStrings 属性集合绑定到 GridView 控件。WebConfigurationManager 类返回 Configuration 类的一个实例，ConnectionStrings 是静态属性。因此，把 ConnectionStrings 属性集合绑定到 GridView 控件。图 28-4 显示了存储在 ASP.NET 应用程序中的连接字符串列表。



Name	ConnectionString	ProviderName	LockItem
LocalSqlServer	data source=.\SQLEXPRESS;integrated security=SSPI;AttachDBFilename=DataDirectory\aspnetdb.mdf;User Instance=true	System.Data.SqlClient	
ExampleApplication	server=ExampleApplicationServer;database=ExampleApplicationDB;uid=WebUser;pwd=P@\$\$word9	System.Data.SqlClient	

图 28-4

在运行期间添加连接字符串是很简单的任务。如果执行程序清单 28-34 中的代码，就会得到配置对象的一个实例。接着创建新类 connectionStringSettings，把这个新类添加到集合中，并调用更新方法。程序清单 28-34 列举了这个例子的 C#代码。

程序清单 28-34 添加连接字符串

```

protected void Button1_Click(object sender, EventArgs e)
{
    // Get the file path for the current web request
    string webPath = Request.ApplicationPath;

    // Get configuration object of the current web request
    Configuration config =
        System.Web.Configuration.WebConfigurationManager.OpenWebConfiguration(webPath);

    // Create new connection setting from text boxes
    ConnectionStringSettings newConnSetting = new
        ConnectionStringSettings(txtName.Text, txtValue.Text, txtProvider.Text);

    try
    {
        // Add the connection string to the collection
        config.ConnectionStrings.ConnectionStrings.Add(newConnSetting);

        // Save the changes
        config.Save();
    }
    catch(ConfigurationErrorsException cEx)
    {
        lblStatus.Text = "Status: " + cEx.ToString();
    }
    catch(System.UnauthorizedAccessException uEx)
    {
        // The ASP.NET process account must have read/write
        // access to the directory
        lblStatus.Text = "Status: " + "The ASP.NET process account must have" +
            "read/write access to the directory";
    }
    catch(Exception eEx)
    {
        lblStatus.Text = "Status: " + eEx.ToString();
    }

    // Reload the connection strings in the list box
    ShowConnectionStrings();
}

protected void ShowConnectionStrings()
{
    GridView1.DataSource = System.Web.Configuration.WebConfigurationManager.ConnectionStrings;
    GridView1.DataBind();
}

```

2. 操作 machine.config 文件

System.Configuration.ConfigurationManager 类的 OpenMachineConfiguration 方法提供了操作 machine.config 文件的方式。OpenMachineConfiguration 是静态方法。

程序清单 28-35 中的简单例子列举了存储在 machine.config 文件中的所有部分组。正如该程序清

单所示, 我们使用 `OpenMachineConfiguration` 方法得到了配置对象的一个实例, 接着把 `SectionGroups` 集合绑定到 `GridView` 控件。

程序清单 28-35 machine.config 文件中的配置组

```
protected void Page_Load(Object sender, EventArgs e)
{
    // List all the SectionGroups in machine.config file
    Configuration configSetting =
        System.Configuration.ConfigurationManager.OpenMachineConfiguration();
    GridView1.DataSource = configSetting.SectionGroups;
    GridView1.DataBind();
}
```

28.2.15 保护配置设置

ASP.NET 使用 Data Protection API (DPAPI) 将配置信息存储到注册表中。例如, 程序清单 28-36 演示了如何在注册表中存储进程模型部分的用户名和密码信息。

程序清单 28-36 在注册表中存储用户名和密码信息, 再在 machine.config 文件中引用这些设置

```
<processModel
    userName="registry:HKLM\SOFTWARE\ExampleApp\Identity\ASPNET_SETREG,userName"
    password="registry:HKLM\SOFTWARE\ExampleApp\Identity\ASPNET_SETREG,password"
/>
```

ASP.NET 4.5 为保护配置系统中存储的敏感数据提供了一种机制, 该机制使用业界标准的 XML 加密方法来加密包含敏感数据的配置部分。

开发人员常常把敏感数据(如连接字符串、密码等)放在 `web.config` 文件中。现在, ASP.NET 允许采用人和计算机均不可读的方式存储这些数据, 并且不需要对加密技术和加密过程中使用的键有太多的了解。

在 `web.config` 文件中, 最常加密的项是 `<connectionStrings>` 部分。程序清单 28-37 是包含连接字符串的 `web.config` 文件示例。

程序清单 28-37 web.config 文件中包含了标准的连接字符串

```
<?xml version="1.0"?>
<configuration>
  <appSettings/>
  <connectionStrings>
    <add name="AdventureWorks"
        connectionString="Server=localhost;Integrated Security=True;
        Database=AdventureWorks"
        providerName="System.Data.SqlClient" />
  </connectionStrings>
  <system.web>
    <compilation debug="false" />
    <authentication mode="Forms">
      <forms name="Wrox" loginUrl="Login.aspx" path="/">
        <credentials passwordFormat="Clear">
          <user name="JasonGaylord" password="Reindeer" />
        </credentials>
      </forms>
    </authentication>
  </system.web>
</configuration>
```



```

        </credentials>
    </forms>
</authentication>
</system.web>
</configuration>

```

在这个例子中，要把这个连接字符串加密到数据库中。为此，ASP.NET 的安装程序提供了工具 `aspnet_regiis.exe`，它位于 `C:\WINDOWS\Microsoft .NET\Framework\v4.0.30319` 目录中。要使用这个工具加密 `<connectionStrings>` 部分，可以打开命令提示窗口，使用 `cd C:\WINDOWS\Microsoft .NET\Framework\v4.0.30319` 导航到指定的文件夹。另一种方法是以管理员身份打开 Visual Studio 2012 的命令提示符。在其中一种环境下，使用程序清单 28-38 中的语法即可加密 `<connectionStrings>` 部分。

程序清单 28-38 加密 `<connectionStrings>` 部分

```
aspnet_regiis -pe "connectionStrings" -app "/EncryptionExample"
```

运行这个脚本，会得到如图 28-5 所示的结果。

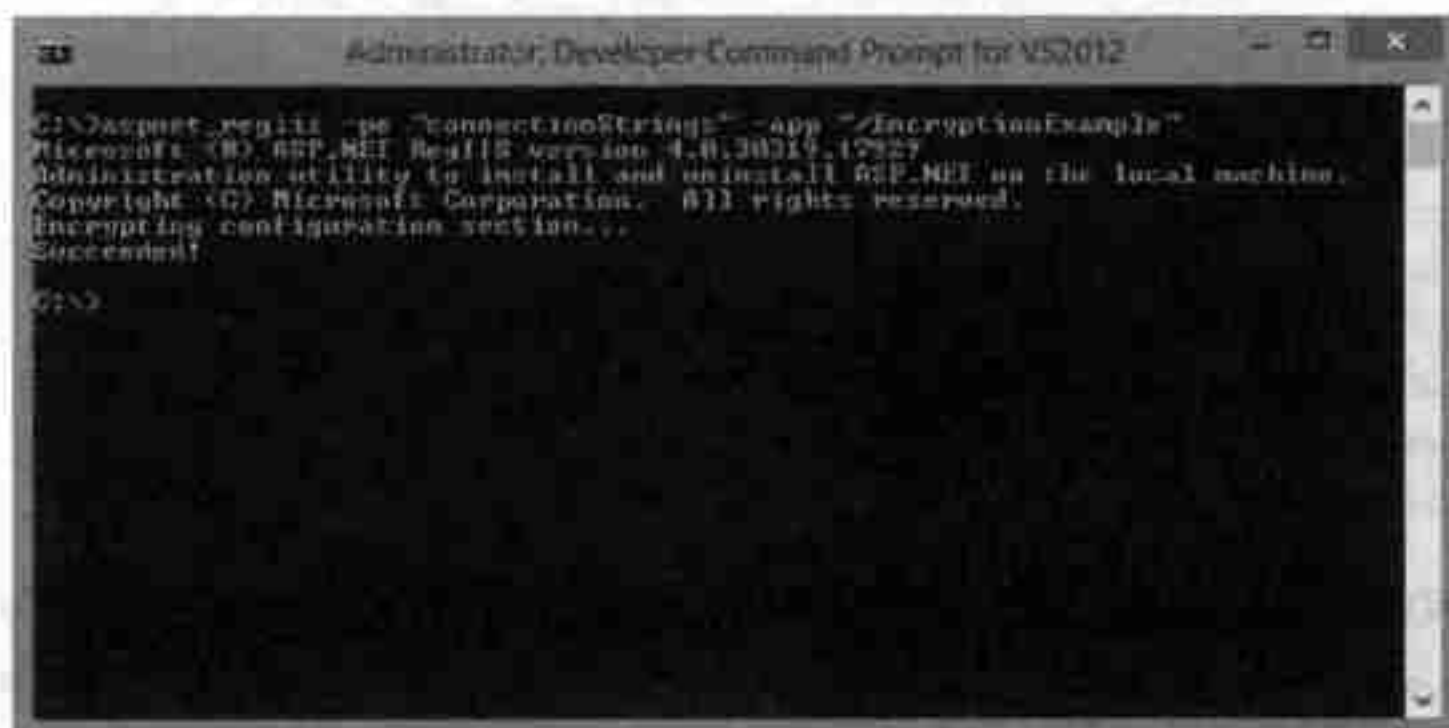


图 28-5

仔细查看加密过程中使用的脚本，会发现 `-pe` 命令指定 `web.config` 文件中要加密的部分，而 `-app` 命令指定要处理的应用程序。如果再查看 `web.config` 文件中的加密部分，就会看到类似于程序清单 28-39 中的代码。

程序清单 28-39 `web.config` 文件中加密的 `<connectionStrings>` 部分

```

<?xml version="1.0"?>
<configuration>
  <connectionStrings configProtectionProvider="RsaProtectedConfigurationProvider">
    <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
      xmlns="http://www.w3.org/2001/04/xmlenc#"
      <EncryptionMethod
        Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#"
        <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#"
          <EncryptionMethod
            Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
          <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#"
            <KeyName>Rsa Key</KeyName>
          </KeyInfo>

```

```

    <CipherData>
      <CipherValue>UAqurj7pvS7WLwt4CSXJN2Fe2yYcrXA1K
        Go33DVzec0nOhUy6FY0751MARiaJvqSepvkuRPSU3ZRP
        xESmc7p4t3N6i9OGT3q3xHIzSJV7rzSSpD2cd+DxQkwPp
        oXeGx7x7H22DSWrxDEyn4EJQf3ZeTY2tcTcDcvH4UNLTD
        wIW2ACM56s/OOZOCOVUq4nKdRC0q4W8enBoNIvhDdL2E5
        ZpTUAsJl4MvrWOMqlVX8F0P6Osn+apX5Zh9QhvPLXz7G8
        nbwNzSc8tLuCWlM4l9dmM6r97vID6qxNgBz3bJbM03BjT
        WUBQEXSN5HBmuAVFllQjzBDZbqFrBl+Mgu7TpQMw==</CipherValue>
    </CipherData>
  </EncryptedKey>
</KeyInfo>
<CipherData>
  <CipherValue>OEr7uMFQU7736eCharUocrRs442uD3Ivi2woGfon
    SnpIReILGlksfMdfdF7CLzfUIt8KIARZeKkAswJrOVDjinM75840
    QLmwFnQhtSqKqphM92kbudTsmVrQkKJFN6Y2PElTpa9BG+nEf0HX
    Y7cERhj2Yv1Hua7B+oYhM5TtfPUTEP3fpqROPNSlWvRmJN2XwDyj
    mJTGON2Lk0phAAf8rLwVB0IGT3lF6CKsK0YCPuiMAcFAHzcgzEXP
    UFXOqJqXBFjoEL0jx/sWV4DHOFJq5p2+DPxHKoI4ZTWolploxFWF
    roleLimzygvjnsUmbHyP</CipherValue>
</CipherData>
</EncryptedData>
</connectionStrings>
<system.web>
  <compilation debug="true" targetFramework="4.5"/>
  <httpRuntime targetFramework="4.5"/>
</system.web>
</configuration>

```

在处理 ASP.NET 应用程序中的连接字符串时，ASP.NET 会自动解密这个部分，以利用其中存储的值。在 web.config 文件的<system.web>部分，有一个子部分，它把用户名和密码显示为明文。对于这些信息也要进行加密，以防止被盗取。因为是子部分，所以使用程序清单 28-40 中的脚本。

程序清单 28-40 加密<authentication>部分

```
aspnet_regiis -pe "system.web/authentication" -app "/EncryptionExample"
```

运行这段代码，程序清单 28-41 显示了得到的部分结果。

程序清单 28-41 web.config 文件中加密的<authentication>部分

```

<?xml version="1.0"?>
<configuration>
  <connectionStrings>
    <add name="ExampleApplication"
      connectionString="server=ExampleApplicationServer;
        database=ExampleApplicationDB;uid=WebUser;pwd=P@$$word9"
      providerName="System.Data.SqlClient" />
  </connectionStrings>
  <system.web>
    <compilation debug="true" targetFramework="4.5"/>
    <httpRuntime targetFramework="4.5"/>
    <authentication
      configProtectionProvider="RsaProtectedConfigurationProvider">

```



```

<EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
  xmlns="http://www.w3.org/2001/04/xmlenc#">
  <EncryptionMethod
    Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
  <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
    <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
      <EncryptionMethod
        Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <KeyName>Rsa Key</KeyName>
      </KeyInfo>
      <CipherData>
        <CipherValue>P4mqGAlV4NmDSA+c3nARXx50QeeOyo6JrKFJ/
          DYYGFomQwq3tvErbaHQhffRr9S3UkeNcloFM/zg00xvdfq9X
          tkR/dOb0o8LOKAnlBwoJ9+sKAAdsd2U6tv+Of+k7h1Qwi3jM4
          guTiBAudpZXr9TnjouwN9KI3xmebagYTkR2NSPFoFMCH5RQb
          +iIRLeiGecVMpz17qz72acd3KWzabYHW2W+zbVA9lm2aATUb
          kDrdk3BahxzdOID62+svSDqzSgyibaAjc4WSNlnTRsNvPKLo
          aIUJiliTxn7APaunH8afdeqMmvOEGi3jVt733Pcz2rljMjUQ
          yPkxfowlOMyhGKfAg==</CipherValue>
      </CipherData>
    </EncryptedKey>
  </KeyInfo>
  <CipherData>
    <CipherValue>kqNlrAcWy2ZHK+cRSpWQUOX4OPfj12x3wqu
      f9cXNFU+TuleXF3EUwCb/SQCYgdJz</CipherValue>
  </CipherData>
</EncryptedData>
</authentication>
</system.web>
</configuration>

```

加密 web.config 文件中的这些部分后, 就需要解密这些部分, 返回它们原来未加密的值。为此, 可使用 aspnet_regiis 工具, 如程序清单 28-42 所示。

程序清单 28-42 解密 web.config 文件中的<connectionStrings>部分

```
aspnet_regiis -pd "connectionStrings" -app "/EncryptionExample"
```

运行这个脚本, 会把加密的值返回为原来的值。

28.2.16 编辑配置文件

本章介绍了配置文件以及每个配置条目的含义。配置条目采用简单、可读的 XML 格式, 但编辑它们是比较麻烦的。为了帮助编辑这些条目, 微软提供了如下 3 个工具:

- Visual Studio 2012 IDE
- Web Site Administration Tool
- Internet Information Services (IIS) Manager

Visual Studio 2012 IDE 的一项优秀功能是为配置文件提供了基于 IntelliSense 的编辑功能, 如图 28-6 所示。



图 28-6

Visual Studio 2012 IDE 还支持对 XML 元素进行语法检查，如图 28-7 所示。

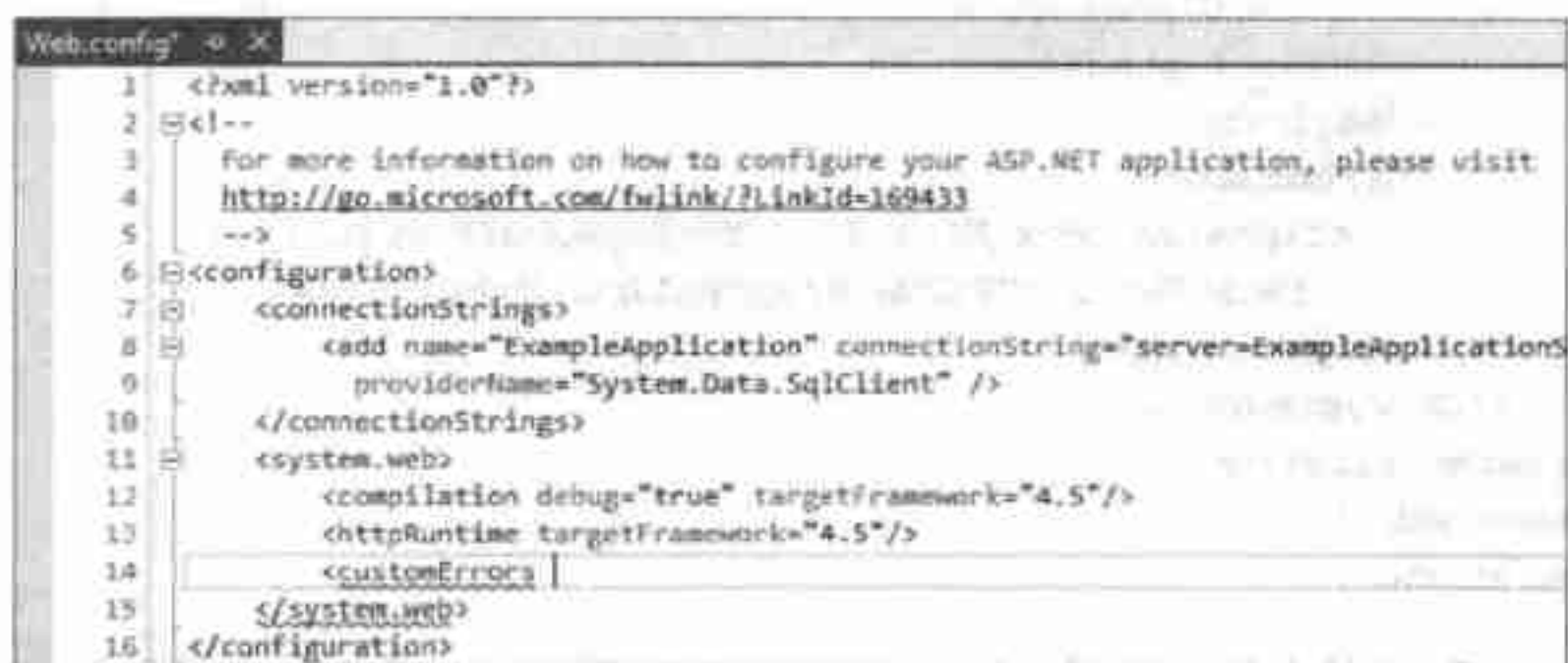


图 28-7



可以使用基于 XSD 的 XML 验证功能来获得 XML 元素的语法检查和 IntelliSense 支持，该功能可用于 Visual Studio 2012 中的所有 XML 文件。XSD 配置文件位于 `<drive>\Program Files\Microsoft Visual Studio 11.0\Xml\Schemas\DotNetconfig.xsd`。

Visual Studio 2012 IDE 还通过 XML 工具栏选项添加了两个有用的新功能，它们有助于格式化配置设置：

- Reformat selection: 这个选项重新格式化当前 XML 节点的内容。
- Format the whole document: 这个选项格式化整个 XML 文档。

Web Site Administration Tool 和 IIS Manager 可以在不知道 XML 元素名及其相应值的情况下编辑配置条目。附录 D 将详细介绍这些工具。

28.3 创建定制部分

除了使用前面讨论的 web.config 文件之外,还可以扩展该文件,给该文件添加自己的定制部分,并可以像使用其他部分一样使用定制部分。

创建定制部分的一种方式是使用某些内置的处理程序,从.config 文件中读取键/值对。下面 3 个处理程序都位于 System.Configuration 名称空间中:

- **NameValueFileSectionHandler**: 这个处理程序处理 web.config 文件的当前<appSettings>部分;使用这个处理程序可创建配置文件的新部分,新部分的行为方式与<appSettings>部分相同。
- **DictionarySectionHandler**: 这个处理程序处理键/值对的字典集合。
- **SingleTagSectionHandler**: 这个处理程序处理配置文件中的单个元素,允许把其中包含的键/值对读取为特性和值。

本章将介绍这些处理程序和定制配置文件的一些编程方式。

28.3.1 使用 NameValueFileSectionHandler 对象

如果打算创建行为类似于 web.config 文件的<appSettings>部分的定制部分,就可以使用这个对象。在 web.config 文件的<system.web>部分的上面,引用了 NameValueFileSectionHandler 对象,并且有 ASP.NET 4.5 应用程序中的其他默认引用。这个额外的引用如程序清单 28-43 所示。

程序清单 28-43 在 web.config 文件中创建键/值对的定制部分

```
<configSections>
  <section name="MyCompanyAppSettings"
    type="System.Configuration.NameValueFileSectionHandler, System,
    Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
    restartOnExternalChanges="false" />
</configSections>
```

引用 System.Configuration.NameValueFileSectionHandler 对象并为之指定名称(在本例中是 MyCompanyAppSettings)后,就可以在 web.config 文件中创建使用这个引用的部分,如程序清单 28-44 所示。

程序清单 28-44 在 web.config 文件中创建自己的定制键/值对部分

```
<configuration>
  <MyCompanyAppSettings>
    <add key="Key1" value="This is value 1" />
    <add key="Key2" value="This is value 2" />
  </MyCompanyAppSettings>
  <system.web>
    <!-- Removed for clarity -->
  </system.web>
</configuration>
```

在 web.config 文件中添加这段代码之后,就可以编程访问这个部分,如程序清单 28-45 所示。

程序清单 28-45 编程访问 web.config 文件中的定制部分

```

NameValueCollection nvc = new NameValueCollection();
nvc = ConfigurationManager.GetSection("MyCompanyAppSettings") as
    NameValueCollection;
Response.Write(nvc["Key1"] + "<br />");
Response.Write(nvc["Key2"]);

```

为了使其工作，需要在文件中导入 System.Collections.Specialized 名称空间，因为该名称空间包含 NameValueCollection 对象。

28.3.2 使用 DictionarySectionHandler 对象

DictionarySectionHandler 对象的工作方式与 NameValueFileSectionHandler 对象基本相同，区别是 DictionarySectionHandler 返回 Hashtable 对象，而不是返回 Object 对象。这个处理程序如程序清单 28-46 所示。

程序清单 28-46 引用 DictionarySectionHandler 对象

```

<configSections>
  <section name="MyCompanyAppSettings"
    type="System.Configuration.DictionarySectionHandler, System,
    Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
    restartOnExternalChanges="false" />
</configSections>

```

完成该配置设置之后，就可以在 web.config 文件中建立 MyCompanyAppSettings 部分，如程序清单 28-47 所示。

程序清单 28-47 在 web.config 文件中创建定制的键/值对部分

```

<configuration>
  <MyCompanyAppSettings>
    <add key="Key1" value="This is value 1" />
    <add key="Key2" value="This is value 2" />
  </MyCompanyAppSettings>
  <system.web>
    <!-- Removed for clarity -->
  </system.web>
</configuration>

```

准备好 web.config 文件之后，就可以在代码中使用 Configuration API 调用配置项，如程序清单 28-48 所示。

程序清单 28-48 编程访问 web.config 文件中的定制部分

```

NameValueCollection nv = new NameValueCollection();
nv = ConfigurationManager.GetSection("MyCompanyAppSettings") as NameValueCollection;
Response.Write(nv["Key1"] + "<br />");
Response.Write(nv["Key2"]);

```


28.3.3 使用 SingleTagSectionHandler 对象

SingleTagSectionHandler 对象的工作方式与前面介绍的 NameValueFileSectionHandler 和 DictionarySectionHandler 对象基本相同,但这个对象处理把键/值对保存为特性的单个元素。这个处理程序如程序清单 28-49 所示。

程序清单 28-49 引用 SingleTagSectionHandler 对象

```
<configSections>
  <section name="MyCompanyAppSettings"
    type="System.Configuration.SingleTagSectionHandler, System,
      Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
    restartOnExternalChanges="false" />
</configSections>
```

完成该配置设置之后,就可以在 web.config 文件中建立另一个 MyCompanyAppSettings 部分,如程序清单 28-50 所示。

程序清单 28-50 在 web.config 文件中创建定制的键/值对部分

```
<configuration>
  <MyCompanyAppSettings Key1="This is value 1" Key2="This is value 2" />
  <system.web>
    <!-- Removed for clarity -->
  </system.web>
</configuration>
```

web.config 文件完成之后,就可以在代码中使用 Configuration API 调用配置项,如程序清单 28-51 所示。

程序清单 28-51 编程访问 web.config 文件中的定制部分

```
Hashtable ht = new Hashtable();
ht = ConfigurationManager.GetSection("MyCompanyAppSettings") as Hashtable;
Response.Write(ht["Key1"] + "<br />");
Response.Write(ht["Key2"]);
```

28.3.4 使用定制的配置处理程序

也可以创建定制的配置处理程序。为此,首先需要创建类以表示 web.config 文件中的定制部分。在 App_Code 文件夹中创建类 MyCompanySettings,如程序清单 28-52 所示。

程序清单 28-52 MyCompanySettings 类

```
using System.Configuration;

public class MyCompanySettings : ConfigurationSection
{
  [ConfigurationProperty("Key1", DefaultValue = "This is the value of Key 1",
    IsRequired = false)]
  public string Key1
```

```

    {
        get
        {
            return this["Key1"] as string;
        }
    }

    [ConfigurationProperty("Key2", IsRequired = true)]
    public string Key2
    {
        get
        {
            return this["Key2"] as string;
        }
    }
}

```

可以看出，这个类继承于 `ConfigurationSection`，并且使用 `ConfigurationProperty` 特性创建两个属性。这里可以使用多个属性，如 `DefaultValue`、`IsRequired`、`IsKey` 和 `IsDefaultCollection`。有了这个类，就可以配置应用程序来使用这个处理程序，如程序清单 28-53 所示。

程序清单 28-53 引用 `MyCompanySettings` 对象

```

<configSections>
  <section name="MyCompanySettings" type="MyCompanySettings" />
</configSections>

```

现在可以在 `web.config` 文件中使用这个部分，如程序清单 28-54 所示。

程序清单 28-54 在 `web.config` 文件中创建定制的键/值对部分

```

<configuration>
  <configSections>
    <!-- Removed for clarity -->
  </configSections>
  <MyCompanySettings Key2="Here is a value for Key2" />
  <system.web>
    <!-- Removed for clarity -->
  </system.web>
</configuration>

```

使用这些配置可以在代码中编程访问这个键/值对部分，如程序清单 28-55 所示。

程序清单 28-55 编程访问 `web.config` 文件中的定制部分

```

MyCompanySettings cs = ConfigurationManager.GetSection("MyCompanySettings") as
    MyCompanySettings;
Response.Write(cs.Key1 + "<br />");
Response.Write(cs.Key2);

```

28.4 使用配置转换

前面了解了 XML 配置系统有多强大。但我们还可能希望知道需要改变多少设置，或者在部署时修改 config 文件的手工过程。过去，除了这两个手工过程之外，还有几个解决方法，但 ASP.NET 团队意识到需要改变一些东西。

在 ASP.NET 4 和 Visual Studio 2010 中，引入了 web.config 转换。ASP.NET 4.5 继续使用这个操作。转换可以用于任何配置元素或特性，包括本章介绍的定制 config 部分。



配置转换可以用于 Web 应用程序，但不能用于 Web 站点。

28.4.1 添加 web.config 转换

web.config 转换用于 ASP.NET Web 应用程序。在创建新的 ASP.NET Web 应用程序后，就可以扩展 web.config 文件，使其包含 web.Debug.config 和 web.Release.config 文件。这两个文件都是相关配置模式的 config 转换文件。每个 ASP.NET Web 应用程序都从 Debug 和 Release 配置模式开始，所以这些文件是自动创建的，如图 28-8 所示。

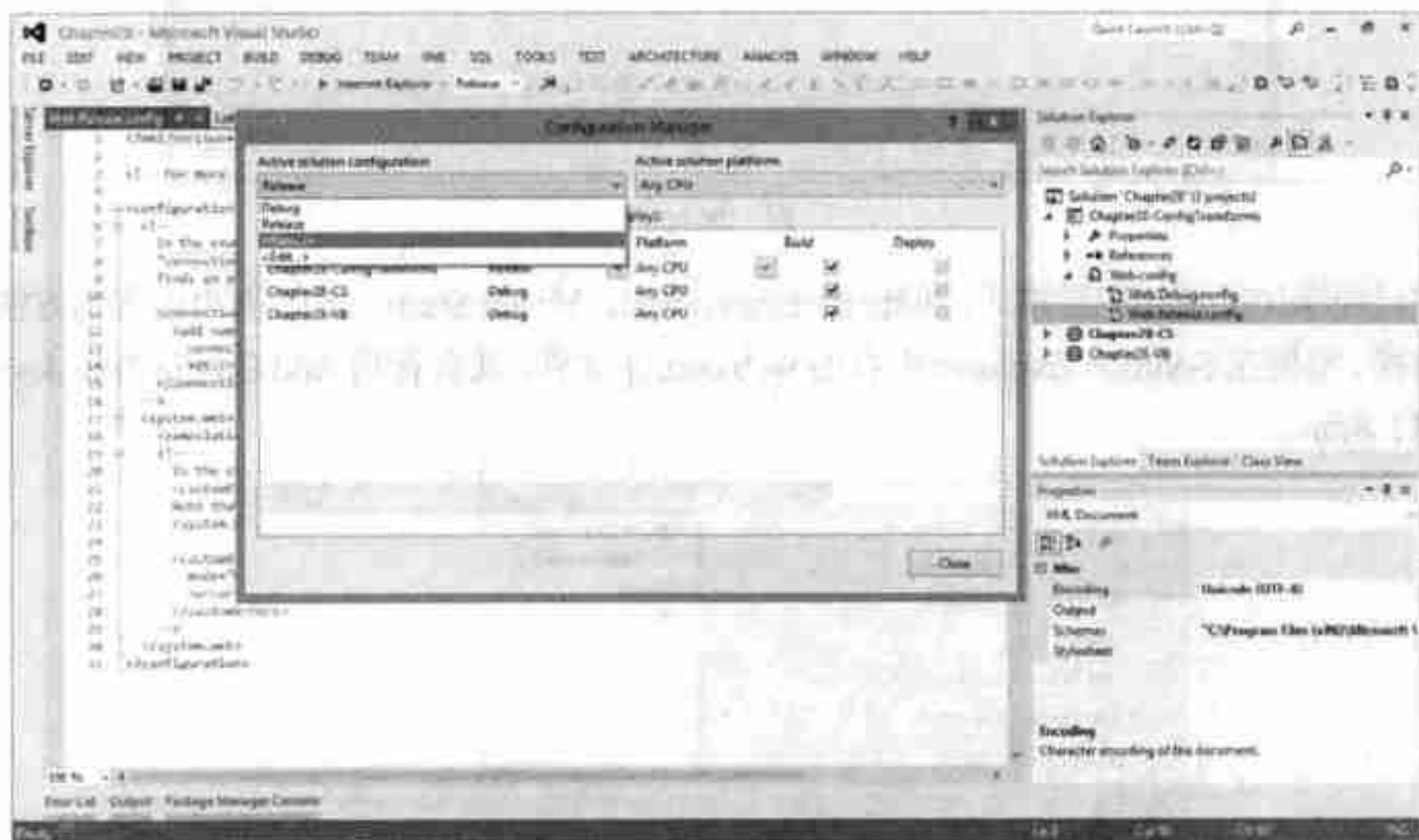


图 28-8

在图 28-8 中，有两个配置模式。在 Solution Explorer 中有另外两个 web.config 文件，如前所述。

但是，假定有另一个定制的配置过程，对应于 Visual Studio 2012 中新的 Publish 功能。这个配置过程包括部署到 Staging 服务器。要添加新的配置模式 Staging，可以在 Configuration Manager 窗口中选择 New 选项，如图 28-8 所示。接着输入新的配置模式名称，复制另一个构建配置中的设置，如图 28-9 所示。

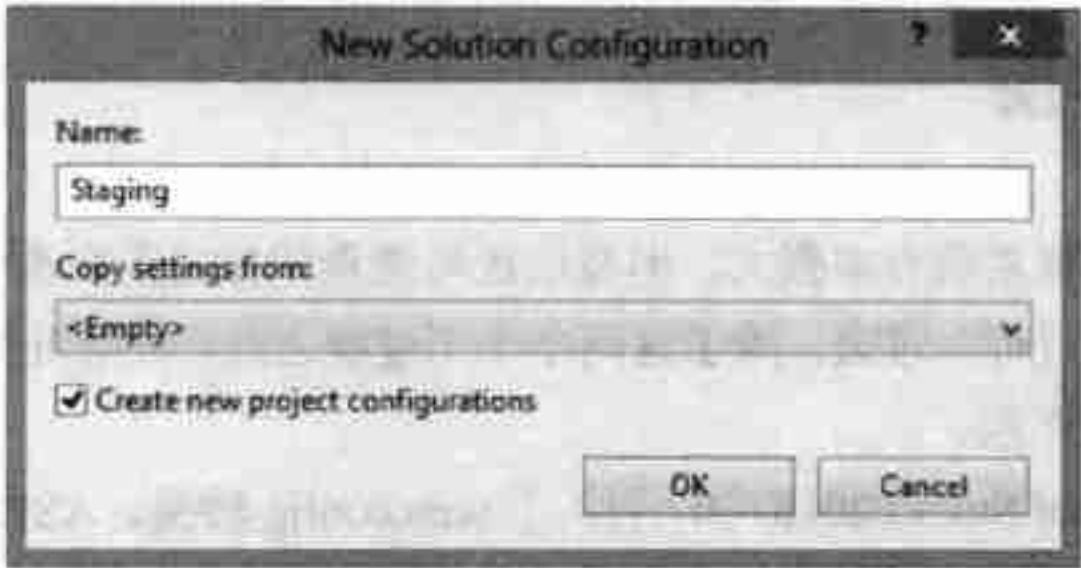


图 28-9

添加了新的配置模式后，新模式就可以在 Configuration Manager 窗口中使用，如图 28-10 所示。

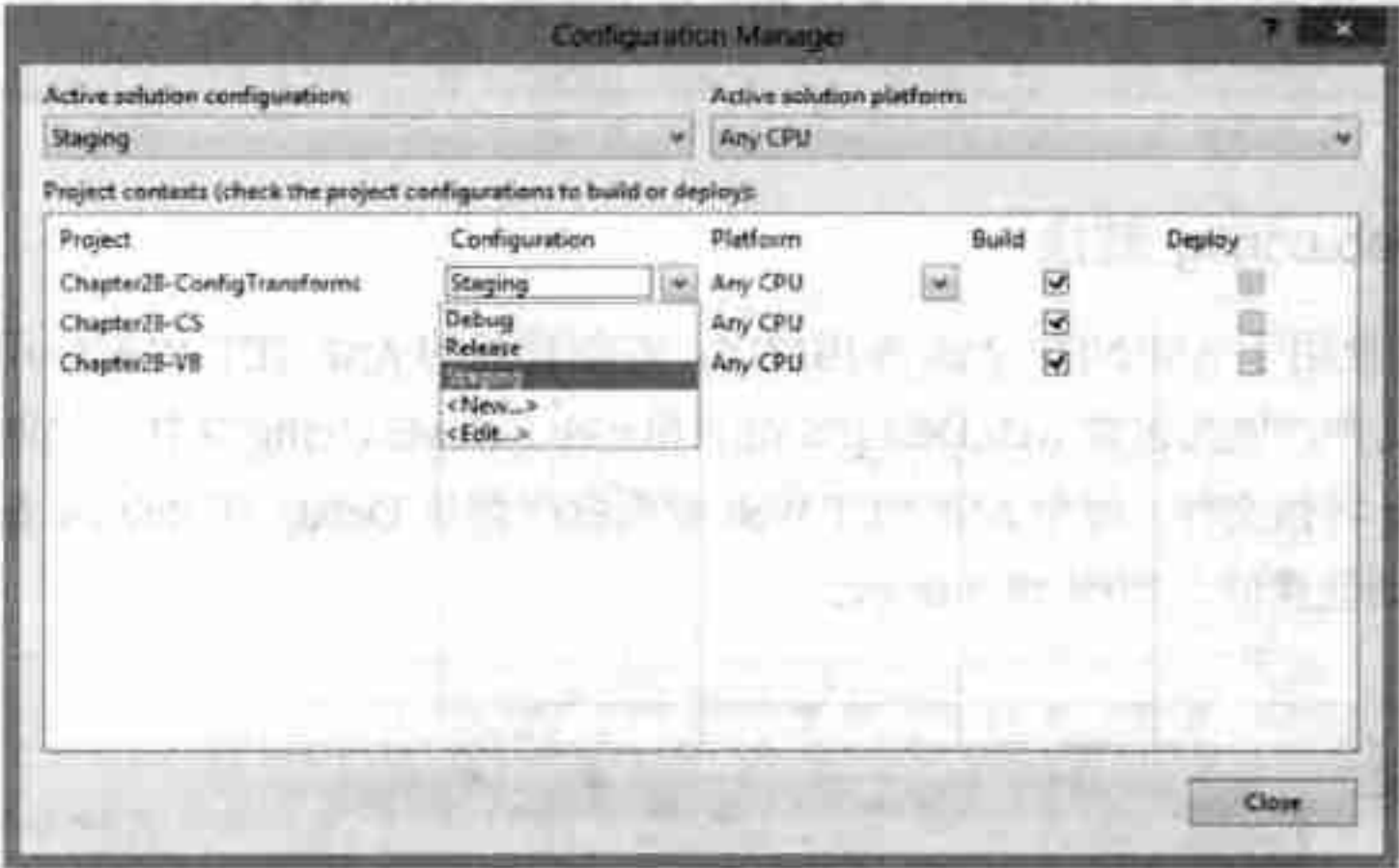


图 28-10

现在已经添加了新的配置模式，但转换还没有添加。Visual Studio 2012 有个内置的方法，有助于添加转换。如果在 Solution Explorer 中右击 web.config 文件，就会看到 Add Config Transform 选项，如图 28-11 所示。

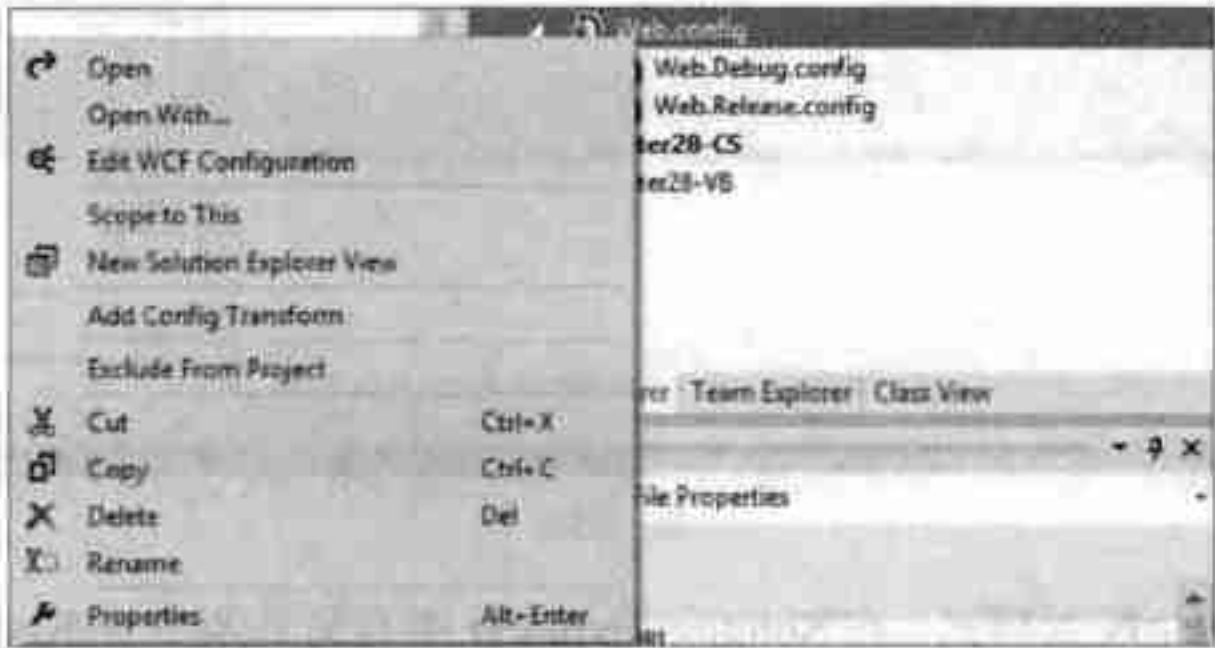


图 28-11

添加了配置转换后，就可以在 Solution Explorer 中看到对应的文件，如图 28-12 所示。

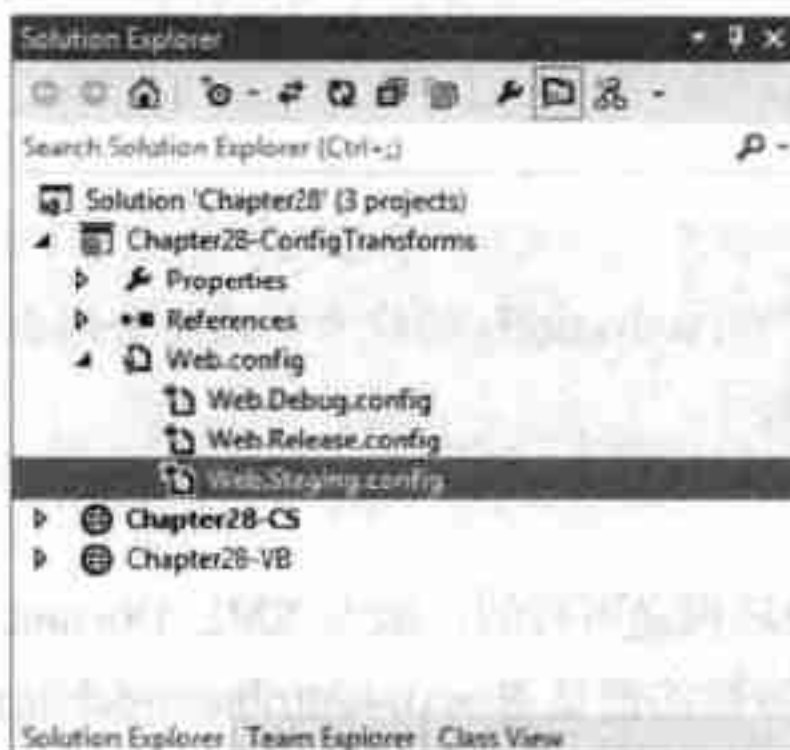


图 28-12

28.4.2 更新配置转换文件

打开配置转换文件后，例如 `web.Staging.config` 文件，就会发现该文件是空的。该文件包含示例转换和用于删除 debug 特性的活动转换，如程序清单 28-56 所示。

程序清单 28-56 浏览 `web.Staging.config` 文件

```
<?xml version="1.0" encoding="utf-8"?>

<!-- For more information on using web.config transformation visit
      http://go.microsoft.com/fwlink/?LinkId=125889 -->

<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-Transform">
  <!--
  In the example below, the "SetAttributes" transform will change the value of
  "connectionString" to use "ReleaseSQLServer" only when the "Match" locator
  finds an attribute "name" that has a value of "MyDB".

  <connectionStrings>
    <add name="MyDB"
        connectionString="Data Source=ReleaseSQLServer;Initial Catalog=MyReleaseDB;
        Integrated Security=True"
        xdt:Transform="SetAttributes" xdt:Locator="Match(name)"/>
  </connectionStrings>
  -->

  <system.web>
    <compilation xdt:Transform="RemoveAttributes(debug)" />
    <!--
    In the example below, the "Replace" transform will replace the entire
    <customErrors> section of your web.config file.
    Note that because there is only one customErrors section under the
    <system.web> node, there is no need to use the "xdt:Locator" attribute.

    <customErrors defaultRedirect="GenericError.htm"
        mode="RemoteOnly" xdt:Transform="Replace">
      <error statusCode="500" redirect="InternalError.htm"/>
    </customErrors>
  -->
```

```
</system.web>
</configuration>
```



正如文件的注释所示, web.config 转换命令的完整列表位于 <http://go.microsoft.com/fwlink/?LinkId=125889>.

编译配置元素包含转换。该转换是可行的, 因为 XML Document Transform 名称空间已添加到配置根元素中。注意 transform 特性的值是 RemoveAttributes(debug)。RemoveAttributes 名是要使用的转换方法。此时, 这个方法从配置元素中删除特性。传递给该方法的值是 debug。这表示, debug 特性会从配置中删除。

XML Document Transform 名称空间有两个特性。第一个是 transform, 在编译配置元素的转换中演示。transform 特性的可能方法是:

- Insert: 把定义好的元素插入为选中元素的子元素
- InsertAfter: 把元素插入到指定路径的后面(XPath 表达式)
- InsertBefore: 把元素插入到指定路径的前面(XPath 表达式)
- Remove: 删除选中的元素
- RemoveAll: 删除所有选中的元素
- RemoveAttributes: 删除当前元素的指定特性
- Replace: 用转换文件中指定的元素替代整个元素
- SetAttributes: 给当前元素的指定特性设置指定的值

第二个特性 location 用于选择特定的元素。location 特性的可能方法是:

- Condition: 一个 XPath 表达式, 在当前元素上计算, 以确定何时进行转换
- Match: 指定一个或多个有匹配值的元素
- XPath: XPath 表达式, 可以导航到另一个元素, 再计算条件以确定是否进行转换

使用 location 和 transform 特性可能是必要的。例如, 本章前面把定制配置部分 MyCompanyAppSettings 创建为 NameValueFileSectionHandler。转换配置文件时, 可能只需要替换集合中某一项的值。例如, 把下面的代码添加到 MyCompanyAppSettings 配置部分:

```
<add key="Key2" value="The is value 2 in staging"
  xdt:Transform="SetAttributes(value)" xdt:Locator="Match(key)" />
```

使用 Staging 配置模式发布到 Staging 服务器上之后, 应注意 web.config 文件如程序清单 28-57 所示。



第 33 章将详细介绍各种发布方法。

程序清单 28-57 配置转换的结果

```
<?xml version="1.0"?>
```



```

<configuration>
  <configSections>
    <section name="MyCompanyAppSettings"
      type="System.Configuration.NameValueFileSectionHandler, System,
        Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
      restartOnExternalChanges="false" />
  </configSections>
  <MyCompanyAppSettings>
    <add key="Key1" value="This is value 1" />
    <add key="Key2" value="The is value 2 in staging" />
  </MyCompanyAppSettings>
  <system.web>
    <compilation targetFramework="4.5" />
    <httpRuntime targetFramework="4.5" />
  </system.web>
</configuration>

```

该实践方式常常用于在发布过程中替换特定连接的 `connectionString` 值。



使用本章没有介绍的配置转换时，有许多选项。建议继续在线搜索配置转换的更多例子。也可以在 MSDN 中搜索 Web.config Transformation Syntax。

28.5 打包和压缩功能

ASP.NET 4.5 包含的一项新功能，就是可以在 Web 应用程序中压缩、打包 CSS 和 JavaScript。如果在 Visual Studio 2012 中使用任意默认的项目模板，这项新功能就是打开的。

28.5.1 什么是打包和压缩功能

大多数静态的内容文件，例如 CSS 和 JavaScript，都包含许多空白和注释。尽管空白和注释不一定占据大量的空间，但根据客户的网速，这些额外的空间会显著减慢站点页面的加载。图 28-13 显示了压缩前的网络流量。



下面是压缩前的 JavaScript 片段：

```
function helloWorld(firstName) {
    // Declare variables
    var message = "Hello, " + firstName;

    // Display an alert message
    alert(message);
}
```

过去，压缩静态内容的最佳技术之一是启用 Web 服务器(如 IIS)上的 HTTP 压缩功能。最常见的压缩方法是对静态文件使用 gzip。但该压缩过程的问题是，很难定制要压缩的文件。在大型组织中，可能需要一位管理员更新 Web 服务器。在 ASP.NET 中，现在可以完全控制压缩过程，而不再需要什么管理员了。

为了使压缩过程更进一步，还可以打包一组文件，例如把所有 CSS 文件打包到某个压缩路径下。

28.5.2 启用打包和压缩功能

脚本和样式的打包、压缩有许多方式。处理打包的一种方式是在项目中添加 bundle.config 文件，并在该文件中添加适当的文件列表。也可以在应用程序启动方法中手动将文件添加到包中。

为了给项目启用打包和压缩功能，应从包管理器中下载 Microsoft ASP.NET Web Optimization Framework 包。下载并启用后，进入 Global.asax 文件，该文件包含名称空间 System.Web.Optimization。最后，为了打包和压缩 Web 中 styles 文件夹下的 CSS 文件，应给 Application_Start 方法添加如下代码：

```
var cssBundle = new Bundle("~/styles/css");
cssBundle.IncludeDirectory("~/styles", "*.css");
BundleTable.Bundles.Add(cssBundle);
```

与 ASP.NET 中的其他事务一样，开发人员也可以完全控制 CSS 和 JavaScript 的打包方式。这包括创建定制转换，包含和去除特定的文件，以及创建多个包。注意，文件压缩和添加的顺序与它们列出的顺序相同。如果脚本文件依赖另一个文件，最好把依赖文件放在一个包中。

图 28-14 显示了压缩后的网络流量。



图 28-14

前面的 JavaScript 片段压缩后变成：

```
function helloWorld(n){var t="Hello, "+n;alert(t)}
```



要了解定制打包的更多信息，可以在线搜索 ASP.NET 4.5 Bundling and Minification。还要注意，自从这个功能在 NuGet(Visual Studio 中的包管理器)上发布后，其更新内容可能在 Visual Studio 的下一个版本发布之前发布。

28.6 本章小结

本章介绍了 ASP.NET 配置系统，讨论 ASP.NET 如何不依赖于 IIS 元数据库，而使用可读的 XML 配置系统。

本章还介绍了两个不同的 ASP.NET XML 配置文件：

- machine.config
- web.config

machine.config 文件把默认设置应用于服务器上的所有 Web 应用程序。但是，如果服务器安装了 .NET Framework 的多个版本，machine.config 文件就应用于某个特定的 .NET Framework 版本。而单个 Web 应用程序可以使用 web.config 文件定制或重写自己的配置信息。使用 web.config 文件还可以逐个配置应用程序或文件夹。

之后，本章讨论了一些可应用于 ASP.NET 应用程序的常见配置设置，例如配置连接字符串、会话状态、浏览器功能等。然后概述了如何使用加密算法保护配置部分，还探讨了在部署时如何替换配置值。最后学习了如何压缩脚本和 CSS 文件，把这些文件放在一个包中。

第 29 章

调试和错误处理技术

本章要点

- 跟踪 ASP.NET 应用程序
- 探究 ASP.NET 调试选项
- 有效地处理异常和错误
- 用 Page Inspector 进行调试

代码总是按照编写它们的方式运行，但我们不可能在第一次就使代码正常运行。仅调试程序就占整个程序所花费时间的 30%，因此我们应学习如何高效地使用可用的调试工具。Visual Studio 提供了大量功能，极大地改善了调试功能，但这么多功能最初会让人不知所措。本章将分别详细介绍这些技术，并全面探讨 Visual Studio、公共语言运行时(Common Language Runtime, CLR)和基类库(Base Class Library, BCL)。

另外，调试不仅仅是单步执行代码，本章将讨论高效的错误和异常处理、跟踪和日志记录以及跨语言(C#、Visual Basic、客户端 JavaScript、XSLT 和 SQL 存储过程)调试。

29.1 设计期间的支持

Visual Studio 总是能在设计期间警告潜在的错误。在编译项目之前，语法通知或波浪下划线将突出显示不能通过编译或可能导致错误的代码。在调试会话中出现异常时，会弹出新的错误通知，并建议一系列防止异常的操作。在每一步中，Visual Studio 都尝试更智能化，预测用户的需求，捕获常见的错误。

Rico Mariani 是 Visual Studio 的首席架构师，他使用术语“*The Pit of Success*”(成功的陷阱)来描述微软希望用户使用 Visual Studio 的情形。在微软设计这些功能时，希望用户能方便地使用这些成功的经验。为此，该公司试图让用户不太容易编写出错误的代码或犯常见的错误。微软公司的开发人员花费很多心思来构建 API，通过它们指引我们正确的方向。

29.1.1 语法通知

Visual Basic 和 C#编辑器都在编译前为许多语法错误显示了下划线和工具提示,如图 29-1 所示。



图 29-1

语法通知不仅用于 CLR 编程语言; Visual Studio 也大大改进了 XML 编辑器,如下所示:

- 完整的 XML 1.0 语法检查
- 支持 DTD 和 XSD 有效性验证和 IntelliSense
- 支持 XSLT 1.0 语法检查

图 29-2 显示了一个详细的工具提示,表示<junk>元素在 web.config 文件中没有任何作用。编辑器知道这个问题,因为 XML 编辑器支持 XSD 有效性验证,还为配置文件(如 web.config)添加了模式。这个改动受到了很多人的欢迎,这些人在手动编辑 web.config 文件时都要确定是否使用了正确的元素。

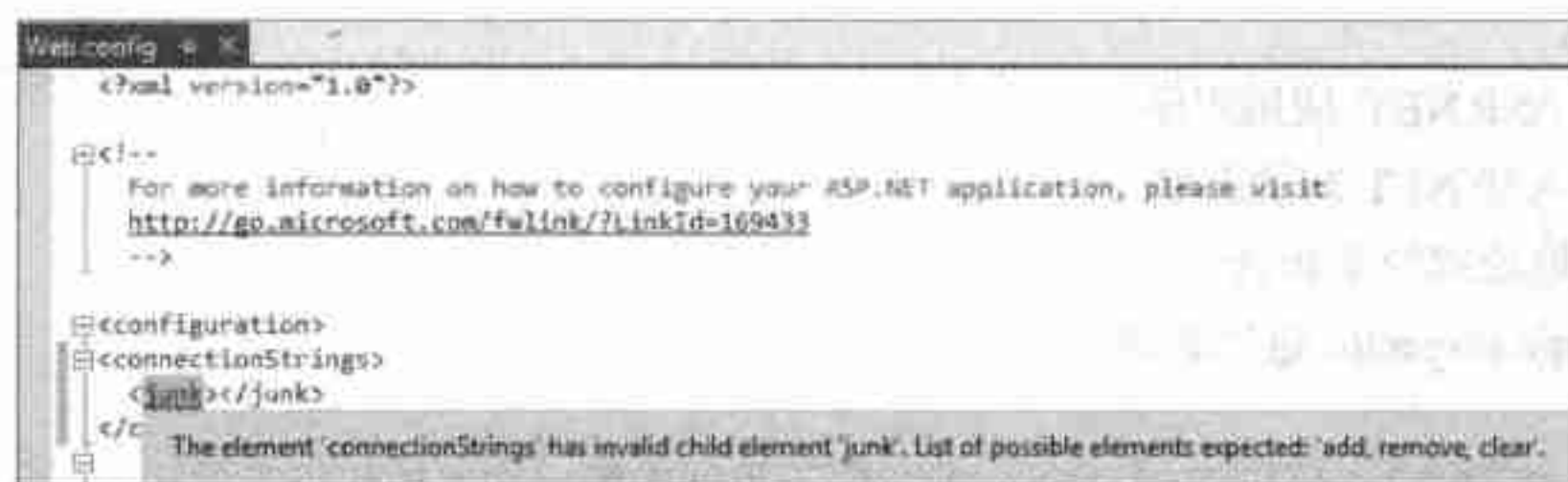


图 29-2

ASPX/HTML 编辑器也从这些功能中获益。例如,图 29-3 显示了一个警告: <badElement/>元素在活动的模式中不可用。在 ASP.NET 页面中,对<script runat="server">块中的代码也进行了分析,并使用下划线标记。这就使页面中的代码更容易理解。还要注意,图 29-3 中的 ASP.NET 页面在第一行有 XHTML DOCTYPE 声明,并且 HTML 元素有默认的 XHTML 名称空间。这个 HTML 页面被看做 XML,因为它们面向 XHTML。

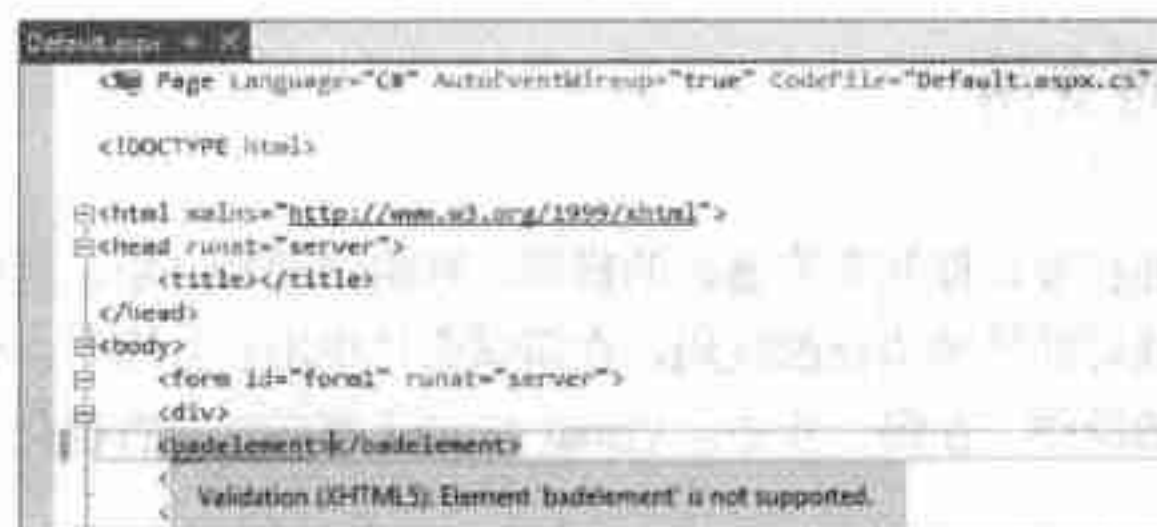


图 29-3

Visual Basic 编辑器使用智能标记提供了进一步的帮助,例如在把光标停留在下划线上时显示的折叠式按钮。单击方框中的代码,会显示非常漂亮的无模式窗口,其中包含一些使代码通过编译的建议。图 29-4 建议应插入遗漏的 End If。进行更正是很简单的,只需单击 Insert the missing 'End If' 即可。

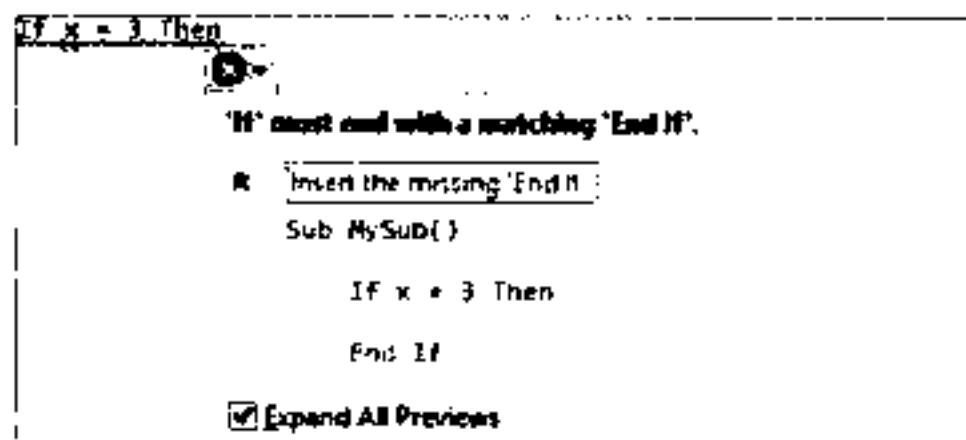


图 29-4

所有这些设计期间的功能都有助于确保在编译和运行代码之前编写出更好的代码。两个相关的功能——Immediate 和 Command 窗口有助于在开发环境中运行任意代码，同时使任务仍然可以完成。

29.1.2 Immediate 和 Command 窗口

Immediate 窗口允许在设计模式下运行任意代码，但不编译应用程序。可以在设计期间或调试时评估代码，还可以快速测试一行代码或静态方法。这个窗口的 Immediate 模式主要用于调试。

选择 Debug | Windows | Immediate 命令就可以访问 Immediate 窗口。要计算变量或运行方法，只需单击 Immediate 窗口，输入问号(?)后跟要计算的表达式、变量或方法。

在命令的前面添加大于号(>)，就可以从 Immediate 窗口切换到 Command 窗口。在 Immediate 或 Command 窗口中输入大于号，就会弹出 IntelliSense，显示完整的 Visual Studio 对象模型和记录的宏。这个窗口的 Command 模式用于执行 Visual Studio 命令，并且无须使用菜单。也可以执行没有菜单项的命令。

如果在 Command 窗口中输入>alias，就会得到当前所有别名及其定义的完整列表。一些有用的命令别名如下：

- >Log filename /overwrite /on|off: Log 命令把 Command 窗口中的所有结果记录到文件中，如果没有指定用于记录的文件名，就使用 cmdline.log。这是调试器的一项比较有用、但用得最少的功能，原因是需要学习 Immediate/Command 窗口的一些内容。
- >Shell args /command /output /dir:folder: Shell 命令可以从 Visual Studio 的 Command 窗口中运行可执行程序，如实用程序、命令脚本、批处理文件等。

29.1.3 任务列表

Visual Studio 中的任务列表要比我们想象的更有用，对其关注不够的人就会错过一项优秀的功能。任务列表支持两个视图：User Tasks 和 Comments。

User Tasks 视图允许添加和修改任务，这些任务可以是“记住要测试”或“买牛奶”。这些任务存储在与.sln 文件类似的.suo(解决方案用户选项)文件中。

Comments 视图显示代码中添加了特定标记前缀的注释文本。Visual Studio 已配置为查找 TODO: 标记，也可以选择 Tools | Options | Environment | Task List 命令以添加自己的标记。

在图 29-5 中，在 Options 对话框中添加了注释标记 HACK。带 HACK:前缀的注释显示在源代码中，因此该注释行会自动显示在 Visual Studio 底部窗口的 Task List 选项卡中。图 29-5 中的 3 个圆表示添加到 Options 对话框中的单词 HACK，与在源代码和 Task List 选项卡中后续出现的 HACK 之间的连接。可以添加任意多个这样的标记。

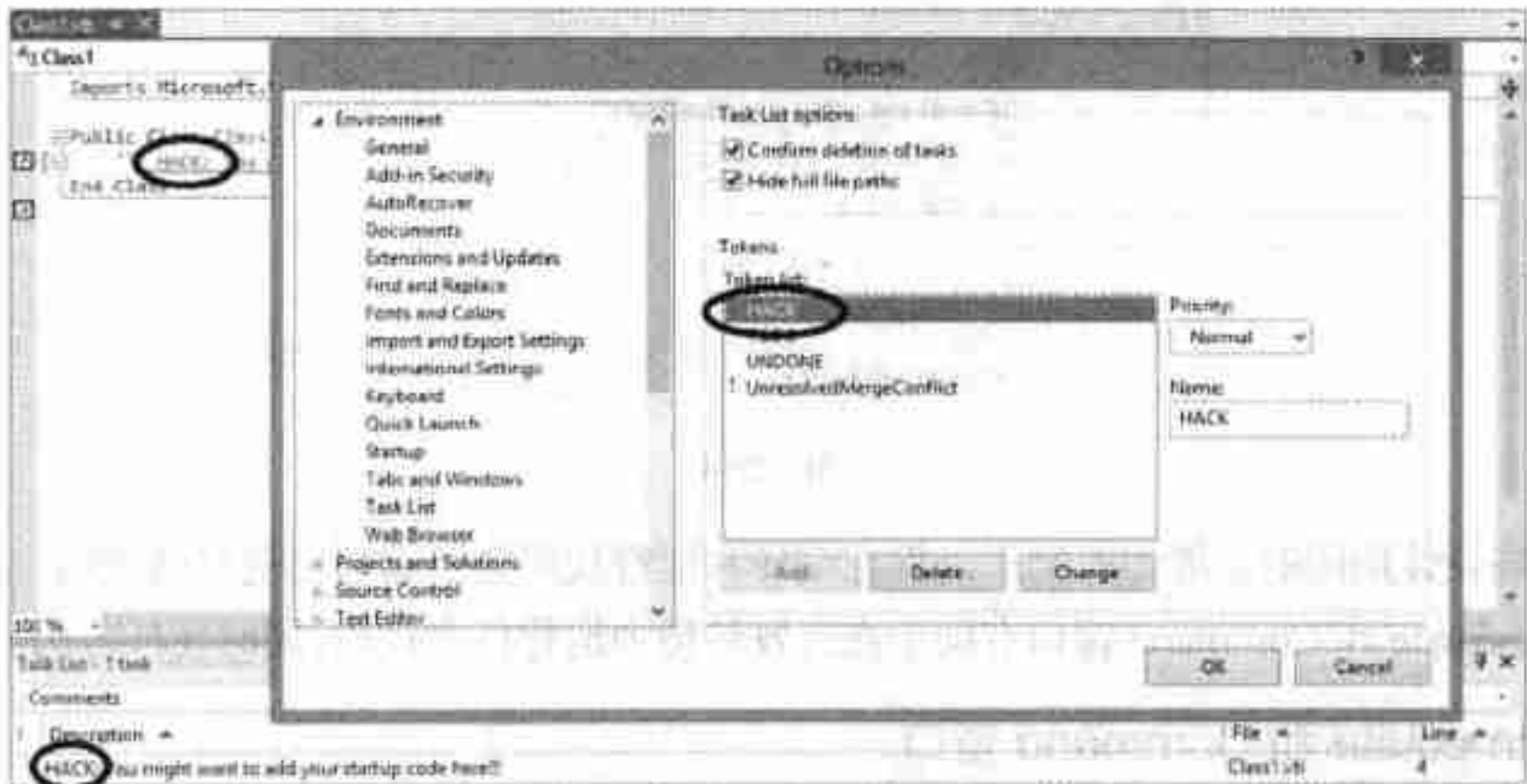


图 29-5

29.2 跟踪

跟踪是监控应用程序执行的一种方式。可以以不影响程序输出的方式记录异常的细节和程序流。在传统的 ASP 中，跟踪和调试功能几乎是不存在的，这就迫使开发人员只能以许多 `Response.Write` 语句的形式使用“到达此处”调试。这种调试方式使用不正规的跟踪语句打乱了生成的 HTML 标记，在执行每行新代码时都会通知程序员——程序已执行到“这个地方”或“那个地方”。这种扰乱程序执行的跟踪方式非常不便于清理，许多程序员最终只有创建自己的非正式跟踪库来解决这些传统 ASP 的问题。

ASP.NET 中为跟踪功能提供了很大的支持。可以使用 `TraceListener` (如 `EventLogTraceListener`) 来配置跟踪输出的目的地。`TraceListener` 的配置详见本节后面的内容。ASP.NET 允许在 ASP.NET 指定页面的 `Trace` 类和由非 Web 开发人员使用的标准基类库 (Basic Class Library, BCL) 的 `System.Diagnostics.Trace` 类之间进行跟踪传送。另外，ASP.NET 跟踪功能对实时输出的分解也提高到了 18 位数，从而获得非常精确的配置信息。

29.2.1 System.Diagnostics.Trace 和 ASP.NET 的 Page.Trace

在 .NET Framework 中，以 `Trace` 命名的对象有好几个，因此跟踪功能在 Web 和非 Web 应用程序中的含义可能并不统一。有个名为 `System.Diagnostics.Trace` 的类，而 `System.Web.UI.Page` 类有公有属性 `Trace`，不要混淆它们。`Page` 类的 `Trace` 属性允许访问 `System.Web.TraceContext` 类和 ASP.NET 专用的跟踪机制。`TraceContext` 类会收集 Web 请求的所有信息和时间选择。该类包含许多方法，其中最常用的是 `Write` 方法。此外还包含调用 `Write` 方法的 `Warn` 方法，并确保 `Warn` 方法生成的输出显示为红色。

如果编写的 ASP.NET 应用程序不支持在非 Web 环境下使用的组件或程序集，那么通常仅使用 ASP.NET 的 `TraceContext` 类就可以获得许多实用程序。但是，ASP.NET 支持的跟踪功能不同于基类库的跟踪功能。下面首先探讨 ASP.NET 的跟踪功能，再介绍如何把它们关联起来，并讨论一些更便于调试的功能。

29.2.2 页面级的跟踪

在 ASP.NET 页面上给 Page 指令添加 Trace="true", 就可以在每个页面上启用 ASP.NET 的跟踪功能:

```
<%@ Page Language="C#" Inherits="System.Web.UI.Page" Trace="true" %>
```

另外, 还可以添加 TraceMode 特性, 可以将它设置为 SortByCategory 或默认的 SortByTime。可以给每个子系统包含类别并使用 SortByCategory 分组它们, 或者使用 SortByTime 查看应用程序中占用 CPU 资源最多的方法。也可以使用 Trace.IsEnabled 属性编程激活跟踪功能。编程激活跟踪功能表示, 可以通过查询字符串、cookie 或 IP 地址激活跟踪功能。

29.2.3 应用程序的跟踪

另外, 还可以通过在 web.config 文件中添加跟踪设置, 为整个应用程序激活跟踪功能。在下面的例子中, 使用了 pageOutput="false" 和 requestLimit="20", 因此存储了 20 个请求的跟踪信息, 但不显示在页面上:

```
<configuration>
  <system.web>
    <trace enabled="true" pageOutput="false" requestLimit="20"
      traceMode="SortByTime" localOnly="true" />
  </system.web>
</configuration>
```

页面级设置优先于 web.config 文件中的设置, 因此如果在 web.config 文件中设置了 enabled="false", 但在页面上设置了 Trace="true", 就会启用跟踪功能。

29.2.4 查看跟踪数据

通过请求特定的页面 trace.axd, 就可以在应用程序级别查看多个页面请求的跟踪信息。注意, trace.axd 实际上并不存在, 而由 System.Web.Handlers.TraceHandler 提供, System.Web.Handlers.TraceHandler 是绑定了 trace.axd 的特殊 IHttpHandler。当 ASP.NET 检测到 HTTP 请求的 trace.axd 时, 请求就由 TraceHandler 处理, 而不是由页面处理。

创建一个 Web 站点和一个页面, 并在 Page_Load 事件中调用 Trace.Write 方法。在 web.config 文件中激活跟踪功能, 如程序清单 29-1 所示。

程序清单 29-1 使用 Page.Trace 进行跟踪

```
<configuration>
  <system.web>
    <trace enabled="true" pageOutput="true" />
  </system.web>
</configuration>

protected void Page_Load(object sender, EventArgs e)
{
  Trace.Write("This message is from the START of the Page_Load method!");
}
```

在浏览器中打开该页面几次, 注意尽管这个页面没有创建任何 HTML, 但却在浏览器中显示了

大量的跟踪信息，如图 29-6 所示，因为设置是 `pageOutput="true"`。

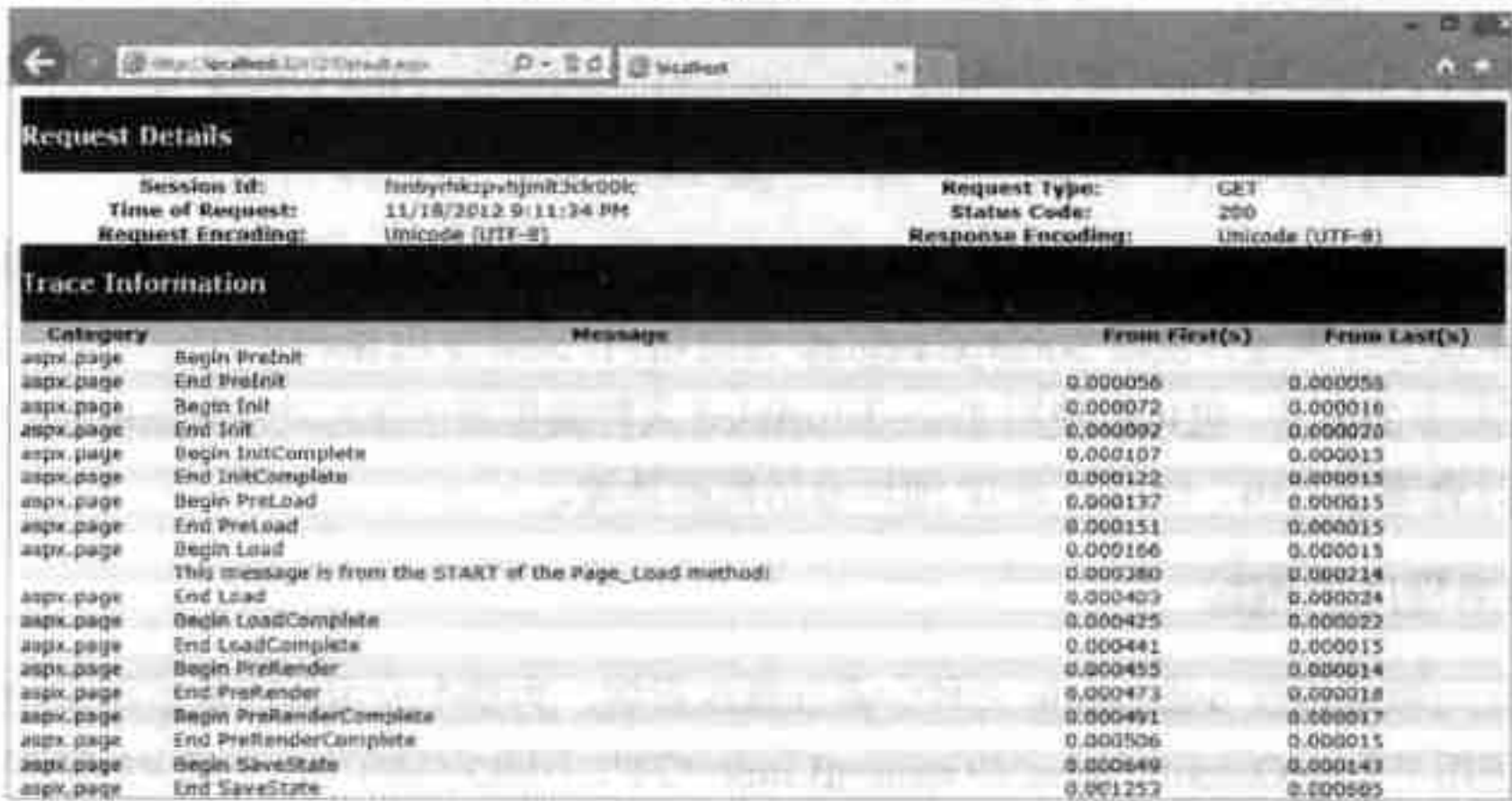


图 29-6

Trace.Write 输出的消息显示在 Begin Load 之后和 End Load 之前，正好位于 Page_Load 方法的中间。页面在运行时会自动进行 JIT 编译，其最初的性能会有所下降。现在已编译为本机代码，以后再运行这个页面时，单击浏览器上的 Refresh 按钮，执行就只需 0.000167 秒，因为页面已经通过编译。在 Trace 语句之间收集这类非常有价值的性能计时数据非常方便，也极为有用。

跟踪信息的 11 个不同的部分提供了大量的细节，尤其详细描述了 ASP.NET 页面的显示过程，如表 29-1 所示。

表 29-1

部 分	说 明
Request Details	包含 ASP.NET 会话 ID、请求和响应的字符编码以及 HTTP 对话的返回状态码。注意请求和响应的编码，尤其是在使用非拉丁字符集时。如果返回的语言不是英语，就要把编码设置为 UTF-8，这是默认值
Trace Information	包含在 HTTP 请求生命周期内调用的所有 Trace.Write 方法和许多有关计时的信息。这是对调试最有用的部分。在配置和搜索应用程序中需要执行很长时间的方法时，这里的计时信息非常有价值
Control Tree	显示 ASP.NET 控件树的 HTML 表示。该部分会显示每个控件的唯一 ID、运行时的类型、显示所需的字节数，以及在 ViewState 和 ControlState 中需要的字节数。不要低估这两部分的作用，尤其是显示每个控件权值的 3 个列。控件的权值表示该控件在 ViewState 和 ControlState 中占用的字节数。注意每个控件使用的字节数，尤其是在编写自己的定制控件时，要使控件返回尽可能少的字节数，使整个页面的权值较低
Session State	列出某个用户会话的所有键、它们的类型和值。只显示当前用户的会话状态
Application State	列出当前应用程序的 Application 对象中的所有键、它们的类型和值
Request Cookies	列出在页面请求过程中传入的所有 cookie
Response Cookies	列出在页面响应过程中传回的所有 cookie

(续表)

部 分	说 明
Headers Collection	在浏览器上显示请求的过程中可能传送的所有报头, 包括 Accept-Encoding(表示浏览器是否支持压缩的 HTTP 响应)、Accept-Language(这是一组 ISO 语言代码, 表示用户的语言首选项顺序)以及 User-Agent(用户浏览器的标识字符串)。该字符串还包含用户的操作系统信息以及在 IE 中运行的 .NET Framework 版本
Form Collection	显示完整的 Form 集合及其所有的键和值
QueryString Collection	显示完整的 QueryString 集合及其包含的所有键和值
Server Variables	Web 服务器了解的应用程序和请求浏览器的所有名/值对

跟踪的页面输出仅显示为当前页面请求收集的数据。但在访问 `http://localhost/your-site/trace.axd` 时, 可以看到为站点的所有请求收集的详细数据。如果使用内置的 ASP.NET 开发服务器, 就从 URL 中删除当前页面, 使用 `trace.axd` 替代它。不要修改自动选中的端口或路径。

`trace.axd` 是内部处理程序, 而不是真正的页面。在从本地浏览器请求它时, 如图 29-7 所示, 浏览器会显示预定期限内所有请求的所有跟踪信息。

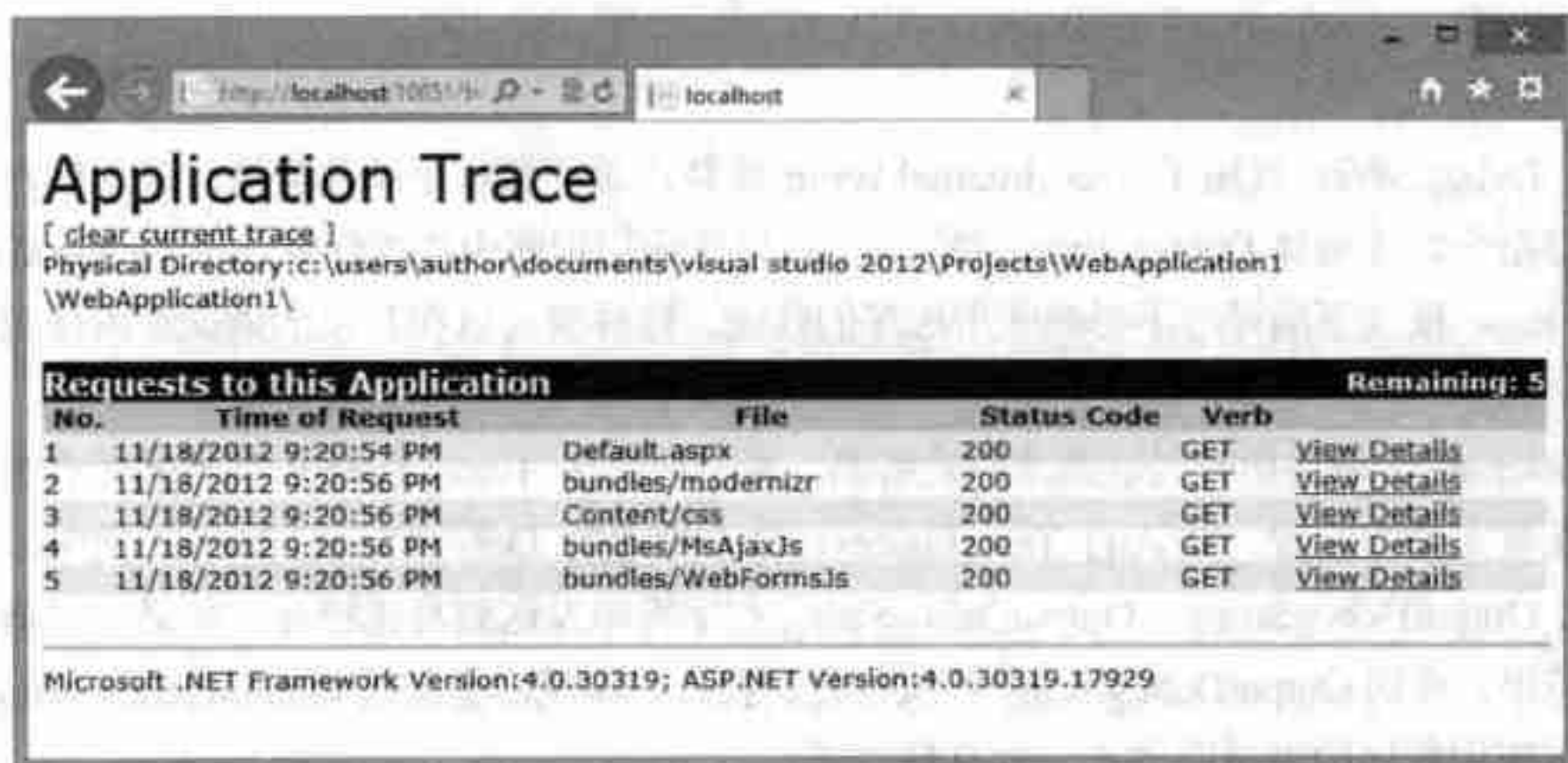


图 29-7

图 29-7 显示了给这个应用程序发出的 5 个请求, 在标题的右边显示 `Remaining: 5`, 表示在停止对这个应用程序的跟踪之前还剩余 5 个请求。在这个请求完成之后, 就会在应用程序再次使用或单击 `trace.axd` 页面上的 `clear current trace` 链接后保存跟踪数据。`RequestLimit` 特性的值可以在 `web.config` 文件中增加, 但要以占用内存为代价:

```
<trace requestLimit="100" pageOutput="true" enabled="true"/>
```

`requestLimit` 特性的最大值是 10 000。如果尝试使用更大的值, ASP.NET 将总是使用 10 000, 并且不给出错误消息。但是, 可以在 ASP.NET 的跟踪部分添加特性 `mostRecent`, 将该特性设置为 `true` 时, 会显示存储在跟踪日志中的大部分最新请求, 而不是按照请求的顺序显示它们(这是默认设置), 并且不会占用许多内存。把 `mostRecent` 特性设置为 `true`, 会使内存只用于存储的跟踪信息, 并自动删除请求数超过 `requestLimit` 时的跟踪信息。

单击 trace.axd 页面上任意请求的 View Details 链接, 就会进入一个专门用于请求的页面, 其中的详细信息如图 29-6 所示。

29.2.5 在组件中跟踪

ASP.NET 的跟踪功能非常强大, 并且是独立的功能。本章前面提到的 System.Diagnostics.Trace 是基类库(BCL)中的跟踪架构, 不是专门用于 Web 的, 在 ASP.NET 应用程序调用不支持 Web 的组件时, 会接收一致而完整的跟踪信息。这很容易产生混淆, 应该使用哪项功能?

System.Diagnostics.Trace 是 .NET Framework 的核心跟踪库, 与 System.Diagnostics.Debug 类一起为所有的应用程序提供了灵活的、非入侵式的跟踪功能和调试结果。但如前所述, System.Web 名称空间内置了丰富的跟踪功能。Web 开发人员可以使用 ASP.NET 的跟踪功能, 有时也需要使专门用于 ASP.NET 的跟踪功能面向基本架构的 System.Diagnostics.Trace。更常见的情况是, 我们希望使不支持 Web 的组件将跟踪调用输出到 ASP.NET 中, 以便利用 trace.axd 和其他 ASP.NET 专用功能。

另外, Trace.Write 和 Debug.Write 函数也会有一些混淆。下面是 Debug.Write 函数的源代码:

```
[Conditional("DEBUG")]
public static void Write(string message)
{
    TraceInternal.Write(message);
}
```

注意, Debug.Write 调用了 TraceInternal.Write 函数, 该函数有个条件特性, 表示仅在设置了调试预处理器指令时才编译 Debug.Write。换言之, 可以在应用程序中放置任意多个 Debug.Write 调用, 并且在 Release 模式下编译时不影响应用程序的性能。这样就可以在开发的调试阶段使应用程序尽可能详尽。

TraceInternal 会遍历所有关联的跟踪监听器, 表示派生于 TraceListener 基类的所有类都在该应用程序的配置文件中配置。默认的 TraceListener 位于相应的 DefaultTraceListener 类中, 并且调用 Win32 API OutputDebugString。OutputDebugString 把字符串发送到高速缓存, 如果调试器在监听, 就显示字符串, 否则 OutputDebugString 不执行任何操作。调试器会监听 OutputDebugString 的输出, 因此是监听应用程序的调试版本的一种高效方式。



为了快速、方便地调试, 可使用 SysInternals 的 DebugView(<http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx>)。DebugView 不需要安装, 可以处理所有对 Debug.Writer 的调用, 并且有许多优秀功能, 如突出显示和记录到文件中。

现在, 查看 Trace.Write 的源代码(此处是 TRACE, 而不是 DEBUG), 如下所示:

```
[Conditional("TRACE")]
public static void Write(string message)
{
    TraceInternal.Write(message);
}
```

在这两段源代码中, Debug.Write 和 Trace.Write 之间的唯一区别是表示预处理器指令 TRACE 的

条件特性。可以对程序集进行有条件的编译，以包含跟踪语句、调试语句，或者不包含这两种语句。许多人甚至在发布版本中都保留了跟踪语句，并使用配置文件来启用或禁用跟踪功能。更重要的是，从为用户激活跟踪功能中获得的收益远远大于它们所带来的性能问题。

与 `Debug.Write` 一样，`Trace.Write` 也调用 `DefaultTraceListener`，因此可以使用任意调试器来接收跟踪信息。那么，它们有什么区别？

在设计应用程序时，要考虑部署模型。是要调试版本还是发布版本？想要给终端用户或系统工程师提供一种方式，从而能让他们使用日志文件或事件查看器调试应用程序吗？是否含有只能由开发人员查看的内容？

一般要使用跟踪功能和 `Trace.Write` 来处理在产品环境下对调试应用程序很有用的正式信息。`Trace.Write` 提供了 `Debug.Write` 的所有功能，只是使用预处理器指令 `TRACE`，并且不受调试或发布版本的影响。

也就是说，建立应用程序时有 4 种选择：`Debug On`、`Trace On`、`Both On` 和 `Neither On`。应选择适合自己的选项。一般情况下，应给调试版本使用 `Both On` 选项，给产品版本使用 `Trace On` 选项。可以在属性页面或编译器的命令行中指定这些条件属性，以及在 C# 中使用 `#define` 关键字或在 Visual Basic 中使用 `#CONST` 关键字指定这些条件特性。

29.2.6 跟踪的传送

已有的 ASP.NET 应用程序常常都经过了很好的测试，并且大量使用了 ASP.NET 的 `TraceContext` 类。ASP.NET 给 `web.config` 文件的 `<trace>` 元素引入了特性 `writeToDiagnosticsTrace`，它可以把 ASP.NET 跟踪功能发出的消息路由到 `System.Diagnostics.Trace`。

```
<trace writeToDiagnosticsTrace="true" pageOutput="true" enabled="true"/>
```

把 `writeToDiagnosticsTrace` 特性设置为 `true` 时，所有对 `System.Web.UI.Page.Trace.Write` (ASP.NET `TraceContext`) 的调用也会启动 `System.Diagnostics.Trace.Write`，允许使用所有的标准 `TraceListener` 和跟踪选项(参见本章后面的内容)。简单的 `writeToDiagnosticsTrace` 设置把 ASP.NET 的跟踪功能与基类库的其他内容连接起来。在调试页面时可以使用这个功能，该功能使用配置开关也很容易关闭。虽然信息多一些总比少好一些，但页面事件的信息实在太多。读者可以试一试，再决定如何操作。

29.2.7 TraceListener

`System.Diagnostics.Trace` 方法的输出可以由 `TraceListener` 路由到文本文件、ASP.NET、外部监控系统甚至数据库中。这个强大的功能未能由许多 ASP.NET 开发人员充分利用。在早期的 ASP.NET 中，一些组件开发人员知道组件要在 ASP.NET 中使用，因此直接引用 `System.Web`，再调用 `HttpContext.Current.Trace`。他们这么做是为了使跟踪信息以对开发人员友好的 ASP.NET 格式显示出来。在 `HttpRequest` 环境中调用的所有组件都自动接收对请求的当前环境的访问权限，允许组件直接与请求通信，以提取 cookie 或收集用户的信息。

但是，假定 `HttpContext` 总是可用是非常危险的，这有许多原因。首先，在声明组件时要假定，该组件只能在 `HttpRequest` 环境中使用。注意是在 `HttpRequest` 环境中，而不是在应用程序环境中。如果在 `Application_Start` 事件中访问 `HttpContext.Current`，就会发现 `HttpContext.Current` 是空的。其次，如果可以在非 Web 环境下使用应用程序，那么把组件的功能与 `HttpContext` 结合起来是很困难

的，并且单元测试尤其困难。

如果有个组件要由 Web 页面使用，并且需要在 Web 环境的外部对它进行单元测试，或者必须在其他环境下调用它，就不要使用 `HttpContext.Current.Trace`，而应使用标准的 `System.Diagnostics.Trace`，并使用稍后将介绍的 `WebPageTraceListener`，把输出重定向到 ASP.NET 跟踪功能。使用标准的跟踪机制，意味着可以在任何环境下使用组件，包括 Web 环境和非 Web 环境。使用 `TraceListener` 还可以查看组件的跟踪输出。

该架构提供了许多非常有用的 `TraceListener`，可以编程添加它们或通过 `config` 文件添加它们。例如，可以把 `TraceListener` 日志编程添加到文件中，如程序清单 29-2 所示。这些代码片段需要 `System.Diagnostics` 和 `System.IO` 名称空间。

程序清单 29-2 配置 `TraceListener`

```
TextWriterTraceListener myTextListener = new
    TextWriterTraceListener(File.Create(@"c:\myListener.log"));
Trace.Listeners.Add(myTextListener);
```

可以在 `web.config` 文件中通过 `add` 元素(作为要使用的 `TraceListener` 类型传送)和所需的初始化数据，以声明的方式完成这项工作。还可以使用 `remove` 标记及其名称删除已经在 `machine.config` 或父 `web.config` 文件中配置的 `TraceListener`：

```
<configuration>
  <system.diagnostics>
    <trace autoflush="false" indentsize="4">
      <listeners>
        <add name="myListener"
            type="System.Diagnostics.TextWriterTraceListener"
            initializeData="c:\myListener.log" />
        <remove name="Default" />
      </listeners>
    </trace>
  </system.diagnostics>
</configuration>
```

`TraceListener`(如 `TextWriterTraceListener`)在访问资源(如文件、事件日志或数据库)时，要求 ASP.NET 工作者进程作为有足够访问权限的用户运行。例如，为了写入 `c:\foo\example.log`，ASP.NET 工作者进程需要在该文件的访问控制列表(Access Control List, ACL)中有显式的写入访问权限。

注意，上面的例子还可以有选择地删除默认的 `TraceListener`。如果编写自己的 `TraceListener`，就必须在 `type` 特性中提供完全限定的程序集名称。

1. 使用 ASP.NET `WebPageTraceListener`

ASP.NET 中的 `WebPageTraceListener` 派生于 `System.Diagnostics.TraceListener`，可以自动把跟踪信息从组件调用传送给 `System.Diagnostics.Trace.Write`。这就允许使用最一般的跟踪提供程序编写组件，并在 ASP.NET 应用程序中查看跟踪结果。

正如下面的例子所示，把 `WebPageTraceListener` 添加到 `web.config` 文件中。注意，代码使用了 `System.Web` 的完全限定的程序集名称：


```

<configuration>
  <system.diagnostics>
    <trace autoflush="false" indentsize="4">
      <listeners>
        <add name="webListener"
              type="System.Web.WebPageTraceListener, System.Web, Version=4.0.0.0,
                  Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"/>
      </listeners>
    </trace>
  </system.diagnostics>
  <system.web>
    <trace enabled="true" pageOutput="false" localOnly="true" />
  </system.web>
</configuration>

```

图 29-8 显示了在引用的库中调用 System.Diagnostics.Trace.Write 的结果,该结果显示在 ASP.NET 的页面跟踪功能中。从引用库中生成的结果已在图 29-8 中圈了出来。

Request Details

Request Details

Session Id: 2e23kduidhjhmtn5qlcxyuph
Time of Request: 11/18/2012 9:26:13 PM
Request Encoding: Unicode (UTF-8)

Request Type: GET
Status Code: 200
Response Encoding: Unicode (UTF-8)

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	0.022162	0.022162
aspx.page	Begin Init	0.022192	0.000030
aspx.page	End Init	0.026110	0.003918
aspx.page	Begin InitComplete	0.026129	0.000019
aspx.page	End InitComplete	0.026497	0.000368
aspx.page	Begin PreLoad	0.026516	0.000019
aspx.page	End PreLoad	0.027077	0.000560
aspx.page	Begin Load	0.027098	0.000022
aspx.page	This message is from a non interesting class in a helpful library	0.027219	0.000121
aspx.page	End Load	0.027326	0.000117
aspx.page	Begin LoadComplete	0.027355	0.000019
aspx.page	End LoadComplete	0.027375	0.000020
aspx.page	Begin PreRender	0.027390	0.000015
aspx.page	End PreRender	0.028829	0.001439
aspx.page	Begin PreRenderComplete	0.028852	0.000024
aspx.page	End PreRenderComplete	0.874785	0.845933
aspx.runtime	Begin SaveState	0.875756	0.000971

图 29-8

2. EventLogTraceListener

也可以使用 EventLogTraceListener 将跟踪信息发送给事件日志。该操作有点困难,因为 ASP.NET 需要对事件日志的显式写入权限:

```

<configuration>
  <system.diagnostics>
    <trace autoflush="false" indentsize="4">
      <listeners>
        <add name="EventLogTraceListener"
              type="System.Diagnostics.EventLogTraceListener"
              initializeData="Wrox"/>
      </listeners>
    </trace>
  </system.diagnostics>
</configuration>

```



```
</system.diagnostics>
</configuration>
```

注意，在添加 `TraceListener` 时，“Wrox”作为字符串传送给 `initializeData` 特性。字符串“Wrox”显示为该事件的应用或来源。在调试应用程序时，这是有效的行为，因为调试用户常常有合适的访问权限。但是，在部署应用程序时，就可能在权限不足的账户下运行，因此必须给注册表键指定明确的写入访问权限，如 `HKLM\System\CurrentControlSet\Services\EventLog\Application\Wrox`，其中“Wrox”就是传送到 `initializeData` 特性中的字符串。注意，该注册表键与文件一样都有 ACL。使用 `RegEdit.exe` 可以修改注册表键上的权限，具体方法是右击该键，选择 `Properties` 命令，设置 ACL，就像处理文件一样。

在使用 `EventLogTraceListener` 时要小心，因为如果应用程序比较冗长，事件日志很快就会填满。图 29-9 显示了事件日志中的一个跟踪结果示例。

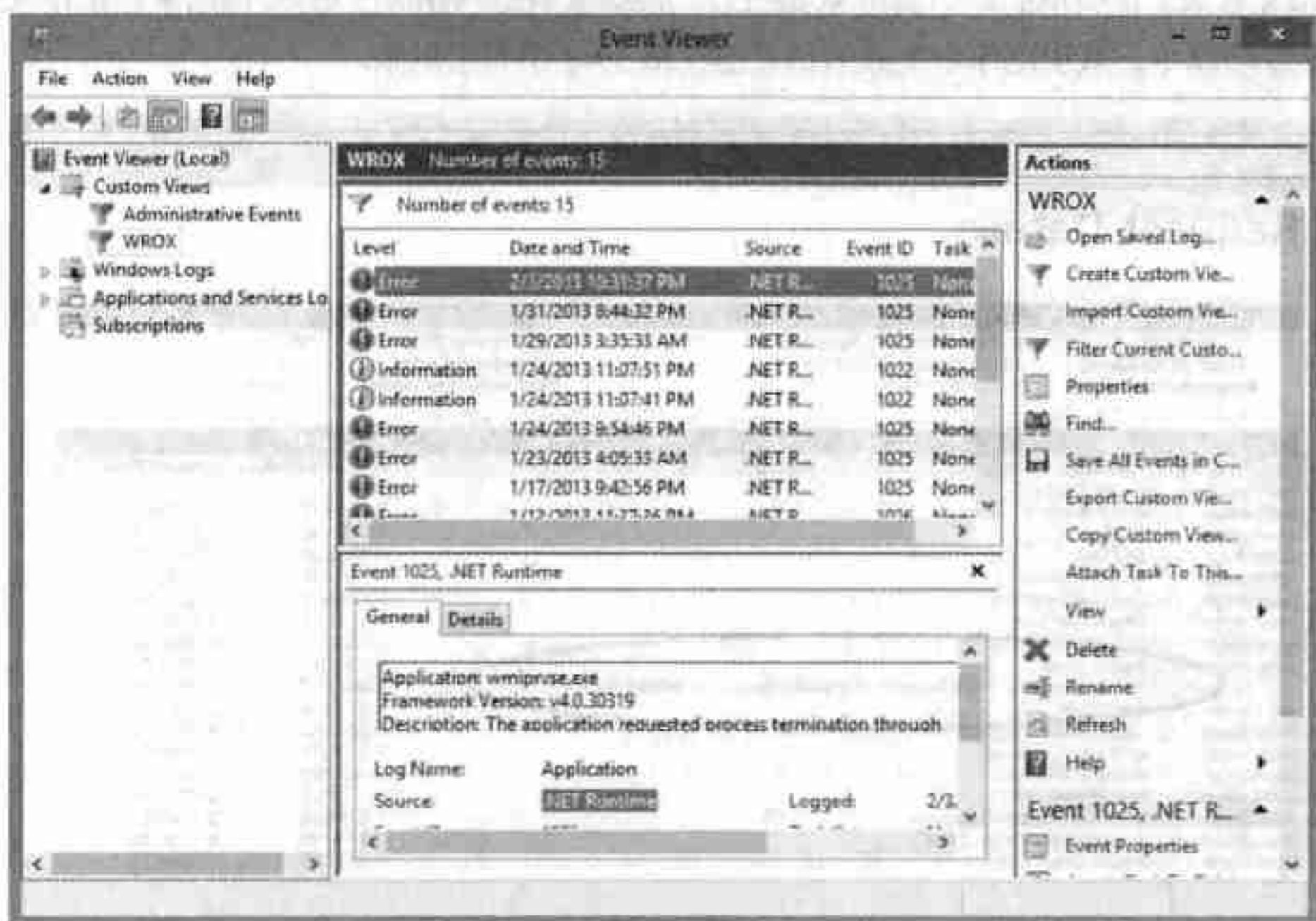


图 29-9

3. 其他有用的 `TraceListener`

除了 `WebPageTraceListener` 之外，.NET Framework 4.5 还添加了另外两个 `TraceListener`：

- `XmlWriterTraceListener`：派生自 `TextWriterTraceListener`，写出强类型化的 XML 文件。
- `DelimitedListTraceListener`：也派生自 `TextWriterTraceListener`，写出使用逗号分隔值(CSV)的文件。

对于 `XmlWriterTraceListener` 创建的 XML，一个有趣的地方是，它不是格式正确的 XML。具体来说，它没有根节点，只是同级节点的集合。这似乎违背了前面有关 XML 的理念，但可以把每个事件看作文档。每个事件都是独立的，可以独立使用。它们只是碰巧放在同一个文件中而已。显然，没有最后的结束标记是为了避免格式正确方面的问题，并且可以方便地在文件的最后添加该标记。

```

<E2ETraceEvent xmlns="http://schemas.microsoft.com/2012/06/E2ETraceEvent">
  <System xmlns="http://schemas.microsoft.com/2012/06/windows/eventlog/system">
    <EventID>0</EventID>
    <Type>3</Type>
    <SubType Name="Information">0</SubType>
    <Level>8</Level>
    <TimeCreated SystemTime="2012-11-05T12:43:44.4234234Z">
    <Source Name="WroxChapter29.exe"/>
    <Correlation ActivityID="{00000000-0000-0000-0000-000000000000}>
    <Execution ProcessName="WroxChapter29.exe" ProcessID="4234" ThreadID="1"/>
    <Channel/>
    <Computer>SCOTTPC</Computer>
  </System>
  <ApplicationData>Your Text Here</ApplicationData>
</E2ETraceEvent>
<E2ETraceEvent xmlns="http://schemas.microsoft.com/2012/06/E2ETraceEvent">
  <System xmlns="http://schemas.microsoft.com/2012/06/windows/eventlog/system">
    <EventID>0</EventID>
    <Type>3</Type>
... the XML continues ...

```

E2ETraceEvent 中的 E2E 表示端对端。注意，其中包含计算机名和关联 id 等信息。

自 .NET Framework 3.5 以来，该列表中添加了另一个 TraceListener，即 IisTraceListener。类似于 WebPageTraceListener 在诊断跟踪功能和 ASP.NET 跟踪功能之间搭起了桥梁，IisTraceListener 在 ASP.NET 的跟踪机制和 IIS 7.0 之间搭起了桥梁。这个监听器可以引发 IIS 7.0 基础结构上的事件。

29.2.8 诊断选项

仅因为要修改跟踪功能就重新编译应用程序常常并不方便。有时要修改配置文件，以添加或删除 TraceListener。有时则要修改配置参数或“按下一个开关”，以调整跟踪功能产生的信息量。此时就应使用 Switch 类。Switch 是抽象基类，支持一系列诊断选项，使用应用程序的配置文件可以控制这些诊断选项。

1. BooleanSwitch

要使用 BooleanSwitch，可以创建一个 Switch 实例，并传送选项名，选项名显示在应用程序的 config 文件中，如程序清单 29-3 所示。

程序清单 29-3 使用诊断选项

```

<configuration>
  <system.diagnostics>
    <switches>
      <add name="ImportantSwitch" value="1" /> <!-- This is for the BooleanSwitch -->
      <add name="LevelSwitch" value="3" /> <!-- This is for the TraceSwitch -->
      <add name="SourceSwitch" value="4" /> <!-- This is for the SourceSwitch -->
    </switches>
  </system.diagnostics>
</configuration>

```

Switch 可以在 if 语句中用于任意目的，但它们在跟踪和 System.Diagnostics.Trace.WriteLine 环境中

最有用：

```
BooleanSwitch aSwitch = new BooleanSwitch("ImportantSwitch", "Show errors");
System.Diagnostics.Trace.WriteIf(aSwitch.Enabled, "The Switch is enabled!");
```

如果在 config 文件中把 ImportantSwitch 设置为 1 或非零值，对 WriteIf 的调用就会发送一个字符串以跟踪输出。

2. TraceSwitch

TraceSwitch 提供了 5 个跟踪级别：0 到 4，分别表示 Off、Error、Warning、Info 和 Verbose 的递增顺序。构建 TraceSwitch 与构建 BooleanSwitch 一样：

```
TraceSwitch tSwitch = new TraceSwitch("LevelSwitch", "Trace Levels");
System.Diagnostics.Trace.WriteIf(tSwitch.TraceInfo, "The Switch is 3 or more!");
```

如果选项在同一级别或者比该属性值的级别高，那么 TraceSwitch 类的许多属性都将返回 true。例如，如果选项的值设置为 3 或更高，TraceInfo 属性就返回 true。

3. SourceSwitch

自从 .NET Framework 2.0 以来，就可以使用对象 SourceSwitch，虽然类似于 TraceSwitch，但提供了更高级别的精确度。可以把 EventType 作为参数来调用 SourceSwitch.ShouldTrace：

```
SourceSwitch sSwitch = new SourceSwitch("SourceSwitch", "Even More Levels");
System.Diagnostics.Trace.WriteIf(sSwitch.ShouldTrace(TraceEventType.Warning),
    "The Switch is 4 or more!");
```

29.2.9 Web 事件

尽管 Web 事件不像调试那样限定太多，但是在 ASP.NET 中，却给 System.Web.Management 名称空间添加了一些新的应用程序监控和运行状况监控工具。这些工具像跟踪信息一样有价值，有助于监控、维护和诊断应用程序的运行状况。系统有事件模型和事件引擎，可以使用运行时的信息更新应用程序。还有许多内置事件，包括应用程序生命周期的事件，如开始、停止、检测信号事件。可以利用这些基类和事件，在它们的基础上创建自己的事件。例如，可以创建事件，说明用户何时下载某个大型文件或何时在个性化数据库中创建新用户；也可以根据统计信息，每天给自己发送一封电子邮件。

例如，可以通过从 System.Web.Management.WebBaseEvent 派生来创建自己的事件，如程序清单 29-4 所示。

程序清单 29-4 Web 事件

```
Imports System
Imports System.Web.Management

namespace Wrox
{
    using System;
```



```

using System.Web.Management;

public class WroxEvent: WebBaseEvent
{
    public const int WroxEventCode = WebEventCodes.WebExtendedBase + 1;
    public WroxEvent(string message, object eventSource) :
        base(message, eventSource, WroxEventCode){}
}

```

然后，在 Page_Load 事件中，给管理子系统触发该事件：

```

protected void Page_Load(Object sender, EventArgs e)
{
    // Raise a custom event
    Wrox.WroxEvent anEvent = new Wrox.WroxEvent("Someone visited here!", this);
    anEvent.Raise();
}

```

这个事件被管理子系统捕获，并可以根据许多规则分派给不同的提供程序。这是比调用 `Trace.WriteLine` 更正式的跟踪功能，因此可以为应用程序专用的事件创建强类型化的事件类：

web.config 文件

```

<?xml version="1.0"?>
<configuration>
  <system.web>
    <healthMonitoring enabled="true">
      <providers>
        <add name="WroxDatabaseLoggingProvider"
              type="System.Web.Management.SqlWebEventProvider"
              connectionStringName="QuickStartSqlServer"
              maxEventDetailsLength="1073741823"
              buffer="false"/>
      </providers>
      <rules>
        <add
          name="Application Lifetime Events Rule"
          eventName="All Events"
          provider="WroxDatabaseLoggingProvider"
          profile="Critical" />
        </rules>
      </healthMonitoring>
    </system.web>
  </configuration>

```

29.3 调试

Visual Studio 默认包含两种配置：调试和发布。调试配置自动定义调试和跟踪常量，允许应用

程序为故障解决人员提供调试环境。生成调试信息的选项在默认时是启用的，可以为每个程序集和解决方案生成程序数据库(或调试)文件(PDB)，它们与程序集位于相同的 bin 文件夹下。但是，本机代码的编译不在 Visual Studio 中进行，而是在运行时使用实时(JIT)编译器编译。JIT 编译器会自动优化代码，以提高速度。然而，优化的代码更难调试，因为生成的操作不直接对应于源代码中的行。为了便于调试，这个选项应设置为 false。

29.3.1 需要的内容

在使用/debug:full 命令行选项调用 C#编译器(csc.exe)或 Visual Basic 编译器(vbc.exe)时，就会创建 PDB。另外，如果使用/debug:pdbonly，那么也会生成 PDB，但仍然是由编译器生成发布模式的代码。

1. 调试和发布

Visual Studio 中的调试和发布配置一般可以满足需要。但是，这些配置只控制隐藏代码文件的编译选项。注意，根据设计 ASP.NET 应用程序的方式，在第一次调用 ASP.NET 的.aspx 文件时可以编译它们，也可以编译整个应用程序。在应用程序的 web.config 文件中，通过<system.web>部分的编译元素可以控制这些编译设置。设置<compilation debug="true">并使用/debug:full 选项，就会生成二进制代码，并创建 PDB。

大多数开发人员最关心的是 PDB 文件是否存在。ASP.NET 应用程序的 bin 文件夹中有这些文件时，运行时就会提供行号。当然，行号在调试时有极大的帮助。在交互式调试会话中，如果没有这些文件，就不能单步执行源代码。



一个有趣的 CLR 内部问题是：即使通过/debug:pdbonly 编译程序集，并且打开了调试器，也要在程序集中调用 System.Diagnostics.Debugger.Launch。JIT 编译器会在第一次调用方法时编译代码，生成的代码是可以调试的，因为 JIT 知道已关联了调试器。

2. 调试和 JIT 对话框

在 ASP.NET 应用程序中出现未处理的错误时，ASP.NET 工作者进程的默认错误处理程序就会捕获并尝试输出一些 HTML，表示发生了错误。但是，在 ASP.NET 外部调试组件时，例如在单元测试环境下调试，在.NET 应用程序抛出未处理的异常时就会打开调试对话框。

如果 ASP.NET 应用程序发生严重错误，Web 服务器就会弹出对话框，等待我们的输入。如果计算机没有连接键盘或显示器，就会特别不方便。在关闭所显示的调试对话框时，有两个选项：

- 可以在注册表中禁用 JIT 调试功能。对应的注册表键是 HKLM\Software\Microsoft\NET Framework\DbgJITDebugLaunchSetting。该选项有 3 个值：
 - 0：通过消息框的方式提示用户。其选项包括 Continue 和 Attach a Debugger。Continue 会得到堆栈转储，并终止进程。Attach a Debugger 表示运行时启动列在 DbgManagedDebugger 注册表键中的调试器。如果没有这个键，调试器就交出控制权，并终止进程。
 - 1：不显示对话框。这会得到堆栈转储，然后终止进程。

- 2: 启动列在 DbgManagedDebugger 注册表键中的调试器。



对于这个选项, 注册表条目必须设置为 0, 才能显示对话框。

- 为了禁用 JIT 调试对话框, 但仍然显示错误对话框, 应在 Visual Studio .NET 中选择 Tools | Options | Debugging | Just-In-Time 命令, 取消选择所提供的选项。此时在未处理的异常中, 不会显示 Select a Debugger 对话框, 而会显示 OK/Cancel 对话框。

29.3.2 启动调试会话

ASP.NET 中有许多种方式可以进入交互式调试会话。Visual Studio 可以启动 ASP.NET 工作者进程, 加载新编译的 Web 站点, 把调试功能自动关联到工作者进程。也可以把调试器关联到已运行的站点。Visual Studio 还有一个新的更简单的远程调试工具, 用于跨计算机的调试。

1. 通过 F5 功能键调试

开始调试 ASP.NET 应用程序时, Visual Studio 会考虑项目属性中的所有 Start 选项, 允许使用当前选中的页面启动调试功能。选择一个页面, 使 Visual Studio 调试器可以自动关联正确的进程, 这可以是 Visual Studio Web 服务器、ASP.NET 工作者进程或远程调试监控器。

2. 关联到进程

进入已在运行的 Web 站点的交互式调试会话常常是很方便的。在已知状态下, 不需要在每次调试时都重新启动应用程序。要开始调试已在运行的站点, 可以从 Visual Studio 的 Debug 菜单中选择 Attach to Process 命令。这个对话框对 Visual Studio 的以前版本进行了改进, 现在包含 Refresh 按钮, 并且通过只显示属于用户并在当前运行的会话中的进程, 简化了最常见的调试用例。

该对话框还包含传输下拉列表, 其中显示默认的传输方式。这种默认传输方式可以在本地计算机或运行 Remote Debugging Monitor 的远程计算机上选择进程。其他选项用于智能客户端或不受管理的调试。



通过 F5 功能键启动调试会话和手动关联进程之间的唯一区别是, 在通过 F5 功能键进行调试时, Visual Studio 会自动启动浏览器或外部应用程序。注意, 如果使用 Attach to Process 命令, 就假定已经完成了启动进程的工作。在至少接收到一个 HttpRequest 后, 站点就会启动 IIS 下的 ASP.NET 工作者进程。调试器现在就可以关联到运行的工作者进程。

3. 更简单的远程调试

近年来, 远程调试更加简单。但出于安全考虑, 必须有合适的凭据才能进行远程调试。在 C:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE 中有 Remote Debugger 文件夹。

首先, 必须在包含要调试的应用程序的计算机上启动远程调试功能。现在可以不进行复杂的安

装，而是使用 Remote Debug Monitor 和可以在文件共享时运行的应用程序。最简单的情况是，直接在 Visual Studio 计算机上共享这些组件，再在共享时运行 msvsmon.exe。

在文件共享时运行 Remote Debug Monitor 可执行程序，可以大大简化对已部署的应用程序进行的 ASP.NET 远程调试，但仍然需要手动关联 ASP.NET 工作者进程，因为不支持自动关联。注意调试器现在有两个版本，一个用于 x86 进程，另一个用于 x64 进程，因此应确保为进程使用正确的调试器。

可以在没有特殊权限的情况下，调试运行在自己的账户和密码下的进程。如果需要调试运行在另一账户名下的进程，例如作为另一个用户运行的 ASP.NET 工作者进程，我们就必须是运行该进程的计算机的管理员。

在远程调试时一定要注意，需要把 Visual Studio 用户账户映射为运行 Remote Debug Monitor(msvsmon.exe)的计算机的合法用户账户，反之亦然。最简单的方式是在这两台计算机上创建用户名和密码都相同的本地用户账户。要把 msvsmon 运行为非 Visual Studio 用户，就必须在每台计算机上创建两个用户账户。

如果一台计算机位于域中，就可以把域账户映射到本地账户。在两台计算机上创建本地用户账户。但是，如果选择与域账户相同的用户名和密码，Visual Studio 就可以作为域账户运行。

29.3.3 有助于调试的工具

Visual Studio 的调试功能提供了许多工具，一些工具很简单，一些工具比较精巧，但都可以在调试会话的每一步中提供帮助。

1. 调试器的数据提示功能

Visual Studio 提供了数据提示功能，可以使用无模式的树状视图显示复杂类型。数据提示的操作类似于工具提示，但提供了更多的信息。在遍历树状视图以找到自己感兴趣的节点时，就可以单击放大镜图标，使用可视化工具查看简单类型，如图 29-10 所示。

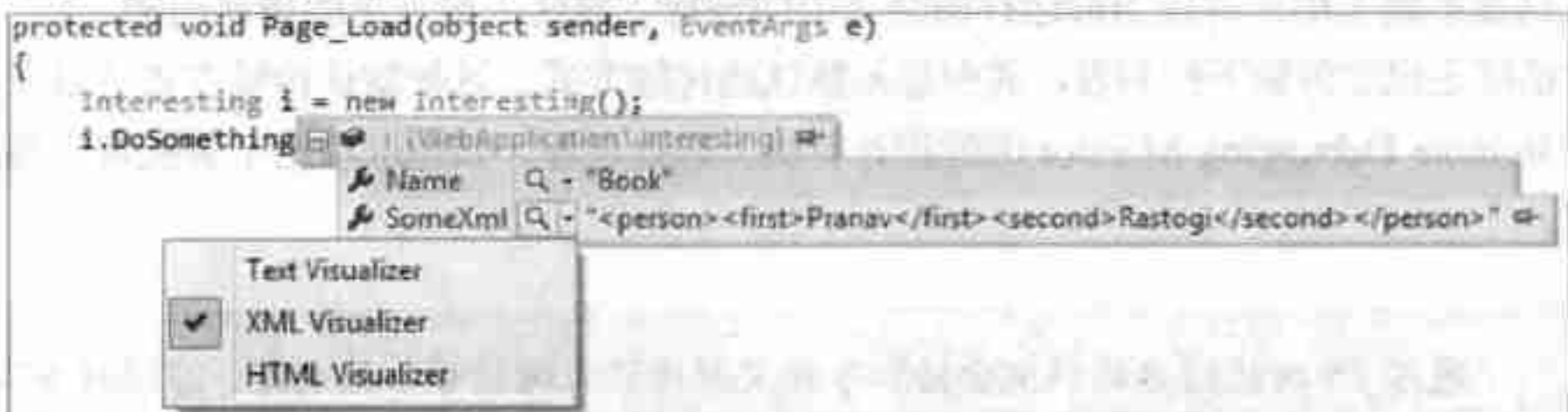


图 29-10

再次单击已经锁定的数据提示的锁状图标，就可以使数据提示处于浮动状态。这样便可以将数据提示移到屏幕的任意位置。如果这时滚动代码，那么无论在 Visual Studio 中执行哪些操作，数据提示都仍然将停留在屏幕中的当前位置。浮动数据提示的优点是可以将数据提示移到代码窗口的外面，甚至可以放到另一台显示器中。

在锁定的数据提示工具栏中注意有个按钮，可以通过该按钮放置在移动数据提示时能随其一起移动的注释。图 29-11 显示了带有注释的数据提示。

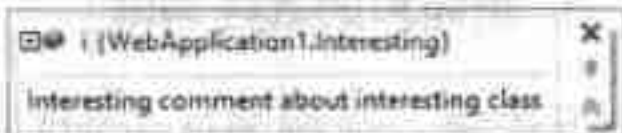


图 29-11

2. 数据可视化工具

如图 29-11 所示,可以使用任意多个数据可视化工具查看简单类型。例如,如果有个简单变量(如字符串)包含一段 XML,就可以以更适合于数据的本机格式的样式查看数据,如图 29-12 所示。

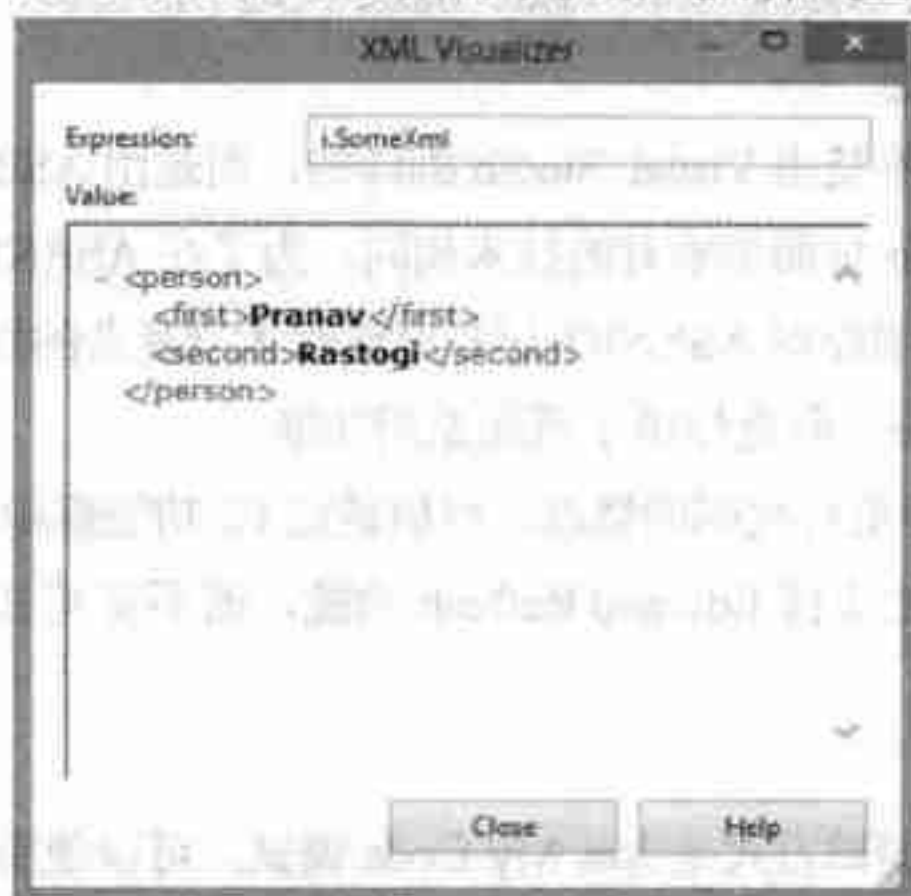


图 29-12

可视化工具的编写很简单, Visual Studio 附带了用于文本、HTML、XML 和 DataSet 的默认可视化工具。Internet 上还有许多用于图像、集合类的其他可视化工具,因此调试有许多不同的方式。

3. 错误通知

在交互式调试会话中, Visual Studio 会提供信息量很大的错误通知。这些通知不仅会报告未处理的异常等事件,还会提供与上下文相关的故障诊断提示和处理该情况的后续步骤。图 29-13 显示了未处理的 `NullReferenceException` 异常和处理建议: 在使用对象的实例之前,可以先使用 `new` 关键字进行创建。

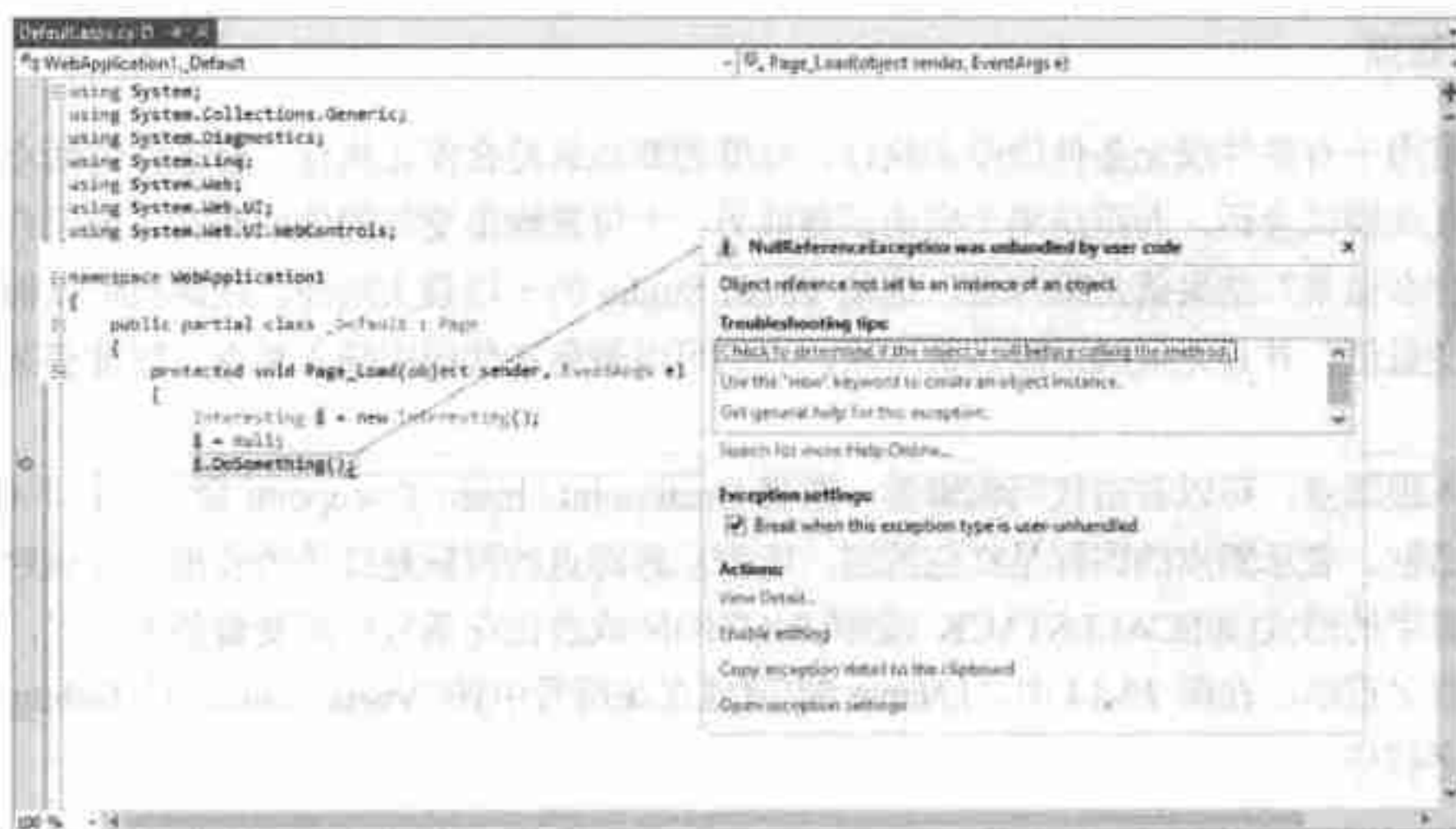


图 29-13

4. 不支持 Edit and Continue 功能，而支持 Edit and Refresh 功能

Visual Basic 提供了 Edit and Continue 功能，该功能可以在调试会话中修改代码，而无须重新启动该会话。在断点模式下，可以修改 C# 和 Visual Basic 代码中的错误，然后继续执行。但 ASP.NET 开发人员还不能使用这个功能。

在 ASP.NET 中，程序集不是由 Visual Studio 编译的，而是由 ASP.NET 运行时编译的，使用的技术与浏览器处理正常的 Web 页面请求时的技术相同。为了在 ASP.NET 中合并调试器和支持 Edit and Continue 功能，开发小组需要对 ASP.NET 运行时进行大量复杂的修改。因此，ASP.NET 开发人员没有提供对这个功能的支持，而是使用了页面重用功能。

也就是说，在调试会话中进行代码的修改，然后通过 F5 功能键刷新整个页面，自动重新编译，并重新运行。ASP.NET 基本上支持 Edit and Refresh 功能，而不支持 Edit and Continue 功能。

5. Just My Code 调试

.NET Framework 的另一调试方式是 Just My Code 调试。可以使用特性 [DebuggerNonUserCode] 明确标记代码中的任意方法。使用这种技术和 CLR 内置的其他启发式方法，调试器会跳过不重要的代码。在 Tools | Options | Debugging 菜单中有 Enable Just My Code 功能。

特性 [DebuggerHidden] 在 .NET 中也是可用的，无论用户的 Just My Code 首选项是什么，都会对调试器隐藏方法。.NET Framework 1.1 版本中的 [DebuggerStepThrough] 特性让调试器单步执行应用了该特性的方法中的代码，而不是跟踪代码。[DebuggerNonUserCode] 特性是一种更常见、更完整的实现方式，在运行时用于委托、虚函数和复杂的代码。

注意，这些特性和这个用户选项有助于高效地调试代码，但不要与调用栈混淆。这些特性非常有用，但不要在组件中使用它们，除非确定自己不会无意中隐藏要调试的错误。这些特性一般用于代理等组件。

6. 跟踪点

断点可用于有条件或无条件地停止执行。标准的断点总是会停止执行。有条件的断点会根据条件进入交互式调试会话。而跟踪用于向调试器或另一个位置输出变量的值或断言。如果合并这些功能，会有什么结果？结果就是跟踪点，这是 Visual Studio 的一项强大功能。跟踪点可以捕获某个边界条件的变量值，并且无须触发断点数十次。它们可以避免在代码中插入断点，以捕获某个奇怪的条件。

要插入跟踪点，可以右击代码编辑器，选择 Breakpoint | Insert Tracepoint 命令，打开如图 29-14 所示的对话框。表示断点的图标是红色的圆，而表示跟踪点的图标是红色的菱形。在该对话框中，可以以关键字的形式（如 \$CALLSTACK 或 \$FUNCTION）或放在花括号中的变量值的形式，使用伪变量创建任意字符串。在图 29-14 中，i.Name 的值（放在花括号中）和 Visual Studio 的 Debug 输出显示为完整的字符串。

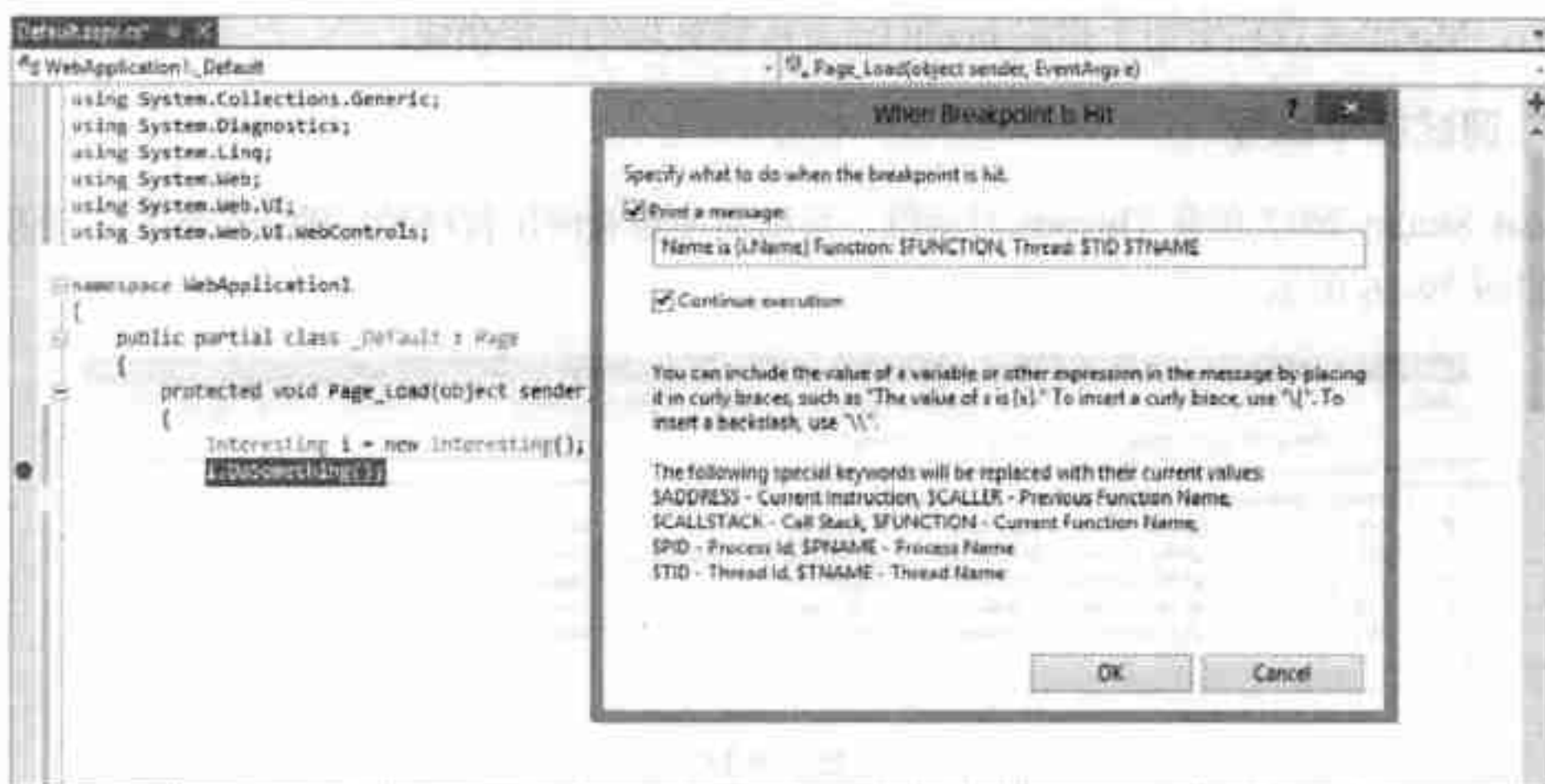


图 29-14

7. 断点选项

在代码中右击断点，就可以访问大量选项，以管理断点的一些标识符和行为。

其中一个选项用于为断点设置条件。右击断点，从提供的菜单中选择 **Condition** 命令，这将打开 **Breakpoint Condition** 对话框。在显示的文本框中可以创建触发断点的条件。

与数据提示一样，Visual Studio 2012 也允许导出断点。要输出所有断点，可以右击每一个断点并从提供的菜单中选择 **Export** 命令。导出操作可以将所有的断点保存到一个 XML 文件中，以后该文件就可以被其他人加载。这种方法方便了对这些项的分发。

29.3.4 使用 IntelliTrace 执行历史调试

Visual Studio 2012 的一项更加激动人心的新功能是 IntelliTrace。IntelliTrace 提供了历史调试器，可以回顾应用程序运行时的历史状态，跳到以前的一些时刻来查看当时的状况。

这项功能只适用于 Visual Studio 2010 开发环境，但在 Visual Studio 2012 中也可以用于产品环境。可以访问丰富的调试信息，其中显示了过去的事件和应用程序的调用信息。使用 IntelliTrace 和 Visual Studio 支持的 **Test Manager**，可以收集诊断跟踪数据，这些数据可以用于找出难以再生的错误。

在应用程序的某个断点处停止运行时，就可以直接在 IntelliTrace 对话框中看到如图 29-15 所示的运行历史信息。

从该对话框中可以跳转到过去的任何时刻，及时地查看当时的 Locals 对话框和 Call Stack 对话框中的值。IntelliTrace 功能也是可以配置的。可以单击 IntelliTrace 工具栏中的 **Open IntelliTrace Settings** 按钮来配置该功能，也可以选择 **Tools | Options** 命令，然后从 Options 对话框左端的项中选择 IntelliTrace 以进行配置。

在 IntelliTrace Events 选项卡中，可以选择要处理的事件。可以从一系列的技术和情况进行适当的选择。

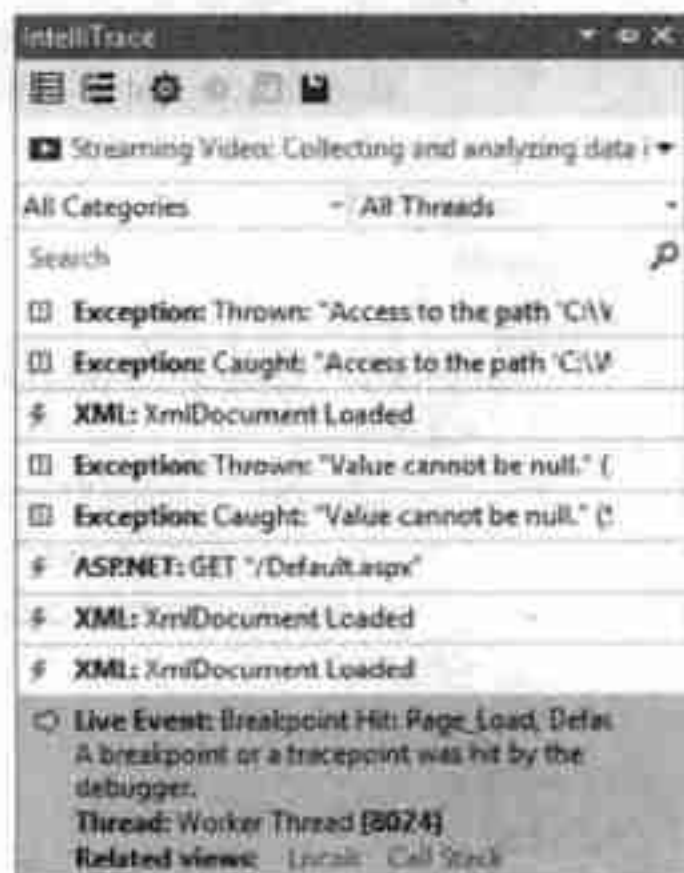


图 29-15

最后，Modules 选项卡用于指定 IntelliTrace 应该处理的特定模块。

29.3.5 调试多个线程

Visual Studio 2012 包含 Threads 对话框，可以从中查看应用程序使用的每个线程中正在进行的操作，如图 29-16 所示。



图 29-16

在调试时双击线程，就会进入线程在代码中的位置。通过打开 Parallel Stacks 对话框，还能得到关于正在运行的线程的可视化家族树视图。

29.3.6 客户端的 JavaScript 调试

Visual Studio 2012 具有优秀的客户端 JavaScript 调试功能。如果在 Internet Explorer 的调试会话中运行 ASP.NET 应用程序，就需要启动脚本调试功能。

启用脚本调试功能后，尝试运行包含一些 JavaScript 的 ASP.NET 页面，单击按钮后，该页面把文本框中的文本改为大写(程序清单 29-5)。

程序清单 29-5 简单的 JavaScript 调试测试

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <script type="text/javascript">
    function MakeItUpper()
    {
      newText = document.getElementById("TextBox1").value.toUpperCase();
      document.getElementById("TextBox1").value = newText;
    }
  </script>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <input type="button" id="Button1" value="Upper"
        onclick="javascript:MakeItUpper()" />
      <input type="text" id="TextBox1" runat="server"/>
    </div>
  </form>
</body>
</html>
```

在客户端 JavaScript 代码的其中一行上放置断点。注意这是在浏览器中运行的代码，而不是在

Web 服务器上运行的代码。使用程序清单 29-5 中的页面启动调试会话。Visual Studio 会在这个断点处中断，如图 29-17 所示。



图 29-17

Visual Studio 中的 JavaScript 调试器支持变量工具提示、可视化工具、调用栈、本地窗口、监视窗口和调试基于 .NET 的语言时使用的的所有功能。

29.3.7 SQL 存储过程的调试

数据库项目是基于文件的项目，可以管理和执行数据库查询。可以在项目中添加已有的 SQL 脚本或创建新的 SQL 脚本，再在 Visual Studio 中编辑它们。在 Visual Studio 的 Express 或 Standard 版本中不支持数据库项目和 SQL 调试功能。它们只能在 Visual Studio SKU 的 Professional Edition 或 Team Edition 版本中使用。

在调试数据库应用程序时，不能在应用程序层的代码和 SQL Server(可以是 T-SQL 或 CLR SQL)的代码之间使用 Step Into 命令(F11 功能键)调试。但是，可以在存储过程代码中设置断点，使用 Continue 命令(F5 功能键)执行到设置断点的代码。

在 SQL Server 2012 中调试 SQL 时，注意可能正在运行的软件或硬件防火墙。有时软件防火墙会对要执行的操作发出警告。在警告对话框中一定要选择解除阻塞，才能确保 SQL Server 2012 和 Visual Studio 可以相互通信。

如果使用 SQL 账户连接 SQL Server，就应确保运行 Visual Studio 的 Windows 用户账户也是 SQL Server 计算机的管理员。可以使用 SQL 命令 “sp_addsrvrolemember 'Domain\Name', 'sysadmin'” 在 SQL Server 的 sysadmin 权限中添加账户。当然，不要在产品系统中这么做。最好在本地安装所有软件的计算机上进行调试。

如果使用 SQL Server 2012 的 NT 验证模型，就要确保账户有运行 sp_enable_sql_debug 存储过程的权限。使用 SQL 命令 CREATE USER UserName FOR LOGIN 'Domain\Name' 后跟 GRANT EXECUTE ON sp_enable_sql_debug TO UserName，就可以给账户授予访问这个存储过程的权限。这会创建直接与特定 Windows 用户关联的 SQL 用户，接着把调试 T-SQL 的权限明确授予该用户。在 SQL Server 2000 中，用户必须有访问扩展存储过程 sp_sdebug 的权限。

29.4 异常和错误处理

当 ASP.NET 应用程序代码中出现异常时，可以使用许多方式来处理，最好的方式是使用多分支结构：

- 捕获希望的内容：
 - 为易出错的代码使用 try/catch 块，这种结构总是可以捕获要处理的特定异常，如 System.IO.FileNotFoundException。
 - 不是在页面级捕获特定代码块中的异常，而是使用页面级的错误处理程序来捕获可能在页面上任意地方出现的某些异常。
- 但是，要对未处理的异常做如下准备工作：
 - 如果某个页面应为未处理的异常显示特定的错误页面，就设置 Page.Error 属性。也可以使用 <%@ Page> 指令或该属性的隐藏代码来完成该操作。
 - 在 web.config 文件中为 400 和 500 错误设置默认错误页面。
 - 建立 Application_OnError 样板处理程序来处理特定的异常以及所有未处理的异常，未处理的异常要记录到事件日志、文本文件或其他介质中。

未处理的异常可能会令人担忧，但不应捕获不能恢复的异常。如果未处理的异常只是例外情况，就没有什么问题。对于这种情况，可以利用全局异常处理程序记录它们，并向用户显示友好的错误页面。



如果可以在一个地方捕获和记录所有的异常，为什么还要到处添加捕获异常的代码？一种常见的错误是为任意代码建立 try/catch 块，捕获最一般的异常类型 System.Exception。一条通用规则是，不要捕获不能作任何处理的异常。因为异常可以由某个方法抛出，但并不意味着必须捕获，这是例外情况。另外，在页面级和应用程序级建立异常处理程序，在这两个集中位置捕获异常比较好。

29.4.1 处理页面上的异常

要处理页面上的异常，应重写 System.Web.UI.Page 从 TemplateControl 类继承的 OnError 方法，如程序清单 29-6 所示。调用 Server.GetLastError 方法，就可以访问刚才抛出的异常。注意所发生的异常链，可以使用 Exception.GetBaseException 方法返回根异常。

程序清单 29-6 页面级的错误处理

```
protected override void OnError(EventArgs e)
{
    System.Exception anError = Server.GetLastError();
    if (anError.GetBaseException() is SomeSpecificException)
    {
        Response.Write("Something bad happened!");
        Response.StatusCode = 200;
        Server.ClearError();
    }
}
```

```

        Response.End();
    }
}

```

29.4.2 处理应用程序异常

在集中位置捕获异常的技术可以应用于 Global.asax 中应用程序级的错误处理, 如程序清单 29-7 所示。如果没有在页面上捕获到异常, 就要检查 web.config 文件中是否有对应的错误页面; 如果没有, 异常就会从应用程序中显示出来, 用户将看到完整的调用栈。

程序清单 29-7 应用程序级的错误处理

```

protected void Application_Error(Object sender, EventArgs e)
{
    System.Exception bigError = Server.GetLastError();
    //Example checking for HttpRequestValidationException
    if (bigError.GetBaseException() is HttpRequestValidationException)
    {
        System.Diagnostics.Trace.WriteLine(bigError.ToString());
        Server.ClearError();
    }
}

```

未处理的应用程序错误会转变成 HTTP 状态码 500, 并在浏览器中显示错误。这些错误, 包括完整的调用栈和其他技术细节, 在开发过程中都是有用的, 但在产品阶段就没有什么用处。我们常常希望创建错误处理程序来记录错误, 并向用户显示友好的页面。



如果试图捕获 System.Exception 类型的异常, 就应查看代码, 确定是否能避免。不应捕获非特定的异常, 而应捕获可以提供有价值的调试信息的异常。查看 API 文档中对所调用的架构方法的解释, 该部分列出了 API 调用可能抛出的异常。不要期望异常的发生会对标准代码路径的获取起作用。

29.4.3 HTTP 状态码

每个 HttpRequest 都会得到相应的 HttpResponse, 而每个 HttpResponse 都包含状态码。表 29-2 中列出了 11 个特别值得注意的 HTTP 状态码。

表 29-2

状 态 码	说 明
200 OK	一切正常
301 Moved Permanently	提醒调用者使用新的、永久的 URL 来代替以前使用的 URL
302 Found	在 Response.Redirect 过程中返回, 这是表示“不, 现在就查找”的一种方式
304 Not Modified	未修改请求的文档时, 返回为条件 GET 的结果。它是所有基于浏览器的高速缓存的基础。在使用 304 状态码时, 不能返回 HTTP 消息主体
307 Temporary Redirect	把对 ASMX Web 服务的调用重定向到其他 URL。在 ASP.NET 中很少用到该状态码


(续表)

状 态 码	说 明
400 Bad Request	请求不正确
401 Unauthorized	请求需要用户的身份验证
403 Forbidden	验证失败，表示服务器理解该请求，但无法实现
404 Not Found	服务器没有找到合适的文件或处理程序来处理这个请求。其含义是这可能是临时状态。在 ASP.NET 中发生这种情况，不仅是因为找不到文件，还因为不能正确映射到 IHttpHandler，所以不能处理该请求
410 Gone	相当于永久的 404，它告诉客户端应尽可能删除该链接的所有引用。404 状态码通常表示服务器不知道该条件是否是永久的
500 Internal Server Error	这个错误的正式文本是“服务器遇到未预料到的条件，因此不能处理请求”，但在 ASP.NET 中的用户发现未处理的异常时，也可能发生这个错误

所有大于等于 400 的状态码都是错误，除非进行配置，否则用户会在浏览器上看到不友好的消息。如果没有在 ASP.NET 运行时中通过检查这些错误的异常类型来处理它们，或者错误在 ASP.NET 的外部发生，但我们希望给用户显示友好的消息，就可以在 web.config 文件中给页面指定状态码，如下所示：

```
<customErrors mode="On" >  
  <error statusCode="500" redirect="FriendlyMassiveError.aspx" />  
</customErrors>
```

对 web.config 文件的定制错误部分进行修改后，确保会向用户显示一个页面。错误重定向中的经典错误是，把用户重定向到会产生错误的页面，使用户陷入错误循环。如果应用程序中有非常复杂的页眉或页脚，它们显示在错误页面上时会导致错误，此时就应非常小心。应避免使用数据库，或者执行需要用户授权或用户会话处于指定状态下的后台操作。换言之，确保错误页面是完全独立的。



所有大于或等于 400 的状态码都会递增 ASP.NET Requests Failed 性能计数器。401 状态码会递增 Requests Failed 和 Requests Not Authorized 性能计数器。404 和 414 状态码会递增 Requests Failed 和 Requests Not Found 性能计数器。500 状态码会递增 Requests Failed 和 Requests Timed Out 性能计数器。如果要返回状态码，就必须注意它们的作用和影响。

29.5 用 Page Inspector 进行调试

Page Inspector 是 Visual Studio 2012 中新的工具选项，它将浏览器中的调试工具带入 Visual Studio。这表示，使用 Page Inspector 可以检查元素，看看哪个文件和源代码行生成了标记。Page Inspector 也允许检查 DOM 元素和 CSS 元素，因此可以修改这些属性，实时查看变化。

图 29-18 显示了应用程序加载到 Page Inspector 中时的外观。

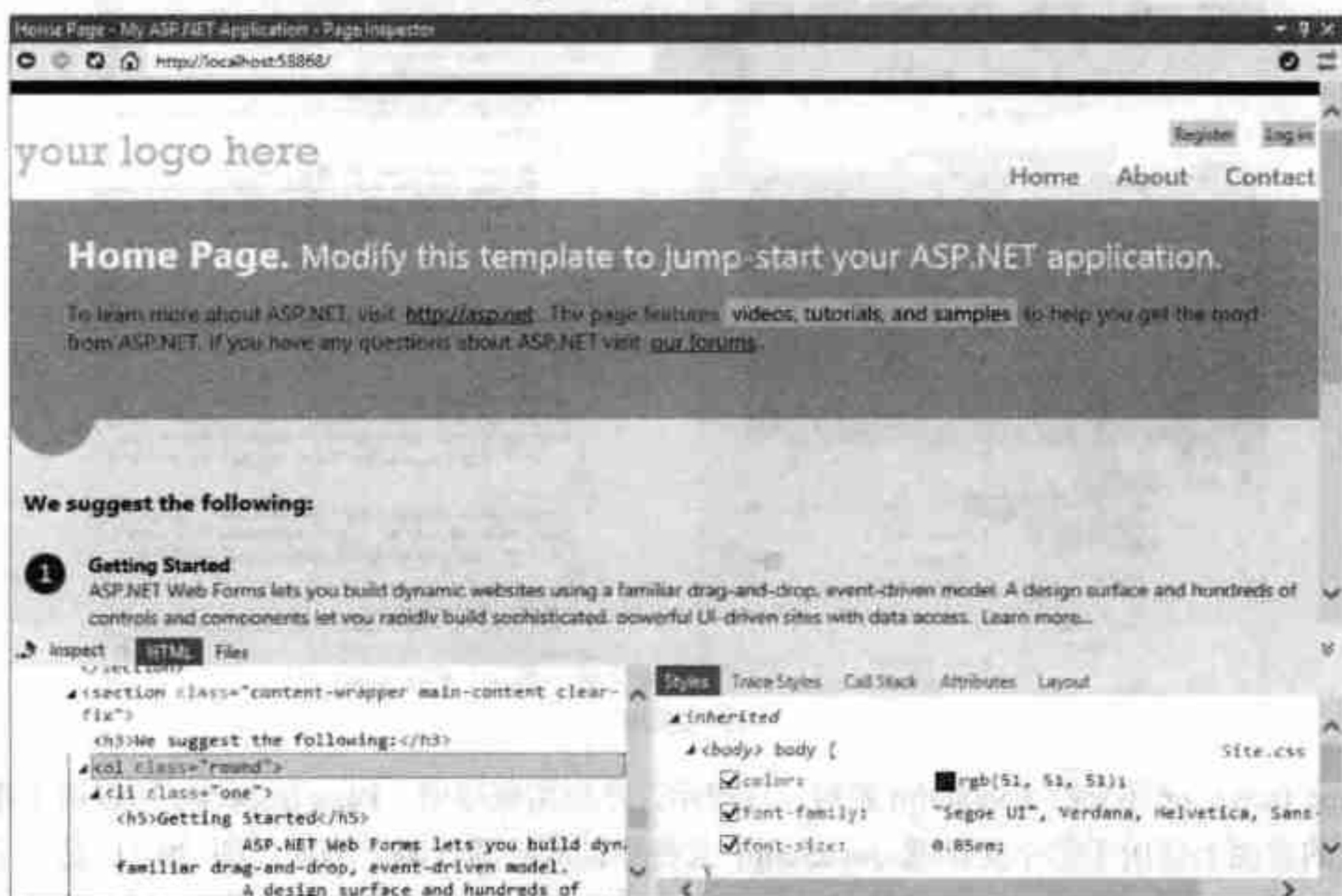


图 29-18

如果希望在 Page Inspector 中加载应用程序, 就可以右击项目, 选择 View in Page Inspector 选项。图 29-19 显示了 Visual Studio 2012 中的这个选项。单击这个选项时, Visual Studio 会编译项目, 把应用程序加载到 Page Inspector 中。

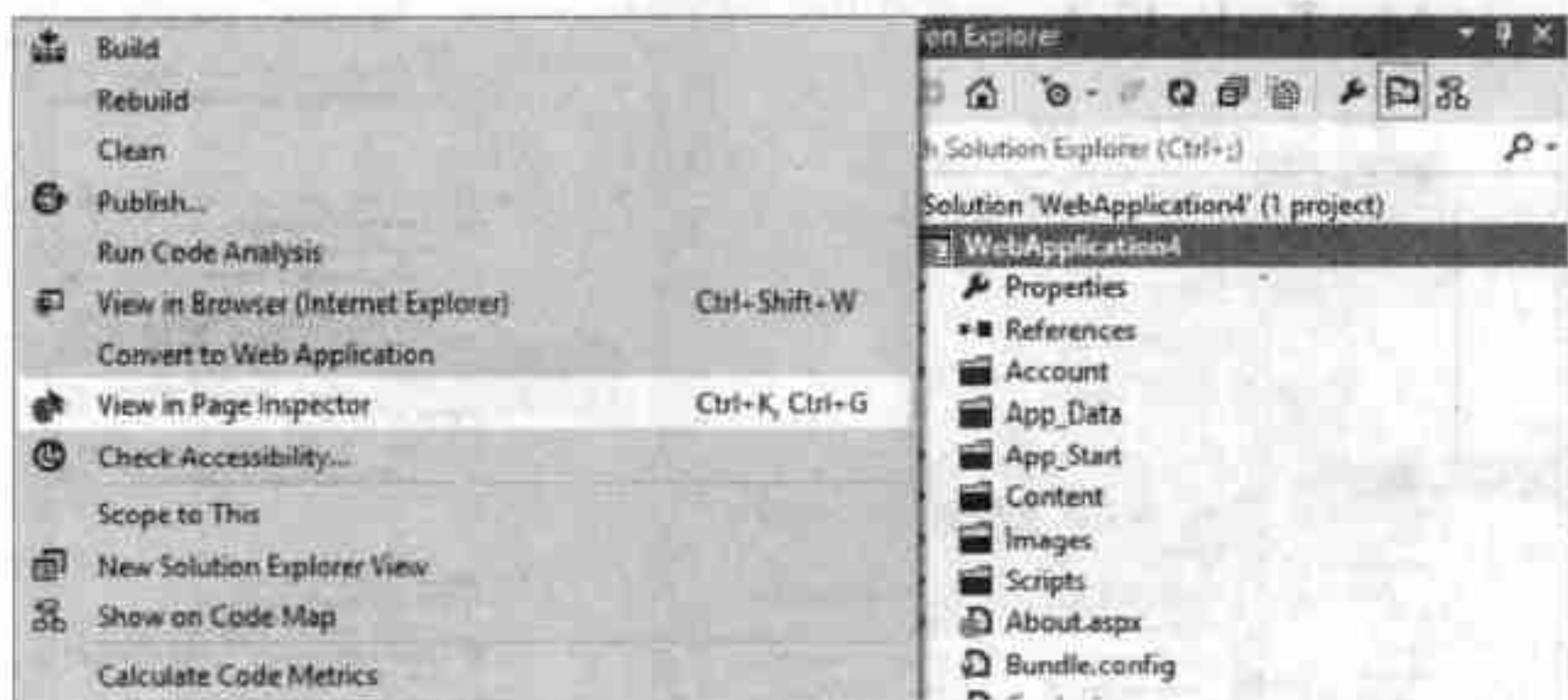


图 29-19

Page Inspector 的主要功能是 Source Mapping。检查页面上的某个元素时, Page Inspector 会在 Visual Studio 的预览模式下打开对应的文件, 突出显示生成该元素的源代码行。例如图 29-20 显示了预览模式下的 Default.aspx, 在 Page Inspector 中检查页面上的标题时, 会突出显示 Page Title 属性。

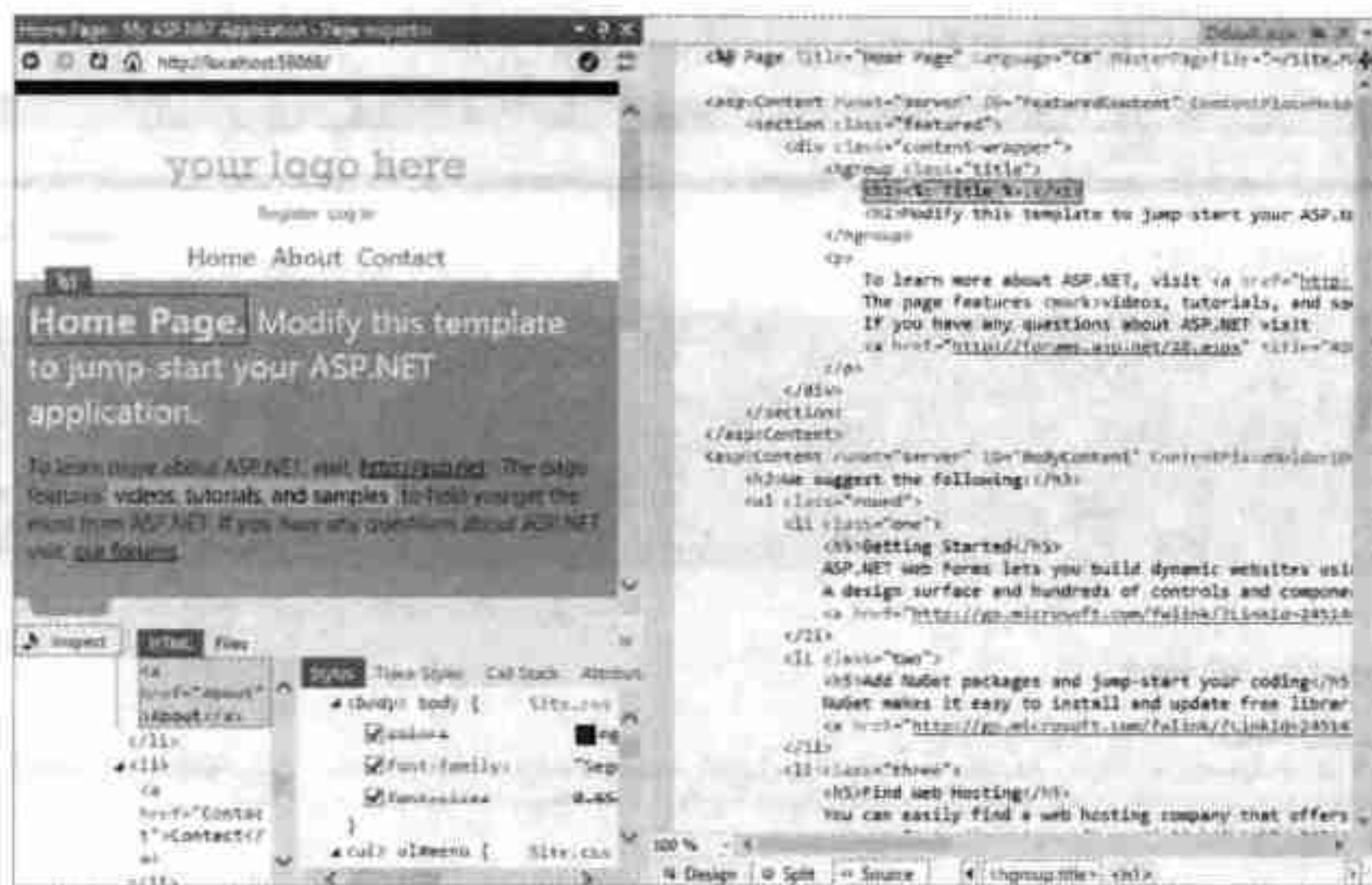


图 29-20

Page Inspector 还支持 JavaScript 映射。这表示，在应用程序中，Page Inspector 可以用于指出当前浏览的页面上使用了哪个文件或 JavaScript 文件中的哪行源代码。例如，图 29-21 显示了使用 Knockout 的 SPA 应用程序。检查元素时，Page Inspector 的 Call Stack 选项卡会显示创建这个元素的 JavaScript 调用堆栈。单击任何调用堆栈函数，Page Inspector 就会显示对应的 JavaScript 代码行。建立使用许多 JavaScript 的应用程序，且 JavaScript 代码修改 DOM 元素时，这个功能很有用。

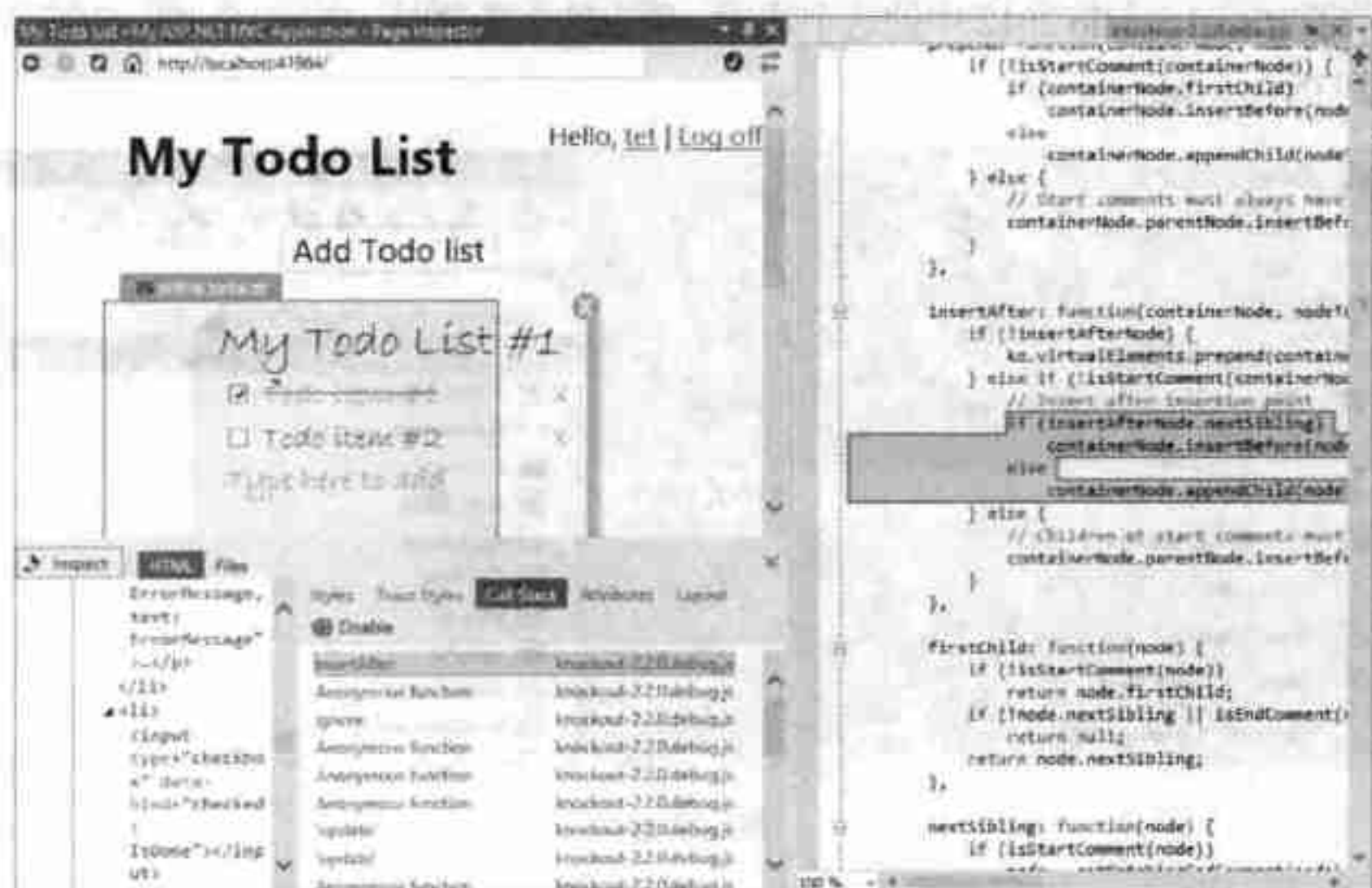


图 29-21

如前所述，Page Inspector 可以用于与 DOM 和 CSS 工具进行交互操作，修改 CSS 特性，实时查看变化。例如，要修改 CSS 特性，可以选择 Page Inspector 中的 Styles 选项卡，查看 Web 页面上使用的所有 CSS 样式。假定希望修改 Web 页面的背景色，就可以选择 background-color 特性，修改颜色，保存 CSS 文件，在 Page Inspector 中刷新页面，查看变化。

29.6 本章小结

本章介绍了为创建健壮的 ASP.NET 应用程序而提供的调试工具。成功的调试过程不仅包括使用数据提示工具、数据可视化工具和错误通知等新功能进行交互式调试，还包括使用可配置的跟踪功能和记录信息等选项。

在 ASP.NET 中，远程调试更容易进行，还可以在不安装 IIS 的情况下编写和调试 ASP.NET 页面，从而清除开发过程中另一复杂的层面。

Visual Studio 及其可扩展的调试机制必然会由无畏的博主和爱好者扩展，使调试比过去更简单。

第 30 章

模块和处理程序

本章要点

- 与 ASP.NET 请求处理管道交互
- 使用 `HttpModule` 和 `HttpHandler`

有时,使用最新的语言和数据库创建动态的 Web 页面并不能给开发人员提供对应用程序的足够控制。我们还需要深入研究,创建可以与 Web 服务器交互的应用程序。我们需要能够与低级过程交互,例如 Web 服务器如何处理入站(incoming)和出站(outgoing)的 HTTP 请求。

在 ASP.NET 推出之前,为了使用 IIS 获得这个级别的控制,必须创建 ISAPI 扩展或过滤器。这对许多开发人员来说是一项令人畏惧的、烦琐的任务,因为创建 ISAPI 扩展和过滤器需要用到 C/C++ 知识,并且需要知道如何创建本机的 Win32 DLL。而在 .NET 中,创建这类低级应用程序不会比创建其他大多数应用程序更困难。本章将介绍 ASP.NET 处理 HTTP 请求的两种方法: `HttpModule` 和 `HttpHandler`。这两种方法为 ASP.NET 的底层处理过程提供了独特的访问级别,它们是创建 Web 应用程序的强大工具。

30.1 处理 HTTP 请求

在开始编写处理程序或模块之前,应首先了解 IIS 和 ASP.NET 如何处理入站的 HTTP 请求,以及在将定制逻辑插入这些请求时可以使用什么选项。IIS 是入站 HTTP 请求的基本端点。从较高的层面上来看,它的工作是监听和验证入站的 HTTP 请求,然后把它们路由到合适的模块进行处理,再把结果返回给最初的请求者。ASP.NET 是处理 IIS 传送过来的请求的模块之一。但是,这些处理如何进行以及如何把自己的逻辑插入管道取决于所使用的 IIS 版本。

30.1.1 IIS 6 和 ASP.NET

如果使用的是 IIS 6, HTTP 请求处理管道对托管代码的开发人员而言就是黑盒。IIS 基本上把

ASP.NET 当作可以处理传送过来的请求的模块，而不是将其当作 IIS 请求处理管道的组成部分。图 30-1 显示了 IIS 6 和 ASP.NET 的基本请求处理管道。

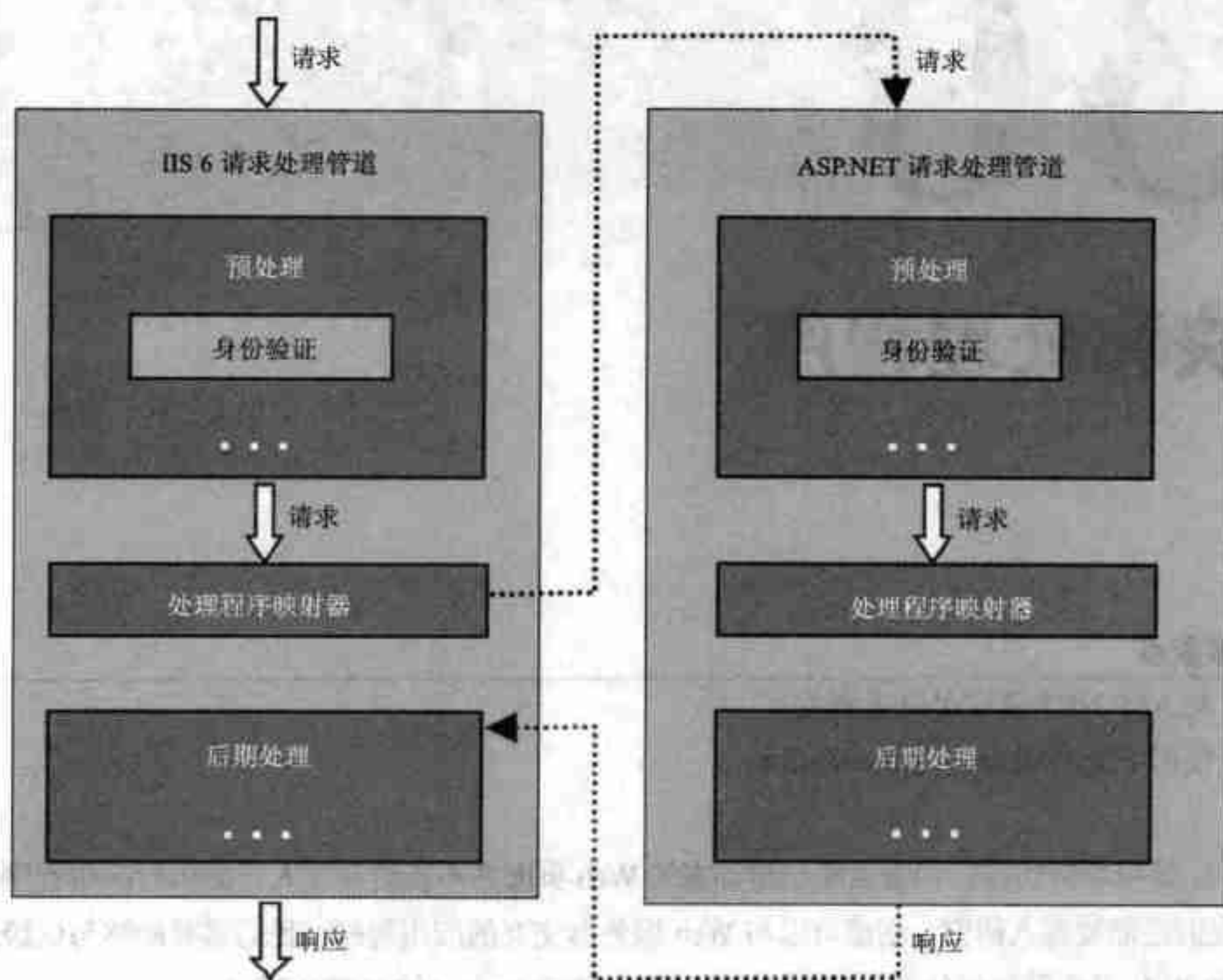


图 30-1

可以看出，IIS 和 ASP.NET 请求管道是非常类似的，有些任务，如身份验证，甚至在这两个管道中是重复的。而且，虽然可以使用托管代码编写处理程序和模块，但是仍然在 ASP.NET 进程的独立环境下处理它们。如果要更深入地集成到 IIS 管道中，就必须使用本机代码创建模块。

30.1.2 IIS 7、IIS 8 和 ASP.NET

从 IIS 7 开始，使用开放的、高度可扩展的、基于模块的系统完全重新构建了 IIS 中的请求处理管道。在 IIS 8 中仍是这样。现在，IIS 不再把 ASP.NET 看作独立的实体，ASP.NET 已经深入集成到 IIS 请求处理管道中。如图 30-2 所示，已经删除了请求处理管道中重复的过程，并且允许在管道中集成托管模块。

因为 ASP.NET 模块是极为优秀的成员，所以可以把它们放在管道的任意地方，甚至使用自己定制的功能完全替代已有的模块。对于以前在非托管代码中编写定制 ISAPI 模块所需的功能，现在可以使用包含自己逻辑的托管代码模块替代。

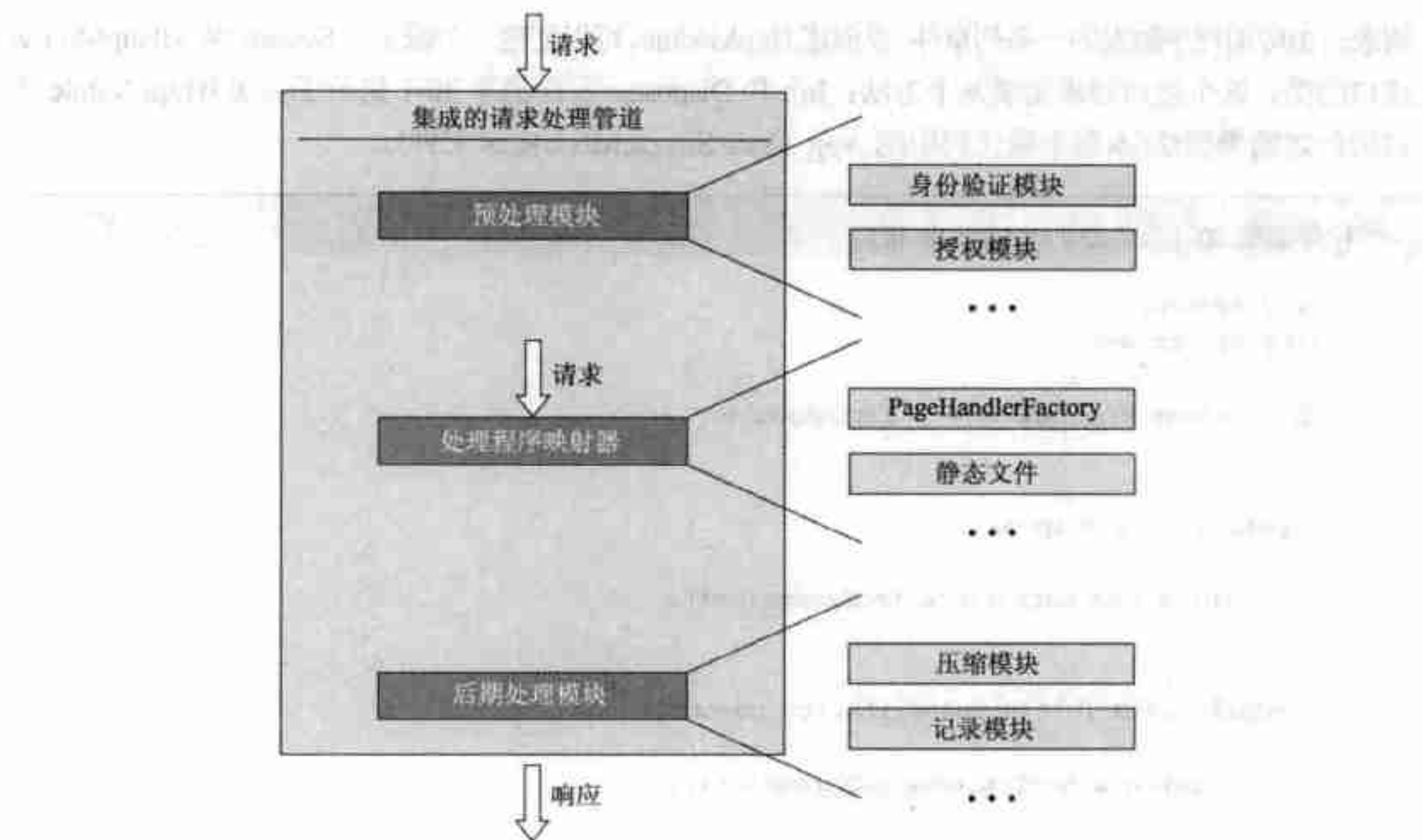


图 30-2

30.1.3 ASP.NET 请求处理

无论使用 IIS 的哪个版本，基本 HTTP 请求管道模型都有处理请求的两个核心机制：HttpModule 和 HttpHandler。ASP.NET 使用这两个机制处理入站的 ASP.NET 请求，生成响应并把响应返回给客户端。事实上，读者已经熟悉了 HttpModule 和 HttpHandler——尽管读者自己可能并不知道这一点。如果读者以前使用过 Inbox 高速缓存或 ASP.NET 的身份验证功能，就已经使用过几个不同的 HttpModule。另外，如果以前接触过 ASP.NET 应用程序，甚至是最简单的 Hello World Web 页面，并在浏览器中查看，就说明已经使用过 HttpHandler。ASP.NET 使用处理程序处理和显示 ASPX 页面和其他扩展名的文件。使用模块和处理程序可以在不同的地方插入请求处理管道，与 IIS 处理的请求交互。

从图 30-1 和图 30-2 中可以看出，ASP.NET 在处理管道中通过预处理 HttpModule 层来传送每个入站的请求。ASP.NET 允许在管道中存在有多个模块以处理每个请求。在入站请求通过每个模块后，就传送给 HttpHandler，它负责处理请求。注意，尽管一个请求要通过许多不同的模块，但只能由一个处理程序来处理。该处理程序一般负责给入站的 HTTP 请求创建响应。在处理程序执行完毕并生成响应后，就通过一系列后期处理模块将响应返回给客户端。

基本了解了 IIS 和 ASP.NET 请求管道以及如何使用 HttpModule 和 HttpHandler 与管道交互之后，下面就深入介绍它们。

30.2 HttpModule

HttpModule 是一些很简单的类，可以把它们插入到请求处理管道中。为此，它们要关联在处理 HTTP

请求时由应用程序触发的一系列事件。要创建 `HttpModule`，可以创建一个派生于 `System.Web.IHttpModule` 接口的类。这个接口要求实现两个方法：`Init` 和 `Dispose`。程序清单 30-1 显示了实现 `IHttpModule` 接口后创建的类模块(本章下载代码中的 `App_Code\SimpleModule.cs` 文件)。

程序清单 30-1 实现 IHttpModule 接口

```
using System;
using System.Web;

public class SimpleModule : IHttpModule
{
    public void Dispose()
    {
        throw new NotImplementedException();
    }

    public void Init(HttpApplication context)
    {
        throw new NotImplementedException();
    }
}
```

`Init` 方法是用于实现 `HttpModule` 功能的主要方法。注意，它有一个方法参数，即 `HttpApplication` 对象 `context`。这个参数允许访问当前的 `HttpApplication` 环境，它用于封装在处理请求的过程中触发的不同事件。表 30-1 列出了可以在 `Init` 方法中注册的事件。“顺序”列表示它们第一次触发的顺序。一些事件在处理请求的过程中会触发多次。

表 30-1

事件名	顺序	说明
AcquireRequestState	9	在 ASP.NET 运行时准备获得当前 HTTP 请求的会话状态时触发
AuthenticateRequest	2	在 ASP.NET 运行时准备验证用户的身份时触发
AuthorizeRequest	4	在 ASP.NET 运行时准备给用户授予资源的访问权限时触发
BeginRequest	1	在 ASP.NET 运行时收到新的 HTTP 请求时触发
Disposed	23	在 ASP.NET 完成 HTTP 请求处理时触发
EndRequest	20	在把响应内容发送给客户端之前触发
Error	N/A	在处理 HTTP 请求的过程中发生未处理的异常时触发
LogRequest	18	在 ASP.NET 对当前请求进行记录前触发
PostAcquireRequestState	10	在获得与当前请求相关的请求状态(例如会话状态)时触发
PostAuthenticateRequest	3	在安全模块建立用户的标识符时触发
PostAuthorizeRequest	5	在给当前请求的用户授权时触发
PostLogRequest	19	在 ASP.NET 处理完 LogRequest 事件的所有事件处理程序时触发
PostMapRequestHandler	8	在 ASP.NET 把当前请求映射到对应的事件处理程序时触发
PostReleaseRequestState	15	在 ASP.NET 执行完所有请求的事件处理程序，并存储了请求状态数据时触发
PostRequestHandlerExecute	12	在 HTTP 处理程序执行完毕后触发

(续表)

事件名	顺序	说明
PostResolveRequestCache	7	在 ASP.NET 绕过当前事件处理程序的执行, 并允许缓存模块处理高速缓存中的请求时触发
PostUpdateRequestCache	17	在 ASP.NET 更新完缓存模块, 在高速缓存中存储了用于处理后续请求的响应时触发
PreRequestHandlerExecute	11	在 ASP.NET 开始执行 HTTP 请求的处理程序之前触发。在这个事件后, ASP.NET 会把请求发送给合适的 HTTP 处理程序
PreSendRequestContent	14	在 ASP.NET 把响应内容发送给客户端之前触发。这个事件可以修改响应内容, 之后把它发送给客户端。可以使用这个事件给页面输出添加所有页面中都有的公共内容, 例如公共菜单、页眉或页脚
PreSendRequestHeaders	21	在 ASP.NET 把 HTTP 响应报头发送给客户端之前触发。这个事件可以修改报头, 之后把它发送给客户端。可以使用这个事件给报头添加 cookie 和定制数据
ReleaseRequestState	13	在 ASP.NET 执行完所有请求的事件处理程序后触发。这个事件会让状态模块保存当前的状态数据
RequestCompleted	22	在释放与请求相关的托管对象时触发
ResolveRequestCache	6	在 ASP.NET 完成给事件授权, 让缓存模块处理高速缓存中的请求, 不执行事件处理程序(例如, 页面或 XML Web 服务)时触发
UpdateRequestCache	16	在 ASP.NET 执行完事件处理程序, 让缓存模块在高速缓存中存储了用于处理后续请求的响应时触发

为了了解如何创建和使用 `HttpModule`, 可以使用一个简单的例子, 该例修改 HTTP 输出流, 之后把它发送给客户端。如果想要给 Web 站点上的每个页面都添加文本(例如日期/时间戳或处理请求的服务器), 而不希望修改应用程序中的每个页面, 那么这就是一种简单而有用的方式。为了开始创建 `HttpModule`, 需要在 Visual Studio 中创建一个 Web 项目, 在 `App_Code` 目录下添加一个类文件。`HttpModule` 的代码如程序清单 30-2 所示(本章下载代码中的 `App_Code\AppendMessage.cs` 文件)。

程序清单 30-2 修改 ASP.NET Web 页面的输出

```
using System;
using System.Web;

public class AppendMessage : IHttpModule
{
    private HttpApplication _application = null;

    public void Dispose()
    {
    }
}
```



```

public void Init(HttpApplication context)
{
    _application = context;
    context.EndRequest += context_EndRequest;
}

void context_EndRequest(object sender, EventArgs e)
{
    string message = string.Format("processed on {0}",
        System.DateTime.Now.ToString());
    _application.Context.Response.Write(message);
}
}

```

程序清单 30-2 处理 EndRequest 事件，以扩展程序清单 30-1 中的类模块。在将 HttpHandler 创建的内容发送给客户端之前触发这个事件，这是修改内容的最后一次机会。

为了修改请求的内容，在 EndRequest 处理方法中只需要将修改写入 HttpResponse 对象的输出流中，该输出流将修改的内容添加到已有内容的末端。该例将当前日期和时间写入输出流，然后将 HTTP 请求发送回客户端。

为了使用这个模块，必须让 ASP.NET 知道，我们要在请求处理管道中包含该模块。为此，可以修改 web.config 文件，使其包含该模块的一个引用。程序清单 30-3 演示了如何把<httpModules>部分添加到 web.config 文件中。

程序清单 30-3 给 web.config 文件添加<httpModule>配置

```

<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.5" />
    <httpRuntime targetFramework="4.5" />
  </system.web>
  <system.webServer>
    <httpModules>
      <add name="AppendMessage" type="AppendMessage, App_Code" />
    </httpModules>
  </system.webServer>
</configuration>

```

<httpModules>部分的一般格式如下：

```

<httpModules>
  <add name="[modulename]" type="[namespace.classname, assemblyname]" />
</httpModules>

```

如果是在 IIS 6 服务器上部署应用程序，那么还必须把模块配置添加到<system.webServer>配置部分：

```

<httpModules>
  <add name="AppendMessage" type=" AppendMessage, App_Code"/>
</httpModules>

```

如果是在运行 IIS 7 或 IIS 8 的服务器上部署应用程序, 并把<httpModules>部分添加到 web.config 文件中, 就应在<system.webServer>配置部分添加如下验证项:

```
<validation validateIntegratedModeConfiguration="false"/>
```

程序清单 30-4(本章下载代码中的 web.config 文件)是在<system.web>和<system.webServer>配置部分注册模块后的 web.config 文件。

程序清单 30-4 注册模块后的 web.config 文件

```
<?xml version="1.0"?>

<!--
  For more information on how to configure your ASP.NET application, please visit
  http://go.microsoft.com/fwlink/?LinkId=169433
-->

<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.5" />
    <httpRuntime targetFramework="4.5" />
    <httpModules>
      <add name="AppendMessage" type="AppendMessage, App_Code" />
    </httpModules>
  </system.web>
  <system.webServer>
    <validation validateIntegratedModeConfiguration="false"/>
    <modules>
      <add name="AppendMessage" type="AppendMessage, App_Code" />
    </modules>
  </system.webServer>
</configuration>
```

如果在 ASP.NET 网站的 App_Code 目录下已经创建 HttpModule, 那么考虑到 ASP.NET 现在是在运行期间动态编译这段代码, 就应清楚 assemblyname 的值。解决方法是把文本 App_Code 用作程序集的名称。这就告诉 ASP.NET, 模块位于动态创建的程序集中。

还可以把 HttpModule 创建为独立的类库, 此时可以使用库的程序集名。

把这个部分添加到 web.config 文件中后, 在浏览器中查看项目的一个 Web 页面, 示例页面 Basic.aspx 已包含在示例项目中。在浏览器中查看页面, 会发现在页面的底部添加了一条消息。如果查看页面的源代码, 就会注意到在 HTML 底部添加的消息。

图 30-3 显示了查看页面源代码的情况。

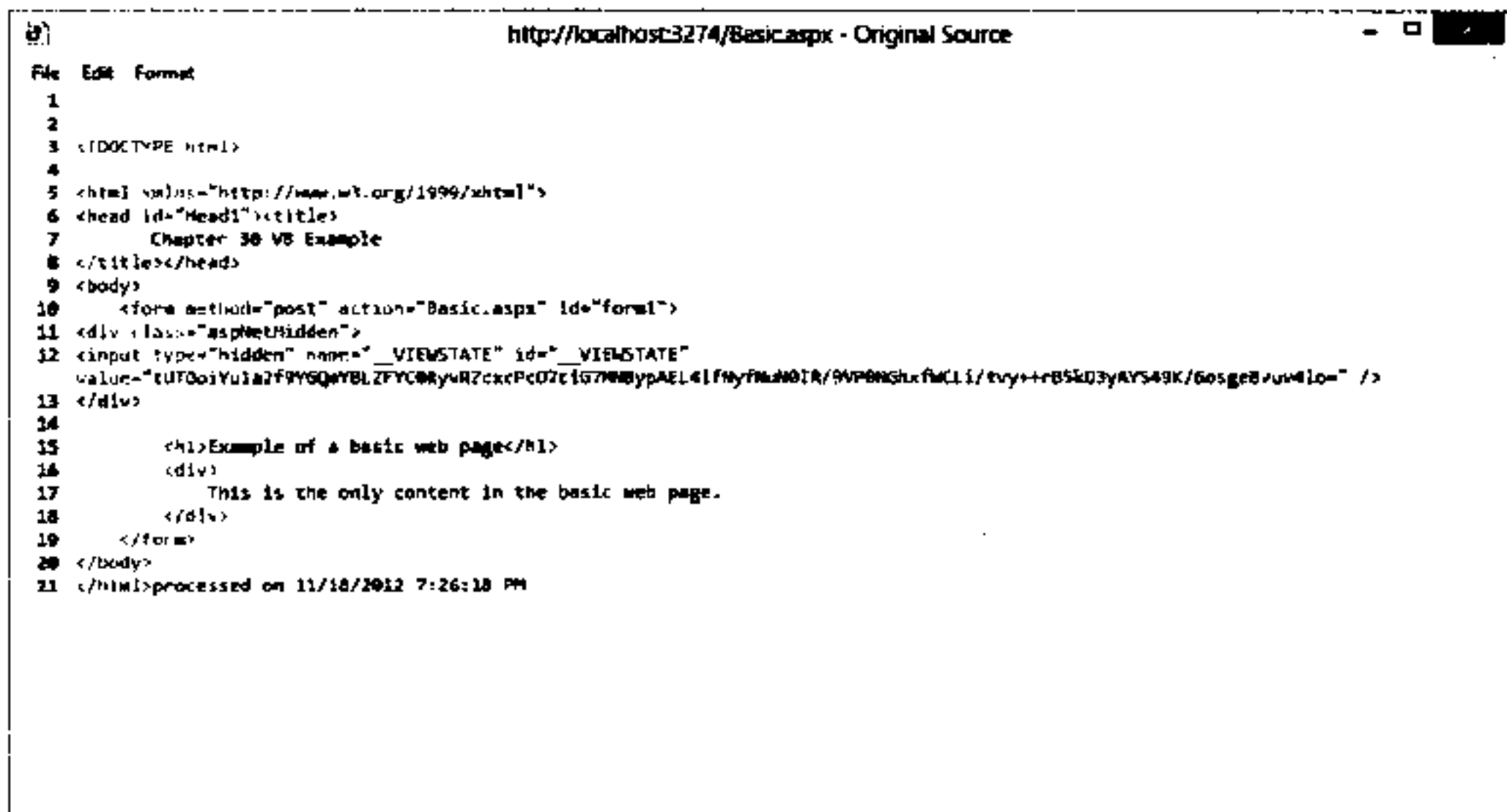


图 30-3

30.3 HttpHandler

HttpHandler 不同于 HttpModule，这不仅因为它们在请求处理管道中的位置不同(如图 30-1 和图 30-2 所示)，而且因为它们必须映射为特定的文件扩展名。处理程序是入站 HTTP 请求的最后一站，也是请求处理管道的终点。无论请求的是 ASPX 页面、HTML、纯文本还是图像，处理程序都负责处理所请求的内容。

在应用程序中，利用 HttpHandler 处理通常使用标准 ASPNET 页面提供的内容(例如动态文件下载请求)是很不错的主意，因为这样可以编写特定的处理程序，以减少标准 ASPNET 处理程序的总体开销。

本节将演示使用两种不同的方式创建简单的 HttpHandler，该 HttpHandler 用于处理基于动态查询字符串数据的图像：

- 首先，使用.ashx 文件扩展名创建一个 HttpHandler，该 HttpHandler 可用于快速开始操作，并且不需要任何服务器配置。
- 然后，使用 IIS 将 HttpHandler 映射到定制的文件扩展名，以创建更全面定制的处理程序。

30.3.1 一般的处理程序

Visual Studio 附带了 HttpHandler 的标准模板，可以帮助我们从头开始创建 HttpHandler。要在项目中添加 HttpHandler，只需从 Add New Item 对话框中选择 Generic Handler 文件类型。图 30-4 显示了选中该文件类型的 Add New Item 对话框。



图 30-4

在把 Generic Handler 文件添加到项目中时，会添加一个扩展名为.ashx 的文件。.ashx 文件扩展名是 ASP.NET 建立的 `HttpHandler` 默认文件扩展名。由于 `HttpHandler` 必须映射为唯一的文件扩展名，因此 ASP.NET 默认使用.ashx 扩展名。这是很方便的功能，否则我们就要自己添加文件扩展名，显然这并不总是可行的任务，也不切合实际。使用 Custom Handler 文件类型有助于避免额外的配置。

注意，可以自动创建文件类型的类模块。程序清单 30-5 显示了该类(本章下载代码中的 `txthandler.ashx.cs` 文件)。

程序清单 30-5 `HttpHandler` 页面模板

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace CS
{
    /// <summary>
    /// Summary description for imghandler
    /// </summary>
    public class imghandler : IHttpHandler
    {

        public void ProcessRequest(HttpContext context)
        {
            context.Response.ContentType = "text/plain";
            context.Response.Write("Hello World");
        }

        public bool IsReusable
        {
            get
            {
                return false;
            }
        }
    }
}
```

```

    {
        return false;
    }
}
}

```

注意，这个类模块实现了 `IHttpHandler` 接口，该接口需要实现 `ProcessRequest` 方法和 `IsReusable` 属性。

- `ProcessRequest` 方法用于处理入站的 HTTP 请求。在默认情况下，类模块会把内容类型改为纯文本，然后把“Hello World”字符串写入输出流。
- `IsReusable` 属性告诉 ASP.NET 入站的 HTTP 请求是否可重用这个 `HttpHandler` 的实例。

模板中生成的处理程序已经准备好立刻运行。在浏览器中执行该处理程序，查看会发生什么情况。注意，这个处理程序会把内容改为 `text/plain`，因此浏览器将根据如下因素，以完全不同的方式处理这个处理程序的响应：

- 浏览器的类型和版本
- 在系统中加载的、可映射为 MIME 类型的应用程序
- 操作系统和服务包的级别

根据这些因素，可以查看浏览器中返回的文本，也可以打开 Notepad 并显示文本，或者接收 IE 的 Open/Save/Cancel 命令提示。一定要理解改变 `ContentType` 报头的潜在后果。

可以继续这个例子，修改它，使其返回一个文件。下面使用处理程序返回一幅图像。图像文件 `Garden.jpg` 已包含在本例中。为此，要修改 `ProcessRequest` 方法中的代码，如程序清单 30-6 所示(本章下载代码中的 `imghandler.ashx.cs` 文件)。

程序清单 30-6 从 `HttpHandler` 输出图像

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace CS
{
    /// <summary>
    /// Summary description for imghandler
    /// </summary>
    public class imghandler : IHttpHandler
    {

        public void ProcessRequest(HttpContext context)
        {
            //Logic to retrieve the image file
            context.Response.ContentType = "image/jpeg";
            context.Response.WriteFile("Garden.jpg");
        }

        public bool IsReusable
        {

```

```

    get
    {
        return false;
    }
}
}

```

我们把 ContentType 报头改为 image/jpeg, 表示要返回一幅 JPEG 图像, 然后使用 WriteFile 方法把图像文件写入输出流。把处理程序加载到浏览器中, 处理程序就会显示图像。图 30-5 显示了得到的 Web 页面。



图 30-5

现在创建一个简单的 Web 页面, 显示图像处理程序。程序清单 30-7(本章下载代码中的文件 ShowImage.aspx)列出了该 Web 页面的 C#代码。

程序清单 30-7 给图像源使用 IHttpHandler 的 Web 页面示例

```

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="ShowImage.aspx.cs"
    Inherits="CS.ShowImage" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Generic Image Viewer</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            
        </div>
    </form>
</body>
</html>

```


这个例子非常简单，可以通过如下操作对其进行改进：把查询字符串参数传送给处理程序，并使用它们在处理程序中执行其他逻辑。

```

```

例如，使用查询字符串数据从 SQL Server 数据库中动态地检索一幅图像，并把它返回给客户端或进行某些类型验证，以确保允许请求者访问这幅图像。

30.3.2 在 IIS 中映射文件扩展名

使用.ashx 文件扩展名非常方便，但还可以给定制的文件扩展名，甚至常用的扩展名创建 HTTP 处理程序。下面使用图像处理程序的代码来演示这一点。

在 Web 项目的 App_Code 目录下创建一个新类。可以把已有的图像处理控件中的代码复制到这个类中，如程序清单 30-8 所示(本章下载代码中的 App_Code\ImgHandler.cs 文件)。

程序清单 30-8 基于类的图像处理程序

```
using System.Web;

public class ImgHandler : IHttpHandler
{
    public void ProcessRequest(HttpContext context)
    {
        //Logic to retrieve the image file
        context.Response.ContentType = "image/jpeg";
        context.Response.WriteFile("Garden.jpg");
    }

    public bool IsReusable
    {
        get
        {
            return false;
        }
    }
}
```

添加了这个类之后，就把应用程序配置为显示这个处理程序所使用的文件扩展名。为此，要在 web.config 文件中添加<handlers>部分。程序清单 30-8 显示了为图像处理程序添加的该部分。

程序清单 30-9 在 web.config 文件中添加 HttpHandler 配置信息

```
<handlers>
  <add name="ImageHandler" verb="*" path="ImageHandler.img" type="ImgHandler,
    App_Code" />
</handlers>
```

在这个配置部分，指示应用程序使用 ImageHandler 类处理对 ImageHandler.img 的入站请求。还可以为路径指定通配符。为路径指定*.img 表示让应用程序使用 ImageHandler 类处理文件扩展名为.img 的所有请求。给 verb 指定*表示要使用该处理程序处理对应用程序的所有请求。

与 HttpModule 一样，如果使用 IIS 6 运行 Web 应用程序，也需要在应用程序配置文件的

<system.web>配置部分添加<handlers>配置部分。在这个部分添加处理程序配置时,也需要包含 name 特性:

```
<add verb="*" path="ImageHandler.img" type="ImgHandler, App_Code" />
```

把 ImageHandler.img 文件加载到浏览器中,就会处理图像。图 30-6 显示了结果。注意,浏览器地址栏中的路径会直接指向 ImageHandler.img 文件。



图 30-6

30.4 本章小结

本章介绍了创建模块的许多方式,模块可以与 ASP.NET 请求处理管道交互。本章首先讨论了 HttpModule,利用它可以直接进入 ASP.NET 页面处理管道。提供给 HttpModule 的事件对应用程序的定制有非常大的功能和灵活性。

本章还介绍了 HttpHandler,处理程序可以完全跳过 ASP.NET 页面处理管道,对架构如何处理请求的数据有完全的控制权。本章讨论了如何创建自己的图像处理程序,再把图像处理程序映射到任意文件或文件扩展名。使用 ASP.NET 的这些功能可以在自己的应用程序中创建功能,控制 ASP.NET 使用的标准页面处理过程。

第 31 章

异步通信

本章要点

- 异步的概念和使用原因
- 在 ASP.NET 中使用异步/等待模式

在建立 Web 应用程序时,最重要的考虑是应用程序开始有较大的通信量时,伸缩得如何?消耗大量服务器资源的应用程序不会伸缩得很好,站点可能不响应,用户体验也很糟糕。在许多情况下,珍贵的服务器资源,例如线程,会等待执行某个基于 I/O 的调用,或者等待数据库调用的结果。这个操作会阻止线程处理可能入站的其他请求。

采用异步编程方式,可以解决大多数这类问题,确保高效地利用资源。.NET Framework 自从 .NET 1.0 开始就支持编写异步代码。这些年来,该架构中的异步功能也演变不少。

本章研究这些异步方式,说明如何在 Web 应用程序中使用它们,解释如何使用 .NET 4.5 引入的异步/等待模式,讨论编写糟糕的异步代码的一些缺点。最后介绍如何调整 Web 服务器,以提高 Web 应用程序的并发性。

31.1 异步编程

本章首先简介需要编写异步代码的原因及其面临的挑战。接着说明在 ASP.NET 中编写异步代码的方式和场合,还要描述在 Web 应用程序中编写异步代码的一些缺点,这不同于在传统桌面应用程序中编写异步代码。

31.1.1 使用异步方式的原因

Web 服务器中用于处理请求的线程是有限的。这催生了编写异步代码的需求,这样应用程序就可以有效地使用这些线程。只要请求进入 ASP.NET,就从 CLR 线程池中分配新的线程来处理请求。如果应用程序用同步方式编写,线程就会忙于处理这个请求。如果处理过程的一部分需要等待数据库调用的结果,这个线程就会等待结果,而不能用于处理其他请求。现在,如果线程池中的所有线

程都被挂起，就不会处理对这个网站的其他请求，用户就只能等待。这最终会导致站点不响应。

这时，以异步方式编写应用程序的好处就会显现出来。如果同一个应用程序是用异步方式编写的，在线程等待数据库调用的结果时，线程就会由 ASP.NET 返回给线程池，而数据库调用在另一个线程上进行。返回到线程池中的线程现在就可用于处理应用程序的其他入站请求。数据库调用结束后，就会通知应用程序，请求就处理完了。

31.1.2 编写异步代码的场合

在 Web 应用程序中何时编写异步代码的考虑，与传统的桌面应用程序大不相同。

- 编写桌面应用程序时，编写异步代码的目的是不阻止 UI 线程(这样应用程序总是会响应用户)，并在另一个线程上执行任何后台操作，例如计算或数据库调用。桌面应用程序不会受限于应用程序可用的线程数。
- 给 Web 应用程序编写异步代码时，限制因素是线程池中的线程数。给 Web 应用程序编写异步代码的目的是最大化线程可以处理的请求数。

给 Web 应用程序编写异步代码的一条基本规则是，只要应用程序需要进行基于 I/O 的网络调用，就编写异步代码。这包括需要访问磁盘上的文件，或者需要调用 Web 服务，或者需要访问数据库服务器。在这些情况下，进行 I/O 调用在 Windows 中进行了高度优化，允许应用程序在等待这个调用的结果时，ASP.NET 重新分配线程去处理其他请求。

编写异步代码的更现代的一种情形是应用程序依赖长时间运行的请求。此时，服务器和客户端之间有长时间打开的连接，发生某个事件时，数据会在服务器和客户端之间传输。这一般用于长时间的投票和 HTTP 流传输过程。不交换任何数据时，就不希望线程处理请求，因为没有数据，但一旦发生某个事件，就希望有线程处理它。第 26 章详细介绍了这些情形。

31.1.3 异步简史

自 .NET Framework 异步编程的起始阶段，处理服务器资源就总是比较困难。至于编写异步代码，开发人员必须与提高了的复杂性进行斗争——编写异步代码在概念上的系统开销。这使异步代码的编写和调试很困难，因为必须处理线程、回调，在请求切换到其他线程时同步状态。.NET Framework 引入了几个模式，尝试为开发人员解决这些问题。在 .NET Framework 4.5 中，推出了一种新模式——异步/等待，它能简化异步代码的编写和调试。下面简要介绍所有这些模式。

1. 早期的异步模式

在 .NET 1.0 中，架构引入了一种模式——异步编程模型(Asynchronous Programming Model, APM)，也称为 *IAsyncResult* 模式。在这个模型中，异步操作需要 *Begin* 和 *End* 方法(例如，*BeginWrite* 和 *EndWrite* 用于异步写入操作)。

在 .NET 2.0 中，引入了基于事件的异步模式(Event-based Asynchronous Pattern, EAP)。这个模式需要一个带 *Async* 后缀的方法，还需要一个或多个事件、事件处理委托类型，以及 *EventArgs* 派生类型。程序清单 31-1 列出了一些同步代码，这些代码进行外部调用，以下载内容并计算下载的总字节数。

程序清单 31-1 同步代码

```
public int RSSFeedLength(IList<Uri> uris)
{
    int total = 0;
    foreach(var uri in uris)
    {
        var data = new WebClient().DownloadData(uri);
        total += data.Length;
    }
    return total;
}
```

程序清单 31-2 说明了上述代码在 EAP 异步模式下的情形。如程序清单 31-2 所示，代码变得复杂多了。必须退出 foreach 循环，才能手工获得枚举器。代码看起来是递归的，而不是迭代的，这样理解和调试代码也就比较困难了。

程序清单 31-2 使用了 EAP 的异步代码

```
public void RSSFeedLengthAsync(IList<Uri> uris)
{
    RSSFeedLengthAsyncHelper(uris.GetEnumerator(), 0);
}

private void RSSFeedLengthAsyncHelper(IEnumerator<Uri> enumerator, int total)
{
    if(enumerator.MoveNext())
    {
        var client = new WebClient();
        client.DownloadDataCompleted += (sender, e) =>
        {
            RSSFeedLengthAsyncHelper(enumerator, total + e.Result.Length);
        };
        client.DownloadDataAsync(enumerator.Current);
    }
    else
    {
        enumerator.Dispose();
    }
}
```



只要是在 .NET Framework 4.0 或 4.5 环境下创建应用程序，就不应使用 EAP 或 APM 编写任何异步应用程序。

2. 任务和 TAP

在 .NET Framework 4.0 中，引入了基于任务的异步模式(Task-based Asynchronous Pattern, TAP)。这个模式使用方法表示异步操作的开始和结束。根据相应的同步方法返回 void 或 TResult 类型，TAP

方法返回 `Task` 或 `Task<TResult>`。任务是可由程序的其他部分独立执行的操作。在这种情况下，方法在语义上等价于线程，但实际却更轻型的对象，没有创建 OS 线程的系统开销。除了 TAP 之外，.NET 4.0 还引入了 `Task`，作为并行化的基本单元，并为基于 `Task Parallel` 库的管理任务提供丰富支持。本章后面将详细介绍任务。

3. 异步/等待

.NET Framework 4.5 引入了异步/等待模式。`async` 和 `await` 关键字是异步编程的核心。使用这两个关键字，就可以像编写同步代码那样编写异步代码。这个模式基于 TAP，为编写异步代码提供了很好的编程范式。

程序清单 31-3 是程序清单 31-1 中的同步代码使用异步/等待模式后的情形。

程序清单 31-3 使用异步/等待模式的异步代码

```
public async Task<int> RSSFeedLengthAsync(IList<Uri> uris)
{
    int total = 0;
    foreach (var uri in uris)
    {
        var data = await new WebClient().DownloadDataTaskAsync(uri);
        total += data.Length;
    }
    return total;
}
```

在程序清单 31-3 中，首先注意，它与程序清单 31-1 中的同步代码很相似。控制流没有改变，也没有回调。异步是通过返回 `Task<int>` 获得的。`Task` 表示当前工作的某个单元。这个方法的调用者可以使用返回的任务，检查任务是否完成，并从任务中获得结果。

方法还被标记为异步，这告诉编译器，以特定的方式编译该方法，允许一部分代码转变为回调，并自动创建一个 `Task<int>` 对象，以便在其中返回结果。`DownloadDataTaskAsync` 方法返回一个 `Task<byte[]>` 对象，一旦数据可用，它就完成了。在这个例子中不希望执行任何操作，除非有数据，所以等待任务的完成。

`await` 似乎进行了阻止调用，但实际上 `await` 把该方法的其他代码放在一个回调中，并立即返回。等待的任务返回时，就调用回调方法，恢复方法的执行。

执行到返回语句时，代码会挂起、恢复执行几次(因为有等待调用)，最后把 `Task<int>` 对象返回给调用者。调用者可以等待任务执行完毕，获得结果。

31.2 ASP.NET 中的异步

如前所述，.NET Framework 已演变为支持异步编程。ASP.NET 中的异步支持也以相同的速度演变。

异步可以在两个地方获得——页面生命周期和应用程序生命周期。在处理页面时，或者请求在 ASP.NET 管道中处理时，开发人员可以编写异步代码。

在 .NET 4.5 中，ASP.NET 异步管道有了很大的变化，支持异步/等待和 TAP。新管道是基于任务

的，并且有了新的基类，编写异步页面、处理程序或模块时，可以从这些基类中派生。在学习如何在 ASP.NET 中使用异步/等待模式前，下面先讨论如何在 ASP.NET 中管理线程。

31.2.1 线程池

线程是 Web 服务器上宝贵的有限资源，所以在构建应用程序的结构时，应明智地使用它们。一个请求在进入 Web 服务器时，会在 IIS 的集成模式下走过下述流程：HTTP.sys 是入口的第一点。HTTP.sys 是一个核心级别的驱动程序，它在 IIS 监听的 I/O 完成端口上传递请求。IIS 把请求交付给线程池中的一个线程，并调用 ASP.NET。ASP.NET 开始在 CLR 线程池中处理这个线程。请求处理完，需要把响应传送回客户端时，响应就从 CLR 线程池切换到 IIS 线程池中的线程。

该流程以这种方式设计的原因是使系统的可伸缩性和适应性更高。例如，请求从核心模式(HTTP.sys)切换到用户模式(IIS)，这样 HTTP.sys 就可以处理更多的请求，把它们交给其他监听器，这些监听器可能不是 IIS。如果请求在某个线程上处理，请求上的死锁或错误就会使整台服务器上的 HTTP 崩溃。因此，即使为上下文切换付出了代价，使核心更可靠的好处也远远多于线程切换的代价。

把请求从 IIS 线程池切换到 ASP.NET 线程池的另一个原因是给 IIS 处理静态和动态请求的情形提高性能。这是 Web 服务器的最常见情形。静态文件在本地缓存，由 IIS 静态文件处理程序来处理(请求根本不进入 ASP.NET)。这也允许 IIS 处理其他请求，再交给其他监听器(不是 ASP.NET)，与 HTTP.sys 的情形一样。

处理完请求，响应准备好发送后，响应就从 CLR 线程池的线程切换到 IIS 线程池中。这么做是因为如果客户端的网络带宽很窄，不希望发送响应时阻止 CLR 线程。CLR 线程应返回，处理更多的请求。

这些线程池也包含数量有限的线程。IIS 线程池中的线程有 256 个，CLR 线程池在每个处理器中有 100 个线程。所以可以想象，如果应用程序是同步运行的，并且因为等待数据库调用而挂起线程，那么线程不久就会用完，不能处理用户的更多线程，网站也会变慢，用户体验就很糟糕。稍后介绍在 ASP.NET 中编写异步代码的几种方式。图 31-1 是两个线程池处理请求的流程。

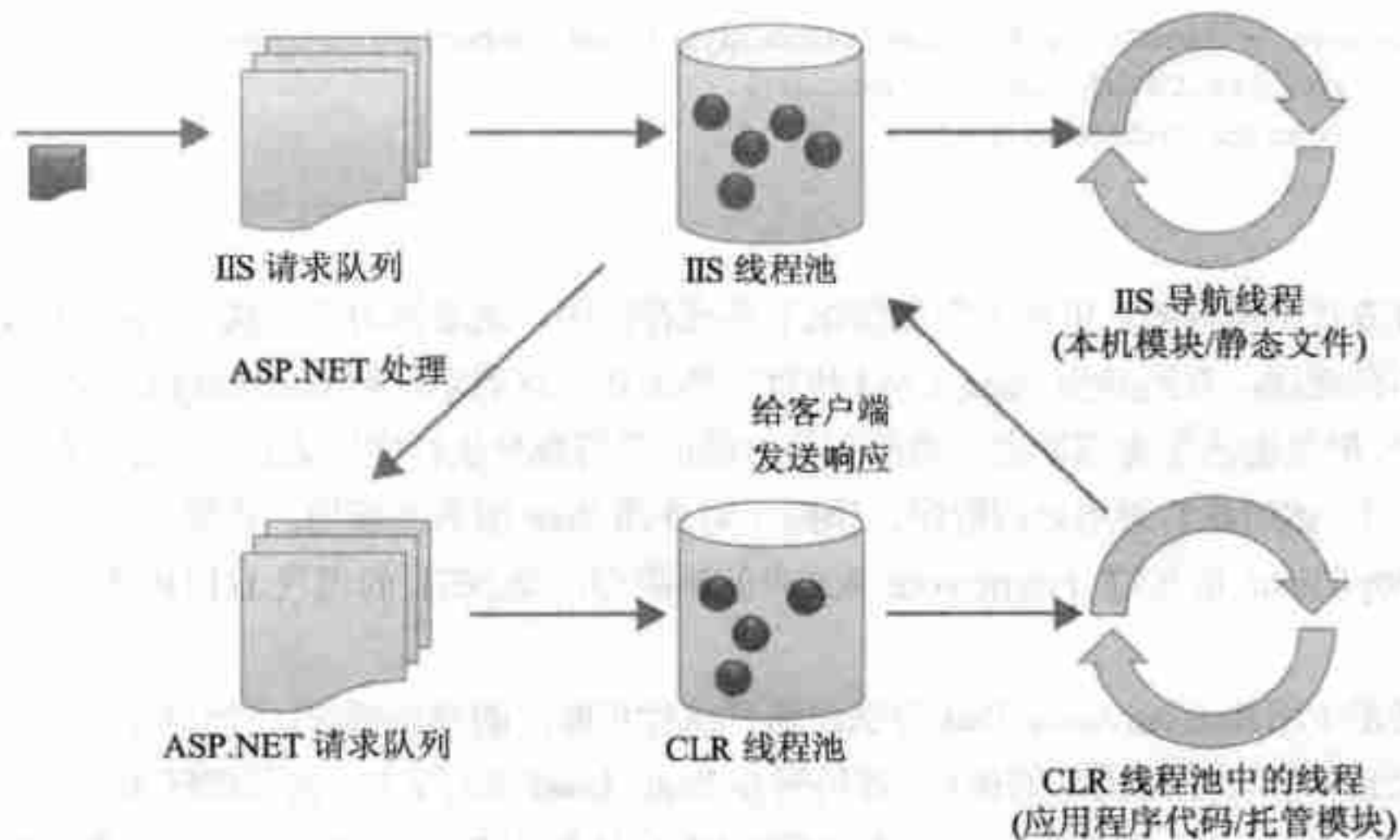


图 31-1

31.2.2 编写异步代码

如前所述，.NET Framework 4.5 把异步/等待引入为编写异步应用程序的一种模式。为了在 ASP.NET 中支持这个模式，已有的异步管道发生了很大的变化。整个异步管道现在都是基于任务的。但没有变化的是可以编写异步代码的场合。仍可以使用异步页面、处理程序和模块来编写异步代码。第 26 章学习了 HTTP 处理程序和模块。本章介绍如何使用异步/等待模式编写异步处理程序和模块。

1. 异步页面

页面访问数据库或 Web 服务中的数据时，应编写异步页面。此时不应停止页面生命周期，等待数据库或 Web 服务调用的结果。如程序清单 31-4 所示设置 Page 指令，就可以告诉 ASP.NET 按异步方式处理页面。设置这个特性时，ASP.NET 会把页面的处理从同步管道切换为异步管道。

程序清单 31-4 把页面标记为异步

```
<%@ Page Async="true" %>
```

如果没有使用这个特性，但在页面中使用了异步代码，ASP.NET 就会抛出错误，指出需要把页面标记为异步。这样可以防止竞态条件的出现，因为代码是异步的，如果没有标记为异步，代码就会以同步方式运行，导致死锁或竞态条件。

之后，就可以在页面中编写异步方法。在 ASP.NET 4.5 中，可以编写异步的 Page_Load 事件，这在以前是不可能的。程序清单 31-5 说明了如何从 Page_Load 中异步调用 Web 服务。

程序清单 31-5 异步的 Page_Load

```
protected async void Page_Load(object sender, EventArgs e)
{
    using (var client = new HttpClient())
    {
        var customersJson = await client.GetStringAsync(url);

        var customers = JsonConvert.DeserializeObject<IEnumerable<Customer>>(customersJson);
        results.DataSource = customers;
        results.DataBind();
    }
}
```

尽管这种方式是可行的，但在大多数情况下不推荐使用。这是因为异步执行 Page_Load 时，页面生命周期不会继续，直到异步 Page_Load 执行完毕为止。这表示，对 GetStringAsync 方法的调用是异步进行的，但页面的生命周期会一直等到该方法的等待部分执行完毕为止。在程序清单 31-5 中，代码调用了一个 ASP.NET 通用处理程序，模拟了对外部 Web 服务的调用。还要注意，代码使用了 HttpClient，HttpClient 是 .NET Framework 4.5 中的新类型，是 .NET 的现代 HTTP 客户端，取代了 WebClient。

推荐方法是使用 RegisterAsyncTask 方法注册并执行页面上的异步操作。使用这个方法的优点是，ASP.NET 不会停止页面生命周期的执行，等待异步 Page_Load 执行完毕，而是继续页面的生命周期，直到到达页面的 PreRender 事件为止。此时，ASP.NET 会执行用 RegisterAsyncTask 注册的所有任务。

PreRender 是生命周期中的一个事件,在该事件中,会创建所有的控件,但没有显示内容,所以如果需要,就可以在这个事件中进行数据绑定。程序清单 31-6 说明了如何使用 PageAsyncTask 通过 ASP.NET 定义 Task, Task 也是 ASP.NET 4.5 引入的新类型。

程序清单 31-6 使用了 PageAsyncTask 的 RegisterAsyncTask

```
protected void Page_Load(object sender, EventArgs e)
{
    RegisterAsyncTask(new PageAsyncTask(GetCustomersAsync));
}
private async Task GetCustomersAsync()
{
    using(var client = new HttpClient())
    {
        var customersJson = await client.GetStringAsync("http://localhost:64927/Customers.ashx");

        var customers = JsonConvert.DeserializeObject<IEnumerable<Customer>>(customersJson);
        results.DataSource = customers;
        results.DataBind();
    }
}
```

2. 异步处理程序

与编写异步页面一样,使用异步/等待模式编写异步 HTTP 处理程序也有基类(HttpTaskAsyncHandler)。程序清单 31-7 说明了如何编写异步处理程序。这个例子从网站上下载内容,把它们显示在页面上。

程序清单 31-7 使用了 HttpTaskAsyncHandler 的异步处理程序

```
public class Listing31_7 : HttpTaskAsyncHandler
{
    public async override Task ProcessRequestAsync(HttpContext context)
    {
        using(var client = new HttpClient())
        {
            var bingTask = await client.GetStringAsync("http://bing.com");
            context.Response.Write(bingTask);
        }
    }
}
```

3. 异步模块

在 ASP.NET 4.5 中编写异步模块的区别是任务在模块中的注册方式。ASP.NET 4.5 中有个新类型 EventHandlerTaskAsyncHelper。这是一个辅助类,它把任务作为输入,打开异步任务的开头和结尾部分,把它们传递为异步事件的开始和结束处理程序,异步事件是要在模块中注册的事件。程序清单 31-8 说明了如何使用这个辅助类编写异步模块。

程序清单 31-8 使用了 EventHandlerTaskAsyncHelper 的异步模块

```

public void Init(HttpApplication context)
{
    EventHandlerTaskAsyncHelper helper =
        new EventHandlerTaskAsyncHelper(DownloadWeb);

    context.AddOnBeginRequestAsync(
        helper.BeginEventHandler, helper.EndEventHandler);
}

public async Task DownloadWeb(object caller, EventArgs e)
{
    using(var client = new HttpClient())
    {
        var result = await client.GetStringAsync("http://bing.com");
    }
}

```

4. 取消异步任务

读者可能就如何在 ASP.NET 应用程序中编写异步代码有好的想法。处理 Web 服务和数据库服务器等外部资源时，最常见的情形是获得响应所需的时间。因为这些调用都取决于网络访问的快慢，所以响应时间也各不相同。在一些情况下，如果服务器失败，就可能得不到响应。此时，应用程序应灵活地处理这些情形，且能轻松地处理这些情形。必须执行的常见操作是检查已有任务的状态，如果任务被取消(例如因为服务器失效)，就应采取某个定制措施。

对象调用可取消的操作(例如创建新任务)，把令牌传递给调用者，对任务执行某个操作。如果需要，该操作还可以把令牌的副本传递给其他操作。程序清单 31-9 说明了如何为运行在页面上的异步任务指定超时时间。在这个例子中，如果异步操作的时间超过了 AsyncTimeout 值，就会得到 TimeoutException 异常，该异常可以在页面上处理。程序清单 31-10 说明了如何在异步代码中处理这个异常。

程序清单 31-9 为页面上的异步处理设置超时时间

```
<%@ Page Async="true" AsyncTimeout="1" %>
```

程序清单 31-10 处理因 AsyncTimeOut 而产生的 TimeoutException 异常

```

protected void Page_Load(object sender, EventArgs e)
{
    RegisterAsyncTask(new PageAsyncTask(GetCustomersAsync));
}

private async Task GetCustomersAsync(Cancellation_token cancelToken)
{
    using (var client = new HttpClient())
    {
        var response = await
            client.GetAsync("http://localhost:64927/Common/SlowCustomers.ashx",
                cancelToken);
        var customersJson = await response.Content.ReadAsStringAsync();
    }
}

```



```

var customers = JsonConvert.DeserializeObject<IEnumerable<Customer>>(customersJson);
results.DataSource = customers;
results.DataBind();
}
}
private void Page_Error(object sender, EventArgs e)
{
    Exception exc = Server.GetLastError();
    if(exc is TimeoutException)
    {
        throw exc;
    }
}
}

```

在上面的程序清单中，如果对外部 Web 服务的调用超过了 1 秒，GetAsync 方法调用就超时了，此时上述代码就会抛出 TimeoutException 异常，该异常可以在应用程序中处理，显示一条更有意义的错误消息，或执行某个定制的错误处理操作。

31.2.3 并行

前面介绍了编写异步应用程序的优点，以及如何在 ASP.NET 中编写异步代码。编写异步代码，可以高效地利用线程，使应用程序的可伸缩性和响应性更高。异步模式还可以缩短应用程序的执行时间，这称为并行。由于硬件取得了许多进步，许多计算机和工作站都有多个核心，允许同时执行多个线程。NET Framework 从一开始，就通过线程和锁定的底层操纵实现了这个功能，但这使应用程序的并行和调试非常困难。

在 .NET Framework 4.0 中，引入了任务来简化并行应用程序的开发，这样就可以编写高效、可伸缩的并行代码，而无须处理底层线程。该架构引入了各种类型的库，以简化在应用程序中编写并行代码的过程。这些库包括 Task Parallel Library 和 Parallel LINQ，Task Parallel Library 简化了循环基本元素的并行操作。

本节介绍使用任务并行化 ASP.NET 应用程序的一些基本技术。任务表示可以独立执行的工作单元。可以合并任务、构造任务、等待任务完成等。本节用例子说明如何构建任务，实现并行。

程序清单 31-7 中的例子等待一项下载网站内容的任务。现在假定有 3 个这样的任务在运行，且希望它们能并行运行。程序清单 31-11 说明了具体的做法。Task.WhenAll 会并行执行所有的任务，并等待所有任务都执行完毕。

程序清单 31-11 Task.WhenAll 并行地执行任务

```

public async override Task ProcessRequestAsync(HttpContext context)
{
    using(var client = new HttpClient())
    {
        var bingTask = client.GetStringAsync(uri);
        var microsoftTask = client.GetStringAsync(uri);
        var twitterTask = client.GetStringAsync(uri);

        await Task.WhenAll(bingTask, microsoftTask, twitterTask);
    }
}

```


31.2.4 服务器配置

对于提高应用程序的可伸缩性，一个最重要的方面是调整 Web 服务器以充分利用。在 IIS 上安装 ASP.NET 时，线程池和并发性的默认配置是支持大多数常见情形，所以可能满足或不满足应用程序的需要。理解这些默认限制是什么，它们面向什么类型的应用程序是非常重要的。理解应用程序需要什么也同样重要。例如，IIS 和 ASP.NET 的默认配置不适合建立实时应用程序，因为实时应用程序需要维护长时间运行的服务器连接，且服务器需要为高并发性和高延时调用进行优化。本节介绍可以对 Web 服务器进行的一些调整，以优化应用程序的性能。

- 只要为应用程序的可伸缩性确定基准，就应使用服务器操作系统，而不是客户端 OS，例如 Windows 8 或 Windows 7，因为客户端操作系统的最大并发请求数是 10。
- 如前所述，请求到达 Web 服务器时，先由 HTTP.sys 处理，HTTP.sys 队列的默认限制是 1000。如果超过这个数字，服务器就开始用 HTTP 状态码 503 来响应。如果觉得默认限制 1000 太低，可以提高到 5000。在 IIS 中使用 IIS Manager 可以给每个应用程序池配置这个设置。
- 如果应用程序通过 HTTP 使用 Web 服务，就可能需要修改 connectionManagement/maxconnection 元素。对于 ASP.NET 应用程序，它被限制为 CPU 数的 12 倍。这表示，在 4 核处理器上，一个端点至多有 $12 * 4 = 48$ 个并发连接。在应用程序起始事件中，修改 System.Net.ServicePointManager.DefaultConnectionLimit 属性的 maxconnection，就可以改变这个设置。
- ASP.NET 还可以使用 CLR 线程池处理托管的请求。CLR 线程池的并发数也有限制。这表示，如果应用程序以异步模式运行，请求数就可以超过可用线程数。在 web.config 中设置 MaxConcurrentRequestsPerCPU，就可以控制并发限制。这个设置的默认限制是 5000，应适合于大多数应用程序。但如果建立的应用程序有很多的异步处理或长时间运行的请求，就应增加这个数字，使应用程序更好地执行。

31.2.5 使用异步模式的缺点

前面介绍了在应用程序中使用异步处理的优点，能使应用程序的可伸缩性和响应性更高。但俗话说，能力越强，责任就越大。尽管异步处理会提高应用程序的性能，但如果用得不好，就会带来灾难。异步代码编写得不好，可能导致竞态条件、线程死锁和同步问题。所有这些问题都很难调试和解决，还可能浪费开发人员的宝贵时间来进行调查。本节学习编写异步代码的一些常见问题，以及避免它们的方式。

- 如果为 Web 编写异步应用程序，就一定要考虑编写异步代码的时机。如果应用程序的某部分对 Web 服务或数据库发出网络调用，并锁定线程来等待调用的响应，就应使用异步代码，使线程不被锁定。但是，如果应用程序的某部分在进行一项计算密集型的任务，异步运行就不会带来什么好处，因为应用程序仅使用线程池中的另一个线程执行相同的操作。实际上，此时还可能为线程上下文的切换付出一定的代价。针对 Web 的异步编程与传统的桌面应用程序大不相同，与 Web 服务器相比，在传统的桌面应用程序中，线程数是没有限制的。所以在桌面应用程序中，应在后台任务中运行计算密集型的任务，这样 UI 会响应，但这不适用于 Web 应用程序。
- 在创建新任务/线程时，最好让架构创建它们，并管理线程生命周期。应用程序开发人员应使用任务 API 管理和构建任务。

- 访问 ASP.NET 内部对象(比如 Request、Context 等)是很常见的。编写异步代码时,并不知道自己在使用哪个线程。所以在尝试访问内部对象时,可能在错误的线程上访问它们。如果使用异步/等待模式,架构就会在线程上同步这些对象。
- 因为在异步处理时,请求可以由多个线程处理,所以如果使用线程本地存储,且在处理的不同阶段,请求由不同的线程处理,就有可能得不到变量的正确值。如果需要存储与线程相关的本地信息,可以使用 `HttpContext.Current`。

31.3 本章小结

本章介绍了异步处理及使用原因。当应用程序越来越流行,吸引越来越多的流量时,就需要确保它们可以很好地伸缩。在编写应用程序时,应确保它们能有效地利用服务器资源,且很好地执行。

.NET Framework 和 ASP.NET 在支持异步处理方面有了长足的进步。本章介绍了编写异步代码的不同模式。.NET Framework 4.5 引入了异步/等待模式,能使异步代码的编写类似于同步代码的编写。这两种编码方式的代码流是相似的。

当希望编写异步页面、处理程序和模块时,ASP.NET 支持异步/等待模式。在 ASP.NET 4.5 中,管道有了很大的变化,以支持异步/等待模式。我们还介绍了可以对 Web 服务器进行的一些调整,使应用程序在负载较大的情况下仍能很好地伸缩。

最后介绍了异步代码编写得不好所带来的-一些问题,以及避免这些问题的一些方法。

第 32 章

国际化应用程序的建立

本章要点

- 国际化应用程序
- 定义服务器端和客户端的区域性环境
- 使用本地和国际化的资源

开发人员通常使用自己的母语建立 Web 应用程序,随着应用程序用户群的不断扩大,开发人员发现需要将应用程序国际化。当然,最理想的情况是从一开始就建立能应对世界各地的用户的 Web 应用程序。但在许多情况下,这是不可能实现的情况,因为需要做大量额外的工作。

国际化和本地化都是建立国际化应用程序的重要过程,但这两个术语是不能互换的,它们是不同的:

- 国际化定义为开发应用程序,使之在多个区域和地区正常工作的过程。
- 本地化是给特定的区域和地区定制应用程序的过程。

ASP.NET Framework 为 Web 应用程序的国际化做了大量的工作。API 的巨大改进、服务器控件的新增功能,甚至 Visual Studio 本身都使 Web 应用程序的国际化变得非常简单。本章将介绍创建国际化的 Web 应用程序时应考虑的一些重要事项。

32.1 区域性和地区

显示在终端用户的浏览器上的 ASP.NET 页面一般在指定的区域性和地区设置下运行。在创建 ASP.NET 应用程序或页面时,在其中运行它们的区域性取决于运行应用程序的服务器上的区域性和地区设置或客户(终端用户)应用的设置。在默认情况下,ASP.NET 运行在服务器定义的区域性设置下。

世界上有多种区域性,每种区域性都有语言,并且都具有显示和使用数字、使用货币、格式化日期、按字母排序等一系列不同的定义方式。.NET Framework 使用 Request for Comments 1766 标准定义(标识语言的标记)来定义区域性和地区,该标准使用两字母代码(字母之间使用一条短线隔开)

指定语言和地区。表 32-1 列出了一些区域性定义的例子。

表 32-1

区域性代码	说 明
en-US	英语：美国
en-GB	英语：英国
en-AU	英语：澳大利亚
en-CA	英语：加拿大
fr-CA	法语：加拿大

在表 32-1 中定义了 4 种不同的区域性。这 4 种区域性有一些相似点，也有一些不同点。它们有相同的语言，即英语。因此，在每种区域性设置中都使用了相同的语言代码 en。在语言设置的后面是地区设置。虽然这 4 种区域性有相同的语言，但可以通过不同的地区设置将它们区分开来，例如 US 表示美国，GB 表示英国，AU 表示澳大利亚，CA 表示加拿大。我们知道，美式英语与英式英语略有区别。除了语言之外，日期和数字值的表示方法也有所不同。这就是把区域性的语言和地区放在一起显示的原因。

这种区别不仅存在于国家之间，有时一个国家也会使用多种语言，并且每个区域在日期和其他方面都有自己的首选项。例如，en-CA 表示使用英语的加拿大人，但加拿大人不仅使用英语，因此还使用区域性设置 fr-CA，表示使用法语的加拿大人。

32.1.1 了解区域性类型

上面的区域性定义称为特定的区域性定义，这个定义应尽可能详细——定义了语言和地区。区域性定义的另一类型是中性区域性定义。每种区域性都有一个与其关联的特定中性区域性。例如，表 32-1 中显示的英语区域性就是独立的，但它们都属于中性区域性 EN(英语)。图 32-1 显示了这些区域性类型是如何彼此联系的。

在图 32-1 中，许多特定的区域性都属于同一中性区域性，比中性区域性层次更高的是不变的区域性，它是与用户无关的区域性设置，应在网络中传送数据项(例如日期和数字)或存储数据时使用。在执行这类操作时，要使后台数据流没有用户特定的区域性设置。相反，应在应用程序的业务层和表示层应用这些设置。

在使用应用程序时还要注意中性区域性。在创建带有 Web 页面的应用程序时，Web 页面更多地依赖中性区域性，而不是某种特定的区域性。例如，如果应用程序使用西班牙语，就应使这个版本可用于所有使用西班牙语的用户，而不管这些用户来自什么地区。在许多应用程序中，西班牙语用户是来自西班牙、墨西哥还是阿根廷并不重要，如果不同地区的用户有一定的区别，就应使用特定的区域性设置。

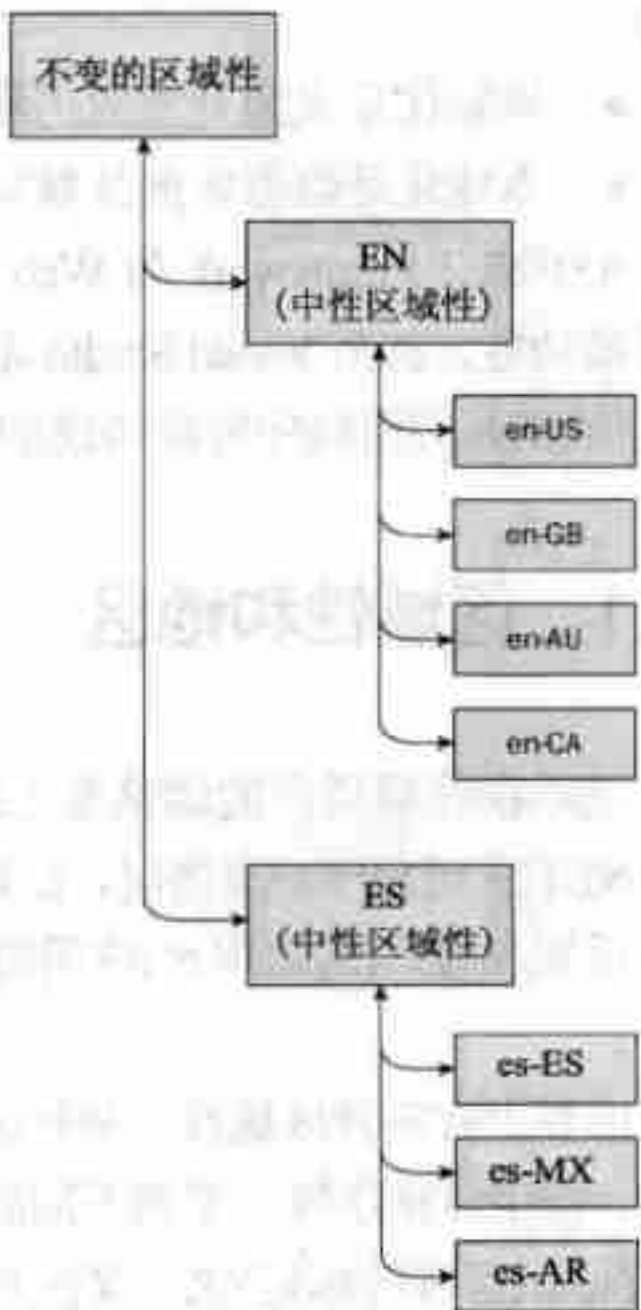


图 32-1

32.1.2 ASP.NET 线程

当终端用户请求 ASP.NET 页面时, Web 页面会在线程池的某个线程上执行。该线程有与之相关的区域性设置。我们可以通过编程获取该线程的区域性信息,再查看该区域性的特定细节,如程序清单 32-1 所示(本章下载代码中的 ViewCultureInfo.aspx.cs)。

程序清单 32-1 查看 ASP.NET 线程的区域性

```
protected void Page_Load(object sender, EventArgs e)
{
    CultureInfo ci = System.Threading.Thread.CurrentThread.CurrentCulture;
    Response.Write("<b><u>CURRENT CULTURE'S INFO</u></b>");
    Response.Write("<p><b>Culture's Name:</b> " + ci.Name.ToString() + "<br>");
    Response.Write("<b>Culture's Parent Name:</b> " + ci.Parent.Name.ToString() + "<br>");
    Response.Write("<b>Culture's Display Name:</b> " + ci.DisplayName.ToString() + "<br>");
    Response.Write("<b>Culture's English Name:</b> " + ci.EnglishName.ToString() + "<br>");
    Response.Write("<b>Culture's Native Name:</b> " + ci.NativeName.ToString() + "<br>");
    Response.Write("<b>Culture's Three Letter ISO Name:</b> " +
        ci.Parent.ThreeLetterISOLanguageName.ToString() + "<br>");
    Response.Write("<b>Calendar Type:</b> " + ci.Calendar.ToString() + "</p>");
}
```

Page_Load 事件中的这段代码检查 CurrentCulture 属性。可以把这个属性的值放在 CultureInfo 对象中。要获得该对象,应在 Web 页面中导入 System.Globalization 名称空间。CultureInfo 对象包含许多属性,它们提供了特定的区域性信息。下面的项显示在一系列简单的 Response.Write 语句中,列出了可用的信息。运行这个页面会得到如图 32-2 所示的结果。

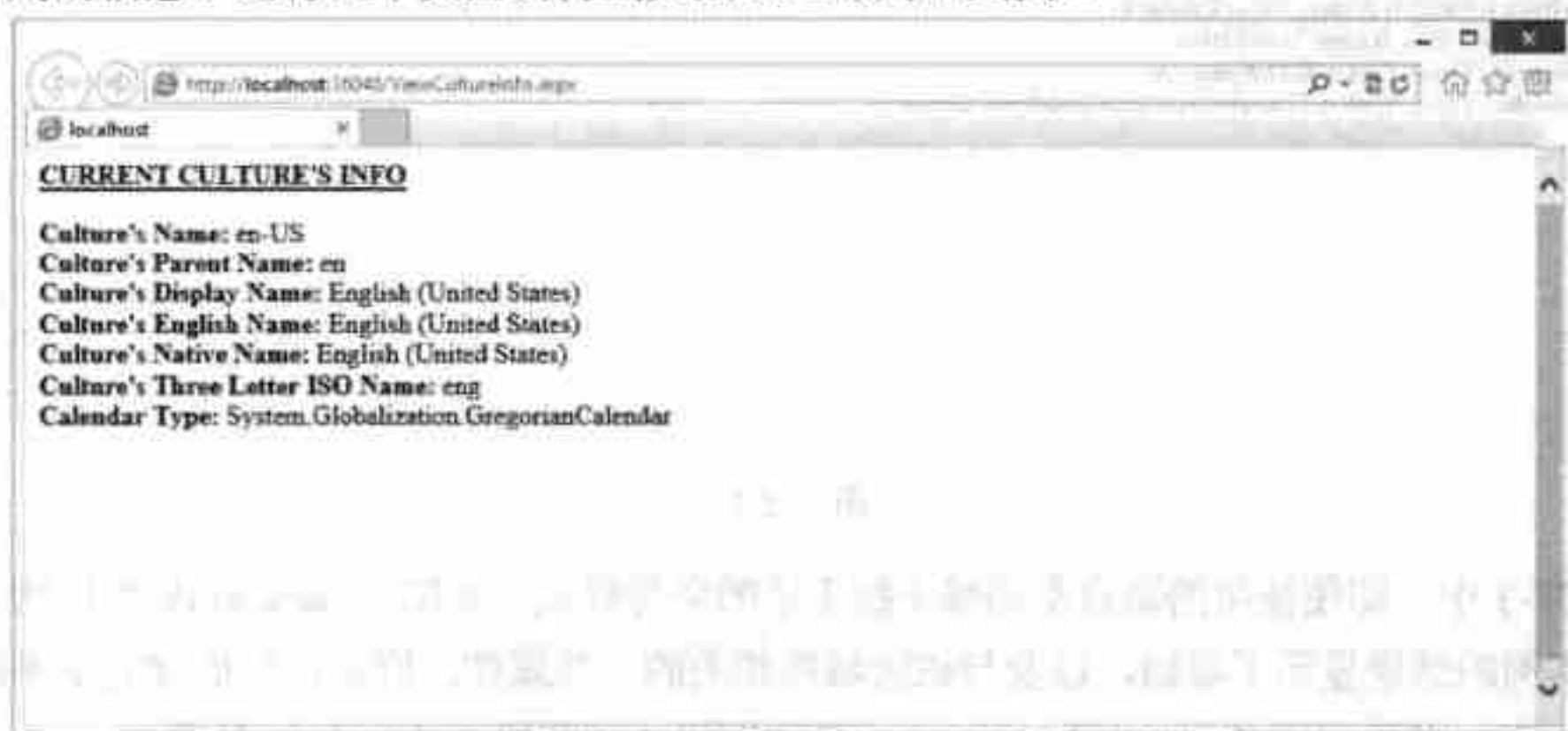


图 32-2

在图 32-2 中, en-US 区域性是 ASP.NET 线程使用的默认设置。另外,使用 CultureInfo 对象可以获取关于区域性的许多其他描述性信息。

通过实例化 CultureInfo 对象,总是可以改变重载方法的线程区域性设置,如程序清单 32-2 所示(本章下载代码中的 ChangeCulture.aspx.cs)。

程序清单 32-2 使用 CultureInfo 对象改变线程的区域性设置

```
protected void Page_Load(object sender, EventArgs e)
```



```
{
    System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("th-TH");
    CultureInfo ci = System.Threading.Thread.CurrentThread.CurrentCulture;
    Response.Write("<b><u>CURRENT CULTURE'S INFO</u></b>");
    Response.Write("<p><b>Culture's Name:</b> " + ci.Name.ToString() + "<br>");
    Response.Write("<b>Culture's Parent Name:</b> " + ci.Parent.Name.ToString() + "<br>");
    Response.Write("<b>Culture's Display Name:</b> " + ci.DisplayName.ToString() + "<br>");
    Response.Write("<b>Culture's English Name:</b> " + ci.EnglishName.ToString() + "<br>");
    Response.Write("<b>Culture's Native Name:</b> " + ci.NativeName.ToString() + "<br>");
    Response.Write("<b>Culture's Three Letter ISO Name:</b> " +
        ci.Parent.ThreeLetterISOLanguageName.ToString() + "<br>");
    Response.Write("<b>Calendar Type:</b> " + ci.Calendar.ToString() + "</p>");
}
```

在这个例子中，只添加了一行代码，将 `CultureInfo` 对象的新实例赋予 ASP.NET 正在执行的线程的 `CultureInfo` 属性：

```
System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("th-TH")
```

区域性设置可以让 `CultureInfo` 对象定义要使用的区域性。本例中指定的是泰语，生成的结果如图 32-3 所示。

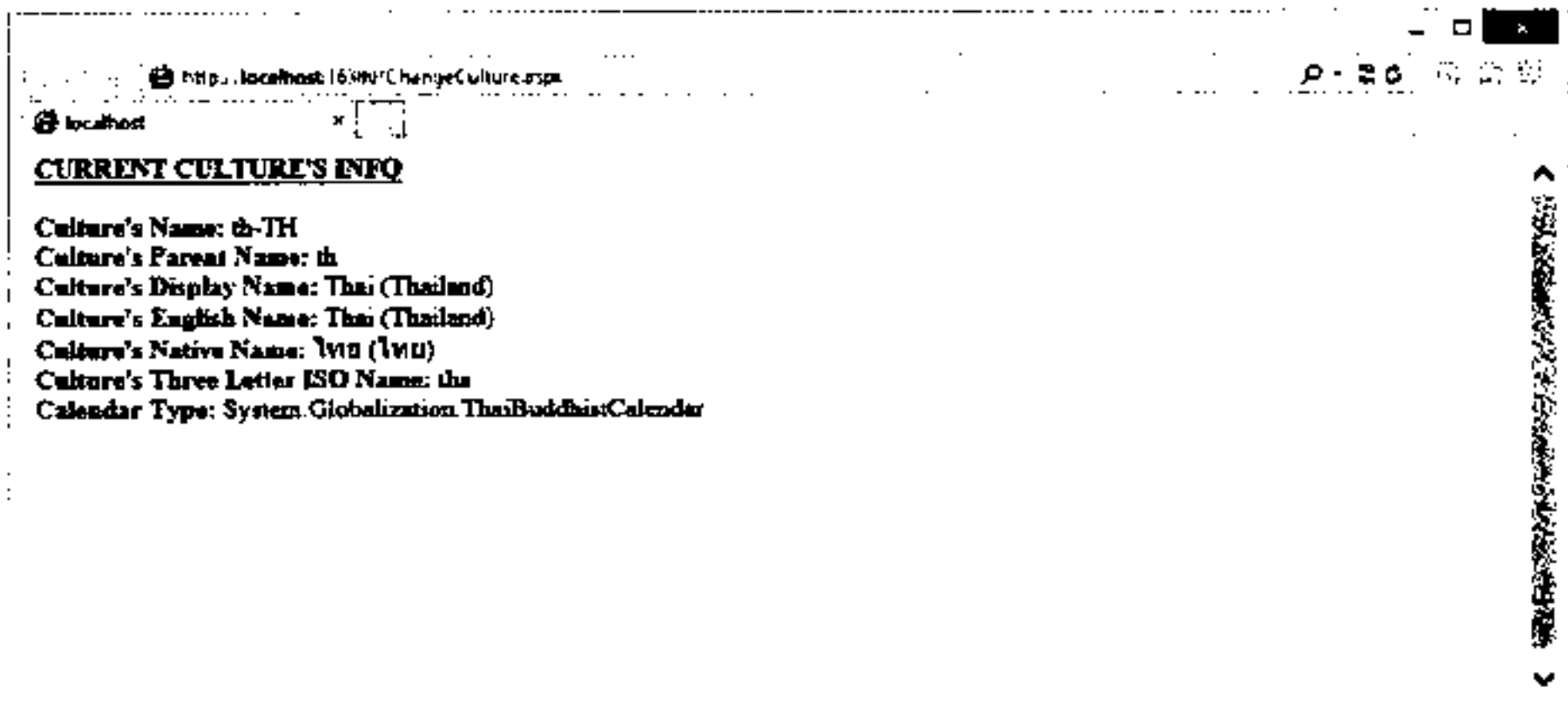


图 32-3

在图 32-3 中，即使使用的语言不是基于拉丁语的字母样式，.NET Framework 也仍然会提供它的原名。本例的结果显示了泰语，以及与该区域性相关的一些属性，例如，日历就完全不同于西欧和美国。注意，必须引用 `System.Globalization` 名称空间才能获得 `CultureInfo` 对象。

32.1.3 服务器端的区域性声明

ASP.NET 能很方便地定义整个 ASP.NET 应用程序使用的区域性，或者定义应用程序中的某个特定页面使用的区域性。借助相应的配置文件，可以指定任意 ASP.NET 应用程序的区域性。在 ASP.NET 的默认安装中，没有指定区域性，通过查看 ASP.NET 4.5 的 CONFIG 文件夹(C:\WINDOWS\Microsoft.NET\Framework\v4.0.30319\CONFIG)中的全局 `web.config.comments` 文件(用于文档编制)可以了解这一点。在 `web.config.comments` 文件中，可以看到配置文档的 `<globalization>` 部分，如程序清单 32-3 所示。

程序清单 32-3 web.config.comments 文件中的<globalization>部分

```
<globalization requestEncoding="utf-8" responseEncoding="utf-8" fileEncoding=""
  culture="" uiCulture="" enableClientBasedCulture="false"
  responseHeaderEncoding="utf-8" resourceProviderFactoryType=""
  enableBestFitResponseEncoding="false" />
```

注意粗体显示的两个特性：**culture** 和 **uiCulture**。**culture** 特性可以定义用于处理入站请求的区域性，而 **uiCulture** 特性可以定义用于处理应用程序中任意资源文件的默认区域性(有关这两个特性的用法，详见本章后面的内容)。

在程序清单 32-3 的配置声明中，没有为区域性设置指定任何内容。在指定服务器的区域性时，一种方法是在 CONFIG 文件夹中的 web.config 文件的服务器版本内定义区域性，从而该服务器上的所有 ASP.NET 4.5 应用程序都使用该区域性设置。另一种方法是在应用程序的 web.config 文件中指定这些设置，如程序清单 32-4 所示。

程序清单 32-4 在 web.config 文件中定义<globalization>部分

```
<configuration>
  <system.web>

    <globalization culture="ru-RU" uiCulture="ru-RU" />

  </system.web>
</configuration>
```

在本例中，为这个 ASP.NET 应用程序建立的区域性是俄语。除了在整台服务器或整个应用程序级别设置区域性之外，还可以在页面级别设置区域性，如程序清单 32-5 所示。

程序清单 32-5 使用@Page 指令在页面级别定义区域性

```
<%@ Page Language="C#" UICulture="ru-RU" Culture="ru-RU" %>
```

这个例子为页面上的所有内容应用俄语和区域性设置。这个操作使用了@Page 指令和页面上一个简单的日历控件，页面 Calendar.aspx 包含在项目中，演示了如何修改页面级的区域性信息。图 32-4 显示了日历输出。

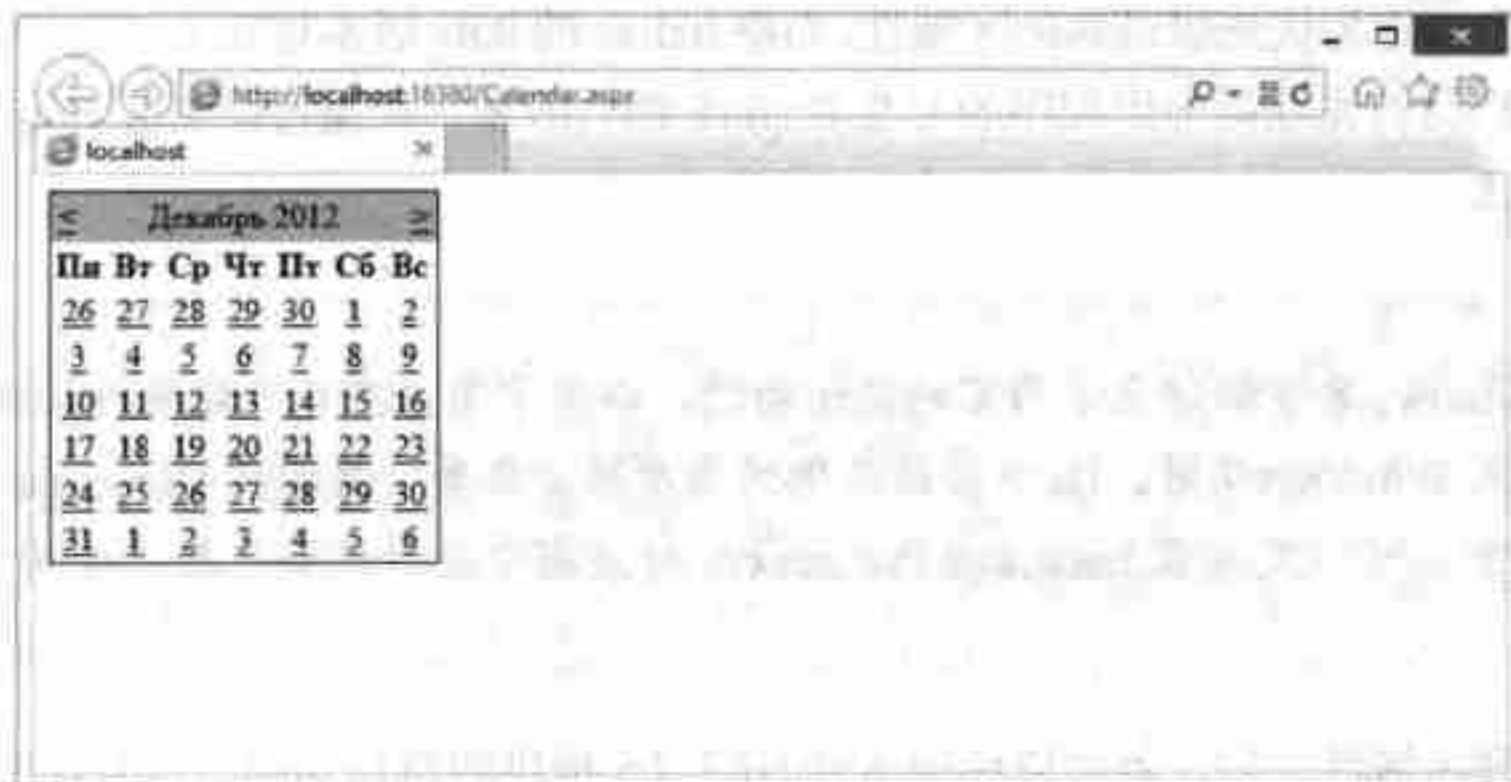


图 32-4

32.1.4 客户端的区域性声明

除了使用服务器端的设置定义 ASP.NET 页面的区域性之外,还可以使用客户在浏览器中指定的首选项定义区域性。

终端用户安装 Internet Explorer(IE)或其他浏览器时,可以按特定的顺序选择首选区域性(假设选择多个区域性首选项)。要在 Windows 8 的 IE 中查看这个过程,可以从 IE 的菜单中选择 Tools | Internet Options 命令。在 General 选项卡的底部有 Languages 按钮。单击该按钮,就会打开 Language Preference 对话框,在该对话框中,单击 Set Language Preferences 按钮,打开控制面板中的 Language Preferences 部分,如图 32-5 所示。

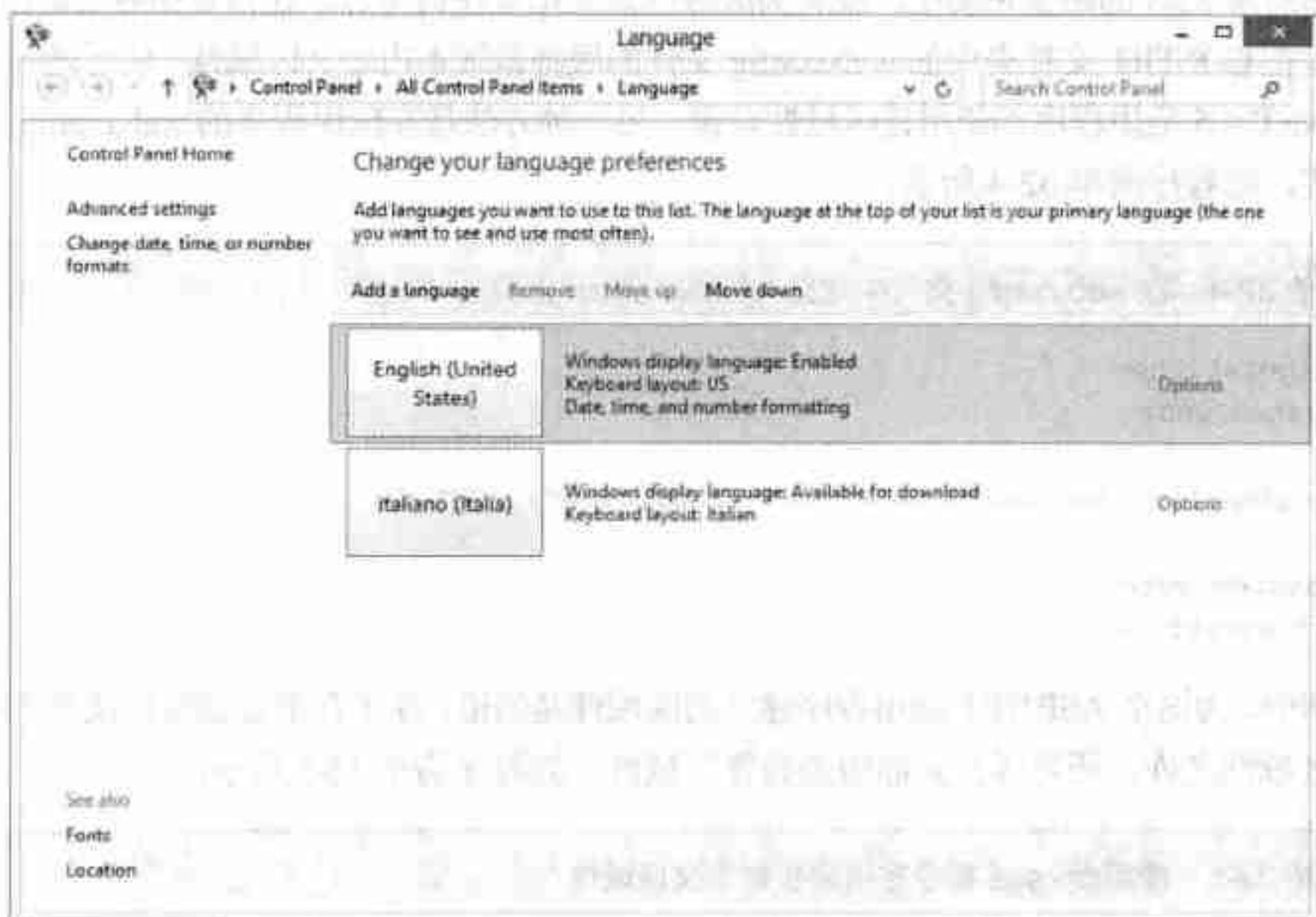


图 32-5

在这个界面中,可以看到两种可用的语言。要在类别中添加其他语言,可以选择 Add a Language,再从列表中下载对应的语言。如果语言有多个区域变体,可以选择需要的区域变体。语言和区域变体的组合就构成了区域性。添加了需要的区域性后,就可以指定使用它们的顺序。在图 32-5 中,将 English 的 US 版本指定为优先级最高的区域性,而将 Italian 的 Italy 版本指定为优先级次高的区域性。使用该设置的用户会首先获得应用程序的 U.S. English 语言版本,如果没有 U.S. English 语言版本,就提供 Italian 版本。



Windows 8 重新建立了语言的显示功能,以便于用户查找、安装和切换不同的语言。使用 Windows 7 时,IE 中当前可用的显示语言会直接显示在 Language Preferences 对话框中。可以直接在 Language Preferences 对话框中选择区域性及其使用顺序。

终端用户选择区域性之后,就可以使用 ASP.NET 4.5 提供的自动功能。没有在配置文件中或从 @Page 指令中指定某个区域性,ASP.NET 会自动选择请求页面的终端用户提供的区域性。使用 auto

关键字实现该操作, 如程序清单 32-6 所示。

程序清单 32-6 把区域性改为终端用户的首选区域性

```
<%@ Page Language="C#" UICulture="auto" Culture="auto" %>
```

在页面上添加上述构造代码后, 就会使用请求者指定的首选区域性显示日期、日历和数字。但如果翻译了资源文件中依赖区域性设置的资源, 会出现什么情况? 如果仅进行了这项翻译工作, 而不能处理可能返回给 ASP.NET 页面的所有区域性, 又该怎么办? 此时可以指定 `auto` 选项, 如果 ASP.NET 找不到用户的区域性设置, 那么还可以额外指定备用选项, 具体用法如程序清单 32-7 所示。

程序清单 32-7 通过 auto 选项提供备用区域性

```
<%@ Page Language="C#" UICulture="auto:en-US" Culture="auto:en-US" %>
```

在这个例子中, 使用了自动检测功能。如果没有提供终端用户的首选区域性, 就使用 `en-US`。

32.1.5 翻译值和行为

在国际化 ASP.NET 应用程序的过程中, 有许多项的操作方式与创建未国际化的应用程序时的方式不同, 包括日期和货币的表示方式。下面就探讨这个问题。

1. 了解日期的不同表示方式

不同的区域性以完全不同的方式指定日期和时间。例如:

```
08/11/2013
```

这个日期的含义是什么? 是 2013 年 8 月 11 日还是 2013 年 11 月 8 日? 在数据库或其他类型的后台系统中存储日期/时间值时, 为避免出错, 应总是为所有的项使用同一区域性(或不变的区域性)。业务逻辑层或表示层应可以转换这些项, 以供终端用户使用。

如前所述, 在服务器端或 `@Page` 指令中设置区域性, ASP.NET 就可以完成这些转换。也可以把新的区域性赋予正在运行 ASP.NET 的线程, 程序清单 32-8 中的页面代码修改当前线程的区域性, 以不同的区域性显示日期/时间数据(本章下载代码中的 `DateAndTime.aspx.cs`)。

程序清单 32-8 在不同的区域性中使用日期/时间值

```
using System;
using System.Globalization;

namespace Chapter32CS
{
    public partial class DateAndTime : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            DateTime dt = new DateTime(2013, 8, 11, 11, 12, 10, 10);
            System.Threading.Thread.CurrentThread.CurrentCulture =
                new CultureInfo("en-US");
        }
    }
}
```

```

        Response.Write("<b><u>en-US</u></b><br>");
        Response.Write(dt.ToString() + "<br>");

        System.Threading.Thread.CurrentThread.CurrentCulture =
            new CultureInfo("ru-RU");
        Response.Write("<b><u>ru-RU</u></b><br>");
        Response.Write(dt.ToString() + "<br>");

        System.Threading.Thread.CurrentThread.CurrentCulture =
            new CultureInfo("fi-FI");
        Response.Write("<b><u>fi-FI</u></b><br>");
        Response.Write(dt.ToString() + "<br>");

        System.Threading.Thread.CurrentThread.CurrentCulture =
            new CultureInfo("th-TH");
        Response.Write("<b><u>th-TH</u></b><br>");
        Response.Write(dt.ToString());
    }
}

```

这段示例代码建立了一个 `DateTime` 变量，使用 4 种不同的区域性在浏览器上显示值，这段代码的运行结果如图 32-6 所示。

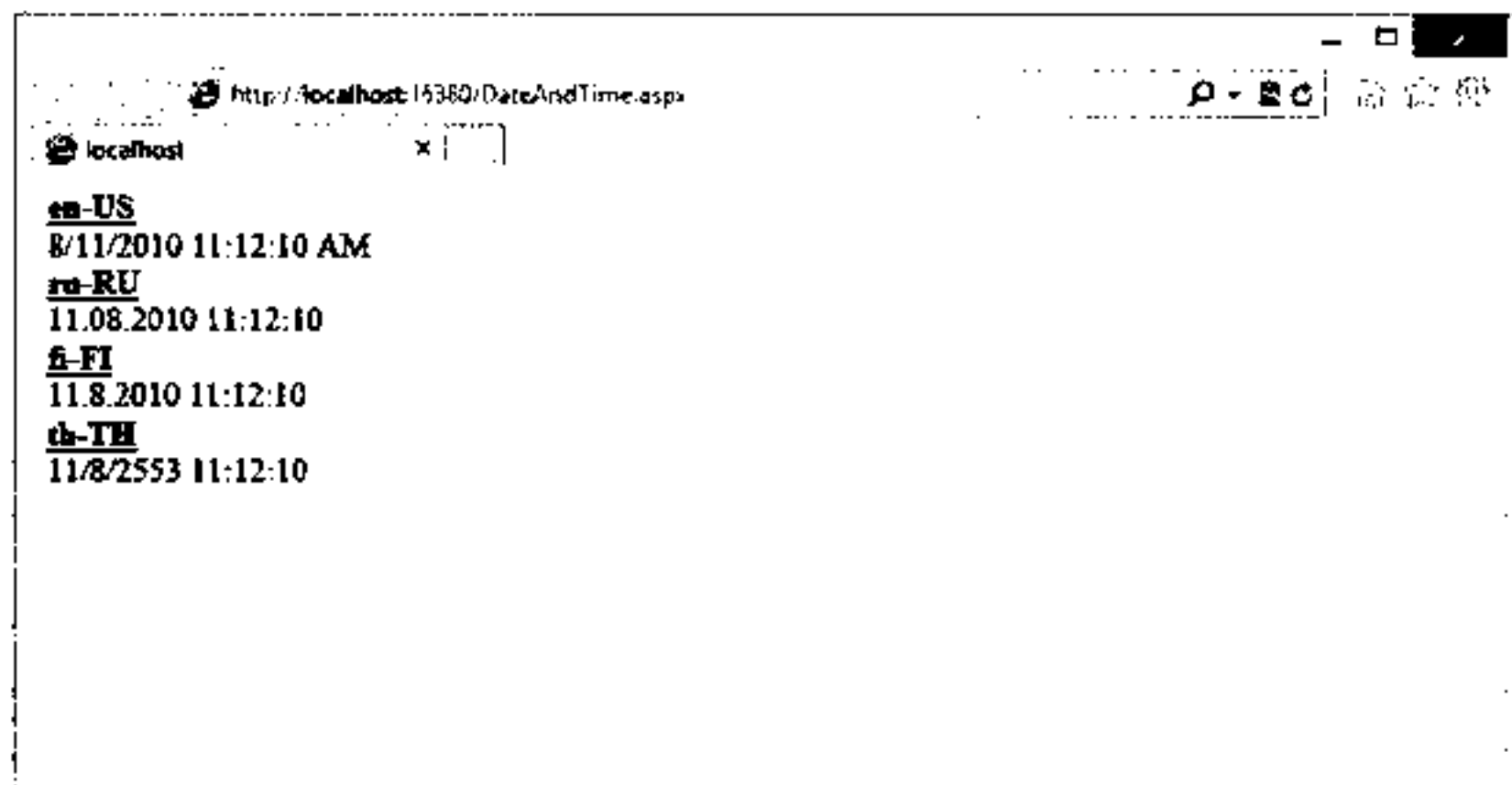


图 32-6

可以看出，用于表示日期/时间的格式彼此大不相同，Thai 区域性(th-TH)甚至使用完全不同的日历，其中把 2012 年标记为 2553。

2. 了解数字和货币表示方式的区别

除了日期/时间值之外，数字的构造方式在不同的区域性中也是不同的。但是，到底有哪些不同？这与实际数字本身没有太大的关系(尽管一些区域性使用不同的数字符号)，而与小数点分隔符以及表示千、百万等千分位的分隔符有关。例如，在美国的英语区域性(en-US)中，以如下方式表示数字：

5,123,456.00

在这个例子中，en-US 区域性使用逗号作为千分位分隔符，使用句点表示小数点。而其他区域

性与 en-US 有很大的不同。程序清单 32-9 列出了使用其他区域性表示数字的例子(本章下载代码中的 Numbers.aspx.cs)。

程序清单 32-9 在不同的区域性中使用数字

```
using System;
using System.Globalization;

namespace Chapter32CS
{
    public partial class Numbers : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            double myNumber = 5123456.00;
            System.Threading.Thread.CurrentThread.CurrentCulture =
                new CultureInfo("en-US");
            Response.Write("<b><u>en-US</u></b><br>");
            Response.Write(myNumber.ToString("n") + "<br>");

            System.Threading.Thread.CurrentThread.CurrentCulture =
                new CultureInfo("vi-VN");
            Response.Write("<b><u>vi-VN</u></b><br>");
            Response.Write(myNumber.ToString("n") + "<br>");

            System.Threading.Thread.CurrentThread.CurrentCulture =
                new CultureInfo("fi-FI");
            Response.Write("<b><u>fi-FI</u></b><br>");
            Response.Write(myNumber.ToString("n") + "<br>");

            System.Threading.Thread.CurrentThread.CurrentCulture =
                new CultureInfo("de-CH");
            Response.Write("<b><u>de-CH</u></b><br>");
            Response.Write(myNumber.ToString("n"));
        }
    }
}
```

运行这个例子,得到的结果如图 32-7 所示。



图 32-7

在这个例子中,其他区域性表示数字的格式与 en-US 区域性完全不同。图 32-7 中的第二种区域

性——vi-VN(越南语)构造数字的方式与 en-US 区域性正好相反。越南语区域性使用句点表示千分位分隔符，使用逗号表示小数点。芬兰语区域性使用空格表示千分位分隔符，使用逗号表示小数点。而瑞士的德语区域性使用撇号表示千分位分隔符，使用句点表示小数点。因此，必须把数字“翻译”为适当的格式，从而应用程序的用户才能正确地理解所表示的数字。

使用货币时也要通过数字来表示。转换货币格式以便于终端用户理解某个项的正确值是一回事，而按照翻译基本数字的方式翻译货币的构造却是另一回事。

每种区域性都使用独特的货币符号，以指出显示的数字实际是货币值。例如，en-US 区域性使用如下格式表示货币：

\$5,123,456.00

en-US 区域性使用美元符号\$，这个符号的位置与符号本身一样重要。对于 en-US 区域性，\$符号位于货币值的前面(符号和数字的第一个字符之间没有空格)。其他区域性使用不同的符号表示货币，并且常常把货币符号放在不同的位置。修改前面的程序清单 32-9，使其把数字显示为货币。需要做的修改如程序清单 32-10 所示(本章下载代码中的 Currency.aspx.cs)。

程序清单 32-10 在不同的区域性中使用货币

```
using System;
using System.Globalization;

namespace Chapter32CS
{
    public partial class Currency : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            double myNumber = 5123456.00;
            System.Threading.Thread.CurrentThread.CurrentCulture =
                new CultureInfo("en-US");
            Response.Write("<b><u>en-US</u></b><br>");
            Response.Write(myNumber.ToString("c") + "<br>");

            System.Threading.Thread.CurrentThread.CurrentCulture =
                new CultureInfo("vi-VN");
            Response.Write("<b><u>vi-VN</u></b><br>");
            Response.Write(myNumber.ToString("c") + "<br>");

            System.Threading.Thread.CurrentThread.CurrentCulture =
                new CultureInfo("fi-FI");
            Response.Write("<b><u>fi-FI</u></b><br>");
            Response.Write(myNumber.ToString("c") + "<br>");

            System.Threading.Thread.CurrentThread.CurrentCulture =
                new CultureInfo("de-CH");
            Response.Write("<b><u>de-CH</u></b><br>");
            Response.Write(myNumber.ToString("c"));
        }
    }
}
```

运行这个例子，查看这些区域性是如何表示货币值的，结果如图 32-8 所示。



图 32-8

从图 32-8 中可以看出，在不同的区域性中，不仅数字的表示方式互不相同，货币符号本身与其相对于数字的位置也是完全不同的。

因为货币表示实际的金额，所以还必须在国际化的应用程序中显示它们。修改用于显示货币值的区域性，并不会转换金额，而只是用不同的方式显示该值。例如，在页面上把区域性指定为 `auto`，就使用用户指定的区域性。如果在包含美元的页面上显示货币值，但该页面的区域性由用户指定为 `Finnish`，货币值就不正确。该值会显示为欧元，但实际的金额并没有从美元转换为欧元。为了避免混淆，可以重写所显示货币的区域性。程序清单 32-11 说明了如何在显示货币时指定要使用的区域性。

程序清单 32-11 在显示货币时使用指定的区域性

```
double myNumber = 5123456.00;
CultureInfo usCurr = new CultureInfo("en-US");
Response.Write(myNumber.ToString("c", usCurr));
```

3. 了解不同字符串排序方法之间的区别

国际化不仅会影响数字、货币、日期/时间值的表示方式，还会影响各种编程操作。区域性设置可以应用于字符串排序等操作。大多数的区域性一般以相同的方式对字符串进行排序，但有时排序方式是有区别的。例如，程序清单 32-12(本章下载代码中的 `SortingEN.aspx.cs`)就列出了页面给字符串数据排序的代码，排序操作使用 `en-US` 区域性执行。

程序清单 32-12 在不同的区域性中进行排序

```
using System;
using System.Collections.Generic;
using System.Globalization;

namespace Chapter32CS
{
    public partial class SortingEN : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            System.Threading.Thread.CurrentThread.CurrentCulture =
```

```

        new CultureInfo("en-US");

        List<string> myList = new List<string>();

        myList.Add("Washington D.C.");
        myList.Add("Helsinki");
        myList.Add("Moscow");
        myList.Add("Warsaw");
        myList.Add("Vienna");
        myList.Add("Tokyo");

        myList.Sort();

        foreach (string item in myList)
        {
            Response.Write(item.ToString() + "<br>");
        }
    }
}

```

要使这个例子正常运行,就必须导入 `System.Collections` 和 `System.Collections.Generic` 名称空间,因为这个例子使用了 `List(Of String)` 对象。

在这个例子中,以随机方式创建了世界各国的首都的列表。然后调用泛型对象 `List(Of String)` 的 `Sort` 方法。这个排序操作根据运行 ASP.NET 线程的指定区域性的排序方式对字符串进行排序。程序清单 32-12 使用 `en-US` 区域性进行排序。这个操作的结果如图 32-9 所示。

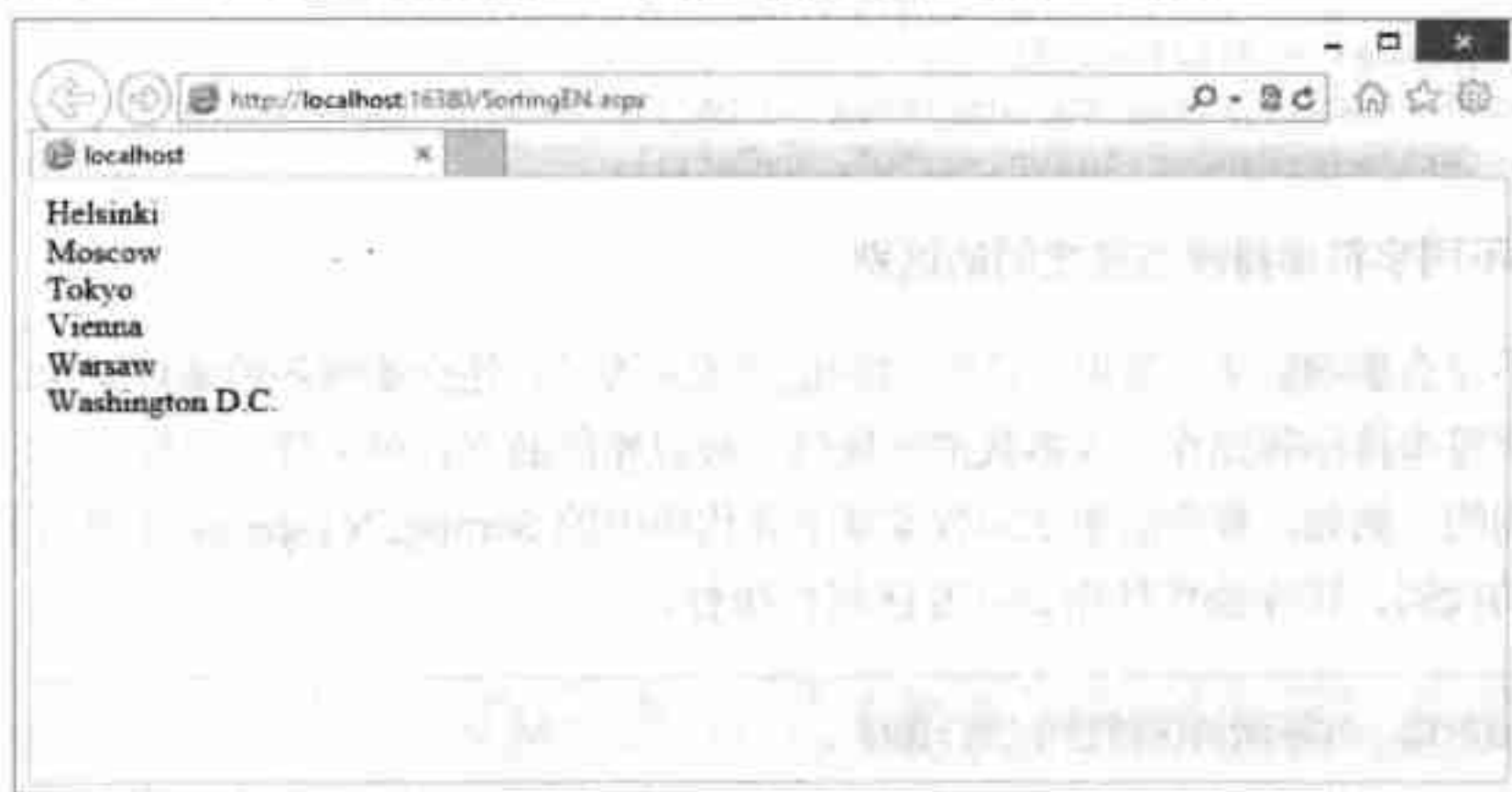


图 32-9

根据区域性,这正是我们希望的结果。但是在其他一些区域性中,数据的排序方式是有区别的。修改程序清单 32-12 中的代码,把区域性设置为 `Finnish`,如程序清单 32-13 所示(本章下载代码中的 `SortingFI.aspx.cs`)。

程序清单 32-13 把区域性改为 Finnish

```

using System;
using System.Collections.Generic;

```



```

using System.Globalization;

namespace Chapter32CS
{
    public partial class SortingFI : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            System.Threading.Thread.CurrentThread.CurrentCulture =
                new CultureInfo("fi-FI");

            List<string> myList = new List<string>();

            myList.Add("Washington D.C.");
            myList.Add("Helsinki");
            myList.Add("Moscow");
            myList.Add("Warsaw");
            myList.Add("Vienna");
            myList.Add("Tokyo");

            myList.Sort();

            foreach(string item in myList)
            {
                Response.Write(item.ToString() + "<br>");
            }
        }
    }
}

```

如果在 Finnish 区域性设置下运行上述代码，就会得到如图 32-10 所示的结果。

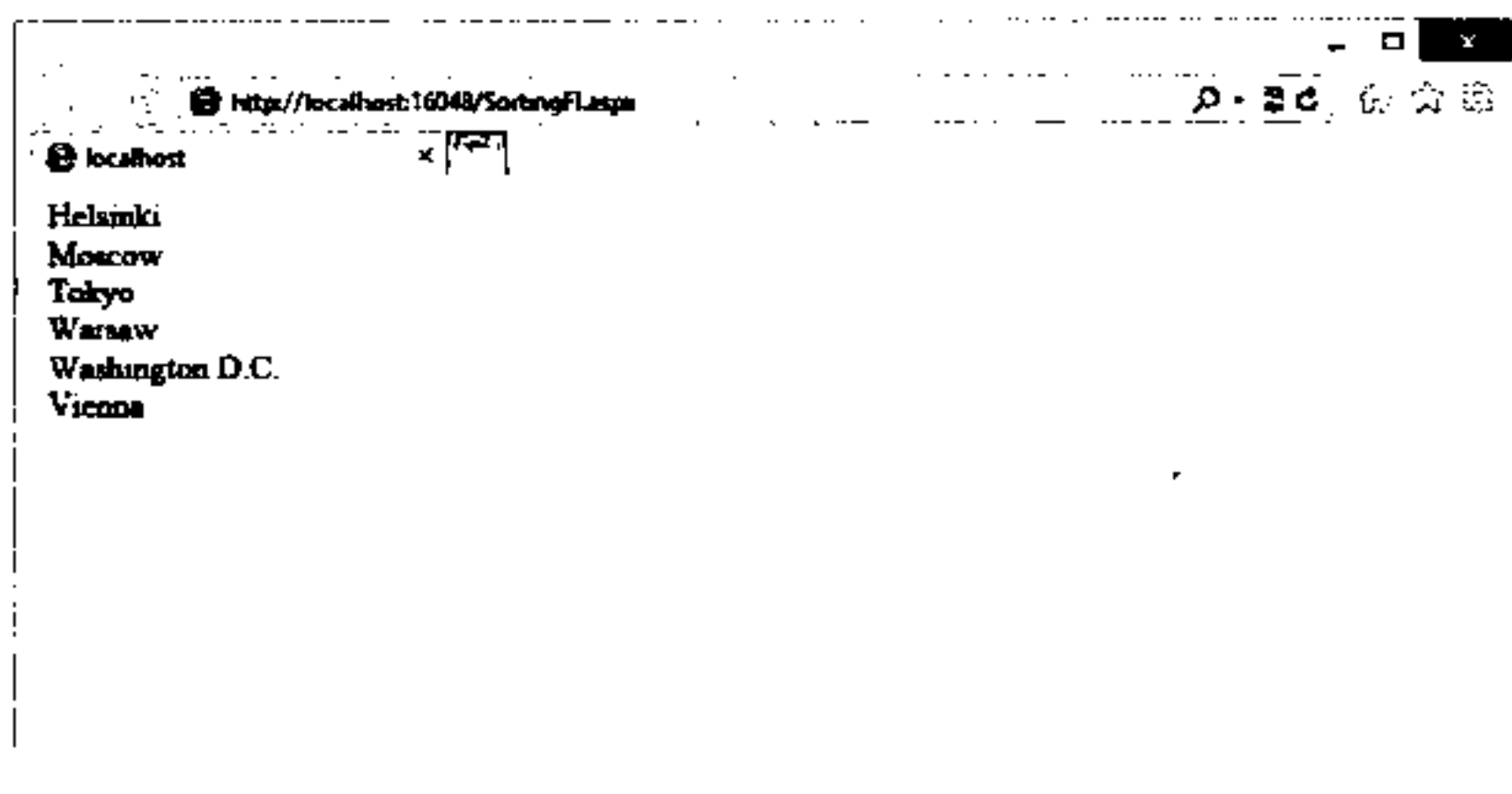


图 32-10

查看图 32-10 中使用芬兰语区域性设置进行的排序和图 32-9 中使用美式英语区域性设置进行的排序，显然城市 Vienna 的排列位置在芬兰语版本中是不同的。这是因为在芬兰语中，字母 V 和字母 W 是没有区别的。如果使用芬兰语区域性设置进行排序，Vi 就会排在 Wa 的后面，因此在排序的字符串列表中，Vienna 位于最后一行。

32.2 ASP.NET 4.5 资源文件

在使用 ASP.NET 时,创建本地化 Web 页面的一种最高效方式是根据用户的语言和区域性,给页面上的文本使用资源,可以创建在资源对象(例如资源文件)中存储的数据。接着 ASP.NET 可以在运行期间,根据用户的语言和区域性,选择正确的属性。这些属性不一定是文本。资源文件也可以存储其他类型的元素,例如图像。资源文件是扩展名为.resx 的 XML 文件。Visual Studio 2012 提供了使用资源文件的工具。资源文件可以存储为本地和全局资源。全局资源文件放在应用程序根目录下的 App_GlobalResources 文件夹中。App_GlobalResources 文件夹中的资源文件具有全局作用域,其属性可以由多个 ASP.NET 页面使用。本地资源文件放在名为 App_LocalResources 的一个或多个文件夹下。与根 App_GlobalResources 文件夹不同,App_LocalResources 文件夹可以位于应用程序的任意文件夹下。本地资源文件仅与一个 ASP.NET 页面相关,并且使用特定的命名约定。

辅助程序集

ASP.NET 中的资源文件使用.resx 扩展名。在运行期间,文件编译到程序集中,有时称为辅助程序集。因为.resx 文件是动态编译的,与 ASP.NET Web 页面相同,所以不需要创建这些辅助程序集。

但可以手工创建辅助程序集。为了创建辅助程序集,可以使用 Visual Studio 和 Windows SDK 安装的两个工具。Resource File Generator (Resgen.exe)用于把包含资源的文本文件和.resx 文件编译为二进制.resources 文件。接着使用 Assembly Linker (Al.exe)把.resources 文件编译为辅助程序集。Al.exe 会从指定的.resources 文件中创建辅助程序集。辅助程序集只能包含资源,不能包含任何可执行的代码。

32.2.1 使用本地资源

建立 ASP.NET 页面是很简单的,因此可以将它本地化为其他语言。为了利用资源文件中存储的资源,只需像往常那样建立 ASP.NET 页面,然后使用 Visual Studio 2012 中的工具,创建资源文件,修改页面以使用它。

为了说明这一点,建立一个简单的 ASP.NET 页面,如程序清单 32-14 所示(本章下载代码中的 LocalResources.aspx)。

程序清单 32-14 建立用于本地化的基本 ASP.NET 页面

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, System.EventArgs e)
    {
        Label2.Text = TextBox1.Text;
    }
</script>

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Local Resources</title>
```

```

</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:Label ID="Label1" runat="server"
        Text="What is your name?"></asp:Label>
      <br />
      <br />
      <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
      <asp:Button ID="Button1" runat="server" Text="Submit Name"
        OnClick="Button1_Click" />
      <br />
      <br />
      <asp:Label ID="Label2" runat="server"></asp:Label>
    </div>
  </form>
</body>
</html>

```

这个 ASP.NET 页面的内容较少，只包含两个 Label 控件、一个 TextBox 控件和一个 Button 控件。该页面的关键是生成文本的元素。第一个标签、按钮甚至页面的标题都包含要显示给用户的文本。如果文本根据页面的区域性设置来修改，就可以使用资源文件。

Visual Studio 2012 中的工具可以用于完成把 Web 页面转换为使用资源文件，进行本地化的所有初始工作。确保页面当前在设计视图中显示，再选择 Visual Studio 菜单中的 Tools | Generate Local Resource 命令。注意只有页面处于设计视图时，才能选择这个工具。该工具不能用于页面的拆分视图或代码视图。

选择 Generate Local Resource 命令后，Visual Studio 就会对项目进行几处修改。在项目中创建 App_LocalResources 文件夹(如果该文件夹还不存在)，然后在这个新文件夹中放置一个本地资源文件。这个本地资源文件的名称基于 ASP.NET 页面的名称，格式为 pageOrControlName.extension.language.resx 或 pageOrControlName.extension.language-culture.resx。如果资源用于不变的区域性，就去掉文件的 language 和 culture 部分。例如，如果使用 LocalResources.aspx 页面，资源文件名就是 LocalResources.aspx.resx。注意这个资源文件用于不变的区域性，图 32-11 显示了对项目所做的这些修改。

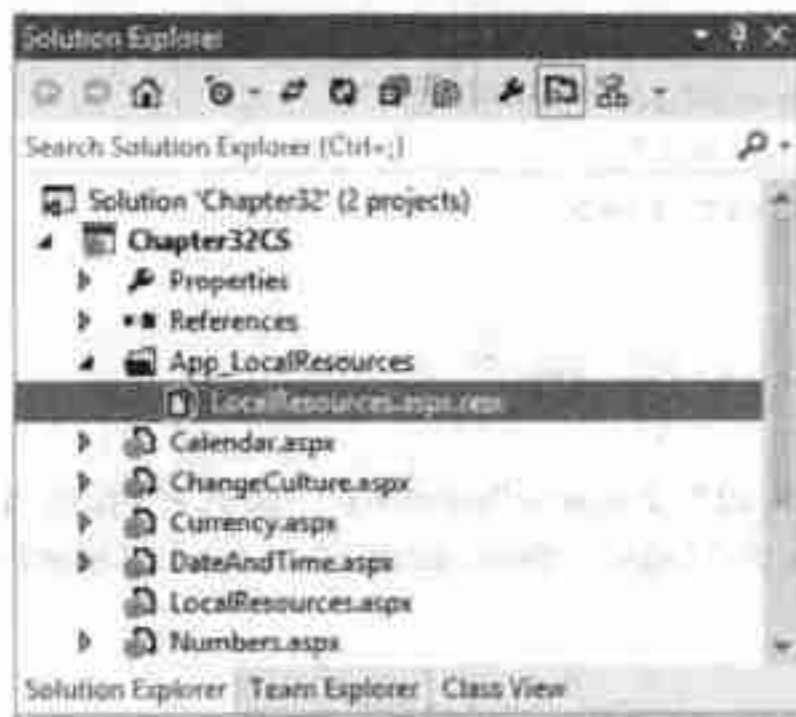


图 32-11

如果打开资源文件，就会显示 Resource Editor。每个属性都有 Name、Value 和 Comment 字段。

图 32-12 在 Resource Editor 中显示了 Visual Studio 自动创建的资源文件。

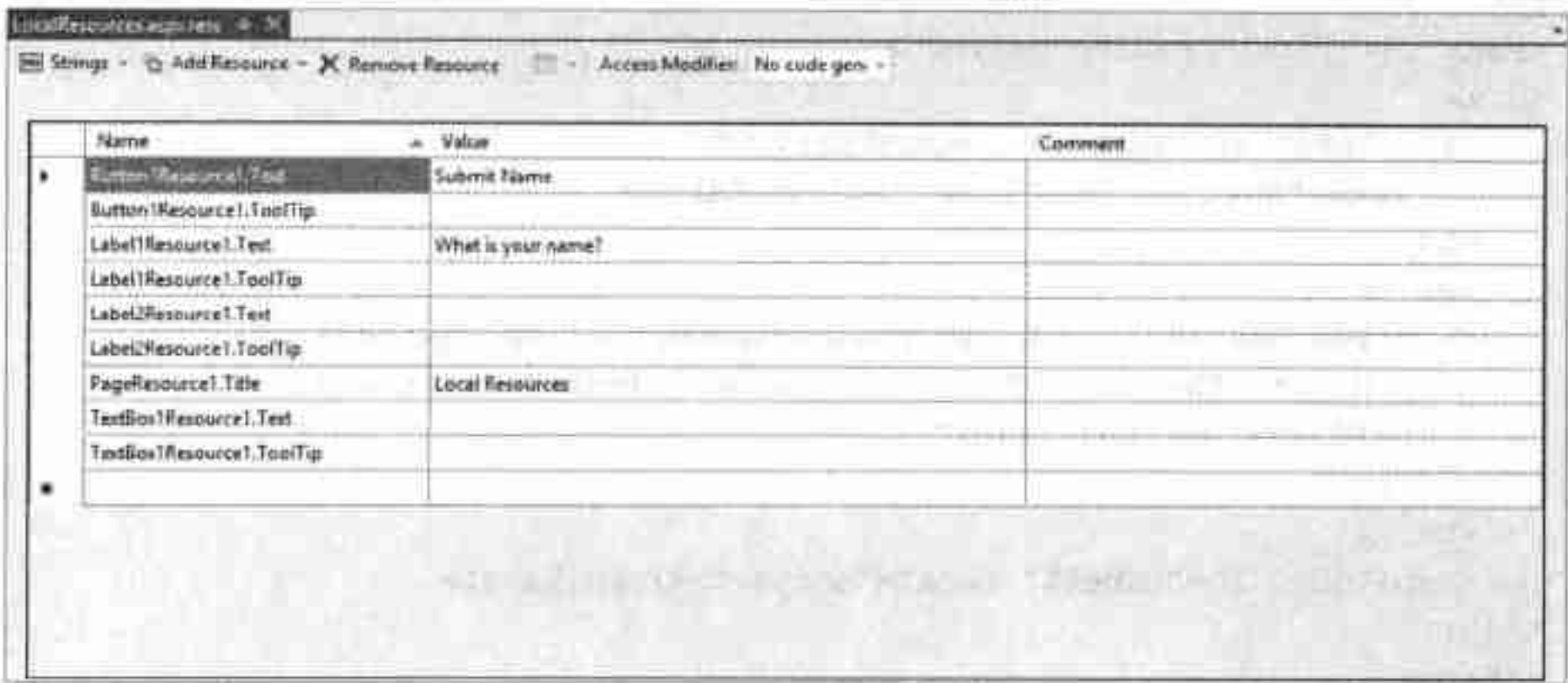


图 32-12

Visual Studio 在页面上搜索有可翻译属性的每个控件，并在资源文件中为每个这样的属性创建一项。甚至为没有在页面上填充的几个属性包含对应的项。注意页面控件中的每个文本值，包括 `title` 元素，都被添加到资源文件中，如图 32-12 所示。这些项都被指定了可以在页面中引用的名称。

除了在本地图源文件夹中创建和添加资源文件之外，Visual Studio 还对页面进行了一些修改。程序清单 32-15 演示了 Visual Studio 修改后的页面(本章下载代码中的 `LocalResourcesAfter.aspx`)。

程序清单 32-15 Visual Studio 改变页面代码的方式

```
<%@ Page Language="C#" culture="auto" meta:resourcekey="PageResource1"
    uiculture="auto"%>

<script runat="server">
    protected void Button1_Click(object sender, System.EventArgs e)
    {
        Label2.Text = TextBox1.Text;
    }
</script>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Local Resources</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server" Text="What is your name?"
            meta:resourcekey="Label1Resource1"></asp:Label>
        <br />
        <br />
        <asp:TextBox ID="TextBox1" runat="server"
            meta:resourcekey="TextBox1Resource1"></asp:TextBox>
        <asp:Button ID="Button1" runat="server" Text="Submit Name"
            OnClick="Button1_Click" meta:resourcekey="Button1Resource1" />
    </div>
    </form>
</body>
</html>
```

```

        <br />
        <br />
        <asp:Label ID="Label2" runat="server"
            meta:resourcekey="Label2Resource1"></asp:Label>
    </div>
</form>
</body>
</html>

```

在这段修改过的代码中，将 `culture` 和 `uiculture` 特性及值 `auto` 添加到 `@Page` 指令中，从而使应用程序可以本地化。另外，还给 `@Page` 指令和上述带有可翻译特性的每个控件添加了特性 `meta:resourcekey` 及其关联的值，赋予每个 `meta:resourcekey` 特性的值是资源文件中自动创建的名称。在图 32-12 中，`Button1` 控件有两个在资源文件中创建的属性——`Text` 和 `ToolTip`，`Text` 属性的值从页面中获得。在资源文件中，这两项的名称以 `Button1Resource1` 开头，以最初属性的名称结尾。查看 Visual Studio 修改后的 `Button` 控件，该按钮控件的 `meta:resourcekey` 值设置为带有 `Button1Resource1` 前缀：

```

<asp:Button ID="Button1"
    runat="server" Text="Submit Name"
    meta:resourcekey="Button1Resource1" />

```

资源文件中使用带这个前缀的所有属性，例如 `Text` 和 `ToolTip` 属性，都在运行期间自动应用于这个按钮控件。

1. 添加另一种语言的资源文件

`LocalResources.aspx.resx` 文件用于不变的区域性。如果没有确定页面的区域性，就使用这个资源文件。可以给所支持的任意多个区域性添加对应的资源文件。要为 `LocalResources.aspx` 页面添加另一个资源文件，以处理另一种语言，就应把 `LocalResources.aspx` 文件复制并粘贴到 `App_LocalResources` 文件夹中，重命名新复制的文件，以指定面向的语言和区域性。如果要创建面向 Finnish 区域性的资源文件，就把文件命名为 `LocalResources.aspx.fi-FI.resx`，修改资源文件中的下述值，以支持芬兰语：

```

Button1Resource1.Text    Lähetä Nimi
Label1Resource1.Text     Mikä sinun nimi on?
PageResource1.Title      Näytesivu

```

在不变的区域性和 Finnish 区域性的资源文件中添加一个新项。在演示如何编程访问资源项时，要使用这一项。将该项命名为 `Label2Answer`。该项的值根据区域性来确定。在不变的区域性文件中，该项的值是：

```
Hello
```

在 Finnish 文件中，该项的值是：

```
Hei
```

2. 编程访问资源属性

不仅可以使用前边的属性命名约定，自动将某些资源应用到页面，还可以编程引用资源。程序清单 32-16 显示了添加引用 Label2Answer 资源项的代码后的服务器脚本块(本章下载代码中的 LocalResourcesCustom.aspx)。

程序清单 32-16 编程引用资源

```
<script runat="server">
    protected void Button1_Click(object sender, System.EventArgs e)
    {
        Label2.Text = GetLocalResourceObject("Label2Answer").ToString() + " " +
            TextBox1.Text;
    }
</script>
```

可以使用 GetLocalResourceObject 方法访问资源文件中的属性。GetLocalResourceObject 是 HttpContext 类的方法成员。在使用 GetLocalResourceObject 方法时，只需将资源项的名称作为参数传递即可，如下所示：

```
GetLocalResourceObject("Label2Answer")
```

使用相同的方法，也可以从资源文件中编程获得控件的任意属性值：

```
GetLocalResourceObject("Button1Resource1.Text")
```

有了程序清单 32-16 中的新脚本块，并完成资源文件之后，就可以运行页面，在文本框中输入姓名，然后单击按钮，获得如图 32-13 所示的响应。

执行 LocalResourcesCustom.aspx 页面时，浏览器会设置为 en-US 区域性。因为没有 en-US 区域性的资源文件，所以 ASP.NET 寻找 EN 中性区域性文件，但也没有 EN 中性区域性文件，因此 ASP.NET 只能使用不变区域性的资源文件。不变区域性的资源文件给 Label2Answer 项使用 Hello 值。

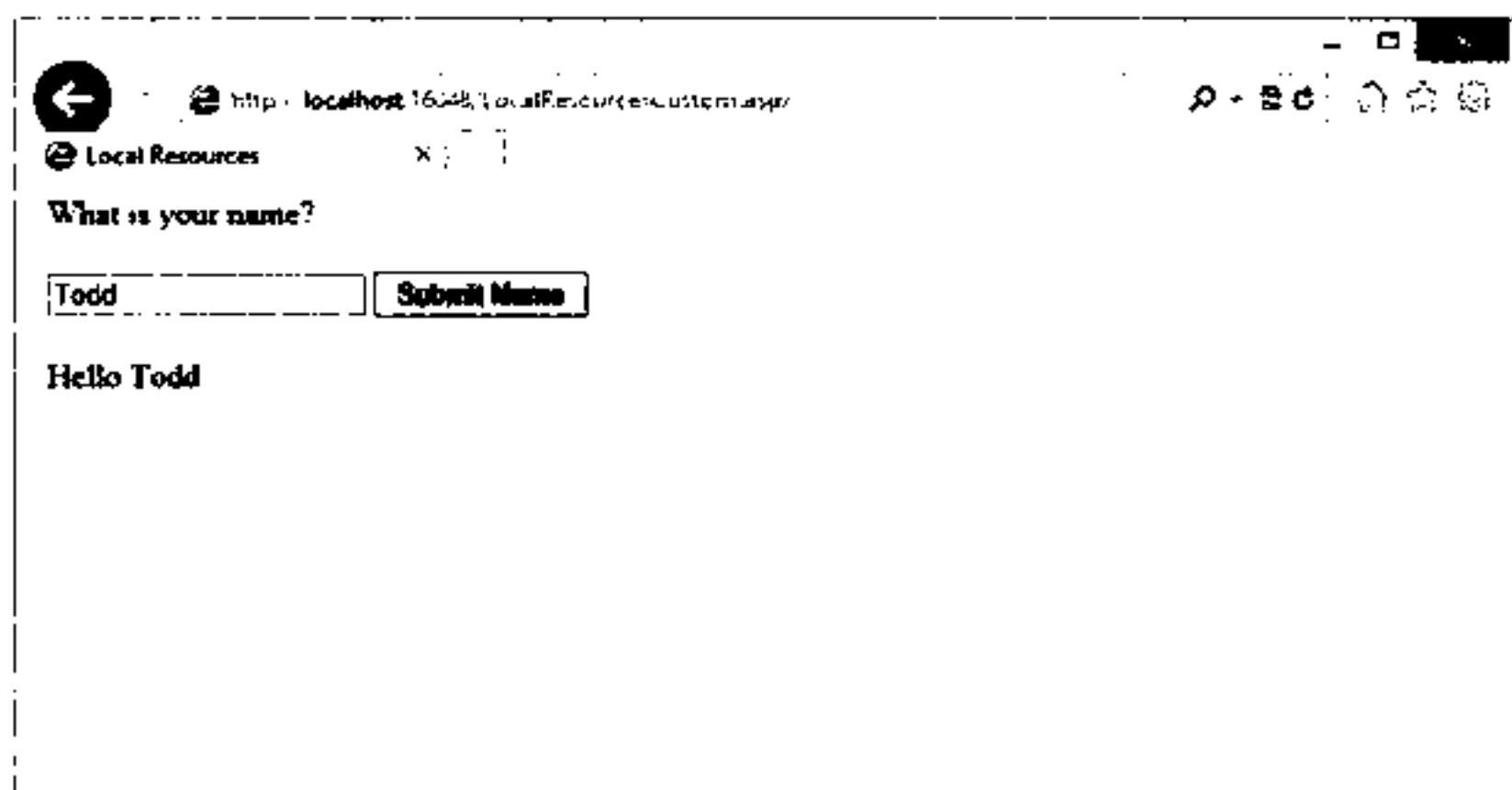


图 32-13

如果把 IE 或 Page 指令的语言首选项设置为 fi-FI，重新运行 Default.aspx 页面，就会得到该页面的芬兰语版本，如图 32-14 所示。

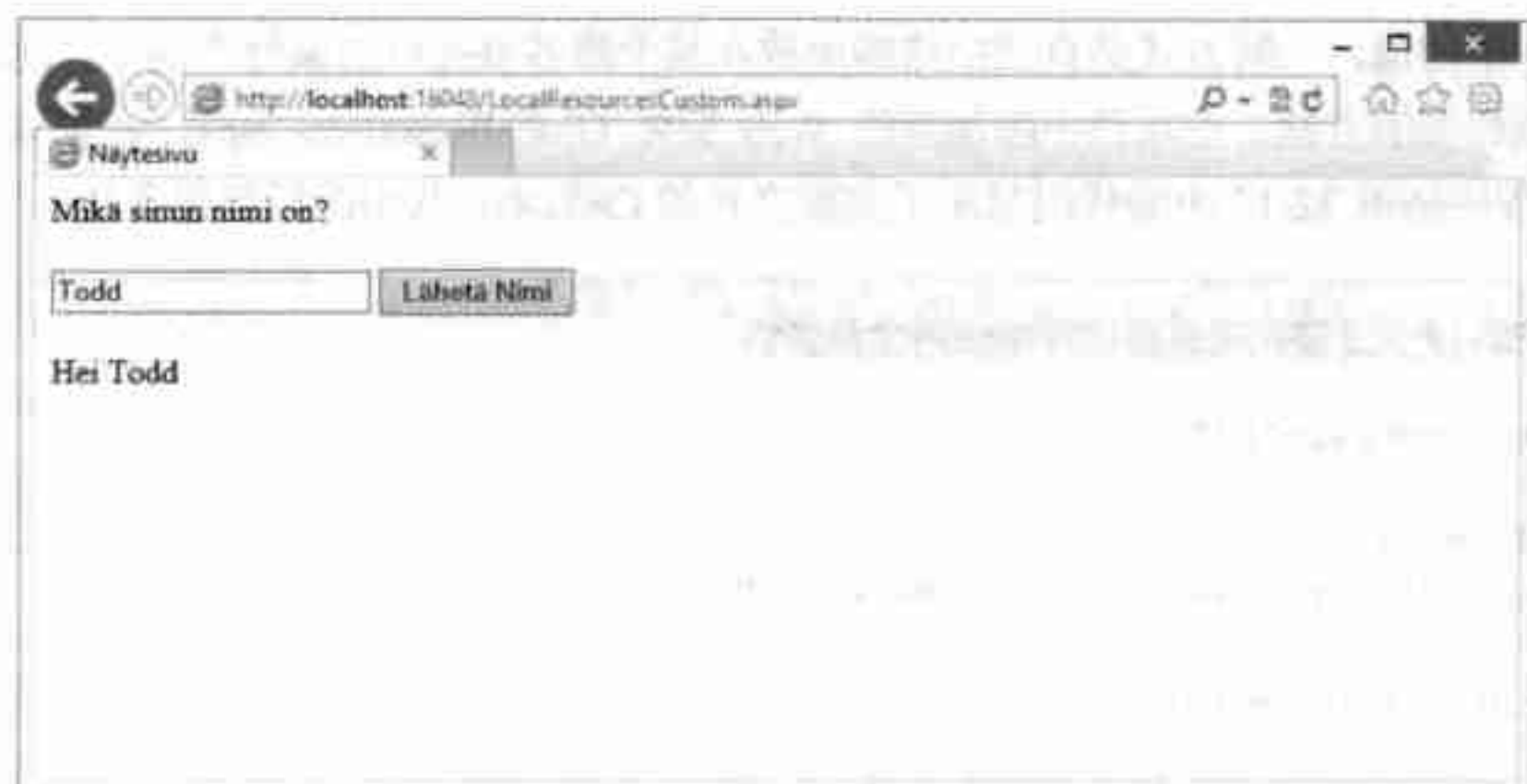


图 32-14

可以看出, ASP.NET 会自动使用已转换并放在芬兰语资源文件中的所有控件属性, 包括 IE 标题栏中显示的页面标题。而且, 会在芬兰语资源文件中获得 Label2Answer 的已编程访问项。

3. 中性区域性一般是首选项

前面的例子仅使用不变区域性的资源文件或指定了语言和区域性的资源文件。示例代码中的 LocalResourcesCustom.aspx.fi-FL.resx 文件同时指定了语言和区域性: 芬兰的芬兰语。另一个选项会使这个文件不用于特定的区域性, 而是用于中性区域性。为此, 只需把该文件命名为 LocalResourcesCustom.aspx.fi.resx。在这个例子中, 特定区域性的声明其实没有什么意义, 因为除了芬兰之外, 其他国家都不使用芬兰语。但是, 这个声明对于德语、西班牙语或法语来说是有意义的, 因为这些语言由多个国家使用。例如, 如果建立 LocalResources.aspx 页面的西班牙语版本, 就可以明确地为特定的区域性建立该资源文件, 如 LocalResources.aspx.es-MX.resx。这个资源文件用于墨西哥的西班牙语版本。如果请求者的语言设置是 es-ES, ASP.NET 就会提供不变区域性的资源文件。此时, 最后创建中性区域性资源文件, 这样使用西班牙语的用户, 无论属于什么国家, 都可以使用西班牙语的資源文件。如果建立了资源文件 LocalResources.aspx.ES.resx, 那么无论终端用户的语言设置为 es-MX、es-ES 还 es-AR, 都会得到页面的相应 ES 中性区域性版本。

32.2.2 使用全局资源

除了只使用本地资源处理 ASP.NET 应用程序中的页面之外, 还可以创建全局资源以用于多个页面。要创建可在整个应用程序中使用的资源文件, 必须先为全局资源创建文件夹。在 Visual Studio 的 Solution Explorer 中右击解决方案, 选择 Add | Add ASP.NET Folder | App_GlobalResources。现在就可以在这个新文件夹中直接添加全局资源文件。右击 App_GlobalResources 文件夹, 选择 Add | Resources File, 把文件命名为 Resource.resx。与本地资源文件类似, 这个文件是不变区域性的期间资源文件。在 Resource Editor 中打开 Resource.resx 文件, 添加字符串资源, 命名为 PrivacyStatement。创建一个任意长度的字符串作为其值。因为这是文件的不变区域性版本, 所以创建一个芬兰语区域性的版本, 把该文件命名为 Resource.fi-FL.aspx。添加相同的 PrivacyStatement 资源项, 但给它指定另一个任意值。

全局资源文件的作用是, 在整个应用程序的范围内访问这些资源。我们可以通过几种方式访问

放在这些文件中的值。一种方式是直接把这些值放在某个服务器控件的属性声明中。这类似于本章前面访问本地资源的方法。在项目中添加新的 Web 窗体 GlobalResources.aspx，在窗体上添加一个 Label 控件，程序清单 32-17 中的代码显示了完整文件的 C# 版本，其中的标签包含访问资源的声明。

程序清单 32-17 直接在服务器控件中使用全局资源

```
<%@ Page Language="C#" %>

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Global Resources</title>
</head>
<body>
  <asp:Label ID="Label1" runat="server" Text='<%$ Resources: Resource,
    PrivacyStatement %>'></asp:Label>
</body>
</html>
```

要提取 PrivacyStatement 全局资源的值，应使用关键字 Resources 后跟一个冒号。接着，指定资源文件的名称。在本例中，资源文件名是 Resource，表示 Resource.resx 和 Resource.fi-FI.resx 文件。在指定要使用的资源文件后，再指定需要的资源，在本例中是 PrivacyStatement。

获得该结果的另一种方式是使用 Visual Studio 2012 的一个内置对话框。在 GlobalResources.aspx 窗体上再添加一个 Label 控件，在设计视图中突出显示这个新的 Label 服务器控件，使之显示在 Properties 窗口中，在该窗口中单击 Data 部分的 Expressions 属性中的按钮，这会打开 Expressions 对话框，在其中将 PrivacyStatement 值绑定到控件的 Text 属性。

突出显示 Bindable 属性列表中的 Text 属性，然后从对话框右边的 Expression 类型下拉列表框中选择 Resources 表达式选项，之后输入 ClassKey 和 ResourceKey 属性的值。ClassKey 属性是要使用的文件名，本例中的文件名是 Resource.resx，因此将 Resource 作为其值。ResourceKey 属性是资源文件中资源项的名称，输入 PrivacyStatement 作为 ResourceKey 属性的值。图 32-15 显示了选择正确项之后的 Expressions 对话框。

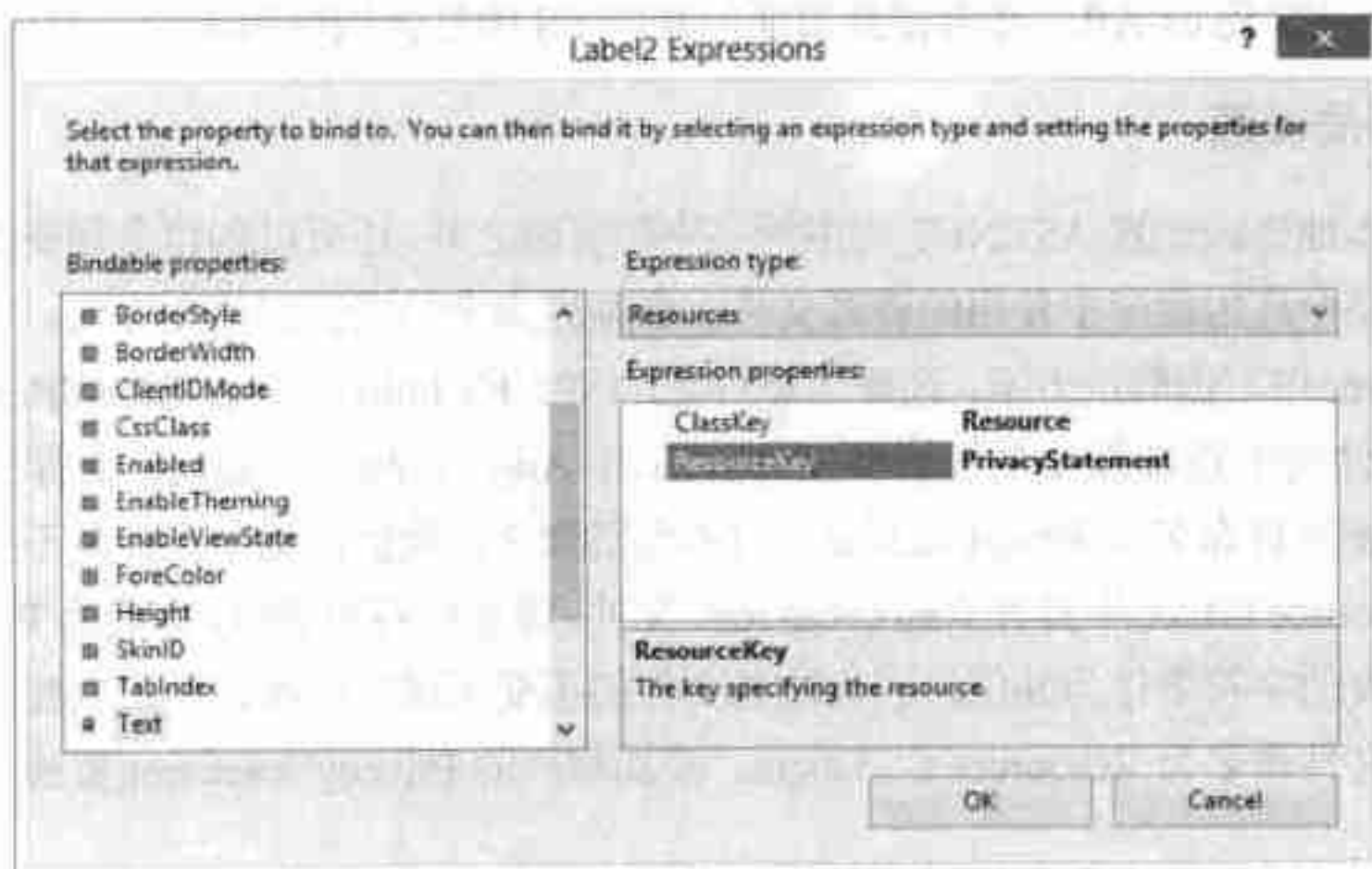


图 32-15

Visual Studio 的优点之一是，可以按照强类型化的方式使用通过全局资源文件提供的资源。例如，可以使用程序清单 32-18 所示的代码，编程获取全局资源值(本章下载代码中的 GlobalResources.aspx.cs)。

程序清单 32-18 编程获取全局资源

```
using System;

namespace Chapter32CS
{
    public partial class GlobalResources : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Label3.Text = Resources.Resource.PrivacyStatement;
        }
    }
}
```

在图 32-16 中，IntelliSense 显示了这些资源值。

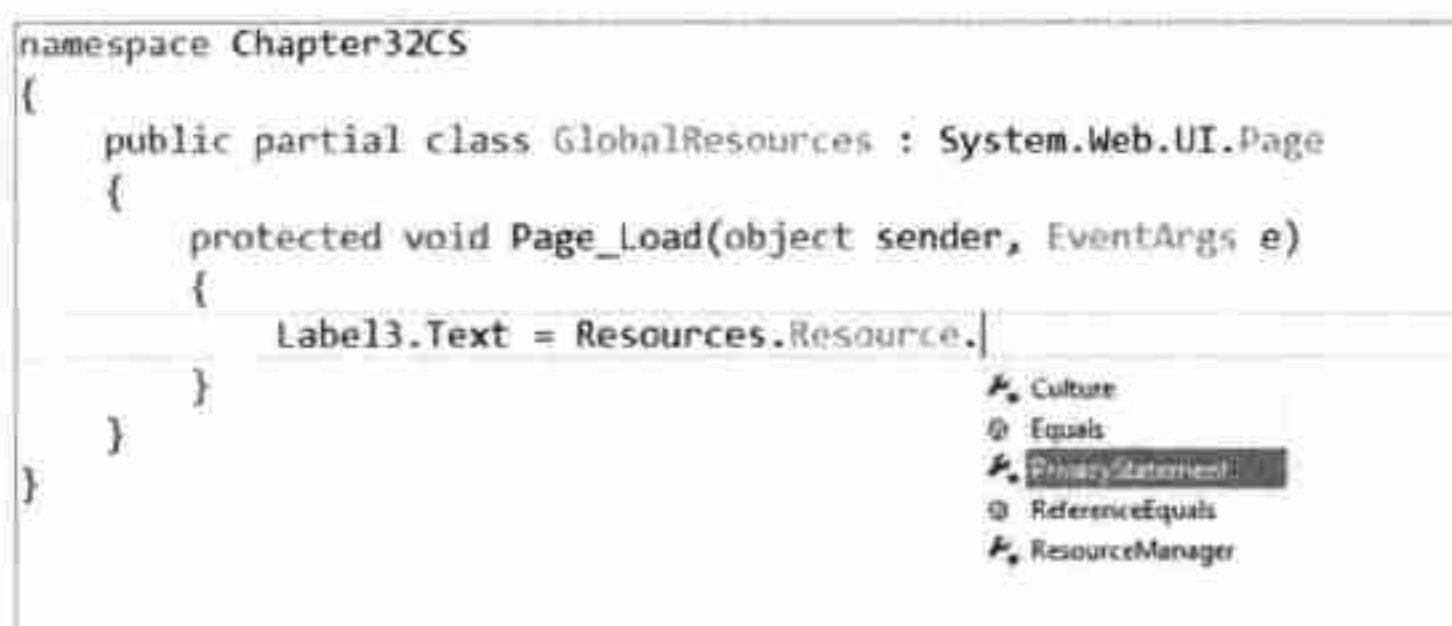


图 32-16

资源编辑器

Visual Studio 为使用资源文件提供了编辑器。本章前面已经给出了该编辑器的用法。在该编辑器中，资源是按照数据类型进行可视化分类的。本章仅仅处理了字符串，但还存在其他类别，如图像、图标、音频文件、杂项文件和其他项。这些选项如图 32-17 所示。

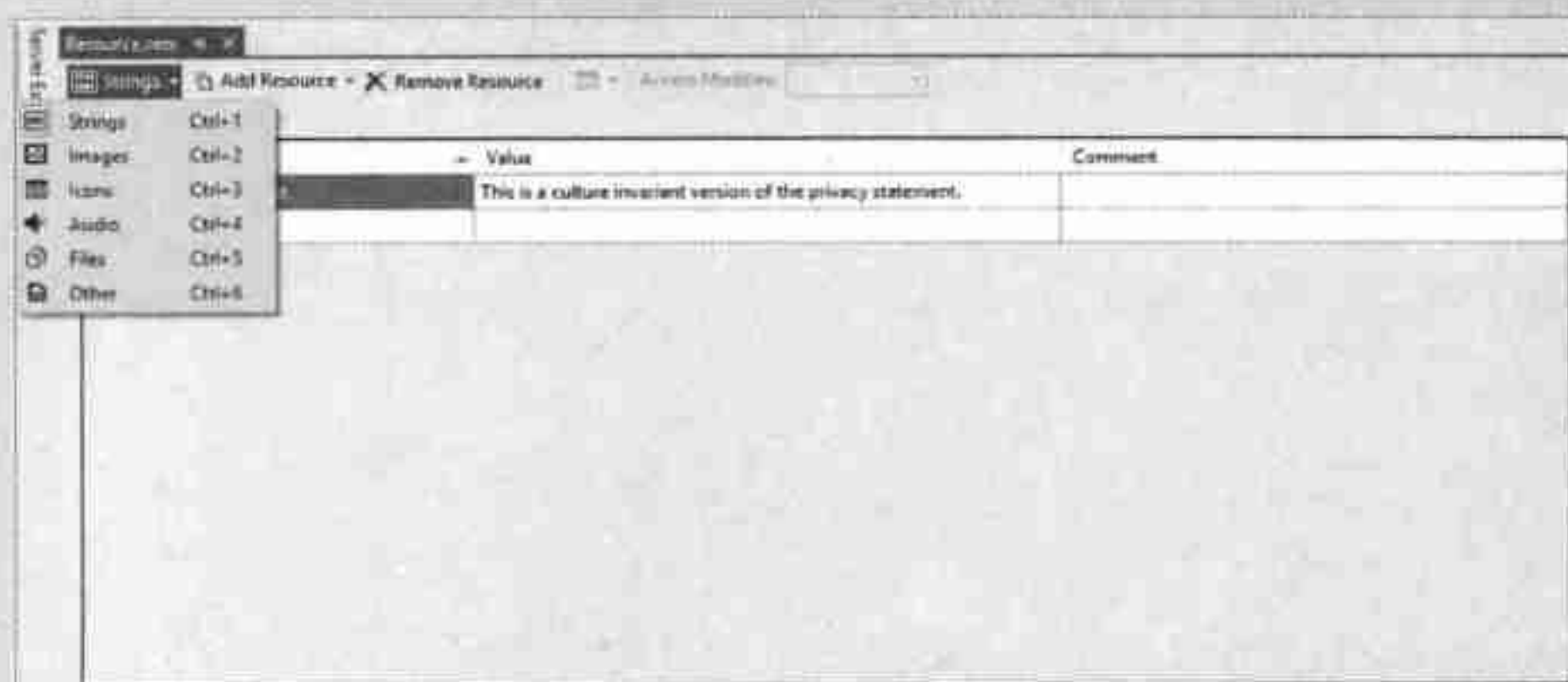


图 32-17

32.3 本章小结

对 ASP.NET 应用程序进行国际化后,就可以处理多种区域性。访问 Web 应用程序的用户目前来自多个国家,使用多种语言,所以添加多区域性支持是很重要的。本章介绍了国际化应用程序时存在的一些问题,Visual Studio 和 .NET Framework 提供的一些内置工具会使这个过程很容易完成。

国际化应用程序非常简单,只需利用已创建好的 ASP.NET 页面并使用相应的 Visual Studio 工具,修改页面以容纳多种语言。

第 33 章

打包和部署 ASP.NET 应用程序

本章要点

- 了解打包和部署的基础知识
- 选择部署方法
- 使用发布配置文件
- 部署到 Windows Azure

人们常常不重视 ASP.NET 应用程序的打包和部署。本章将深入探讨在建立 ASP.NET 应用程序后如何打包和部署它们。在开发计算机上建立 ASP.NET 应用程序后，就要把完成的产品部署到质量保证或开发用的服务器上，并且最终部署到产品服务器上。

考虑 ASP.NET 应用程序的正确打包和部署，一个重要原因是许多应用程序都设计为可销售的产品、初学者的工具集或解决方案。任何人都可以下载它们，在自己的环境下安装这些产品，而这些人环境我们根本无法控制。此时，最好给使用者提供安装文件，确保在任意环境下都能正确安装应用程序。

但是，无论是否在公司外部发布 Web 应用程序，都需要一种方式，在进行产品部署之前把它们部署到另一台经过测试的服务器上。而不应仅因为应用程序在自己的计算机上能正常工作，就认为是完美的。在大多数情况下，我们仅使用 Visual Studio 中的 IIS Express 进行开发，因此需要使用 IIS 进行全面测试，才能确保一切运行正常。即使使用 IIS 在计算机上进行了测试，也仍然有一些需要消除的与部署相关的因素，因此在应用程序进入产品阶段之前，必须进行全面测试。



Visual Studio 2012 不再支持 Web 部署项目。使用发布配置文件是把应用程序部署到服务器上的推荐方式。如果使用 Windows 安装程序部署应用程序，应使用 WiX(Windows Installer XML)或 InstallShield。

33.1 部署各个部分

我们到底要部署什么？ASP.NET 包含了整个应用程序的许多组成部分，它们都需要和应用程序一起部署，从而应用程序才能正常运行。下面列出了 ASP.NET 应用程序的一些重要组成部分，在迁移应用程序时需要部署它们：

- .aspx 页面
- .aspx 页面的隐藏代码页面(.aspx.cs 文件)
- 用户控件(.ascx)
- Web 服务文件(.asmx 和.wSDL 文件)
- .htm 或.html 文件
- 图像文件，如.jpg 或.gif
- ASP.NET 系统文件夹，如 App_Code 和 App_Themes
- JavaScript 文件(.js)
- 层叠样式表(.css)
- 配置文件，如 web.config 文件
- .NET 组件和编译好的程序集
- 数据文件，如.mdb 文件
- IIS 设置
- HTTP 处理程序和模块

33.2 部署之前的准备步骤

在部署 ASP.NET Web 应用程序之前，应执行一些基本操作，以确保应用程序准备好部署。这些操作很容易忘记，这里提及它们是为了说明如何确保部署的应用程序正常运行。

首先，在 web.config 文件中关闭调试功能。为此，要把<compilation>元素中的 debug 特性设置为 false，如程序清单 33-1 所示。

程序清单 33-1 在部署应用程序之前把 debug 特性设置为 false

```
<configuration>
  <system.web>
    <compilation debug="false" targetFramework="4.5" />
  </system.web>
</configuration>
```

在默认情况下，大多数开发人员在开发应用程序时都会把 debug 特性设置为 true，这样就会把调试符号插入已编译好的 ASP.NET 页面中。这些符号会降低应用程序的性能。在建立应用程序并准备好部署后，就不需要这些调试符号。

对于已经编写 ASP.NET 代码一段时间的开发人员而言，一定要注意 Visual Studio 菜单的下拉列表中的 Debug 选项应没有完成修改配置文件或类似的工作(如图 33-1 所示)。在 ASP.NET 1.0 和 1.1 中，Visual Studio .NET(当时的名称)实际地控制把 ASP.NET 项目编译到 DLL 中的工作。自从 ASP.NET

2.0 推出以后, ASP.NET 自己控制运行期间的编译过程。

因此, 虽然在下拉列表中仍然保留了 Debug 这个选项, 但它对 ASP.NET 项目的建立已没有任何作用。我们可以通过 web.config 文件中的设置完全控制编译过程, 如程序清单 33-1 所示。

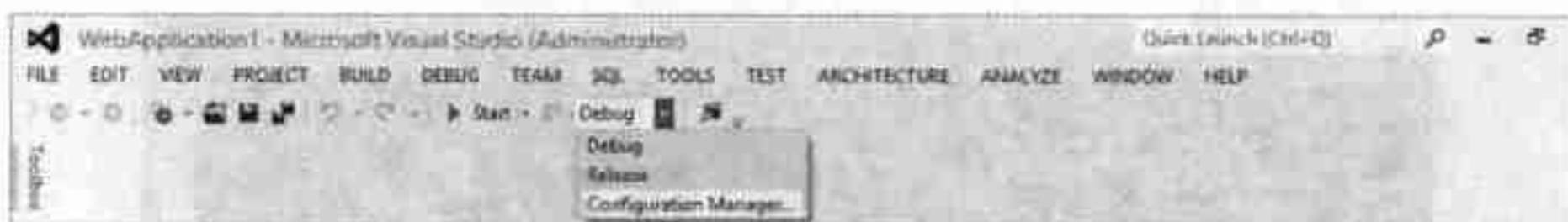


图 33-1

也可以在应用程序中提供 Web.Debug.config 和 Web.Release.config 文件。使用 Web.Debug.config 文件可以在配置文件中添加设置, 在调试模式下部署或运行应用程序时将会利用这些设置。在发布模式下发布应用程序时, 可以通过 Web.Release.config 文件使用相同的方法。利用这种通过配置文件执行的方法, 可以使用不同的数据库连接字符串、改变身份验证模式等。如果要处理多种环境(例如测试、开发和产品环境), 那么使用这些不同的配置文件就会非常有帮助。

33.3 部署 Web 应用程序的方法

注意, 部署是过程中的最后一步。首先要建立程序, 把程序打包到最适合部署的组件中。部署 Web 应用程序有许多方式。在演示时可以使用 XCOPY 功能(因为非常简单)。第二种方法是使用 Visual Studio 2012 的 Copy Web Site 功能把 Web 站点从一个位置复制到另一个位置。第三种方法是使用 Visual Studio 部署预编译的 Web 应用程序。最后一种方法是使用 Visual Studio 发布配置文件把应用程序发布到服务器上。在学习这些方法后, 我们就可以确定应使用什么方法。

33.3.1 使用 XCopy

应用程序在 .NET 中编译为程序集, 这些程序集包含由公共语言运行时(Common Language Runtime, CLR)执行的代码。程序集的优点是, 它们自我描述的。程序集的所有信息都存储在程序集中。 .NET 程序集自己存储这些信息, 因此 XCopy 部署是可行的操作, 不需要注册表设置。安装程序集与把它复制到另一台服务器上一样简单, 在这个过程中不需要停止或启动 IIS。

这里提及 XCopy, 是因为这基本上是对要移动的文件和文件夹进行复制/粘贴操作的命令行方式。但 XCopy 提供的功能绝不仅仅是复制/粘贴, 还可以把文件、目录甚至整个驱动器从一个位置移动到另一个位置。

XCopy 命令的默认语法如下:

```
xcopy [source] [destination] [/w] [/p] [/c] [/v] [/q] [/f] [/l] [/g]
[/d[:mm-dd-yyyy]] [/u] [/i] [/s [/e]] [/t] [/k] [/r] [/h] [/a] [/m] [/n] [/o]
[/x] [/exclude:file1[+file2][+file3]] [/y|/y] [/z]
```

下面是一个使用 XCopy 功能的例子。假定在开发人员的计算机(C:\)上工作, 要把 ASP.NET 应用程序复制到产品服务器(Y:\)上。完成该任务的命令的最简单形式是:

```
xcopy c:\Websites\Website1 Y:\Websites\ /f /e /k /h
```

这条命令把文件和文件夹从源驱动器移到目标驱动器。图 33-2 是在命令行上使用该命令的示例。

使用 XCopy 复制文件时，注意这个方法不能在 IIS 上自动创建虚拟目录。要复制新的 Web 应用程序，需要先在目标服务器上创建一个虚拟目录，再把这个虚拟目录与要复制的应用程序关联起来。这是一个简单的过程，但必须完成这一步，才能完成站点复制操作。



图 33-2

可以给这条 XCopy 命令提供许多参数，使其按照我们希望的方式运行。表 33-1 列出了这些参数。

表 33-1

参 数	说 明
/w	显示消息：Press any key to begin copying file(s)。等待我们的响应以开始复制过程
/p	要求对每个要复制的文件进行确认，采用逐个文件进行确认的工作方式
/c	忽略在复制过程中发生的错误
/v	对要复制的文件进行验证，确保它们与源文件相同
/q	不显示任何 XCopy 消息
/f	在复制过程中，显示源文件和目标文件的名称
/l	显示要复制到目标驱动器的文件列表
/g	为目标驱动器建立解密的文件
/d	只使用/d 时，就只复制比目标位置的现有文件更新的文件。另一选择是使用/d[:mm-dd-yyyy]，以复制在指定日期当天或之后修改的文件
/u	只复制目标位置已有的源文件
/i	如果要复制的是包含通配符的目录或文件，并且该项在目录位置不存在，就创建新目录。XCopy 过程还会把所有相关的文件都复制到这个目录中
/s	复制包含文件的所有目录及其子目录。在此过程中不复制空目录或空子目录
/e	无论目录是否包含文件，都复制所有的子目录
/t	复制子目录，但不复制它们包含的文件
/k	在默认情况下，XCopy 过程会删除源文件中包含的只读设置。使用/k 可确保在复制过程中保留这些只读设置
/r	仅把只读文件复制到目标位置
/h	指定包含隐藏文件和系统文件，在默认情况下不包含这些文件
/a	只复制设置了归档文件特性的文件，并在 XCopy 目的地保留归档文件特性
/m	只复制设置了归档文件特性的文件，并关闭归档文件特性
/n	使用 NTFS 短文件名和短目录名进行复制

(续表)

参 数	说 明
/o	除了复制文件外, 还复制随机访问控制列表(DACL)
/x	除了复制文件外, 还复制审查设置和系统访问控制列表(SACL)
/exclude	可以指定不复制某些文件, 语法结构是 exclude:File1.aspx+File2.aspx+File3.aspx
/y	在 XCopy 过程中不显示询问是否覆盖目标文件的提示
/-y	添加提示, 确认覆盖目标位置的已有文件
/z	通过网络以可重新启动的方式复制文件和目录
/?	显示 XCopy 命令的帮助

使用 XCopy 是把应用程序从一台服务器移到另一台服务器的一种简单方式。如果建立自己的虚拟目录, 这种部署模式就可以很好地工作。

复制 Web 应用程序后(放在正确的虚拟目录中), 就可以在浏览器中调用。

33.3.2 使用 Copy Web Site 选项

复制 Web 站点的另一选择是使用 Visual Studio 2012 提供的 GUI。这个 GUI 可以把 Web 站点从开发服务器复制到同一台服务器或远程服务器上(与 XCOPY 命令一样)。如果使用 Web 站点项目, 而不是 Web 应用程序项目, 就可以使用这个选项。

打开 Visual Studio 中的 Copy Web Site 对话框有两种方式。第一种方式是在 Visual Studio 的 Server Explorer 中单击 Copy Web Site 图标, 打开 Copy Web Site GUI 的另一种方式是从 Visual Studio 菜单中选择 Website | Copy Web Site 命令。这两种方式都会在 Document 窗口中打开 Copy Web Site GUI, 如图 33-3 所示。

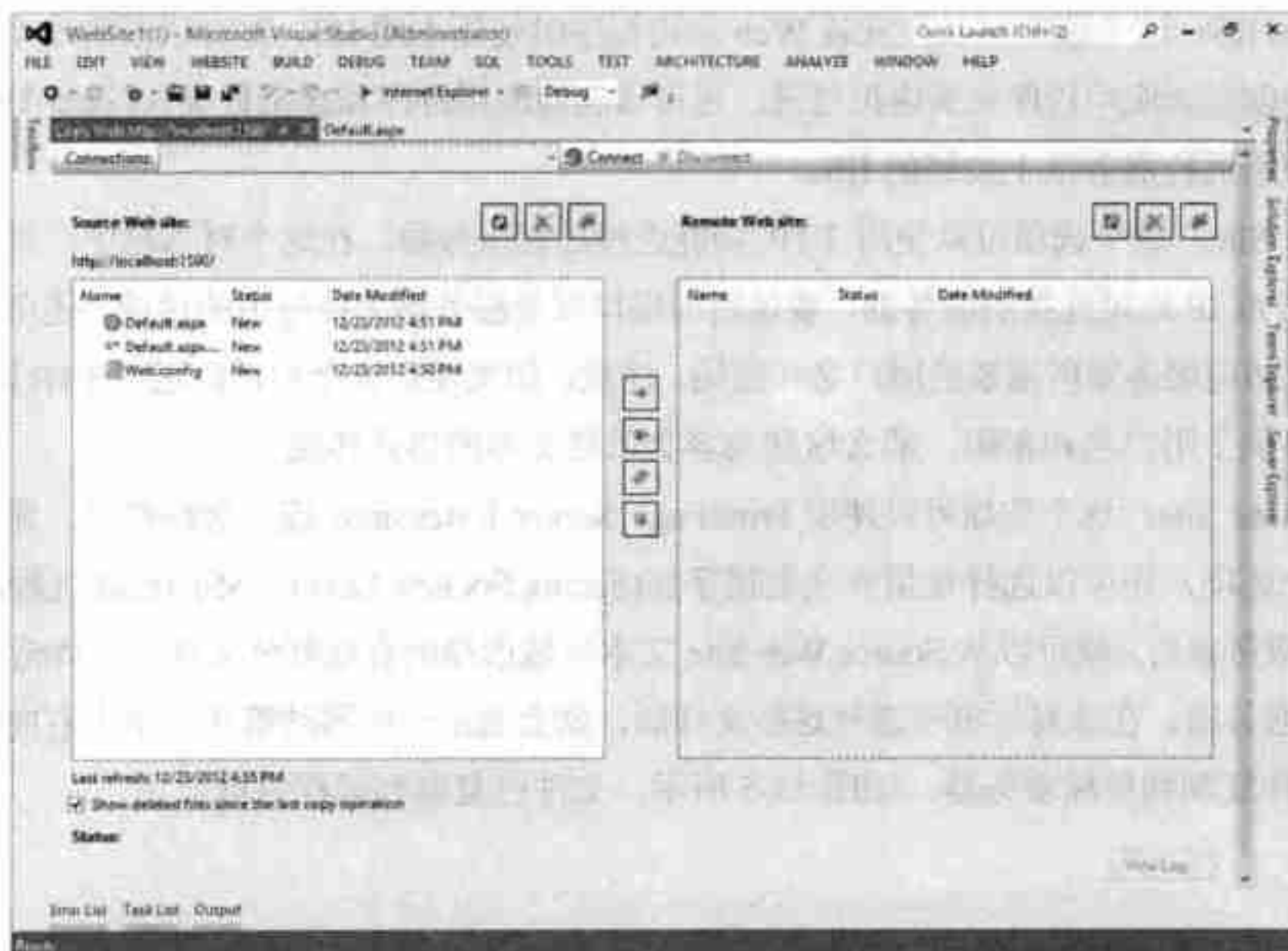


图 33-3

在这个 GUI 中,可以单击 Connect To a Remote Server 按钮(在 Connections 文本框的旁边),打开 Open Web Site 对话框,如图 33-4 所示。

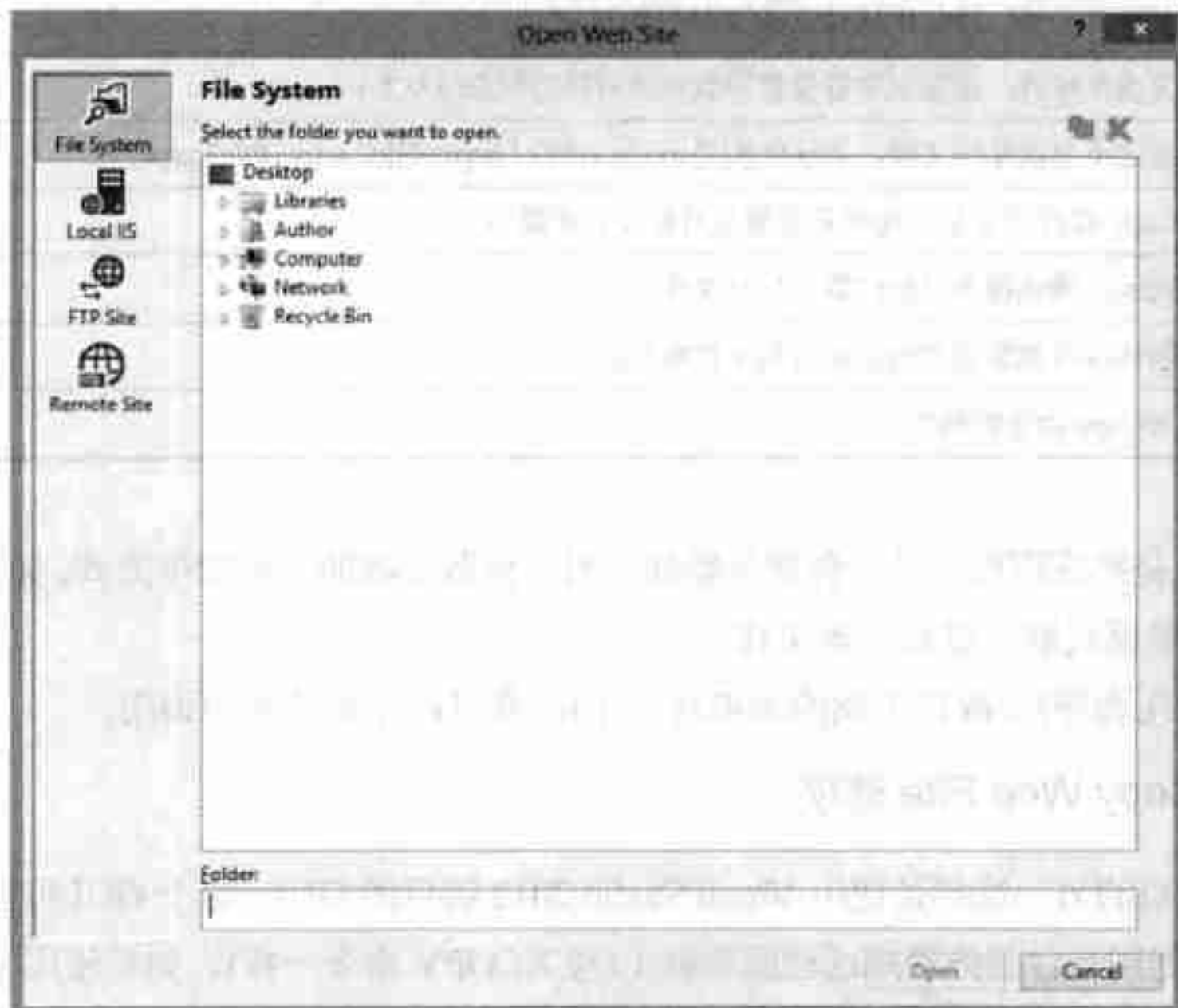


图 33-4

从这个对话框中可以看出,有几个选项可以连接并复制 Web 应用程序。这些选项包括:

- **File System:** 这个选项可以在计算机的文件浏览器视图中导航。如果要在远程服务器上安装,就必须把驱动器映射到安装位置。
- **Local IIS:** 这个选项可以在安装 Web 应用程序时使用本地 IIS。在对话框的这个部分,可以直接创建新应用程序和新虚拟目录。还可以删除应用程序和虚拟目录。Local IIS 选项不允许使用远程服务器上安装的 IIS。
- **FTP Site:** 这个选项可以使用 FTP 功能连接远程服务器。在这个对话框中,可以指定使用 URL 或 IP 地址连接的服务器、要使用的端口以及服务器上要使用的目录。还可以指定通过 FTP 访问服务器所需要的用户名和密码。注意,如果使用这个对话框通过 FTP 访问服务器,并提供了用户名和密码,那么这些内容会以纯文本的格式传送。
- **Remote Site:** 这个选项可以使用 FrontPage Server Extensions 连接远程站点。通过对话框的这个选项,还可以选择使用安全套接字层(Secure Sockets Layer, SSL)连接远程服务器。

连接到服务器后,就可以从 Source Web Site 文本区域选择所有或部分文件,把 Web 应用程序的内容复制到服务器。在该对话框中选择这些文件后,就会激活一些移动箭头。单击右向箭头,就会把选中的文件复制到目标服务器。如图 33-5 所示,文件已复制到远程目标。

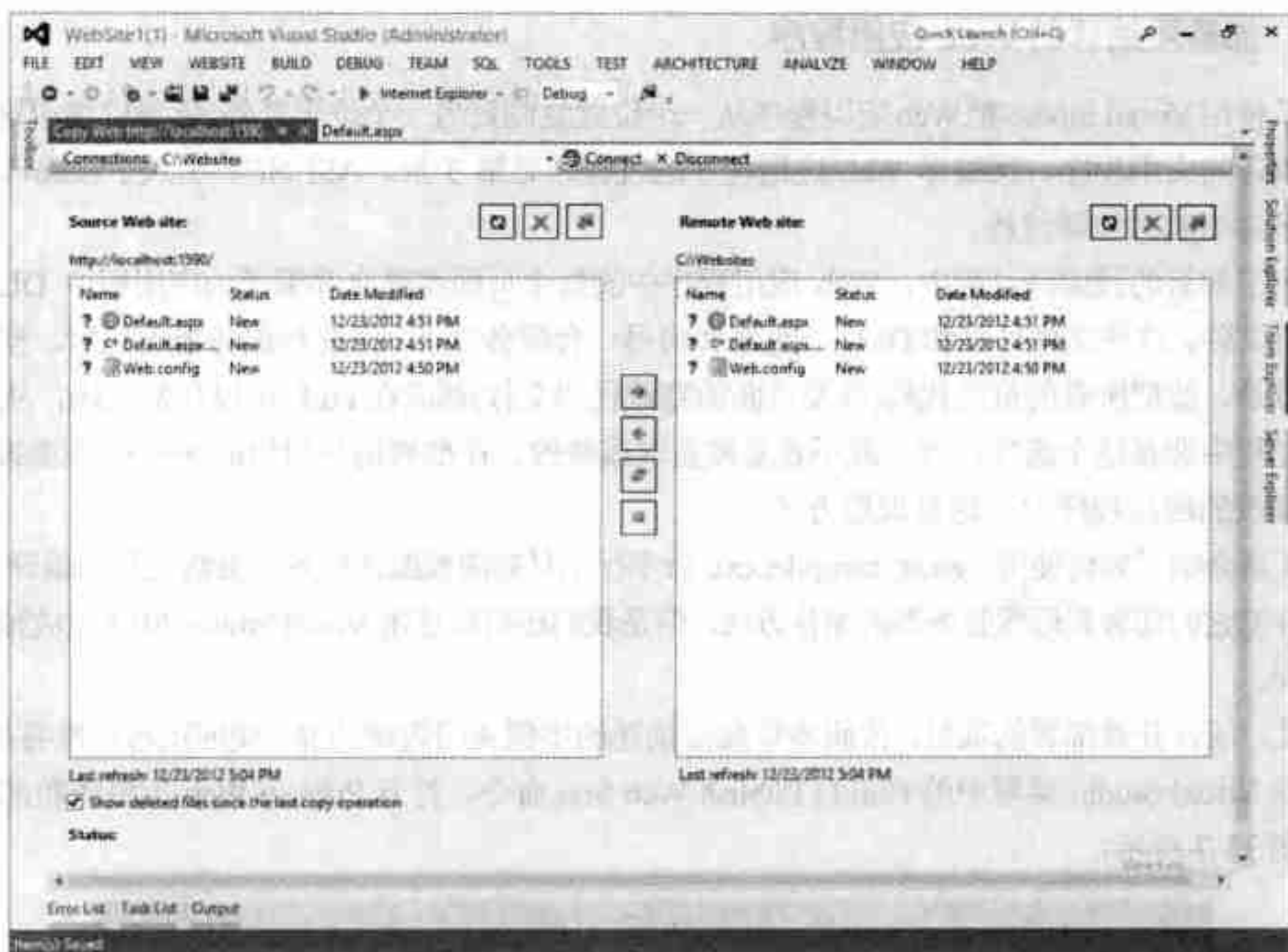


图 33-5

如果修改文件，之后再打开这个复制对话框，就会在已修改的文件旁边看到箭头图标，表示该文件比目标服务器上的文件要新，如图 33-6 所示。

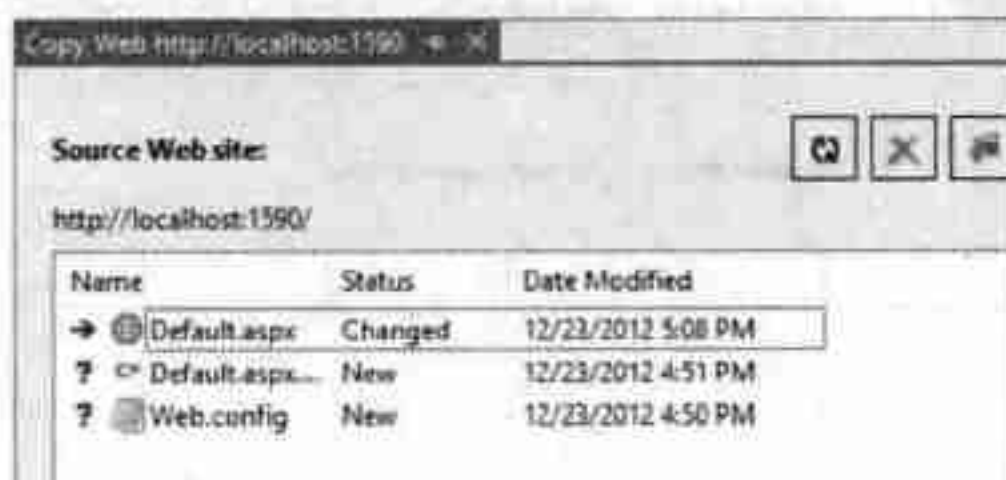


图 33-6

这些箭头仅允许用户选择必须复制的文件。所有的复制操作都会在日志文件中记录下来。要查看日志文件的内容，可以在 Copy Web Site 对话框的底部单击 View Log 按钮，打开 CopyWebSite.log 文本文件。从以前执行的复制操作中，可以看到已完成的事务。示例日志条目如下：

```
Copy from 'C:\Websites\Websitel' to 'E:\Websitel' started at 10/6/2009 7:52:31 AM.
Create folder App_Data in the remote Web site.
Copy file Default.aspx from source to remote Web site.
Copy file Default.aspx.cs from source to remote Web site.
Copy file About.aspx from source to remote Web site.
Copy file About.aspx.cs from source to remote Web site.
Copy file web.config from source to remote Web site.
Copy from 'C:\Websites\Websitel' to 'E:\Websitel' is finished.
Completed at 10/6/2009 7:52:33 AM.
```


33.3.3 部署预编译的 Web 应用程序

除了使用 Visual Studio 把 Web 应用程序从一个位置复制到另一个位置之外,还可以使用这个 IDE 部署预编译的应用程序。预编译 Web 应用程序的过程详见第 3 章。ASP.NET 引入了预编译过程,称为用于部署的预编译过程。

在用于部署的预编译过程中,Web 应用程序中的每个页面都建立并编译为应用程序 DLL 和一些占位符文件。这些文件可以和 DLL 一起部署到另一台服务器上,并在该服务器上运行。预编译过程的优点是,能把所有的页面代码(以及页面的隐藏代码文件)都放在 DLL 中以打乱代码,从而如果在编译过程中选择这个选项,代码就不容易被盗取或修改。在部署用户付费的应用程序或部署后完全不会修改的应用程序时,这是理想方式。

第 3 章介绍了如何使用 `aspnet_compiler.exe` 命令行工具完成预编译任务。虽然这是预编译 Web 应用程序并把它们部署到远程服务器的绝佳方法,但是我们还可以使用 Visual Studio 2012 完成预编译和部署过程。

为此,应打开要部署的项目,按照本章前面描述的步骤关闭调试功能,使应用程序准备好部署。然后选择 Visual Studio 菜单中的 Build | Publish Web Site 命令,打开 Publish Web 预编译和部署对话框,如图 33-7 所示。

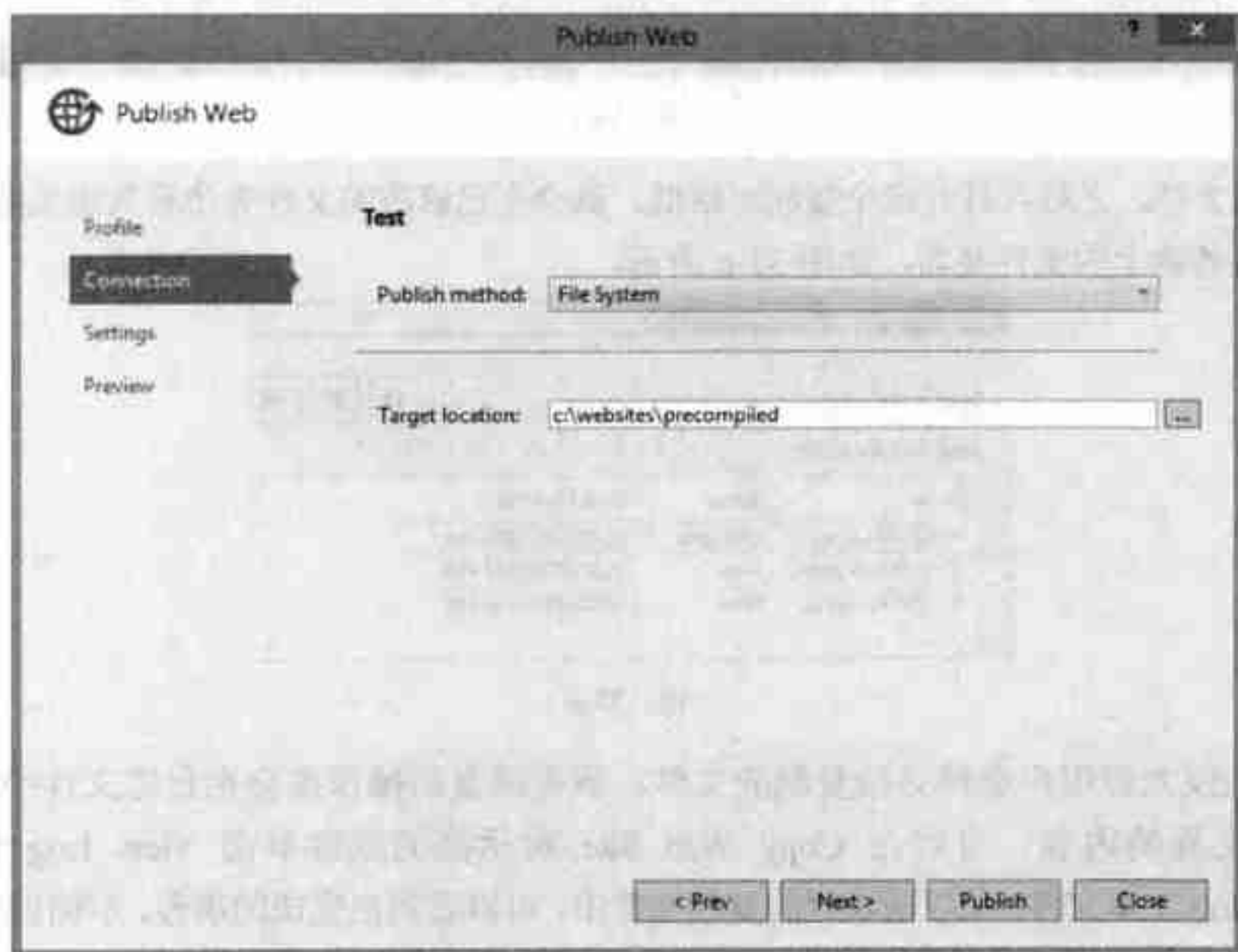



图 33-7

单击这个对话框中的  按钮,可以选择要部署应用程序的远程位置。与前面的例子一样,可用的选项有文件系统位置、本地 IIS 中的位置、使用 FTP 访问的位置或通过 FrontPage Server Extensions 访问的位置。

本章后面的图 33-16 显示了 Advanced Precompile 设置窗口,在其中可以看到这个窗口中的其他选项,例如 Allow precompiled site to be updatable 复选框。选中该复选框时,将编译和复制站点,但不对.aspx 页面进行任何修改。也就是说,在预编译过程后,仍然可以对底层的页面进行细小修改,

例如修改 Web 页面的标记,而应用程序会正常工作。如果未选中这个复选框,就会将页面中的所有代码剥离,并放在一个或多个 DLL 中。在这种状态下,不能更新应用程序,因为不能在编译过程中更新任何占位符文件。

该对话框中的另一个选项是为在该过程中创建的 DLL 指定强名。可以选中相应的复选框,指定在标记过程(signing process)中使用的键。这样,预编译过程创建的 DLL 就称为强程序集——使用我们选中的键来标记。强名确保名称的唯一性依赖唯一键对,并提供了强名的完整性检查。通过 .NET Framework 安全检查,可以确保程序集中的内容自从建立以来就没有改变过。

准备好部署后,单击对话框中的 OK 按钮,建立并发布打开的应用程序。“发布”表示将应用程序部署到指定的位置。我们已在指定的位置添加了 bin 目录,其中包含预编译的 DLL,图 33-8 中给出了我们的 Web 应用程序。



图 33-8

在这种状态下,将包含在 ASP.NET 页面中的代码剥离,并放在 DLL 中。我们看到的文件只是 DLL 用于引用的占位符。

33.3.4 创建 ASP.NET Web Package

部署 Web 应用程序的最简单方式是使用 Visual Studio 2012 中新的内置发布功能。在后台,该功能使用微软的 Web 部署工具——Web Deploy,这意味着如果希望使用该功能把应用程序部署到服务器上,那么服务器就必须包含 Web Deploy,从而应用程序才能真正工作起来。

创建和传送的程序包实际上是物理文件——.zip 文件,该文件包含了在新环境中使用其所有设置重新部署 Web 应用程序所需的所有内容。除了 .zip 文件外,该程序包还包括用于在主机上初始化程序包安装的清单文件和命令文件。

部署应用程序的首选方式是使用 Visual Studio 2012 的一键发布功能。在 Visual Studio 的 Solution Explorer 中右击项目并从弹出的菜单中选择 Publish 命令，这个选项可用于 Web Site 项目和 Web Application 项目，打开 Publish Web 对话框，如图 33-9 所示。

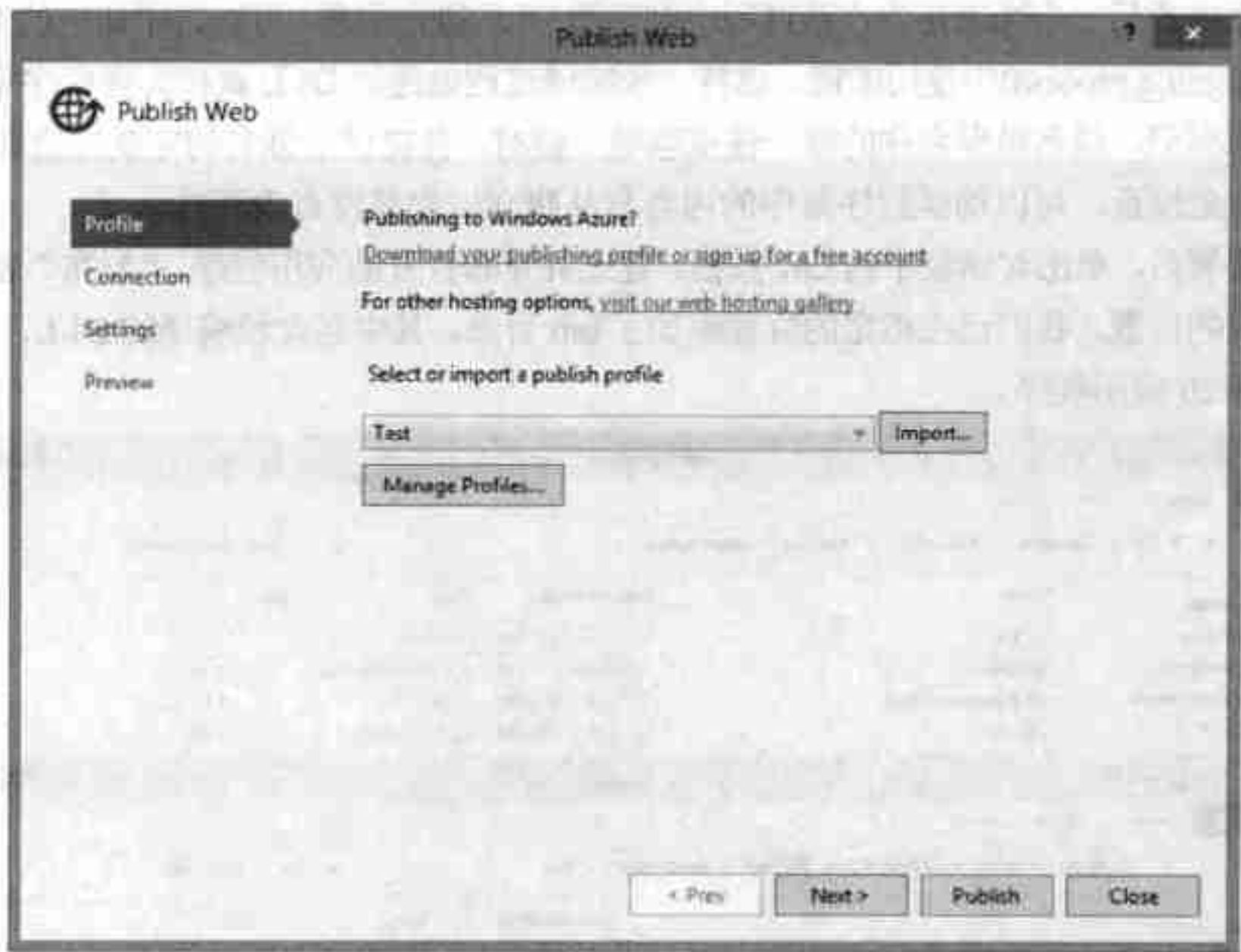


图 33-9

因为许多开发人员都一直要把 Web 应用程序部署到他们的测试、开发和产品环境中，所以可以把部署选项存储到配置文件中，以便反复使用。在选择 Publish 命令之前，在 Visual Studio 2012 工具栏中进行该设置。

使用发布向导发布应用程序时，可以配置表 33-2 中的设置。

表 33-2

设 置	说 明
Profile Name	保存的配置文件的名称。对于重复部署，可以重用配置文件中的设置作为默认设置
Bulid Configuratioin	指定生成编译是在 Debug 模式下还是在 Release 模式下完成。这个选项不限于 Debug/Release，而可以给项目使用任意构建配置文件
Publish Method	执行部署时要使用的方法。可能的选项包括 Web Deploy Publish、FTP、File System 和 FPSE(FrontPage Server Extensions)
Service URL	指定 Web Deploy 在主机服务器上的位置。这个 URL 指向具有 Web Deploy 功能的当前 IIS 处理程序，结构类似于 http://myhostserver:8172/MsDeploy.axd
Site/Application	指定部署应用程序的位置。这里可以指定放置应用程序的站点以及虚拟目录。例如，MyDomain.com/MyApplication

(续表)

设 置	说 明
Remove additional files at the destination	如果没有选择该选项，那么一键发布功能在应用文件和设置之前将首先删除主机服务器所在位置的所有内容
User Name	用于 IIS 连接的用户名
Password	用于 IIS 连接的密码
Save password	指定 Visual Studio 发布功能是否在配置文件中保存密码
Destination URL	成功发布后，在浏览器中打开目的地 URL
File Publish Options	指定要在目标服务器上发布或删除哪些文件
Databases	指定是运行 Entity Framework Code First Migrations，还是递增式发布数据库
Preview	预留已发布到服务器的文件

除了使用 Web Deploy 选项外，也可以方便地使用 Publish Web 对话框提供的其他选项。图 33-10 显示了该对话框提供的其他 3 个部署选项的设置。

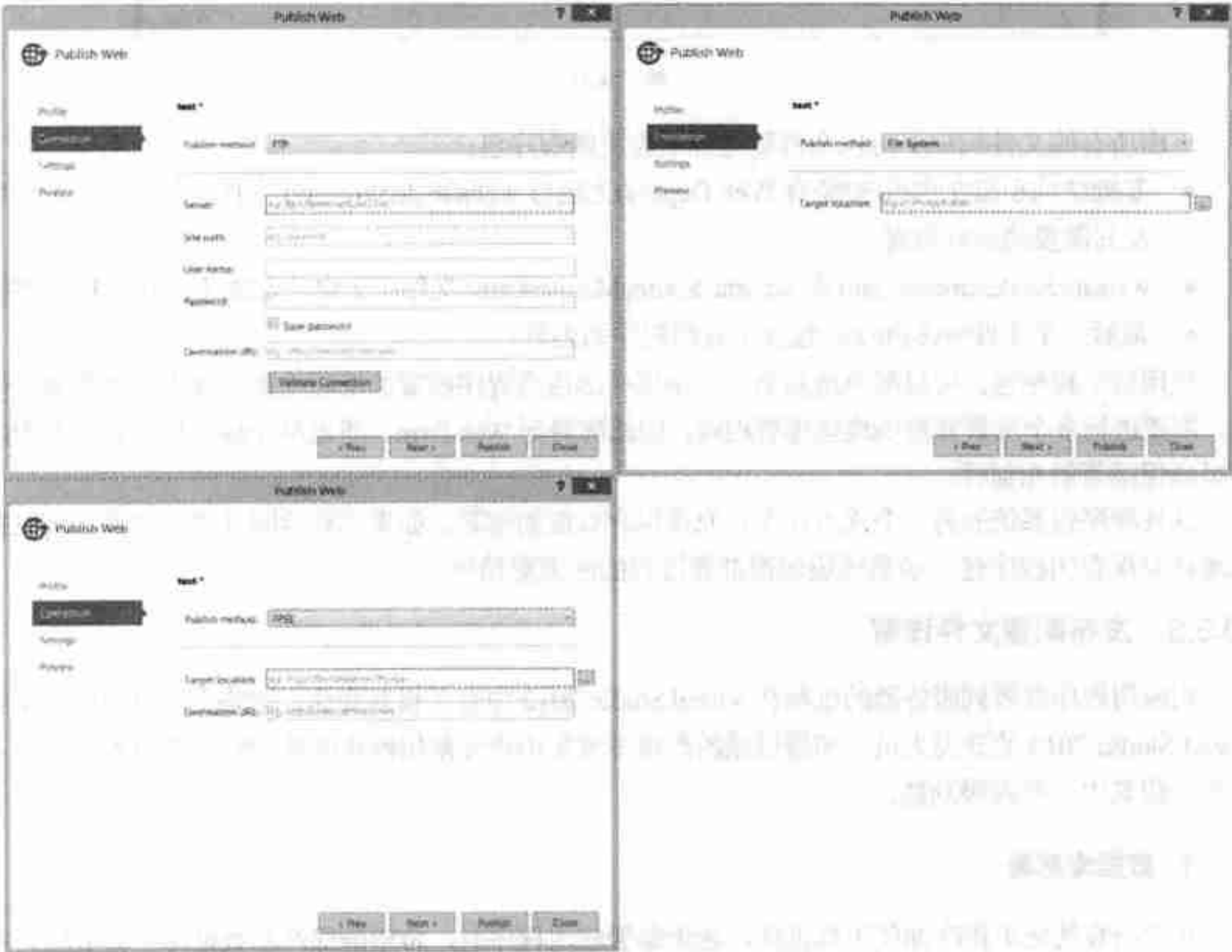


图 33-10

在图 33-10 中可以看到使用 FTP、文件系统或 FrontPage Server Extensions 进行部署的一些标准设置。在 Visual Studio 2012 中使用 Web Deploy 功能时，一件有趣的事情是不再只是连接远程的 Web Deploy 处理程序并实时运行部署，而是可以创建可随时运行的 Web Deploy 程序包。Visual Studio 2012 允许创建一些程序包，这些程序包可以通过电子邮件或其他方式提供给其他

人，以便在他们的系统中运行。要创建 Web 部署程序包，可以在 Publish 菜单中选择 Web Deploy Package 选项。选择这个选项并指定创建程序包的位置后，Visual Studio 状态栏中会显示创建程序包的过程。获得发布成功的通知之后，就可以在发布窗口指定的位置找到整个程序包。

该文件夹包含了构成程序包的所有文件，如图 33-11 所示。

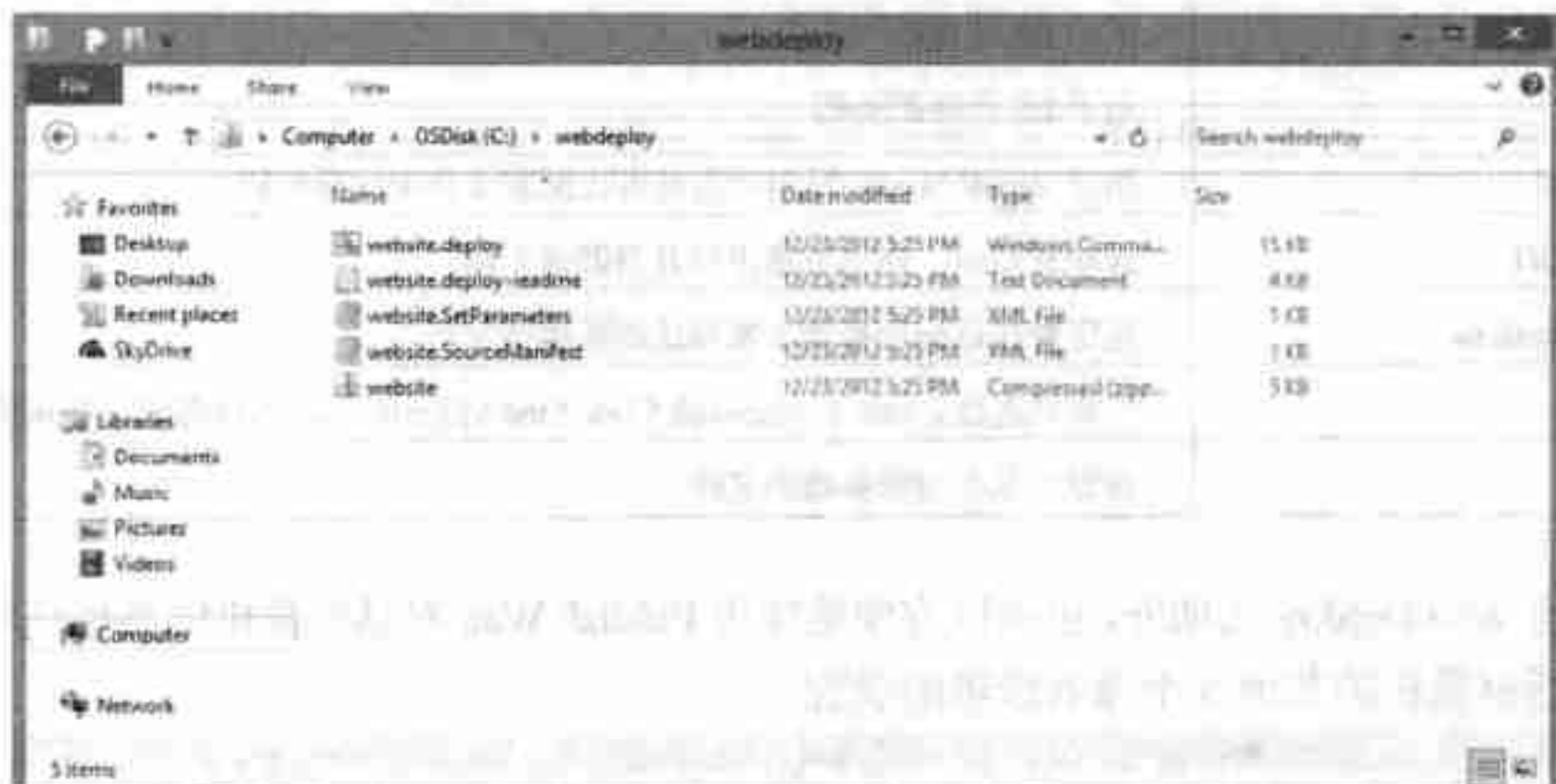


图 33-11

下面所有的文件构成了可以在部署过程中使用的程序包：

- 基础结构小组在主机(安装有 Web Deploy)上运行 website.deploy.cmd 文件，以安装应用程序及其需要的所有设置。
- website.SetParameters.xml 和 website.SourceManifest.xml 文件用于定义安装过程中使用的设置。
- 最后一个文件 website.zip 包含了应用程序的内容。

使用这个程序包，可以简单地向另一个小组传送应用程序所需的安装文件。该用法的优秀示例是，需要进行多个部署并希望这些部署相同，因此部署到 Web farm。通过单个程序包运行安装程序可以确保部署的相似性。

这种程序包系统的另一个优点在于：允许保存以前的部署。如果需要返回并确认部署，可以轻松获取保存的程序包。该系统还使得部署过程的回滚更简单。

33.3.5 发布配置文件详解

把应用程序部署到服务器的过程在 Visual Studio 2012 中有了很多变化。这些变化也可用于使用 Visual Studio 2010 的开发人员。部署领域的改进主要集中在使常用操作的执行和扩展更容易完成。本节介绍其中一些高级功能。

1. 数据库部署

几乎所有的应用程序都使用数据库。逐步部署应用程序时，希望确保产品数据库不被破坏，在开发机器上部署应用程序时，可以使用最新的模式变化更新数据库。

Entity Framework Code First

Entity Framework 引入了 Code First Migrations，Code First Migrations 有助于逐步地更新数据库，而不是在每次部署应用程序时，都必须重建数据库。可以定制 Code First Migrations，指定如何逐步

地更新已有数据库的模式和数据。使用 Visual Studio 部署应用程序时，可以确定在部署应用程序时是否执行 Code First Migrations。图 33-12 显示了 Publish Web 对话框的 Settings 选项卡，在该图中，选择了 Execute Code First Migrations 选项，这表示在服务器上部署应用程序时，执行 Code First Migrations。

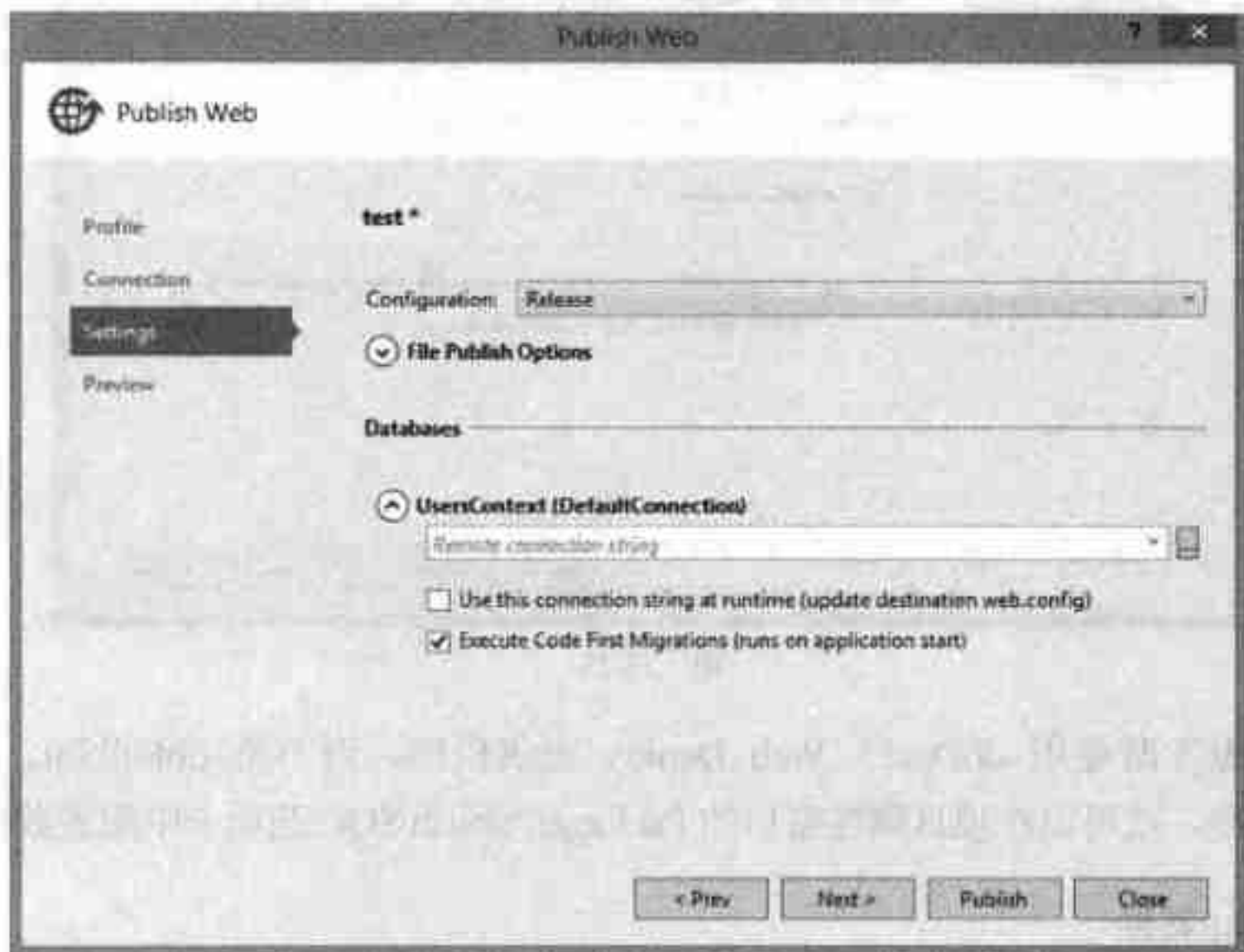


图 33-12



Entity Framework Code First 详见第 11 章。

部署应用程序时，Visual Studio 在部署过程中不会对数据库做任何处理。第一次访问数据库时，Code First 会自动创建数据库，并根据需要更新模式。如果应用程序实现了 Seed 方法，就在创建或更新数据库后执行 Seed 方法。

递增式数据库更新

即使应用程序没有使用 Entity Framework Code First，也仍可以使用 dbDacFx Web Deploy 提供程序更新数据库。这个提供程序使用新的 Data-Tier Applications Framework (Dacpac Framework) 来同步数据库。dbDacFx 提供程序是对 dbFullSql 提供程序的改进。dbFullSql 提供程序在 Visual Studio 2010 中使用，用于每次部署应用程序时对数据库进行完整同步。这表示，如果修改已有的表，提供程序对数据库进行完整同步时就会出错。dbDacFx 进行了改进，因为这个提供程序会比较两个数据库，计算需要应用于目标数据库的修改，使之类似于源数据库。

图 33-13 说明了如何使用 dbDacFx 提供程序。这个选项用于应用程序不使用 Entity Framework Code First 的 Context 类来访问的 SQL Server 数据库。这个选项还允许指定在部署过程中应运行在目标数据库上的定制 SQL 脚本。当希望给表提供初始的种子数据时，就可以使用定制 SQL 脚本。

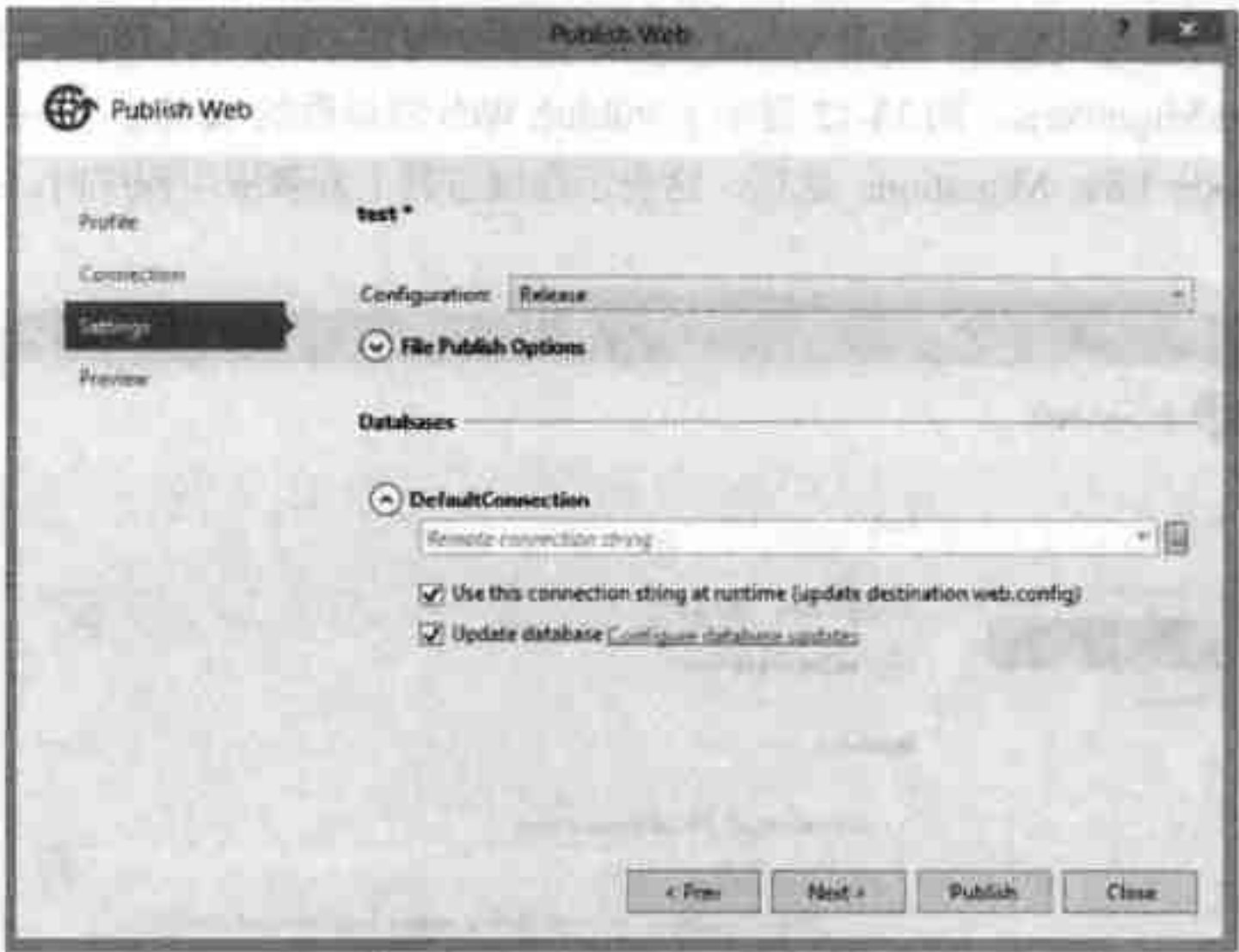


图 33-13

建议部署数据库时使用 dbDacFx Web Deploy 提供程序，而不是 dbFullSql。如果希望使用 dbFullSql 提供程序，就可以在项目属性窗口的 Package/Publish SQL 选项卡中配置数据库部署设置。

2. web.config 转换

发布应用程序时，两个最常见的操作是根据是发布到产品服务器还是测试服务器，转换 web.config 的一些特性，发布特定的 web.config。例如，如果要发布到产品服务器，就关闭应用程序的调试选项。

在 Visual Studio 中创建 Web 应用程序时，会得到 web.config 文件。展开 web.config 文件，其中有两个子文件——web.debug.config 和 web.release.config。图 33-14 显示了这两个文件。

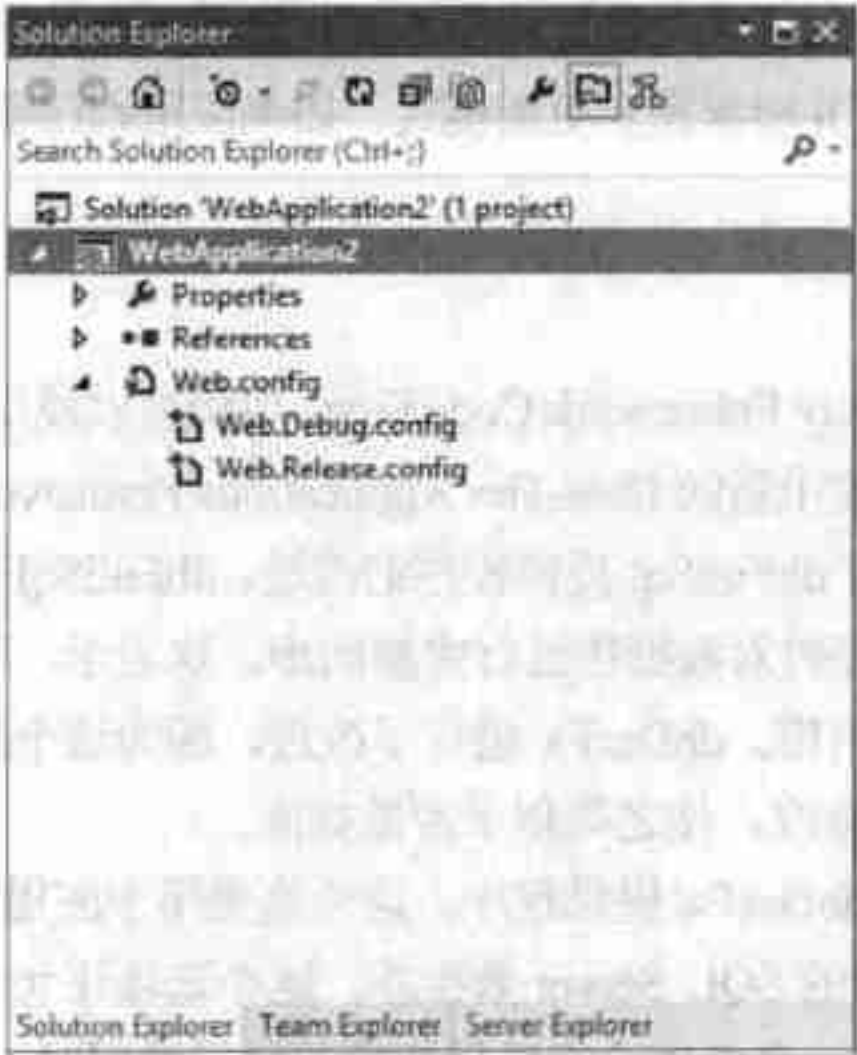


图 33-14

使用 Visual Studio 部署应用程序时，可以选择构建项目使用的构建配置，以及该配置是否有相

关的转换。图 33-15 指出如何在 Publish Web 窗口的 Settings 选项卡中配置这个选项。在该图中，使用 Release 配置来发布任何应用程序。这表示发布时，Visual Studio 会以 Release 模式发布应用程序，禁用 web.config 中的调试标记，如程序清单 33-15 所示。

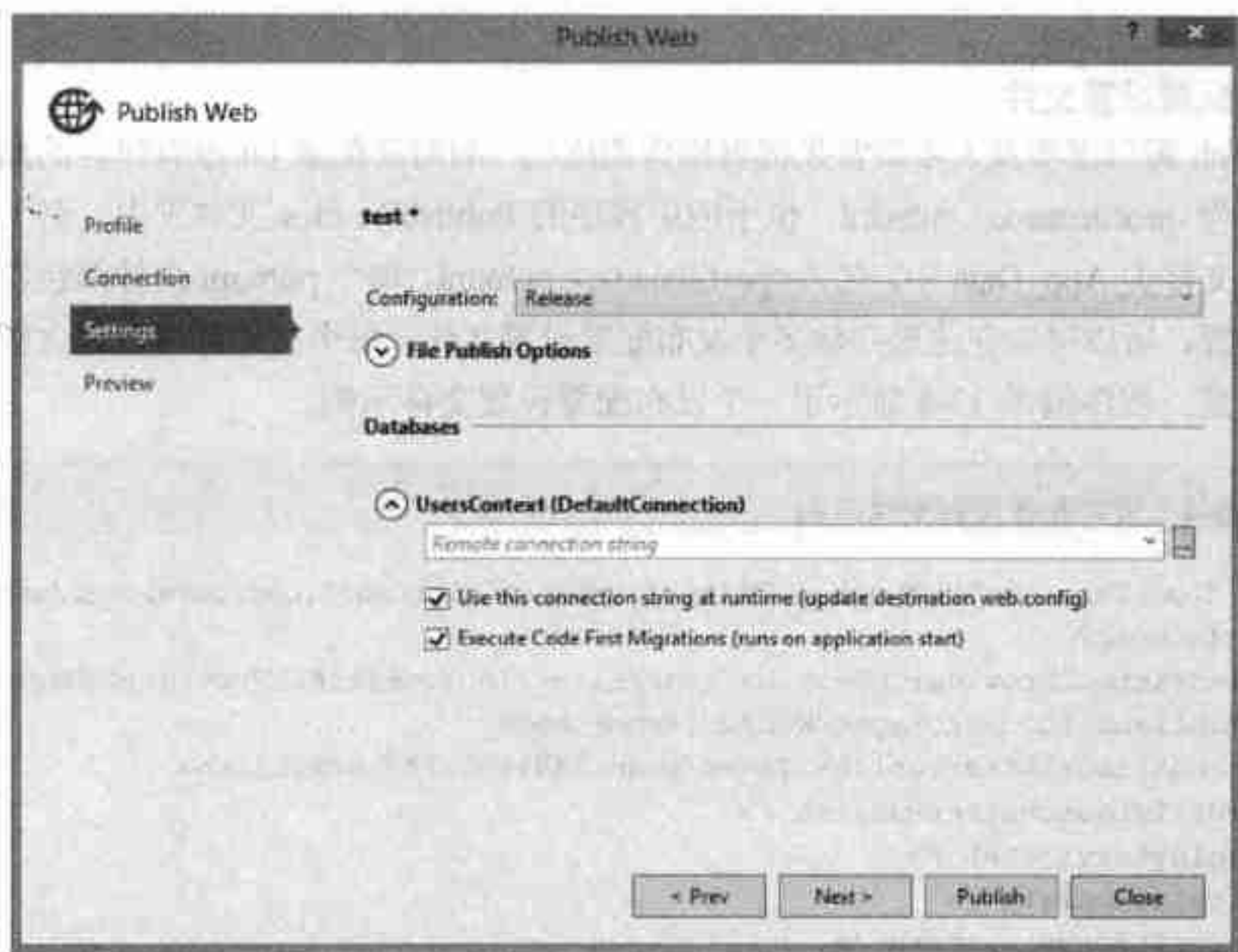


图 33-15

这些文件最重要的一项功能是部署应用程序时，指定 Visual Studio 如何转换这些文件。转换是修改 web.config 文件中不同设置的一项关键功能，程序清单 33-2 显示了比较通用的 web.config 文件，其中的 debug 选项是打开的。程序清单 33-3 显示了如何应用转换，关闭 web.config 中的调试功能。

程序清单 33-2 一般的 web.config 文件

```
<system.web>
  <compilation debug="true" />
</system.web>
```

程序清单 33-3 web.release.config 转换关闭了调试功能

```
<system.web>
  <compilation xdt:Transform="RemoveAttributes(debug)" />
</system.web>
```

转换操作使用 XML-Document-Transform 名称空间中定义的 XML 特性来指定，该名称空间映射到 xdt 前缀。XML-Document-Transform 名称空间定义了两个特性——Locator 和 Transform:

- Locator 特性指定要以某种方式修改的一个或一组 web.config 元素。
- Transform 特性指定要对 Locator 特性找到的元素执行什么操作。

使用转换功能可以执行其他转换，例如启用登录功能或设置定制错误。

3. 其他发布选项

Publish Web 窗口除了支持部署时的大多数常见操作之外，还有一些高级设置，允许执行更高级的操作。

编辑发布配置设置文件

Publish Web 窗口仅涉及大多数常见部署操作的皮毛，有时要配置 UI 没有包含的高级操作。发布配置文件名为<profilename>.pubxml，位于应用程序的 PublishProfiles 文件夹中。对于网站项目，这些文件位于文件夹 App_Data 中，名为<profilename>.pubxml。每个 pubxml 文件都包含应用于发布配置文件的设置，所以可以在这里存储多个发布配置设置文件，每个文件都可以包含应用程序的不同发布目标信息。程序清单 33-4 显示了一个发布配置设置文件示例。

程序清单 33-4 发布配置设置文件示例

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <IncludeSetACLProviderOnDestination>False</IncludeSetACLProviderOnDestination>
    <WebPublishMethod>Package</WebPublishMethod>
    <LaunchASiteUrlAfterPublish>True</LaunchASiteUrlAfterPublish>
    <SiteUrlToLaunchAfterPublish />
    <MSDeployServiceURL />
    <DeployIisAppPath />
    <RemoteSitePhysicalPath />
    <AllowUntrustedCertificate>False</AllowUntrustedCertificate>
    <SkipExtraFilesOnServer>True</SkipExtraFilesOnServer>
    <DeployAsIisApp>True</DeployAsIisApp>
    <MSDeployPublishMethod>WMSVC</MSDeployPublishMethod>
    <UserName />
    <SavePWD>True</SavePWD>
    <PublishDatabaseSettings>
      <!-- this section omitted to keep the example short -->
    </PublishDatabaseSettings>
  </PropertyGroup>
</Project>
```

发布配置设置文件是 MSBuild 文件，所以可以编辑这个文件，配置一些选项。例如，如果在服务器上控制 ACL，希望在部署时禁用默认的 ACL 操作，就可以删除 IncludeSetACLProviderOnDestination 元素。Visual Studio 没有设置默认的 ACL(它们在应用程序根目录下是只读的，对 App_Data 文件夹有写入权限。)

高级预编译设置

如果预编译 Web 应用程序，就在预编译窗口中配置高级设置。要启动这个窗口，可以在 Publish Web 窗口的 Settings 选项卡中展开 File Publish Options。图 33-16 显示了 Advanced Precompile Settings 窗口。

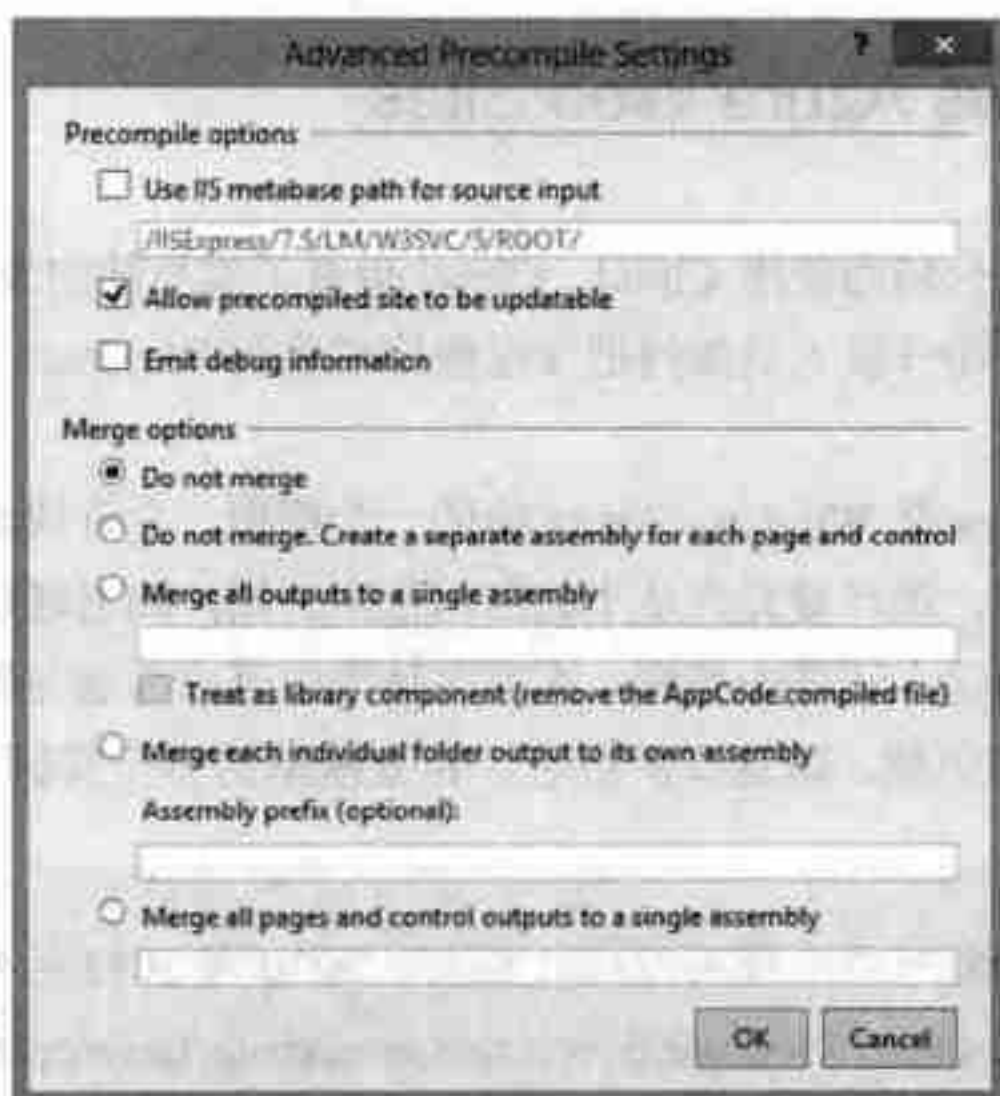


图 33-16

Package/Publish Web

Publish Web 窗口包含 Web 部署需要的大多数常见设置，但某些设置可以在项目属性的 Package/Publish Web 选项卡中配置。图 33-17 显示了 Package/Publish Web 选项卡。使用这个选项卡可以配置应转换的 IIS 设置，例如应用程序池设置。

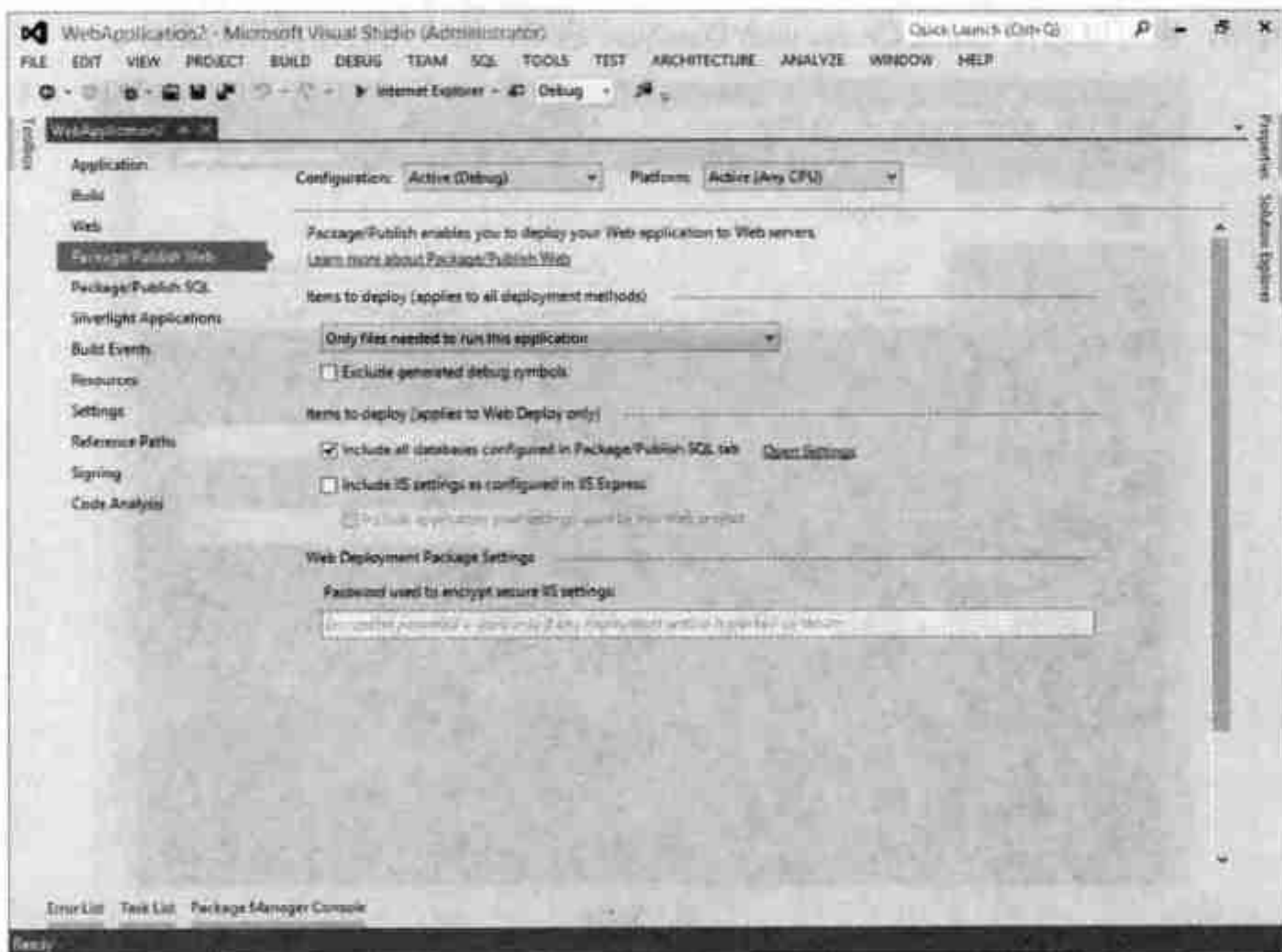


图 33-17

33.4 部署 Windows Azure Web Sites

随着越来越多的宿主平台转而使用 Cloud, Cloud 也有了实质性的改进, 更容易把 Web 应用程序部署到 Cloud 中。本节介绍开发人员如何把应用程序部署到 Windows Azure, 使开发机器上的应用程序可用于用户。

Windows Azure Web Sites 是 Windows Azure 中的一个模型, 允许快速、方便地把站点部署到可伸缩性很高的 Cloud 环境中。该环境允许从小的应用程序开始, 随着通信量的增加而扩展。这个模型提供的功能可以在共享的宿主环境中找到, 在宿主环境中可以与服务器上的其他网站共享资源, 一旦应用程序开始有更多的负载, 就逐步扩展它, 建立网站的多个实例。



改进的发布功能可以由使用 Visual Studio 2010 或 2012 的开发人员使用。最新的 Azure SDK 可以在 <http://www.windowsazure.com/en-us/develop/net/> 上下载。

在创建新的网站之前, 需要在 Windows Azure 上创建一个账户。要创建账户, 可以进入 Windows Azure 管理入口 <http://www.windowsazure.com/>, 注册以创建账户。

要把应用程序部署为 Windows Azure Web Site, 就必须在 Windows Azure 上提供网站。图 33-18 说明了如何使用 Windows Azure 管理入口中的 Quick Create 选项创建网站。可以在地理上最近的位置创建网站, 也可以选择 Quick Create with Database 选项, 给应用程序提供数据库。



图 33-18

一旦创建网站, 就可以把发布配置文件下载到开发机器上。可以从管理入口中选择网站, 再选择 Dashboard 选项卡。这个选项卡列出了网站的使用统计数据 and 所有可用的配置选项。可以从这个部分下载发布配置文件。也可以把发布设置文件下载并保存到计算机的任意位置。图 33-19 显示了

Windows Azure 管理入口中的 Download Publish Profile 选项，使用它可以下载发布配置文件。发布配置文件包含连接网站和数据库需要的连接设置。



图 33-19

下载发布配置设置文件后，就可以进入应用程序，导入这些设置。一旦启动 Publish Web 窗口，就可以导入刚才下载到机器上的发布配置设置文件。导入发布配置文件后，Visual Studio 就会读取该文件，获得 Windows Azure 中网站和数据库的连接值。一旦导入发布配置文件，Visual Studio 就会把该文件保存为项目的一部分。这表示，发布配置设置文件已被添加到项目中，可以把这个文件和项目一起放在任意源控制系统中。这很有用，因为如果把项目复制到另一台机器上，并希望重新发布应用程序，就不必下载发布配置设置文件了。右击网站，就可以启动 Publish Web 窗口，在该窗口中可以导入配置文件，如图 33-20 所示。

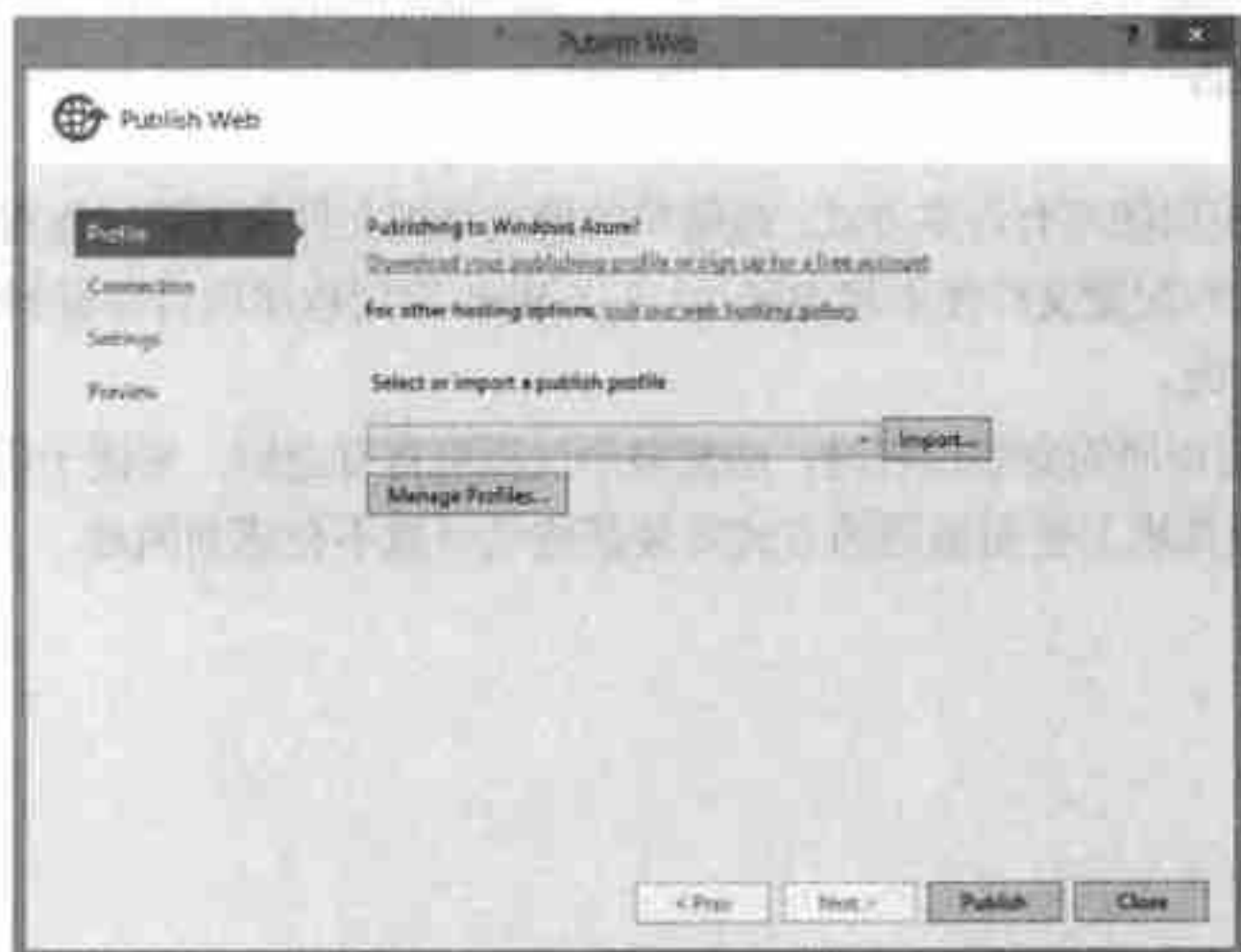


图 33-20

在 Publish Web 对话框中有 Connection 选项卡，单击 Validate Connection 按钮，可以验证发布配置设置。程序尝试连接所提供的 Azure Web Site，确定发布配置设置文件中提供的连接是否有效。

如果应用程序使用数据库，就可以在 Settings 选项卡中设置部署应用程序时使用的连接字符串。如果使用 Entity Framework Code First，就可以配置在把应用程序部署到 Azure Web Sites 时，是否应运行 Code First Migrations。

把应用程序部署到 Windows Azure 的最后一步是设置 Preview 选项卡。这个选项卡显示部署了哪些文件，使用这一步可以检查要发布的所有文件是否都发布了。还可以取消选择不想发布的文件。图 33-21 显示了 Preview 选项卡。

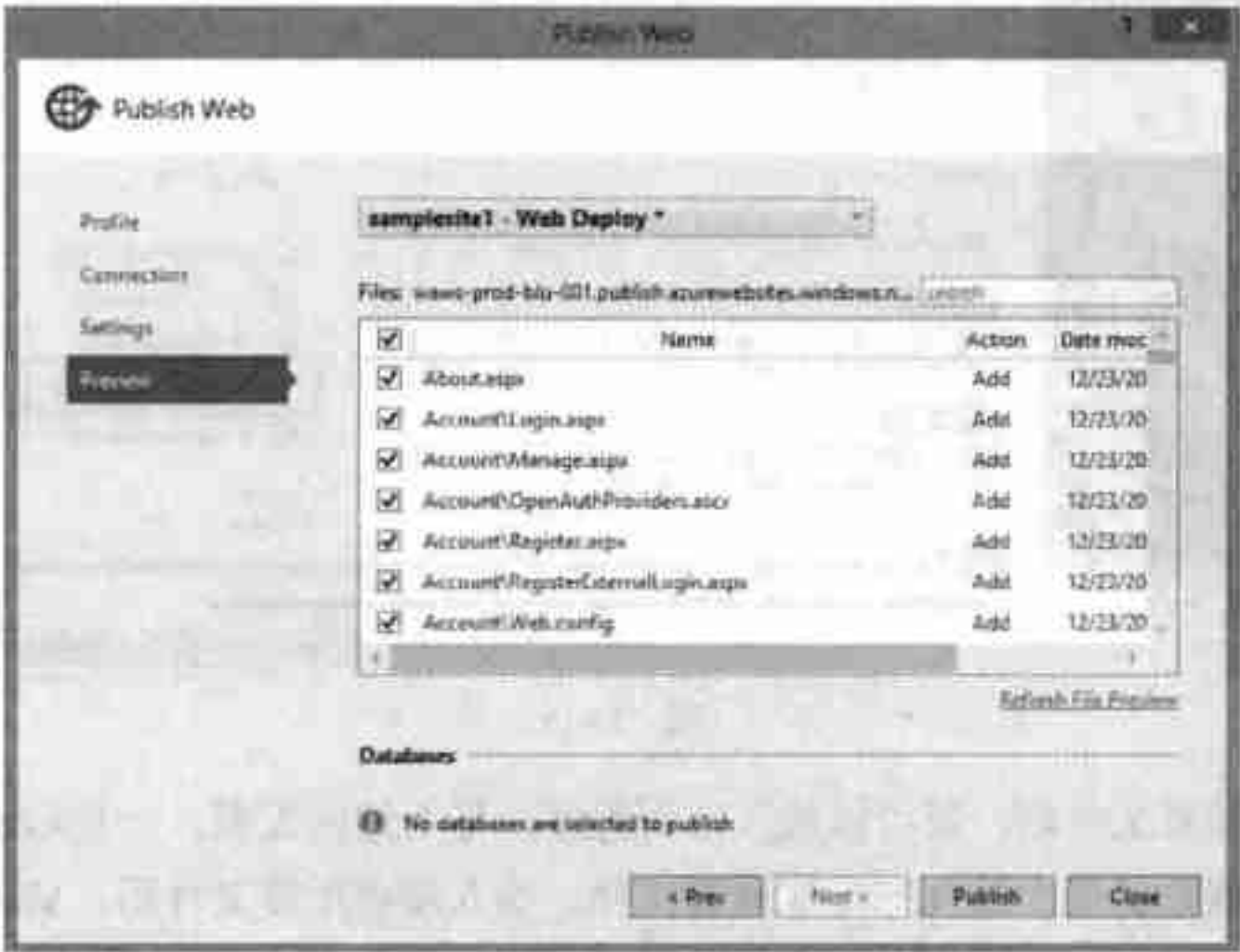


图 33-21

单击 Publish 按钮后，所有的文件就被复制到服务器上，数据库用本地数据库中的模式更新，或通过 Entity Framework Code First Migrations 更新。如果部署成功，就用网站的 URL 启动默认浏览器，应用程序就部署到 Cloud 中。

33.5 本章小结

安装 ASP.NET 应用程序有许多方式：最简单的模式是把文件复制到远程服务器上，这是一种保存并运行的模式；发布配置文件便于把 Web 站点或 Web 应用程序项目部署到任意服务器，尤其是使用 Windows Azure 时。

在处理 Web 应用程序的安装过程时，应使整个过程更富有逻辑，更便于用户和开发团队理解，这样用户在另一台计算机上使用编程的方式安装某些项时就不会感到困难。

第Ⅸ部分

其他ASP.NET技术

- 第 34 章 ASP.NET MVC
- 第 35 章 ASP.NET Web Pages 和 Razor

第 34 章

ASP.NET MVC

本章要点

- 理解 MVC 和 ASP.NET
- 使用路由和 URL
- 理解控制器和视图

很多年来, MVC(Model-View-Controller, 模型-视图-控制器)一直都是计算机科学领域非常重要的体系结构模式。1979 年, 它最初被命名为 Thing-Model-View-Editor, 随后简化为 Model-View-Controller。在分离应用程序中的关注点时, 该模式是一种强大且灵活的方式, 而且它在 Web 应用程序中运用得非常好。MVC 的关注点分离虽然向应用程序的设计添加了少量额外的复杂性, 但是其特有的优势超过了使用其所付出的代价。自引入以来, MVC 被还应用到了很多其他的架构中。在 Java 和 C++ 语言、Mac 和 Windows 操作系统以及许多架构的内部都可以找到 MVC。

理解 MVC 的核心概念对于高效地使用它至关重要。本章将讨论 MVC 模式的发展史以及现在如何将它应用到 Web 编程中。

ASP.NET MVC 1.0 是 Visual Studio 2008 中一个可下载的插件。ASP.NET MVC 2 内置于 Visual Studio 2010 中。ASP.NET MVC 3 是 Visual Studio 2010 中一个可下载的插件。目前, ASP.NET MVC 4 内置在 Visual Studio 2012 中。本章还将介绍 ASP.NET Web 窗体的一些使用限制以及 ASP.NET MVC 是如何试图将开发人员从这些限制中解放出来的。

34.1 MVC 的定义

MVC 是用于将应用程序分成如下 3 个主要方面的体系结构模式:

- 模型(Model): 一组类和业务规则, 类用于描述正在处理的数据, 业务规则用于说明如何修改和操纵这些数据。
- 视图(View): 应用程序的用户界面(UI)。

- **控制器(Controller):** 一组类, 用于处理来自用户、整个应用程序流以及特定应用程序逻辑的通信。

该模式经常用于 Web 编程。对于 ASP.NET MVC 而言, 可以将该模式大致地解释为如下:

- **模型**是表示你感兴趣的域的类。这些域对象经常封装存储在数据库中的数据以及代码, 这些代码用于操纵这些数据并实施特定域的业务逻辑。对于 ASP.NET MVC 而言, 模型可能是使用了某个工具的一种数据访问层, 这个工具可以是 LINQ to SQL、Entity Framework, 也可以是与包含特定域逻辑的定制代码组合在一起的 NHibernate。
- **视图**是动态生成的页面。在 ASP.NET MVC 中, 通过继承于 `System.Web.UI.Page` 类的 `System.Web.Mvc.ViewPage` 来实现视图。
- **控制器**是一种特殊的类, 用于管理视图和模型之间的关系。控制器与模型通信, 并决定显示哪个视图(如果存在的话)。在 ASP.NET MVC 中, 通常以后缀“Controller”来表示这种类。

34.2 当今 Web 上的 MVC

从 1993 年的 Mosaic 开始, 直到图形化的浏览器开始占据整个市场, Web 才真正开始变得日益流行起来。随后不久, 动态的 Web 页面开始出现, 使用的语言包括 Perl, 支持的技术包括 CGI(Common Gateway Interface, 公共网关接口)。Web 初期可供使用的技术更多地关注于 HTML 脚本, 以减轻内容驱动这种开发方式(与深奥的应用程序逻辑相对)的工作量, 这种开发方式后期已经很少使用。

随着 Web 的发展以及 HTML 标准开始支持更加丰富的交互功能, 作为应用程序平台的 Web 概念开始逐渐明朗。在微软王国, 关注点集中在快速和简单上, 动态服务器页面(Active Server Page, ASP)于 1996 年应运而生。

ASP 使用 VBScript 语言, VBScript 是一种非常简单的、轻量级的语言。在创建应用程序时, 它为开发人员提供了很多“难以形容的自由”。对 ASP 页面的请求都是通过带有 .asp 扩展名的文件来处理的, 该文件由与 HTML 标记混合在一起的服务器端脚本构成。由于是使用这种面向过程的编程语言编写的, 因此许多 ASP 页面的代码通常十分混乱。在这种代码中, 标记以一种难以管理的方式与代码混合在一起。尽管编写简洁的 ASP 代码是可行的, 但是需要花费很大的精力, 而且语言和工具还不足以提供有效的帮助。即便如此, ASP 确实提供了对所生成标记的完全控制, 只是需要花费很多的精力而已。

2002 年 1 月, 微软发布了 .NET 平台的 1.0 版本。该版本包括了 ASP.NET 的最初版本, 因此 Web 窗体也随之诞生, 从而提供了对高级工具和用于构建 Web 站点的面向对象语言的支持。

ASP.NET 在过去的 10 年里得到了巨大的发展。通过将 Web 开发中重复的任务抽象为简单的拖放控件, ASP.NET 使得 Web 开发的效率大大提高。这种抽象能够起到很大的作用, 但是一些开发人员已经发现他们想要的不仅仅是生成 HTML 和浏览器脚本的控件, 他们还希望能够简单地测试 Web 页面的逻辑。

随着语言的不断成熟和 Web 服务器软件在功能上的不断改进, 人们很快就发现 MVC 在 Web 应用程序体系结构中的作用。但是 MVC 直到 2004 年 7 月才开始占据主流地位, 此时一位 24 岁的开发人员 David Heinemeier Hansson(家住美国伊利诺伊州芝加哥市的 37Signals)向世界介绍了他所想象的 MVC。

David(在社区中称为 DHH)创建了 Ruby on Rails, 这是一个 Web 开发架构, 该架构使用 Ruby 语言和 MVC 模式创建特殊的 Web 程序。

现在进一步钻研 ASP.NET MVC, 并且回答如下问题: “为什么不是 Web 窗体?”。

2007 年 2 月, 微软公司的 Scott Guthrie 在飞往美国东海岸参加会议的飞机上勾画出了 ASP.NET MVC 的核心。这是一个简单的应用程序, 包含了几百行代码, 但是它向部分 Web 开发人员提供的前景和潜力是巨大的。

34.3 MVC 和 ASP.NET

ASP.NET MVC 所依赖的很多核心策略都与其他 MVC 平台所使用的策略完全相同, 而且它还提供了代码编译和管理上的优势, 并挖掘了 .NET Framework 最新版本中新语言的功能。每一个在 Web 上使用的 MVC 架构通常都共享了如下一些基本原则:

- 约定胜于配置
- 不要重复自己(即 DRY 原则)
- 尽可能地实现可插入
- 尽量提供帮助, 但如有必要, 不要妨碍开发人员

34.3.1 为方法而不是文件服务

Web 服务器最初是为存储在磁盘上的静态文件中的 HTML 服务的。随着动态 Web 页面日益流行, Web 开始为从动态脚本中生成的 HTML 服务, 这些脚本也位于磁盘上。使用 MVC 服务 HTML 则有所不同。URL 告知路由机制应该实例化哪个控制器、调用哪个操作方法, 并且向方法提供必需的参数。然后, 控制器的方法决定使用哪个视图, 接着视图便进行显示。

除了与 Web 服务器硬盘上的文件有直接的关系之外, URL 还与控制器对象的方法存在某种关系。ASP.NET MVC 实现的是 MVC 模式的“前端控制器”变体, 而且控制器位于除路由子系统之外的所有对象的前面。

理解 MVC 在 Web 环境中的工作方式的较好方法是: 知道 MVC 是为方法调用的结果服务的, 而不是为动态生成的(也称为脚本化的)页面服务的。事实上, 曾有人将 ASP.NET MVC 称作“用于 Web 的 RPC”, 这一说法很贴切, 尽管在范围上有点狭隘。

34.3.2 ASP.NET MVC 是 Web Forms 4.5?

通过与他人交流, 我们发现, 关于 ASP.NET MVC 的一个重要问题是, 有些人认为它的发布意味着 Web 窗体的终结。情况并非如此。ASP.NET MVC 并不是 ASP.NET Web Forms 4.5。它是使用 Web 窗体之外的另一种选择, 也是 ASP.NET 架构完全支持的部分。在 Web 窗体继续进行改革和发展的同时, ASP.NET MVC 作为微软完全支持的另一种选择也将继续发展。

要理解这一概念, 一种有趣的方法是查看这些技术所处的名称空间。如果找到某个名称空间是 ASP.NET 所在的地方, 那么这个地方就是 System.Web 名称空间。ASP.NET MVC 位于 System.Web.Mvc 名称空间中, 不是 System.Mvc, 也不是 System.Web2。

ASP.NET MVC 现在已经包含到 .NET Framework 4 及以后版本中, 并内置于 Visual Studio 2012 中。ASP.NET MVC 牢固奠定了它在 ASP.NET 中的基础地位。

34.3.3 为什么不是 Web 窗体

在 ASP.NET Web 窗体中，创建 `System.Web.UI.Page` 的一个实例并将服务器控件放置在其中(例如，一个日历和一些按钮)，以便用户输入或查看信息。然后把这些控件连接到 `System.Web.UI.Page` 的事件中以允许进行交互。接着，编译这个页面，并且由 ASP.NET 运行时调用它，创建服务器端的控件树，树中的每个控件都将经历整个事件的生命周期，显示自己，并将结果作为 HTML 保存。作为结果，新的 Web 艺术开始显现——Web 窗体层在 HTTP 顶端进行事件处理和状态管理——真正无状态的协议。

为什么需要这种抽象呢？请记住，Web 窗体是引入给非常熟悉 Visual Basic 6 的微软开发人员的。开发人员使用 VB 6 将按钮拖到设计界面上并双击按钮时，会立即创建 `Button_Click` 事件处理程序。这是创建业务应用程序的一种非常强大的方式，让所有人因为快速应用程序开发工具(Rapid Application Development, RAD)的产生而兴奋。而当开发人员开始使用传统 ASP 时，实际上是使他从使用的 Visual Basic 丰富环境中倒退了一步。不论好坏，Web 窗体将 RAD 体验带给了 Web 开发人员。

但是，随着 Web 的不断成熟以及 CSS(Cascading Style Sheets, 层叠样式表)和 XHTML 的引入，越来越多的人对 HTML 有了自己的理解，新的 Web 艺术开始显现。Web 窗体仍然保持很高的开发效率，可以快速创建基于 Web 的业务程序。HTML 虽然看起来生成良好，但是却足以触动制作 XHTML 和 CSS 站点的人的敏感神经。HTML5 尤其如此，人们重新关注语义更丰富的 HTML。虽然诸如 ViewState 和 Postback 事件模型的 Web 窗体概念有其存在的理由，但是许多开发人员都希望拥有较低级的替代品，不仅可以提供 HTML，而且可以提供 HTTP 自身。

此外，Web 窗体的体系结构让使用当前的单元测试工具进行测试变得困难，这些工具包括 NUnit、MbUnit 和 xUnit.NET。ASP.NET Web 窗体在设计时没有考虑单元测试，因此即使在 Web 上出现许多黑客时，平心而论，Web 窗体也不能够很好地进行测试驱动开发。ASP.NET MVC 提供了对 HTML 的绝对控制，但并不否认 HTTP 的存在，而且从一开始就是朝着可测试性这一目标而设计的。

34.3.4 ASP.NET MVC 是完全不同的

ASP.NET MVC 是完全不同的，这是整个关键所在。ASP.NET MVC 构建于系统的价值观和体系结构规则之上，这一点不同于 Web 窗体。ASP.NET MVC 注重可扩展性、可测试性以及灵活性。它使用起来很轻便，不需要对如何使用它做很多假设(除了假设你欣赏 MVC 模式之外)。

34.3.5 为什么“(ASP.NET>ASP.NET MVC)==True”

创建 MVC 应用程序是很简单的。可以使用 Visual Studio 2012 的任何版本来创建基本的应用程序，依照如下步骤：

- (1) 打开 Visual Studio 2012，选择 File | New Project 命令。
- (2) 在 New Project 对话框(如图 34-1 所示)中选择 ASP.NET MVC 4 Web Application 选项。



图 34-1

(3) 输入项目名并设置项目在磁盘上的位置，单击 OK 按钮。此时打开 New ASP.NET MVC 4 Project 对话框，如图 34-2 所示。该对话框可以选择用于创建项目的特定 MVC 模板。选择 Internet Application 项目模板。

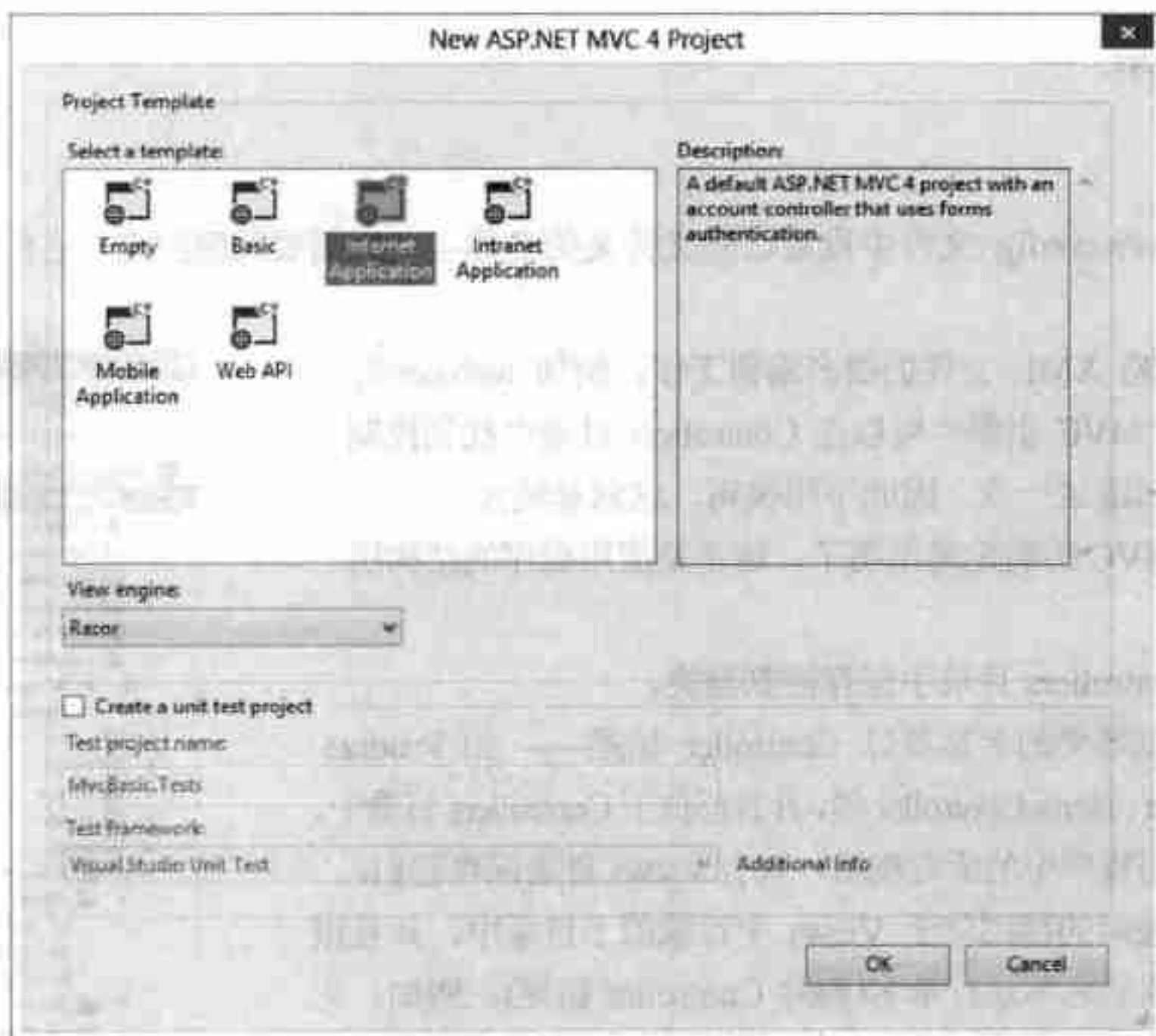


图 34-2

(4) New ASP.NET MVC 4 Project 对话框还包含创建 Unit Test 项目的设置。选中 Create a unit test project 复选框，创建包含基本 ASP.NET MVC 项目和附加 Unit Test 项目的解决方案。默认情况下，

Test Framework 下拉列表中包括 Visual Studio Unit Test 选项。如果安装有第三方单元测试架构(如 MbUnit 或 NUnit), 该对话框中就会显示其他的选项。

(5) 单击 OK 按钮, 解决方案中的项目如图 34-3 所示。注意, 尽管这是 ASP.NET 应用程序, 也含有标准的类库, 但是其中仍然包含一些前面没有看到的文件夹。

事实上, 在应用程序中有很多可能使用过的目录, 这是通过设计得到的目录。ASP.NET MVC 与其他 MVC 架构一样, 在很大程度上依赖于这样的思想, 即依靠应用程序中一些基本的结构规则来减少工作量和代码。Ruby on Rails 非常简洁地描述了这个强大的思想: 约定胜于配置。

34.3.6 约定胜于配置

Ruby on Rails 使“约定胜于配置”的思想在几年前就开始流行起来, 其本质是指: 至此, 我们已经知道了如何创建 Web 应用程序, 把该经验应用到架构中, 这样就不需要再配置所有的内容。

查看使应用程序运行的如下 3 个核心目录, 我们可以看到这个思想在 ASP.NET MVC 中的应用情况:

- Controllers
- Models
- Views

不需要在 web.config 文件中设置这些文件夹的名称——只需要按照约定将它们放在期望的位置即可。

这样可以省略 XML 文件的很多编辑工作, 例如 web.config 文件显式地告知 MVC 引擎“可以在 Controllers 目录中找到控制器”。由于已经知道这一点, 因此不用编辑, 这就是约定。

ASP.NET MVC 的约定简单明了, 这正是应用程序的结构所期望的情况。

- 只在 Controllers 目录中保存控制器类。
- 每个控制器类的名称都以 Controller 结尾——如 ProductController、HomeController 等, 并且都位于 Controllers 目录中。
- 对于应用程序中的所有视图, 只有 Views 目录保存它们。
- 控制器使用的视图位于 Views 主目录的子目录中, 并且根据控制器的名称进行命名(删除 Controller 结尾)。例如, 先前讨论的 ProductController 的视图将位于/Views/Product 中。
- 所有可重用的 UI 元素都位于 Views 目录的 Shared 子目录中。

如果想更进一步查看, 可以展开样本应用程序最初的结构, 如图 34-4 所示, 可以看到这些约定正在起作用。



图 34-3



图 34-4

Controllers 目录中有两个控制器: HomeController 和 AccountController。Views 目录中有许多视图。下面的讨论将集中于/Views/Home 目录中的 About 和 Index 视图。

虽然在命名视图时没有指定约定,但是可以按照 ASP.NET MVC 约定给视图赋予与操作一样的名称。使用这个约定还可以让其他的开发人员轻松地回顾和理解应用程序。

在使用模板创建 Index 和 About 视图时,可以看到这个约定的运行情况。这些也是调用的控制器操作的名称,并且显示这些视图的 C#代码很简单:

```
return View();
```

这看起来有一点混乱。在对应用程序稍做修改并进一步分析后,可以看到一个清晰的示例:

(1) 打开 HomeController.cs, 复制并粘贴 About 方法, 创建名为 Foo 的副本, 如下所示:

```
public ActionResult Foo()
{
    ViewBag.Message = "Foo page.";

    return View();
}
```

(2) 在添加一些内容之后, 开始调试该应用程序。ASP.NET Development Server 将自动选择高端口号并启动浏览器。浏览器最后导航到一个地址, 例如 http://localhost:3098, 如图 34-5 所示。

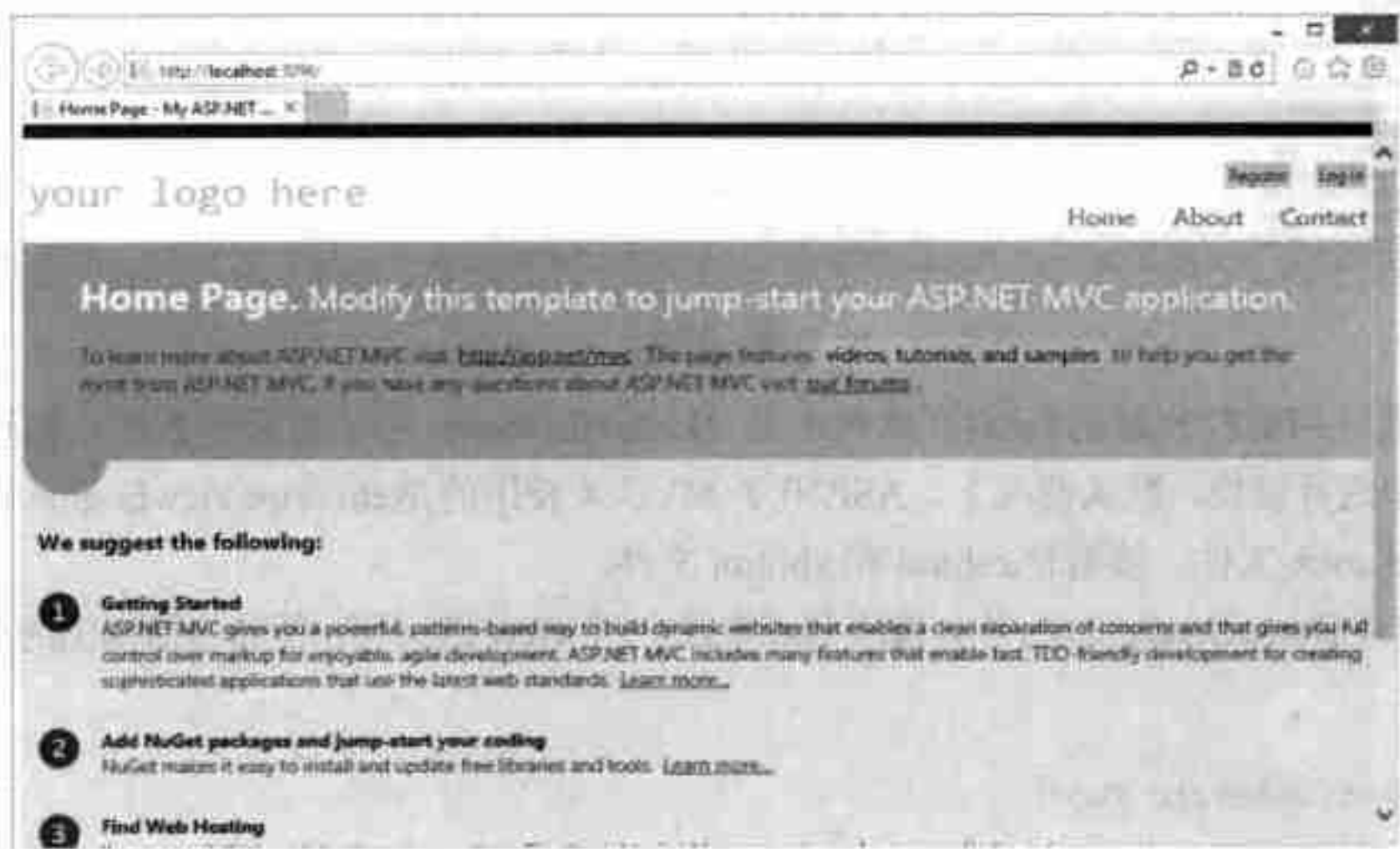


图 34-5

(3) 注意为什么没有.aspx 扩展名? ASP.NET MVC 允许完全控制 URL。现在, 在浏览器的地址栏中把相对 URL 从 “/” 修改为 “/Home/Foo”。事情开始变得有趣, 如图 34-6 所示。请记住, 正在使用 Foo 方法给 View 返回调用的结果。当位于 HomeController 的 Foo 方法中时, 系统将在多个位置查找名为 Foo 的视图。ASP.NET MVC 足够智能, 能够提示错误消息, 这些消息可以提供很多有用信息。这确实很令人激动。

System.InvalidOperationException: The view 'Foo' or its master was not found or no view engine supports the searched locations. The following locations were searched:

```
~/Views/Home/Foo.aspx
```



```

~/Views/Home/Foo.ascx
~/Views/Shared/Foo.aspx
~/Views/Shared/Foo.ascx
~/Views/Home/Foo.cshtml
~/Views/Home/Foo.vbhtml
~/Views/Shared/Foo.cshtml
~/Views/Shared/Foo.vbhtml

```

错误消息按照查询的顺序列出了系统查找视图的位置(如图 34-6 所示),从而提供足够的信息来推断出视图的命名约定。

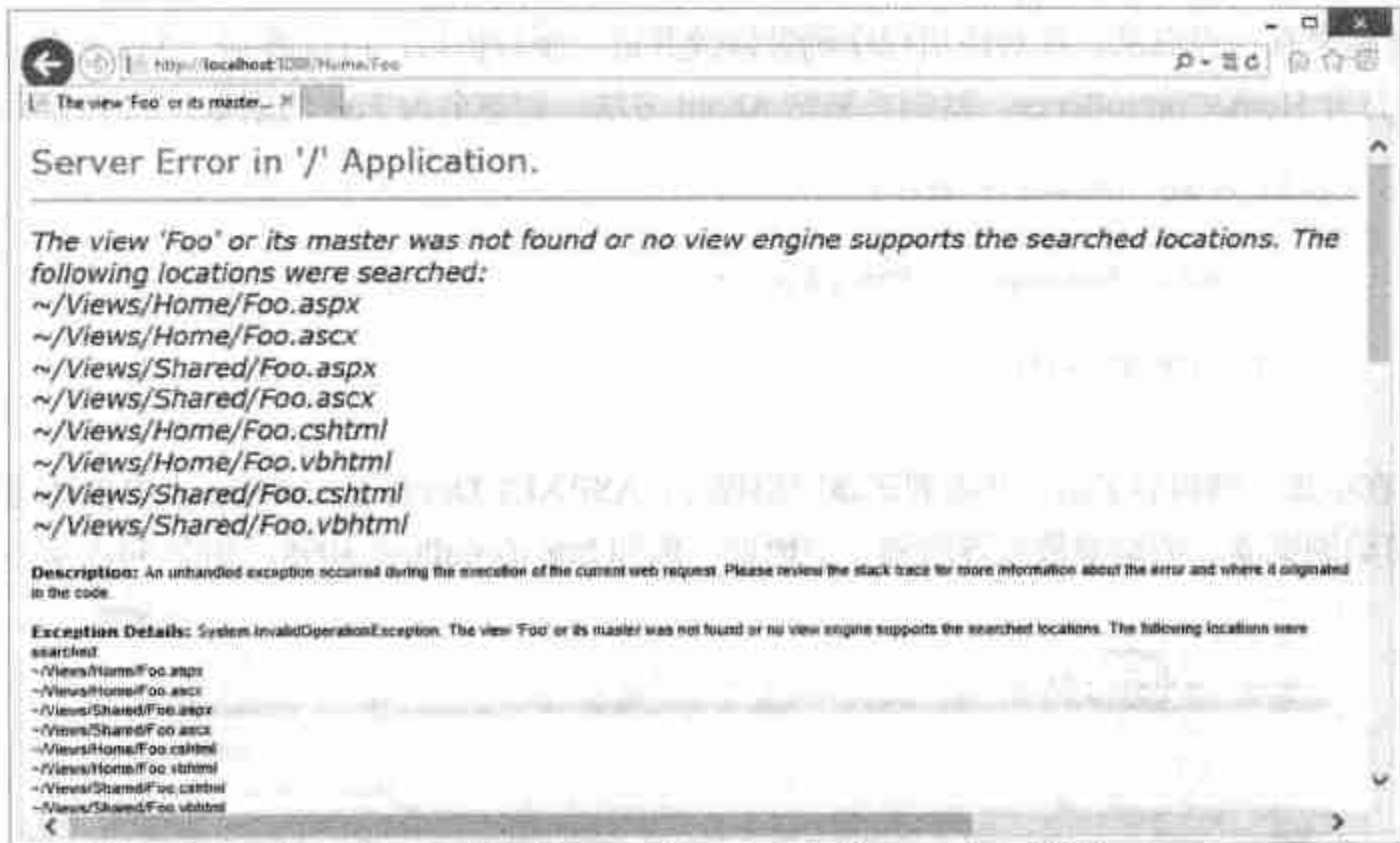


图 34-6

首先,通过当前控制器的名称(在本例中是 Home)在/Views 的子目录中查找,然后在/Views/Shared 目录中进行查找。默认情况下,ASP.NET MVC 4 使用的 WebFormsViewEngine 将查找.aspx 页面,然后是.ascx 文件,接着是.cshtml 和.vbhtml 文件。

(4) 返回到 HomeController.cs 中,修改 Foo 方法中对 View 的调用,使其包括视图的名称以作为参数:

```

public ActionResult Foo()
{
    ViewBag.Message = "Foo page.";

    return View("Index");
}

```

(5) 重新启动应用程序,并再次访问/Home/Foo 目录。此时将显示 Index 视图,浏览器的标题中将显示 Title 字符串。

(6) 切换到 Visual Studio,在返回视图结果的行中设置断点。刷新浏览器,确定仍然在/Home/Foo 目录中,并且准备好做进一步研究。

34.3.7 第三个请求是 Charm

花点时间思考这里发生了什么事情。应用程序中的事件状态是什么？Visual Studio 如图 34-7 所示。

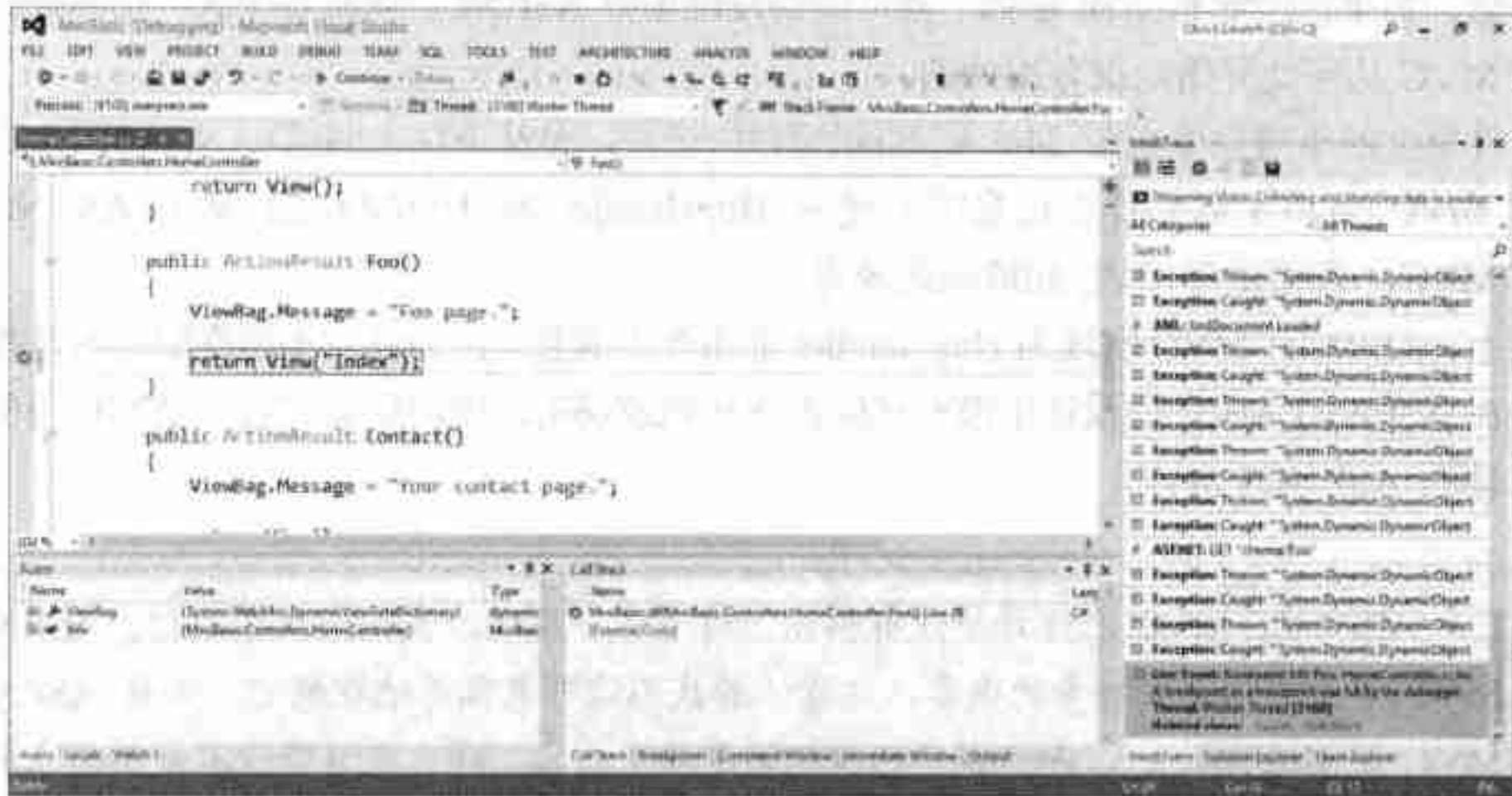


图 34-7

花点时间查看 Visual Studio，并试着确定 Visual Studio 告诉你什么事情。如何到了这个地方？现在位于什么地方？

在浏览器中访问/Home/Foo，你将神奇般地处于 Foo 操作方法的断点位置。Call Stack 工具窗口证明了这一点，但是说明得还不够充分。我们是如何到达这里的？右击调用栈的空白区域，并选择 Show External Code 命令。也可以拖动 Call Stack 工具窗口，使其脱离 Visual Studio 以更好地分析将要展现的信息。如果有多台显示器，那么记住 Visual Studio 2012 支持多台显示器，因此如果喜欢的话，可以使用 Call Stack 工具窗口填充完整的显示屏幕。

Call Stack 工具窗口包括很多信息，事实上，作者在图 34-8 中突出显示了信息的重要部分。请记住，调用栈是从下向上阅读的，底部是开始的地方，而顶部是当前正在调试的行。在这个调用栈中，有些部分比其他部分重要得多。



图 34-8

从底部开始,可以看到由 ASP.NET(特别是 System.Web.HttpRuntime)处理的请求。这是 ASP.NET 的“开始”。注意,这是 System.Web,并且我们目前位于 System.Web.dll 内——没有发生任何与 MVC 有关的事情。如果已经熟悉 Web 窗体,就可能会发现牢记 ASP.NET 到底是什么、在何处结束以及 ASP.NET MVC 从何处开始是很有用处的。

第一件重要的事情发生在图 34-8 底部的标注中(记住,是从下往上阅读)。从 ASP.NET 切换到 ASP.NET MVC 可以了解到什么信息呢?使用 `HttpHandler` 和 `HttpModule` 基于 ASP.NET 构建 ASP.NET MVC,这就是与 MVC 相联系的地方。

事实上,ASP.NET MVC 实现为 `HttpHandler` 非常令人满意,因为开发小组在编写时是按照规则操作的。在 ASP.NET MVC 的设计中没有任何内部知识或秘密。使用所有开发人员都可用的公有构造和 API 编写 ASP.NET MVC。



ASP.NET MVC 在设计中没有秘密,对于这一发现,我们感到很欣慰。ASP.NET MVC 中没有那么多神奇的内容,这意味着我们能够更容易地理解它。如果 ASP.NET MVC 是 `HttpHandler`,而且我们已经编写了许多内容,那么它可能没有我们想象的那么神奇。此外,我们还可以很高兴地看到,ASP.NET 是很具灵活性和可扩展性的,从而允许创建类似 ASP.NET MVC 之类的程序。

从这些发现中还可以看出,因为 ASP.NET MVC 使用 `HttpHandler`(和 `HttpModule`)来完成工作,所以 MVC 构建在 ASP.NET 之上。对于某些人来说,这可能是一种很明显的表述,但其实却是很常见的问题:“ASP.NET MVC 是全新的 ASP.NET 吗?”从图 34-8 中可以看出,不是。ASP.NET MVC 是在相同的基础结构之上构建的,即我们已经使用多年的“核心”ASP.NET。

回头查看图 34-8 中的调用栈。请记住我们目前位于 `HomeController` 内的 `Foo` 方法的断点处。是谁创建了 `HomeController`? 有人新建了它。谁调用了 `Foo`?

在 `MvcHandler` 的 `ProcessRequest` 方法中, `DefaultControllerFactory` 创建了控制器的一个实例。`DefaultControllerFactory` 是 ASP.NET MVC 创建并使用的内置工厂,用于搜索和创建控制器。ASP.NET MVC 创建了 `HomeController` 的一个实例,然后调用控制器的 `Execute` 方法。这个方法又依赖于控制器的操作调用器(默认情况下是 `ControllerActionInvoker`)来实际调用这个方法。

请记住,我们打开了浏览器并请求 `/Home/Foo`。ASP.NET MVC 应用程序将请求发送给 `MvcHandler`。该处理程序创建 `HomeController` 的一个实例并调用 `Foo` 方法。ASP.NET MVC 同时处理对象激活和方法调用。

`UrlRoutingModule` 截获了 `/Home/Foo` 的 URL 地址。该模块负责分析 URL 并创建一些路由数据,确保正确的 URL 进入到正确的控制器中。MVC 管道使用 `ControllerFactory` 和 `ControllerActionInvoker` 来分别创建控制器和调用其方法。

控制器的存在是为了“做事”。至于要做什么事情,则取决于开发人员。控制器与模型进行交流并计算。但是,控制器不显示 HTML,也不和数据库交流。这就是关注点分离。控制器关心的是控制。

控制器将 `ViewData` 传送给视图,视图关心的是显示 HTML(或任何其他想要显示的内容)。HTML 包括到其他 URL 的链接,这种循环将持续下去。

34.4 理解路由和 URL

软件开发人员经常会对一些小的细节问题倍加关注，特别是当涉及源代码的质量和结构时更是如此。他们常常为了代码的缩排样式以及花括号的位置而进行长期思想斗争。因此，当碰到大多数使用 ASP.NET 创建的站点并遇到如下所示的 URL 地址时，你可能会感到有些奇怪：

```
http://example.com/products/list.aspx?id=17313&catid=33723&page=3
```

既然在代码上花费了很多心思，为什么不给予 URL 同样多的关注呢？虽然看上去并不是那么重要，但 URL 是一种合法的、广为使用的 Web 用户接口。

可用性专家 Jakob Nielsen(www.useit.com)力劝开发人员关注 URL，并为建立高质量的 URL 提供了如下原则：

- 易于记住和拼写的域名
- 简短的 URL
- 易于输入的 URL
- 反映站点结构的 URL
- 可编程的 URL，从而允许用户通过修改 URL 的末尾来进入信息体系结构的更高层次
- 持久不变的 URL

从传统意义上来说，在很多 Web 架构中(如传统 ASP、JSP、PHP、ASP.NET 等)，URL 代表了磁盘上的物理文件。例如，当看到如下请求时：

```
http://example.com/products/list.aspx
```

可以肯定该 Web 站点有目录结构，目录中包括 products 文件夹，并且该文件夹中有 list.aspx 文件。在本例中，URL 和磁盘上物理存在的内容之间有直接的关系。当 Web 服务器接收到这一请求时，Web 架构执行与该文件相关联的代码，以响应请求。在很多情况下，代码包含了某个模板，或者与这个模板相关联，该模板混合了服务器端声明和 HTML 标记，进而生成通过响应回送给浏览器的结果标记。

ASP.NET MVC 架构中的路由选择主要有两个用途：

- 与入站请求相匹配，并把请求映射到控制器的操作中。
- 构建出站 URL 以响应控制器的操作。

了解了 URL 和路由选择的一些知识后，就可以开始进一步了解路由以及路由与 URL 重写的区别。

34.4.1 路由选择与 URL 重写的比较

为了更好地理解路由选择，许多开发人员都喜欢将其与 URL 重写进行比较。毕竟，这两种方法在创建 URL 与处理 URL 的代码之间的分离时都很有用，这些代码可以帮助创建用于搜索引擎优化(SEO)的“漂亮的”URL。但是，其中的关键区别在于，URL 重写代表了“以页面为中心”的 URL 视图。ASP.NET 的大多数重写方案都可以为一个页面重写 URL，以便另一个页面使用。例如，如下所示的 URL 地址：

```
/product/bolts.aspx
```

可以重写为:

```
/product/display.aspx?productid=111
```

而另一方面,路由选择采用了“以资源为中心”的 URL 视图。在这种情况下,URL 代表 Web 上的资源(不一定是页面)。通过 ASP.NET 路由,资源成了代码片段,并在入站请求匹配路由时执行。路由决定了如何基于 URL 的特征发送请求——不会重写 URL。

路由选择和重写的另一重要区别在于,通过使用与匹配入站 URL 相同的映射规则,路由选择还可以帮助生成 URL。另一种看法是 ASP.NET 路由选择更多地类似于双向(bidirectional)的 URL 重写。简而言之,这种比较在于 ASP.NET 路由选择实际上从来没有重写 URL。用户在浏览器中发出的 URL 请求与应用程序在整个请求生命周期内看到的 URL 是一样的。

34.4.2 路由的定义

每个 ASP.NET MVC 应用程序都至少需要一个路由,以定义该应用程序如何处理请求,但是通常最终都只会得到少量的处理方法。可以想象,一个非常复杂的应用程序将会包含非常多的路由。

本节讨论如何定义路由。路由的定义是从 URL 开始的,URL 指定路由将要匹配的模式。与路由的 URL 一起,路由还可以指定默认的值并约束 URL 的各个部分,并且提供了关于路由如何匹配入站 URL 请求的严密控制。

下面从非常简单的路由开始介绍,并以此开始创建路由。

1. 设置路由的 URL

在创建新的 ASP.NET MVC Web 应用程序项目后,快速查看 Global.asax.cs 文件的代码。注意,Application_Start 方法包含对 RegisterConfig 类中 RegisterRoutes 方法的调用。该方法是注册应用程序的所有路由的地方。



不是在 Global.asax 中直接调用配置和设置方法,它们都放在一些类的静态方法中,这些类放在 ASP.NET 文件夹 App_Start 中,这只是为了使 Global.asax 更简洁,但强制了关注点的分离。

在 RegisterConfig 类的 RegisterRoutes 方法中清除路由,并使用下面这个非常简单的 C# 路由替换它们:

```
routes.MapRoute("simple", "{first}/{second}/{third}");
```

MapRoute 方法的最简单形式是采取路由的名称及其 URL 模式。本节稍后会讨论名称,目前我们主要关注 URL 模式。

注意,路由的 URL 由一些 URL 片段(片段是指斜杠之间的所有内容,但是不包括斜杠)组成,每一个 URL 片段都包括一个使用花括号来划分的占位符。这些占位符称作 URL 参数。这是一种模式匹配规则,用于决定路由是否应用到入站请求中。在本例中,因为默认情况下 URL 参数可以匹

配所有的非空值，所以该规则将通过 3 个片段与任何 URL 相匹配。当该规则与使用了 3 个片段的 URL 相匹配时，该 URL 的第一个片段中的文本就与 URL 参数 {first} 相对应，第二个片段中的值则与 URL 参数 {second} 相对应，第三个片段的值与 {third} 参数相对应。

如本例所示，可以将这些参数命名为想要的任何名称。当请求进来时，路由将请求的 URL 分析成字典(即通过 RequestContext 中的 RouteData 访问的 RouteValueDictionary)，使用 URL 参数名称作为键，并使用对应位置的 URL 分部作为值。当在 ASP.NET MVC 应用程序的上下文中使用路由时，存在一些附带特殊用途的特定参数名称。表 34-1 给出了刚刚定义的路由如何将特定的 URL 转换成 RouteValueDictionary。

表 34-1

URL	URL 参 数 值
/products/display/123	{first}=products {second}=display {third}=123
/foo/bar/baz	{first}=foo {second}=bar {third}=baz
/a.b/c-d/e-f	{first}="a.b" {second}="c-d" {third}="e-f"

如果实际地请求了表 34-1 中列出的 URL，就会注意到 ASP.NET MVC 应用程序会出现中断。虽然可以使用任何想要的参数名称来定义路由，但是为了使路由能正确运转，ASP.NET MVC 会要求一些特定的参数名称——{controller} 和 {action}。

{controller} 参数的值用于实例化处理请求的控制器类。按照约定，MVC 将后缀“Controller”添加到 {controller} 值的后面，并试图定位该名称的类型(区分大小写)，该类继承了 System.Web.Mvc.IController 接口。

返回到之前那个简单路由示例，将其从：

```
routes.MapRoute("simple", "{first}/{second}/{third}")
```

修改为：

```
routes.MapRoute("simple", "{controller}/{action}/{id}")
```

这样便包含了特殊的 URL 参数名称。

再次查看表 34-1 中的第一个示例，可以看到对 /products/display/123 的请求是在请求名为“products”的 {controller}。ASP.NET MVC 采用了该值，并在后面添加“Controller”作为后缀，从而得到类型名称 ProductController。如果存在实现 IController 接口的名称类型，那么将实例化该类型并将其用于处理请求。

{action} 参数的值用于指明调用控制器的哪个方法来处理当前请求。注意，这个方法调用只应用于继承于 System.Web.Mvc.Controller 基类的控制器类。继续对 /products/display/123 的请求示例，MVC

调用的 `ProductsController` 的方法是 `Display`。

注意，表 34-1 中的第 3 个 URL 虽然是有效的路由 URL，但却很可能与任何真正的控制器和操作不匹配，因为试图实例化名为 `a.bController` 的控制器并调用名为 `c-d` 的方法，而这个名称不是有效的方法名称。

除 `{controller}` 和 `{action}` 外的任何路由参数都是作为参数传送到操作方法中的(如果这些参数存在的话)。程序清单 34-1 是带有操作的产品控制器(本章下载代码中的 `Controllers/ProductsController.cs`)。

程序清单 34-1 带有操作的产品控制器

```
using System.Web.Mvc;

public class ProductsController : Controller
{
    public ActionResult Display(int id)
    {
        //Do something
        return View();
    }
}
```

对 `/products/display/123` 的请求将导致 MVC 实例化该类并调用 `Display` 方法，将 123 作为 `id` 传递进来。

对于前面使用了路由 URL `{controller}/{action}/{id}` 的示例，每个片段都包含一个占据整个片段的 URL 参数。但是，情况并不总是如此。路由 URL 确实允许在片段中使用字面值。例如，如果把 MVC 集成到现有的站点中，并希望所有的 MVC 请求都以单词 `site` 开头，就可以按如下所示进行操作：

```
site/{controller}/{action}/{id}
```

这指明要匹配该请求，URL 的第一个片段必须以 `site` 开头。因此，`/site/products/display/123` 与该路由匹配，但是 `products/display/123` 不匹配。

可能存在字面值和参数混合在一起的 URL 片段。唯一的限制在于，不允许有两个连续的 URL 参数。因此：

```
{language}-{country}/{controller}/{action}
{controller}.{action}.{id}
```

是有效的路由 URL，但：

```
{controller}{action}/{id}
```

也不是有效的路由。路由没有办法知道入站请求 URL 的控制器部分何时结束，操作部分何时会开始。

请查看一些其他的例子(如表 34-2 所示)，从中可以了解 URL 模式如何响应匹配的 URL。

表 34-2

路由 URL 模式	匹配的 URL 示例
{controller}/{action}/{category}	/products/list/beverages/blog/posts/123
Service/{action}-{format}	/service/display-xml
{reporttype}/{year}/{month}/{date}	/sales/2008/1/23

2. 深入分析：路由如何将 URL 与操作联系起来

刚才讨论了如何将路由映射到 MVC 架构中的控制器操作。下面将深入分析这是如何发生的，从而更好地了解路由选择和 MVC 之间的分界线。

常见的误解是路由选择只是 ASP.NET MVC 的功能之一。在 ASP.NET MVC 实现的前期阶段，情况的确如此，但是经过一段时间之后，情况就变得明朗起来，即路由选择是更为常见的有用功能。ASP.NET Dynamic Data 小组特别感兴趣的是如何把路由选择用于 Dynamic Data 中。此外，路由选择成了一项更具有通用用途的功能，既没有包含 MVC 的内部知识，也不依赖 MVC。

路由选择是单独存在的另一外在证明不仅在于是单独的程序集，而且位于 System.Web.Routing 名称空间中，而不是位于理论上的 System.Web.Mvc.Routing 名称空间中。可以从名称空间中获取很多信息。



这里的讨论集中在 IIS 7 集成模式和 IIS 8 的路由选择上。当通过 IIS 7 传统模式或 IIS 6 使用路由选择时，会有一些细微的差别。在使用 Visual Studio 2012 内置的 Web 服务器时，其行为非常类似于 IIS 8 的集成模式。

3. 高层次请求的路由选择管道

关于路由选择进行的所有讨论都可能是要处理的许多信息。但是，理解这些信息很重要，因为路由选择的确是控制应用程序的 URL 的最强大工具。

把路由选择分解为组件，可以发现路由选择管道由如下 5 个高层次的步骤组成：

- (1) UrlRoutingModule 试图通过在 RouteTable 中注册的路由来与当前的请求相匹配。
- (2) 如果路由匹配，那么路由选择的模块将获取来自路由的 IRouteHandler。
- (3) 路由选择的模块将调用来自 IRouteHandler 的 GetHandler，将返回一个 IHttpHandler 对象。请记住，典型的 ASP.NET Page(又称作 System.Web.UI.Page)类仅仅是 IHttpHandler。

(4) 在 IHttpHandler 上调用 ProcessRequest，这将传递即将处理的请求。

(5) 对于 MVC 而言，在默认情况下，IRouteHandler 是 MvcRouteHandler 的一个实例，它将返回一个 MvcHandler 对象(实现了 IHttpHandler)。MvcHandler 负责实例化正确的控制器并调用该控制器的操作方法。

4. 路由匹配

路由选择的核心仅仅是匹配请求，从请求中提取路由数据并将其传递给 `IrouteHandler` 对象。从高层次的角度来看，路由匹配算法很简单。

当请求进来时，`UrlRoutingModule` 按顺序遍历通过 `RouteTable.Routes` 访问的 `RouteCollection` 中的每个路由，然后询问每个路由：“你可以处理这个请求吗？”如果路由回答：“是的，我可以！”就完成路由查找，路由将开始处理请求。

通过调用 `GetRouteData` 方法，可以询问路由是否可以处理请求。如果当前的请求与路由不匹配（换句话说，模块和路由之间没有真正的对话），那么方法将返回空值。

5. 路由数据

记住，当调用 `GetRouteData` 方法时，它返回 `RouteData` 的一个实例。到底什么是 `RouteData`？`RouteData` 包含了关于与特定请求匹配的路由的信息，包括匹配请求的具体上下文信息。

在前面的章节中，我们讨论了具有如下 URL 的路由：`{foo}/{bar}/{baz}`。当对 `/products/display/123` 的请求进来时，路由试图与请求相匹配。如果确实匹配，就创建一个字典，该字典包含从 URL 分析的信息。特别是，它将在字典中为路由 URL 中的每个 url 参数添加一个键。

因此，在示例 `{foo}/{bar}/{baz}` 中，可以预计字典至少包含 3 个键，即“foo”、“bar”和“baz”。对于 `/products/display/123` 的情形，URL 用来提供这些字典键的值。在本例中，`foo=products`、`bar=display`，`baz=123`。

6. 参数的默认值和可选参数

有时希望给参数指定默认值。在注册路由时，可以给参数指定默认值。甚至可以给特定的参数 `{controller}` 和 `{action}` 指定默认值。为了给路由设置默认值，应使用 `MapRoute` 方法的重载版本，指定 `Route` 类的 `Defaults` 属性。创建新 MVC 应用程序时的默认路由，给控制器和操作指定了默认值。



如果在本章前面修改了默认的路由，就应把它改回其他示例使用的这个默认路由值。

```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index",
        id = UrlParameter.Optional }
);
```

还要注意，在默认映射中给 `id` 参数使用了 `UrlParameter.Optional`。在某些情况下，需要指定参数没有默认值，且是可选参数。为此，可以使用特定值 `UrlParameter.Optional` 给参数指定默认值。如果希望在操作中访问这个参数，就把该参数声明为 `Nullable` 类型。程序清单 34-2 修改了程序清单 34-1，接收一个可选参数（本章下载代码中的 `Controllers/ProductsController.cs`）。

程序清单 34-2 带有可选参数的产品控制器操作

```
using System.Web.Mvc;

public class ProductsController : Controller
{
    public ActionResult Display(int? id)
    {
        //Do something
        return View();
    }
}
```

34.5 控制器

你可能希望记住如下快速定义：MVC 模式中的控制器负责响应用户的输入，通常需要修改模型以响应用户的输入。

这样，MVC 模式中的控制器关注的就是应用程序中的程序流，处理引入的数据以及提供输出到相关视图的数据。

34.5.1 控制器的定义：IController 接口

可扩展性和灵活性是 ASP.NET MVC 关注的核心问题。在以这些方面为目标构建软件时，尽可能使用接口来利用抽象化是很重要的。

要想成为 ASP.NET MVC 中的控制器，类就必须尽可能少地实现 IController 接口，而且按照约定，这种类的名称必须以后缀 Controller 结尾。命名约定实际上是很重要的，在 ASP.NET MVC 中可以发现很多这种约定，这样就不需要定义配置的设置和特性，从而使编程更简单。就抽象化的能力来说，IController 接口是很简单的：

```
public interface IController
{
    void Execute(RequestContext requestContext);
}
```

创建 Icontroller 对象是很简单的过程：当请求进来时，路由选择系统标识出控制器，负责调用 Execute 方法。

ControllerBase 是抽象基类，基于 IController 接口添加了更多的 API 接口。该类提供了 TempData、ViewData 和 ViewBag 属性，其 Execute 方法负责创建 ControllerContext。ControllerContext 提供了当前请求的 MVC 特有的上下文，其工作方式与 HttpContext 实例通常为 ASP.NET 提供上下文的方式相同(提供请求和响应、URL 以及服务器信息等元素)。

这个基类仍然是轻量级的，允许开发人员向自己的控制器提供非常个性化的实现，同时还受益于 ASP.NET MVC 中操作过滤器的基础结构。该基类没有提供将操作转换成方法调用的功能。因此，控制器类应运而生。

34.5.2 控制器类和操作

在理论上,可以通过仅实现 `ControllerBase` 或 `Controller` 的类来创建整个站点,而且这个站点可以正常运转。路由选择将按照名称来查找 `Controller`,然后调用 `Execute` 方法,这样就会得到极为基本的 Web 站点。

但是,这种方法与使用原始的 `HttpHandler` 来处理 ASP.NET 非常相似,虽然可以正常运作,但是剩下的工作是做不必要的重复劳动和了解核心的架构逻辑。有趣的是,ASP.NET MVC 位于 HTTP 处理程序的顶部,而且没有必要对 ASP.NET 做内部信息的修改以实现 MVC。相反,ASP.NET MVC 小组只是在已有的 ASP.NET 可扩展点之上放置这层新的架构。编写控制器的标准方法是让控制器继承 `System.Web.Mvc.Controller` 抽象基类,这个抽象基类派生自 `ControllerBase` 基类。控制器类用作所有控制器的基类,因为它向派生自基类的控制器提供了很多不错的行为。

接下来介绍另一个简单的控制器示例,但是这次将添加一个公有方法。打开先前使用的项目,右击 `Controllers` 文件夹并选择 `Add | Controller` 命令以创建一个新的控制器,然后将其命名为 `Simple2Controller`。程序清单 34-3 显示了添加必要代码后的 `Simple2Controller`(本章下载代码中的 `Controllers/Simple2Controller.cs`):

程序清单 34-3 `Simple2Controller` 类

```
using System.Web.Mvc;

public class Simple2Controller : Controller
{
    public void Hello()
    {
        Response.Write("<h1>Hello World Again!</h1>");
    }
}
```

按下 `Ctrl+F5` 键(或选择 `Debug | Run` 命令),在浏览器中导航到 `/Simple2/Hello`。图 34-9 显示了结果。

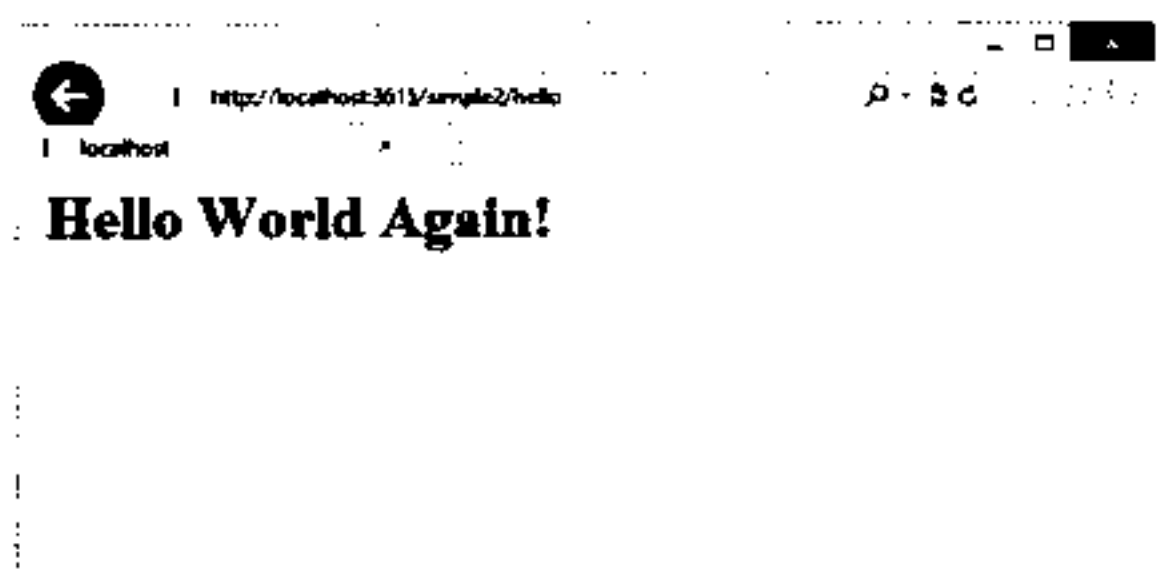


图 34-9

与之前一样,该例没有太多突出的地方,但是却更加有趣。注意,地址栏中的 URL 直接与控制器的操作方法相关联。MVC 的默认路由将 URL 分成了 3 个主要部分: `/ {controller} / {action} / {id}`。

接下来查看路由是如何应用于该例的。URL 的 `Simple2` 部分对应于控制器的名称。MVC 架构将 `Controller` 后缀添加到控制器名称中并定位控制器类 `Simple2Controller`:

```
/Simple2/Hello
```

URL 的最后一部分对应于操作。MVC 架构将通过名称定位公有方法并试图调用该方法。

34.5.3 处理参数

可以将任意数量的公有方法(为了遵循约定, 这里把这些公有方法称为操作)添加到控制器类中, 可以通过这种模式来调用这些方法。操作也包含参数。返回到 `Simple2Controller`, 添加采用一个参数的新操作方法。程序清单 34-4 显示了添加 `Goodbye` 操作后的 `Simple2Controller`(本章下载代码中的 `Controllers/Simple2Controller.cs`):

程序清单 34-4 添加了操作的 Simple2Controller

```
using System.Web;
using System.Web.Mvc;

public class Simple2Controller : Controller
{
    public void Hello()
    {
        Response.Write("<h1>Hello World Again!</h1>");
    }

    public void Goodbye(string name)
    {
        Response.Write("Goodbye " + HttpUtility.HtmlEncode(name));
    }
}
```

通过 URL 调用这个方法:

```
/Simple2/Goodbye?name=World
```

注意, 可以将参数通过查询字符串传递给操作方法。此外, 还可以通过 URL 片段将参数传递进来, 可以通过路由中定义的位置来发现该 URL 片段。例如, 从美学角度来看, 下面的 URL 更能够令开发人员和 Internet 用户高兴:

```
/Simple2/Goodbye/World
```

要使用由 URL 片段传递的参数, 就必须定义路由选择如何标识 URL 中的这些参数。幸运的是, 默认路由(当选择 `File | New` 命令时自动创建)已经自动设置并包含常用的 URL 模式:

```
{controller}/{action}/{id}.
```

稍微修改 `Simple2Controller` 中操作方法的签名(将参数 `name` 重命名为 `id`), 如下所示:

```
public void Goodbye(string id)
{
    Response.Write("Goodbye " + HttpUtility.HtmlEncode(id));
}
```

这就允许使用“更整洁的”URL 地址调用该方法, 而且路由选择将会通过结构化的 URL(而不

是查询字符串参数)来传递参数:

```
/Simple2/Goodbye/World
```

34.5.4 处理多个参数

如果方法带有多个参数,那么应该如何处理?这种情况很常见,我们仍然可以使用查询字符串,但是如果希望通过 URL 片段传递两个参数,就必须定义该情形下特定的新路由。

例如,假设存在如下计算两点间距离的操作方法:

```
public void Distance(int x1, int y1, int x2, int y2)
{
    double xSquared = Math.Pow(x2 - x1, 2);
    double ySquared = Math.Pow(y2 - y1, 2);
    Response.Write(Math.Sqrt(xSquared + ySquared));
}
```

使用唯一的默认路由,请求如下:

```
/Simple2/Distance?x2=1&y2=2&x1=0&y1=0
```

定义允许以更简单的格式来指定参数的路由,以稍微改进上述情况。本章前面提到,如果用一个常量分隔多个参数,就可以在一个片段中指定多个参数。下面就是一个示例。在 Global.asax.cs 文件的 RegisterRoutes 静态方法内,可以使用 MapRoute 方法定义新的路由:

```
routes.MapRoute("distance",
    "Simple2/Distance/{x1},{y1}/{x2},{y2}",
    new { Controller = "Simple2", action = "Distance" }
);
```

注意,这里使用逗号字符来分隔 x 坐标和 y 坐标。现在就可以通过 URL 地址来调用该操作方法:

```
/Simple2/Distance/0,0/1,2
```

URL 中的逗号字符看起来可能有些奇怪,但该路由选择是非常强大的!

前面的这些示例使用了 Response.Write,这违反了关注点分离的原则。管理数据的“视图”并不是控制器的任务。MVC 中的“V”(即视图)可以对此做更好的处理。

34.6 视图

视图用于向用户提供用户界面(UI)。视图提供对模型(Model)的引用,并把模型转换为准备呈现给用户的格式。在 ASP.NET MVC 中,这包括检查由控制器提交的 ViewDataDictionary(通过 ViewData 属性访问)以及将其转换为 HTML。注意,ViewBag 是 ViewData 对象的封装器,可以为 ViewBag 创建动态属性,而不是在 ViewData 对象中把“神奇的”字符串用作键。

在强类型化视图中(34.6.2 节中会对此做深入讨论),ViewDataDictionary 含有由视图显示的强类型化模型对象。模型可能代表了实际的域对象,如 Product 实例;也可能是视图特有的显示模型对象,如 ProductEditViewData 实例。

快速查看一个视图示例。下面的代码示例给出了默认情况下 ASP.NET MVC 项目模板中 `Index` 视图的 C# 版本：

```
@{
    ViewBag.Title = "Home Page";
}
@section featured {
    <section class="featured">
        <div class="content-wrapper">
            <hgroup class="title">
                <h1>@ViewBag.Title.</h1>
                <h2>@ViewBag.Message</h2>
            </hgroup>
            <p>
                To learn more about ASP.NET MVC visit
                <a href="http://asp.net/mvc" title="ASP.NET MVC
                    Website">http://asp.net/mvc</a>.
                The page features <mark>videos, tutorials, and samples</mark> to
                help you get the most from ASP.NET MVC.
                If you have any questions about ASP.NET MVC visit
                <a href="http://forums.asp.net/1146.aspx/1?MVC"
                    title="ASP.NET MVC Forum">our forums</a>.
            </p>
        </div>
    </section>
}
<h3>We suggest the following:</h3>
<ol class="round">
    <li class="one">
        <h5>Getting Started</h5>
        ASP.NET MVC gives you a powerful, patterns-based way to build dynamic
        websites that enables a clean separation of concerns and that gives you
        full control over markup for enjoyable, agile development. ASP.NET MVC
        includes many features that enable fast, TDD-friendly development for
        creating sophisticated applications that use the latest web standards.
        <a href="http://go.microsoft.com/fwlink/?LinkId=245151">Learn more . . . </a>
    </li>
    <li class="two">
        <h5>Add NuGet packages and jump-start your coding</h5>
        NuGet makes it easy to install and update free libraries and tools.
        <a href="http://go.microsoft.com/fwlink/?LinkId=245153">Learn more . . . </a>
    </li>
    <li class="three">
        <h5>Find Web Hosting</h5>
        You can easily find a web hosting company that offers the right mix of
        Features and price for your applications.
        <a href="http://go.microsoft.com/fwlink/?LinkId=245157">Learn more . . . </a>
    </li>
</ol>
```

虽然这是视图的极为简单的情况，但却对于指出 ASP.NET MVC 中视图的关键细节是很有帮助的。需要注意的第一点是，使用 `@{和}` 分隔某些文本块。这些字符用于给 Razor 视图引擎定义代码块，Razor 视图引擎是显示 ASP.NET MVC 视图的默认视图引擎。本章后面将详述 Razor 语法。另外

要注意其中遗漏了一些标准的 HTML 页面元素，没有 doctype、html、head 和 body 元素。与 Web 窗体中的母版页一样，ASP.NET MVC 中的布局允许给站点定义公共模板，由站点的所有视图继承。布局详见本章后面的内容。

如果熟悉 Web 窗体的 Page 指令，就会注意到其中也没有 Page 指令，这是因为这个视图不是 Web 窗体，实际上，该视图甚至不使用 WebFormViewEngine 显示。ASP.NET MVC 允许交错地使用不同的视图引擎，但默认的视图引擎是 RazorViewEngine。

ASP.NET MVC 中的视图要么继承于基类 System.Web.Mvc.ViewPage，该类继承于 System.Web.UI.Page(对于 WebFormViewEngine)；要么继承于 System.Web.Mvc.WebViewPage，该类继承于 System.Web.WebPages.WebPageBase(对于 RazorViewEngine)。强类型化视图继承于泛型 ViewPage<T>或 WebViewPage<T>，这也取决于视图引擎。

而在关注点分离原则中，视图不应该包括应用程序和业务逻辑。事实上，视图应该包括尽可能少的代码。虽然视图包含视图逻辑是完全可以接受的，但视图通常是应用程序中最难以按照自动化方式进行测试的部分，因此它们往往能从很少的代码中受益。

34.6.1 指定视图

至此，本章已经介绍了视图能做的事情以及不能做的事情，但是还没有涉及如何指定视图，以显示特定操作的输出。事实证明，在遵循架构中的隐式约定时，这是很简单的任务。

在创建新的项目模板时，项目会包含以非常特别的方式结构化的 Views 目录，如图 34-10 所示。

按照约定，Views 目录为每个控制器都包含一个文件夹，其名称与控制器的名称相同，但是没有后缀 Controller。在每个控制器文件夹中，每个操作方法都有一个视图文件，其名称与操作方法名称相同。这就提供了视图与操作方法进行关联的基础。

例如，操作方法可以通过 View 方法返回 ViewResult，如下所示：

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        ViewBag.Title = "Home Page";
        ViewBag.Message = "Welcome to ASP.NET MVC!";
        return View();
    }
}
```

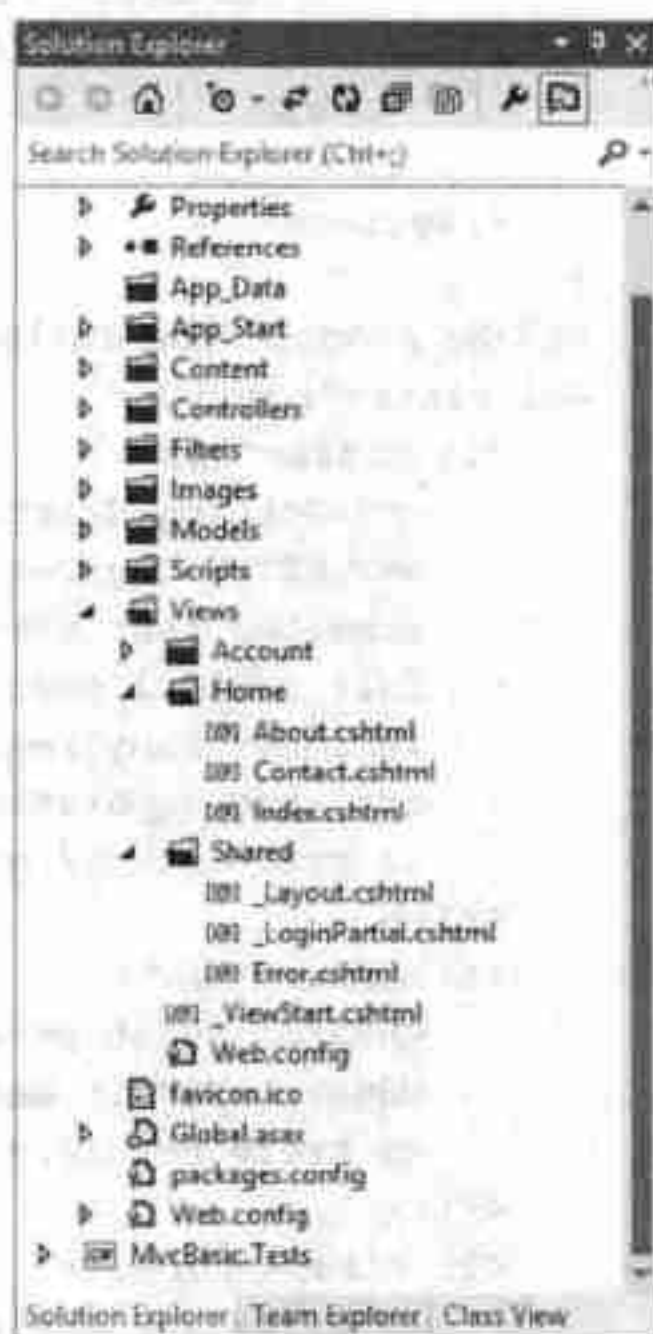


图 34-10

该方法看起来很熟悉，它是默认项目模板中 HomeController 的 Index 操作方法。因为没有指定视图名称，所以该方法返回的 ViewResult 将查找与 /Views/ControllerName 目录中操作名称相同的名称。在本例中，选择的视图是 /Views/Home/Index.cshtml 或 /Views/Home/Index.cbhtml。

与 ASP.NET MVC 中的大部分对象一样，也可以重写这个约定。假设希望 Index 操作显示不同的视图，那么可以提供如下所示的另一视图名称：


```
public ActionResult Index()
{
    ViewBag.Title = "Home Page";
    ViewBag.Message = "Welcome to ASP.NET MVC!";
    return View("NotIndex");
}
```

在本例中，仍然在 `/Views/Home` 目录中查找，但是选择 `NotIndex.cshtml` 作为视图。在一些情况下，我们甚至可能希望指定位于完全不同的目录结构中的视图。

可以使用 `~` 字符语法来提供视图的完整路径，如下所示：

```
public ActionResult Index()
{
    ViewBag.Title = "Home Page";
    ViewBag.Message = "Welcome to ASP.NET MVC!";
    return View("~/Some/Other/View.cshtml");
}
```

在使用这种语法时，必须提供视图的文件扩展名，因为这将避开视图引擎的内部查找机制，该机制的目的是查找视图。

34.6.2 ASP.NET MVC 布局

我们一般希望给整个站点，至少是站点的各个部分，创建类似的外观和操作方式。在典型的 Web 窗体应用程序中使用 `.aspx` 页面时，ASP.NET 2.0 引入了母版页的概念来实现这个功能。Razor 提供了 Layouts 功能，在 Razor 视图中支持这个概念。

在 Razor 视图中查看布局的最简单方式是查看本章前面创建的默认项目。在图 34-10 中，Views 文件夹下的 Shared 文件夹没有对应的控制器。大多数视图都位于对应控制器的文件夹中，控制器给文件夹提供了数据，而 Shared 文件夹中的视图由多个视图共享。还要注意，Shared 文件夹中的许多视图都遵循如下命名约定：最前面是一条下划线。这个约定从 ASP.NET Web 页面转入，在各个 Views 文件夹和进入 MVC 的路由中没有内置对这个约定的相同保护。布局页面不能直接使用和浏览。Web Pages 架构配置为不允许直接请求以下划线开头的文件。下面的代码示例显示了布局文件的 C# 代码：

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>@ViewBag.Title - My ASP.NET MVC Application</title>
    <link href="~/favicon.ico" rel="shortcut icon" type="image/x-icon" />
    <meta name="viewport" content="width=device-width" />
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
  </head>
  <body>
    <header>
      <div class="content-wrapper">
        <div class="float-left">
          <p class="site-title">
            @Html.ActionLink("your logo here", "Index", "Home")
          </p>
        </div>
      </div>
    </body>
  </html>
```

```

        <div class="float-right">
            <section id="login">
                @Html.Partial("_LoginPartial")
            </section>
            <nav>
                <ul id="menu">
                    <li>@Html.ActionLink("Home", "Index", "Home")</li>
                    <li>@Html.ActionLink("About", "About", "Home")</li>
                    <li>@Html.ActionLink("Contact", "Contact",
                        "Home")</li>
                </ul>
            </nav>
        </div>
    </div>
</header>
<div id="body">
    @RenderSection("featured", required: false)
    <section class="content-wrapper main-content clear-fix">
        @RenderBody()
    </section>
</div>
<footer>
    <div class="content-wrapper">
        <div class="float-left">
            <p>&copy; @DateTime.Now.Year - My ASP.NET MVC Application</p>
        </div>
    </div>
</footer>

    @Scripts.Render("~/bundles/jquery")
    @RenderSection("scripts", required: false)
</body>
</html>

```

首先要注意，第一次查看 Index 视图时，doctype、html、head 和 body 标记指定为遗漏。这个布局文件默认由站点的所有视图继承。下一个要点是调用@RenderBody 方法。基于这个布局文件的视图把内容插入这个调用所在的地方。

视图可以设置 WebPageBase 类的 Layout 属性，指定是否应使用布局页面。这可以在视图中使用如下 Razor 代码块来实现：

```

@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

```

这个属性必须在要使用布局的所有视图中设置。一个站点可以有多个布局文件，在每个视图中只要改变布局文件名，就可以指定要使用的布局文件。在所有视图中设置 Layout 属性，需要许多重复的代码。为了避免这么多重复，可以使用 Razor 的另一功能，在一个地方为所有视图定义布局逻辑。这不仅减少了重复，还使代码更容易维护。在 Views 文件夹下添加文件_ViewStart.cshtml。_ViewStart 文件可以用于定义在开始显示每个视图时执行的公共视图代码。可以把 Layout 属性的设置从视图移到_ViewStart 文件中，使之应用于所有视图。在任意视图中，只要把 Layout 属性设置为另一个布局，就可以覆盖上述设置。在默认模板中，_ViewStart 文件包含上述设置 Layout 属性的代

码块。在 Visual Studio 中使用 Add View 对话框创建视图时，可以指定要在视图中使用的布局。图 34-11 显示的 Add View 对话框选中了 Use a layout or master page 选项。注意如果使用 _ViewStart 文件，布局文件名字段就可以为空。



图 34-11

在布局文件中，另一重点是对 `@RenderSection` 方法的调用。前面只介绍了使用 `@RenderBody` 方法显示布局文件的主体部分，Razor 还支持创建多个“命名的部分”，它们可用于为最终响应的多个不连续区域显示动态内容。`@RenderSection` 的第一个参数是部分的名称，这个名称用于指定视图中某部分的内容。可以设置 `required` 参数，指定某个部分是必需的还是可选的。再看看前面的布局文件代码段，调用 `@RenderSection` 以显示命名的部分 `featured`：

```
@RenderSection("featured", required: false)
```

现在看看 Views/Home 文件夹中的 Index 视图，该视图包含 `featured` 部分。无论 `featured` 部分在视图文件的什么地方，它都会显示在布局中调用 `@RenderSection` 的地方。下面的示例代码显示了 Index 视图中的部分块：

```
@section featured {
    <section class="featured">
        <div class="content-wrapper">
            <hgroup class="title">
                <h1>@ViewBag.Title.</h1>
                <h2>@ViewBag.Message</h2>
            </hgroup>
            <p>
                To learn more about ASP.NET MVC visit
                <a href="http://asp.net/mvc" title="ASP.NET MVC
                Website">http://asp.net/mvc</a>.
                The page features <mark>videos, tutorials, and samples</mark> to
                help you get the most from ASP.NET MVC.
                If you have any questions about ASP.NET MVC visit
```



```

        <a href="http://forums.asp.net/1146.aspx/1?MVC" title="ASP.NET MVC
            Forum">our forums</a>.
    </p>
</div>
</section>
}

```

注意视图中的部分可以包含静态和动态内容。使用部分可以组织视图文件中的内容，而无须考虑它们最终显示的顺序。部分还允许布局文件为视图显示的动态内容指定多个位置。

34.6.3 强类型化视图

假设有一个希望在视图中显示的 **Product** 实例列表，为此，一种方法是向视图数据字典添加产品并在视图中遍历它们。

例如，控制器操作中的代码如下所示：

```

public ActionResult List()
{
    var products = new List<Product>();
    for(int i = 0; i < 10; i++)
    {
        products.Add(new Product {ProductName = "Product " + i});
    }
    ViewBag.Products = products;
    return View();
}

```

在视图中，可以使用 **Razor** 语法遍历并显示产品，如下所示：

```

<ul>
@foreach(Product p in ViewBag.Products){
    <li>@p.ProductName</li>
}
</ul>

```

ViewBag 包含动态属性，且不是强类型化的，因此在 **foreach** 循环中需要做转换。如果提供的视图类型与传送进来的模型类型匹配，那么代码会更简洁。此时，强类型化视图就应运而生。

在控制器方法中，可以通过重载 **View** 方法(通过该方法可以传递模型)来指定模型：

```

public ActionResult List()
{
    var products = new List<Product>();
    for(int i = 0; i < 10; i++)
    {
        products.Add(new Product {ProductName = "Product " + i});
    }
    return View(products);
}

```

在后台，这段代码将 **ViewData.Model** 属性的值设置为传送给 **View** 方法的值。接下来，把视图的类型修改为继承于 **WebViewPage<T>**。在 **MVC** 模型中，视图确实不能包含隐藏代码文件。事实上，在默认情况下，**ASP.NET MVC** 中不存在视图的隐藏代码文件。如果希望强类型化视图，只需

在视图中给@Model 指令添加该类型，如下所示：

```
@Model System.Collections.Generic.List<MvcBasic.Models.Product>
```

这是得到强类型化视图的首选方式。现在，在视图的标记中，可以使用完全支持的 IntelliSense 来访问强类型化的 ViewData.Model 属性：

```
<ul>
@foreach (Product p in Model) {
    <li>@p.ProductName</li>
}
</ul>
```

34.6.4 使用 HTML 辅助方法

ASP.NET MVC 架构的显著特征是提供了对应用程序的完全控制，包括 HTML 标记。许多人声称这是该架构的一大优点，毕竟完全控制良好。这确实是该架构的特征之一，至于是否是优点则要视具体情况而定。

有时我们并不希望对标记进行控制。因为我们并不关心外观，所以在删除控件时最好弄清楚标记的问题。而在有些时候，我们则希望完全控制标记。虽然有控制权很不错，但是这也意味着更多的责任。现在，我们应负责输出标记的结果，否则就应该通过 Web 窗体中的服务器控件来处理标记。

HTML 辅助方法提供了介于两者之间的功能。这些辅助方法包含在架构中，可以帮助显示通常情况下的标记。在大多数情况下，它们处理的是常见错误，例如忘记编码特性值等。

34.6.5 HtmlHelper 类和扩展方法

WebViewPage 类有名为 Html 的 HtmlHelper 属性。在查看 HtmlHelper 的方法时，将会注意到该属性是很少见的。该属性是附加扩展方法的定位点。在导入 System.Web.Mvc.Html 名称空间时(默认情况下在默认模板中导入)，Html 属性通过大量辅助方法突然之间变得很出彩。

在如图 34-12 所示的屏幕截图中，通过灰色向下箭头表示扩展方法。

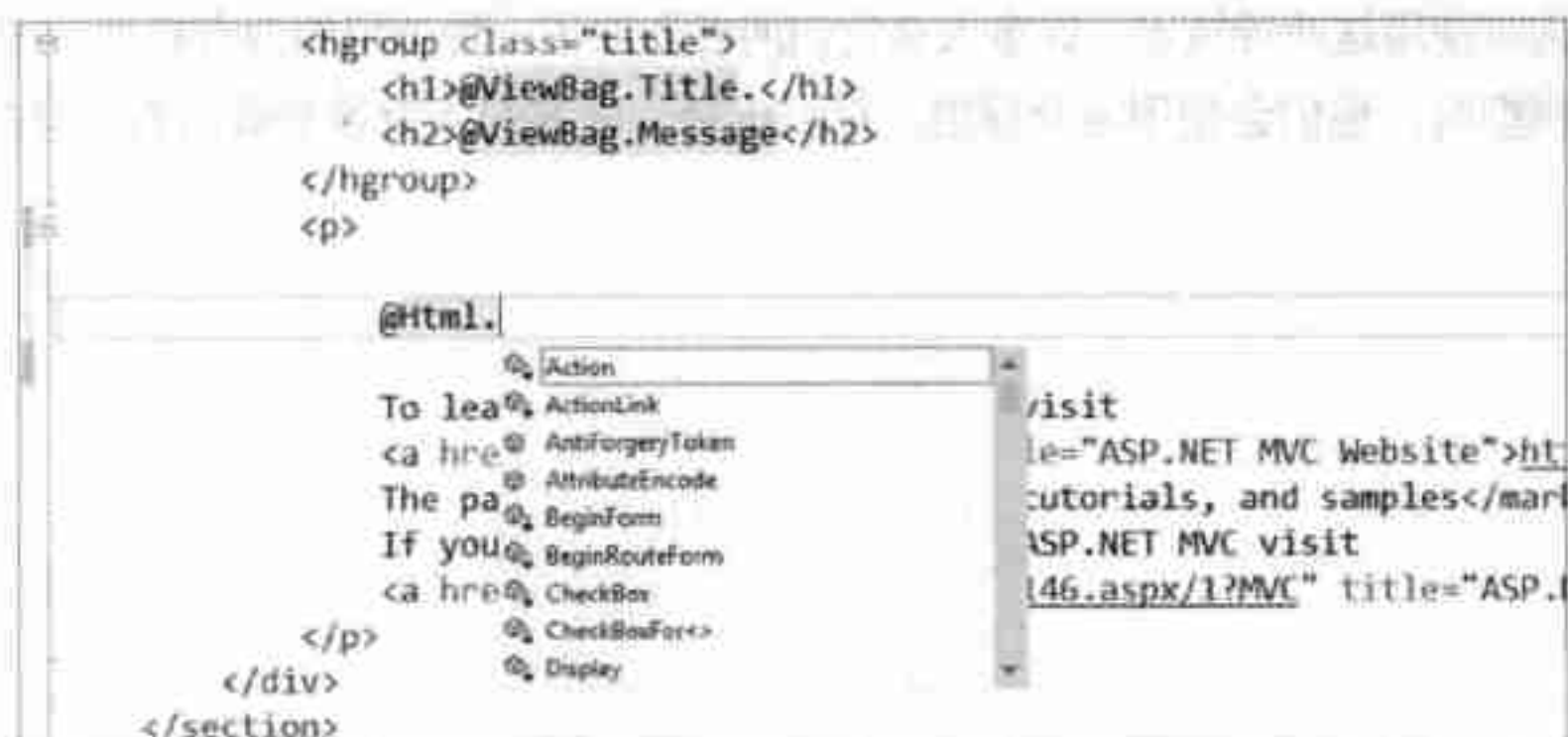


图 34-12

这种方法的优点在于，因为这是常规的扩展方法，所以如果不喜欢辅助方法包含在架构中，就可以删除这个名称空间，并添加自己的 HTML 辅助扩展方法。同样，可以提供方便的常规位置，以便于我们编写 HtmlHelper 类的扩展方法，从而添加自己的辅助方法。

所有的辅助方法都共享一些常见模式，有必要强调这些模式：

- 所有辅助方法的属性都对属性值进行编码。
- 所有辅助方法的 HTML 都对显示的值进行编码，如链接文本。
- 接受 `RouteValueDictionary` 的辅助方法包含对应的重载方法，该重载方法允许将匿名对象指定为字典。
- 同样，接受 `IDictionary<string,object>` (用于指定 HTML 特性)的辅助方法也包含对应的重载方法，该重载方法允许将匿名对象指定为字典。
- 用于显示表单字段的辅助方法会在 `ModelState` 字典中自动地查找这些字段的当前值。该辅助方法的名称参数将作为字典的键。
- 如果 `ModelState` 包含错误，那么与错误相关的表单辅助方法将显示 `input-validation-error` 的 CSS 类以及任何显式指定的 CSS 类。包括在项目模板中的默认样式表 `style.css` 中包含了这个类的样式。

视图及 `ViewEngines` 的用途非常特殊和有限。它们的存在是为了获取从控制器传送过来的数据，并生成格式化的输出，通常是 HTML。开发人员除了这些简单的职责或“关注点”之外，还需要以任何喜欢的方式来达到视图的目标。

34.7 本章小结

在使用 ASP.NET MVC 时，ASP.NET Web 窗体开发人员需要适应很多的不同之处(与 Web 窗体相比)。在很多方面，使用 ASP.NET MVC 会让人感觉“像是回到了 10 年前”的传统 ASP 开发方式——特别是在使用 UI 和视图时。

对某些人而言，这是一处很受欢迎的改动。但是，对于另一些人而言，情况并非如此。这的确需要适应，但是无论使用哪一种开发方式，核心的 ASP.NET 功能和 .NET 架构通常都会提供支持。

最后，最重要的是 ASP.NET Web 窗体和 ASP.NET MVC 都基于 ASP.NET。可以认为 ASP.NET > Web 窗体，并且 ASP.NET > ASP.NET MVC。这两种技术具有相同的基础，因此可以使用这两种技术中的任何一种，或者同时使用这两种技术。许多人从它们两者发现了一种可用的混合模型，或者当从一种模型移至另一种模型时，他们会使用混合模型。可以选择一种我们认为效率最高的模型并运行。

第 35 章

ASP.NET Web Pages 和 Razor

本章要点

- ASP.NET Web Pages 简介
- Razor 语法概述
- 用 Razor 显示数据
- 使用辅助函数

本书提到了“经典的”ASP。读者可能没有用过这个杰出的技术。但如果用过它，使用 Razor 语法开发 ASP.NET Web Pages 可能会令人想起经典 ASP 的一些东西。但该技术完全不同。本章将介绍如何使用 Web Pages with Razor 提供比经典 ASP 更强大的解决方案。

上一章讨论了 ASP.NET MVC。现在就有机会使用 ASP.NET Web Form 视图引擎了。本章还要介绍如何使用另一视图引擎：Razor 视图引擎。Razor 视图引擎使用的 Razor 语法与 Web Pages 相同。如果应用程序对于 ASP.NET Web Pages 而言过于复杂，最好使用 Razor 视图引擎把一些部分移入 ASP.NET MVC。



Razor 不仅仅是视图引擎，它的主要目标是成为模板引擎。

尽管本章不介绍 ASP.NET MVC Razor 视图引擎，但阅读完本章后，你能理解 Razor 语法的工作方式，并能使用该语法和上一章学习的 MVC 规则。



本章的目标是概述 Web Pages、Razor 语法和 ASP.NET WebMatrix。但本书没有足够的篇幅讨论 Web Pages 和 Razor 的每个主题。如果读者希望学习 Web Pages 和 Razor 的更多知识，可参阅《ASP.NET MVC 4 高级编程》，或者在线搜索 Web Pages、Razor 或 WebMatrix 的更多信息。

35.1 ASP.NET Web Pages 概述

ASP.NET Web Pages 是相当新的技术，微软引入该技术是为了使用 ASP.NET 开发动态的 Web 应用程序，但不一定需要 Visual Studio 2012。相反，要开发 ASP.NET Web Pages，可以使用基本的文本编辑器(如记事本)，或者选择首选方法——使用 ASP.NET WebMatrix 等 IDE。



ASP.NET WebMatrix 有许多 ASP.NET Web Pages 没有的功能。在 WebMatrix 中，开发人员可以连接 NuGet，远程访问站点，使用 IIS Express 在本地运行站点，在 IIS Express 中捕获请求和响应消息流，甚至在该 IDE 中开发 PHP、node.js 和其他应用程序。与其他 ASP.NET 技术类似，ASP.NET WebMatrix 团队正在缩短发布时间表，请在线搜索最新版本。

与静态的 HTML 页面不同，动态页面允许终端用户与 Web 窗体交互操作，操纵和保存数据，与社交媒体服务等第三方服务交互。



与 Web Pages 的第一版相比，Web Pages 2 提供了几个新功能和好处。本章只使用 Web Pages 2，介绍 ASP.NET WebMatrix 2(这是一款免费的 IDE)。但可以使用 Visual Studio 2012 开发相同的页面。

与 ASP.NET MVC 类似，Web Pages 也允许扩展几个领域。扩展 Web Pages 的一种方式创建并使用辅助方法。辅助方法非常类似于 ASP.NET 服务器控件，因为大多数辅助方法都封装了可重用的标记。如本章所示，几个辅助方法内置于 Web Pages 中。使用 NuGet(详见附录 G)或通过定制开发，可以包含其他辅助方法。

35.2 使用 Razor 创建 HTML 窗体

开始使用 Razor 建立 Web 页面之前，需要了解使用 Razor 语法的更多知识。在 ASP.NET Web 窗体中，使用如下脚本块可以定义内联代码：

```
<script runat="server">
    // Insert code here
</script>
```

Razor 语法略有不同。仍可以定义内联代码，但不使用上述格式的脚本块，而可以定义如程序清单 35-1 所示的脚本块。

程序清单 35-1 用于 C# 的 Razor 脚本块

```
@{
```



```
<!-- Insert code here -->
}
```

在介绍 Web Pages 中的特定功能和辅助方法之前，先创建一个可以开始使用的 HTML 页面。为此，使用 WebMatrix。打开 WebMatrix，窗口就会提示用户选择动作，如图 35-1 所示。



图 35-1

在这个屏幕中，有 4 个选项。可以打开已有的站点、使用站点模板、使用应用程序库加载应用程序。也可以选择不再显示这个快捷菜单，而是用文件浏览器功能加载 WebMatrix。

站点模板列表如图 35-2 所示，其中包含用于 ASP.NET、PHP、node.js 和 HTML 的几个 Web 模板。ASP.NET 模板用于使用 Web Pages 的示例应用程序。

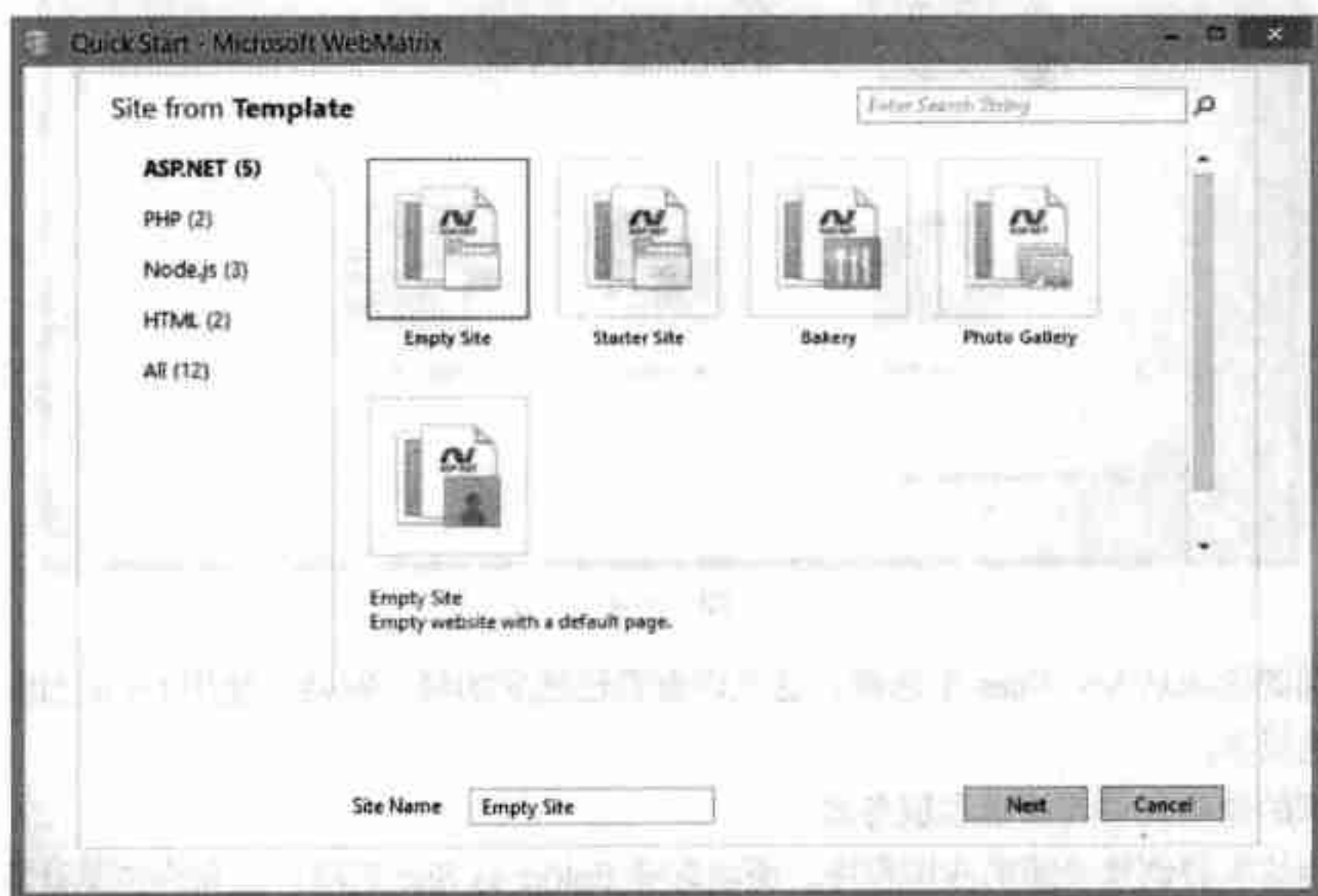


图 35-2

应用程序库如图 35-3 所示，其中包括许多可加载的预建应用程序。这个库中包含的大多数应用程序都使用 ASP.NET 或 PHP。



图 35-3

这个练习选择使用已有的站点。选择后，屏幕就会显示 3 个选项，如图 35-4 所示。



图 35-4

- 顶部的选项从 My Sites 中选择。这允许查看已建立的每个网站，使用 IIS 或 IIS Express 的本地版本。
- 底部的选项允许连接远程服务器。
- 因为这里要创建全新的应用程序，所以选择 Folder as Site 选项。这允许浏览计算机中(或网络上)的文件夹，创建绑定了随机端口号的应用程序。

应用程序在 WebMatrix 中运行，添加空白的 robots.txt 文件，如图 35-5 所示。

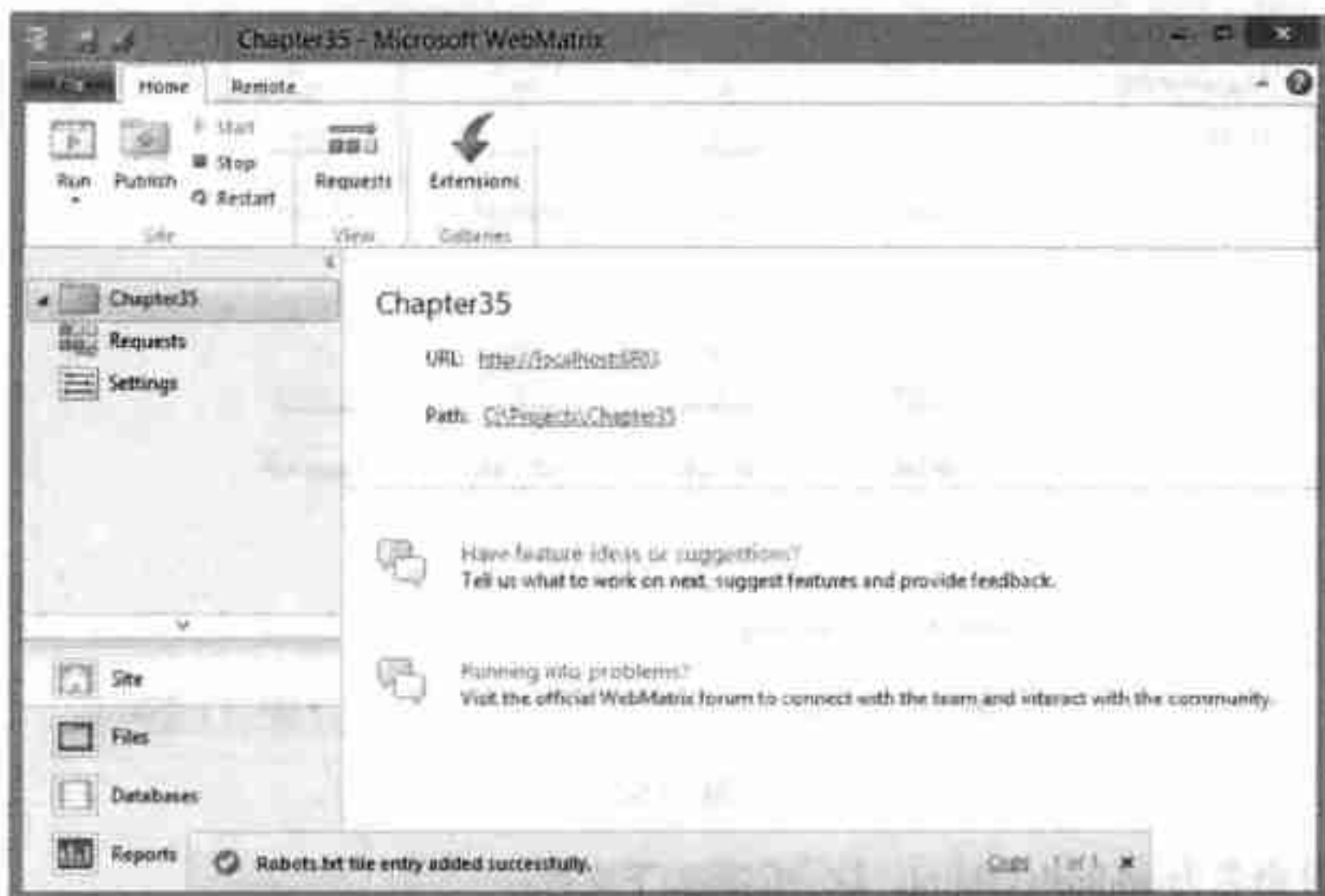


图 35-5

建立 Web Pages 应用程序的下一步是切换到 Files 视图。注意在图 35-6 中，即使使用 WebMatrix 建立这个应用程序，也可以随时在 Visual Studio 2012 中打开该应用程序。

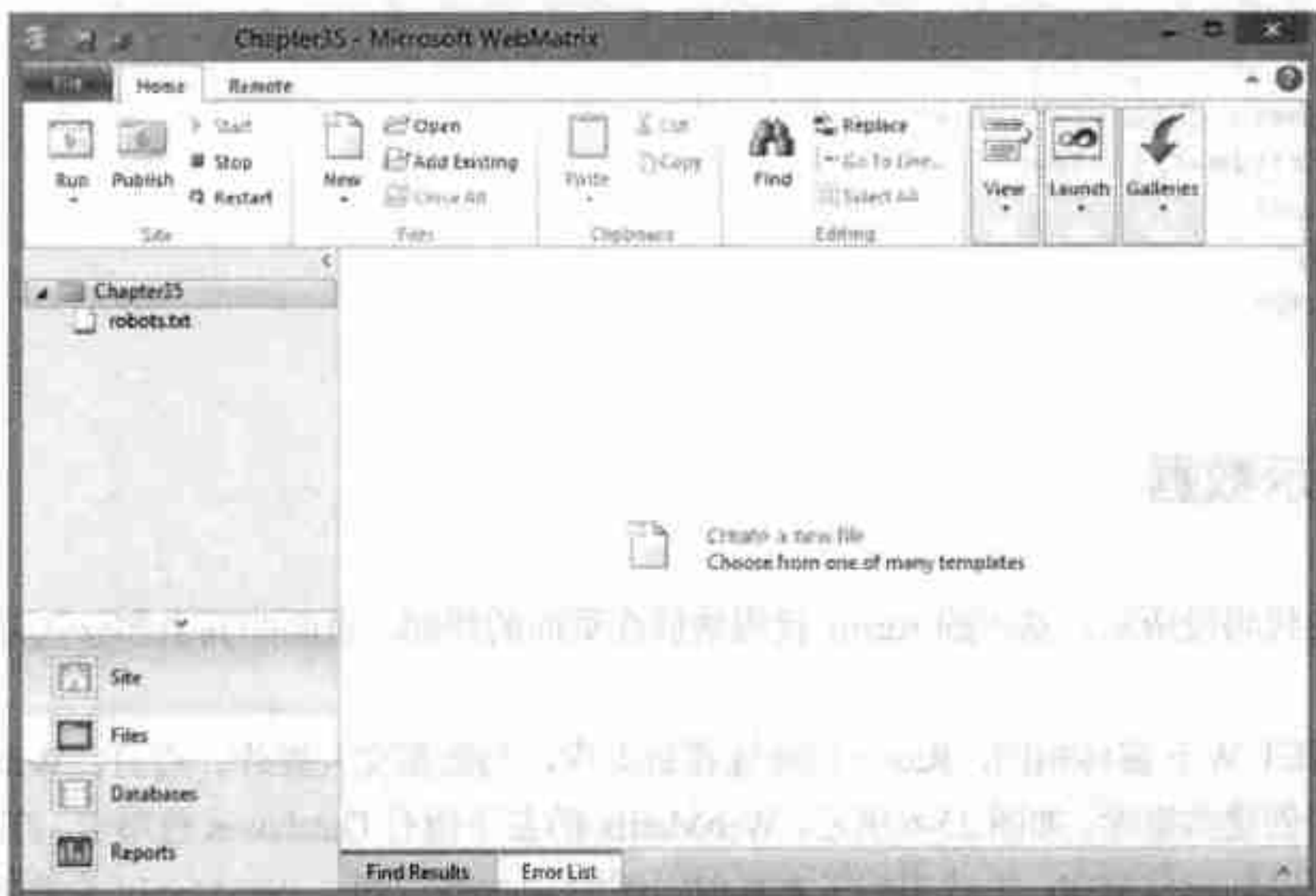


图 35-6

现在已经创建了 Web Pages 应用程序，目前在 WebMatrix 中，正要给应用程序添加文件。如图 35-6 所示，添加文件是很简单的。选择创建新文件的选项，接着选择特定的文件类型，如图 35-7 所示。尽管下一步是创建 HTML，但为 Web Pages 选择两种 Razor 格式中的一种，给 C# Web Pages 选择.cshtml，给 VB Web Pages 选择.vbhtml。



图 35-7

把 C# 文件命名为 addtask.cshtml，这会创建如下文件：

```
@{
}
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title></title>
  </head>
  <body>
  </body>
</html>
```

35.3 显示数据

如前面的代码段所示，基本的 Razor 代码块放在页面的顶部，页面的其余部分包含基本页面的 HTML。

与 ASP.NET Web 窗体相同，Razor 也能连接数据库，与数据交互操作。而且，WebMatrix 可以在应用程序中创建数据库。如图 35-6 所示，WebMatrix 的左下角有 Databases 选项卡。打开 Databases 选项卡，就可以创建数据库，方式类似于本章前面添加 addtask 文件。WebMatrix 为数据库提供了 3 个内置选项。默认选项是 SQL Server CE (Compact Edition)，这个数据库的扩展名是.sdf。也可以创建 SQL Server 数据库或 MySQL 数据库。对于这里的应用程序，选择默认选项，如图 35-8 所示创建数据库。

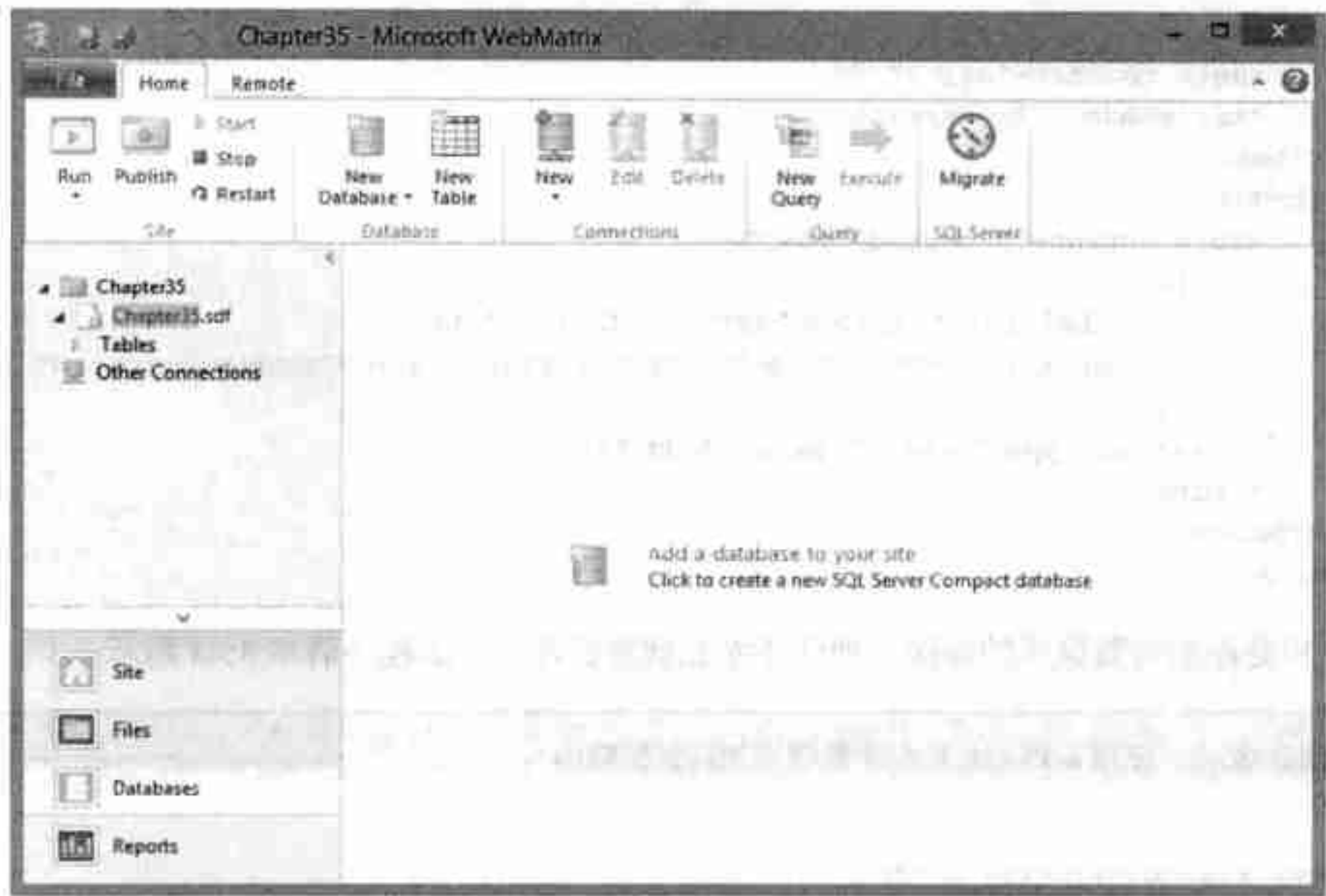


图 35-8

创建数据库后，添加数据表来存储任务。数据表很简单——数据表的定义如图 35-9 所示。

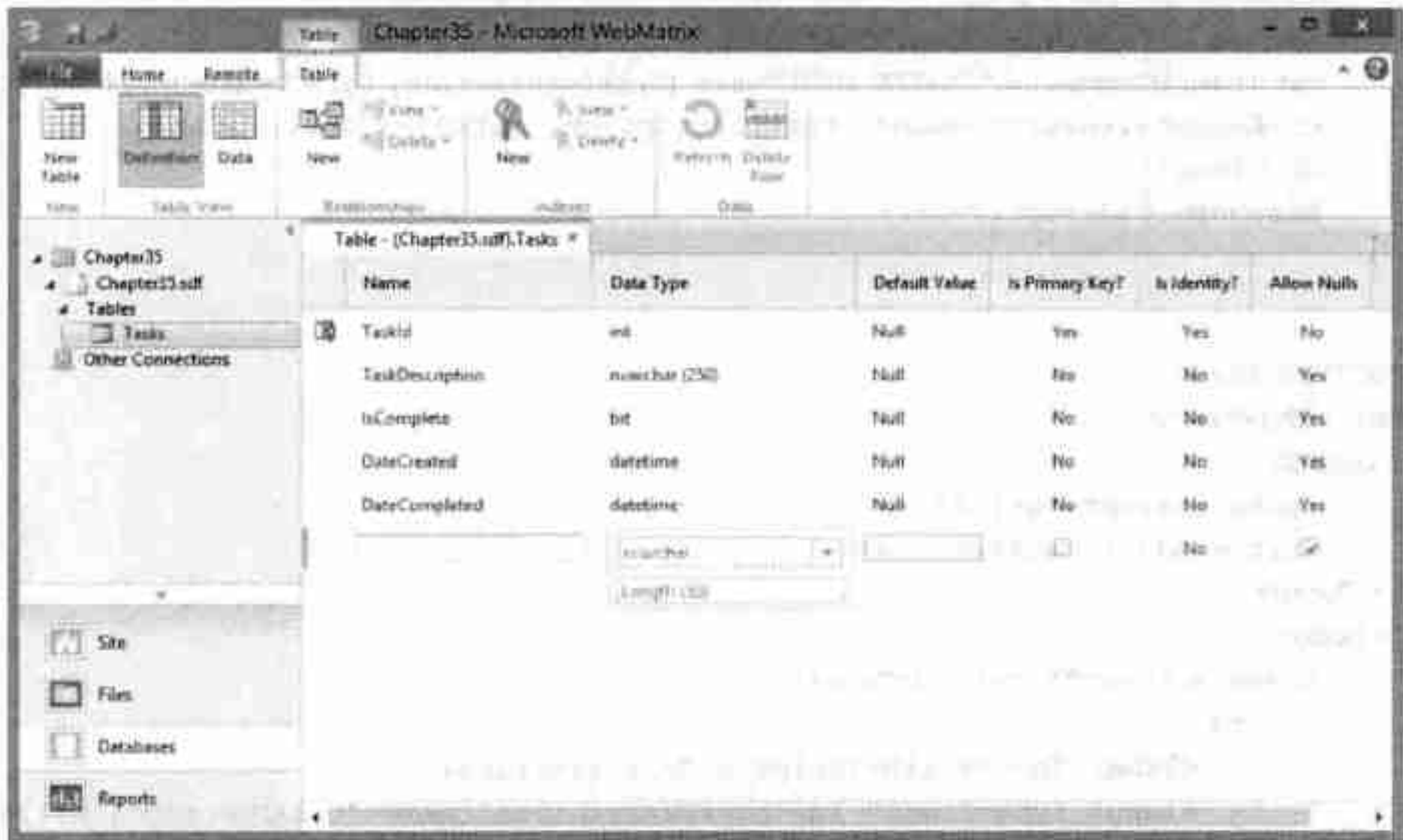


图 35-9

现在，有了数据结构，就该看看 addtask 页面了。下面添加窗体元素以便添加任务。在这个例子中，只需一个文本框和提交按钮就可以添加任务。addtask 页面的 HTML 如程序清单 35-2 所示(本章下载代码的 addtask.cshtml)。

程序清单 35-2 应用程序的 addtask 页面

```
<!DOCTYPE html>
<html lang="en">
```

```

<head>
  <meta charset="utf-8" />
  <title>Add a Task</title>
</head>
<body>
  <form action="" method="post">
    <p>
      <label for="taskDescription">Task:</label>
      <input type="text" id="taskDescription" name="taskDescription" />
    </p>
    <input type="submit" value="Add Task" />
  </form>
</body>
</html>

```

下一步是添加对数据库的连接，把任务添加到数据库中，如程序清单 35-3 所示。

程序清单 35-3 使用 addtask 页面把数据添加到数据库中

```

@{
  var taskDescription = "";

  if (IsPost) {
    taskDescription = Request.Form["taskDescription"];

    var db = Database.Open("Chapter35");
    var insertCommand = "INSERT INTO Tasks (TaskDescription, DateCreated) Values (@0, @1)";
    db.Execute(insertCommand, taskDescription, DateTime.Now);
    db.Close();
    Response.Redirect("~/");
  }
}

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Add a Task</title>
  </head>
  <body>
    <form action="" method="post">
      <p>
        <label for="taskDescription">Task:</label>
        <input type="text" id="taskDescription" name="taskDescription" />
      </p>
      <input type="submit" value="Add Task" />
    </form>
  </body>
</html>

```

这个页面还没有完成。稍后需要再次访问这个页面，进行最后的调整。

项目的下一步是列出所有的任务。因为这是应用程序的主要目的，所以任务列表会显示在主页面上。给应用程序添加 default.cshtml 文件，以便可以开始添加必要的代码以显示任务。数据不应太多，不需要分页显示，所以在主页面上添加两项。第一项是从 WebGrid 辅助方法中生成的输出，第

二项是允许用户将更多的任务添加到列表中的链接。完成的主页面的源代码如程序清单 35-4 所示。

程序清单 35-4 在主页面上列出输入的任务

```
@{
    var db = Database.Open("Chapter35");
    var tasks = db.Query("SELECT * FROM Tasks WHERE IsComplete = 0 or IsComplete IS NULL");
    var grid = new WebGrid(source: tasks);
}
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title></title>
        <style type="text/css">
            .grid { margin: 4px; padding: 2px; border: 1px solid #666; }
            .grid td, th { border: 1px solid #fff; padding: 5px; }
            .head { background-color: #36648B; }
            .head a { color: #FFF; text-decoration-style: none; }
            .alt { background-color: #F0F8FF; }
        </style>
    </head>
    <body>
        <h1>Open Tasks</h1>
        <div>
            @grid.GetHtml(
                tableStyle: "grid",
                headerStyle: "head",
                alternatingRowStyle: "alt",
                columns: grid.Columns(
                    grid.Column("TaskDescription", "Task")
                )
            )
        </div>
        <p>
            <a href="addtask.cshtml">Add More Tasks</a>
        </p>
    </body>
</html>
```

与 Razor 语法中的大多数辅助方法一样，WebGrid 辅助方法允许在代码中设置一组定制属性。在这个例子中，设置了 3 个样式属性，添加了一个列的定制列表。列的列表只包含任务描述。列的第一个参数是 SQL 列名，第二个参数是友好的列标题，它们会被添加到显示的 HTML 中。

35.3.1 验证

下面再次访问 addtask 页面。目前，对允许插入数据库表中的数据没有任何限制，所以可以在字段中输入超过 250 个字符，或者让字段为空。显然，这不是我们期望的。因此必须更新 addtask 页面，以包含验证过程。要添加的验证包括 ValidationSummary 辅助方法，该方法非常类似于第 6 章讨论的 Web 窗体控件 ValidationSummary。还要给 taskDescription 元素添加验证过程。

在源代码中，需要定义验证规则。在这个例子中，把任务设置为必选，且字段的长度不得超过 250 个字符。使用 Validation.RequireField 和 Validation.Add 就可以完成这项任务。Validation.Add 方

法的第一个参数接受元素名称，第二个参数接受验证类型。可用的验证类型包括：

- `Validator.DateTime([error message])`
- `Validator.Decimal([error message])`
- `Validator.EqualsTo([other element],[error message])`
- `Validator.Float([error message])`
- `Validator.Integer([error message])`
- `Validator.Range([min],[max],[error message])`
- `Validator.RegEx([regex pattern],[error message])`
- `Validator.Required([error message])`
- `Validator.StringLength([maximum length])`
- `Validator.Url([error message])`

与 Web 窗体相同，这里也需要检查页面是否有效。调用 `Validation.IsValid` 就可以确保这一点。另外，为了说明可以使用单独的错误，而不是只能提供汇总，下面使用 `ValidationMessage` 辅助方法显示用户友好的错误消息。程序清单 35-5 是实现了验证功能的 `addtask` 页面。

程序清单 35-5 用验证功能更新 `addtask` 页面

```
@{
    // Variables
    var taskDescription = "";

    // Validation
    Validation.RequireField("taskDescription", "A task is required.");
    Validation.Add("taskDescription", Validator.StringLength(250));

    // If the page has been posted back and if it's valid, insert the data and redirect
    if (IsPost) {
        if (Validation.IsValid()) {
            taskDescription = Request.Form["taskDescription"];

            var db = Database.Open("Chapter35");
            var insertCommand =
                "INSERT INTO Tasks (TaskDescription, DateCreated) Values(@0, @1)";
            db.Execute(insertCommand, taskDescription, DateTime.Now);
            Response.Redirect("~/");
        }
    }
}

<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>Add a Task</title>
    </head>
    <body>
        <form action="" method="post">
            <div>
                @Html.ValidationSummary()
```

```

</div>
<p>
    <label for="taskDescription">Task:</label>
    <input type="text" id="taskDescription" name="taskDescription" />
    @Html.ValidationMessage("taskDescription")
</p>
<input type="submit" value="Add Task" />
</form>
</body>
</html>

```

35.3.2 使用布局

Razor 提供的最有用的一项功能是其包含的模板功能。通常这是指可以创建的数据模板。但 Razor 还以布局的形式包含网站模板。布局非常类似于 ASP.NET Web 窗体中的母版页。使用 `RenderBody` 可以把文件中的所有内容显示在布局容器中。还可以使用 `RenderSection` 指定在某块区域显示的内容。程序清单 35-6 是 `_layout.cshtml` 页面中这两个辅助方法的示例。

程序清单 35-6 派生自 `_layout.cshtml` 文件的示例布局页面

```

@{
    var sidebarStyle = "#Sidebar { float: right; margin: 0; width: 15%; " +
        "padding: 5px; background-color: #f2f2f2; border: 1px solid #999; } " +
        "#MainContent { float: left; width: 80%; }";
}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>@Page.Title</title>
    <style>
        body { font-family: Arial; font-size: 11pt; }
        header { background-color: #3E5CA2; color: #fff; }
        footer { clear: both; text-align: center; font-size: 9pt; }
        @if (IsSectionDefined("Sidebar")) {
            @sidebarStyle
        }
    </style>
</head>
<body>
    <header>
        <h1>My Honey-Do List</h1>
    </header>
    @if (IsSectionDefined("Sidebar")) {
        <aside id="Sidebar">
            @RenderSection("Sidebar")
        </aside>
    }
    <section id="MainContent">
        @RenderBody()
    </section>
    <footer>
        Copyright 2013. All Rights Reserved.
    </footer>

```

```

        </footer>
    </body>
</html>

```

创建布局页面后,就要更新 addtask 页面(如图 35-7 所示)和主页面(如图 35-8 所示),让它们使用新的布局页面。注意,如果 Sidebar 部分没有定义在内容页面中,就忽略程序清单 35-6 中 if 语句的标记部分。

程序清单 35-7 使用新布局模板的更新过的 addtask 页面

```

@{
    Layout = "~/_layout.cshtml";
    Page.Title = "Add a task";
    // Removed the code below to simplify this example
    // . . .
}

```

程序清单 35-8 使用新布局模板的更新过的主页面

```

@{
    Layout = "~/_layout.cshtml";
    Page.Title = "My Tasks";
    // Removed the code below to simplify this example
    // . . .
}

<style type="text/css">
    .grid { margin: 4px; padding: 2px; border: 1px solid #666; }
    .grid td, th { border: 1px solid #fff; padding: 5px; }
    .head { background-color: #36648B; }
    .head a { color: #FFF; text-decoration-style: none; }
    .alt { background-color: #F0F8FF; }
</style>
<h1>Open Tasks</h1>
<div>
    @grid.GetHtml(
        tableStyle:= "grid",
        headerStyle:= "head",
        alternatingRowStyle:= "alt",
        columns:= grid.Columns(
            grid.Column("TaskDescription","Task")
        )
    )
</div>
@section Sidebar
<p>
    <a href="addtask.vbhtml">Add More Tasks</a>
</p>
End Section

```

使用新布局的页面示例如图 35-10 所示。

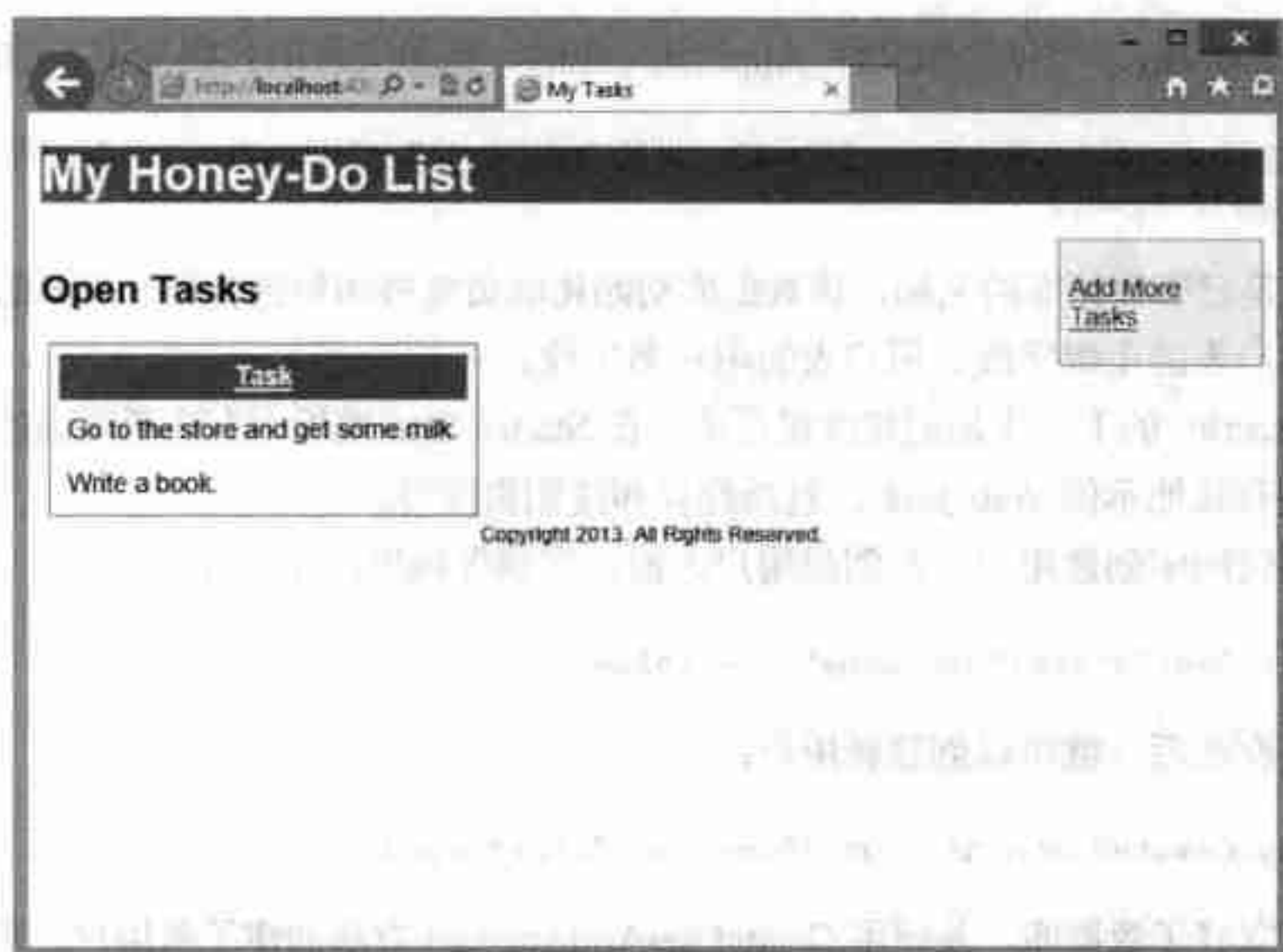


图 35-10

35.4 使用辅助方法

前面的 Web Pages 示例使用了几个内置的辅助方法。使用 WebGrid 显示数据，使用 ValidationSummary 和 ValidationMessage 显示数据异常信息，还使用 RenderBody 和 RenderSection 显示其他 Web Pages 文件的信息。这只是开始。还可以使用 Web Pages 内置的其他几个辅助方法。

35.4.1 核心辅助方法

使用几个核心辅助方法可以给应用程序添加基本功能，包括：

- WebSecurity
- OAuthWebSecurity
- ReCaptcha
- WebMail
- WebCache

1. WebSecurity 辅助方法

WebSecurity 辅助方法允许用户在应用程序中验证身份。还允许定义角色，再把它们分配给用户。这非常类似于 ASP.NET 处理成员资格和角色管理的方式。



这里不可能涵盖 WebSecurity 辅助方法提供的所有功能。建议查看 WebMatrix 中的 Starter Site 模板，更好地理解 WebSecurity 辅助方法。如果使用 Starter Site 模板，就会有 Account 文件夹包含处理成员资格和角色管理的大多数必需页面。

要使用 WebSecurity, 可以添加文件 _AppStart.cshtml, 在该文件的代码块中, 添加如下代码:

```
WebSecurity.InitializeDatabaseConnection("Chapter35", "UserProfile", "UserId",
    "Username", true);
```

第一个参数是已有数据库的名称, 该数据库初始化成员资格和角色管理。后面的 3 个参数分别用于用户表、用户表的主键字段、用户表的用户名字段。

使用 WebSecurity 的下一步是创建注册页面。在 Starter Site 模板中有注册页面的很好示例。在这个模板中, 还有其他示例 Web 页面, 包括登录和注销页面等。

还可以在源代码中创建用户。在创建用户之前, 应确保该用户不存在:

```
WebSecurity.UserExists("username") == false
```

验证用户不存在后, 就可以创建新用户:

```
WebSecurity.CreateUserAndAccount("username", "password")
```

现在, 就建立好了数据库, 并使用 CreateUserAndAccount 方法创建了新用户。用户创建好后, 就应以该用户的身份登录了。最好看看 Starter Site 模板中的注册页面是如何实现的。准备以该用户的身份登录时, 可以使用下面的代码检查凭证:

```
WebSecurity.Login("username", "password")
```

用户登录到站点后, 就要问候他。使用下面的代码可以显示用户的用户名:

```
if (WebSecurity.IsAuthenticated())
{
    @WebSecurity.CurrentUserName;
}
```

这段代码先检查用户是否通过验证。如果通过, 就显示当前的用户名。

要研究的最后一个方法用于注销用户。使用如下代码完成这个任务:

```
WebSecurity.Logout()
```

2. OAuthWebSecurity 辅助方法

OAuthWebSecurity 辅助方法允许在站点中使用 OAuth 和 OpenID 进行身份验证。换言之, 如果以前访问过一个网站, 而该网站要求用 Facebook、Google、Microsoft、Twitter 或 Yahoo! 登录, 该站点就在使用 OAuth 或 OpenID。Google 和 Yahoo! 使用 OpenID, 所以它们的实现方案相当简单。其他 3 个需要应用程序或客户端 ID 以及应用程序或客户端密钥, 才能完成身份验证。这三个使用 OAuth 标准。

如前所述, 数据库需要初始化。在 InitializeDatabaseConnection 之后, 可以添加下面的一个或多个方法:

```
OAuthWebSecurity.RegisterGoogleClient("displayName");
OAuthWebSecurity.RegisterYahooClient("displayName");
OAuthWebSecurity.RegisterFacebookClient("appId", "appSecret");
OAuthWebSecurity.RegisterMicrosoftClient("clientId", "clientSecret");
OAuthWebSecurity.RegisterTwitterClient("consumerKey", "consumerSecret");
```

3. ReCaptcha 辅助方法

ReCaptcha 辅助方法使用流行的 ReCaptcha 图片来帮助保护 Web 窗体，确保提交窗体的用户是人而不是自动系统。ReCaptcha 辅助方法放在 Web Page Helpers 库中，该库可以从 NuGet 下载。

在 _AppStart.cshtml 页面中包含 Microsoft.Web.Helpers，就可以使用 ReCaptcha。在该页面的代码块中，添加如下代码：

```
ReCaptcha.PublicKey = "your-public-key";
ReCaptcha.PrivateKey = "your-private-key";
```

在要使用 CAPTCHA 图片的页面中，可以给 HTML 标记添加如下代码：

```
@ReCaptcha.GetHtml()
```

在这个页面的代码块中，还需要添加如下代码：

```
if (ReCaptcha.Validate()) {
    // Do something because the ReCaptcha was validated
}
```

4. WebMail 辅助方法

WebMail 辅助方法允许以 HTML 格式发送电子邮件。与其他几个辅助方法不同，WebMail 辅助方法不显示标记，而用于简化发送电子邮件消息的过程。使用 WebMail 辅助方法的示例如程序清单 35-9 所示。

程序清单 35-9 WebMail 辅助方法的语法示例

```
@{
    // . . .
    // Set the common WebMail properties
    WebMail.SmtpServer = "SMTP Server";
    WebMail.SmtpPort = 25;
    WebMail.UserName = "Username";
    WebMail.Password = "Password";
    WebMail.From = "From Email Address";

    // Send the email
    WebMail.Send(to: "To Email Address",
                subject: "Subject",
                body: "<strong>Body</strong>");
    // . . .
}
```

5. WebCache 辅助方法

缓存是 ASP.NET 的重要部分，所以 Web Pages 也有缓存机制。WebCache 辅助方法就用于此，非常类似于在 ASP.NET Web 窗体中缓存数据项的方式。程序清单 35-10 演示了 WebCache 辅助方法。

程序清单 35-10 使用 WebCache 辅助方法缓存当前时间

```
@{  
    var toCache = WebCache.Get("cacheTimestamp");  
  
    if(toCache == null) {  
        toCache = DateTime.Now;  
        WebCache.Set("cacheTimestamp", toCache, 1, false);  
    }  
}
```

35.4.2 使用辅助方法添加功能

除了前面介绍的辅助方法之外，其他几个辅助方法也可以给 Web Pages 添加功能。比较流行的包括 WebImage、FileUpload、Video、Bing、Maps 和各种社交辅助方法。

1. WebImage 辅助方法

WebImage 辅助方法允许通过传递流或路径来操纵图片。WebImage 辅助方法有几种方式用于重置图片的大小，还可以用其他方式操纵图片。要获得从窗体传递来的图片，可以使用如下代码：

```
WebImage photo = WebImage.GetImageFromRequest();
```

2. FileUpload 辅助方法

FileUpload 辅助方法允许在 Web 页面上实现文件上传功能。该辅助方法有几个属性，可以根据值改变所显示的标记。例如，如果希望显示三个上传框，在访问者选择时显示更多的框，就可以把控件添加到页面中，如下所示：

```
@FileUpload.GetHtml(initialNumberOfFiles:3, allowMoreFilesToBeAdded:true,  
    includeFormTag:true, uploadText:"Upload")
```

3. Video 辅助方法

HTML5 越来越流行，所以 Video 辅助方法用得比较少。但许多 Web 应用程序仍使用 Windows Media、Flash 或 Silverlight 显示视频。Video 辅助方法允许根据指定的属性显示标记。例如，如果希望在 600×400 帧中使用 Silverlight 显示 Welcome 视频，就可以使用下面的代码：

```
@Video.Silverlight(  
    path: "Videos/Welcome.xap",  
    width: "600",  
    height: "400",  
    bgColor: "black",  
    autoUpgrade: true)
```

4. Bing 辅助方法

Web 应用程序的另一共同特性是包含搜索框，用于搜索站点和互联网。当然，为了使站点搜索功能有效，Web 应用程序必须能让公众访问，Bing 至少要索引一次站点。如果满足这两个条件，就可以在 _AppStart.cshtml 文件中添加如下代码，配置站点搜索功能：

```
Bing.SiteUrl = "your-site.com";
Bing.SiteTitle = "Search This Site";
```

接着，为了在页面上显示搜索框，在标记的需要位置添加如下代码：

```
@Bing.SearchBox()
```

5. Maps 辅助方法

Maps 辅助方法允许在 Bing、Google、MapQuest 或 Yahoo! 中显示地图。要在 Google 上显示华盛顿州雷德蒙德市雷德蒙德路 1 号的地图，应在标记的需要位置添加如下代码：

```
@Maps.GetGoogleHtml("1 Redmond Way, Redmond, WA", width: "600", height: "400")
```

6. 社交联网辅助方法

与社交媒体集成的功能对目前的 Web 应用程序非常重要。为了简化这个任务，人们添加了几个辅助方法来完成许多常见的任务，包括：

- Facebook
- Twitter
- LinkShare
- Gravatar
- GamerCard

Facebook 辅助方法目前只有一个目标：在 Web 页面上显示一个按钮，它可以很简单，使用 `@Facebook.LikeButton()` 即可实现，也可以很复杂，将用户定向到非当前页面。

```
@Facebook.LikeButton(href: "http://wrox.com", action: "recommend",
    buttonLayout: "button_count", showFaces: true, colorScheme: "dark")
```

Twitter 有两个方法，一个用于帮助链接到用户的 Twitter 配置，在标准的 Twitter 小窗口中显示结果。另一个用于获得 Twitter 搜索结果，把它们显示在标准的 Twitter 小窗口中。下面的代码可以使用这两个方法：

```
@Twitter.Profile("jgaylord")
@Twitter.Search("Razor Syntax")
```

LinkShare 辅助方法显示社交媒体图标，允许页面共享定制的页面标题，该辅助方法给 Delicious、Digg、Facebook、Reddit、StumbleUpon 和 Twitter 显示图标，用法示例如下：

```
@LinkShare.GetHtml("Social Sharing Example")
```

Gravatar 辅助方法连接 Gravatar 网站，根据用户名显示个人全球统一标识。个人全球统一标识是用户名的图片表示。它们常用在论坛、聊天室和评论窗体方面。例如，要显示作者 Jason Gaylord 的个人全球统一标识，可以使用：

```
@Gravatar.GetHtml("jason@jasongaylord.com")
```

GamerCard 辅助方法显示 Xbox 游戏卡的副本。游戏卡包含玩家图标、最近玩的游戏和所得分数。显示游戏卡的示例如下：

```
@GamerCard.GetHtml("Major Nelson")
```

35.4.3 创建定制的辅助方法

创建定制的辅助方法其实比想象得简单许多。可以使用本章前面开始的任务应用程序，或者创建新的 Web Pages 应用程序。首先给应用程序添加新文件夹 App_Code。

在 App_Code 文件夹中，创建新文件 AmazonBooks.cshtml。这个文件包含三个不同的辅助方法。该文件的名称非常重要，因为它将是辅助方法的名称。所以要确保使用名称 AmazonBooks，因为 AmazonBooks 辅助方法不存在。

在 Amazon 文件中，添加程序清单 35-11 中的代码。

程序清单 35-11 Web Pages 应用程序的 Amazon 辅助方法

```
@helper TextAndImage(string isbn) {
    var src = "http://rcm.amazon.com/e/cm?ltl=_blank&bcl=000000&" +
        "IS2=1&bgl=FFFFFF&fcl=000000&lcl=0000FF&o=1&p=8&l=as4&" +
        "m=amazon&f=ifr&ref=ss_til&asins=" + isbn;

    <iframe src=@src style="width:120px;height:240px;" scrolling="no"
        marginwidth="0" marginheight="0" frameborder="0"></iframe>
}

@helper TextOnly(string isbn, string title) {
    var src = "http://www.amazon.com/gp/product/" + isbn + "/" +
        "ref=as_li_ss_tl?ie=UTF8&linkCode=as2&camp=1789&" +
        "creative=390957&creativeASIN=" + isbn;
    var img = "http://www.assoc-amazon.com/e/ir?l=as2&o=1&a=" + @isbn;

    <a href=@src>@title</a><img src=@img width="1" height="1"
        border="0" alt="" style="border:none !important; margin:0px !important;" />
}

@helper ImageOnly(string isbn) {
    var src = "http://www.amazon.com/gp/product/" + isbn + "/" +
        "ref=as_li_ss_il?ie=UTF8&linkCode=as2&camp=1789&" +
        "creative=390957&creativeASIN=" + isbn;
    var img = "http://ws.assoc-amazon.com/widgets/q?_encoding=UTF8&" +
        "Format=_SL110_&ASIN=" + isbn + "&MarketPlace=US&" +
        "ID=AsinImage&WS=1&ServiceVersion=20070822";

    <a href=@src><img border="0" src=@img></a>
}
```

如程序清单 35-11 所示，AmazonBooks 中有三个辅助方法。每个方法都显示不同格式的图书列表和到 Amazon 网站上页面的链接。现在可以在任何页面中使用：

```
@AmazonBooks.ImageOnly("1118311825")
```

给这个例子传递的 ISBN 是本书的 10 位 ISBN 号。

35.5 本章小结

本章简介了 Razor 语法和 ASP.NET Web Pages。首先学习了 ASP.NET WebMatrix，复习了在 WebMatrix 中创建 Web 应用程序的不同方式，包括使用文件夹、模板或应用程序库中的应用程序。接着论述了如何在应用程序、文件和数据库的配置之间跳转。我们创建了一个 SQL Server CE 数据库，在该数据库中创建了一个数据表。

接着讨论了 Razor 语法的功能，学习了如何使用 Razor 连接数据源，如何与源中的数据交互操作。我们创建的 Web 页面改为使用布局页面。本章还使用 Razor 的内置辅助方法，包括：

- WebGrid
- ValidationSummary 和 ValidationMessage
- RenderBody 和 RenderSection
- WebSecurity、OAuthWebSecurity 和 ReCaptcha
- WebMail
- WebCache
- WebImage、FileUpload 和 Video
- Bing 和 Maps
- Facebook、Twitter、LinkShare 和 Gravatar

最后说明了如何以 Razor 为基础创建定制的辅助方法。尽管本章没有介绍 Razor、Web Pages 或 WebMatrix 的所有内容，但读者应能很好地理解 Razor 的工作方式，以及 ASP.NET Web Pages 使用 Razor 的方式。还要明白给 ASP.NET MVC 使用 Razor 视图引擎时，Razor 很有效。

第 X 部分

附 录

- 附录 A 迁移 ASP.NET 旧项目
- 附录 B COM 集成
- 附录 C ASP.NET 终极工具
- 附录 D 管理
- 附录 E 动态类型与语言
- 附录 F ASP.NET 联机资源
- 附录 G 使用 NuGet 扩展 Visual Studio

迁移 ASP.NET 旧项目

在一些情况下，需要从头开始建立 ASP.NET 4.5 应用程序。但在许多情况下，需要利用以前在 .NET Framework 1.0、1.1、2.0、3.5 或 4.0 平台上建立的 ASP.NET 应用程序，迁移它们，使它们能在 .NET Framework 4.5 平台上运行。

本附录介绍将 ASP.NET 1.x、2.0、3.5 或 4.0 应用程序迁移到 .NET Framework 4.5 的过程。

A.1 迁移过程并不困难

微软做了大量工作，以确保 ASP.NET 1.x 的迁移过程的简单性。在大多数情况下，不需要对应用程序进行任何修改。

将 ASP.NET 1.x、2.0、3.5 或 4.0 应用程序迁移到 4.5 版本时，不必将 ASP.NET 应用程序放在新服务器上，也不需要当前的服务器进行任何修改，而只需安装 .NET Framework 4.5。

在安装 .NET Framework 4.5 后，服务器上的 .NET Framework 版本就位于 C:\WINDOWS\Microsoft .NET\Framework 中，如图 A-1 所示。如果服务器是 64 位的，那么对应 64 位版本的目录是 C:\WINDOWS\Microsoft .NET\Framework64。

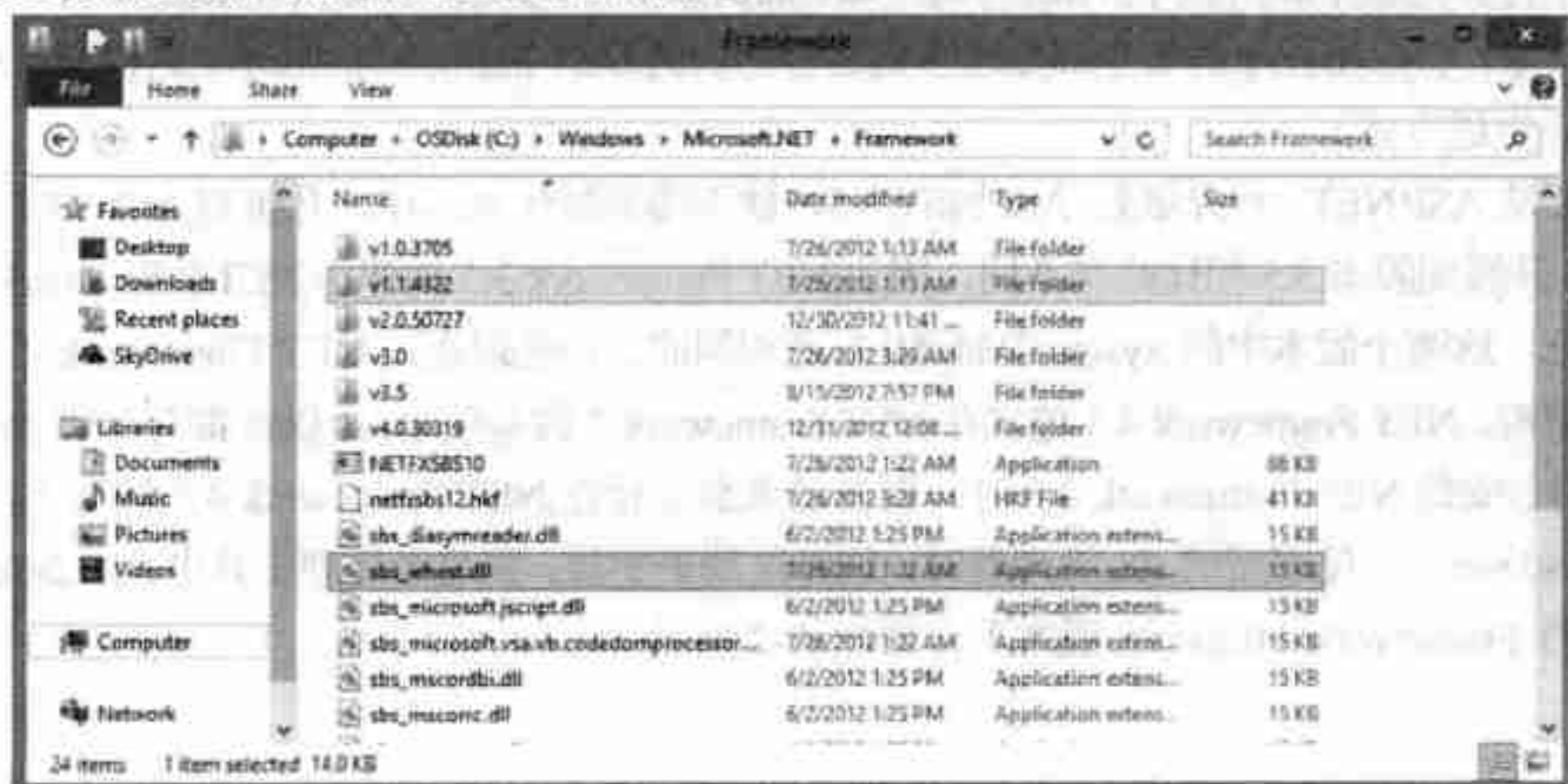


图 A-1

在图 A-1 中, 安装了 .NET Framework 所有官方版本: v1.0.3705、v1.1.4322、v2.0.50727、v3.0、v3.5、v4.0 和 v4.5。

A.1.1 同时运行 .NET Framework 的多个版本

从图 A-1 中可以看出, 可以同时运行 .NET Framework 的多个版本。ASP.NET 1.0、ASP.NET 1.1、ASP.NET 2.0、ASP.NET 3.5、ASP.NET 4 和 ASP.NET 4.5 应用程序可以在同一台服务器上运行。在同一台服务器上运行的不同版本的 ASP.NET 应用程序都有各自的工作者进程, 互不干扰。

A.1.2 就地升级

.NET Framework 4.5 是替代 .NET Framework 4 的就地升级。安装 .NET Framework 4.5 时, 安装程序会替换 .NET Framework 4 安装的已有文件, 把它们升级到 .NET Framework 4.5。图 A-1 中没有 v4.5 文件夹, 而以前的每个架构都会创建类似的文件夹。

如果 ASP.NET 4 应用程序运行在安装了 .NET Framework 4 的服务器上, 在安装 .NET Framework 4.5 后, 所有的 ASP.NET 4 应用程序都开始运行在 .NET Framework 4.5 架构下。安装 .NET Framework 4.5 时, 运行 ASP.NET 4 的所有应用程序池将运行 ASP.NET 4.5。

因为 .NET Framework 4.5 是就地升级, 所以微软做了大量工作, 以确保 .NET Framework 4.0 和 .NET Framework 4.5 之间的向后兼容性。这表示, 在安装 .NET Framework 4.5 后, ASP.NET 4 应用程序应能继续正常工作。

A.1.3 升级 ASP.NET 应用程序

在安装 .NET Framework 4.5 时, 并没有重新映射所有的 ASP.NET 应用程序, 从而使它们运行在新的架构实例中。我们需要选择性地重新映射应用程序, 使它们运行在 ASP.NET 4.5 架构下。



应该始终首先在开发环境或阶段环境中测试 ASP.NET 旧应用程序在 ASP.NET 新版本环境下的运行情况。不要在没有首先进行测试的情况下就将产品系统改为新版本。

如果不打算把整个应用程序升级到 ASP.NET 的新版本, 可以在应用程序的根虚拟目录下创建另一个虚拟目录, 将应用程序的某些部分迁移到运行它们的 .NET Framework 版本。这样就可以在升级过程中采取步进方式。

如果要从 ASP.NET 2.0 升级到 ASP.NET 3.5, 就不需要做什么工作。升级到 ASP.NET 4 与之前从版本 2.0 升级到版本 3.5 相比略有不同, 因为 .NET Framework 3.5 构建于 .NET Framework 2.0 基础之上。因此, 这两个版本中的 System.Web DLL 是相同的。但是现在, .NET Framework 4 是完全重新编译的架构。.NET Framework 4.5 建立在 .NET Framework 4 的基础上, 所以在带有 .NET Framework 4 的服务器上安装 .NET Framework 4.5 时, 服务器就会运行在 .NET Framework 4.5 环境下。

在 Windows 8 上使用新的 IIS 管理器时, 这种区别更明显。在这个管理工具中, DefaultAppPool 运行在 .NET Framework 4.0.xxxxx 版本下, 如图 A-2 所示。

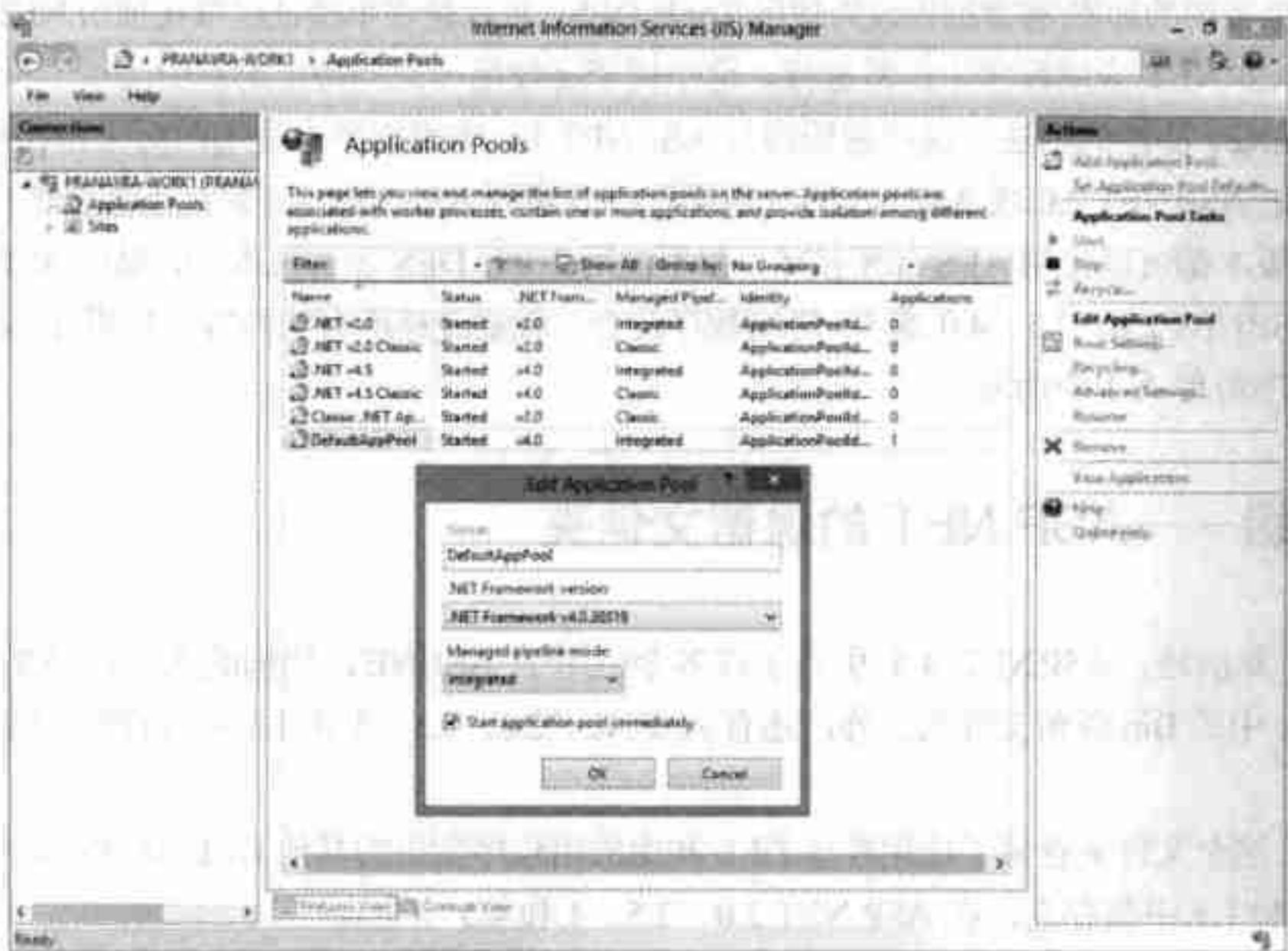


图 A-2

使用 Visual Studio 2012 把应用程序升级到 ASP.NET 4.5，会使 IDE 完成对应用程序配置文件的所有必要修改，详见本附录后面的内容。

A.2 混合版本——表单验证

如果 ASP.NET 应用程序使用了 .NET Framework 的多个版本，如前所述，就必须考虑表单验证在 ASP.NET 2.0、3.5、4 和 4.5 中的工作方式。

在 ASP.NET 1.x 中，表单验证过程使用 Triple DES(3DES)加密技术执行身份验证 cookie 的加密或解密过程。但是，自 ASP.NET 2.0 版本以来，已经改为使用 AES(Advanced Encryption Standard)加密技术。

AES 更快速，也更安全。但这两种加密技术是有区别的，因此必须改变 ASP.NET 4.5 生成这些密钥的方式。为此，可以在 ASP.NET 4.5 应用程序的 web.config 文件中修改 <machineKey> 部分，使其使用 Triple DES 加密，如程序清单 A-1 所示。

程序清单 A-1 将 ASP.NET 4.5 应用程序改为使用 Triple DES 加密

```
<configuration>
  <system.web>
    <machineKey validation="3DES" decryption="3DES"
      validationKey="1234567890123456789012345678901234567890"
      decryptionKey="1234567890123456789012345678901234567890" />
  </system.web>
</configuration>
```

把计算机密钥的加密/解密过程改为使用 Triple DES, 可以使表单验证过程在使用 .NET Framework 1.x 和 4 的 ASP.NET 应用程序中正常进行。这个例子还使用一系列特定的密钥演示了 `validationKey` 和 `decryptionKey` 特性的用法, 这些密钥应与 ASP.NET 1.x 应用程序中使用的对应密钥相同。

注意, 把 ASP.NET 2.0 或 ASP.NET 3.5 应用程序升级到 ASP.NET 4.5 时, 不需要进行这些修改, 因为这两个版本都可以使用 AES 加密技术, 都不使用 Triple DES 加密技术。如果将 ASP.NET 1.x 应用程序与 ASP.NET 2.0、3.5、4.0 或 4.5 应用程序混合, 就要迁移所有的内容以使用 Triple DES 加密技术, 如程序清单 A-1 中所示。

A.3 升级——ASP.NET 的保留文件夹

如第 3 章所述, ASP.NET 4.5 引入了许多专门用于 ASP.NET 架构的应用程序文件夹。除了 ASP.NET 1.x 中的 `Bin` 保留文件夹之外, 还有 ASP.NET 2.0、3.5、4 和 4.5 中所有的保留文件夹, 如下所示:

- **Bin:** 这个文件夹存储了应用程序 DLL 和由应用程序使用的其他 DLL。该文件夹在 ASP.NET 1.0 和 1.1 中都存在, 在 ASP.NET 2.0、3.5、4 和 4.5 中也存在。
- **App_Code:** 这个文件夹用于存储类、.wsdl 文件和类型化的数据集。存储在这个文件夹中的所有项都可自动用于解决方案中的所有页面。
- **App_Data:** 这个文件夹保存应用程序使用的数据存储。这是一处集中存储位置, 存储了应用程序可能使用的所有数据存储。App_Data 文件夹可以包含 SQL Express 文件(.mdf 文件)、Access 文件(.mdb 文件)、XML 文件等。
- **App_Themes:** 主题是一种为站点的所有页面提供统一外观和操作方式的方式。在实现主题时, 要使用 .skin 文件、CSS 文件和站点的服务器控件使用的图像。所有这些元素就构成了主题, 它们存储在解决方案的 App_Themes 文件夹中。
- **App_GlobalResources:** 这个文件夹可以存储资源文件, 如果应用程序需要根据区域性的改动来改变内容, 就可以把资源文件用作应用程序的数据字典。可以在 App_GlobalResources 文件夹中添加 Assembly Resource Files(.resx), 它们会动态编译, 并成为解决方案的一部分, 以供应用程序中的所有 .aspx 页面使用。
- **App_LocalResources:** 类似于 App_GlobalResources 文件夹, 这个文件夹可以集中存储由应用程序中的特定页面使用的资源。
- **App_WebReferences:** 这个文件夹可用于自动访问在应用程序中引用的远程 Web 服务。
- **App_Browsers:** 这个文件夹保存了 .browser 文件, .browser 文件是 XML 文件, 用于标识向应用程序发出请求的浏览器, 并说明这些浏览器具有的功能。

给文件夹的名称添加 App_前缀, 可确保不与 ASP.NET 1.x 应用程序中具有类似名称的文件夹相混淆。如果某个文件夹已经具有上述名称, 就应修改该文件夹的名称, 因为这些 ASP.NET 4.5 应用程序的文件夹名称是不能改变的。

A.4 ASP.NET 4.5 页面是 XHTML

在默认情况下, ASP.NET 4.5 建立的页面是与 HTML5 兼容的。在 Visual Studio 2012 IDE 中可以看到 HTML5 的设置, 如图 A-3 所示。

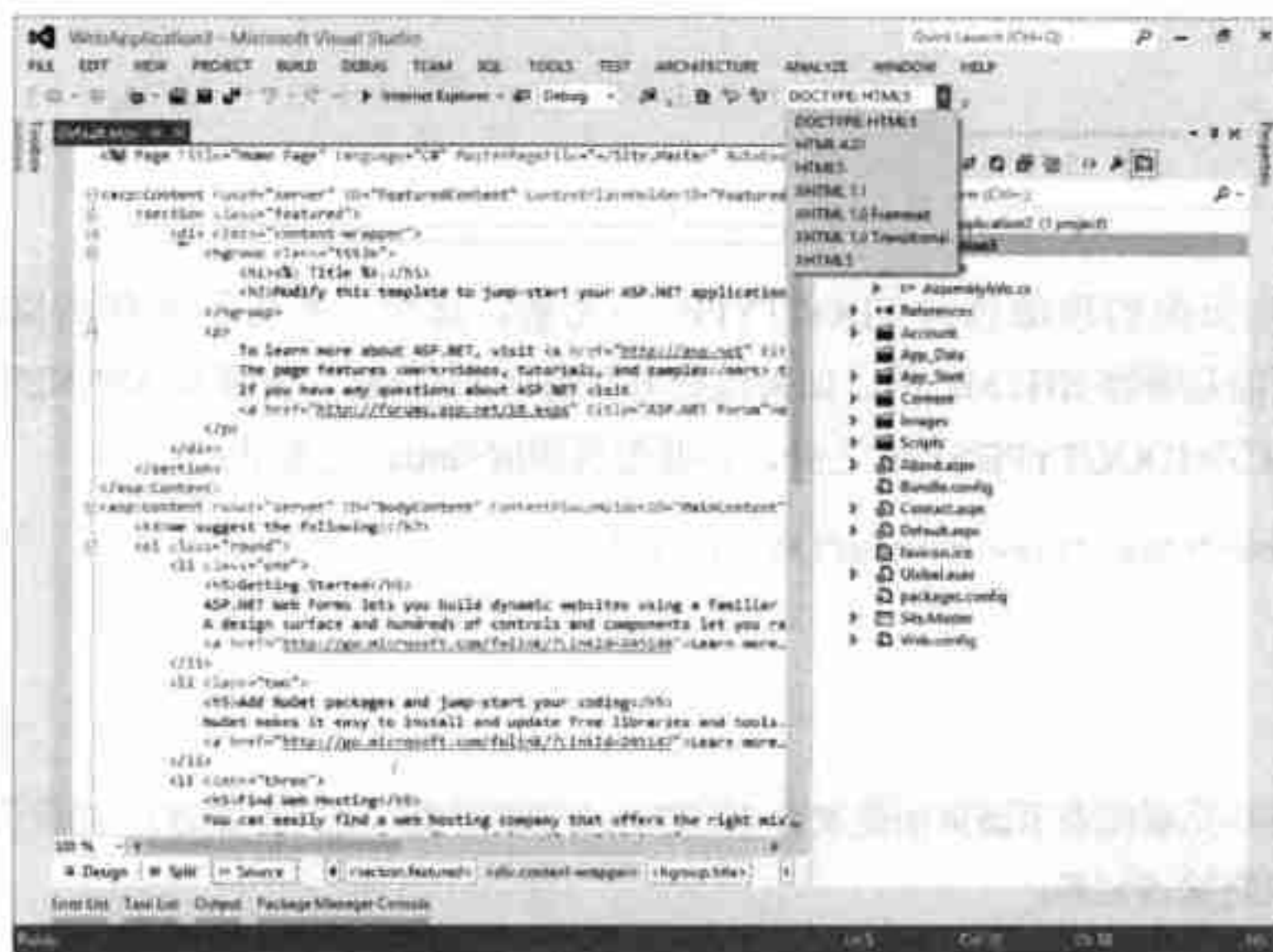


图 A-3

在图 A-4 中, 有一组选项用于确定 ASP.NET 应用程序如何输出页面的代码。默认设置为 HTML5。也可以修改 web.config 文件, 使输出不专用于 XHTML, 如程序清单 A-2 所示。

程序清单 A-2 修改 ASP.NET 4.5 应用程序的 XHTML 功能

```
<configuration>
  <system.web>
    <xhtmlConformance mode="Legacy" />
  </system.web>
</configuration>
```

把 mode 特性设置为 Legacy, 表示不使用 XHTML。相反, ASP.NET 4.5 将使用 ASP.NET 1.x 中使用的编码方式。

注意, 如果使用 AJAX, 那么把 Legacy 设置作为 mode 特性的值有时会使应用程序出现问题。其中一个问题是不进行部分页面的更新(这是 AJAX 的功能), 而是回送整个页面。这是因为页面不兼容 XHTML。解决方法是把 mode 特性设置为 Traditional 或 Strict, 使页面兼容 XHTML。

如果采用这种方法, 那么还必须对在 Visual Studio 2012 中创建的 ASP.NET 4.5 页面进行其他一些修改。在 Visual Studio 2012 中创建新的 ASP.NET 4.5 页面, 会得到如程序清单 A-3 所示的代码。

程序清单 A-3 典型的 ASP.NET 4.5 页面

```
<%@ Page Language="C#" %>
<!DOCTYPE HTML>
```



```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title></title>
</head>
<body>
  <form id="form1" runat="server">
    <div>

    </div>
  </form>
</body>
</html>
```

上述代码在页面的顶部包含<!DOCTYPE...>元素。这个元素告诉某些浏览器(如 Internet Explorer), 该页面是兼容 XHTML 的。如果该页面不兼容 XHTML, 就要从 ASP.NET 4.5 页面中删除这个元素。除了<!DOCTYPE>元素之外, 还要把页面的<html>元素从:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

改为:

```
<html>
```

原来的<html>元素也表示该页面是兼容 XHTML 的(即使实际上并不兼容)。如果页面与 XHTML 不兼容, 就必须删除该元素。

A.5 在 ASP.NET 4.5 中没有硬编码的.js 文件

ASP.NET 1.x 把一些必需的 JavaScript 文件提供为硬编码的.js 文件。例如, 在 ASP.NET 中, 验证服务器控件和智能导航功能都需要 JavaScript 才能工作。如果在 ASP.NET 1.x 应用程序中使用这些功能, ASP.NET 就可以查找已安装的.js 文件, 直接使用它们。

这些.js 文件在 C:\WINDOWS\Microsoft .NET\Framework\v1.1.4322\ASP.NETClientFiles 下。在这个文件夹中, 有 3 个.js 文件, 其中两个处理智能导航功能(SmartNav.js 和 SmartNavIE5.js), 另一个处理验证服务器控件(WebUIValidation.js)。因为它们都是硬编码的.js 文件, 所以可以打开它们, 修改其中的代码, 以更好地满足需要。在一些情况下, 开发人员就是如此操作的。

如果以任意方式修改这些 JavaScript 文件, 在将 ASP.NET 应用程序迁移到 ASP.NET 2.0、3.5、4 或 4.5 时, 就必须修改一些代码。ASP.NET 4.5 现在可以在 System.Web.dll 中动态包含.js 文件, 而不是在服务器上硬编码它们。在 ASP.NET 4.5 中, 通过处理程序 WebResource.axd 包含这些文件。

A.6 Visual Studio 2012 项目的兼容性

如前所述, 如果已经有 ASP.NET 1.x 应用程序, 那么只要在 IIS 中对应用程序池进行适当修改, 就可以在 ASP.NET 4 运行库中运行 ASP.NET 1.x 应用程序。使用 IIS 管理器或 MMC 管理单元, 可以从提供的下拉列表框中选择运行应用程序的合适架构。

ASP.NET 4.5 应用程序使用 Visual Studio 2012 IDE。如果仍然处理 ASP.NET 1.0 或 1.1 应用程序，就应分别保留计算机上安装的 Visual Studio .NET 2002 或 2003。安装 Visual Studio 2010 会得到 Visual Studio 的完整、全新副本，但并不升级以前的 Visual Studio .NET 2002 或 2003 IDE。Visual Studio 的所有副本可以同时运行。

如果要在 .NET Framework 4.5 环境下运行 ASP.NET 4.x 应用程序，并将应用程序的整个 ASP.NET 项目转换到 ASP.NET 4.5，可以使用 Visual Studio 2012 完成该转换过程。以这种方式转换项目后，就可以在 Visual Studio 2012 中建立并运行应用程序。现在应用程序就建立并运行在 ASP.NET 4 运行时环境下。

Visual Studio 2012 具有与 Visual Studio 2010 在解决方案和项目级别的兼容性。这表示，如果使用 Visual Studio 2010 创建 ASP.NET 应用程序，就可以在 Visual Studio 2012 中打开这个应用程序，进行修改，再在 Visual Studio 2010 中打开同一个应用程序。应用程序的所有修改都会保留。这是一项非常有用的功能，支持混合模式的开发，现在就可以使用 Visual Studio 2012 建立面向 ASP.NET 4.0 和 ASP.NET 4.5 的应用程序了。



在开发环境中，如果没有预先测试程序，就不要升级产品解决方案，以确保应用程序不受 .NET Framework 不同版本之间区别的影响。

A.7 从 ASP.NET 2.0/3.5/4.0 迁移到 ASP.NET 4.5

Visual Studio 2012 允许在多个架构中建立应用程序。例如，Visual Studio .NET 2002 仅允许建立 .NET Framework 1.0 应用程序。如果要建立 .NET Framework 1.1 应用程序，就需要安装并使用 Visual Studio .NET 2003。同样，Visual Studio .NET 2003 也不允许建立 .NET Framework 1.0 应用程序。如果要处理使用这两个架构的应用程序，就需要在计算机上安装两个 IDE。

在 Visual Studio 2012 中创建新项目时，可以使项目面向如下架构：

- .NET Framework 2.0
- .NET Framework 3.0
- .NET Framework 3.5
- .NET Framework 4
- .NET Framework 4.5

如果打开建立在 .NET Framework 2.0 基础之上的 ASP.NET 应用程序，就可以从 IDE 中很容易地把应用程序重新设定到较新的架构版本。在 Solution Explorer 中右击项目，从弹出菜单中选择 Property Pages 命令，会打开一个对话框，从中可以改变应用程序的目标架构。在本例中，可以看到 Windows 8 计算机上的默认选项，如图 A-4 所示。



图 A-4

根据图 A-4 所示改变目标架构，注意 Visual Studio 2012 需要关闭并重新打开解决方案。完成该操作后，会看到由于使用了较新版本的架构，因此甚至改变了 web.config 文件。还需要解决在处理 ASP.NET 不同版本间存在的显著改动时引发的问题，但是，通过在 Visual Studio 中构建解决方案，可以获得这些问题的快速列表。

A.8 小结

Visual Studio 2012 IDE 的优点在于，可以只升级 ASP.NET 解决方案，而不需要升级解决方案的目标架构版本。但是，在把 ASP.NET 解决方案升级到 .NET Framework 4.5 的过程中，使用 Visual Studio 可以简单地完成这项任务。

该附录讨论了使用 IDE 升级的有关内容以及不同发布版本之间的一些重要改动，而进行这些改动的目的则是为了使迁移尽可能简单。

附录 B

COM 集成

编程的最佳实践方式是把应用程序分隔为可工作的、单独的组件——也称为业务对象。这使应用程序的管理方便得多，且能达到代码重用的目的，因为可以在同一应用程序的不同部分或完全独立的应用程序之间共享这些组件。

在引入.NET之前，许多应用程序都使用COM作为编写业务对象的方式。如果把旧应用程序或其中的一部分迁移到ASP.NET环境，就需要利用各种COM对象。本附录介绍如何在ASP.NET页面和代码中使用.NET和COM组件。

本附录概述COM组件如何在ASP.NET中使用。

B.1 COM Interop: 在.NET中使用COM

最初推出.NET时，微软就知道，如果自己的开发团队不能使用多年来已建立、维护和改进的数千个COM控件，就会让人非常失望。

为此，微软提供了COM Interoperability，简称为COM Interop，该技术允许.NET用.NET组件的接口封装COM对象的功能，这样.NET代码就可以与COM对象通信，而无须在代码中使用COM技术和接口。

图B-1是Runtime Callable Wrapper (RCW)，这是在.NET代码和COM组件之间定向消息流的中间组件。

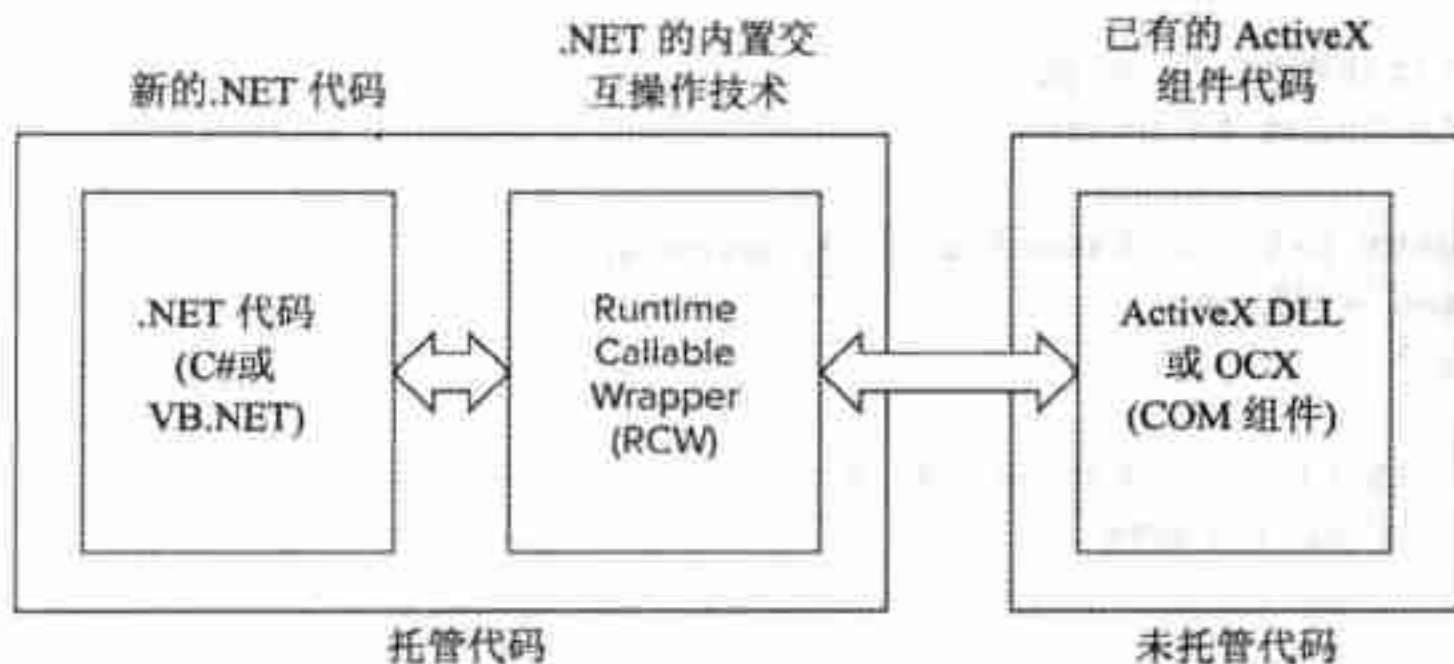


图 B-1

B.1.1 Runtime Callable Wrapper

Runtime Callable Wrapper (RCW)是允许在.NET 和 COM 之间交互操作的一段代码。在项目中给每个 COM 组件创建 RCW。为此可以使用 Visual Studio 2012。

为了在项目的 References 部分添加 ActiveX DLL, 可以选择 Website | Add Reference, 或者在 Solution Explorer 中右击项目的根节点, 再选择 Add Reference 菜单项。

Interop 库是从 ActiveX DLL 中自动创建的, 以便让 Visual Studio 2012 使用。Interop 库是为 ActiveX 控件定制的 RCW 组件, 如图 B-1 所示。Interop 文件名是 Interop.OriginalName.DLL。

还可以手工创建 RCW 文件, 而不是让 Visual Studio 2012 创建。在 .NET Framework 中, 有一种方法可以通过命令行工具 Type Library Importer 为控件手工创建 RCW Interop 文件。使用 tlbimp.exe 可执行文件可以调用 Type Library Importer。

例如, 要为 SQLDMO 对象创建 Interop 库, 可以在开始菜单中启动 Visual Studio 2012 | Visual Studio Tools 组中的 Visual Studio 2012 Command Prompt, 在命令行提示符中, 输入如下命令:

```
tlbimp sqldmo.dll /out:sqldmoex.dll
```

在这个例子中, /out:参数指定要创建的 RCW Interop 库的名称。如果忽略这个参数, Visual Studio 就会自动生成名称。

使用 Type Library Importer 的场合是: 不使用 Visual Studio 2012 作为开发环境, 或者希望更多地控制自动创建的程序集, 或者自动完成连接 COM 组件的过程。

Type Library Importer 是封装器, 其中封装了 System.Runtime.InteropServices 名称空间中的 TypeLibConverter 类。

B.1.2 在 ASP.NET 代码中使用 COM 对象

为了继续完成其他一些示例, 下面是一个在 ASP.NET 页面中使用 COM 对象的简单例子, 该 COM 对象用 Visual Basic 6 编写。

首先, 创建一个 ActiveX DLL, 用于后面的例子。把程序清单 B-1 中的 Visual Basic 6 代码添加到 NameFunctionsClass 类中, 编译为名为 NameComponent.dll 的 ActiveX DLL。

程序清单 B-1 用于 ActiveX DLL—— NameComponent.DLL 的 VB 代码

```
Option Explicit

Private m_sFirstName As String
Private m_sLastName As String

Public Property Let FirstName(Value As String)
    m_sFirstName = Value
End Property

Public Property Get FirstName() As String
    FirstName = m_sFirstName
End Property

Public Property Let LastName(Value As String)
    m_sLastName = Value
```



```

End Property

Public Property Get LastName() As String
    LastName = m_sLastName
End Property

Public Property Let FullName(Value As String)
    m_sFirstName = Split(Value, " ")(0)
    If (InStr(Value, " ") > 0) Then
        m_sLastName = Split(Value, " ")(1)
    Else
        m_sLastName = ""
    End If
End Property

Public Property Get FullName() As String
    FullName = m_sFirstName + " " + m_sLastName
End Property

Public Property Get FullNameLength() As Long
    FullNameLength = Len(Me.FullName)
End Property

```

创建完要在 ASP.NET 页面中使用的 ActiveX DLL, 下一步就是使用 Visual Studio 2012 创建新的 ASP.NET 项目。用程序清单 B-2 中的 HTML 代码替代 Default.aspx 文件中的 HTML 代码。这些代码给 HTML 页面添加了许多文本框和标签, 还通过 C# 代码添加了功能。

程序清单 B-2 使用 NameComponent.dll

```

<%@ Page Language="C#" %>

<script runat="server">
    protected void AnalyzeName_Click(object sender, System.EventArgs e)
    {
        NameComponent.NameFunctionsClass Name =
            new NameComponent.NameFunctionsClass();

        if(FirstName.Text.Length > 0)
        {
            string firstName = FirstName.Text.ToString();
            Name.set_FirstName(ref firstName);
        }

        if(LastName.Text.Length > 0)
        {
            string lastName = LastName.Text.ToString();
            Name.set_LastName(ref lastName);
        }

        if(FullName.Text.Length > 0)
        {
            string fullName = FullName.Text.ToString();
            Name.set_FullName(ref fullName);
        }
    }

```



```

    }

    FirstName.Text = Name.get_FirstName();
    LastName.Text = Name.get_LastName();
    FullName.Text = Name.get_FullName();
    FullNameLength.Text = Name.FullNameLength.ToString();
}
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
  <head runat="server">
    <title>Using COM Components</title>
  </head>
  <body>
    <form id="form1" runat="server">
      <p>
        <asp:Label ID="Label1" runat="server">First Name:</asp:Label>
        &nbsp;
        <asp:TextBox ID="FirstName" runat="server"></asp:TextBox>
      </p>
      <p>
        <asp:Label ID="Label2" runat="server">Last Name:</asp:Label>
        &nbsp;
        <asp:TextBox ID="LastName" runat="server"></asp:TextBox>
      </p>
      <p>
        <asp:Label ID="Label3" runat="server">Full Name:</asp:Label>
        &nbsp;
        <asp:TextBox ID="FullName" runat="server"></asp:TextBox>
      </p>
      <p>
        <asp:Label ID="Label4" runat="server">Full Name Length:</asp:Label>
        &nbsp;
        <asp:Label ID="FullNameLength" runat="server"
          Font-Bold="True">0</asp:Label>
      </p>
      <p>
        <asp:Button ID="AnalyzeName" runat="server"
          OnClick="AnalyzeName_Click" Text="Analyze Name"></asp:Button>
      </p>
    </form>
  </body>
</html>

```

接下来为上一步创建的 ActiveX DLL 添加引用。为此，执行下面的步骤：

- (1) 在 Solution Explorer 对话框中右击项目。
- (2) 选择 Add Reference 菜单项。
- (3) 在 Add Reference 对话框中，选择 Browse 选项卡。
- (4) 浏览并找到 NameComponent.dll 对象。
- (5) 单击 OK，把 NameComponent.dll 添加到所选组件的列表中，关闭对话框。



即使不使用 Visual Studio 2012 或代码隐藏页面，也仍可以添加对 COM 控件的引用，方法是使用 Type Library Converter 手工创建 RCW，再在页面中添加 using 语句。

使用 Add Reference 对话框选择组件后，就为组件创建了 RCW 文件，并添加到应用程序中。这就完成了！运行应用程序，就会看到 COM Interop 层。

单击 Analyze Name 按钮，First Name、Last Name 和 Full Name 文本框中的字段就会发送给 RCW，而 RCW 会传递给 ActiveX 组件 NameComponent.DLL。数据用相同的方式检索，重新填充到文本框中，并指出全名的长度。

1. 在 C# 中访问 COM 成员

有时，COM 对象的成员没有正确显示给 C#。在前面的例子中，String 属性就没有显示出来，但 Long 属性(FullNameLength)显示出来了。

这是问题，因为尽管可以看到属性，但不能编译应用程序。例如，不使用程序清单 B-2 中的 C# 代码，而使用下面的代码设置 ActiveX 组件 NameComponent.dll 的 FirstName 属性：

```
if(FirstName.Text.Length > 0)
    Name.FirstName = FirstName.Text.ToString();
```

尝试编译这些代码，会得到如下错误：

```
c:\inetpub\wwwroot\wrox\Default.aspx.cs(67): Property, indexer, or event
'FirstName' is not supported by the language; try directly calling accessor methods
'NameComponent.NameFunctionsClass.get_FirstName()' or
'NameComponent.NameFunctionsClass.set_FirstName(ref string)'
```

FirstName 属性似乎正确，显示在 IntelliSense 中，但不能使用。因此，必须使用 set_FirstName(get_FirstName 用于读取属性)。这些方法没有显示在 IntelliSense 中，但它们肯定是存在的。

而且，这些方法需要 ref string 参数，而不是 String。在程序清单 B-2 中，使用两步可以正确显示属性。首先，把 String 分配给本地变量，接着使用 ref 把本地变量传递为方法。

2. 手工释放 COM 对象

.NET 的优点之一在于垃圾回收——可以在使用完进行清理。但使用 COM Interop 时，却不总是这样。因为 COM 对象没有 .NET 依赖的内置垃圾回收机制，所以 .NET 不知道何时从内存中释放 COM 对象。

因为这个限制，所以应使用 System.Runtime.InteropServices.Marshal 类的 ReleaseComObject 类，尽快从内存中释放 COM 对象：

```
System.Runtime.InteropServices.Marshal.ReleaseComObject(Object);
```

注意，如果尝试在 COM 对象超出作用域之后再次使用，就会抛出异常。

B.2 错误处理

.NET 中的错误处理使用异常，而不是 Visual Basic 6 应用程序使用的 HRESULT 值。幸好，RCW 做了许多工作，能转换它们。

以程序清单 B-3 中的代码为例。在这个例子中，如果分子或分母大于 1000，就抛出用户定义的错误。注意不捕获除 0 错误，还要注意 ActiveX 组件抛出错误时会发生什么。

这个例子先把程序清单 B-3 中的代码编译为 ActiveX 组件 DivideComponent.dll 的类 DivideClass。

程序清单 B-3 在 VB 中抛出错误

```
Public Function DivideNumber(Numerator As Double, _
                             Denominator As Double) As Double

    If ((Numerator > 1000) Or (Denominator > 1000)) Then
        Err.Raise vbObjectError + 1, _
            "DivideComponent:Divide.DivideNumber", _
            "Numerator and denominator both have to " + _
            "be less than or equal to 1000."

    End If

    DivideNumber = Numerator / Denominator

End Function
```

接着，创建一个新的 ASP.NET 项目，添加对 DivideComponent.dll 的引用(调用 Visual Studio 2012，创建 RCW 的副本)。注意也可以使用 `tlbimp` 可执行文件手工创建该引用。

现在给 ASP.NET 页面添加程序清单 B-4 中的代码。

程序清单 B-4 .NET 中的错误处理

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Calculate_Click(object sender, System.EventArgs e)
    {

        DivideComponent.DivideClass myDivide = new DivideComponent.DivideClass();

        try
        {
            double numerator = double.Parse(Numerator.Text);
            double denominator = double.Parse(Denominator.Text);
            Answer.Text = myDivide.DivideNumber(ref numerator,
                ref denominator).ToString();
        }

        catch (Exception ex)
        {
            Answer.Text = ex.Message.ToString();
        }
    }
}
```



```

    }

    System.Runtime.InteropServices.Marshal.ReleaseComObject(myDivide);

}
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
    <title>Using COM Components</title>
  </head>
  <body>
    <form id="form1" runat="server">
      <p>
        <asp:Label ID="Label1" runat="server">Numerator:</asp:Label>
        &nbsp;
        <asp:TextBox ID="Numerator" runat="server"></asp:TextBox>
      </p>
      <p>
        <asp:Label ID="Label2" runat="server">Denominator:</asp:Label>
        &nbsp;
        <asp:TextBox ID="Denominator" runat="server"></asp:TextBox>
      </p>
      <p>
        <asp:Label ID="Label3" runat="server">
          Numerator divided by Denominator:</asp:Label>
          &nbsp;
          <asp:Label ID="Answer" runat="server" Font-Bold="True">0</asp:Label>
        </p>
      <p>
        <asp:Button ID="Calculate"
          runat="server"
          OnClick="Calculate_Click"
          Text="Calculate">
        </asp:Button>
      </p>
    </form>
  </body>
</html>

```

程序清单 B-4 中的代码把用户给 Numerator 和 Denominator 输入的值传递给 ActiveX 组件 DivideComponent.dll，使它们相除。用无效的数据运行应用程序，会得到程序清单 B-4 所示的异常消息。

根据用于运行 ASP.NET 应用程序的语言，不同的数据集会显示不同的值。对于有效的输入，当然总是会看到正确的结果，而对于超过 1000 的输入，会看到 Visual Basic 6 指定的错误描述“Numerator and denominator both have to be less than or equal to 1000”（分子和分母都必须小于等于 1000）。

但是，对于无效的字符串，Visual Basic 2012 会报告 Cast from string "abc" to type 'Double' is not valid（字符串"abc"转换为'Double'类型是无效的），而 C#报告 Input string was not in a correct format（输入字符串的格式不正确）。对于除 0 错误，它们都会报告 Divide by Zero，因为错误直接来自 Visual Basic 6

运行时。

B.3 用.NET 应用程序部署 COM 组件

用.NET 应用程序部署 COM 组件非常简单,尤其是在与部署 ActiveX 控件相比较时。用 COM 组件部署.NET 应用程序有两种情形:

- 使用私有程序集
- 使用共享或公共程序集

B.3.1 私有程序集

把 ActiveX 组件的全部或部分本地安装到.NET 应用程序中,就是在安装私有程序集。在这种情形下,.NET 应用程序在同一台计算机上的每次安装,至少是给要引用的 ActiveX 组件安装 Interop 库的副本,如图 B-2 所示。

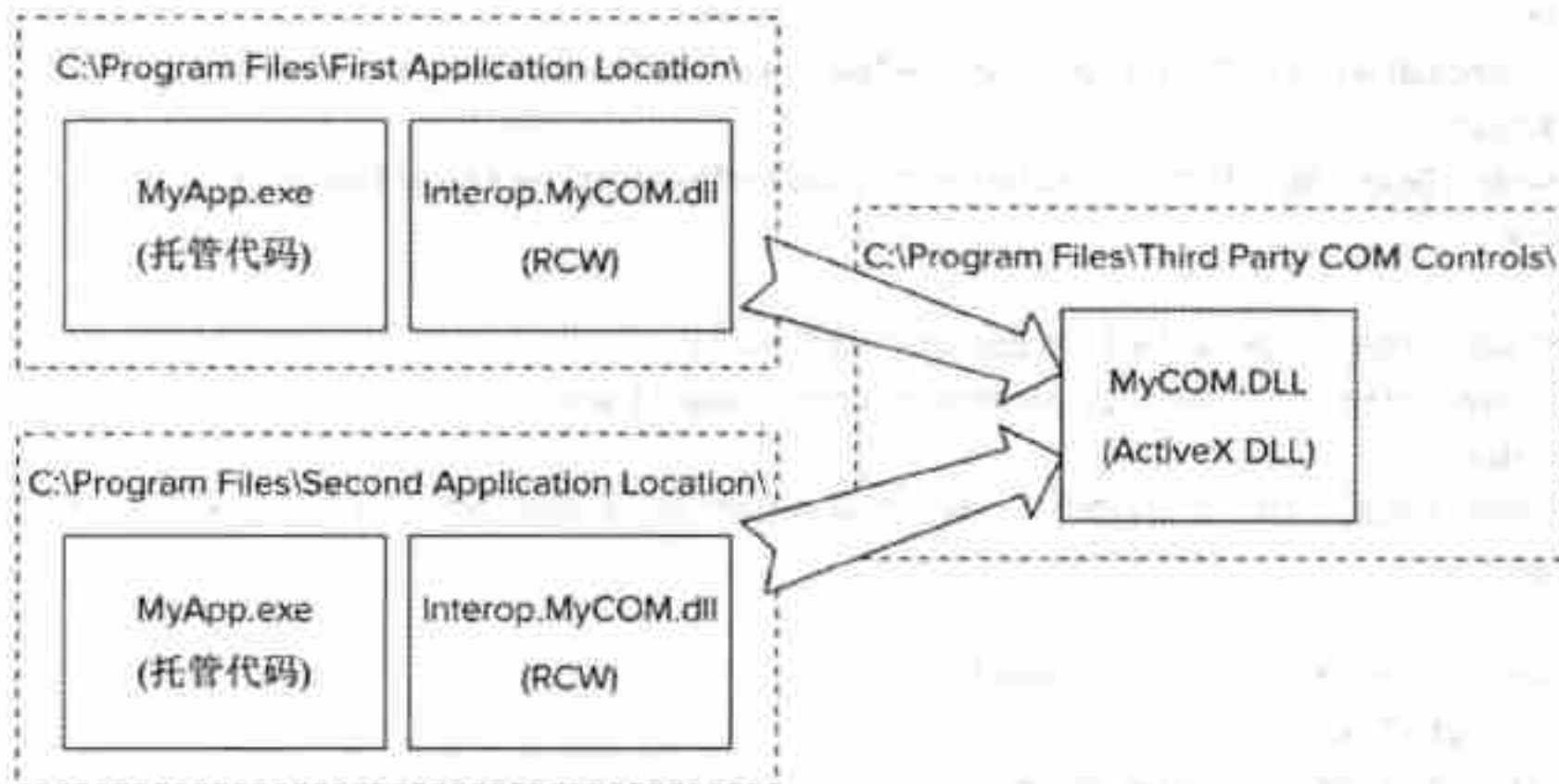


图 B-2

是把 ActiveX 组件本地安装到应用程序中,还是把它们安装到所有调用应用程序的共享目录下,取决于开发人员。



把 ActiveX 组件单独放在某个目录下一度被认为是正确的实践方式,这样如果这些组件再次由其他应用程序引用,就无须再次注册或安装文件了。使用这种方式意味着,升级组件时,会自动升级使用这个组件的所有应用程序。但这种实践方式并不好。实际上,这是导致 DLL 灾难的主要因素,也是微软开始改进安装私有.NET 组件程序集的过程的主要原因。

组件就位后,就只剩下一个任务:使用 `regsvr32` 注册 ActiveX 组件,就像部署支持 ActiveX 的应用程序那样。

B.3.2 公共程序集

私有程序集的反面就是公共程序集。公共程序集共享其他应用程序的 RCW Interop DLL。要创建公共程序集，必须把 RCW 文件放在 *Global Assembly Cache (GAC)* 中，如图 B-3 所示。

GAC 在 C:\Windows\assembly 下。在 GAC 中安装数据项很简单：通过 Windows 资源管理器把该项拖放到这个文件夹中。尽管 GAC 对每个人开放，但不推荐盲目地把组件安装到这里，除非有很好的理由。

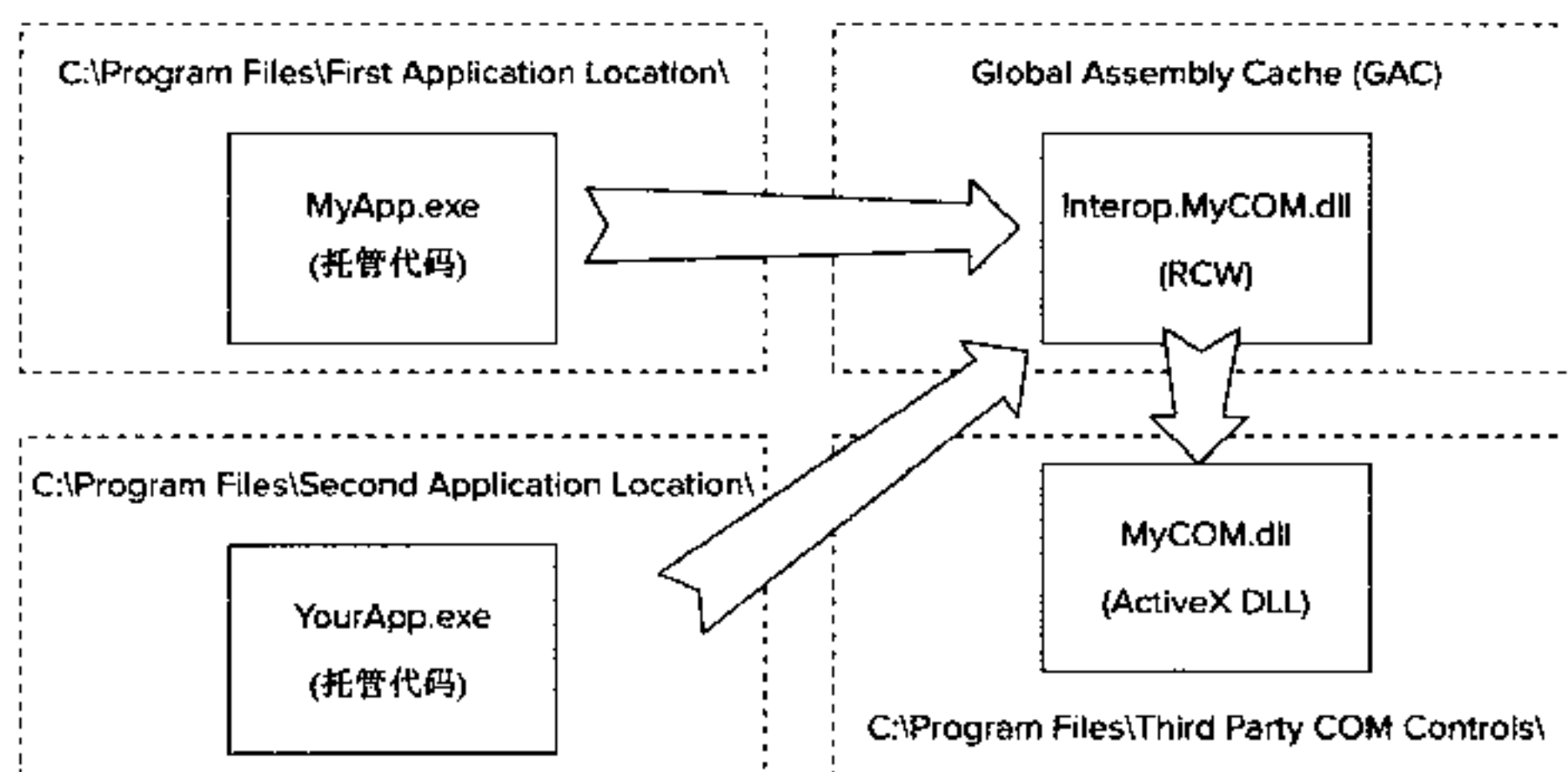


图 B-3

也可以在命令行中使用 Global Assembly Cache Tool (Gacutil.exe) 把数据项添加到 GAC 中。允许查看和操纵 GAC 的内容，下载缓存。GAC 的 Explorer 视图提供了类似的功能，也可以在构建脚本、makefile 文件和批处理文件中使用 Gacutil.exe。

很难找到把程序集安装到 GAC 的好理由。如果选择这么做，理由就是 ActiveX 组件得到了高度共享，许多 .NET 应用程序都在同一台计算机上使用它们。在协作环境下，把服务器上许多应用程序都使用的已有业务逻辑从 ActiveX 升级到 .NET 时，可能出现这种情形。在商业设置下，应避免使用 GAC。

B.4 小结

本附录学习了 COM Interop 如何访问已有的 COM 组件，以便在 ASP.NET 中使用 .NET 和 COM 组件。

ASP.NET 终极工具

我总是相信能使用好自己的工具。我花费好几年的时间在 Internet 上收集出色的工具，以提高效率。Internet 上有上千种工具，许多工具的功能都是相互重叠的。一些工具能非常好地完成某项任务，而其他一些工具就好像瑞士军刀，在它们的小工具栏中有数十个方便的工具。本附录是我一直在使用的 ASP.NET 工具列表。在开发基于 ASP.NET 的 Web 站点时，我一直在使用这些工具。如果读者觉得它们还不错，就不妨试试它们。许多工具都是免费的，但一些工具不免费。在我看来，每个工具都至少值得一试，许多工具都值得购买，因为它们可以节省你宝贵的时间。

C.1 简化调试

“在计算机的历史中，调试阶段从来不会很短。”

—— Steven Levy

C.1.1 Firebug

这个应用程序包括许多值得介绍的内容，可以使用一本书来介绍。Firebug 实际上是 Firefox 插件，因此需要下载并安装 Firefox 才能使用。

图 C-1 显示了 Firebug 分析下载 Web 页面时所需的所有网络流量。该工具显示了一张非常详细的图表，说明何时下载每个部分，从下载第一个字节到下载最后一个字节需要多长时间。

该工具有非常丰富的有趣功能，可以查看 HTML，深入分析 CSS，包括一些较复杂 CSS 技术的可视化，例如偏移、页边距、边框和内边距。Firebug 还包含强大的 JavaScript 调试器，可以在 Firefox 中调试 JavaScript。更有趣的是它的 JavaScript 探查器以及一个非常详细的错误处理程序，该处理程序有助于找出甚至是最模糊的错误。

最后，Firebug 包含交互式的控制台功能，类似于 Visual Studio 的 Immediate 窗口，可以随时执行 JavaScript 以及控制台调试，控制台调试功能可以进行经典的“到达此处”(got here)调试。Firebug 对 Web 开发人员而言是不可或缺的，强烈推荐使用。



图 C-1

还有采用 JavaScript 文件形式的 Firebug Lite 工具。可以把它添加到页面上, 在这些页面中, 需要控制台调试器在 Internet Explorer、Opera 或 Safari 中工作。通过这个文件可以使用 Firebug JavaScript 的 console.log 方法进行“到达此处”(got here)调试。

C.1.2 YSlow

YSlow 是插件的插件，由 Yahoo! 提供。YSlow 扩展了 Firebug，使用 Yahoo! 关于快速 Web 站点的 13 条规则分析 Web 页面。在图 C-2 中，Yahoo! 的 YSlow 正在分析 Web 站点。

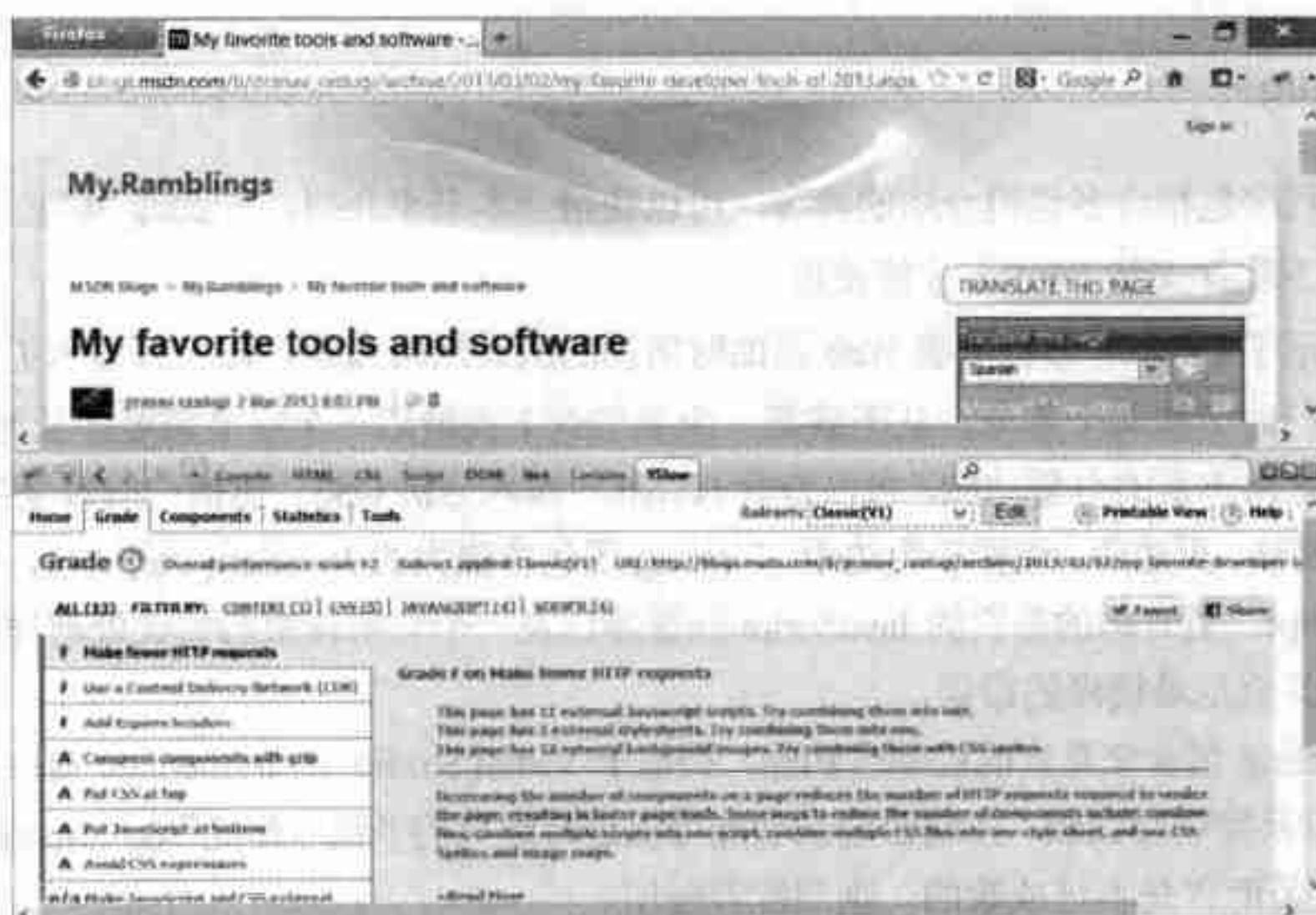


图 C-2

YSlow 是免费工具,也是非常优秀的资源,有助于我们清晰地理解客户的浏览器必须完成哪些工作才能查看 Web 站点。

C.1.3 IE10 Developer Tools

IE8 Developer Tools 由微软提供,内置于 IE10 中,既是免费的,也是 Web 开发所必需的工具栏。只需按下 F12 功能键即可激活。该工具扩展了 IE10 的功能,提供 DOM 检查、JavaScript 探查以及元素大纲视图等功能。甚至可以可视化框模型,如图 C-3 所示。



图 C-3

ASP.NET 开发人员现在需要使他们的站点看上去很美观,此时就可以使用这个工具栏。在此,我强烈推荐这个工具栏。

C.1.4 jQuery 和 jQuery UI

虽然不是明确要使用的“工具”,但是使用 JavaScript 库 jQuery 和 jQuery UI 来完成复杂的 JavaScript 任务已成为一种乐趣。尽管在 jQuery 出现之前 JavaScript 库就已经存在,但是 jQuery 不仅改变了 Web,而且使 JavaScript 更有趣。

jQuery 包括简洁的选择器引擎,使操作 HTML DOM(Document Object Model)变得非常简单。使用该引擎还可以选择和过滤节点,并能很容易地对这些节点应用事件和动画。

jQuery 还包括能够方便地进行 AJAX 调用的方法。ASP.NET MVC 小组决定给 ASP.NET MVC 配备 jQuery 这个优秀的库,从而使其成为 .NET 中第一个能够完全支持 jQuery 的开源产品。Visual Studio 2012 IDE 中 IntelliSense 的改进意味着可以方便地使用 jQuery。

jQuery UI 是附加的库,在 jQuery 的基础上添加了更多的动画支持,更为重要的是,为可设定主题的高级窗口部件(例如滑块、日历等)添加了基本结构。关于 jQuery 和 jQuery UI 的更多信息,可以查看 <http://jquery.com>,如图 C-4 所示。

如果还没有这么做,那么可以考虑把 ASP.NET 应用程序的探查功能添加到软件开发生命周期中。很少有开发人员会正式分析和探查他们的应用程序。花费一些时间探查以前从来没有探查过的应用程序,你会惊讶地发现使用 ANTS 或 dotTrace 等探查器可以非常快速地完成探查工作。

C.2 参考资料

“把书借给别人的人是傻瓜,还书的人比傻瓜还傻。”

——阿拉伯谚语

C.2.1 QuirksMode.org 和 HTMLDog.com

当创建需要在所有浏览器中都很美观的 Web 站点时,要了解常用浏览器中的错误、“功能”和一般区别。Web 页面由许多标准(HTML、CSS、JS)组成。这些标准不仅具有开放性的解释,它们的实现方式也有微妙的区别,尤其是在它们交互时,这种区别更明显。

参考 Web 站点,如 QuirksMode,其中收集了数百个问题和解决办法,还对它们进行了分类,以便于使用。许多功能都没有专门设计,但很容易找到或无意中发现。

HTMLDog 是 Web 设计人员非常喜欢的 HTML 和 CSS 资源。这个网站上包含许多教程、文章和专门针对 XHTML 的大量参考信息。QuirksMode 有许多用于学习 JavaScript 和 CSS 的资源,包括许多测试题和演示突发事件的页面。

C.2.2 Visibone

Visibone 因为漂亮的参考卡片和陈列颜色、字体、HTML、JavaScript 和 CSS 的图表而著名。Visibone 参考卡片和手册可以在线获得,价格很合理。最有价值的是 Browser Book,可以从 www.visibone.com/products/browserbook.html 获得,这里推荐购买压缩版。一定要把自己的姓名写在上面,因为你的同事会把它弄丢。

C.2.3 www.asp.net

www.asp.net 是学习 ASP.NET 及其各种技术的庞大资源。图 C-6 显示了这个站点,其中的链接可用于访问我的博客和其他团队资源。



图 C-6

C.2.4 www.webdevchecklist.com

www.webdevchecklist.com 详细列出了可运行的工具，确保网站遵循 Web 开发人员的最佳实践方式。

C.2.5 SlowCheetah

这是 Visual Studio 扩展，可以从 <http://visualstudiogallery.msdn.microsoft.com/69023d00-a4f9-4a34-a6cd-7e854ba318b5> 下载。使用这个工具可以根据构建配置，通过不同的转换文件来转换 app.config 或任意文件。例如，在调试和运行应用程序时，可以有不同的数据库和应用程序设置。

C.3 整理代码

“每次战争后，总是会有人打扫战场。”

——Wisława Szymborska

C.3.1 来自 DevExpress 的 Refactor! for ASP.NET

Visual Studio 2012 中的重构支持在持续改进。第三方实用程序继续给 IDE 增强新的功能，例如，Refactor! for ASP.NET 就给 ASP.NET 源代码视图添加了重构功能。

Refactor! for ASP.NET 可以从 www.devexpress.com/Products/NET/IDETools/RefactorASP/ 免费下载，其中包含的重构功能更便于简化代码和 ASP.NET 标记。

C.3.2 Ajax Minifier——JavaScript 最小化工具

在创建 ASP.NET Web 站点时，常常要创建定制的 JavaScript 文件。在开发过程中，需要给这些文件添加注释，以便于阅读。但在生产过程中，需要计算每个字节，因此最好使用 JavaScript 最小化工具减小 JavaScript 文件的大小。

Ajax Minifier 是一款 C# 应用程序，提供了 JavaScript 的压缩功能，或者通过去除注释和空白简单地“缩小文件”。在 CodePlex 上，该应用程序已经发布到 ASP.NET 项目中 (<http://aspnet.codeplex.com>)。

这些技术工作得很好。例如，Steve Kallestad 曾经报告过，在进行专门的 JavaScript 压缩之前，JavaScript 库 Prototype 1.50 的副本有 70KB，压缩后就变成 30KB，而在应用了 gzip HTTP 压缩后，则变成 14KB。从 70KB 到 14KB 是非常显著的空间节省。

专门的 JavaScript 压缩会把变量重命名为单个字母，考虑全局变量和本地变量的重命名，以及去除不必要的空白和注释。

Ajax Minifier 包含用于命令行和 MSBuild 项目的压缩实用程序。MSBuild 目标可以添加到构建过程中。因此，集成服务器可以持续运行，从而自动并容易地获得这些好处。

例如，一个 JavaScript 库的开头如下所示：

```
var Prototype = {  
    Version: '1.5.0',  
    BrowserFeatures: {
```

```

XPath: !!document.evaluate
},

ScriptFragment: '(?:<script.*?>)(\\n|\\r|.)*?(?:</script>)',
emptyFunction: function() {},
K: function(x) { return x }
}

```

执行压缩后, JavaScript 库变成如下所示(只是示例), 但仍然可以工作:

```

(c(){f 7.2q(/<\\/?[">]+>/5a,"")}),2C:(c(){f 7.2q(P 5d(1m.5s,"9n"),"")}),9j:(c(){k 9m=P
5d(1m.5s,"9n");k 9k=P
5d(1m.5s,"ce");f(7.E(9m)||[]).1F((c(9l){f(9l.E(9k)||["",""])[1]}}})),3P:(c(){f7.9j().1F((
c(4s){f 6A(4s)}})),cd:(c(){k 1h=N.4f("1h");k 2V=N.cc(7);1h.63(2V);f 1h.2P}),cb:(c(){k
1h=N.4f("1h");1h.2P=7.9i();f 1h.2O[0]? (1h.2O.o>1?SA(1h.2O).2A("",(c(3Y,1G){f
3Y+1G.4j})):1h.2O[0].4j):""),6J:(c(9h){k E=7.4d().E(/(["?#]*) (#.*)?$/);h(!E){f{}}f
E[1].3m(9h||"&").2A({),(c(2E,Q){h((Q=Q.3m("="))[0]){k v=9g(Q[0]);k
l=Q[1]?9g(Q[1]);1b;h(2E[v]!==1b){h(2E[v].3k!=1M){2E[v]=[2E[v]>h(1){2E[v].M(1)})1k{2E[v]=1
}}f 2E)}})),2F:(c(){f 7.3m("")})

```

有许多 JavaScript 最小化工具, 这只是其中的一个。但是, Ajax Minifier 的选项、完整性和与 MSBuild 的集成使其值得一试。

C.4 扩展 ASP.NET

“哦, 我射中了自己的脚! 对不起, 这只是一个玩笑!”

——Matz

C.4.1 ASP.NET AJAX 控件工具集

ASP.NET AJAX 控件工具集是微软和较大型 ASP.NET 社区合作的成果, 目标是提供可用的 Web 客户端组件的最大集合。其中包括许多很优秀的例子, 用户可以从中学学习如何编写 ASP.NET AJAX, 并有机会得到信息反馈以及在社区中共享自己的代码。

这个工具集包含数十个控件, 它们建立在 ASP.NET AJAX 架构的基础之上, 并扩展了该架构。一些控件很简单, 提供了美丽的外观, 例如阴影、圆角、水印和动画。其他控件则非常实用, 如日历、弹出窗口和滑块。

所有控件的源代码都是完全开放的, 因此用户可以扩展和改进它们。这些控件不仅仅是示例, 它们也是完整的组件, 可以在应用程序中使用。

这个工具集可以从站点 <http://ajax.asp.net/ajax/> 获得, 如图 C-7 所示。该站点甚至还包括内容分发网络(Content Distribution Network, CDN), 以便让微软为驻留这些 JavaScript 库提供带宽, 从而用户可以更快速、更方便地获得它们。

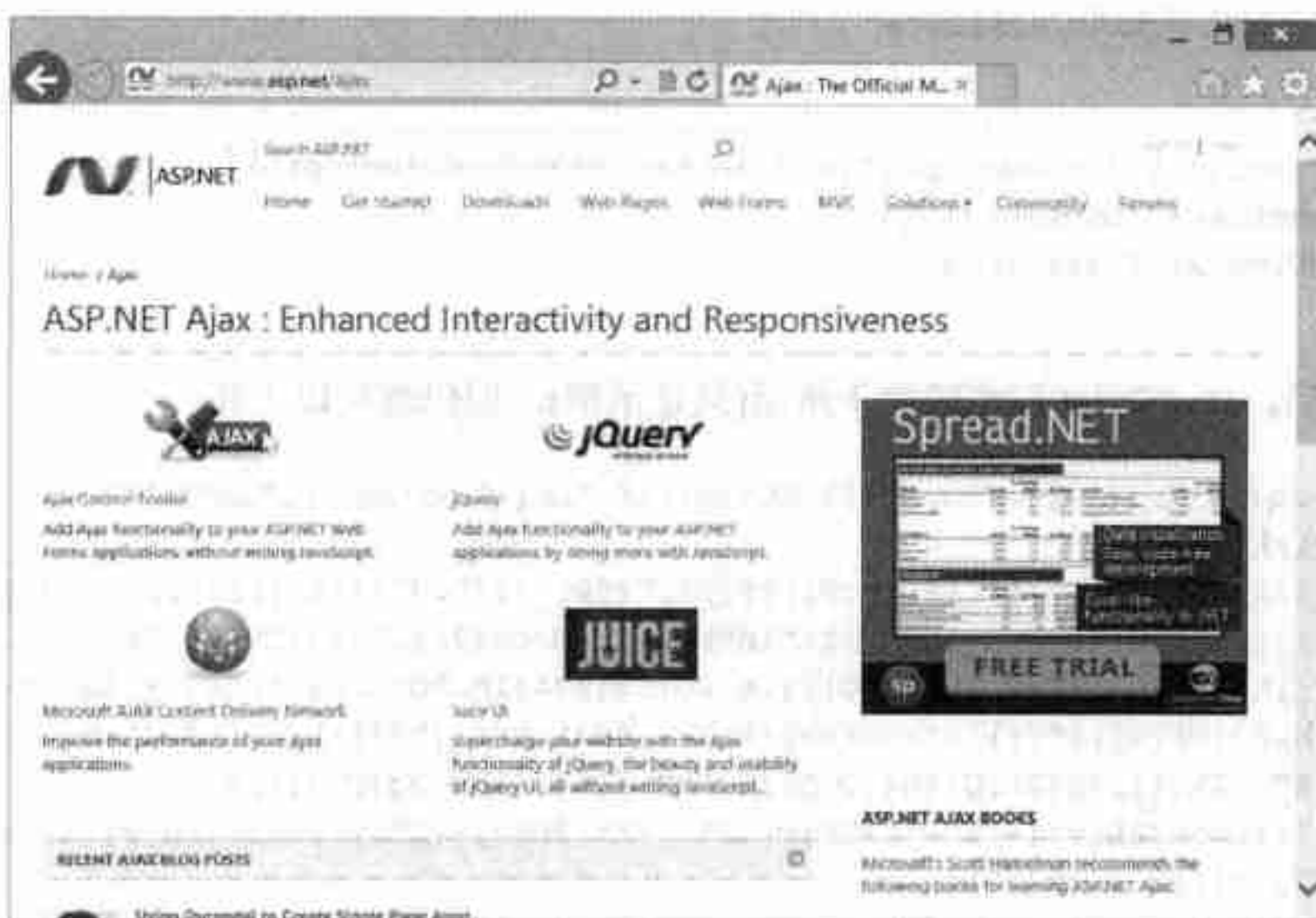


图 C-7

C.4.2 Atif Aziz 的 ELMAH——错误记录模块和处理程序

找出应用程序中的错误和未处理的异常并解决它们是一项需要大量时间的工作。并不需要每次都编写自己的定制全局异常处理程序,而是可以使用 Atif Aziz 的 ELMAH(Error Logging Modules and Handlers, 错误记录模块和处理程序)。这是一款非常灵活的、适用于整个应用程序的错误记录工具,并带有可插入的扩展,该扩展面向几乎处于每个位置的接口。甚至可以在应用程序中配置该工具,而无须重新编译或部署。只要修改 `web.config` 文件,包含错误记录模块和处理程序,就会接收到一个 Web 页面,以远程检查未处理异常的完整日志。

ELMAH 捕获大量的异常信息,因此即使关闭 `customErrors`,也可以重新构造 ASP.NET 为给定异常生成的原始的“死亡黄屏”。它就像异常的 TiVo! 图 C-8 中的 ELMAH 提供了用于开发人员的视图,其中包含调试该错误所需的所有细节。

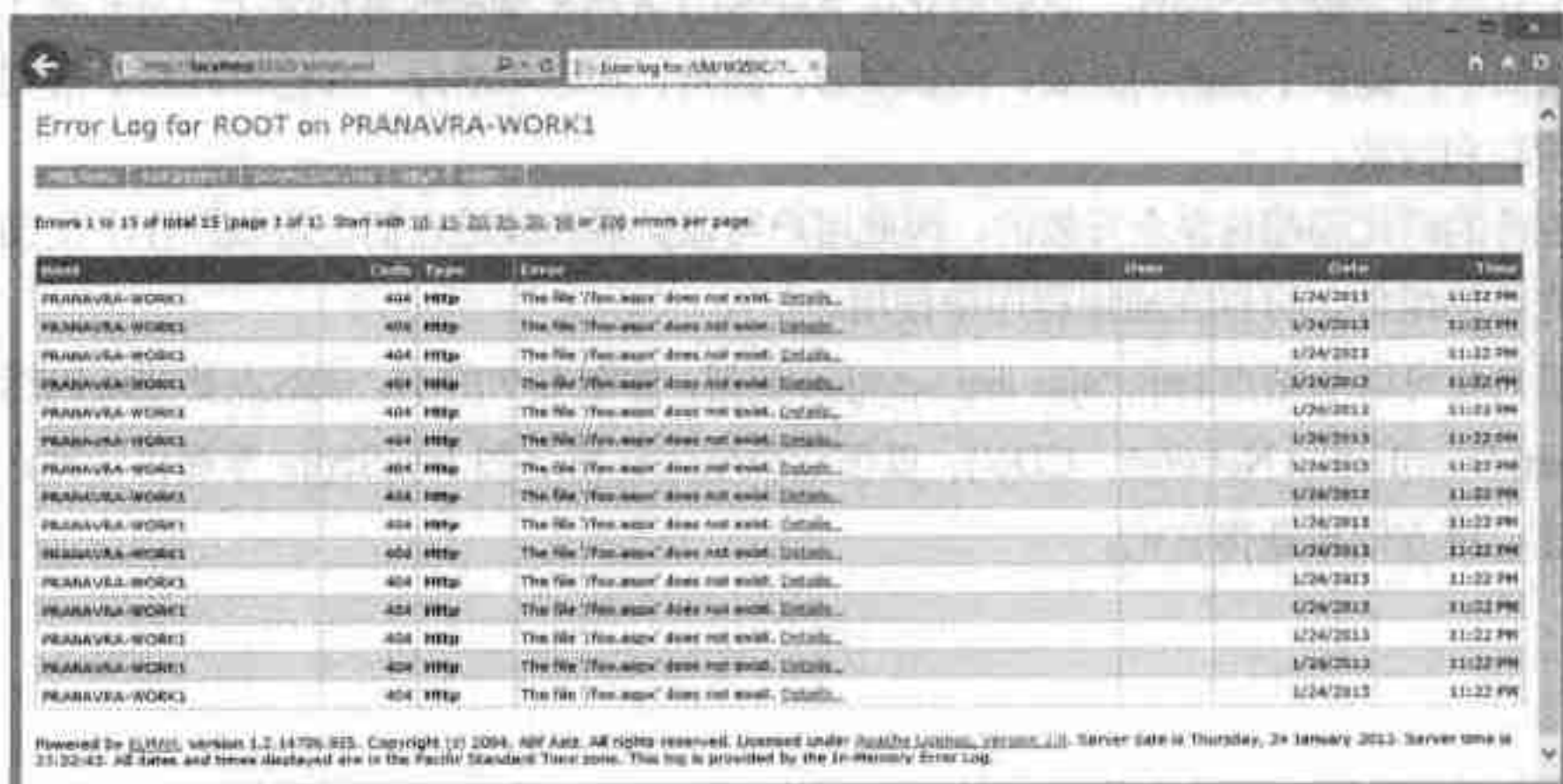


图 C-8

另一不错的功能是 RSS 提要，其中显示了最近 15 年的日志。这个灵活的工具是开放源代码的，并且最近的测试版支持中级信任环境。可以把它插入 SQL Server，或者使用 XML 文件管理错误日志。建议最好花费一些时间学习 ELMAH。

C.4.3 Helicon 的 ISAPI_Rewrite 和 IIS7 URLRewrite

Apache Web 服务器的用户非常推崇 mod_rewrite 的功能——URL 重写机制。IIS 6 用户可以使用 Helicon 的 ISAPI_Rewrite 模块获得这个功能。该工具运行起来非常快速，因为它用纯 C 语言编写的。它可以在 IIS “Classic” 管道中与 ASP.NET 很好地集成，因为在 ASP.NET 意识到发生了什么之前就已经重写了 URL。IIS 7 用户可以使用 URL 重写模块(稍后介绍)。

因为 ISAPI_Rewrite 使用正则表达式，所以一开始很容易因为过于简洁的语法而很难理解。但是，只要有耐心，它就将工具集中一款非常强大的工具。

还有 IIS 7 专用的 URI 重写模块，可以从 <http://www.iis.net/extensions/URLRewrite> 获得该模块。它比 ISAPI_Rewrite 更易于使用，因为它包括用于管理和创建重写的完整 UI，以及用于将现有的、Apache 样式的重写放入 IIS 7 中的导入工具。

另外，在 ASP.NET 应用程序中使用 IIS 7 重写工具时，该工具运行在托管的管道中以提升性能。

C.5 通用的开发人员工具

“如果棍子的一端很脏，那么可以削尖它，把它变成一个有用的工具。”

——Colin Powell

C.5.1 Telerik 的联机代码转换器

如果不借助 Telerik 的 CodeChanger.com 等工具，创建同时以 C# 和 Visual Basic 显示的示例就是很麻烦的工作。

这个小型应用程序不是官方支持的工具，但在 Visual Basic 和 C# 之间转换时，它能帮助我们完成 80% 的工作。

C.5.2 WinMerge 和其他合并工具

每个人都有自己喜欢的合并工具。无论是 WinMerge 还是 Beyond Compare，抑或旧的备用程序 WinDiff，只要确保工具列表中有自己非常熟悉的工具即可。在成员众多的软件开发团队中管理大量的改动时，优秀的合并工具有助于解开甚至最为复杂的、相互冲突的已登记改动。

WinMerge 有许多不同的插件，这些插件扩展了它的功能，以包含 Word、Excel 文档和 XML 文件的比较。

其他推荐的合并工具有 Scooter Software 的 Beyond Compare 和 SourceGear 的 DiffMerge。这 3 个工具都与 Windows Explorer 集成在一起，因此只需要右击即可比较文件。

C.5.3 .NET Reflector

如果没有使.NET Reflector, 那么.NET 开发人员的经验就会有所欠缺。Reflector 是对象浏览器、反编译器、帮助系统、强大的插件集合和易于学习的工具。微软开发人员 Lutz Roeder 编写的这个小型实用程序(现在由 Redgate 负责维护和升级工作)已经成为继 Visual Studio 之后.NET 开发人员最重要的、不可或缺的工具。

Reflector 很令人惊讶, 因为它不仅可以把 IL 转换回 C#或 VB, 从而满足程序员的意图, 它包含的分析工具还有助于可视化.NET 基类库中的方法与自己的代码或任何第三方代码中的方法之间的依赖关系。

C.5.4 Process Explorer

最后一个工具是 Mark Russinovich 的 Process Explorer。把该工具称为“任务管理器”显得并不公平。Process Explorer 把 Windows 操作系统放在显微镜下, 以观察活动的进程、线程和环境, 从而帮助你更清楚地了解当前的状况。所有的开发人员都应了解这个工具的高级用法, 以及整个 SysInternals 工具套件的用法。

在图 C-9 中, 显示了正在执行的所有进程。使用这个工具可以找到哪个进程锁定了 DLL。

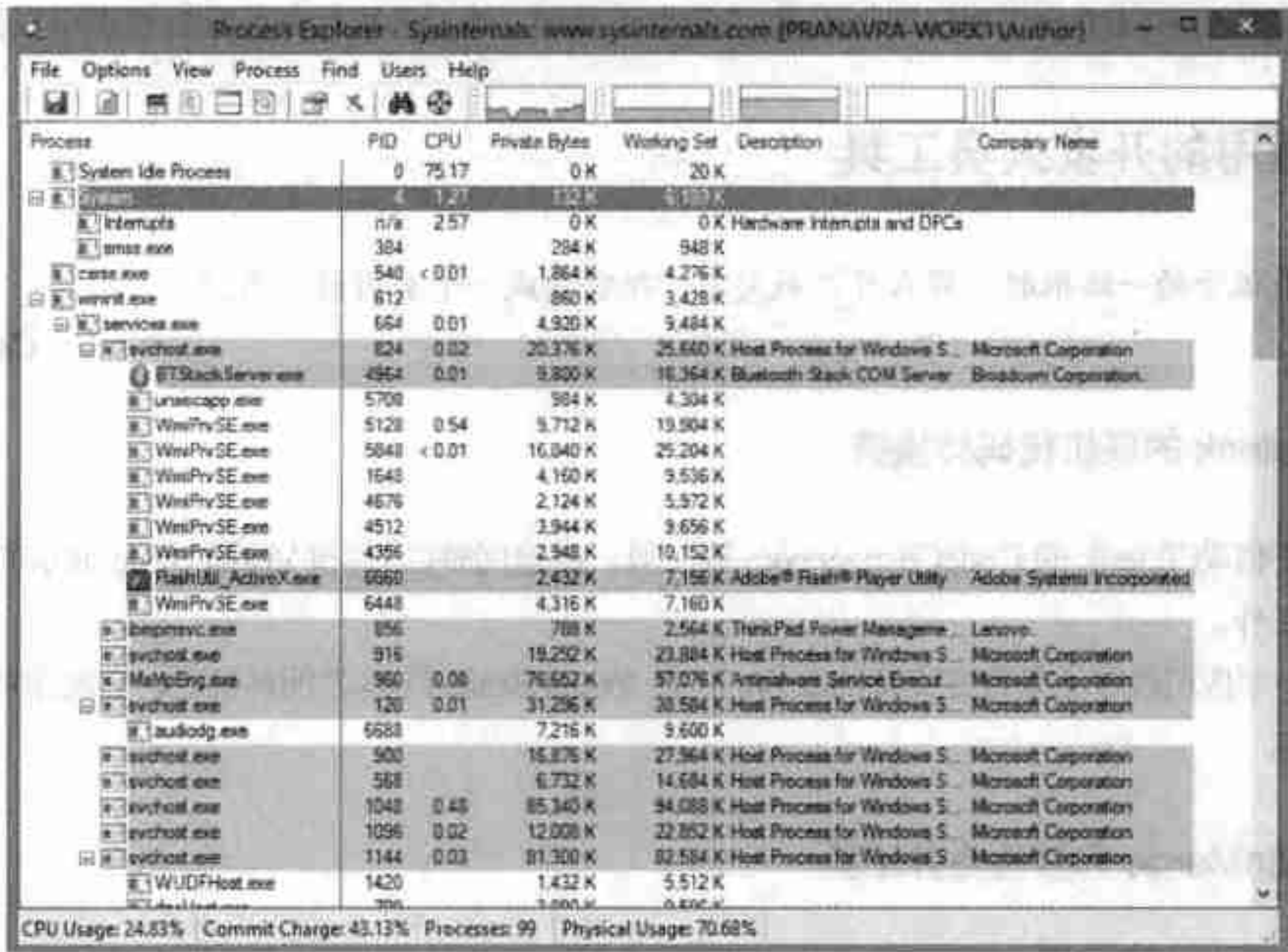


图 C-9

C.6 小结

拥有正确的工具意味着在调试时冥思苦想一周时间和 5 分钟快速分析之间的区别。正确的工具还意味着繁琐的、艰难的大量代码录入工作和令人愉悦的重构过程之间的区别。希望读者尝试使用本附录介绍的每个工具, 并搜索可用的工具, 从而找出使开发过程更高效、更愉快的工具。

D

附录

管 理

我们已经快要完成本书的学习：前面已介绍了 ASP.NET 4.5 的各种功能。但是，每向前一步都会增加复杂性，ASP.NET 的配置和管理领域也不例外。ASP.NET 开发小组也意识到了这一点，因此提供了可以让开发人员轻松、可靠地配置和管理基于 ASP.NET 的应用程序的工具和 API。

本附录将详细介绍这些工具，解释所有可用的选项。本章主要探讨两个强大的配置工具：ASP.NET Web 站点管理工具，这是一个基于 Web 的应用程序；IIS 管理器，它用于配置 ASP.NET 应用程序。

D.1 ASP.NET Web 站点管理工具

ASP.NET 在最初发布时就为 Web 应用程序引入了基于 XML 的配置文件的概念。这个文件称为 web.config，位于应用程序所在的目录。它可以存储大量的配置设置，其中一些设置可以覆盖 machine.config 文件或根服务器的 web.config 文件中定义的配置设置。但是，ASP.NET 2.0 的以前版本没有提供管理工具来配置这些设置。许多开发人员只好创建自己的配置工具，以避免手动处理 XML 文件。



如果以 ASP.NET 的 Empty 项目模板开始工作，就必须使用 NuGet 安装通用提供程序和 LocalDB(因为 SQL Server Express 默认不随 Visual Studio 2012 一起安装)。安装 Microsoft ASP.NET Universal Providers for LocalDB，就可以在 NuGet 中找到正确的软件包(详见附录 G)。

ASP.NET Web 站点管理工具(WAT)可以通过简单、易用的 Web 接口管理 Web 站点的配置，不需要手动编辑 web.config 文件。如果第一次使用该管理工具时没有 web.config 文件，就会创建该文件。ASP.NET Web 站点管理工具还会在 Web 站点的 App_Data 文件夹中默认创建 LocalDB 文件，以存储应用程序数据。在 ASP.NET Web 站点管理工具中对大多数设置进行的修改会立即起作用，并反

映在 web.config 文件中。

默认设置会自动从 Web 服务器根文件夹中的任何配置文件进行继承。ASP.NET Web 站点管理工具允许创建或更新 Web 应用程序的设置。还可以重写从上一级配置文件继承的设置。如果某些设置不允许重写，它们会在配置工具中淡显。

在安装 .NET Framework 4.5 时，会自动安装 ASP.NET Web 站点管理工具。要使用该管理工具管理 Web 站点，必须登录为站点的注册用户，并且有 web.config 文件的读写权限。

我们不能远程访问 ASP.NET Web 站点管理工具，甚至不能通过 IIS 进行本地访问。但是，我们可以使用 Visual Studio 2012 访问，Visual Studio 2012 会使用集成的 Web 服务器(IIS Express)访问该管理工具。

如果通过 Visual Studio 2012 使用这个工具，就打开 Web 站点或 Web 应用程序，在位于 Website 或 Project 窗格顶部的菜单中单击 ASP.NET Configuration 按钮。还可以在 Visual Studio 主菜单的 WebSite 菜单中选择 ASP.NET Configuration 命令，以启动该工具。图 D-1 显示了 ASP.NET Web 站点管理工具的欢迎页面。

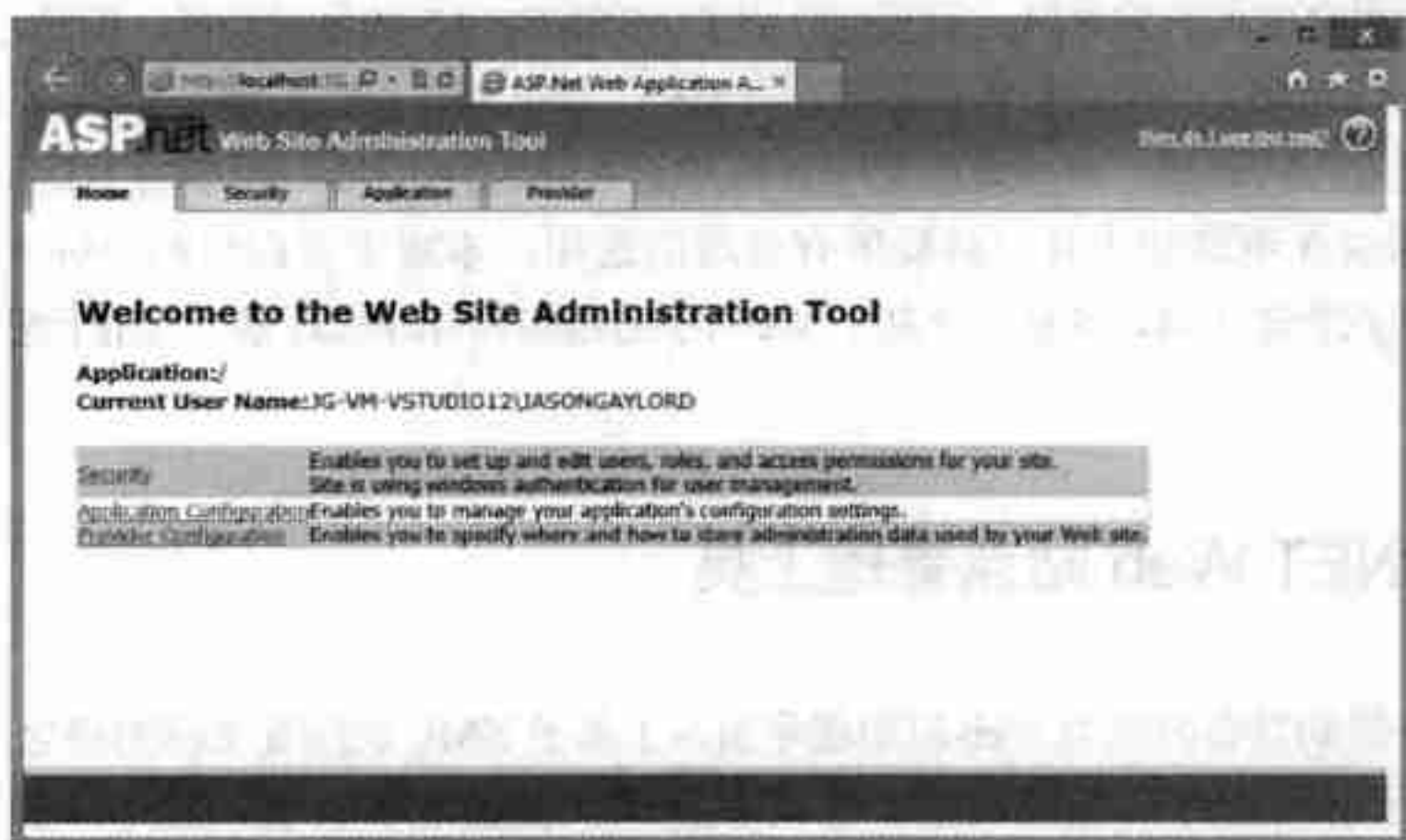


图 D-1

ASP.NET Web 站点管理工具具有一个带选项卡的界面，该界面分组相关的配置设置。各个选项卡和它们管理的配置设置将在下面详细介绍。



ASP.NET Web 站点管理工具要求把 SQL Server 或 SQL Server Express 用作默认的 SQL 连接字符串。WSAT 当前不支持把 LocalDB 作为数据库，也不支持把 Simple Membership 作为身份验证或角色提供程序。

D.1.1 Home 选项卡

Home 选项卡(如图 D-1 所示)提供了正在监控或修改的应用程序的基本信息，包括应用程序的名称，以及访问应用程序的当前用户。另外，还有指向其他管理工具选项卡的链接，提供了其中设置的概要。要修改 Web 应用程序，可以单击相应的选项卡或链接。

使用这个管理工具对配置设置进行的大多数修改都会立即起作用,使 Web 应用程序重新启动,如果使用的是 InProc 会话,当前活动的会话就会丢失。管理 ASP.NET 的最佳方式是在应用程序的开发阶段修改配置,以后再把这些修改发布到产品应用程序中。因此,这个工具不能在 Visual Studio 外部使用。

一些设置(在管理工具的界面中,这些设置有专门的 Save 按钮)不会自动保存。如果没有单击 Save 按钮,保存对 web.config 文件的修改,在这些窗口中输入的信息就会丢失。ASP.NET Web 站点管理工具在一段不活动的时间过后会超时,没有立即起作用并且没有保存的设置会丢失。

ASP.NET Web 站点管理工具只管理可用于 Web 应用程序的一些配置设置。对于其他设置,则需要使用 IIS 管理器或 Configuration API,通过 Visual Studio 手动修改配置文件。

D.1.2 Security 选项卡

使用 Security 选项卡可以管理 Web 应用程序的安全区域、用户账户和角色的访问权限。在这个选项卡中,可以选择 Web 应用程序是从内联网访问,还是从互联网访问。如果指定内联网,就使用基于 Windows 的验证方式,否则就配置基于表单的验证方式,后者需要在定制数据库(如 SQL Server 数据库表)中管理用户。基于 Windows 的验证方式使用用户的 Windows 登录信息来验证身份。



本节讨论使用 WSAT 配置安全性的几种方式。这一主题的更多内容可查阅第 19 章。

用户信息默认存储在数据库中,在 Web 应用程序的 App_Data 文件夹中自动创建该数据库。最好把敏感信息存储在另一个更安全的数据库中,该数据库可能位于一台独立的服务器上。修改数据存储,就意味着需要修改底层的数据提供程序。为此,使用 Provider 选项卡选择另一个数据提供程序。有关 Provider 选项卡的介绍,详见本附录后面的内容。

在 Security 选项卡中可以通过两种方式配置安全设置:选择 Setup 向导或使用为 Users、Roles 和 Access Management 部分提供的链接。图 D-2 显示了 Security 选项卡。

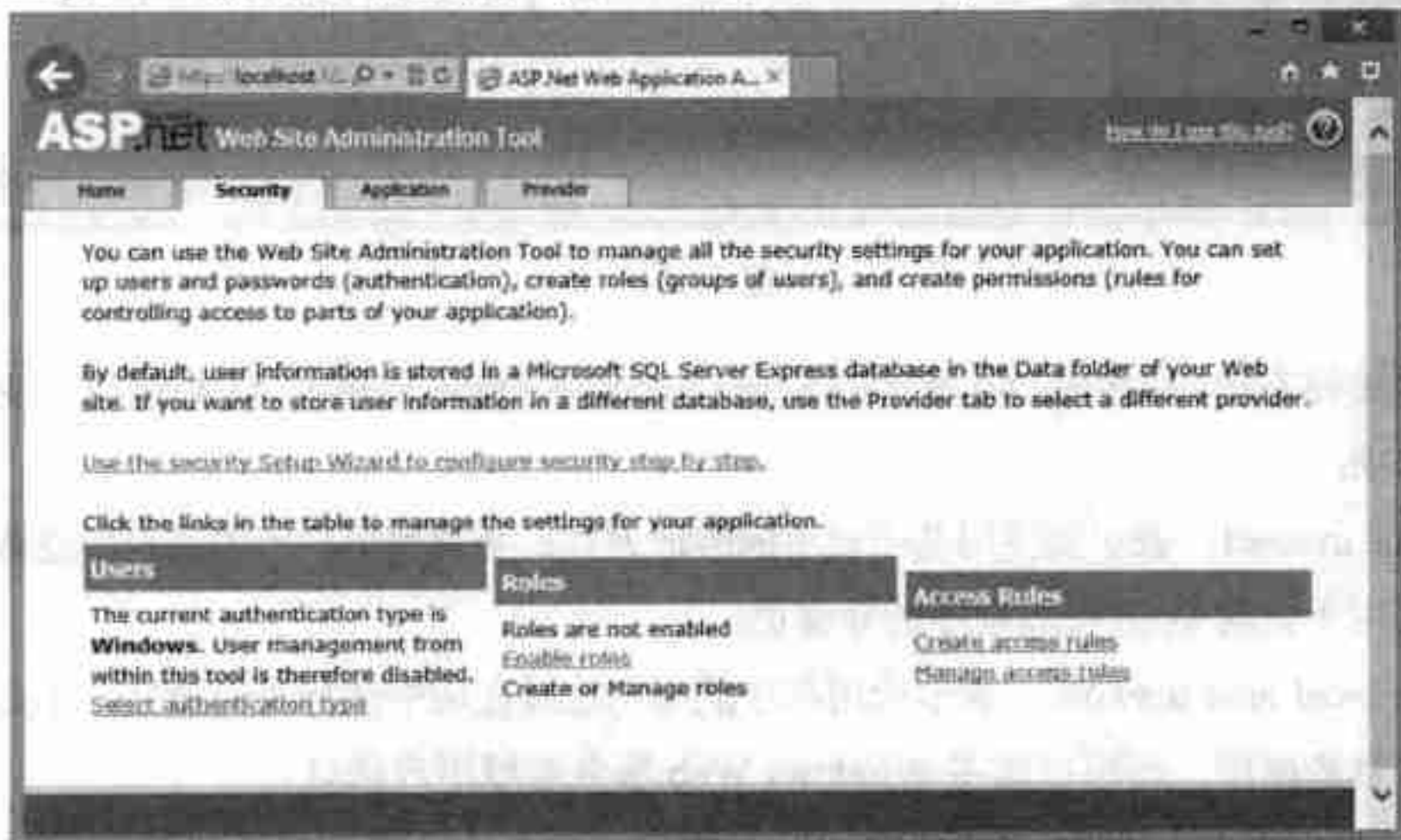


图 D-2

可以使用向导来配置初始设置，后面将介绍创建和修改安全设置的其他方式。

1. Setup 向导

Setup 向导使用 7 步过程完成选择验证用户身份的方式，以及选择存储用户信息的数据源，之后依次定义角色、用户和访问规则等任务。



在执行该向导之前，一定要创建需要特殊权限的所有文件夹。

按照下面的步骤使用 Setup 向导：

(1) 欢迎：在 Security 选项卡中，单击“Use the security Setup Wizard to configure security step by step”链接。向导的欢迎屏幕只显示了一些信息，如图 D-3 所示，告诉我们 ASP.NET 中安全管理的基本信息。阅读完这些信息后，单击 Next 按钮。

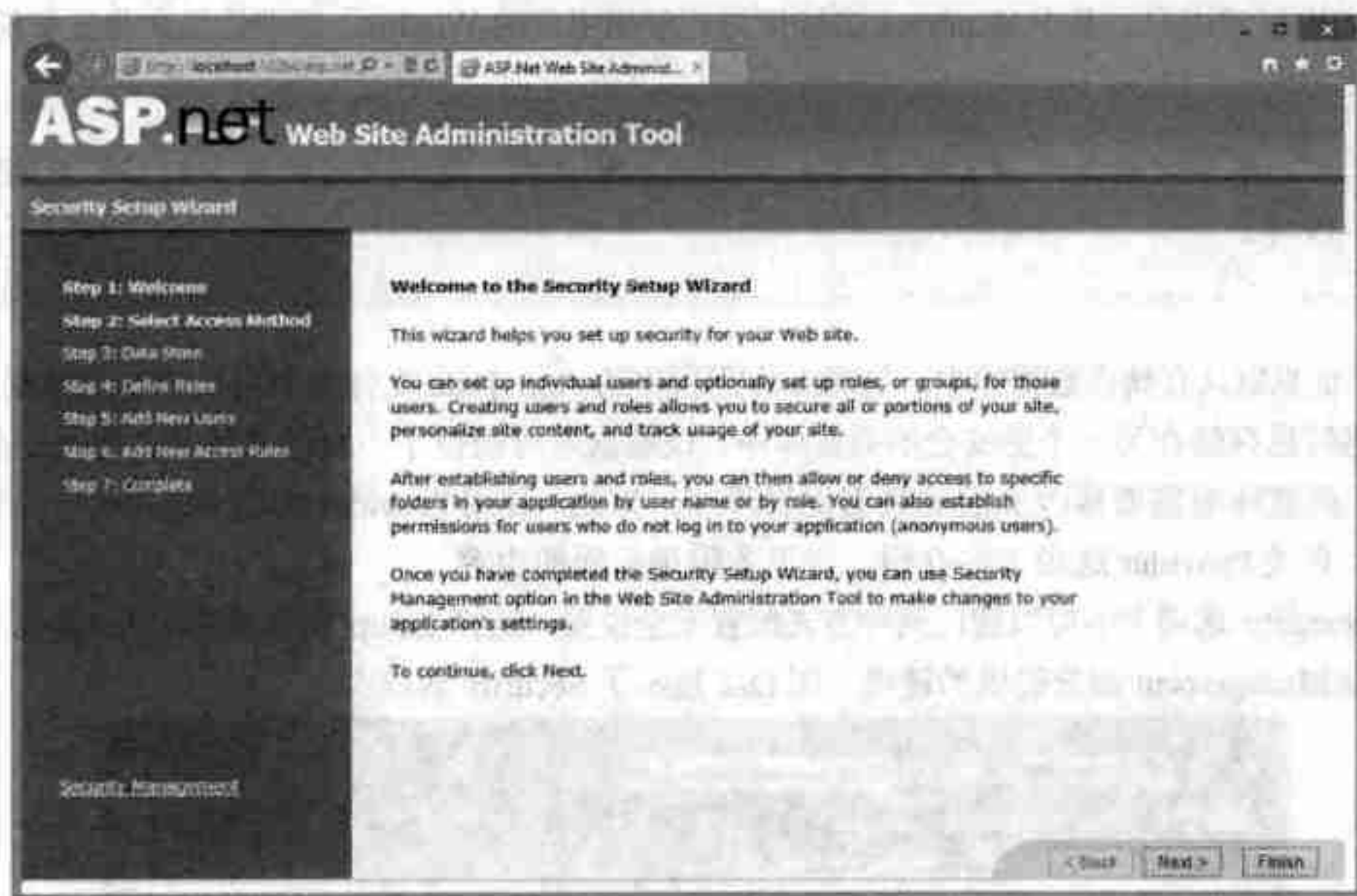


图 D-3

(2) 选择访问方法。从如图 D-4 所示的 Select Access Method 屏幕中，选择访问方法(验证机制)。这里有两个选项：

- From the internet: 表示要使用基于表单的验证方式。必须使用自己的用户信息数据库。这个选项可用于访问 Web 应用程序的非雇员。
- From a local area network: 表示应用程序的用户已经在域中进行身份验证，不必使用自己的用户信息数据库，而可以使用 Windows Web 服务器域用户信息。

选择 From the internet 选项，然后单击 Next 按钮。

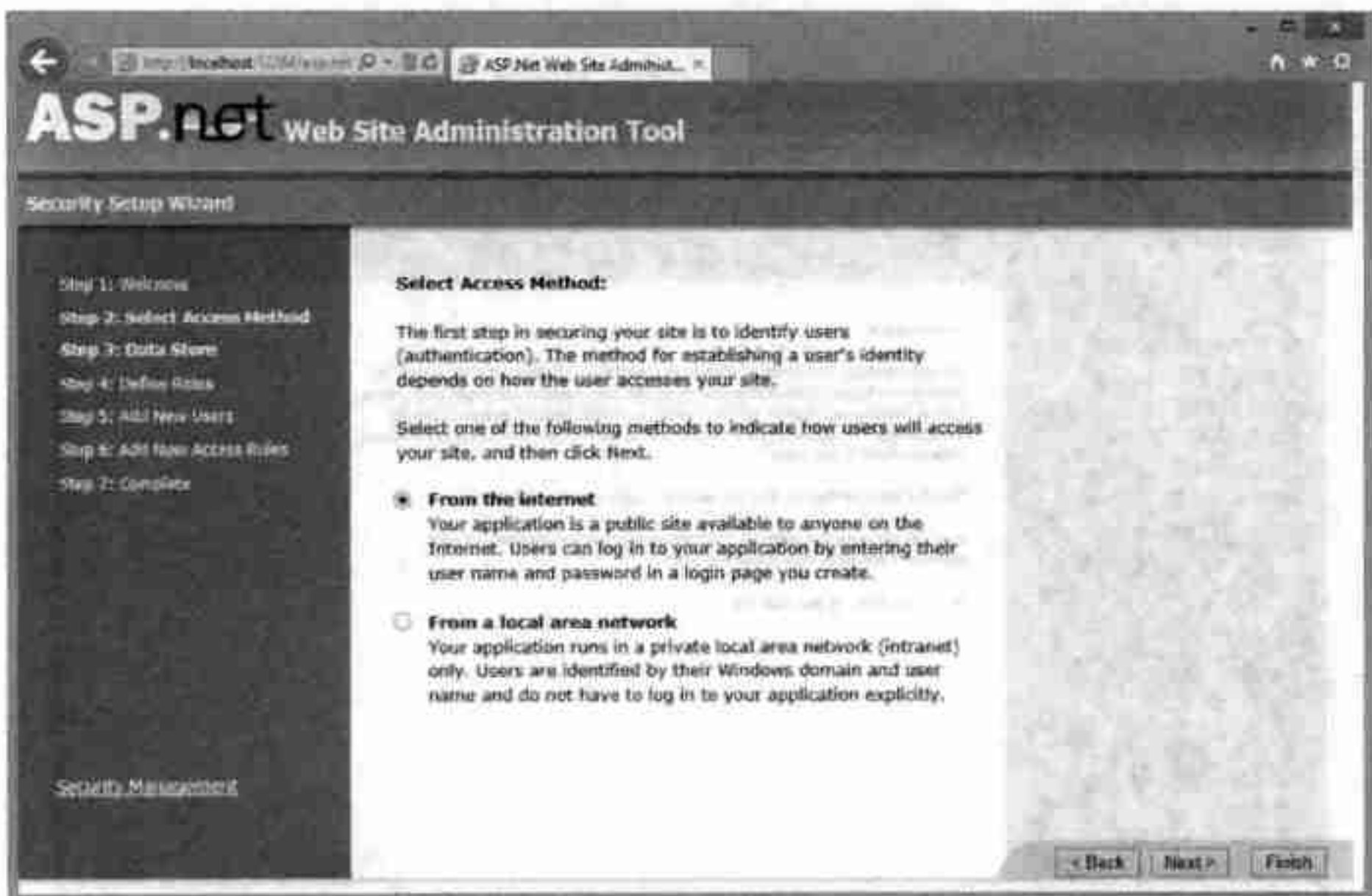


图 D-4

(3) 存储数据。如前所述，ASP.NET Web 站点管理工具默认使用 SQL Server Express Edition。可以在 Provider 选项卡中配置其他提供程序。在如图 D-5 所示的步骤(3)中，只显示一个高级提供程序，因为还没有配置其他提供程序。单击 Next 按钮。

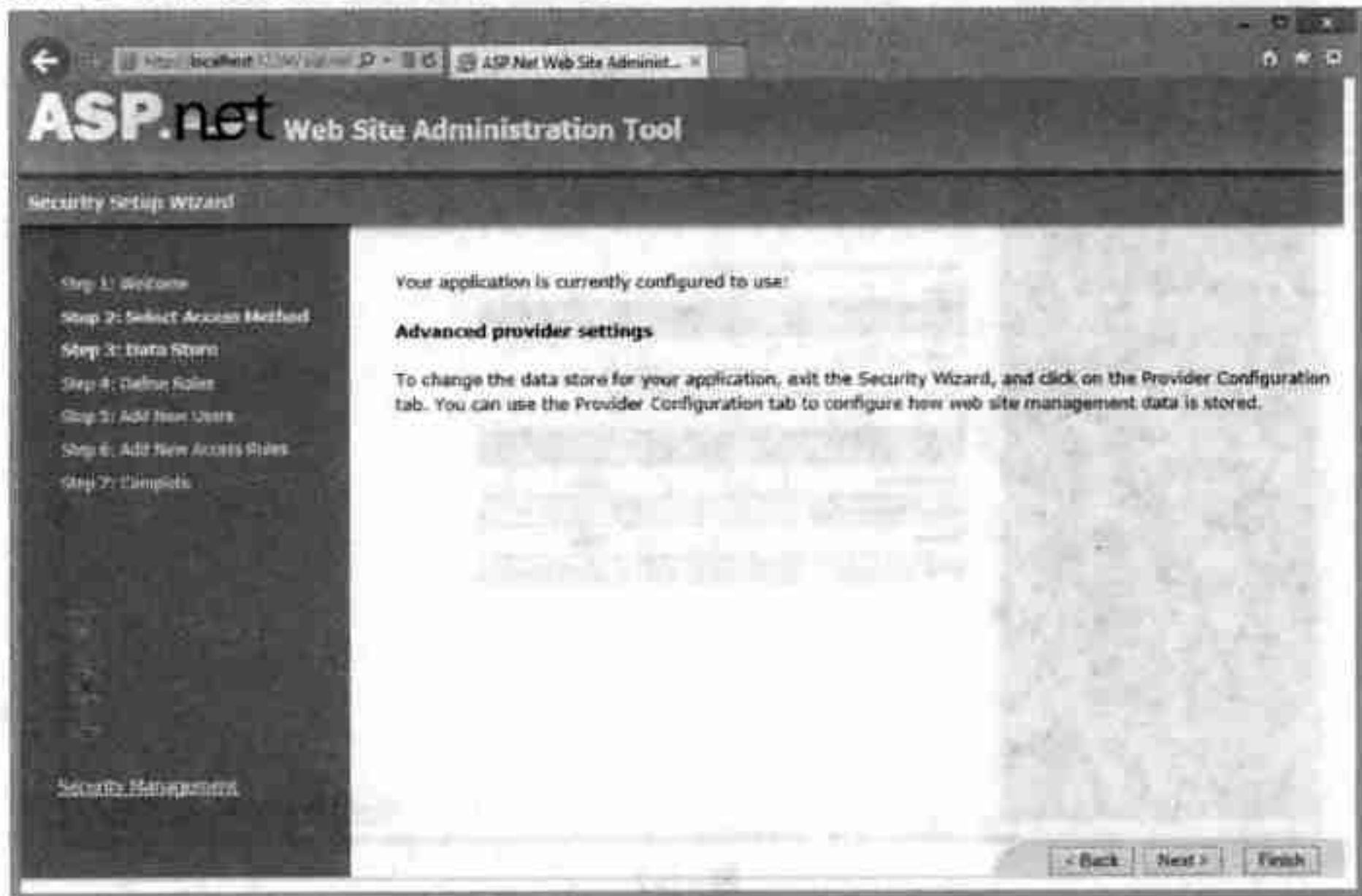


图 D-5

(4) 定义角色。如果希望所有的用户都有相同的访问权限，就可以取消对 Enable roles for this Web site 复选框的选中，跳过这一步(如图 D-6 所示)。如果没有选中这个复选框，单击 Next 按钮就会直接进入 User Management 屏幕。选中这个复选框将显示如何使用该向导定义角色。一切就绪后，单

击 Next 按钮。



图 D-6

向导的下一个屏幕如图 D-7 所示，通过该屏幕可以创建和删除角色。角色仅定义用户的类别，以后可以根据这些角色提供用户和访问规则。为 Administrator、Human Resources、Interns 和 Sales 创建角色。单击 Next 按钮。

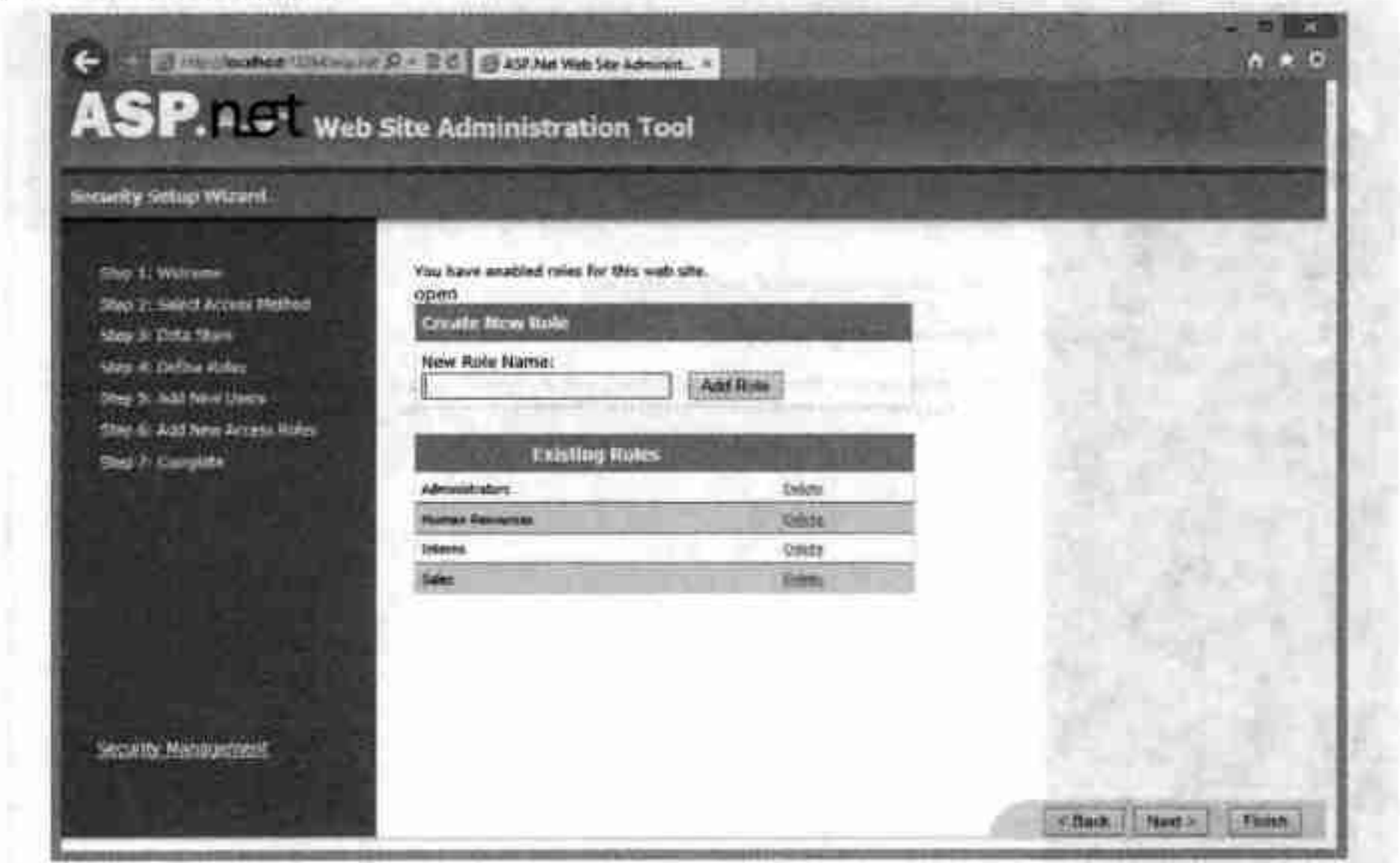


图 D-7

(5) 添加新用户。前面选择了 From the internet 选项，因此向导假定我们要使用表单验证方式，并提供创建和管理用户的选项(From a local area network 选项使用基于 Windows 的验证方式)。
可以在 Add New Users 屏幕(如图 D-8 所示)中输入用户名、密码、电子邮件地址、安全性问题和答案。

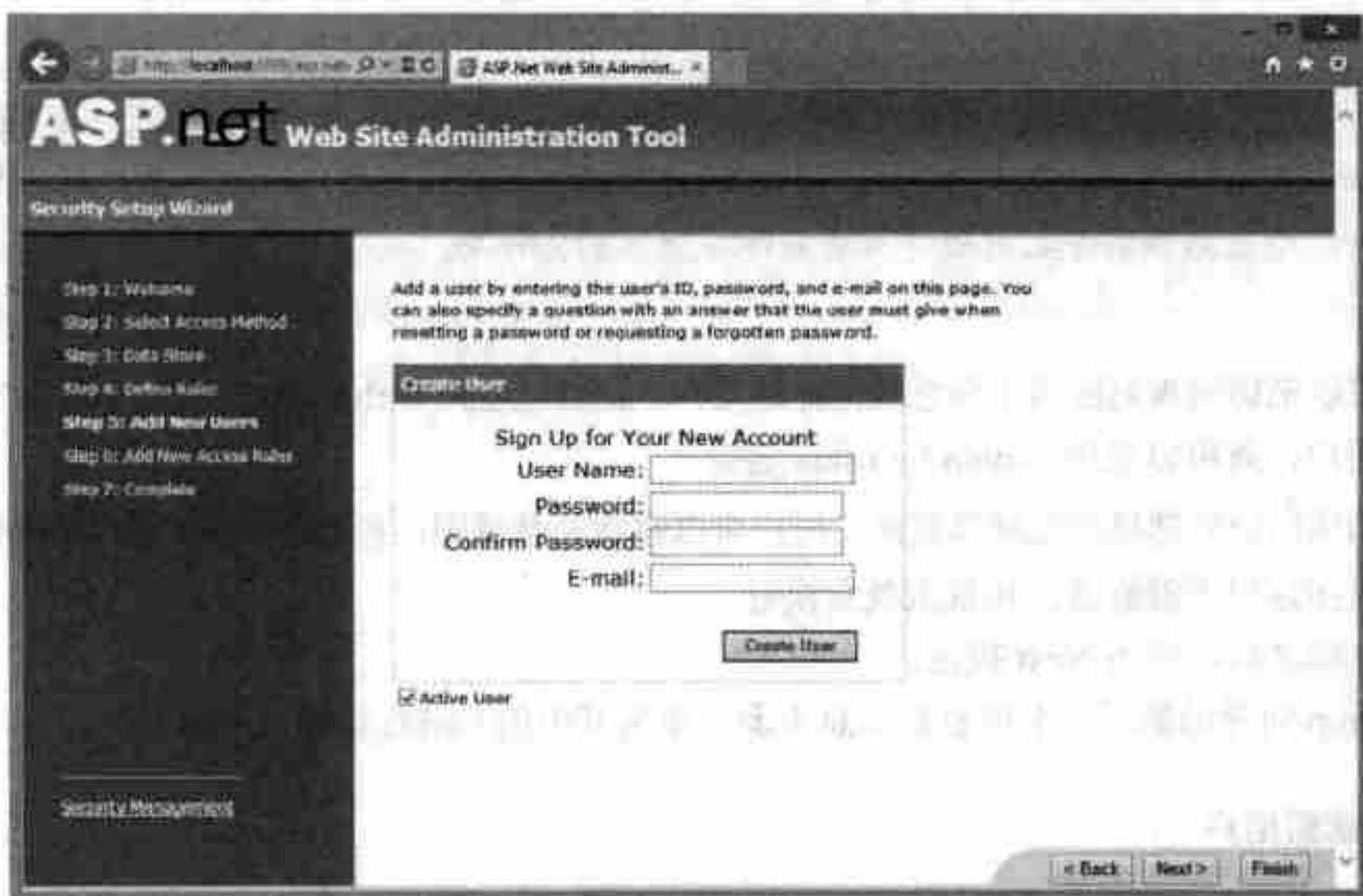


图 D-8

可以创建任意多个用户，但要删除或更新用户的信息，就必须退出向导，独立管理用户。如前所述，向导仅用于创建初始配置，以供以后管理。单击 Next 按钮。

(6) 添加新的访问规则，如图 D-9 所示。首先，在 Web 应用程序中选择需要特殊安全设置的文件夹，然后选择使用该规则的角色或用户。选择权限(Allow 或 Deny)，并单击 Add This Rule 按钮。例如，选择 Secure 文件夹、Administrator 角色、Allow 单选按钮，就允许 Administrator 角色中所有的用户访问 Secure 文件夹。

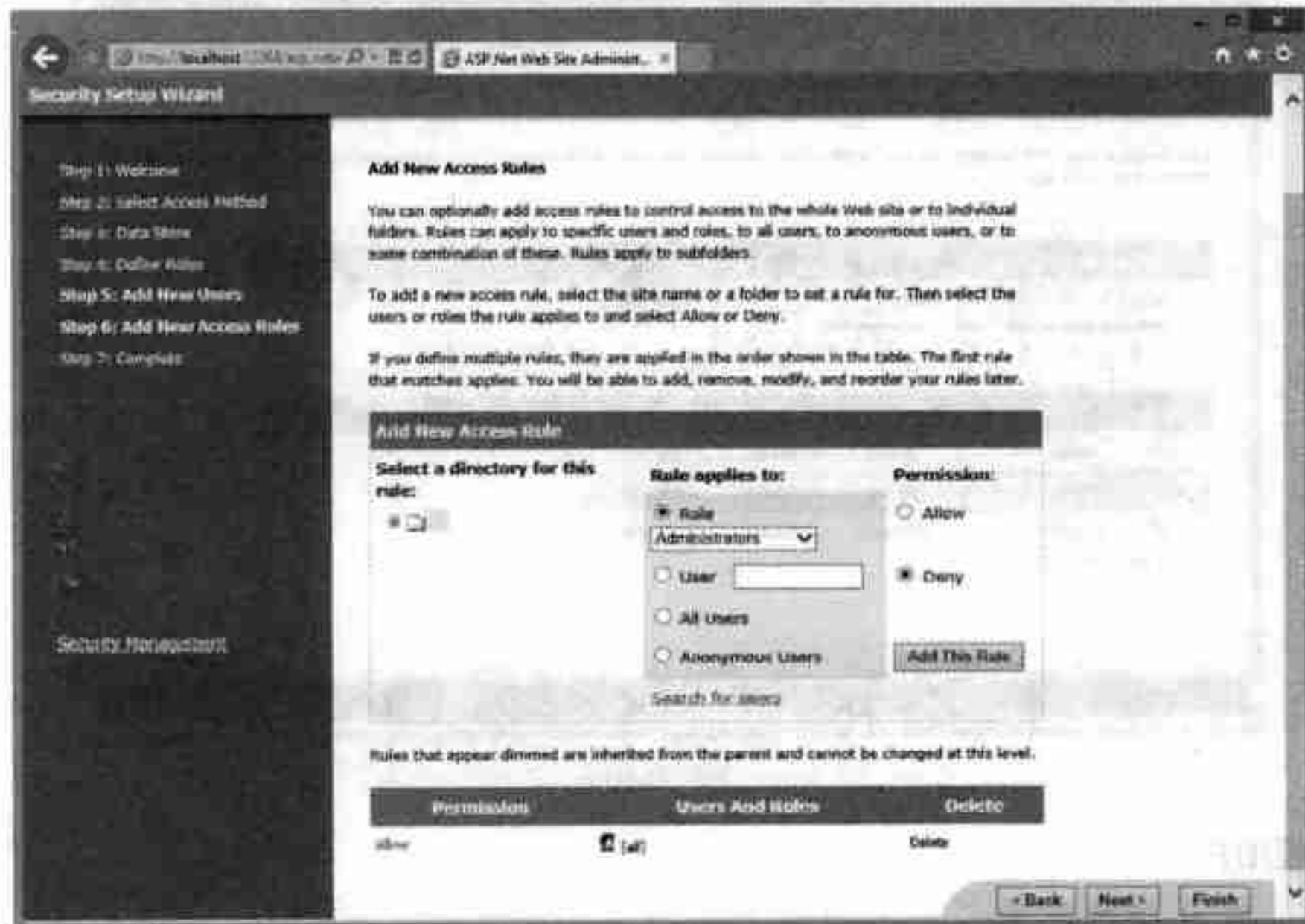


图 D-9



所有需要特殊权限的文件夹都必须事先创建。向导中显示的信息都已高速缓存，如果在这个屏幕中要为 Web 应用程序创建新的文件夹，这些信息就不会更新。因此，一定要在启动向导之前创建具有特殊安全性的文件夹。

向导可以把访问规则应用于角色或特定的用户。如果已经为 Web 站点定义了许多用户，并且要搜索某个用户，就可以使用 Search for users 链接。

所有的访问规则都显示在屏幕底部，用户可以删除某条规则，再重新启动。如果规则继承于父配置，并且在这里不能修改，该规则就会淡显。

一切就绪之后，单击 Next 按钮。

(7) Setup 向导的最后一个屏幕是信息页面，单击其中的 Finish 按钮以退出向导。

2. 创建新用户

ASP.NET Web 站点管理工具的 Security 选项卡提供了在不使用向导的情况下管理用户的功能，这非常有助于用户、角色和访问权限的维护。

要创建新用户，只需单击 Security 选项卡主页上的 Create new user 链接，如图 D-10 所示。打开 Create User 屏幕，如图 D-10 所示。可以在该屏幕中输入用户名、密码、电子邮件、安全性问题和答案。可以把新用户赋予 Roles 列表中的任意角色，这些是当前为 Web 应用程序定义的角色。使用这个工具创建用户 Admin、HRUser 和 SalesUser，把它们赋予对应的角色。

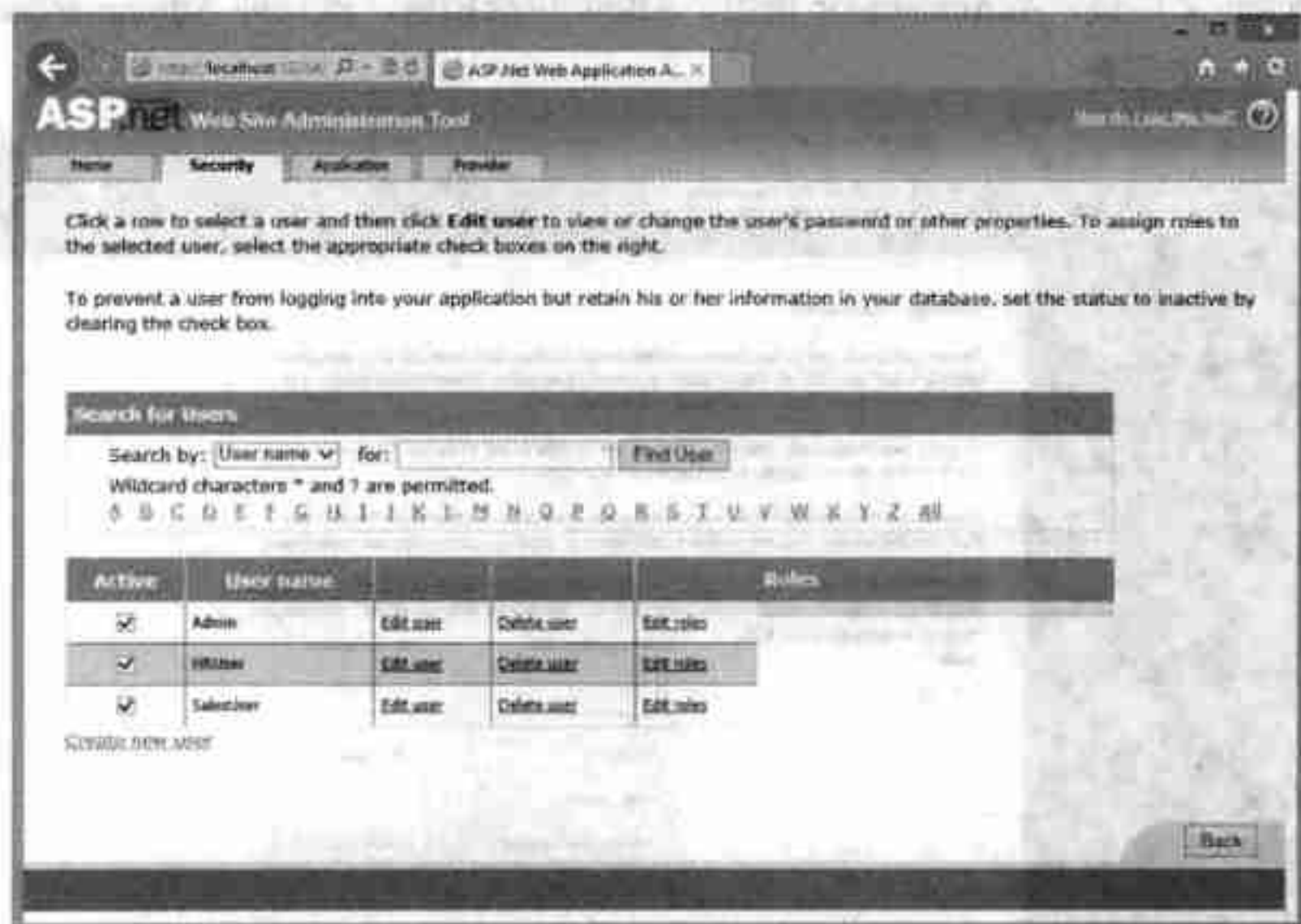


图 D-10

3. 管理用户

要管理已有的用户，可以单击 Security 选项卡中的 Manage users 链接，打开一个新屏幕，其中列出了所有已存在的用户，如图 D-10 所示。还有搜索选项，如果用户列表较长，利用搜索选项就

可以快速查找特定的用户。

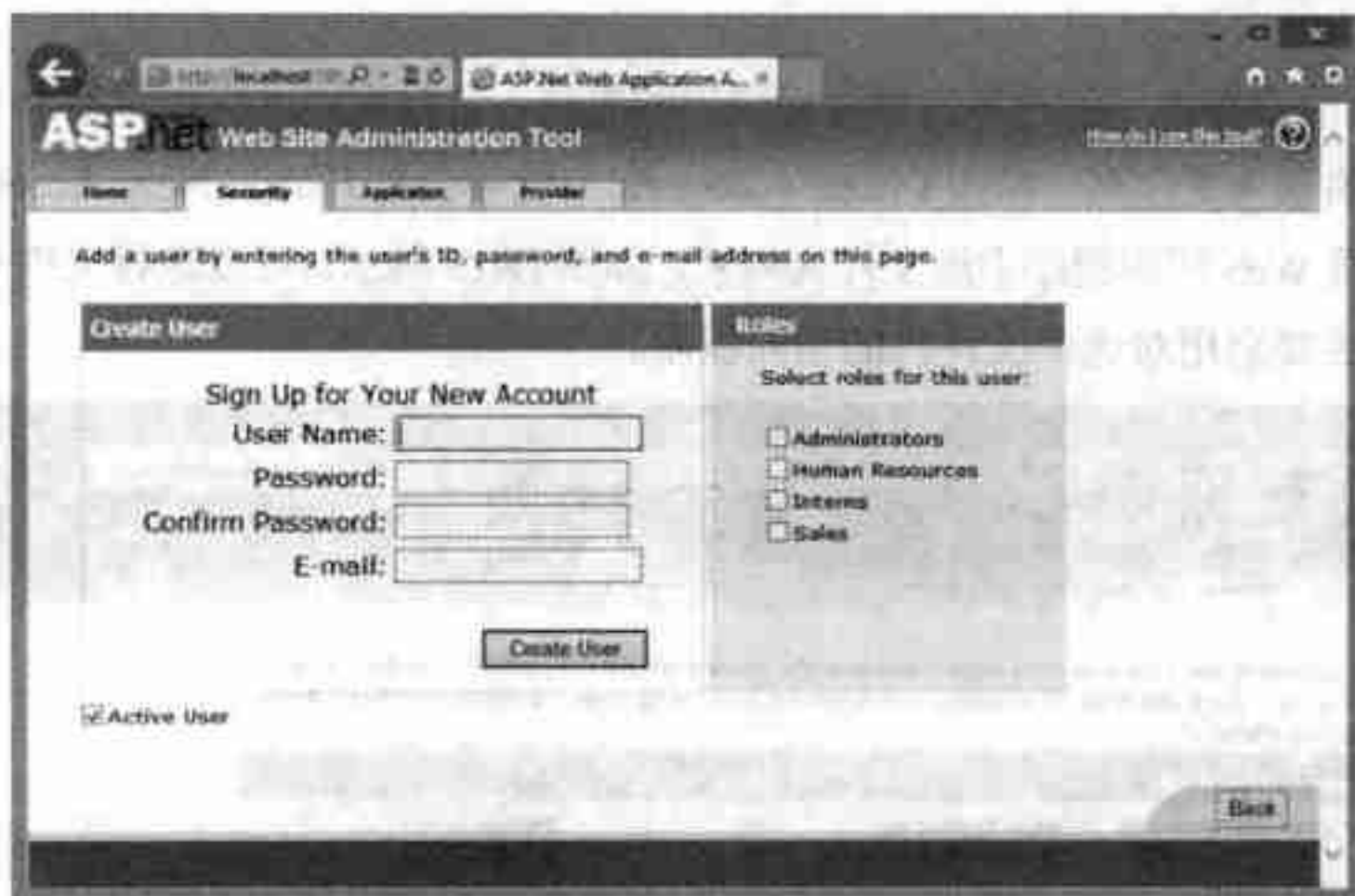


图 D-11

找到要管理的用户后，就可以更新用户信息、删除用户、重新赋予角色，或者把用户设置为活动的或非活动的。

4. 管理角色

Security 选项卡中提供了管理角色的两个链接：Disable roles 和 Create or Manage roles。单击 Disable roles 链接只会禁用 Web 应用程序的角色管理功能，并且会使另一个链接淡显。

单击 Create or Manage roles 链接将开始管理角色，把用户赋予特定的角色。所显示的屏幕给出了前面定义的所有角色。可以添加新角色、删除已有的角色或管理特定的角色。

单击特定角色旁边的 Manage 链接，会显示一个屏幕，其中列出了当前赋予该角色的所有用户，如图 D-12 所示。搜索用户名，就可以找到其他用户，然后把它们赋予选中的角色，或者从选中的角色中删除它们。

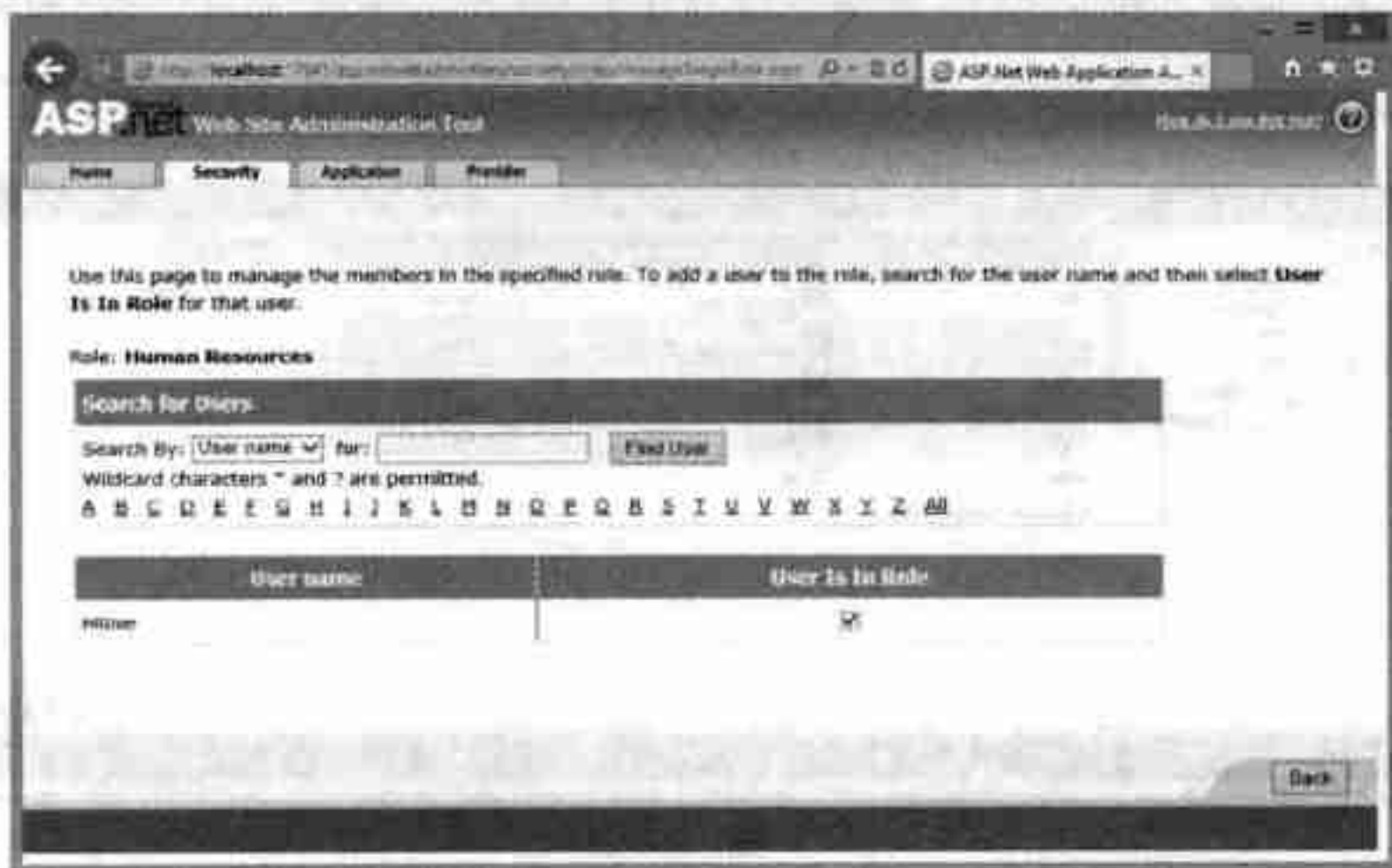


图 D-12

5. 管理访问规则

Security 选项卡还提供了创建和管理访问规则的选项。访问规则可应用于整个 Web 应用程序，也可应用于其中的某些文件夹。单击 **Create access rules** 链接会打开 **Add New Access Rule** 屏幕，在该屏幕中可以查看 Web 应用程序中的文件夹列表，如图 D-13 所示。可以选择文件夹、角色或用户，然后可以选择是否要启用对选中文件夹的访问权限。

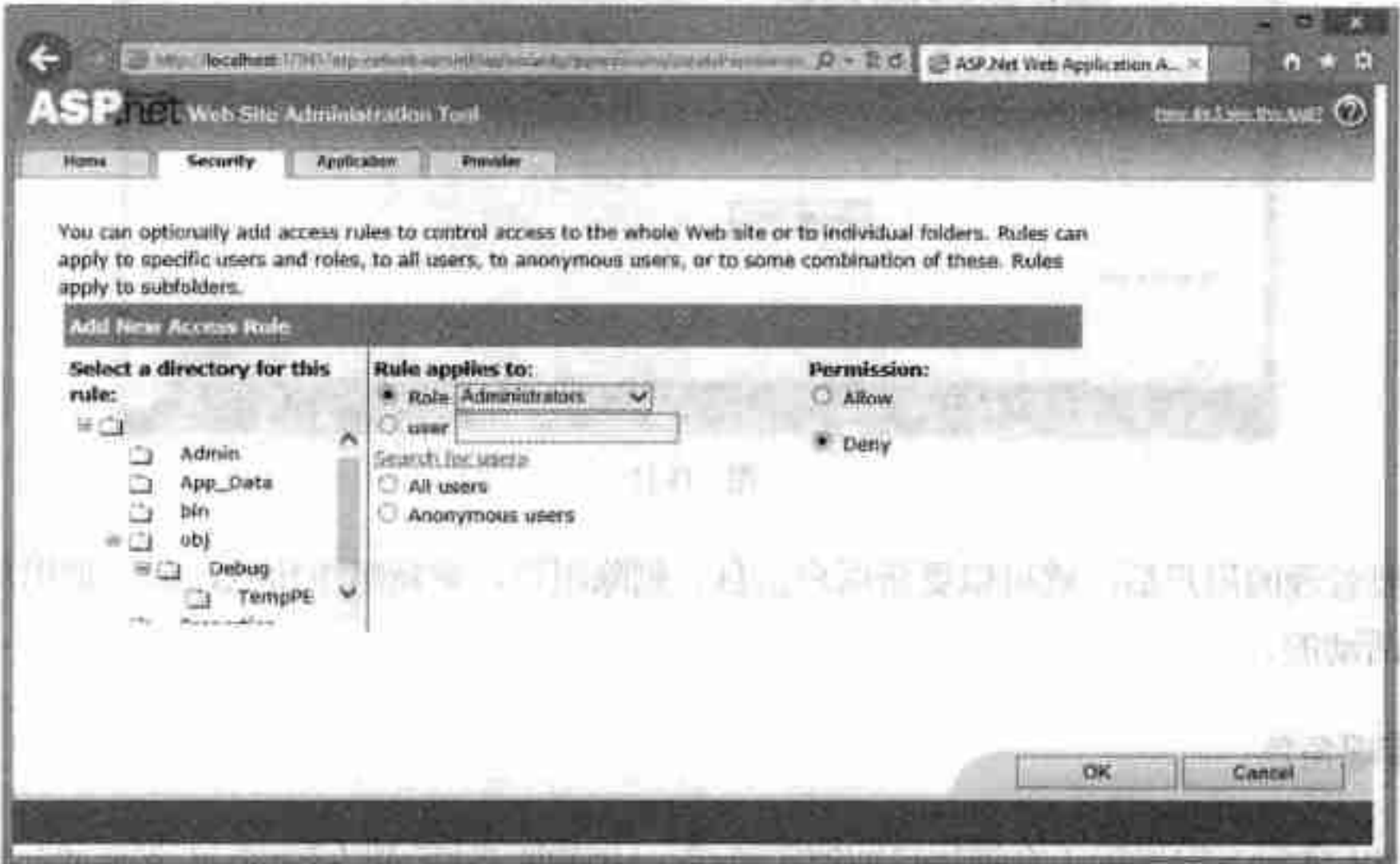


图 D-13

单击 Security 选项卡中的 **Manage access rules** 链接，会打开一个屏幕，其中显示了所有已有的访问规则，如图 D-14 所示。可以删除其中的规则，也可以添加新规则。如果要以某种顺序应用规则，还可以重新调整访问规则列表。

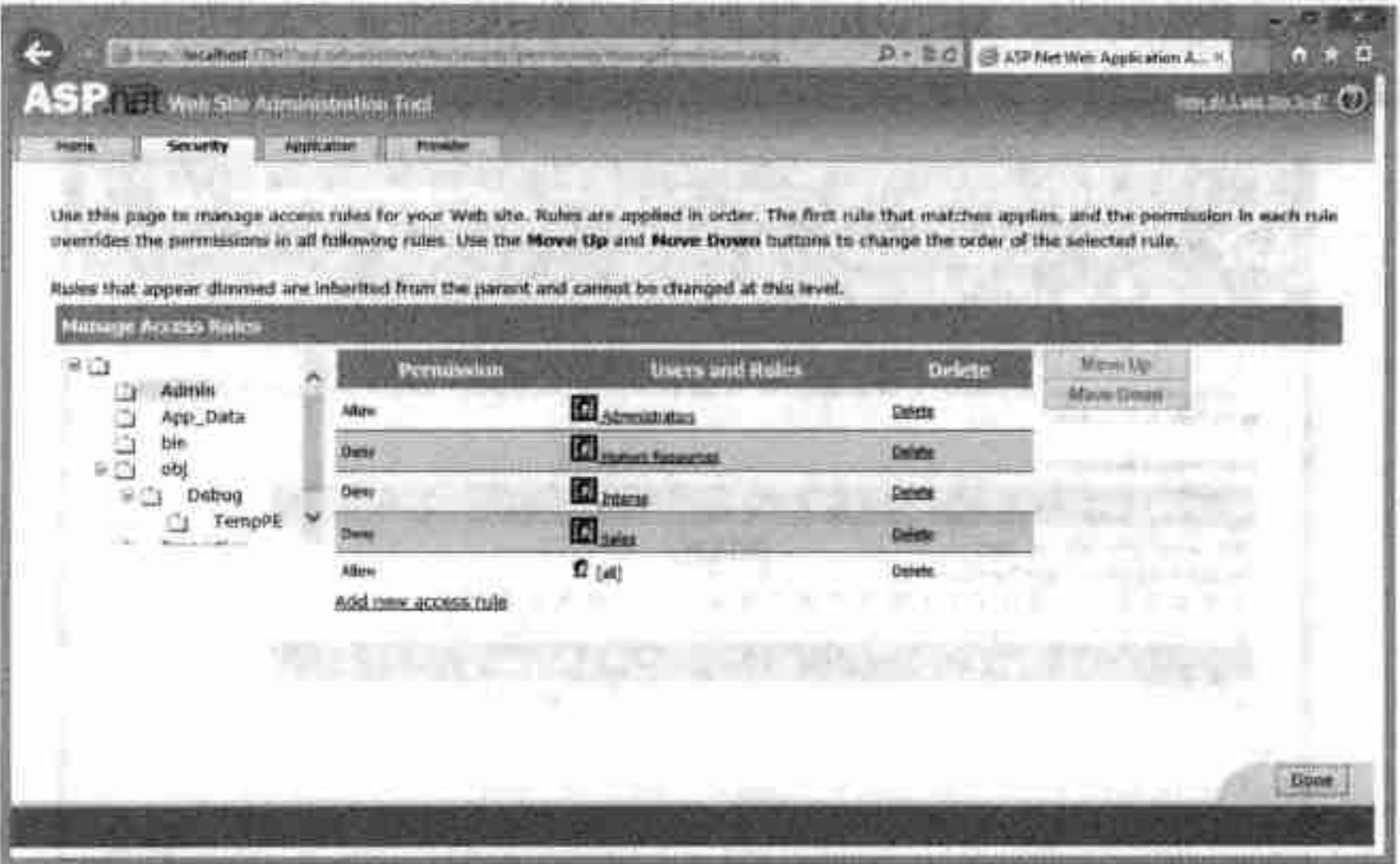


图 D-14

D.1.3 Application 选项卡

Application 选项卡提供了许多与应用程序相关的配置,包括 appSettings 的配置、SMTP 邮件服务器设置、调试和跟踪设置以及整个 Web 应用程序的启动和停止。



本节讨论使用 WSAT 更新应用程序配置的几种方式。这个主题的更多内容可查阅本书的第三部分。

1. 管理应用程序设置

Application 选项卡的左边显示了创建和管理应用程序设置的链接。这些设置存储在 web.config 文件的<appSettings>部分。大多数 ASP.NET 程序员都习惯于在 ASP.NET 的以前版本中手动修改这个标记。图 D-15 显示了 Application 选项卡。

单击 Create application settings 链接,会打开一个屏幕,在该屏幕上用户可以提供名称和值信息。单击 Manage application settings 链接,也会打开一个屏幕,在该屏幕上用户可以查看已有的设置,编辑或删除它们。还可以在这个屏幕中创建新设置。

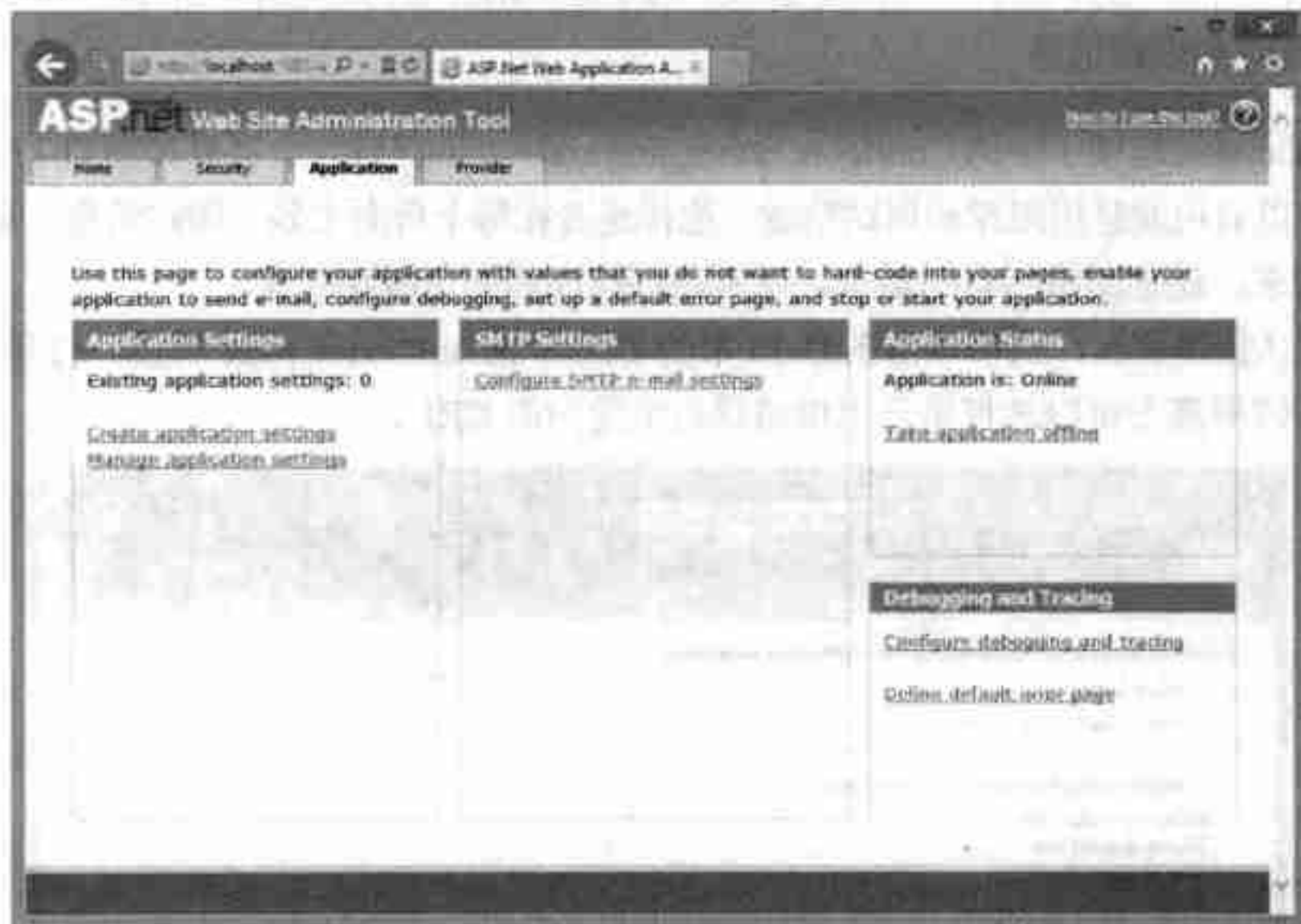


图 D-15

2. 管理 SMTP 配置

单击 Configure SMTP e-mail settings 链接,将打开如图 D-16 所示的屏幕。如果 Web 应用程序可以发送自动生成的电子邮件,配置 SMTP 邮件设置功能就很有用。我们不是在代码中表示 SMTP 服务器配置,而是在管理工具的配置文件中表示。

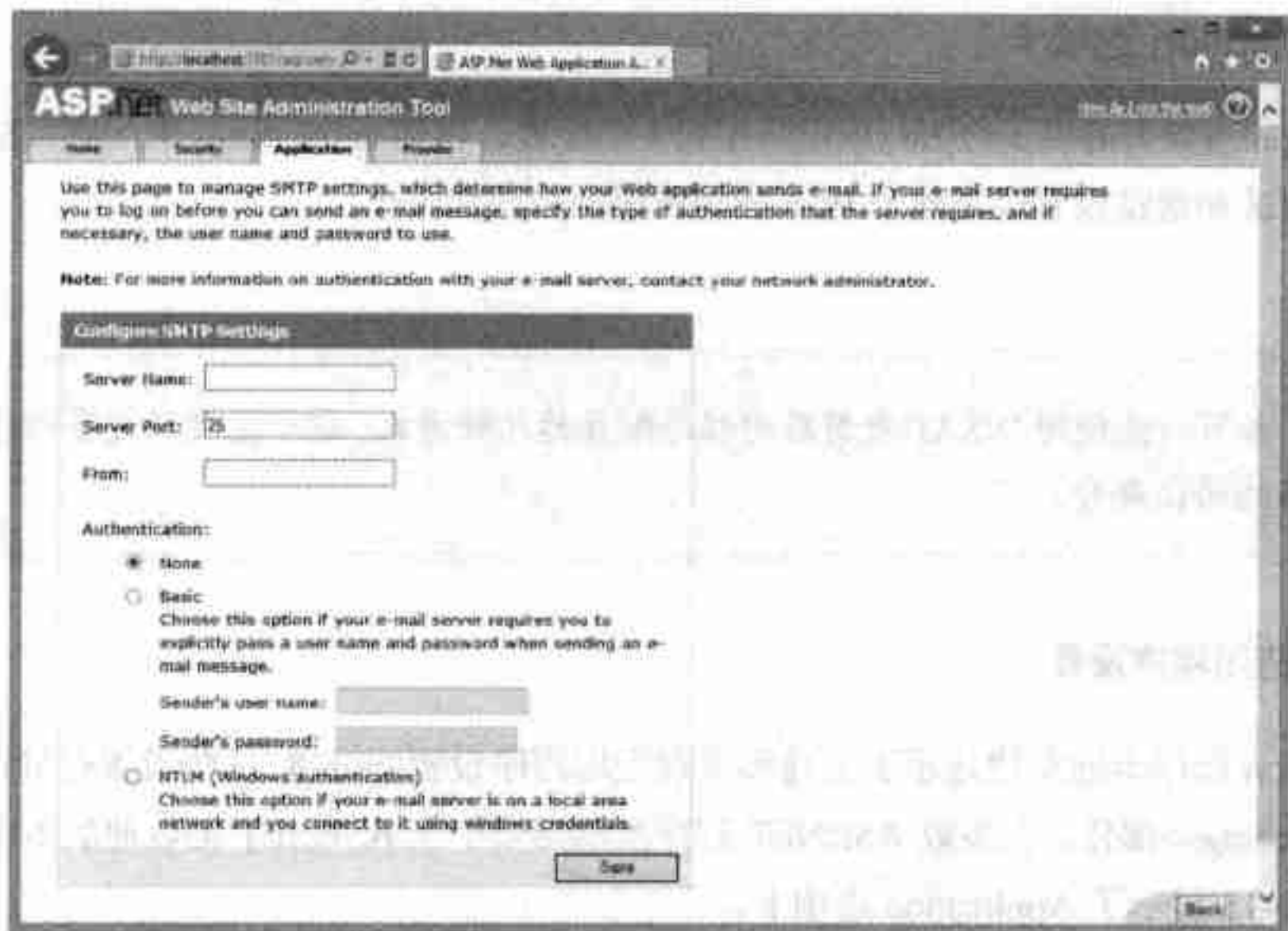


图 D-16

可以在该屏幕上指定服务器名、端口、发送者的电子邮件地址和身份验证类型。

3. 管理跟踪和调试信息

单击 Application 选项卡中的 **Configure debugging and tracing** 链接, 会打开如图 D-17 所示的屏幕, 在该屏幕上可以启用或禁用跟踪和调试功能。选择是否在每个页面上显示跟踪信息。还可以指定是只跟踪本地请求, 还是跟踪所有的请求, 或者跟踪排序和高速缓存配置。

要配置默认错误页面, 只需单击图 D-15 中的 **Define default error page** 链接。打开如图 D-18 所示的屏幕, 在该屏幕上可以选择用于出现错误时重定向的 URL。

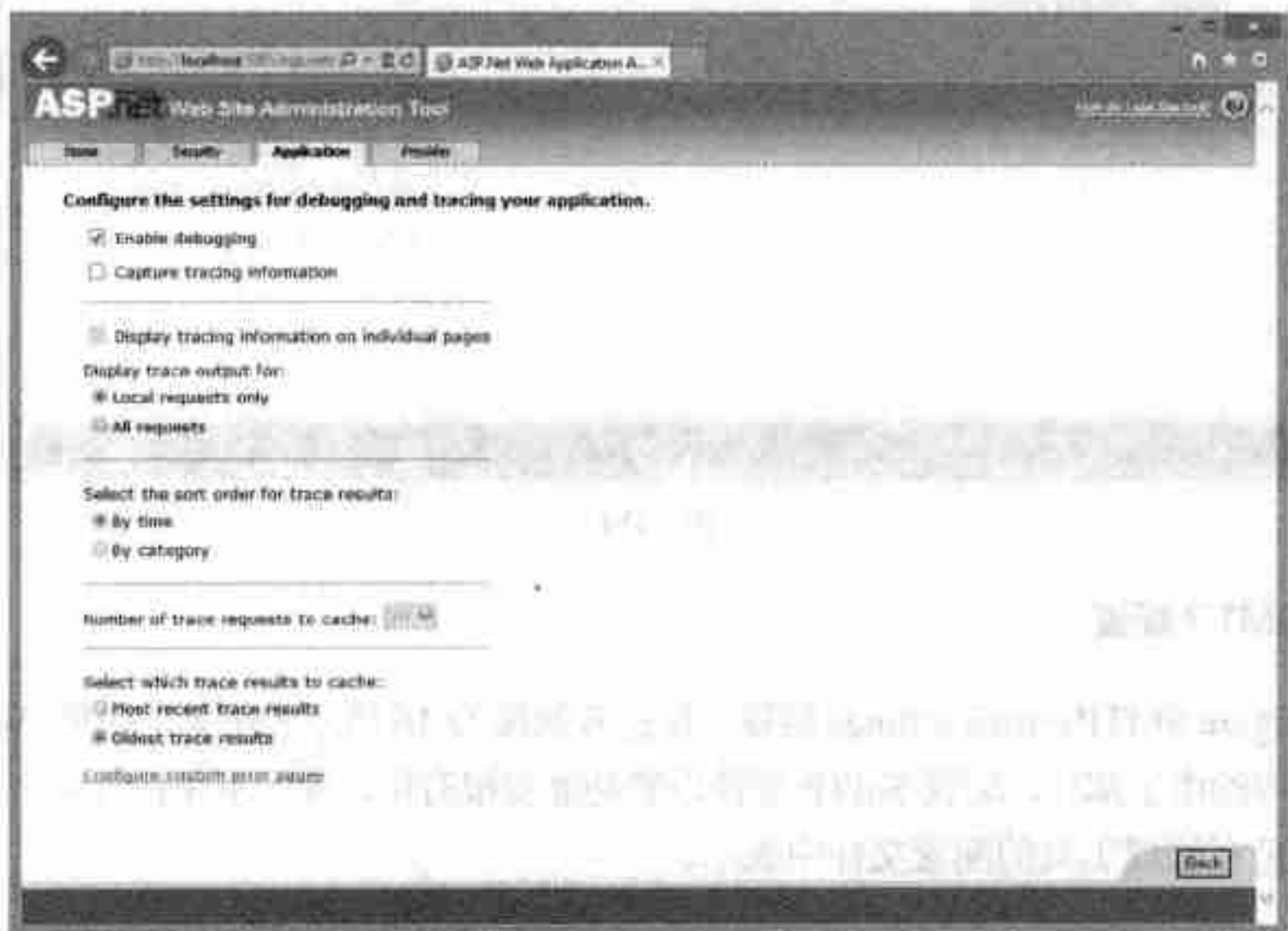


图 D-17

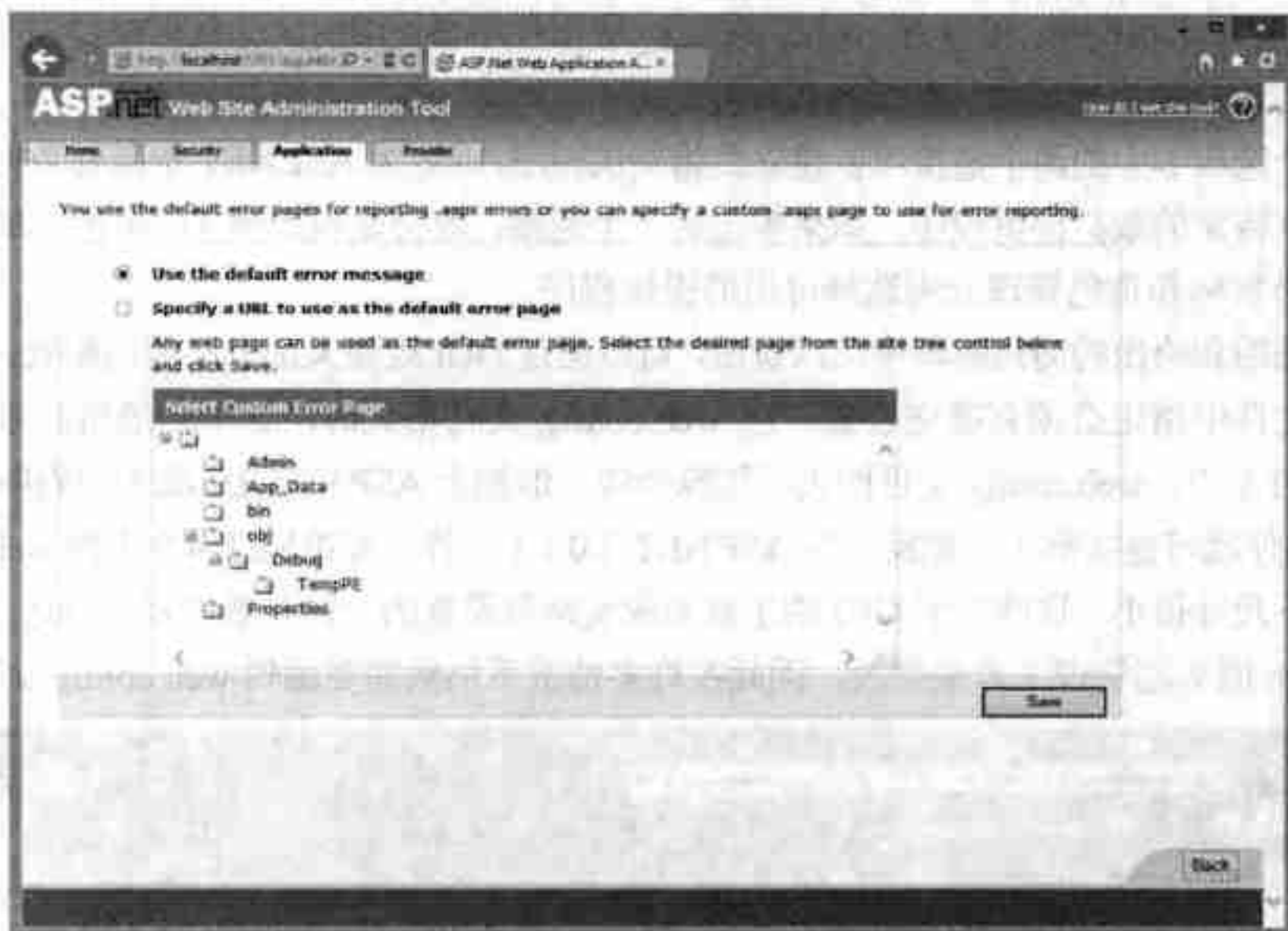


图 D-18

4. 使应用程序脱机

要使整个 Web 应用程序脱机，只需单击图 D-15 中的 **Take application offline** 链接，该链接就会停止 Web 应用程序的应用程序域。如果要对应用程序进行定期维护，就可以使用该链接。

D.1.4 Provider 选项卡

ASP.NET Web 站点管理工具的最后一个选项卡是 **Provider** 选项卡，如图 D-19 所示。使用它可以建立其他提供程序，确定应用程序要使用的提供程序。

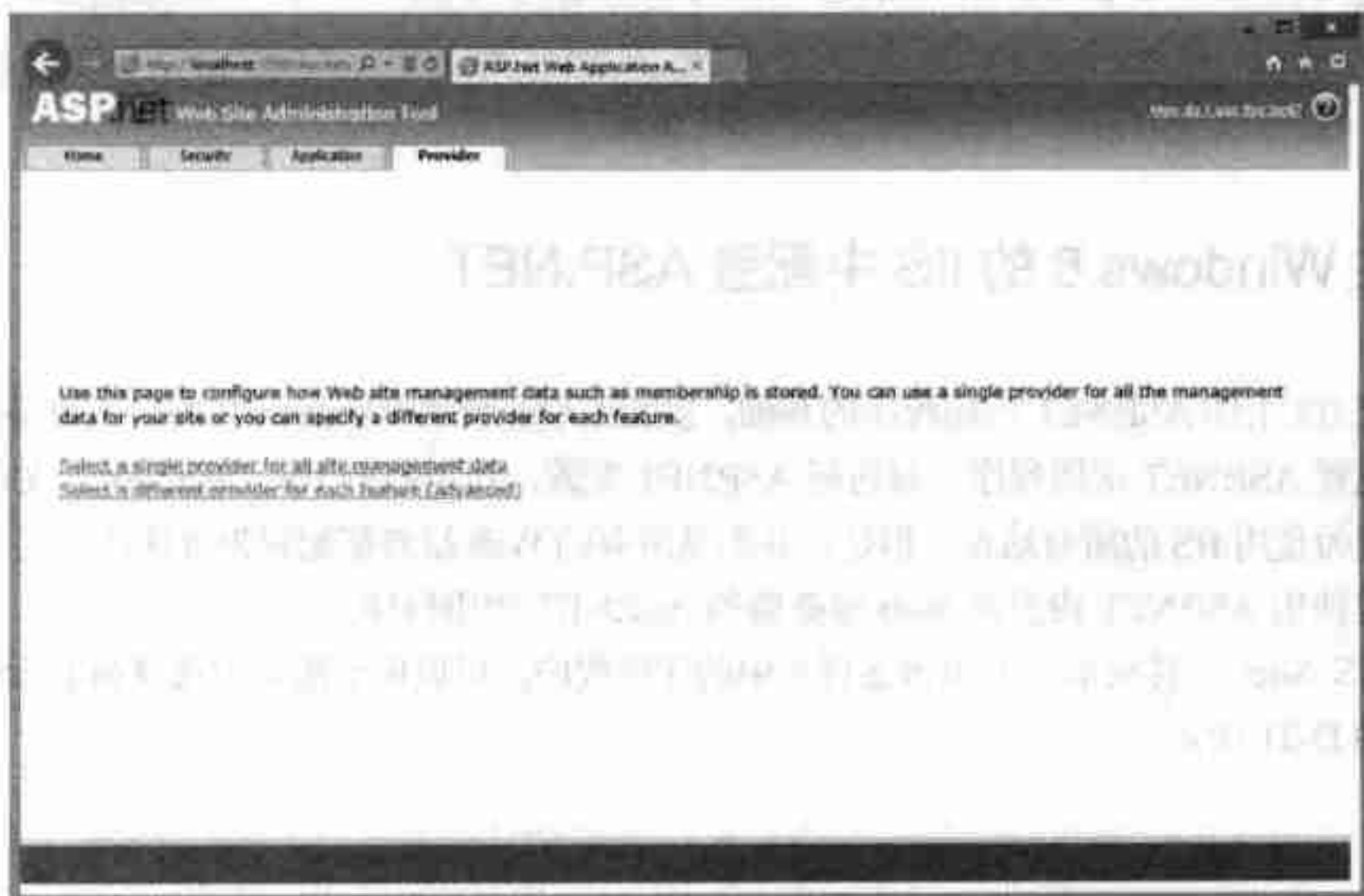


图 D-19

Provider 选项卡很简单,但包含一项重要信息:应用程序要使用的默认数据提供程序。在图 D-19 中,将应用程序设置为使用默认数据提供程序。

使用这个选项卡中的两个链接可以建立数据提供程序,或者为 ASP.NET 中需要数据提供程序的每个功能建立特定的数据提供程序。如果单击后一个链接,就会显示如图 D-20 所示的屏幕,在其中可以为成员资格和角色管理分别选择可用的提供程序。

从屏幕截图和给出的简短解释中可以看出,可以通过 GUI 处理大部分必需的配置。不再需要在 web.config 文件中指定必须有哪些设置。当 web.config 文件增大时,这个功能将越来越重要。在 ASP.NET 1.0/1.1 中,web.config 文件的大小还算合理,但加上 ASP.NET 2.0 或 3.5 提供的所有功能,web.config 文件就可能非常大。相反,与 ASP.NET 1.0/1.1 一样,ASP.NET 4.5 中的 web.config 文件在默认情况下尺寸很小。这些基于 GUI 的工具是配置常见设置的一种有效方式。但是,有许多设置不能通过 Web 服务器管理工具来修改,因此在许多情况下仍然需要编辑 web.config 文件。

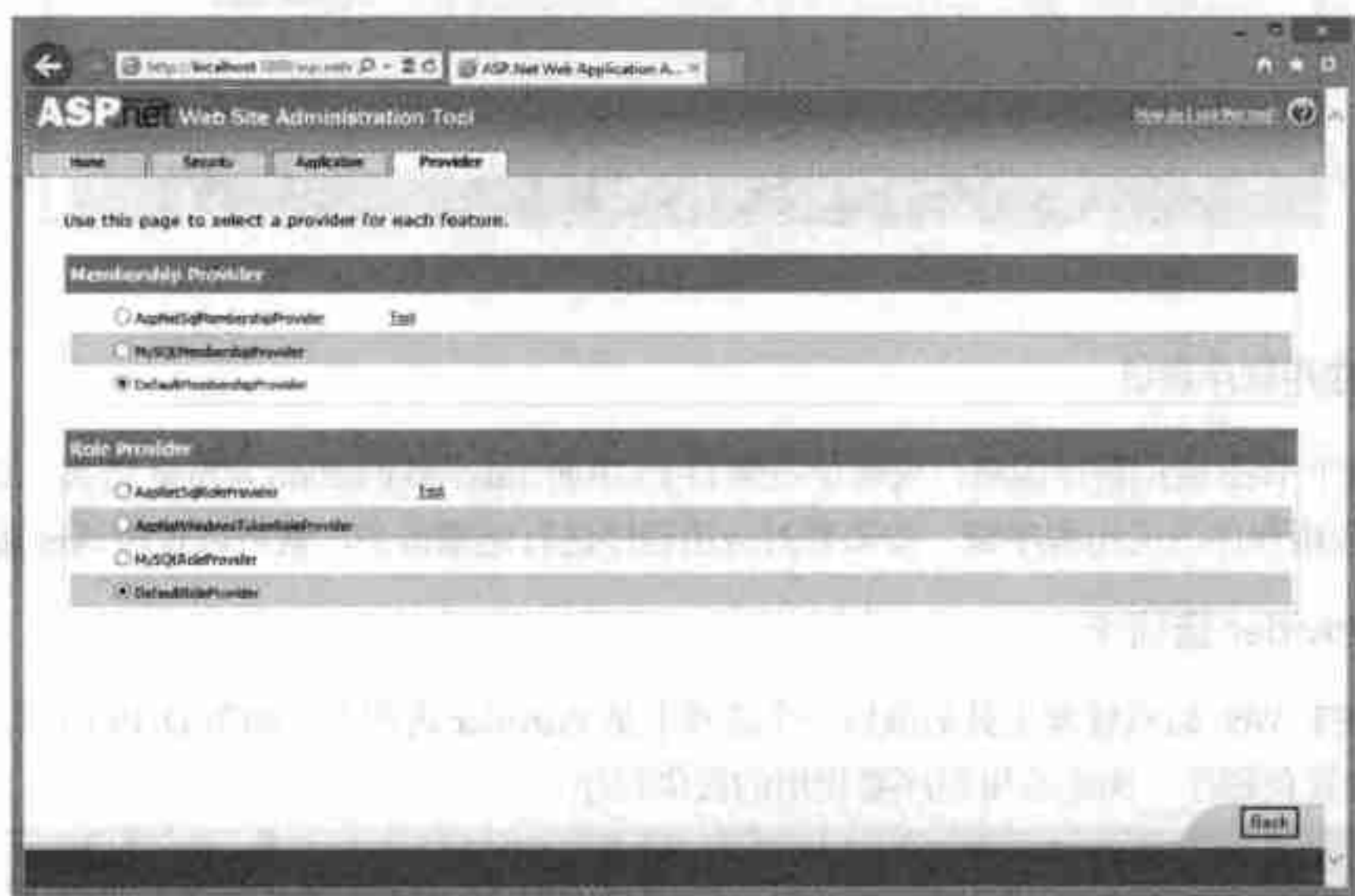


图 D-20

D.2 在 Windows 8 的 IIS 中配置 ASP.NET

如果把 IIS 用作 ASP.NET 应用程序的基础,就会发现在使用 Windows 8 时,很容易通过 IIS 管理器直接配置 ASP.NET 应用程序。要访问 ASP.NET 配置,可以打开 IIS,展开 Sites 文件夹,其中包含了配置为使用 IIS 的所有站点。但是,并不是所有的 Web 站点都配置为以这种方式工作,因为还可以创建使用 ASP.NET 内置的 Web 服务器的 ASP.NET 应用程序。

展开 IIS Sites 文件夹后,右击该文件夹中的应用程序,可以用于配置的选项将显示在 IIS 管理器中,如图 D-21 所示。



图 D-21

这些可用选项能完全配置 ASP.NET，甚至可以配置 IIS。本附录的重点是 ASP.NET 部分的选项。除了可以从可用图标中选择的选项之外，还可以在 IIS 管理器右边的 Actions 窗格中单击 Basic Settings 链接，配置应用程序的一些基本设置。单击 Basic Settings 链接后，会打开 Edit Site 对话框，如图 D-22 所示。



图 D-22



在 IIS 管理器中进行的修改会应用于应用程序的 web.config 文件。

通过这个对话框可以修改如下项：

- **Web 站点名：**在图 D-22 中，Web 站点名是 ProASPNET45，表示 URL 是 http://[IP 地址或域名]/ProASPNET45。
- **应用程序池：**用于应用程序的应用程序池。注意，默认有两个选项：Default AppPool(使用 .NET Framework 4.5 和集成的管道模式)和 Classic .NET AppPool(使用 .NET Framework 4.5 和传统的管道模式)。这个例子使用新的应用程序池 ProASPNET45。
- **物理路径：**ASP.NET 应用程序所在的文件夹位置，这里是 C:\ProASPNET45。

下面介绍可以通过 IIS 管理器中的图标使用的一些选项。

D.2.1 .NET Compilation

使用 Application 选项卡可以在应用程序上下文中进行针对页面的修改。在如图 D-23 所示的 .NET Compilation 对话框中(通过 IIS 管理器访问), 可以改变页面的编译和运行方式。还可以修改应用程序中的全局设置。

IIS 管理器的这一部分处理 ASP.NET 应用程序的编译和应用程序中一些页面的操作方式。Batch 部分处理应用程序的批处理编译——首先检查是否支持该功能, 再指定批处理的大小和编译所需的时间。

Behavior 部分处理编译是生成发布版本还是调试版本, 还有一些专门用于 VB 的编译指令, 用于确定在整个应用程序中是激活 Option Explicit 还是 Option Script。

General 部分关注的是引用的程序集, 如果打算把 App_Code 文件夹分解为几个已编译的实例(在同一应用程序中结合使用 VB 和 C#代码时需要执行该操作), 这部分还会关注代码的子目录。也可以指定在编译过程中使用的默认语言, 如 C#。

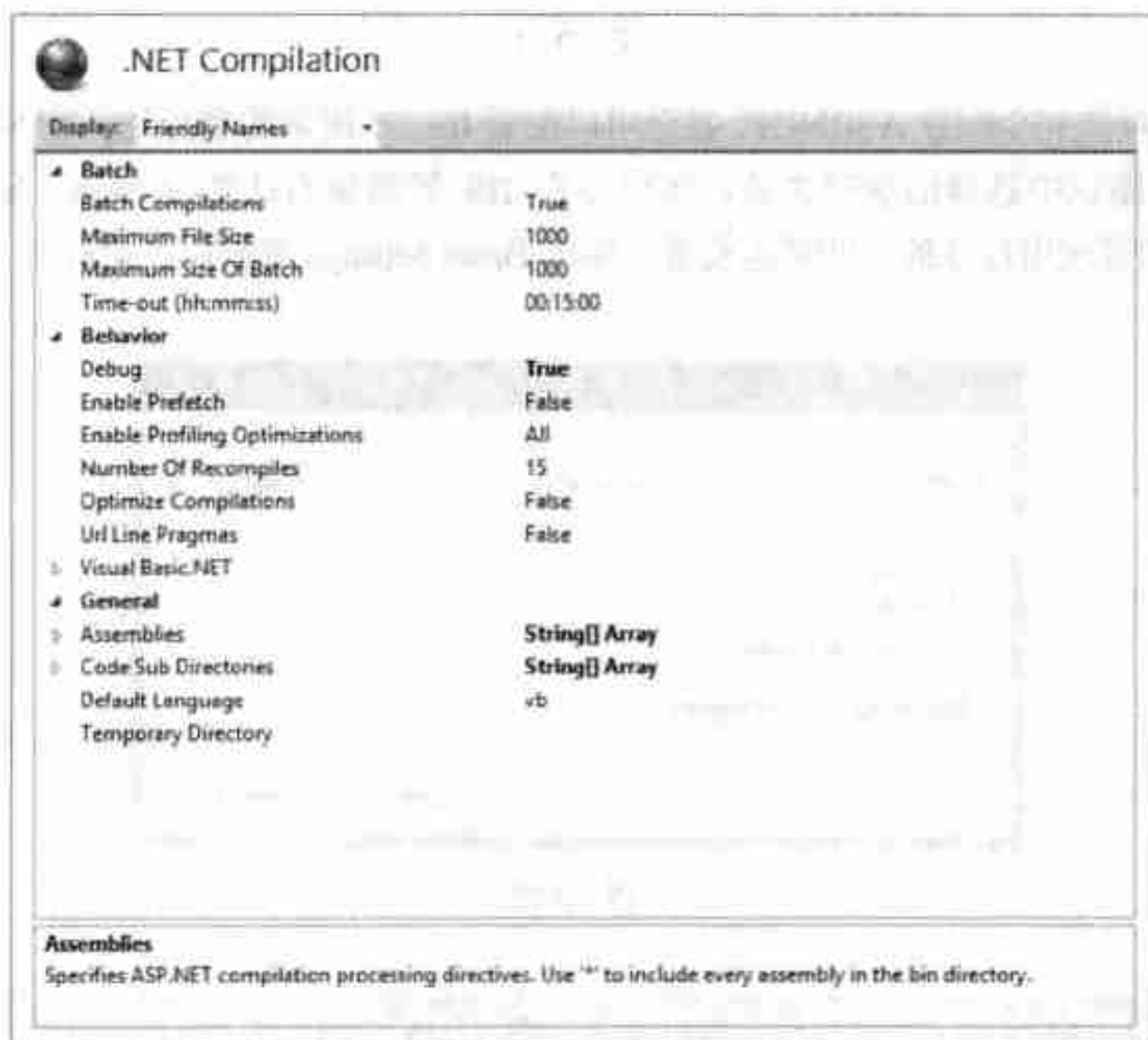


图 D-23

D.2.2 .NET Globalization

IIS 管理器的 .NET Globalization 对话框可以定制 ASP.NET 应用程序如何处理区域性以及请求和响应的编码。图 D-24 显示了这个对话框中的选项。

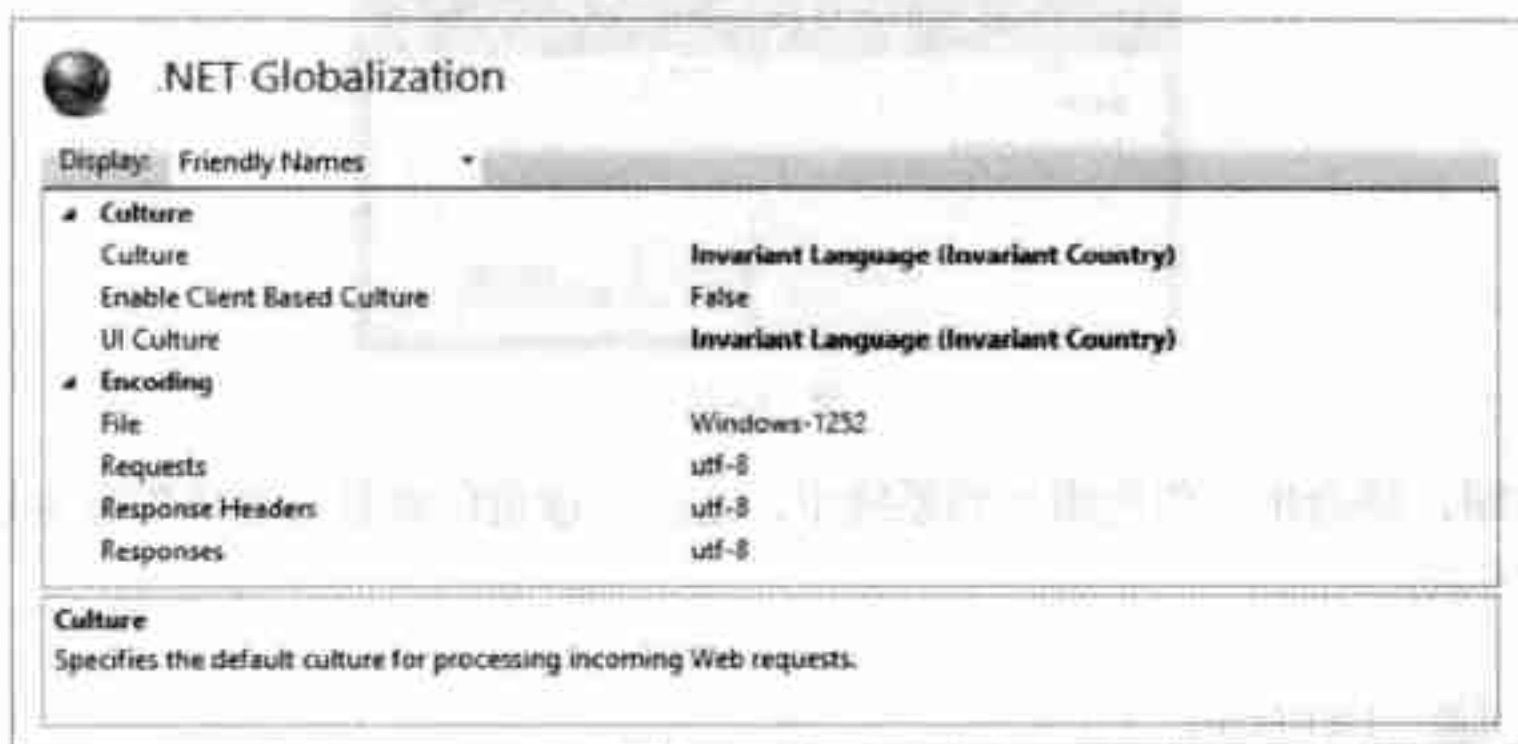


图 D-24

除了选择特定的 Culture 或 UI Culture 设置之外, 还可以选择 Auto Detect 选项, 它会选择可用的客户端区域性。默认情况下, 请求和响应的编码设置为 utf-8, 它适用于大多数基于拉丁语的语言。

D.2.3 .NET Profile

IIS 管理器的 .NET Profile 选项可以定制 ASP.NET 应用程序如何处理 ASP.NET 个性化系统。关于这个系统的讨论参见本书的第 18 章。图 D-25 显示了在个性化系统中添加新配置信息时打开的对话框。

如图 D-25 所示, 可以指定个性化属性的名称、所使用的数据类型、默认值、序列化的方式、是只读还是可用于匿名用户。为了更好地理解这些设置, 可查阅第 18 章。

除了建立在个性化系统中使用的属性之外, 还可以指定整个系统使用的提供程序。默认使用 `AspNetSqlProfileProvider` 提供程序, 如图 D-26 所示。通过单击 .NET Profile 部分的 Set default provider 链接, 可以打开该对话框。

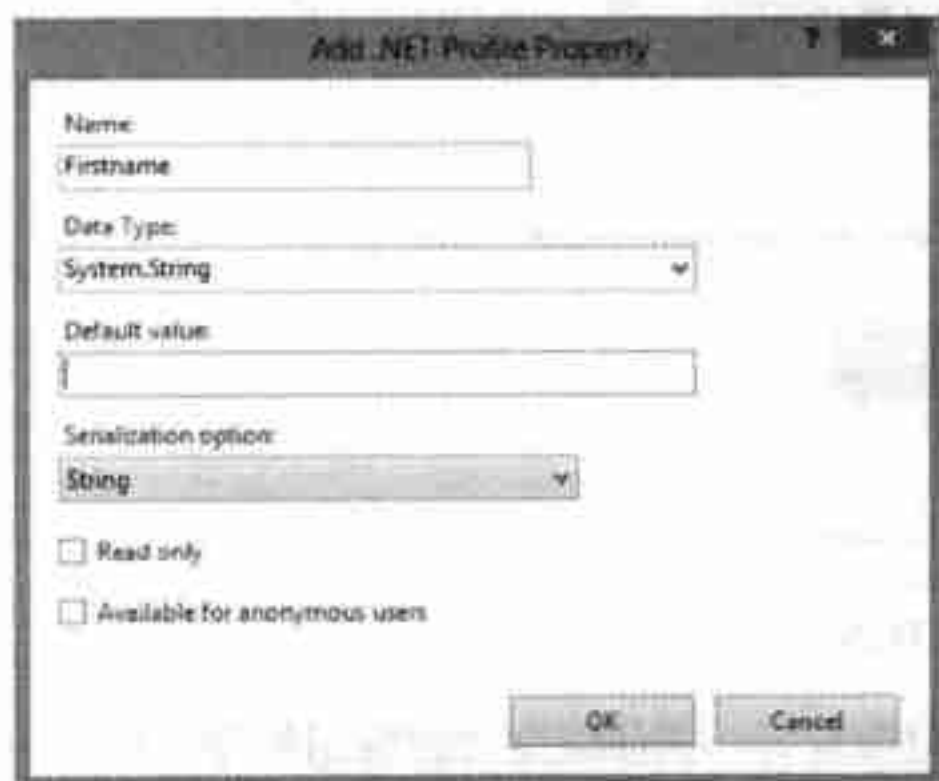


图 D-25



图 D-26

D.2.4 .NET Roles

使用 IIS 管理器的 .NET Roles 选项可以为应用程序添加角色, 进而激活基于角色的管理功能。图 D-27 是在单击 Actions 部分的 Add 链接后向应用程序中添加 Administrators 角色的示例。

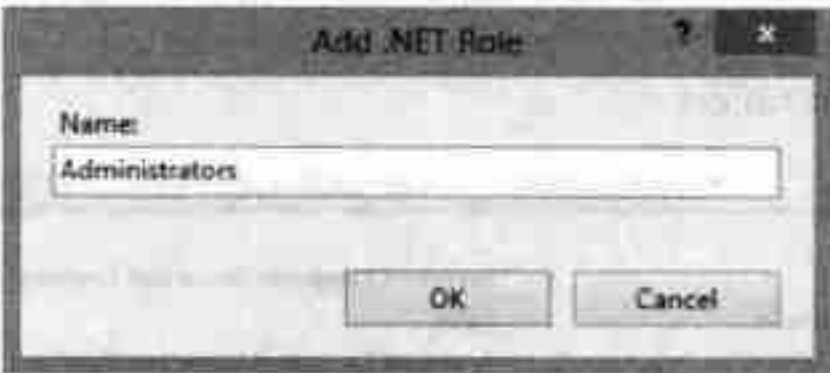


图 D-27

单击 OK 按钮，就会把该角色添加到系统中。之后，该角色就显示在该部分主屏幕的角色列表中，如图 D-28 所示。

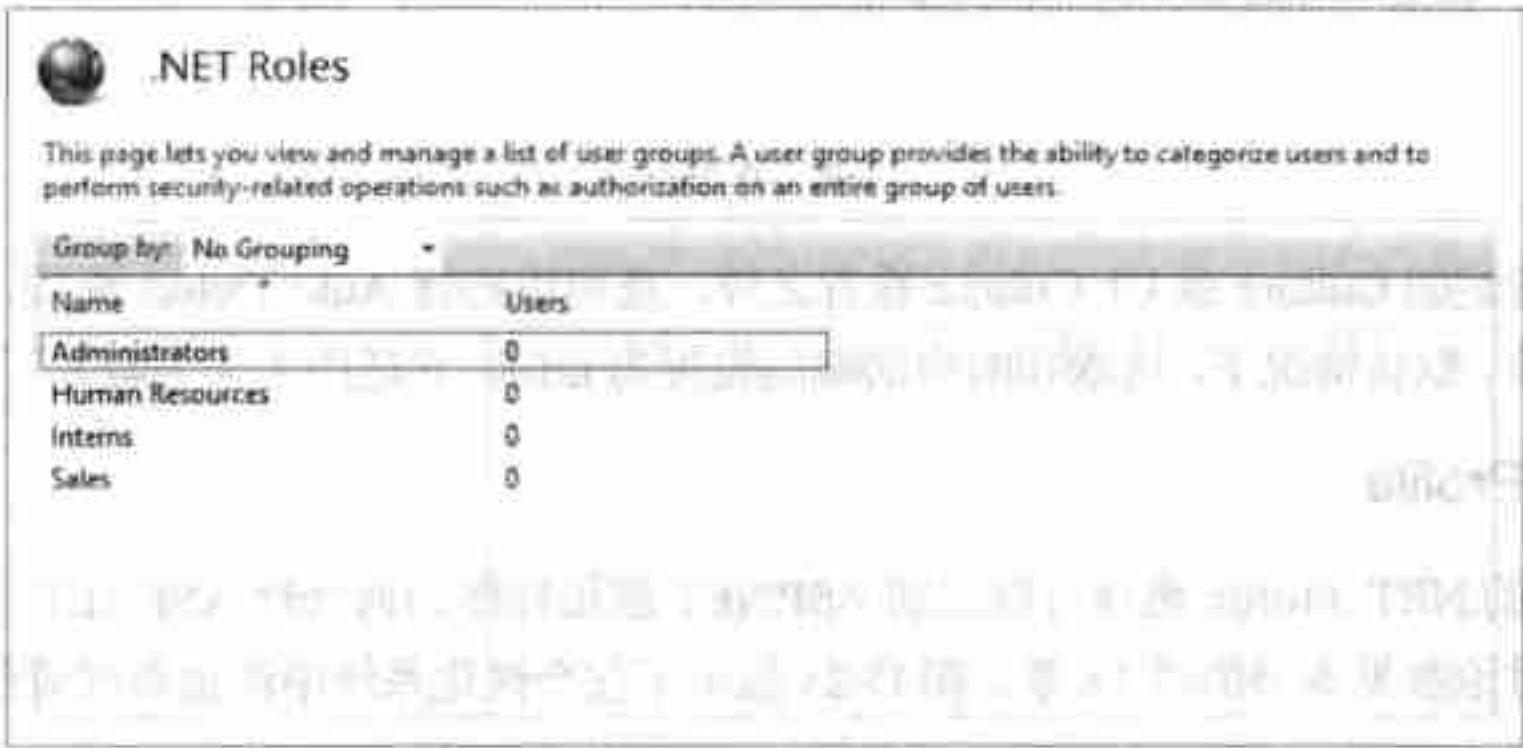


图 D-28

默认情况下，没有向该角色添加任何用户。但是，通过.NET Users 部分可以给角色添加用户。

D.2.5 .NET Trust Levels

使用 IIS 管理器的.NET Trust Levels 选项，可以通过选择指定的、预先生成的配置文件来指定应用于应用程序的安全级别，如图 D-29 中的选项列表所示。

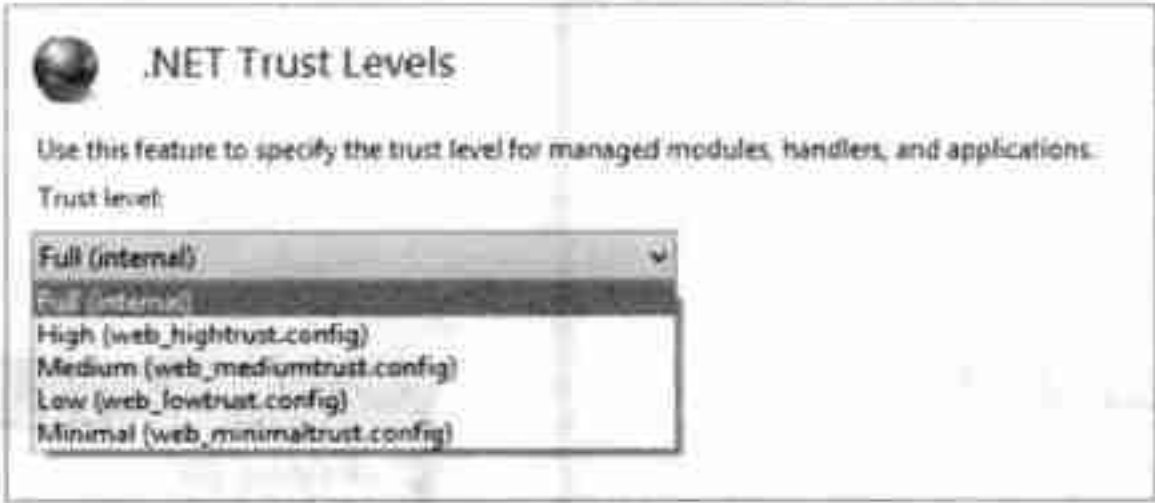


图 D-29

默认情况下，应用程序使用 web.config 文件，但指定另一信任级别会使应用程序使用另一个.config 文件。所有这些.config 文件都在 C:\Windows\Microsoft .NET\Framework\ v4.0.xxxxx\CONFIG 文件夹中。

D.2.6 .NET Users

使用 ASP.NET 成员资格系统(参见本书的第 19 章)的最简单方式是在 IIS 管理器的.NET Users 部分创建用户。通过提供的对话框可以很容易地添加用户，如图 D-30 所示。

图 D-30

如图 D-30 所示，可以在简单的向导中提供用户名、密码以及安全问题和答案。图 D-31 显示了向导的第二个屏幕。

图 D-31

在向导的第二个屏幕中，可以给用户赋予角色管理系统中包含的角色。本附录在前面创建了 Administrators 角色，因此可以给用户指定这个角色，因为该角色存在于系统中。

创建用户后，就可以在.NET Users 主屏幕上看到这个应用程序的完整用户列表，如图 D-32 所示。

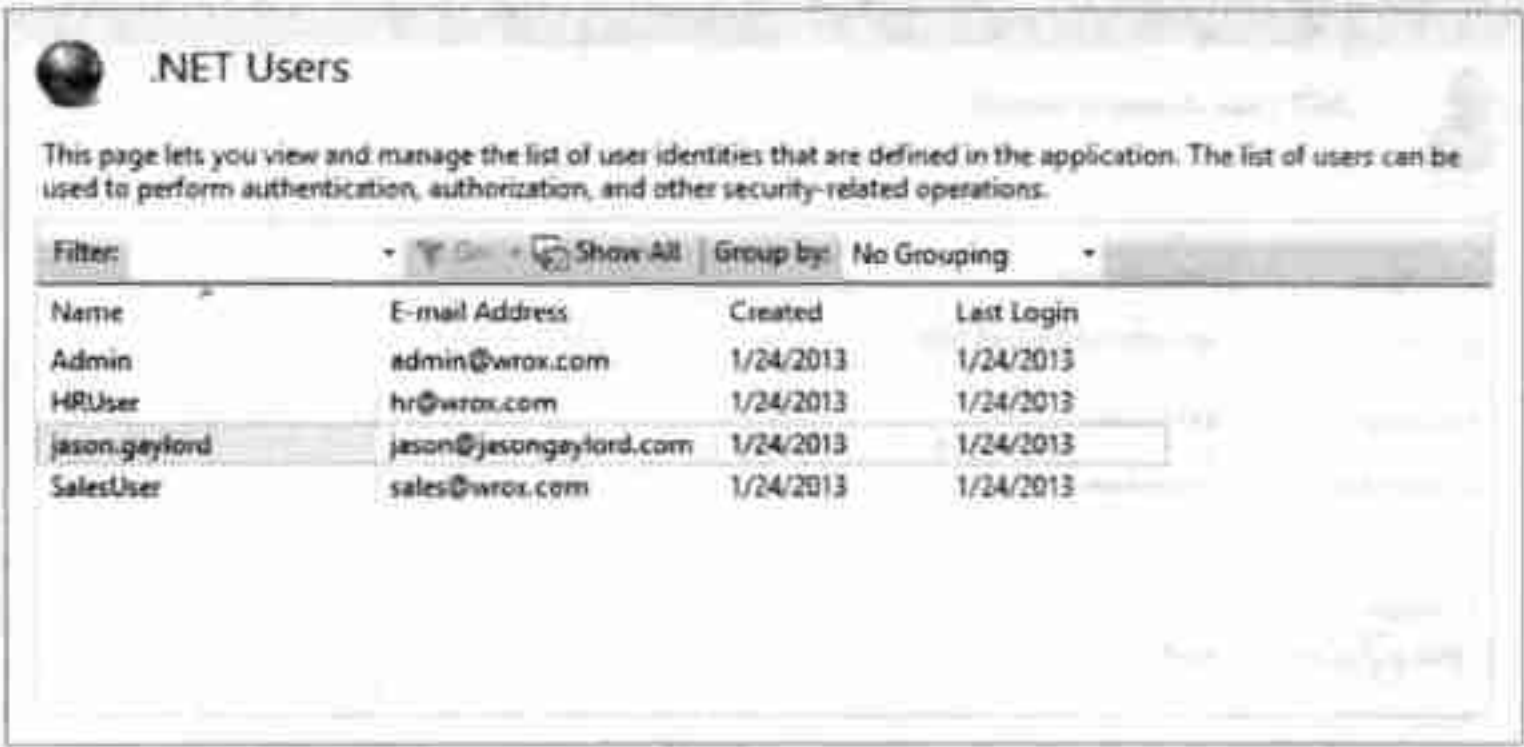


图 D-32

D.2.7 Application Settings

IIS 管理器的另一部分是 Application Settings。单击其中的 Add 或 Edit 按钮，会打开 Edit/Add Application Setting 对话框，如图 D-33 所示。

输入键/值对后，单击 OK 按钮。设置就会显示在主对话框的列表中，接着就可以编辑或删除应用程序中的设置。



图 D-33

D.2.8 Connection Strings

IIS 管理器的下一部分是 Connection Strings。要在应用程序中添加连接字符串，只需单击其中的 Add 按钮。还可以编辑或删除已有的连接字符串。图 D-34 显示了默认连接字符串 DefaultConnection 的 Edit Connection String 对话框。

添加新的连接也非常简单，如图 D-35 所示。



图 D-34



图 D-35

D.2.9 Pages and Controls

IIS 管理器的 Pages and Controls 部分处理控制应用程序中全部 ASP.NET 页面(.aspx)和用户控件(.ascx)的设置组。图 D-36 显示了这部分的可用设置。

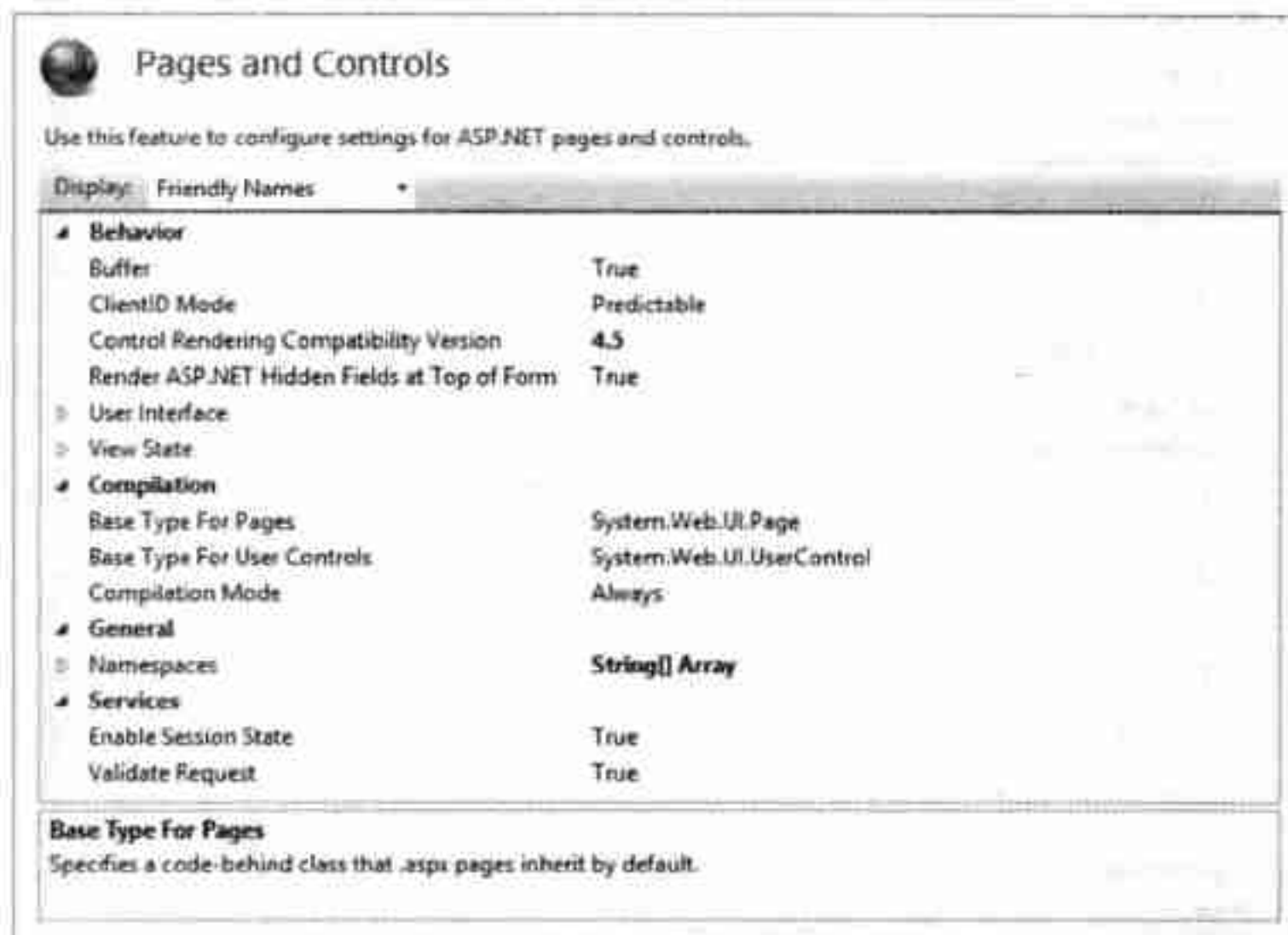


图 D-36

D.2.10 Providers

IIS 管理器的 Providers 部分处理在应用程序中定义的所有提供程序。在图 D-37 中, 只为 .NET Roles 引擎定义了 3 个提供程序: SQL Server 角色提供程序、Windows Token 角色提供程序和 MySQL 角色提供程序。

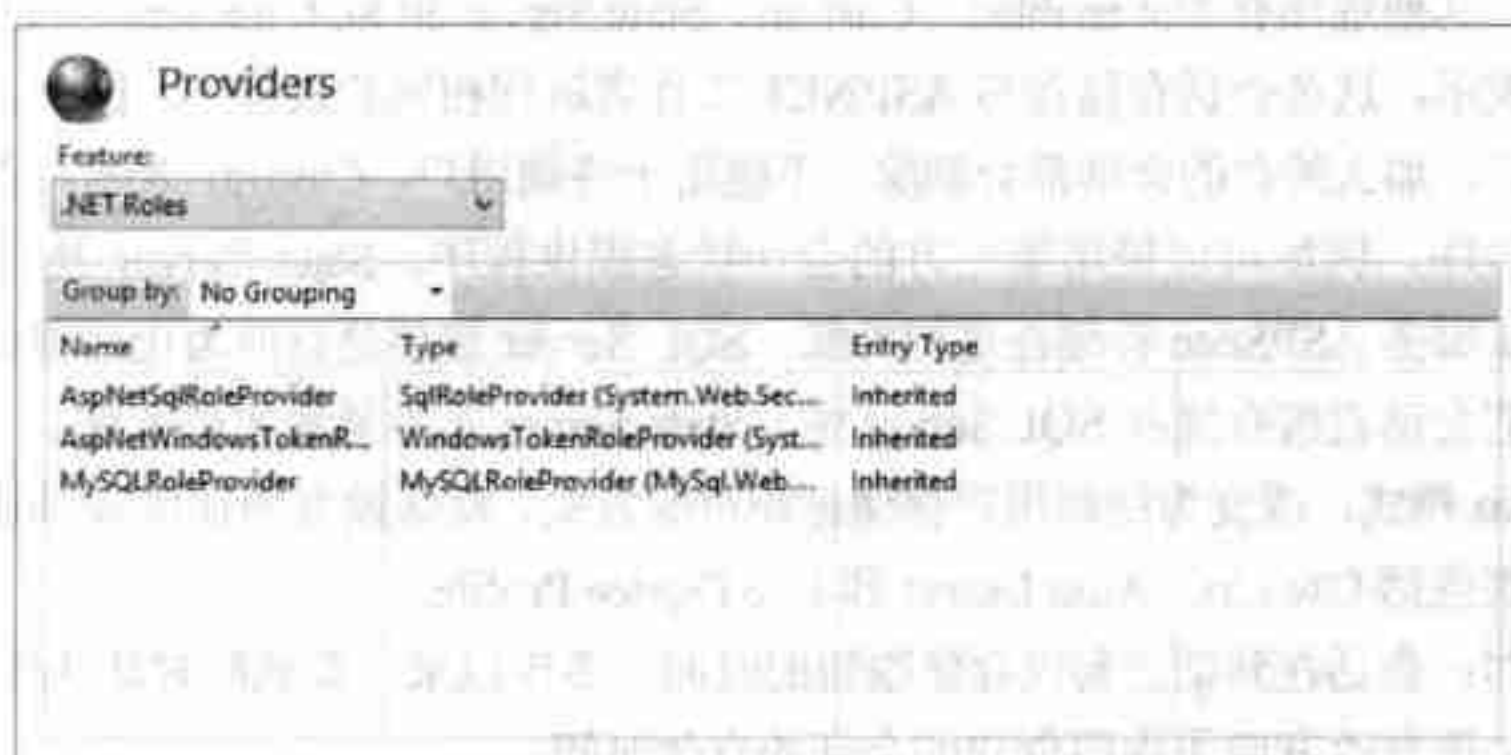


图 D-37

在对话框顶部的下拉列表框中选择相应的选项, 就可以查看 ASP.NET 中的所有其他引擎。

D.2.11 Session State

ASP.NET 应用程序在本质上是无状态的, 它们高度依赖于状态的存储方式。IIS 管理器的 Session

State 部分如图 D-38 所示, 在其中可以改变许多不同的设置, 这些设置确定了如何完成状态管理。



图 D-38

可以使用许多方式把状态管理应用于应用程序。这个对话框提供了许多不同的设置, 根据所选的内容启用或禁用其中一些设置。下面列出了 Session State Settings 部分的可用项:

- **会话状态模式:** 确定 ASP.NET 应用程序如何存储会话。默认选项(如图 D-38 所示)是 In Process, 其他选项有 Not enabled、Custom、State Server 和 SQL Server。在进程中(InProc)运行会话表示, 这些会话存储在与 ASP.NET 工作者进程相同的进程中。因此, 如果关闭 IIS, 再打开它, 那么所有的会话都会删除, 不能用于终端用户。Custom 表示可以指定定制的存储提供程序, 例如可以使用第三方的会话状态提供程序。State Server 选项表示, 会话由 Windows 服务 ASPState 存储在进程外部。SQL Server 选项是目前为止处理会话的最安全方式——把会话直接存储在 SQL Server 中。State Server 是性能最差的方法。
- **无 cookie 模式:** 改变为终端用户存储标识符的方式。默认设置为使用 cookie(Use Cookies)。其他设置包括 Use Uri、Auto Detect 和 Use Device Profile。
- **会话超时:** 会话在到期之前仅存储很短的时间。多年以来, 会话超时默认值都是 20 分钟。修改这个值会改变应用程序创建的会话的有效时间。

D.2.12 SMTP E-mail

如果需要处理发送电子邮件的应用程序, 就必须为此指定设置。在 IIS 管理器的 SMTP E-mail 部分定义使用 SMTP 发送电子邮件所需的设置, 如图 D-39 所示。

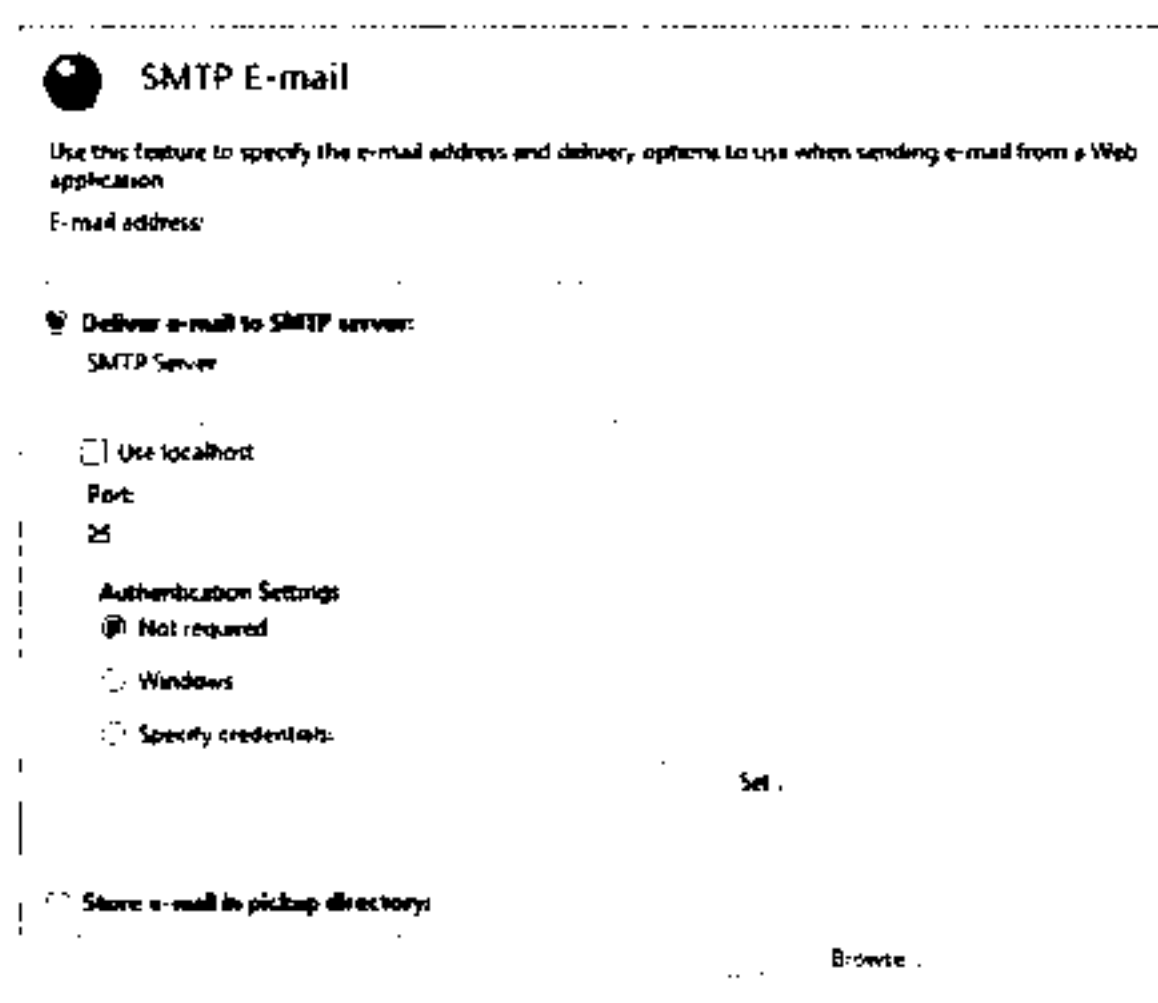


图 D-39

D.3 小结

本附录介绍了 ASP.NET 中的一些管理工具，这些工具使 web.config 文件更容易管理，因为它们会设置应用程序配置文件中的相应值。

Windows 8 中的 IIS Manager 控制台是广受欢迎的工具，可以管理配置为使用 IIS 的应用程序。ASP.NET Web 站点管理工具为管理员和开发人员提供了更多的选项，从而使他们能够更方便地管理设置。

动态类型与语言

微软最初引入.NET 时,许多开发人员,尤其是使用 Visual Basic 6(或更早版本)的开发人员,被带入静态类型语言的新世界。开发人员不再能够创建变体类型,也不能够在运行时使用后期绑定技术创建对象实例。新的基于.NET 的语言迫使他们使用熟悉的类型定义所有对象,如果编译器不能识别该类型,就会抛出错误。

一些.NET 追随者会辩论,.NET 确实支持后期绑定技术,而 VB.NET 也确实可以通过禁用 Option Strict 命令来实现后期绑定,但是在大多数情况下,这会令人气馁。在使用 C#时,如果想要模仿后期绑定,就必须借助于映射。这通常意味着需要编写更多的代码行,并且会加大应用程序的复杂性。

尽管静态类型语言有许多优点,例如严格的类型安全性和利用编译器基于常用类型的信息执行优化的能力,但是也确实损失了一些可以从动态类型语言那里获得的灵活性。另外,缺乏动态性确实使在.NET 和动态语言之间建立互操作变得更加困难。当需要创建要求 COM 互操作的应用程序时,你便会充分体验到这一点。

本附录将介绍微软为使开发人员接受动态语言的概念所做的工作,以及微软简化开发人员直接使用动态语言及与之进行互操作的过程。

E.1 隐式类型

当.NET Framework 引入隐式类型时,就增加了变量声明的灵活性。在 C#中使用 var 关键字表示隐式类型,而在 VB.NET 中则通过声明不带 As 操作符的变量来表示隐式类型。隐式类型允许声明变量,变量的类型是从用于初始化变量的表达式中隐式推断出来的。换句话说,隐式类型允许在代码中以相当动态的方式声明变量,同时可以通过编译器在运行时保留静态类型变量的优点。使用隐式类型的例子如下:

```
protected void Page_Load(object sender, EventArgs e)
{
    var foo = 1;
}
```


在这个例子中，将变量 `foo` 声明为隐式类型，并立即为之赋予数字值 1。因为为该变量赋予了数字值，所以编译器会自动地推断变量 `foo` 实际上应是 `Int32` 类型。事实上，如果将代码反编译成中间语言，就会发现由编译器输出的代码将变量发布为正确的类型。试着给该变量赋予另一不同类型的值，将导致抛出类型不匹配异常。

```
protected void Page_Load(object sender, EventArgs e)
{
    var foo = 1;
    foo = "abc";
}
```

尝试运行这段代码将导致产生编译器错误，表明不能将字符串赋予 `foo` 变量。

E.2 动态语言运行时

为了帮助开发人员获得一些灵活性，并将 .NET Framework 提供的优点带给更多的开发人员，在 2007 年，微软宣称已经在开始研究新的举措，称为动态语言运行时(Dynamic Language Runtime, DLR)。在发布 .NET 4.0 时，公共语言运行时(CLR)附带了 DLR，给 CLR 带来了动态类型系统、动态方法调度、动态代码生成和宿主 API。DLR 允许动态语言和 .NET 运行库一起运行。使用 DLR，这些动态语言获得了两种环境下的优点。它们运行在动态环境下，类型在运行期间确定，并可以访问更丰富的 .NET 基类库。IronPython 和 IronRuby 是两种流行的动态语言，目前它们采用 DLR。

IronPython 和 IronRuby 均在微软技术领域发展起来，用开源许可发布。IronPython 和 IronRuby 项目最初都是微软为提高 .NET Framework 对动态语言的支持而开发的，它们丰富了编程语言的种类。2010 年的冬季，微软把这些 Iron 项目的开发交给了社区。其他动态语言都是由第三方开发人员开发和维护的。



关于 IronPython 语言(Python 语言的 DLR 实现)的更多信息，请参考 www.IronPython.net。关于 IronRuby 语言(Ruby 语言的 DLR 实现)的更多信息，请参考 www.IronRuby.net。

由于基于 DLR 语言的开发与基于 CLR 语言的开发区别很大，对基于 DLR 语言的支持现在交给了社区，因此 Visual Studio 中对这种语言的支持相对缺乏，一些工具可以使 IronPython 和 IronRuby 与 Visual Studio 集成起来，但功能区别很大。

将语言加载到系统后，就可以使用这些语言开发独立的应用程序，或者利用其他静态语言(如 C#)的语言库。程序清单 E-1(本附录下载代码中的 IronPythonCS 项目中的 IronPythonList.aspx)演示了如何在使用 C# 编写的 ASP.NET 应用程序中使用 IronPython 代码。注意要执行示例代码，需要安装 IronPython 2.7，并引用默认安装位置 `Program Files (x86)\IronPython 2.7\Platforms\Net40` 的程序集 `Microsoft.Scripting.dll` 和 `IronPython.dll`。

程序清单 E-1 从 ASP.NET 中调用 IronPython 库

```

<%@ Page Language="C#" %>

<!DOCTYPE html>

<script runat="server">
int[] items = Enumerable.Range(1, 5).ToArray();
int[] subitems = Enumerable.Range(1, 7).ToArray();
dynamic random;

protected void Page_Load(object sender, EventArgs e)
{
    System.IO.Directory.SetCurrentDirectory(
        Environment.GetFolderPath(
            Environment.SpecialFolder.ProgramFiles) + @"IronPython 2.7\Lib");

    Microsoft.Scripting.Hosting.ScriptRuntime py =
        IronPython.Hosting.Python.CreateRuntime();
    random = py.UseFile("random.py");

    this.Repeater1.DataSource = items;
    this.Repeater1.DataBind();
}

protected void Repeater1_ItemDataBound(object sender, RepeaterItemEventArgs e)
{
    Label lbl = (Label)e.Item.FindControl("Label");
    lbl.Text = string.Format("List Number: {0}", e.Item.ItemIndex);

    BulletedList list = (BulletedList)e.Item.FindControl("BulletedList");
    random.shuffle(subitems);
    list.DataSource = subitems;
    list.DataBind();
}
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Repeater ID="Repeater1" runat="server"
                OnItemDataBound="Repeater1_ItemDataBound">
                <ItemTemplate>
                    <p>
                        <asp:Label runat="server" ID="Label" />
                        <br />
                        <asp:BulletedList runat="server" ID="BulletedList" />
                    </p>
                </ItemTemplate>
            </asp:Repeater>
        </div>
    </form>

```



```

    </form>
</body>
</html>

```

从这个程序清单中可以看出，当加载页面时，使用 DLR 的宿主 API 加载 IronPython 安装目录下的 IronPython 库 random.py。因为 items 数组中的每一项都绑定到 Repeater，所以使用 random.py 库将子项以随机顺序混乱摆放。

E.3 动态查找

在 C# 4 中，添加了一项新功能，称为动态查找。虽然 .NET 开发人员很熟悉使用反射进行后期绑定的功能，但动态查找功能给 C# 带来了真正的动态类型声明机制，它允许在代码中显式地将变量声明为动态类型，并在运行时针对该类型动态地调用方法。动态类型与隐式类型不同，因为动态类型甚至在运行期间也保持真正的动态性，而隐式类型则在编译时转换为静态类型。下面的代码显示了使用新的 dynamic 关键字的简单示例：

```

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        dynamic dynvalue =
            "Even though assigned a string, the type is still dynamic";
    }
</script>

```

在这个示例中，给 dynvalue 属性赋予一个简单字符串作为值；然而与隐式类型不同的是，变量在运行时仍然是动态类型。在 Visual Studio 中尝试访问该变量的任意成员时就会看出这一点。通常情况下，Visual Studio 会显示可用属性和方法的列表以及变量的类型，但是由于类型是动态的，因此直到运行时才能知道这些信息。Visual Studio 会使用工具提示告诉你这一点，如图 E-1 所示：

```

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        dynamic dynvalue = "Even though assigned a string, the type is still dynamic";
        dynvalue.
    }
</script>

```

(dynamic expression)
This operation will be resolved at runtime.

图 E-1

在运行期间，动态语言的动态调度系统使用动态调用执行该类型对象的属性和方法，这意味着可以在运行期间添加或删除类型对象的成员。.NET 为此提供了两种机制：ExpandObject 类和 DynamicObject 类。

ExpandObject 类可用于需要动态添加或删除成员的相对简单的情况。下面的代码演示了 ExpandObject 类的用法：

```

dynamic contact = new System.Dynamic.ExpandoObject();
contact.Name = "John Doe";
contact.Phone = "201-555-5555";

```



```

contact.Address = new System.Dynamic.ExpandoObject();
contact.Address.Street = "123 Main St";
contact.Address.City = "Anywhere";
contact.Address.State = "WA";
contact.Address.Postal = "12345";

```

在这个程序清单中，创建了一个 `ExpandoObject` 对象并添加了几个属性。在运行时动态添加这些属性，并在内部将它们存储为 `IDictionary<String, Object>`，`ExpandoObject` 类在内部实现该字典以维护成员列表。

`ExpandoObject` 对象是密封的，不能进一步继承。如果想要获得针对动态对象执行的操作的更多控制，或者当诸如获取和设置属性的操作发生或方法调用发生时，想要更多地控制发生的情况，可以创建继承于 `DynamicObject` 的对象。下面的代码显示了从 `DynamicObject` 类派生 `JsonObject` 类的例子(本附录下载代码中 `DynamicRest` 项目的 `DynamicRest/JsonObject.cs`)。 `DynamicObject` 类的部分代码如下：

```

namespace DynamicRest {

    public sealed class JsonObject :
        DynamicObject,
        IDictionary<string, object>,
        IDictionary<string, object>
    {

        private Dictionary<string, object> _members;

        public JsonObject()
        {
            _members = new Dictionary<string, object>();
        }

        public JsonObject(params object[] nameValuePairs) : this() {
            if(nameValuePairs != null) {
                if(nameValuePairs.Length % 2 != 0) {
                    throw new ArgumentException(
                        "Mismatch in name/value pairs.");
                }

                for(int i = 0; i < nameValuePairs.Length; i += 2) {
                    if(!(nameValuePairs[i] is string)) {
                        throw new ArgumentException(
                            "Name parameters must be strings.");
                    }

                    _members[(string)nameValuePairs[i]] =
                        nameValuePairs[i + 1];
                }
            }
        }

        public override bool TryConvert(ConvertBinder binder,
            out object result)
        {

```

```

        Type targetType = binder.ReturnType;

        if((targetType == typeof(IEnumerable)) ||
            (targetType ==
                typeof(IEnumerable<KeyValuePair<string, object>>)) ||
            (targetType == typeof(IDictionary<string, object>)) ||
            (targetType == typeof(IDictionary)))
        {
            result = this;
            return true;
        }

        return base.TryConvert(binder, out result);
    }

    public override bool TryDeleteMember(DeleteMemberBinder binder)
    {
        return _members.Remove(binder.Name);
    }

    public override bool TryGetMember(GetMemberBinder binder,
                                      out object result)
    {
        object value;
        if(_members.TryGetValue(binder.Name, out value))
        {
            result = value;
            return true;
        }

        return base.TryGetMember(binder, out result);
    }

    public override bool TrySetMember(SetMemberBinder binder,
                                      object value)
    {
        _members[binder.Name] = value;
        return true;
    }

    // ++++
    // Non-related interface implementations removed for clarity
}

```

JsonObject 类是某个较大型库的一部分，这个库基于一个由 Nikhil Kothari 编写的样本，该样本可以简化 JSON 格式数据的检索和分析。在 .NET Framework 4.5 中，添加了 System.Json 名称空间，从而可以更容易地处理 JSON 数据。但在此之前，要在 .NET 中处理 JSON 数据，就必须创建映射 JSON 结构的代理类型，然后执行复杂的分析操作：将 JSON 数据分析为定制类型的集合。使用最初在 C# 4 中引入的动态功能，可以将 JSON 数据分析为泛型以简化这个过程，在运行期间推断泛型，通过动态属性和方法提供数据。在下面的代码示例中，可以重写 TryGetMember 和 TrySetMember 方法，以控制获取和设置类型属性的方式。在这个例子中，成员存储在 JsonObject 类的内部对象

Dictionary 中。

微软开发小组还使用 .NET Framework 4.5 中的动态功能来简化 COM 互操作。Office Primary Interop Assemblies (PIA) 针对 Office COM Automation API 提供了托管层，该托管层已更新为利用 C# 中的动态功能。下面的代码示例(本附录下载代码中的 `ExcelInterop.cs`)演示了如何使用 PIA 从 .NET 中与 Excel 进行交互。

```
var excelApp = new Microsoft.Office.Interop.Excel.Application();
excelApp.Visible = true;
excelApp.Workbooks.Add();

Microsoft.Office.Interop.Excel.Worksheet workSheet = excelApp.ActiveSheet;
workSheet.Cells[1, "A"] = "ID Number";
workSheet.Cells[1, "B"] = "Current Balance";

var row = 1;
foreach (var acct in accounts)
{
    row++;
    workSheet.Cells[row, "A"] = acct.ID;
    workSheet.Cells[row, "B"] = acct.Balance;
}

workSheet.Columns[1].AutoFit();
workSheet.Columns[2].AutoFit();
```

在 PIA 以前的版本中，访问某些 API(如示例中显示的 `Columns` 集合)会要求将返回的对象强制转换为相应类型，以访问该类型的属性和方法。然而，新版本中使用 `dynamic` 关键字取消了这种需求，并允许对这些属性和方法进行动态查找。

E.4 小结

微软继续扩展 .NET Framework 的功能和语言，通过投入更多的功能来给语言注入更多的动态性。这些功能给我们提供了更多的编程工具，允许我们使用对特定应用程序最有意义的功能或编程语言。从 C# 和 VB.NET 的内置功能(如隐式类型和动态查找)到全新的编程语言(如 IronPython 和 IronRuby)，可用的选项还在继续扩大。

ASP.NET 联机资源

F.1 作者的博客和 Twitter ID

Jason Gaylord: jasongaylord.com

@jgaylord

Scott Hanselman: www.hanselman.com/blog/

@shanselman

Todd Miranda: xperimentality.com

@tmiranda

Pranav Rastogi: blogs.msdn.com/b/pranav_rastogi/

@rustd

Christian Wenz: www.hauser-wenz.de/blog

@chwenz

F.2 ASP.NET 方面有影响力的博客

Scott Guthrie: weblogs.asp.net/scottgu/

Rick Strahl: www.west-wind.com/weblog/

K. Scott Allen: odetocode.com/blogs/scott/

Phil Haack: www.haacked.com/

Steve Smith: www.stevesmithblog.com/

G. Andrew Duthie: <http://devhammer.net/>

Scott Mitchell: scottonwriting.net/sowBlog/

Nikhil Kothari: www.nikhilk.net/

F.3 Web 站点

123ASPX Directory: www.123aspx.com
4 Guys from Rolla: www.4guysfromrolla.com
ASP Alliance: aspalliance.com
ASP Alliance Lists: aspadvice.com
The ASP.NET Developer Portal: msdn.microsoft.com/asp.net
ASP.NET Homepage: www.asp.net
ASP.NET Resources: www.aspnetresources.com
ASP.NET World: www.aspnetworld.com
International .NET Association: www.ineta.org
Microsoft's ASP.NET AJAX Site: www.asp.net/ajax/
Microsoft's ASP.NET MVC Site: www.asp.net/mvc/
Microsoft's ASP.NET Web API Site: <http://www.asp.net/web-api>
Microsoft Developer Centers: msdn.microsoft.com/developercenters
Microsoft Community Site: answers.microsoft.com
Microsoft's Open Source Project Community: www.codeplex.com
RegExLib: www.regexlib.com
XML for ASP.NET: www.xmlforasp.net
Stackoverflow: stackoverflow.com
Channel 9: channel9.msdn.com

F.4 Twitter 上值得关注的人员

@scottgu
@haacked
@brada
@ambroselittle
@sondreb
@chrislove
@wtrox
@dseven
@randyholloway
@migueldeicaza
@donxml
@moon
@kvgros
@richcampbell

@christoc
@csells
@keyvan
@danwahlin
@devhammer
@jglozano
@shawnwildermuth
@julielermanvt
@codinghorror
@spolsky
@elijahmanor
@robconery
@jeremydmiller
@angrycoder
@rickstrahl
@csharpfritz
@DamianEdwards
@davidfowl
@coolcsh

使用 NuGet 扩展 Visual Studio

多年来，.NET 开发者社区开发了大量可重用的库和实用工具，以帮助完成常见的任务。这些第三方可重用库中的许多都进入了.NET 开发人员的主工具箱。另外，.NET 日益成为开源开发的平台。其中一些最流行的库已发布为开源项目。

在.NET 开发的所有方面中，可重用库无论开源与否，都成为很有价值的功能源。这在 Web 开发中更加明显，Web 开发使用第三方.NET 和 JavaScript 库的比比皆是。实际上，Visual Studio 中的许多项目模板都包含第三方库。

要把这些库合并到项目中，只需包含对一个或多个程序集的引用。但这常常不那么简单。如果不知道哪个库提供了需要的功能，就需要找到该库。一旦找到这个库，就需要下载它。还常常需要从多个版本或选项中选择正确的下载。下载正确的文件后，不要忘了解压 ZIP 文件。应使用下载页面提供的散列验证文件。接着需要把文件解压缩到解决方案的一个文件夹中。我们常常创建 lib 文件夹来解压缩文件。在 Visual Studio 的 Solution Explorer 中，需要添加对新解压的程序集的引用。现在开始在 app.config 或 web.config 文件中设置正确的配置，编写库需要的初始化文件。确定正确的配置设置常常需要阅读文档，或再次搜索论坛和博客。

开发这些库时，它们会合并并依赖其他库，而其他库又可能需要配置。依赖得越多，配置要求也越多。

NuGet 是 Visual Studio 的扩展，用于简化添加、删除和更新 Visual Studio 项目中使用 .NET Framework 的库和工具。

NuGet 允许建立库的开发人员打包他们的库，存储在资源库中，以方便他人寻找。包是.nupkg 文件，其中包含在解决方案中安装和配置库的所有内容。资源库中的所有包都捆绑到种子，种子可以在 NuGet 网站上访问，或者直接通过 Visual Studio 访问。寻找库的开发人员可以搜索种子，通过命令行界面或图形化界面安装包。

G.1 在 Visual Studio 中使用 NuGet

在 Visual Studio 2012 之前，NuGet 要使用 Visual Studio Extension Manager 安装。但是，NuGet 客户端已安装在 Visual Studio 2012 中。为了验证 NuGet 已安装，以及在需要时安装 NuGet，可以使

用 Visual Studio Extension Manager。图 G-1 是 Visual Studio 中的 Extension Manager，其中显示了目前安装的扩展。

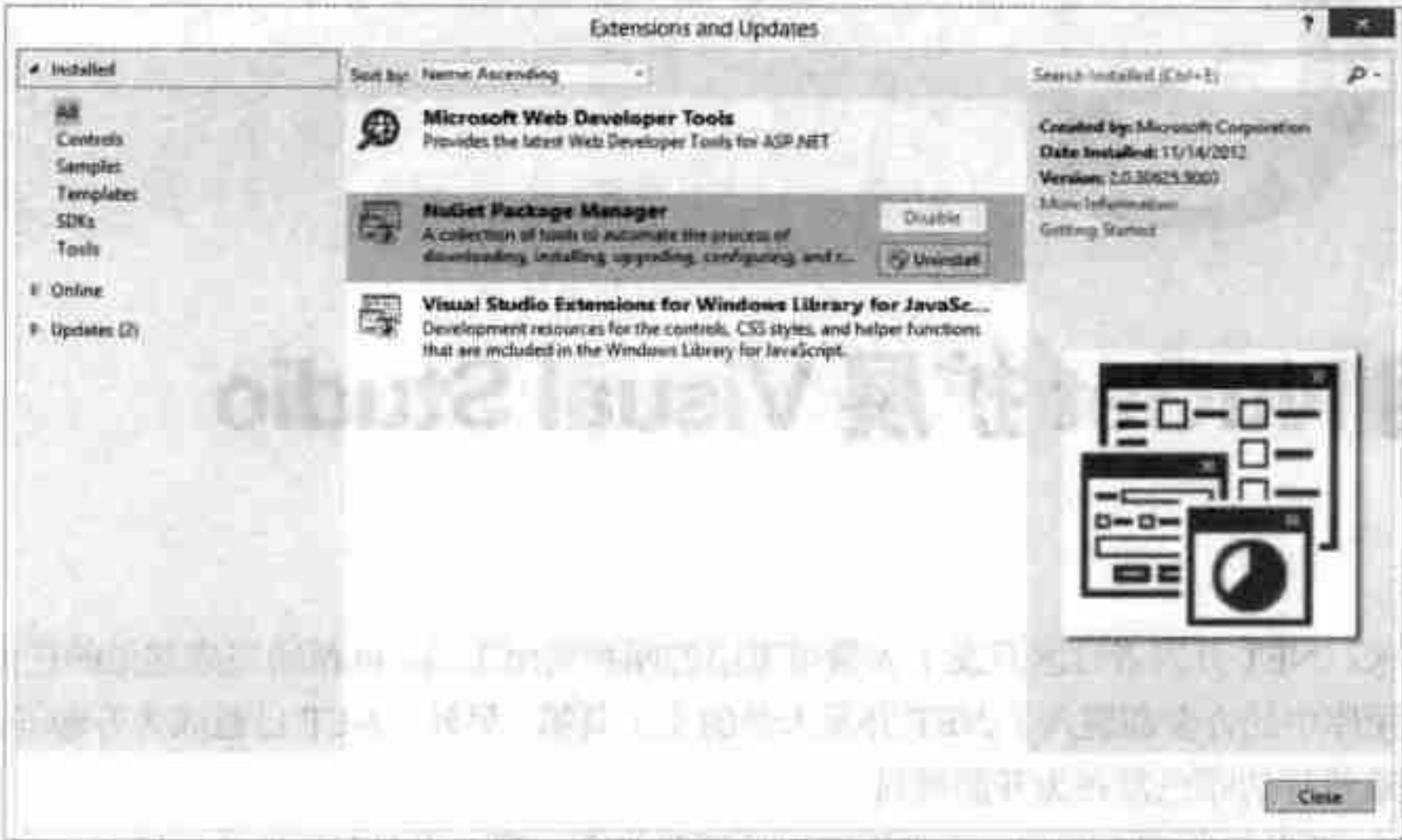


图 G-1

Extension Manager 还用于在 NuGet 有新版本可用时更新扩展。如果有 NuGet 的更新版本，就可以在 Extension Manager 的 Updates 选项卡中看到。图 G-2 中的 Extension Manager 显示了 NuGet 的更新版本。该更新版本可以在这个界面中直接安装。

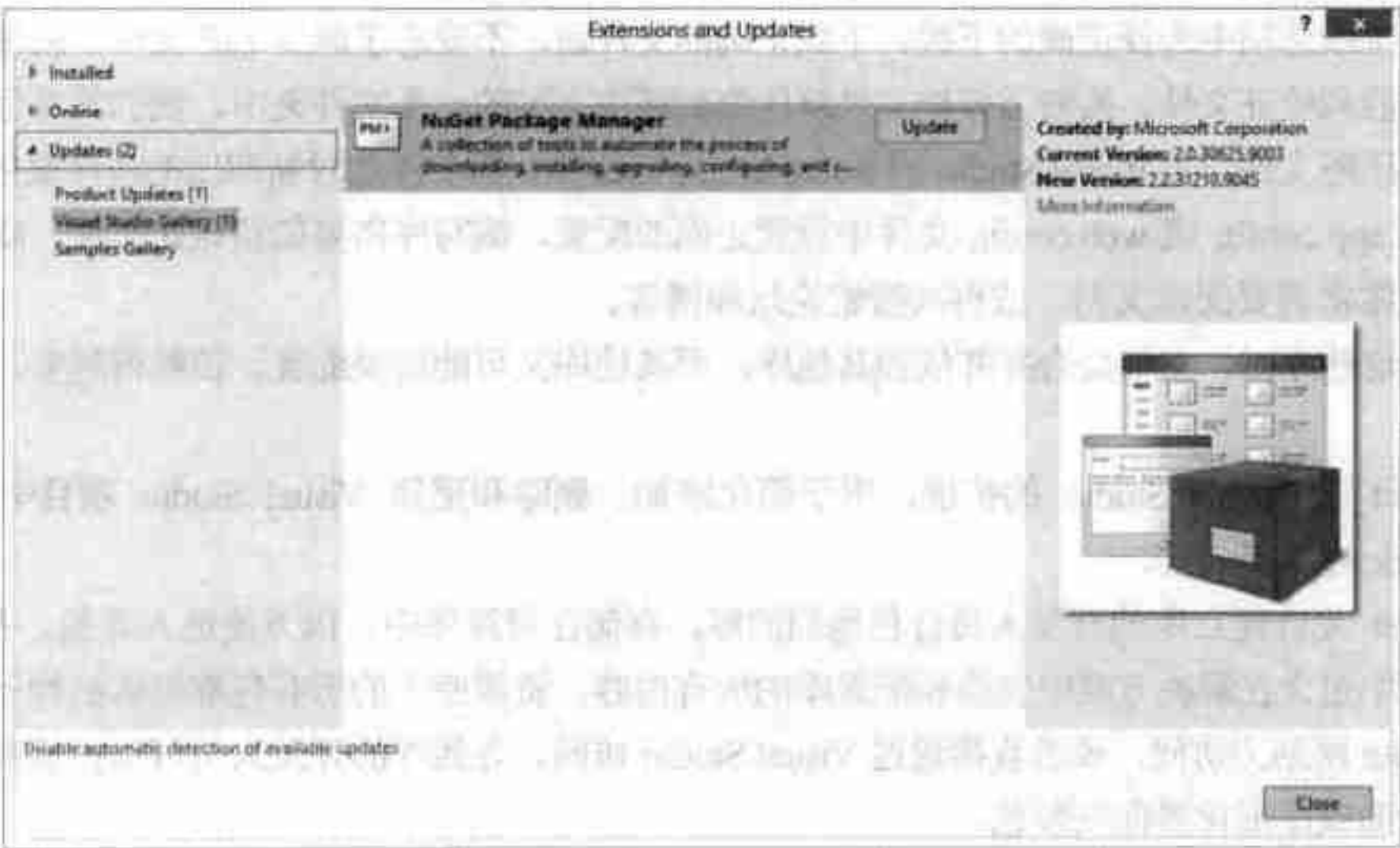


图 G-2

安装了 NuGet 后，就可以使用 Manage NuGet Packages 窗口或 Package Manager Console 中的 PowerShell 命令查找、安装和管理包。

G.1.1 用窗口管理 NuGet 包

Visual Studio 的 NuGet 客户端以 Manage NuGet Packages 窗口的形式提供了图形化界面。为了使

用这个界面，需要把一个解决方案加载到 Visual Studio 中。对于本例，创建一个新的 ASP.NET MVC 4 Web Application 项目，选择 Internet Application 作为模板类型。为了打开 Manage NuGet Packages 窗口，应右击项目，选择 Manage NuGet Packages。图 G-3 显示了 Manage NuGet Packages 窗口。

注意已经加载了许多包。Visual Studio 中的许多项目模板也使用 NuGet 包，在新项目中包含某些第三方和开源库。如图 G-3 所示，窗口左边包含的选项卡允许开发人员查看当前安装的包和可安装的包。还可以看出已安装的包是否有更新版本。

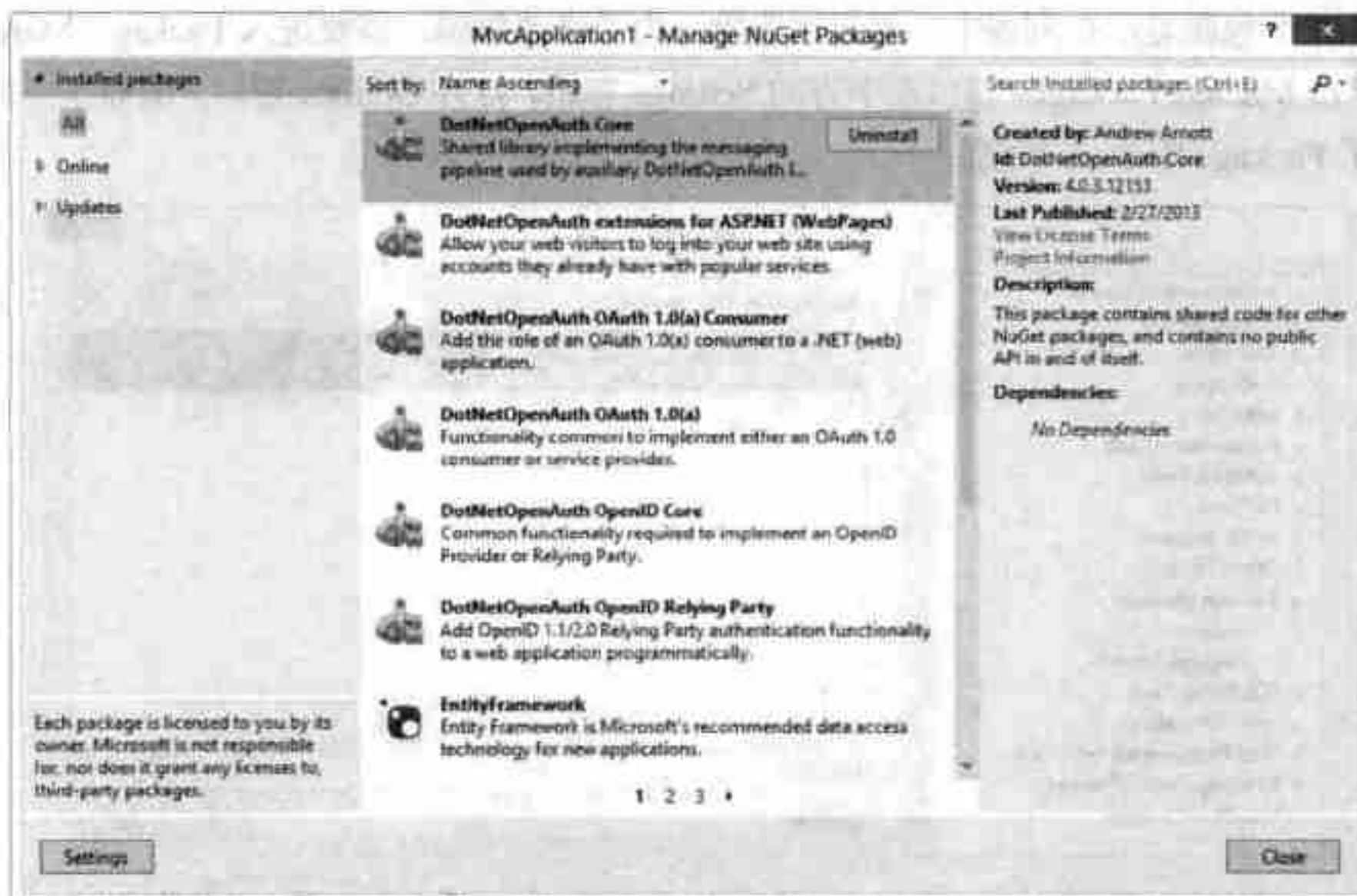


图 G-3

单击 Online 选项卡，查看可安装的包。图 G-4 是选择了 Online 选项卡的窗口。

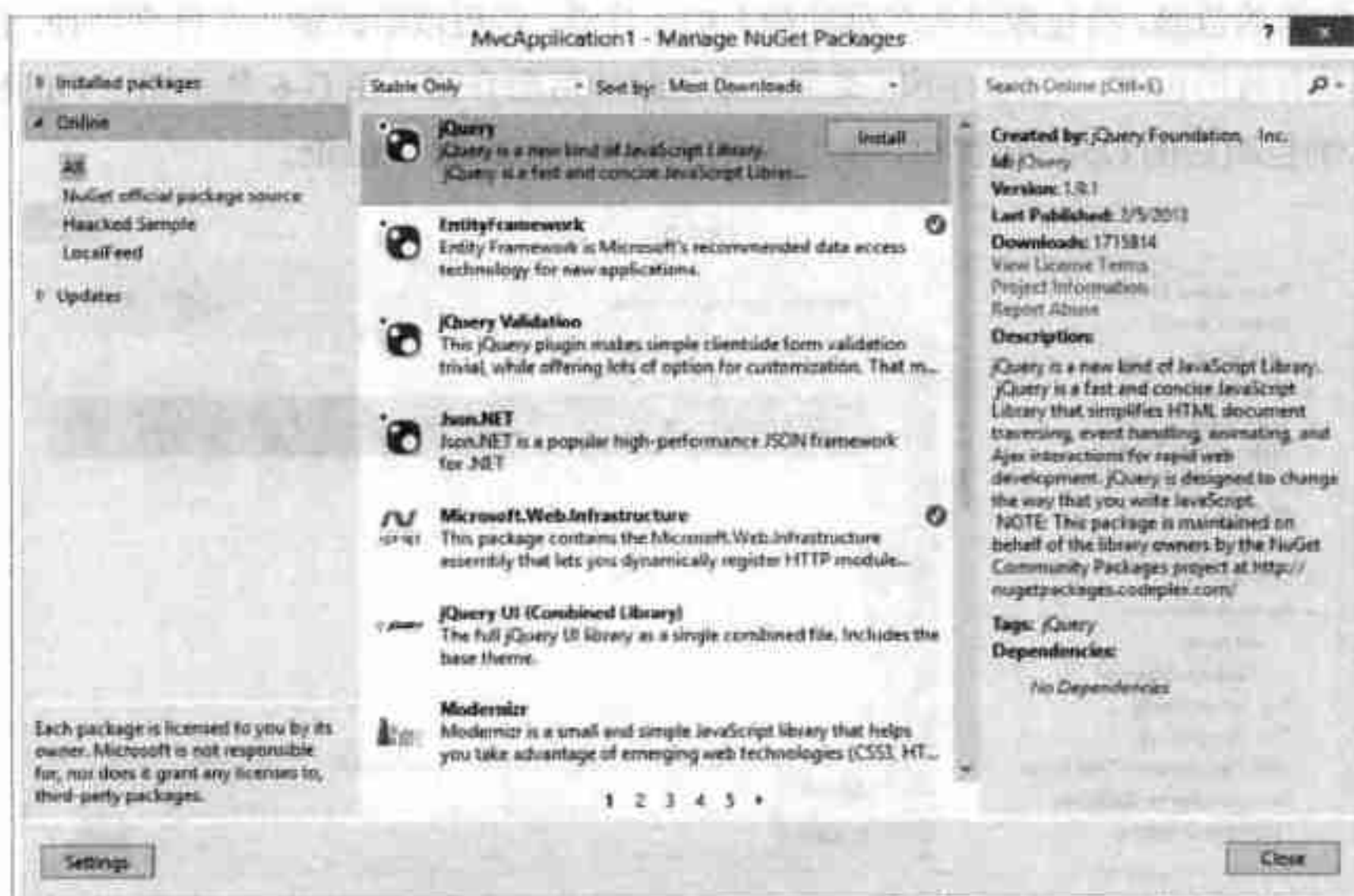


图 G-4

图 G-4 中有几个要注意的地方。首先是列表中的一些包在包条目的右上角有绿色的标记。这些标记表示，包已经安装在当前项目中。包可以安装时，突出显示该包，就会出现 **Install** 按钮。突出显示包，在窗口的右面板上还会显示所选包的更多信息。

NuGet 包被发布到 NuGet 资源库中。NuGet 资源库把包捆绑到种子，种子最终显示在可用包的列表中。这些资源库称为包源。在图 G-4 中可以看到，Online 选项卡的下面有默认的包源。默认的包源由微软拥有，名称是 NuGet 官方包源。可以添加其他包源。也可以启用或禁用包源。这样就可以临时禁止某个包源显示在列表中，而无须删除。为了修改包源，需要进入 Package Manager 的设置界面。单击 **Manage Packages** 窗口左下方的 **Settings** 按钮，打开 **Options** 窗口。图 G-5 中的 **Options** 窗口选择了 **Package Sources** 设置。

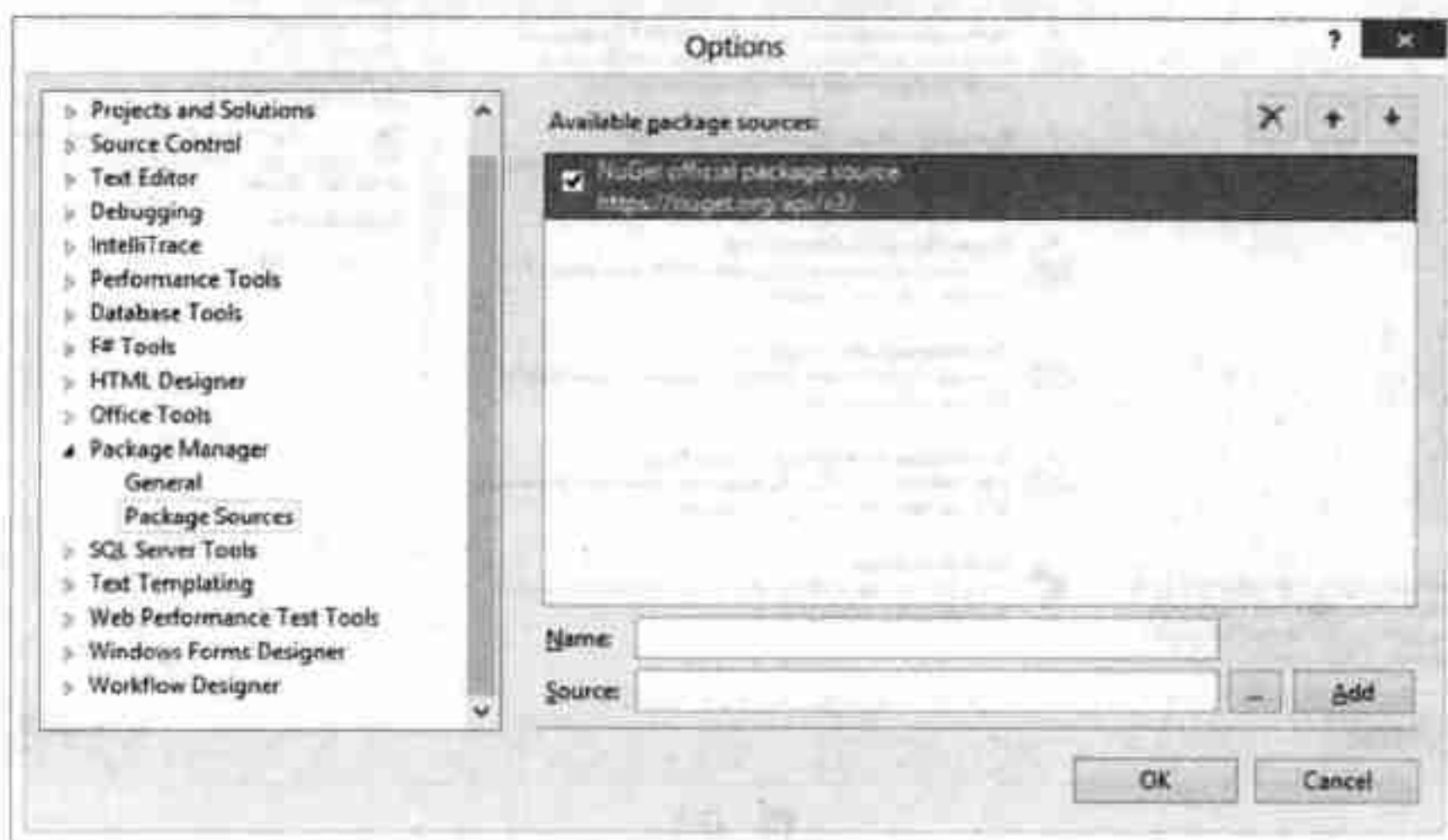


图 G-5

为了添加新的包源，给包源输入名称和源 URL。注意，还可以给源输入文件夹路径。包含 NuGet 包的文件夹是有效的包源。包源的不同类型在本附录的后面介绍。图 G-6 是添加了 Phil Haack 拥有的 NuGet 示例包源后的 **Options** 窗口。新包源的名称是 **Haacked Sample**。

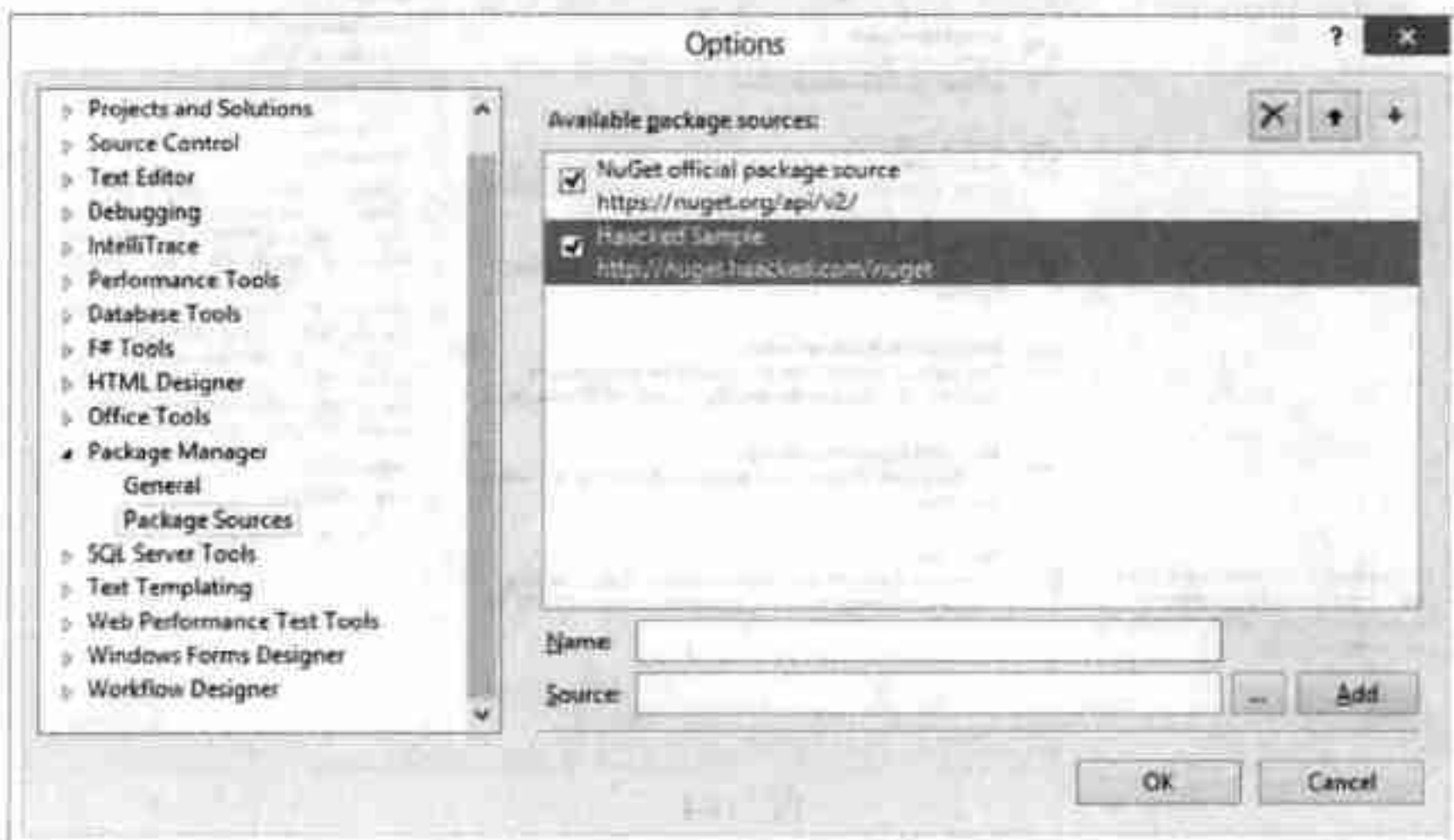


图 G-6

关闭 Options 窗口后，就会刷新 Manage NuGet Packages 窗口中的种子。现在其中包含 Haacked Sample 包源中的包。默认情况下，所有可用包源中的所有包都显示在列表中，如果需要，还可以分页显示。可以选择一个包源，只显示其中的包。图 G-7 显示了 Haacked Sample 包源中的包。

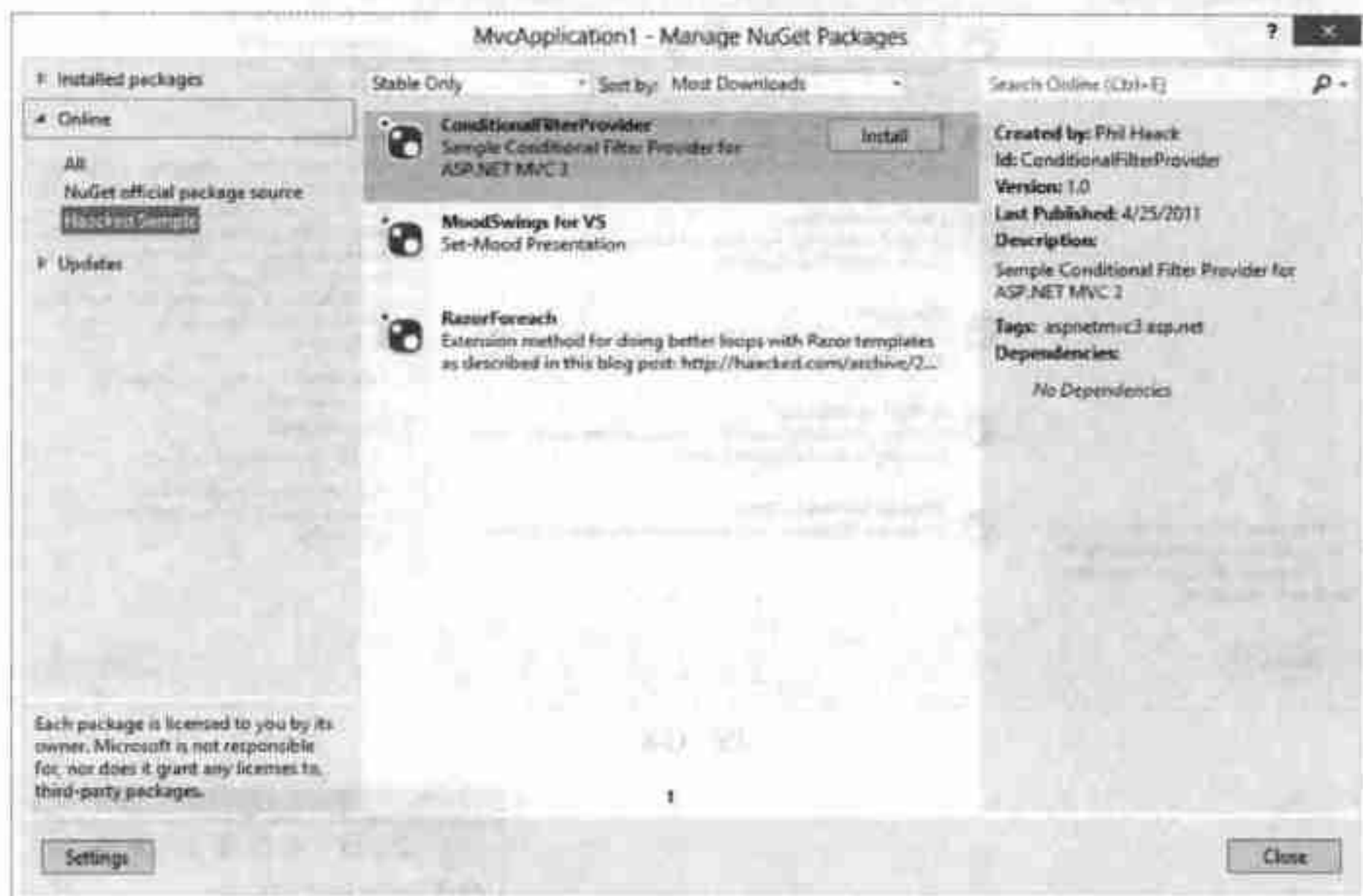


图 G-7

1. 安装 NuGet 包

要安装包，只需单击包的 Install 按钮。但安装 NuGet 包时，会发生什么？为了查看安装时对 NuGet 包进行的一些修改，查找 ELMAH 的包。

ELMAH 允许给 ASP.NET Web 应用程序记录应用程序级别的错误。这是好例子，因为有其他依赖关系，所以需要修改 web.config 文件来加以配置。在 Manage NuGet Packages 窗口中，使用搜索框搜索 ELMAH。注意搜索“记录”或“错误记录”，在搜索结果中也能找到 ELMAH。所以可以搜索内容和包名，如果不知道包名，但知道需要什么功能，就可以搜索内容。图 G-8 显示了搜索 ELMAH 的结果。

单击 Install 按钮，会开始包的下载和安装。安装包时会发生如下额外事件：还会下载并安装依赖的其他包。例如，图 G-9 是下载和安装过程中显示的安装窗口。

注意版本号大于 1.2.2 的 elmah.corelibrary 的依赖关系。ELMAH 依赖的包 elmah.corelibrary 也会下载并安装。这不需要做任何工作，会自动进行。NuGet 包包含依赖关系信息，并处理任何依赖包的定位和安装事务。安装完成后，在 ELMAH 包的旁边会显示绿色的标记，表示已安装。有时在安装包后，会在 Visual Studio 中显示文本文件，以显示额外的指令或说明修改了什么。包安装后，就可以立即在应用程序中使用库了。

安装包时会发生什么？图 G-10 说明对 ELMAH 程序集的引用已被添加到项目中。

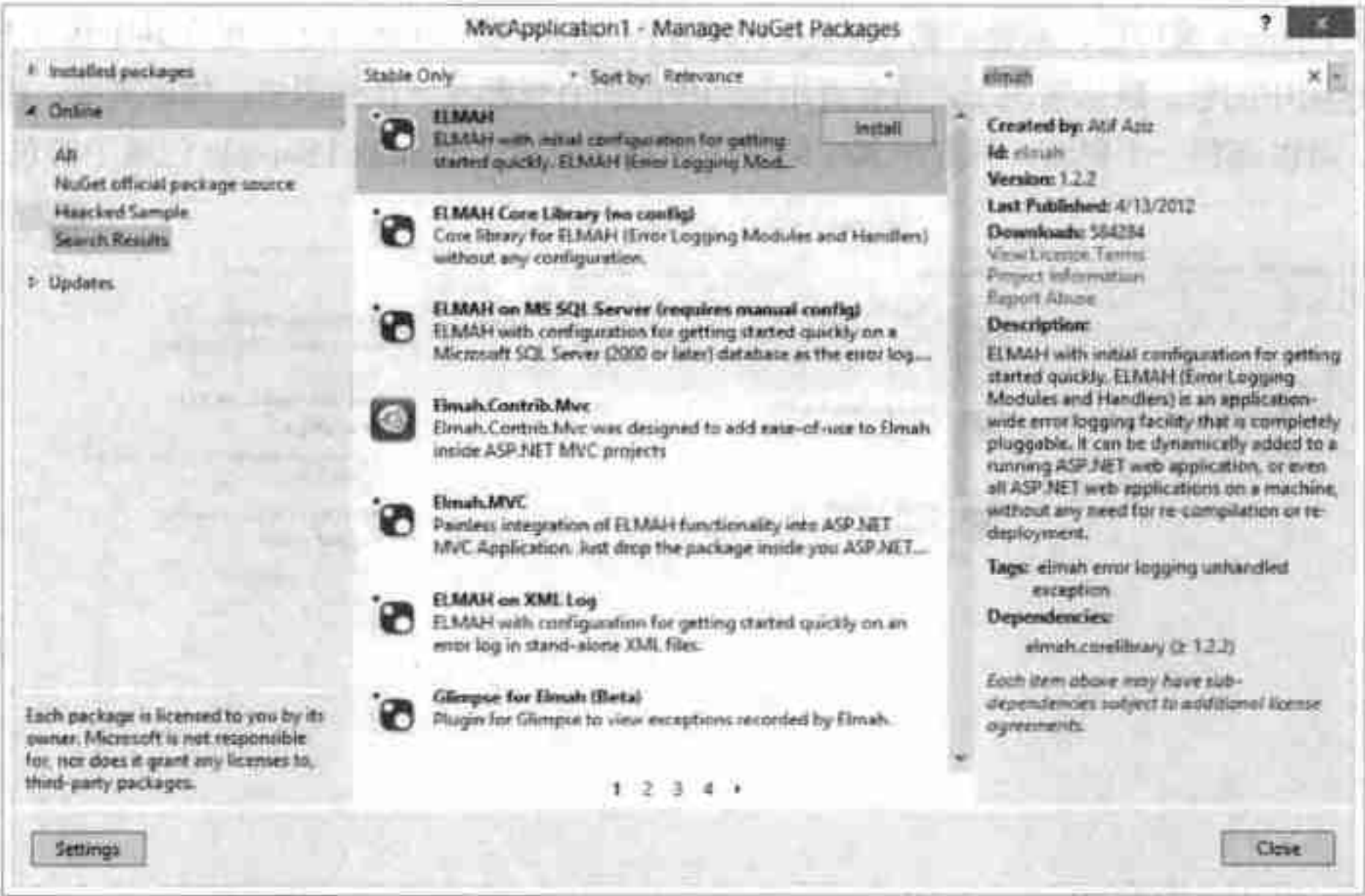


图 G-8

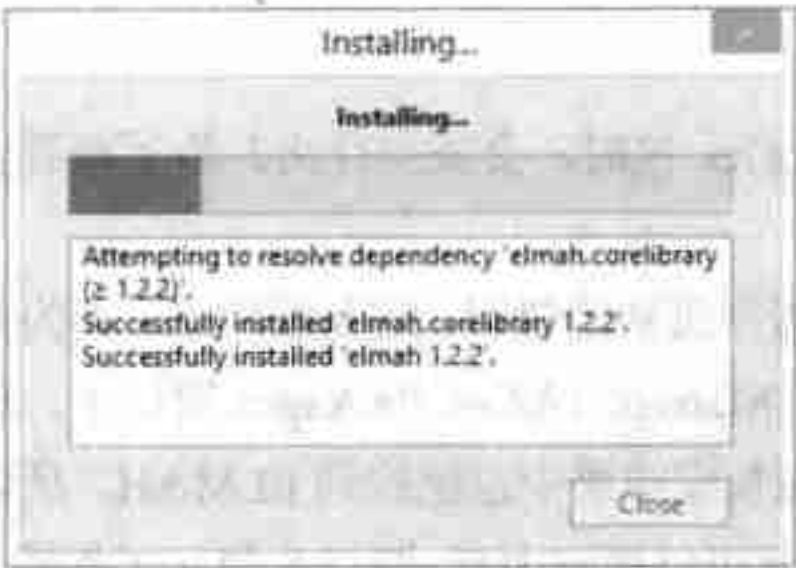


图 G-9

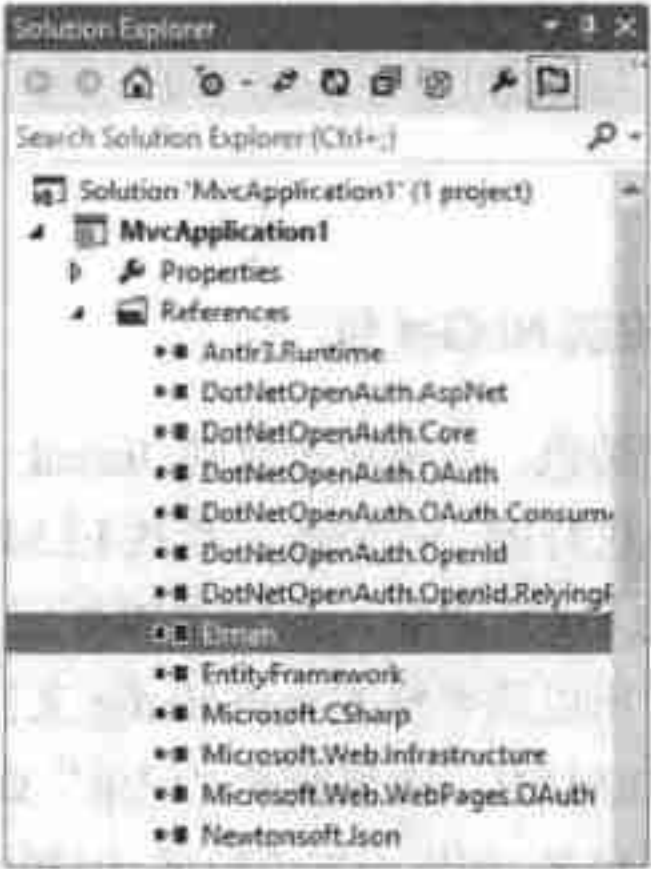


图 G-10

不仅要引用程序集，还要修改 web.config 文件，以支持、配置 ELMAH 及其提供的处理程序和模块。图 G-11 显示了 web.config 文件的一部分，以及对 NuGet 包进行的一些修改。

项目中安装的第一个包会创建 Packages 文件夹。在这个文件夹中，会为每个已安装的包创建子文件夹。在这个例子中，用于所建项目的 Visual Studio 模板已安装了一些包，所以这个文件夹已经存在。该文件夹位于解决方案文件夹中。如果项目没有解决方案文件夹，Packages 文件夹就在项目文件夹中创建。安装 ELMAH 包时，会创建两个子文件夹，一个用于 ELMAH，另一个用于依赖的包，即 ELMAH 核心库。图 G-12 显示了 Packages 文件夹的内容，其中包含由 ELMAH NuGet 包创建的子文件夹。

这些子文件夹都包含包安装的所有文件，包括程序集、定义必要配置修改的转换文件，支持的文档文件，以及包文件本身。

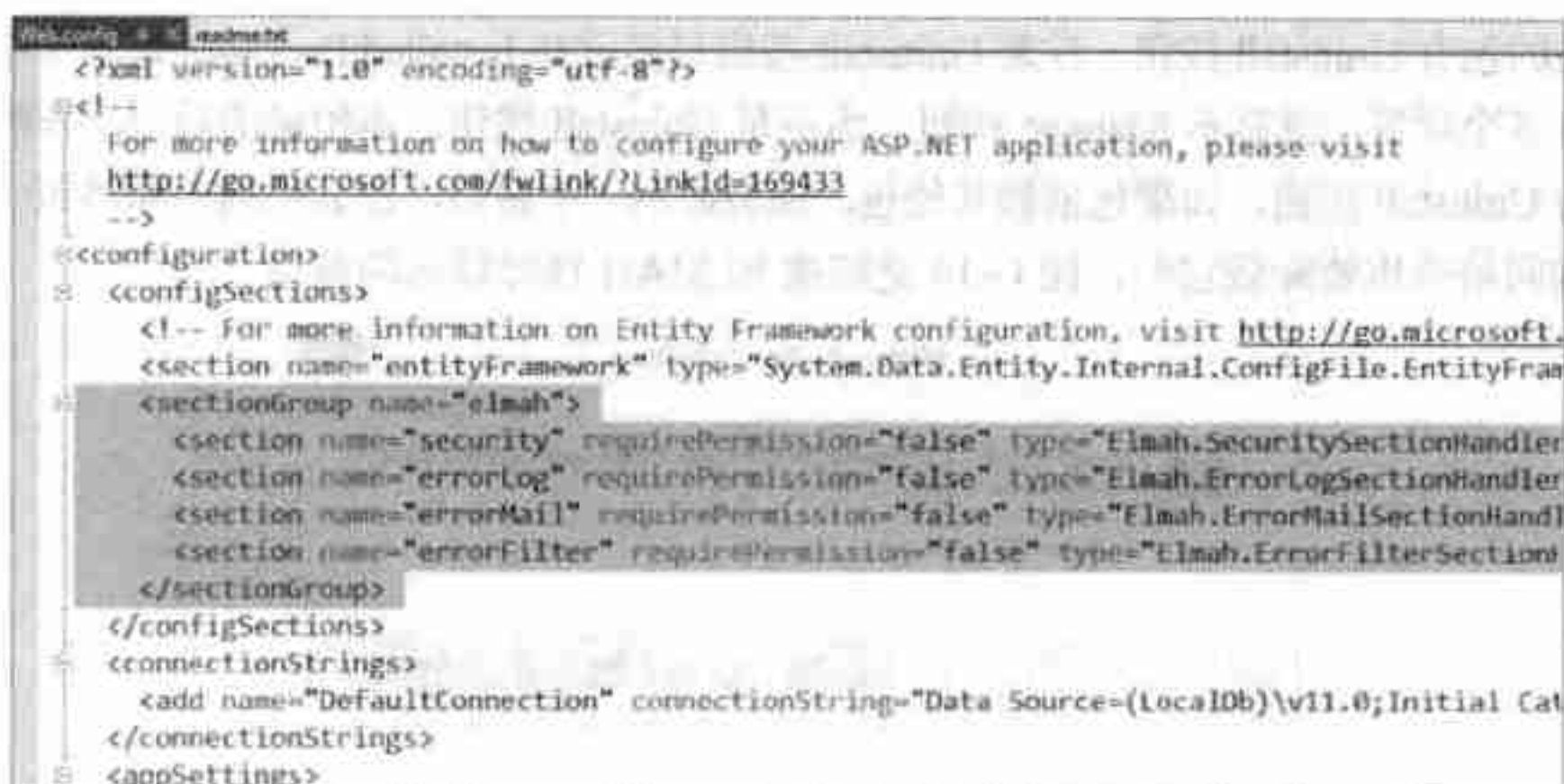


图 G-11

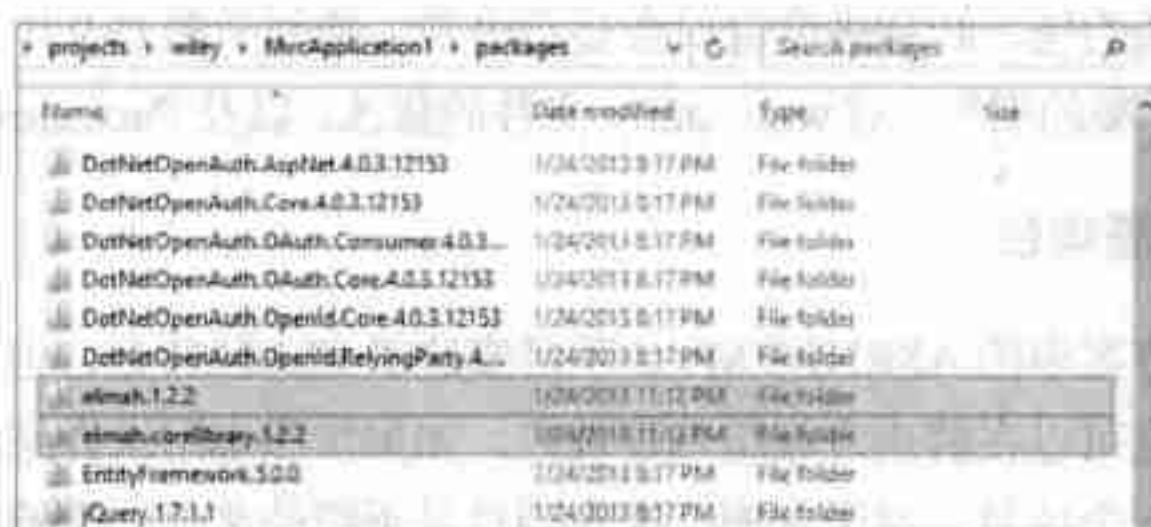


图 G-12

2. 卸载 NuGet 包

如果希望删除或卸载包，该怎么办？删除包与安装包一样简单。再次打开 Manage NuGet Packages 窗口，选择 Installed Packages 选项卡。图 G-13 中的 Manage NuGet Packages 窗口选中了已安装的 ELMAH 包。

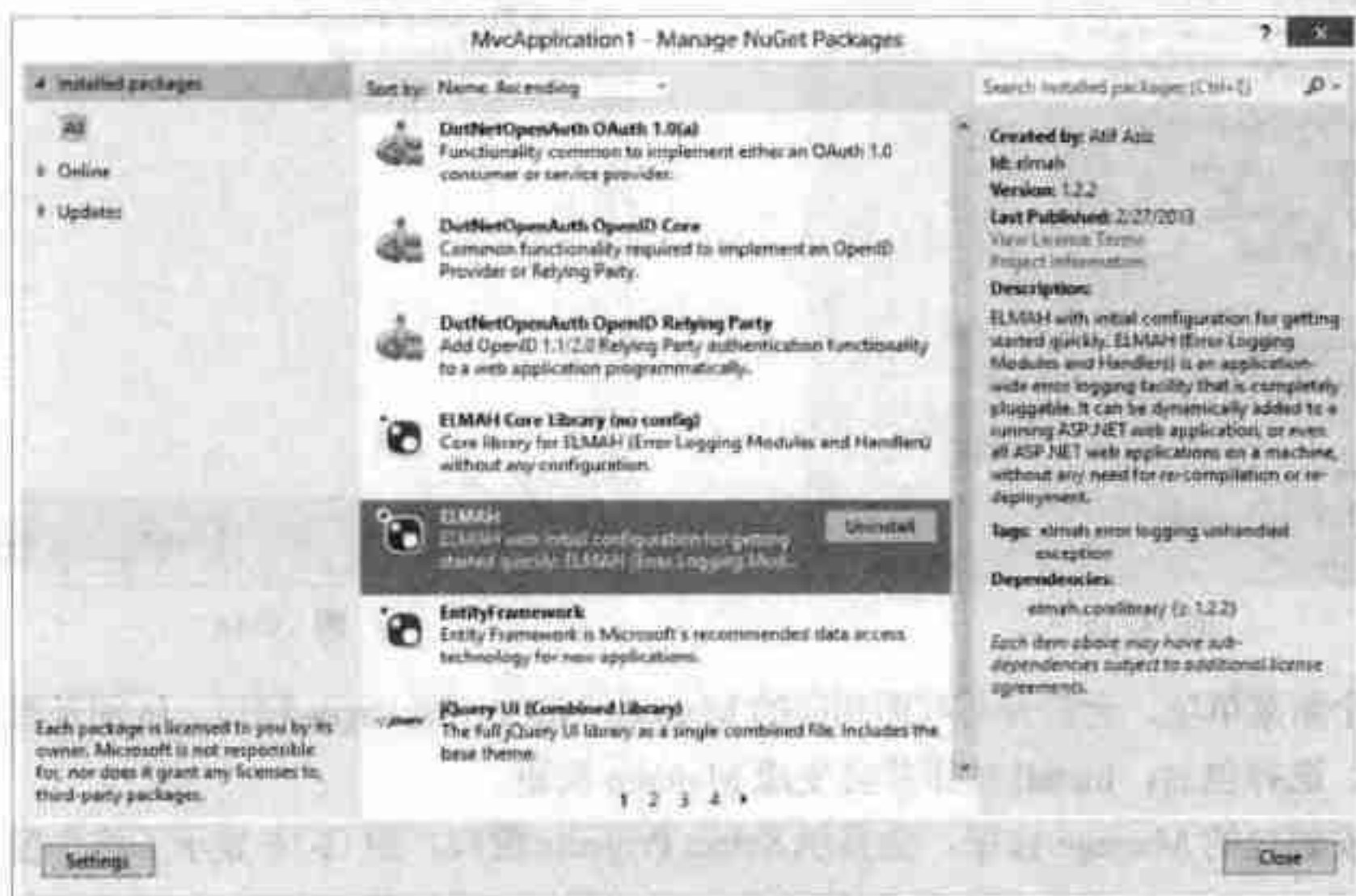


图 G-13

已安装的包有 Uninstall 按钮。注意 Uninstall 按钮只显示在 Installed Packages 选项卡中。如果解决方案中有多个项目，就显示 Manage 按钮，而不是 Uninstall 按钮。本附录的后面会详细介绍这种情况。单击 Uninstall 按钮，如果包依赖其他包，就会显示一个窗口，在其中列出和该包一起安装的所有包，询问是否也要卸载它们。图 G-14 是卸载 ELMAH 包时显示的窗口。

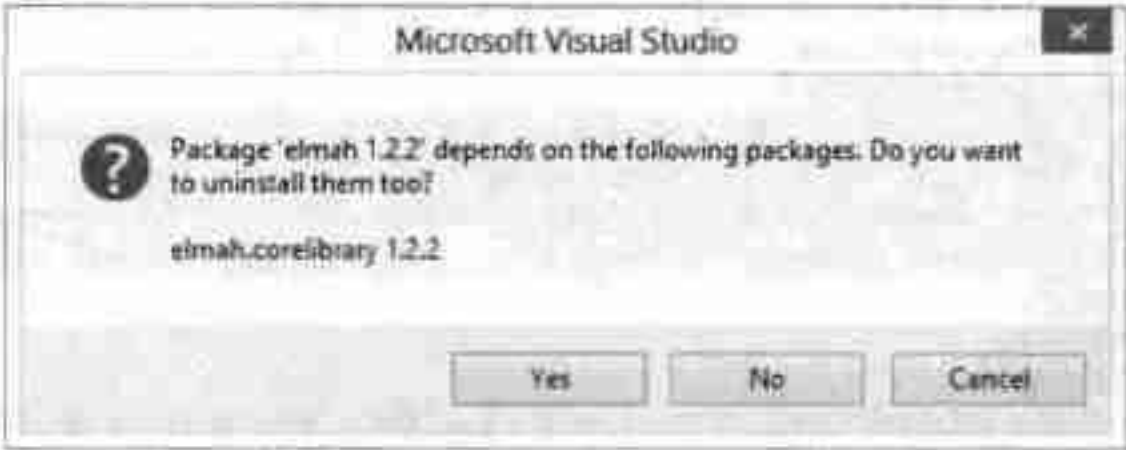


图 G-14

单击 Yes 会卸载依赖的包。卸载进度窗口会显示出来，类似于图 G-9 中显示的安装进度窗口。卸载包后，就删除对程序集的引用、对 web.config 文件的修改，以及 Packages 文件夹中的子文件夹。

3. 在解决方案级别管理包

前面的示例给解决方案中的 ASP.NET MVC 项目安装了 ELMAH 包。但是，如果解决方案中有多个项目，该怎么办？也可以在解决方案级别管理包。如果按照本附录介绍的步骤进行，就给当前使用的解决方案添加另一个项目。对于本例，类库项目是不错的选择。给解决方案添加另一个项目后，右击解决方案。在上下文菜单中，是 Manage NuGet Packages for Solution 选项，而不是 Manage NuGet Packages 选项。图 G-15 显示了包含新菜单项的上下文菜单。

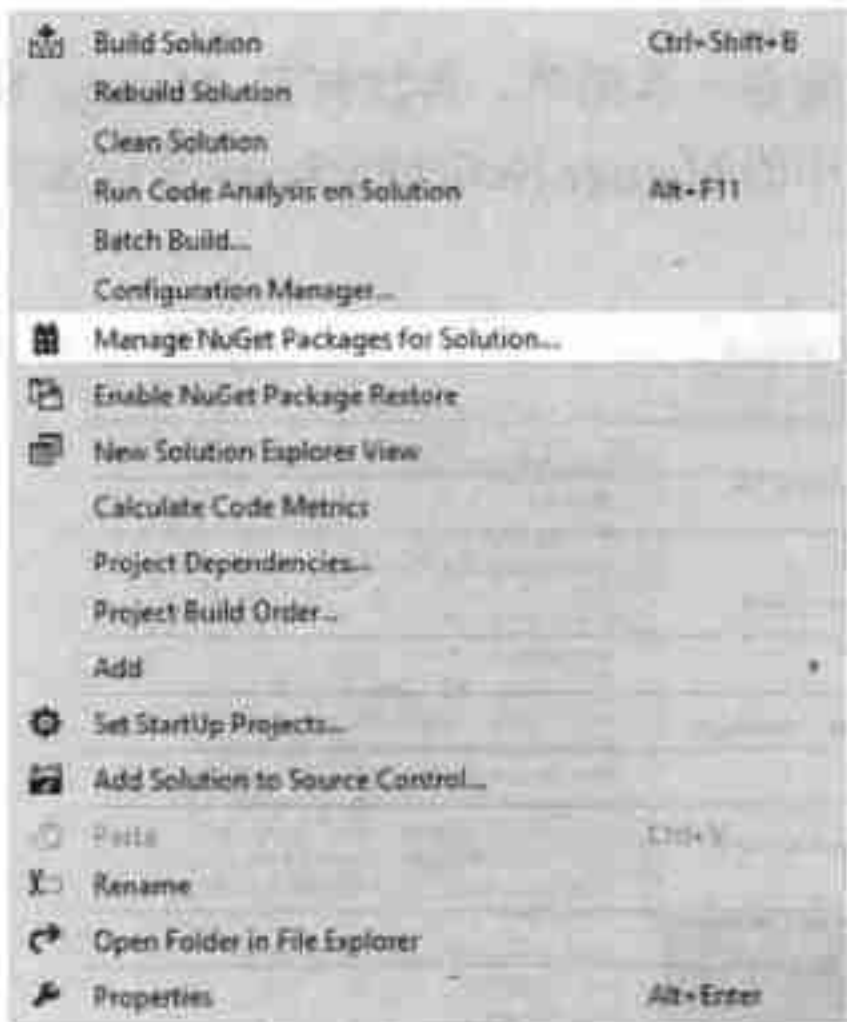


图 G-15

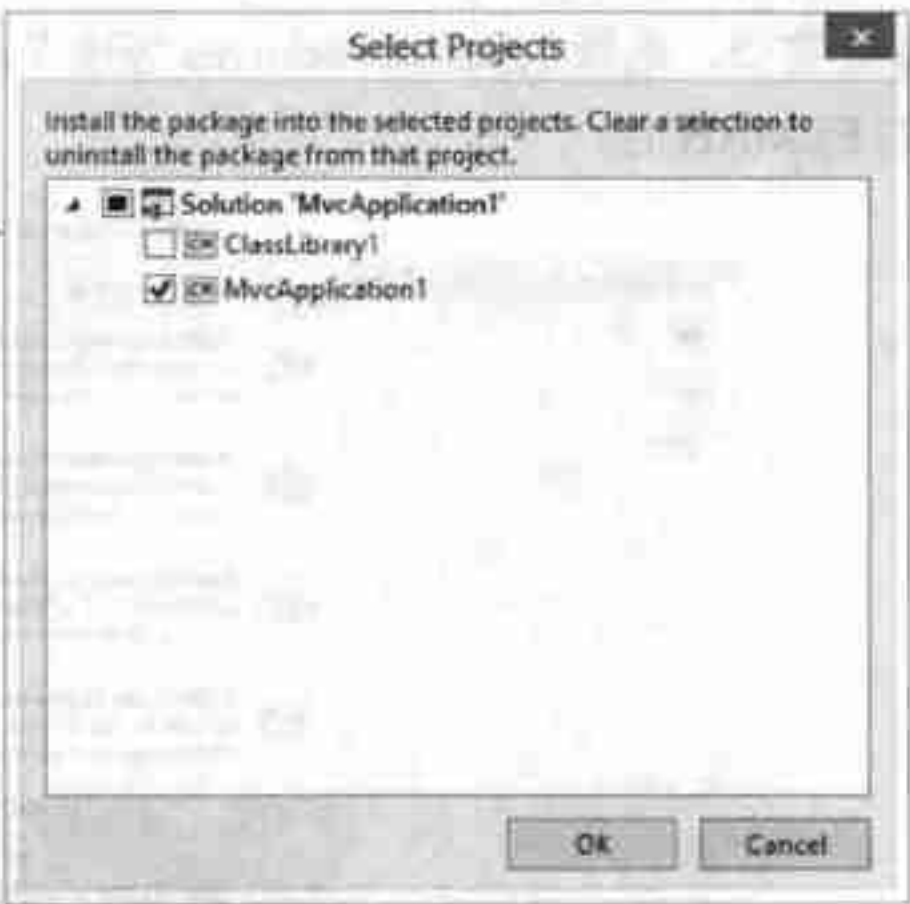


图 G-16

单击这个新菜单项，会打开与前面相同的 Manage NuGet Packages 窗口。区别是查看当前安装的包列表时，选择包后，Install 按钮就会变成 Manage 按钮。

单击已安装包的管理按钮，会显示 Select Projects 窗口，图 G-16 显示了这个窗口。

显然，Select Projects 窗口允许选择给哪些项目安装包。这个窗口也用于从解决方案中删除用于

项目的包。如果希望安装新包，那么过程与为单个项目安装包相同，只是使用 **Select Projects** 窗口。在 **Package Manager** 的解决方案版本中安装包时，会提示选择要给哪些项目安装包。

把包安装到多个项目中，是将图 G-12 中的 **Packages** 文件夹放在 **Solution** 文件夹中的原因之一。如果把包安装到解决方案的多个项目中，就不希望有同一文件的多个副本。使用包的每个项目只需在解决方案级别的某个位置引用其文件即可。

G.1.2 用控制台管理 NuGet 包

如果不愿意使用图形化界面，或者开发人员喜欢使用键盘而不是鼠标，就可以通过 **PowerShell** 命令使用 NuGet 包。NuGet 有控制台窗口。要打开控制台窗口，可以从 **Visual Studio** 菜单中选择 **Tools | Library Package Manager | Package Manager Console**。这会打开 **Package Manager Console** 窗口，如图 G-17 所示。

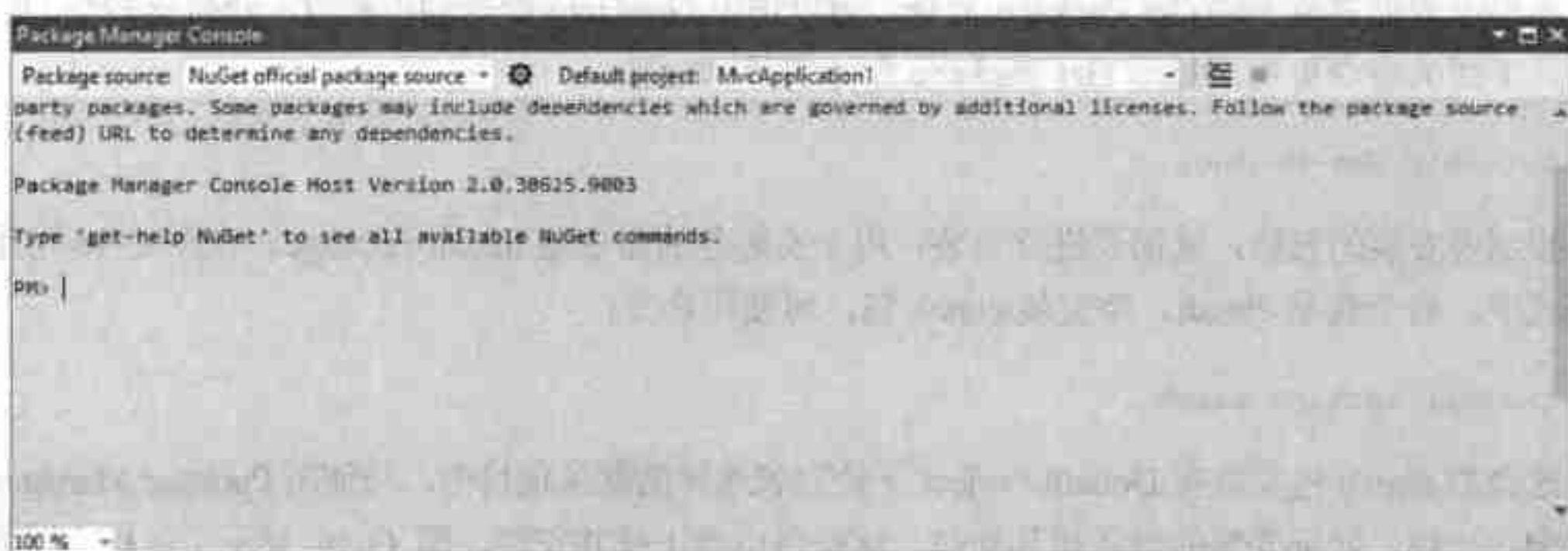


图 G-17

在 **Package Manager Console** 窗口中有两个下拉菜单，第一个允许选择默认的包源，在控制台中输入的命令会使用默认包源。第二个允许选择默认项目，命令会针对默认项目执行。这些默认设置可以使用 **PowerShell** 命令的参数覆盖。首先应查看可用于安装的包。在包源下拉列表中选择包源后，下面的命令列出了所有可用的包：

```
Get-Package -ListAvailable
```

因为这个命令列出了所有可用的包，所以会滚动以显示所有的包。单击红色的停止符号，会停止在控制台中滚动显示包列表。编写本附录时，默认包源种子中有超过 11 000 个不同的包。显示整个列表可能没有什么帮助。使用 **Filter** 参数可以过滤结果。可以把上面搜索框中使用的术语指定为 **Filter** 参数。下面的命令说明了如何搜索 **ELMAH** 包：

```
Get-Package -Filter Elmah -ListAvailable
```

运行这个命令，会列出描述或包标题中包含 **elmah** 的所有包。图 G-18 显示了运行这个命令的结果。

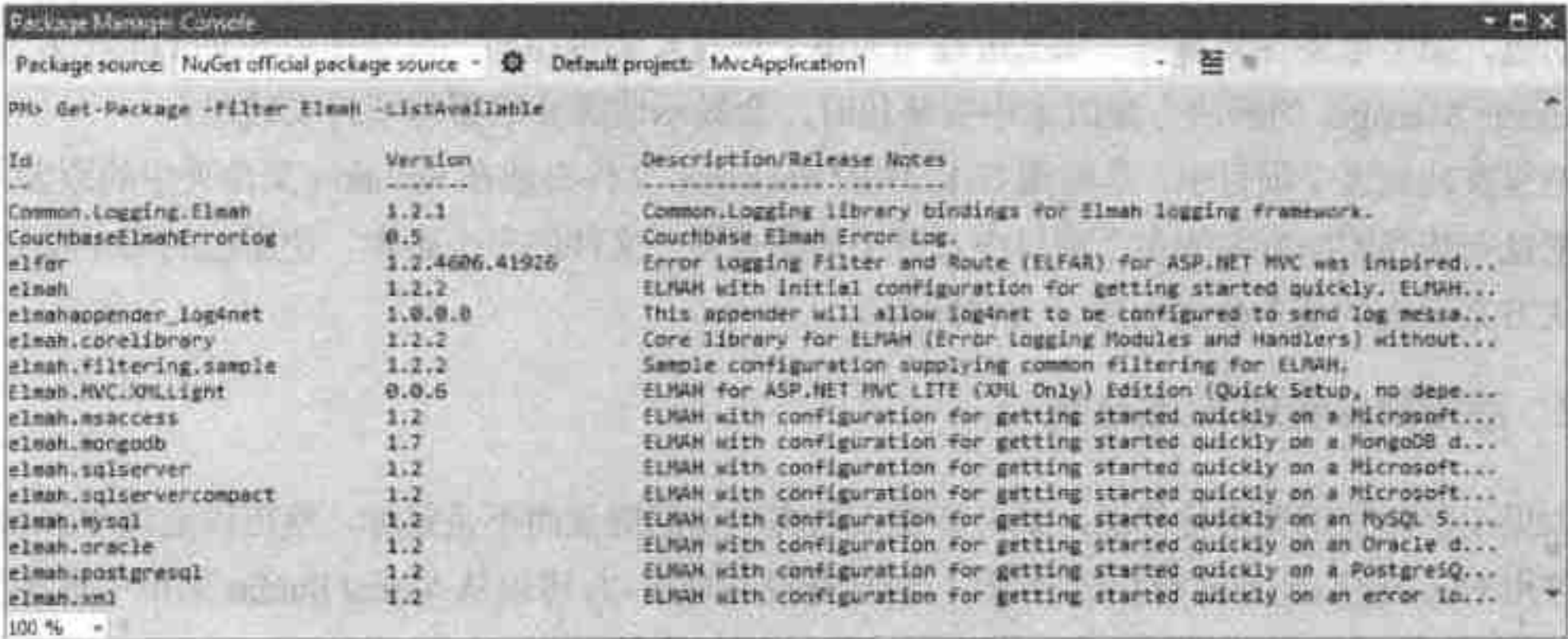


图 G-18

如果希望更多地了解 Get-Package 命令，可以使用 Get-Help 命令列出参数，获得该命令的描述信息。下面的命令可用于显示 Get-Package 命令的帮助信息：

```
Get-Help Get-Package
```

找到要安装的包后，就需要进行安装。用于安装包的命令是 Install-Package。在图 G-18 可用包的列表中，有个包是 elmah。要安装 elmah 包，可使用命令：

```
Install-Package elmah
```

这会把 elmah 包安装在 Default Project 下拉列表选择的默认项目中。与使用 Package Manager 窗口安装包一样，如果安装的包依赖其他包，这些包也会下载并安装。图 G-19 显示了运行上述安装命令的结果。

注意 elmah.corelibrary 的依赖关系已检测出来，这个包也会被安装，无须执行任何操作。



图 G-19

为了查看已安装包的列表，只须运行没有参数的 Get-Package 命令。图 G-20 显示了运行这个命令的结果。

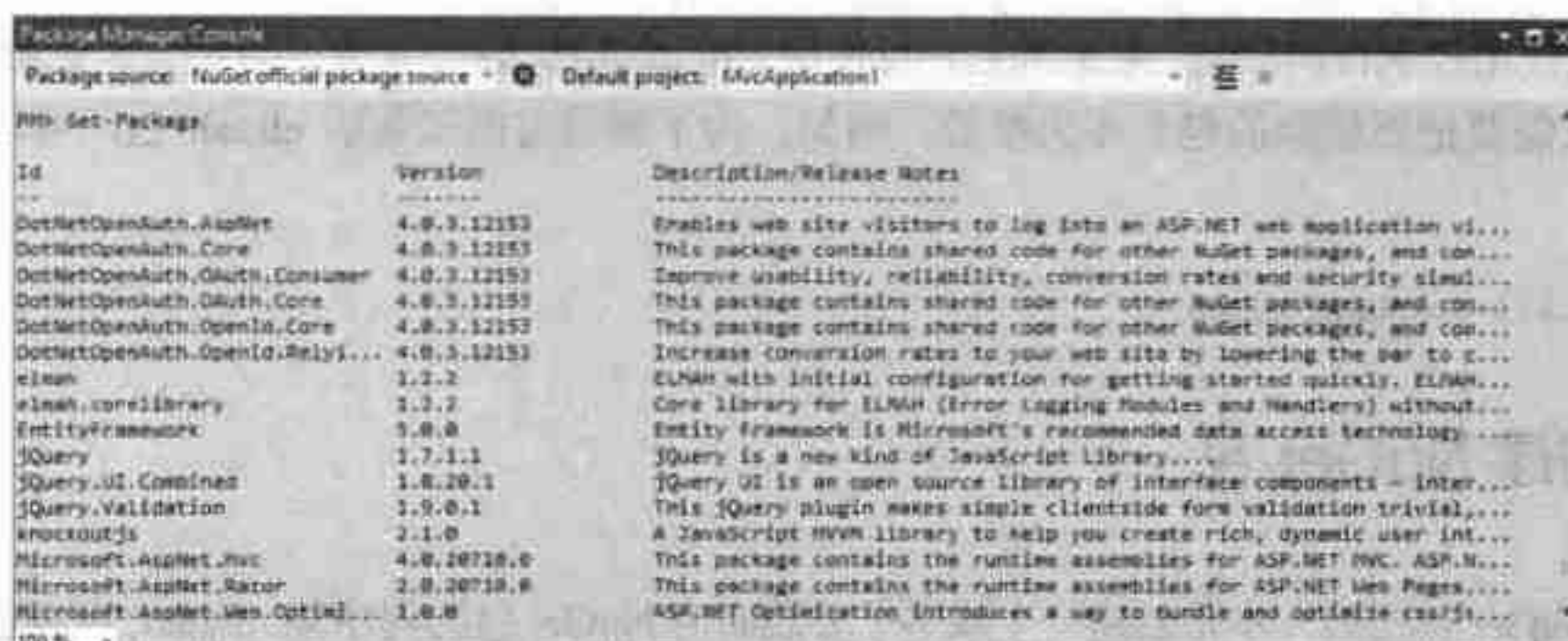


图 G-20

安装完包后,就可能需要把包更新到最新版本。使用带 Updates 参数的 Get-Package 命令,就可以查看可用更新的列表。下面的命令可用于确定已安装的包是否有更新版本:

```
Get-Package -Updates
```

图 G-21 显示了运行这个命令的结果,以查看可用的更新。



图 G-21

从图 G-21 中可以看出, jQuery 包有可用更新。为了更新包,应使用 Update-Package 命令。与安装命令一样,只需把要更新的包名传递给更新命令。为了更新 jQuery 包,可运行下面的命令:

```
Update-Package jQuery
```

图 G-22 显示了运行这个更新命令的结果。



图 G-22

还可以使用控制台删除包。要删除包，可使用 `Uninstall-Package` 命令。与安装和更新命令相似，卸载命令只需要把要删除的包名作为参数。例如，为了删除前面安装的 `elmah` 包，可运行下面的命令：

```
Uninstall-Package elmah
```

G.2 创建 NuGet 包

创建 NuGet 包的第一步是创建一个或多个要发布为 NuGet 包的程序集。例如，可以创建简单的类库项目，其中只包含一个类。为此，程序清单 G-1 中的示例代码可用于创建简单的类。

程序清单 G-1 类库中的示例类

```
public class NugetMath
{
    public int AddNumbers(int number1, int number2)
    {
        return number1 + number2;
    }
}
```

有了一个或多个要用于建立 NuGet 包的程序集后，就需要下载 `NuGet.exe` 引导程序，以创建包。`NuGet.exe` 引导程序可以 NuGet CodePlex 站点 nuget.codeplex.com 的 Downloads 页面下载。下载 `NuGet.exe`，把它放在某个可访问的地方。为了测试 NuGet 是可访问的，可以打开命令行提示，输入 `nuget help`，获得 `NuGet.exe` 引导程序的帮助信息。



还需要告诉 NuGet，可以下载需要运行的任何包。为此，应进入 Visual Studio Options 的 Package Manager 设置，选中 `Allow NuGet to Download Missing Packages During Build` 选项。

如果需要把 NuGet 包发布到需要 API 键的 NuGet 资源库中，例如默认的 NuGet 资源库，就需要获得 API 键，将 NuGet 包标识为属于自己。为了获得默认 NuGet 资源库的 API 键，可以进入 www.nuget.org 并注册。注册后，就会在 My Account 页面上找到 API 键。图 G-23 显示了隐藏 API 键的 My Account 页面。

有了自己的 API 键，就可以用 `setApiKey` 参数运行 NuGet，让它记住自己的 API 键。这样，每次创建 NuGet 包时，就不必指定 API 键了。下面的命令会让 NuGet 记住 API 键，其中的 *your-key* 应使用自己的 API 键替代：

```
nuget setApiKey your-key
```

创建 NuGet 包有多种方式，使用哪种方式取决于库的复杂度。创建 NuGet 包的第一步是创建清单文件，扩展名是 `.nuspec`。`.nuspec` 文件是 XML 清单，描述了 NuGet 包及其依赖关系。

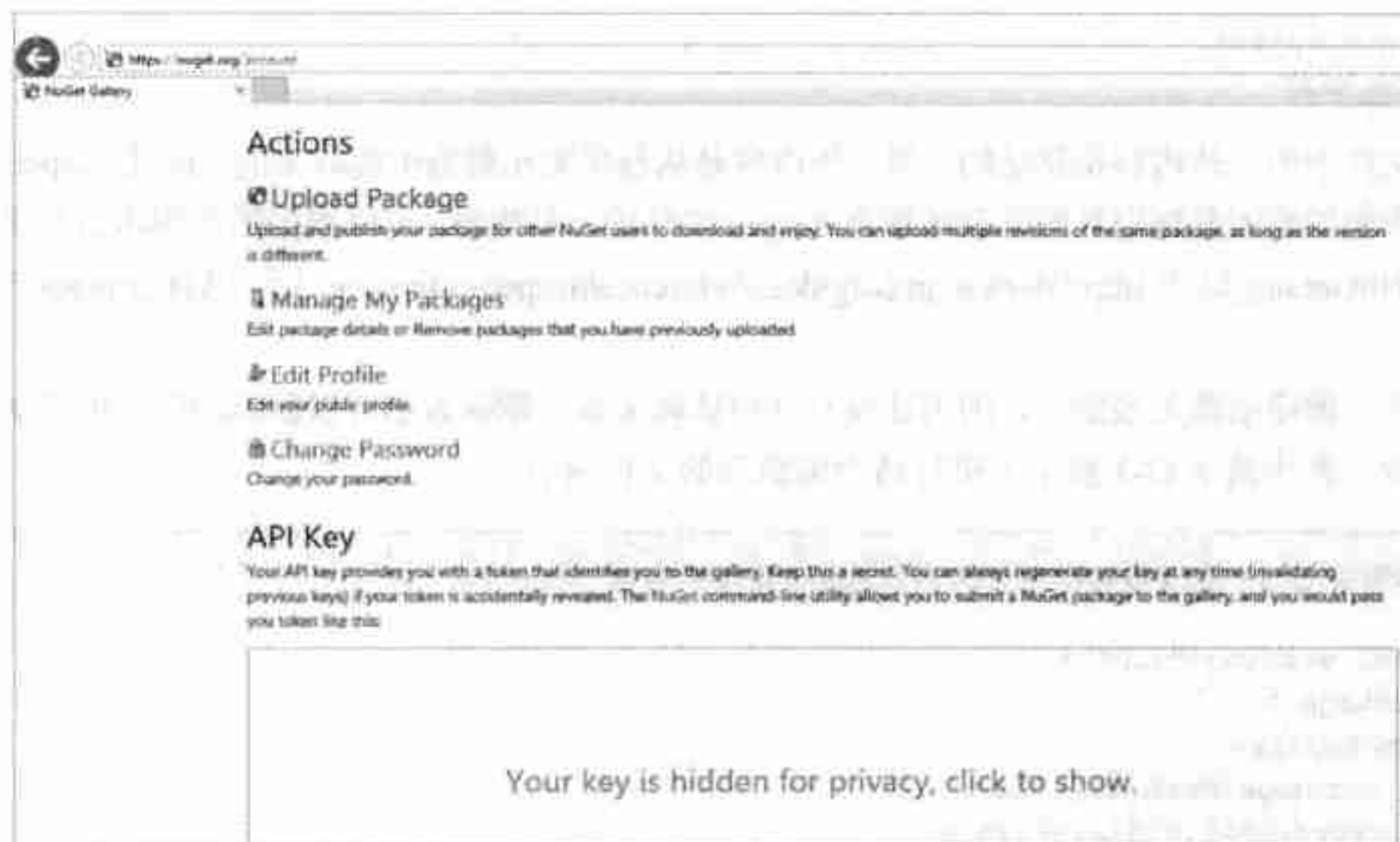


图 G-23

G.2.1 从程序集中创建 NuGet 包

可以从程序集中创建.nuspec 文件。为了给程序集创建该文件，需要用 spec 参数和程序集的路径调用 NuGet.exe 可执行程序。从示例程序集中生成.nuspec 文件的命令如下：

```
nuget spec NugetMath.dll
```

运行这个命令后，就会创建新文件 NugetMath.dll.nuspec。程序清单 G-2 显示了运行这个命令后创建的文件的内容。

程序清单 G-2 .nuspec 文件的内容

```
<?xml version="1.0"?>
<package >
  <metadata>
    <id>NugetMath.dll</id>
    <version>1.0.0</version>
    <authors>Myname</authors>
    <owners>Myname</owners>
    <licenseUrl>http://LICENSE_URL_HERE_OR_DELETE_THIS_LINE</licenseUrl>
    <projectUrl>http://PROJECT_URL_HERE_OR_DELETE_THIS_LINE</projectUrl>
    <iconUrl>http://ICON_URL_HERE_OR_DELETE_THIS_LINE</iconUrl>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>Package description</description>
    <releaseNotes>
      Summary of changes made in this release of the package.
    </releaseNotes>
    <copyright>Copyright 2013</copyright>
    <tags>Tag1 Tag2</tags>
    <dependencies>
      <dependency id="SampleDependency" version="1.0" />
    </dependencies>
```



```
</metadata>
</package>
```

该文件中的一些内容是默认的，但一些内容是从程序集元数据中提取来的。创建.nuspec文件时引用程序集的原因是使用元数据自动填充.nuspec文件的一些内容。可以根据需要编辑这个文件的内容。在NuGet.org站点<http://docs.nuget.org/docs/reference/nuspec-reference>上可以查看.nuspec文件的完整规范。

现在，删除依赖关系部分，因为还没有任何依赖关系。删除表示可以删除的行，根据需要编辑其他字段。程序清单G-3显示了进行适当编辑后的文件内容。

程序清单 G-3 删除依赖关系部分后.nuspec文件的内容

```
<?xml version="1.0"?>
<package >
  <metadata>
    <id>NugetMath.dll</id>
    <version>1.0.0</version>
    <authors>Myname</authors>
    <owners>Myname</owners>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>A sample library to do addition</description>
    <releaseNotes>This is the initial release.</releaseNotes>
    <copyright>Copyright 2013</copyright>
    <tags>Sample Math Addition</tags> .
  </metadata>
</package>
```

有了清单后，还必须执行一个步骤，才能创建NuGet包。在创建包时，NuGet.exe引导程序会给予文件夹和文件使用特定的约定。在项目中安装包时，为了让NuGet包自动创建对程序集的引用，需要把程序集放在lib文件夹中。在包含.nuspec文件的文件夹中，创建子文件夹lib，把NugetMath.dll移入其中。为了给这个.nuspec文件创建包，只需用pack参数和.nuspec文件名运行NuGet命令。下面是该命令的用法示例：

```
nuget pack NugetMath.dll.nuspec
```

上述命令生成了文件NugetMath.dll.1.0.0.nupkg，这就是NuGet包文件，nupkg文件只是重命名的.zip文件。如果把扩展名从nupkg改为.zip，就可以解压该文件，查看其中的内容。.nuspec文件、其他支持文件和文件夹应在ZIP文件的根目录下。lib子文件夹包含了程序集。nupkg文件现在可以发布到NuGet资源库中，包含在该资源库的可用包的种子中。

G.2.2 从项目中创建NuGet包

从程序集中生成.nuspec文件是创建所需清单的最简单方式。但是，可能需要考虑略复杂一些的情形。例如，如果在库项目中安装了一个或多个NuGet包，这些包就是NuGet包的依赖文件。从程序集中生成.nuspec文件，不会选择这些依赖文件。此时，可以从.NET项目中生成.nuspec文件。为了演示这一点，把一个NuGet包安装到类库项目中。在项目中安装Structuremap，Structuremap是控件库的一个开源注入/反向依赖文件。如果解决方案中有多个项目，就把要安装包的项目作为默认项目。可以在Package Manager窗口中搜索Structuremap，或者在Package Manager Console窗口中执行

如下命令：

```
Install-Package structuremap
```

把 NuGet 包安装到项目中，因此为 Structuremap 创建了依赖关系，之后就可以从项目文件中生成.nuspec 文件了。这么做是为了在创建 NuGet 包时考虑依赖关系。打开控制台窗口，查看包含.csproj 或.vbproj 文件的目录。在该目录中，执行如下命令：

```
nuget spec
```

执行这个命令会生成.nuspec 文件。如果查看生成的文件，就会发现与直接从程序集中生成的文件有一些区别。不是从程序集中读取元数据，放在.nuspec 文件中，而是在一些元素中包含标记，例如\$version\$、\$author\$和\$description\$，创建包时会替代这些标记。这些标记的值会用作项目的元数据，但直到创建包之前，是不会提取这些值的，因为它们的值可能随时间而变化。典型示例是\$version\$。还要注意，没有 dependencies 部分。程序清单 G-4 显示了生成的.nuspec 文件。

程序清单 G-4 从项目中生成的.nuspec 文件的内容

```
<?xml version="1.0"?>
<package >
  <metadata>
    <id>$id$</id>
    <version>$version$</version>
    <title>$title$</title>
    <authors>$author$</authors>
    <owners>$author$</owners>
    <licenseUrl>http://LICENSE_URL_HERE_OR_DELETE_THIS_LINE</licenseUrl>
    <projectUrl>http://PROJECT_URL_HERE_OR_DELETE_THIS_LINE</projectUrl>
    <iconUrl>http://ICON_URL_HERE_OR_DELETE_THIS_LINE</iconUrl>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>$description$</description>
    <releaseNotes>
      Summary of changes made in this release of the package.
    </releaseNotes>
    <copyright>Copyright 2013</copyright>
    <tags>Tag1 Tag2</tags>
  </metadata>
</package>
```

与前面的例子一样，生成.nuspec 文件后，可以根据需要进行编辑。这里应删除表示可以删除的代码行，用硬编码的值替代作者标记，修改与包相关的其他值。程序清单 G-4 中已修改的标记以粗体显示。程序清单 G-5 显示了编辑后的文件。

程序清单 G-5 .nuspec 文件编辑后的内容

```
<?xml version="1.0"?>
<package >
  <metadata>
    <id>$id$</id>
    <version>$version$</version>
    <title>$title$</title>
```

```

<authors>Myname</authors>
<owners>Myname</owners>
<requireLicenseAcceptance>false</requireLicenseAcceptance>
<description>$description$</description>
<releaseNotes>Initial release.</releaseNotes>
<copyright>Copyright 2013</copyright>
<tags>Sample Math Addition</tags>
</metadata>
</package>

```

根据需要编辑.nuspec文件后,就可以创建NuGet包了。要从项目中创建包,应使用pack参数和项目的文件名执行NuGet:

```
nuget pack NugetMath.csproj
```

注意从项目中生成NuGet包时,指定了项目文件,而没有指定创建包时的.nuspec文件。.nuspec文件仍自动包含在包中。如果把NuGet包的扩展名改为.zip,解压缩该文件,查看NugetMath.nuspec文件,就可以看出所有的标记都被替换为项目元数据中的值。因为在NuGet包Structuremap中有依赖关系,所以还添加了dependencies部分,其中有用于Structuremap依赖关系的项。

G.2.3 发布NuGet包

创建NuGet包后,就要把它发布到资源库中。可以直接把NuGet包发布到www.nuget.org网站。登录到该网站,进入账户页面后,就可以单击Upload Package链接,按照指令进行发布。图G-24显示了Upload Your Package页面的一部分。



图 G-24

如果使用命令行发布包,那么该过程只涉及一个命令。如前所述创建包后,就用push参数执行NuGet。把包发布到默认NuGet资源库的命令如下:

```
nuget push NugetMath.1.0.0.0.nupkg
```


上述命令假定在以前对 NuGet 的调用中，使用 `setApiKey` 参数存储了 API 键。使用命令行参数可以重写 API 键和包源位置。

G.3 拥有 NuGet 包

如果无法找到 NuGet 包，NuGet 包就没有什么用。NuGet 资源库常常称为 NuGet Gallery。默认的 NuGet Gallery 由微软拥有。本附录一直在使用这个默认的 NuGet Gallery，位于 www.nuget.org。默认 NuGet Gallery 中的包可用于搜索它的任何人。但如果希望某个包只能由自己、自己的公司或小组使用，该怎么办？有一些选项可以拥有自己的 NuGet Gallery。

G.3.1 文件共享宿主

拥有自己的 NuGet Gallery 的最简单选项是使用本地文件夹或文件共享。尽管这个选项支持必须手工更新的只读库，但却是建立私有库的快捷方式。要创建本地库，可以在本地磁盘上创建一个文件夹，把可用的 NuGet 包(*.nupkg 文件)复制到该文件夹中。

在如图 G-5 所示的 Visual Studio Package Manager 选项中，添加新的包源。把本地库的文件夹路径指定为源。现在查看可安装的包，并指定刚才添加的包源时，会看到库文件夹中的所有包。这个选项非常适用于测试自己的 NuGet 包，之后把它们发布到公共或私有网络库中。

在网络磁盘上创建文件共享的工作方式与上面相同。访问网络文件共享中的库时可以像其他文件共享那样进行保护。无论包是存储在本地文件夹还是网络文件夹，都必须手工更新或添加包。必须手工把包复制到库文件夹中。还不能使用 OData 查询搜索包，而库放在 IIS 中时是可以搜索的。

G.3.2 IIS 宿主

拥有 NuGet Gallery 的另一方法是把它放在 ASP.NET 网站的 IIS 中。幸好，也许很讽刺，有一个 NuGet 包有助于创建库。为了创建驻留在 IIS 中的 NuGet Gallery，首先在 Visual Studio 中创建一个新的 ASP.NET 空白 Web 应用程序项目，接着安装 NuGet 包 `nuget.server`。如果使用 Package Manager 窗口，就搜索 `nuget.server` 包并安装。如果使用 Package Manager Console，就应使用如下命令安装 `nuget.server` 包：

```
Install-Package nuget.server
```

`nuget.server` 包安装文件，创建必要的文件夹，以包含自己的 NuGet Gallery。默认情况下，包存储在 `Packages` 文件夹中。可以把任何初始包放在这个文件夹中，把它们标记为要编译的内容，使它们随网站一起发布。在 `web.config` 文件的 `appSettings` 部分修改 `packagesPath` 键，就可以修改用于存储包的文件夹。还有另外两个 `appSettings`，用于控制 NuGet Gallery 是否允许包使用 `NuGet.exe` 引导程序发布。如果 `requireApiKey` 的值设置为 `false`，包就可以发布到库中，且不使用 API 键。

如果 `requireApiKey` 的值设置为 `true`，库是否接受要发布的包就取决于 `apiKey` 的值。如果 `apiKey` 设置被删除，或者它的值是空白，库就是只读的，不支持使用 `NuGet.exe` 发布新包。为了支持包的发布，可以创建强密码，把 `apiKey` 的值设置为该密码。这个密码是共享的公钥，把包发布给库的任何人都可以使用。



应验证在 `system.web` 部分没有多个 `compilation` 元素。在面向 .NET Framework 4.5 的网站时安装 `nuget.server`，会添加另一个 `compilation` 元素，以指定目标框架是 4.5 版本。此时，只需删除面向 4.0 版本的元素实例即可。

不需要对默认设置进行任何修改(除了上述特殊段落中的内容)，库就准备好运行了。如果按照本附录的指令进行，就把前面创建的 `NuGetMath` 包复制到 `Packages` 文件夹中，运行网站。图 G-25 显示了库的主页面。

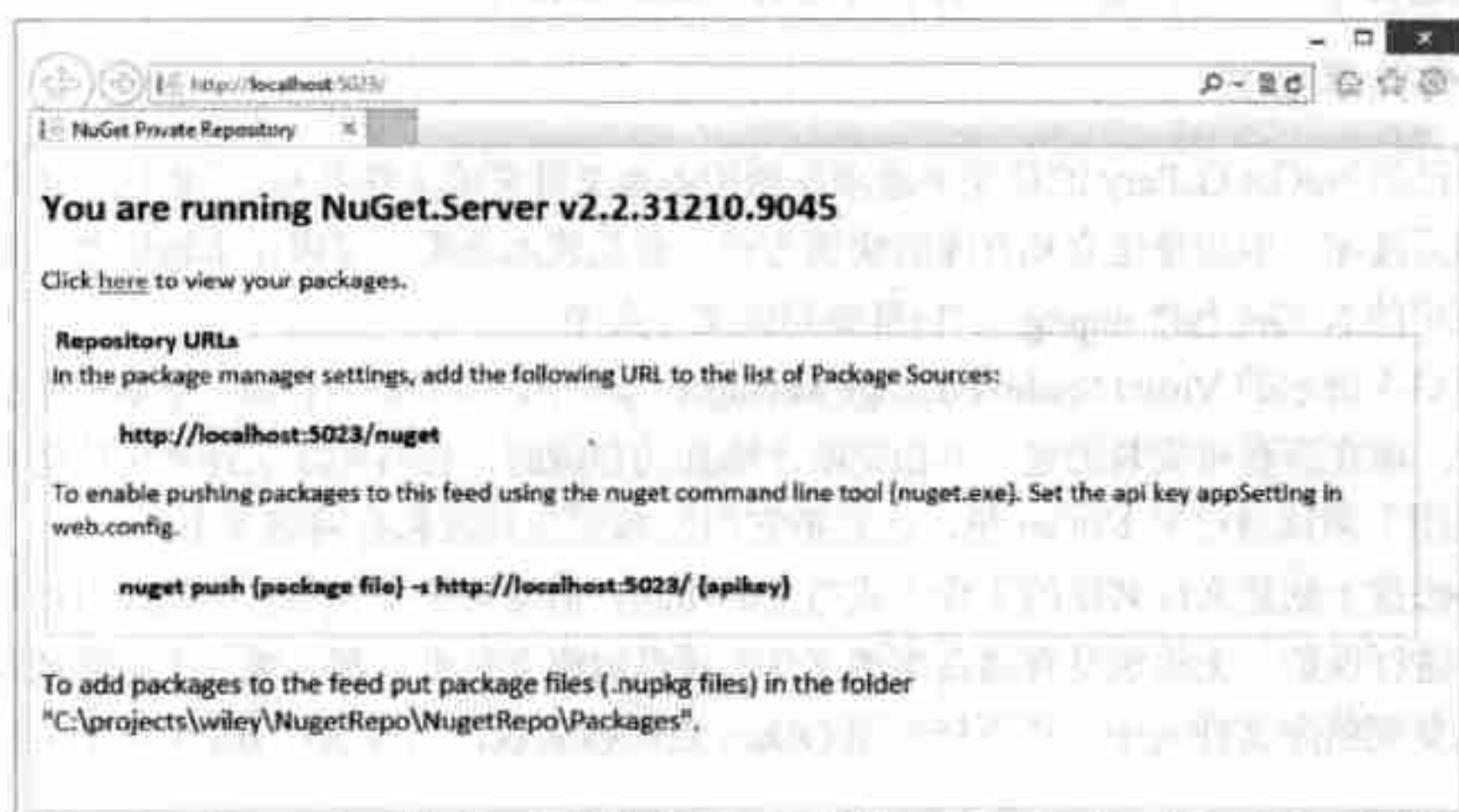


图 G-25

注意图 G-25 中“Click here to view your packages”所在的一行。单击该链接会显示库中包的 OData over ATOM 种子。图 G-26 是单击该链接后显示的页面。

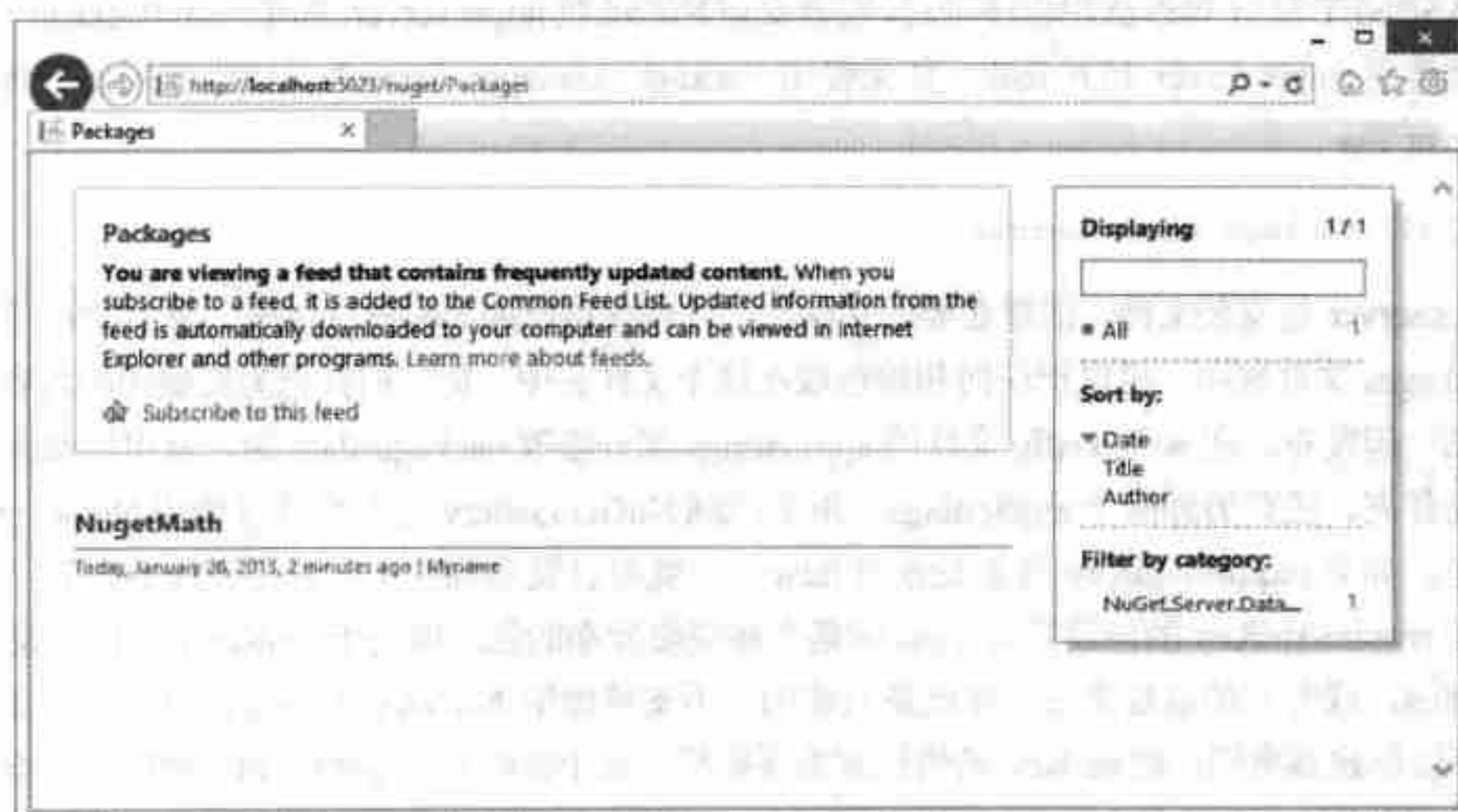


图 G-26

如果希望测试库，就可以在 Visual Studio 的 Package Manager 设置中把如图 G-26 所示的地址添加为新的包源。发布网站与发布其他网站没有区别。发布后，就有了自己的私有 NuGet Gallery。

G.4 用 NuGet 扩展 Visual Studio

Package Manager Console 是 Visual Studio 的 PowerShell 控制台。前面一直使用该控制台与 NuGet 交互操作。还可以使用该控制台与 Visual Studio 交互操作，并让 Visual Studio 自动运行。使用 NuGet 包可以给 Package Manager Console 添加命令，执行独立的操作，或者与 Visual Studio 和项目交互操作。

在前面介绍创建 NuGet 包的内容中，使用 NuGet.exe 引导程序创建了包。还有一个 Windows GUI 应用程序可以创建 NuGet 包。NuGet Package Explorer 是 click-once 应用程序，大大简化了包的创建。它还可以用于查看其他 NuGet 包，了解它们的结构。NuGet Package Explorer 是 CodePlex 项目，位于 npe.codeplex.com。从这里可以下载并安装 NuGet Package Explorer。

G.4.1 创建 PowerShell 脚本文件

作为给包管理器添加命令的例子，创建一个 NuGet 包，添加一个命令，生成长度可变的随机密码。创建文本文件 `init.ps1`，在解决方案中安装包时，第一次运行 `init.ps1` 文件，只给每个解决方案运行一次。如果在解决方案的多个项目中安装了相同的包，脚本在这些安装过程中就不运行。脚本还在解决方案每次打开时运行。程序清单 G-6 显示了 `init.ps1` 文件的内容。

程序清单 G-6 `init.ps1` 文件的内容

```
param($installPath, $toolsPath, $package)
Import-Module (Join-Path $toolsPath GenRandomPassword.psm1)
```

第一行声明了脚本的参数集。这些参数通过 NuGet 传递给脚本。脚本的第二行用于导入 PowerShell 模块。这个程序清单指定了 `GenRandomPassword.psm1` 脚本。

接着需要创建这个 PowerShell 模块。创建 `GenRandomPassword.psm1` 文件。程序清单 G-7 显示了 `GenRandomPassword.psm1` 文件的内容。

程序清单 G-7 `GenRandomPassword.psm1` 文件的内容

```
$rand = New-Object System.Random

function Get-RndPassword($num1) {
    1..$num1 | ForEach {
        $NewPassword = $NewPassword + [char]$rand.next(33,127)
    }
    return $NewPassword
}

Export-ModuleMember Get-RndPassword
```

第一行创建了一个 `System.Random` 对象，它在下一行定义的 `Get-RndPassword` 函数中使用。`Get-RndPassword` 函数接受一个输入参数，该参数用作传入 `ForEach` 的数字范围的上限。`ForEach` 为

范围内的每个数字选择 33 到 127 之间的随机字符代码，并把该字符添加到变量 `$NewPassword` 中。`ForEach` 执行完后，就从函数中返回得到的 `$NewPassword`。最后一行为函数调用 `Export-Module`，使之可用于 `Package Manager Console`。

G.4.2 给 NuGet 包添加文件

创建了两个文件后，就使用 `NuGet Package Explorer` 创建 NuGet 包。打开 `NuGet Package Explorer`，从 `Common tasks` 对话框中选择 `Create a new package`。图 G-27 显示了选择 `Create a new package` 选项后的 `NuGet Package Explorer`。

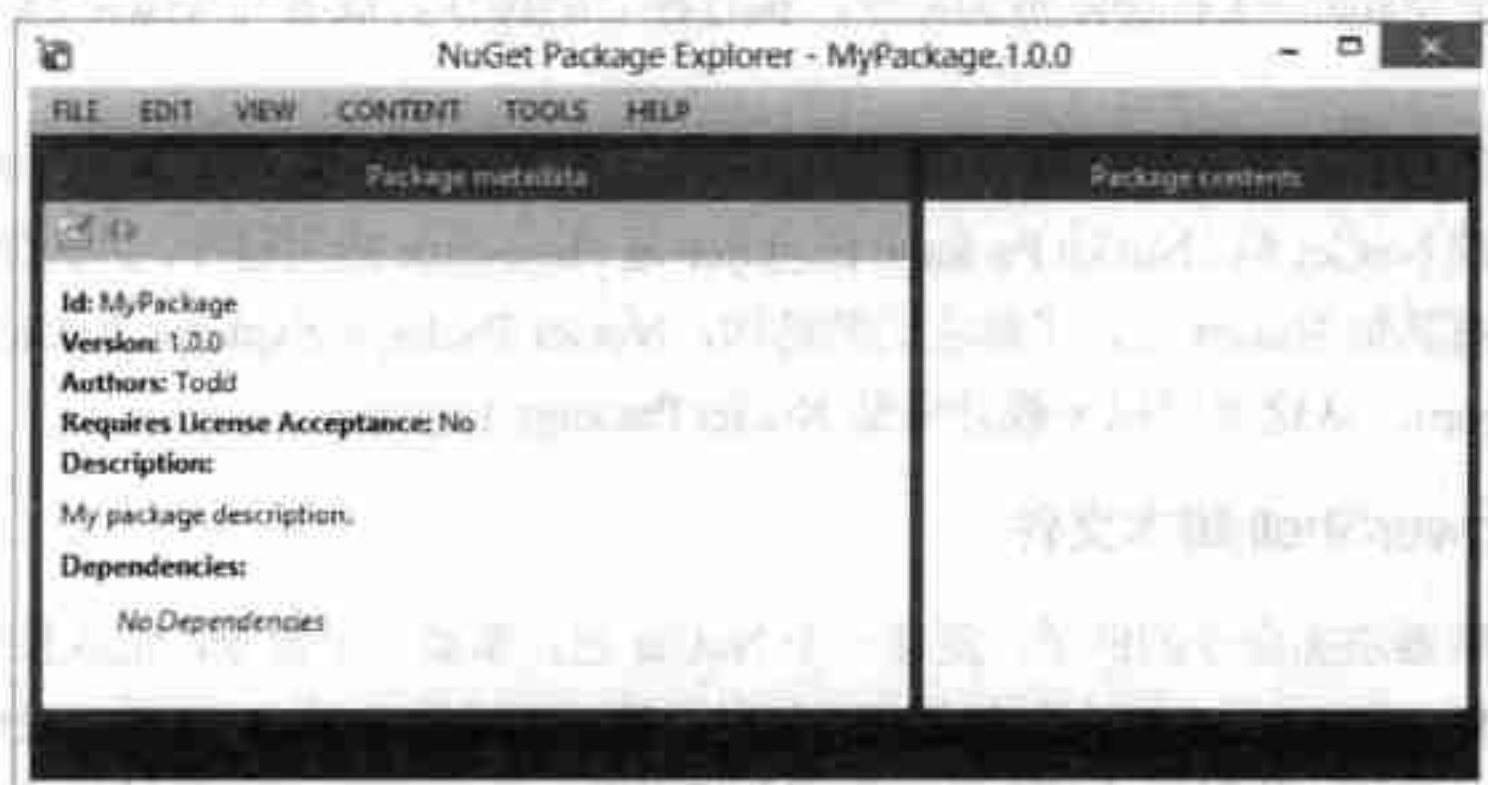


图 G-27

把 `init.ps1` 和 `GenRandomPassword.psm1` 文件拖放到 `NuGet Package Explorer` 的 `Package contents` 面板中。对于这两个文件，`NuGet Package Explorer` 会把它们识别为 `PowerShell` 脚本，询问是否把这些文件放在 `tools` 文件夹中，给这两个文件回答 `Yes`，就会创建 `tools` 文件夹，把这两个文件放在 `tools` 文件夹中。现在需要编辑包的元数据。

G.4.3 编辑 NuGet 包的元数据

需要编辑包的元数据，为特定的包定制。从菜单中选择 `Edit | Edit Metadata`。应编辑的主要字段是 `Id`。`Id` 字段应设置为 NuGet 包名。在这个例子中，把 `Id` 字段改为 `GenRandomPassword`。图 G-28 显示了编辑元数据后的 `NuGet Package Explorer`。

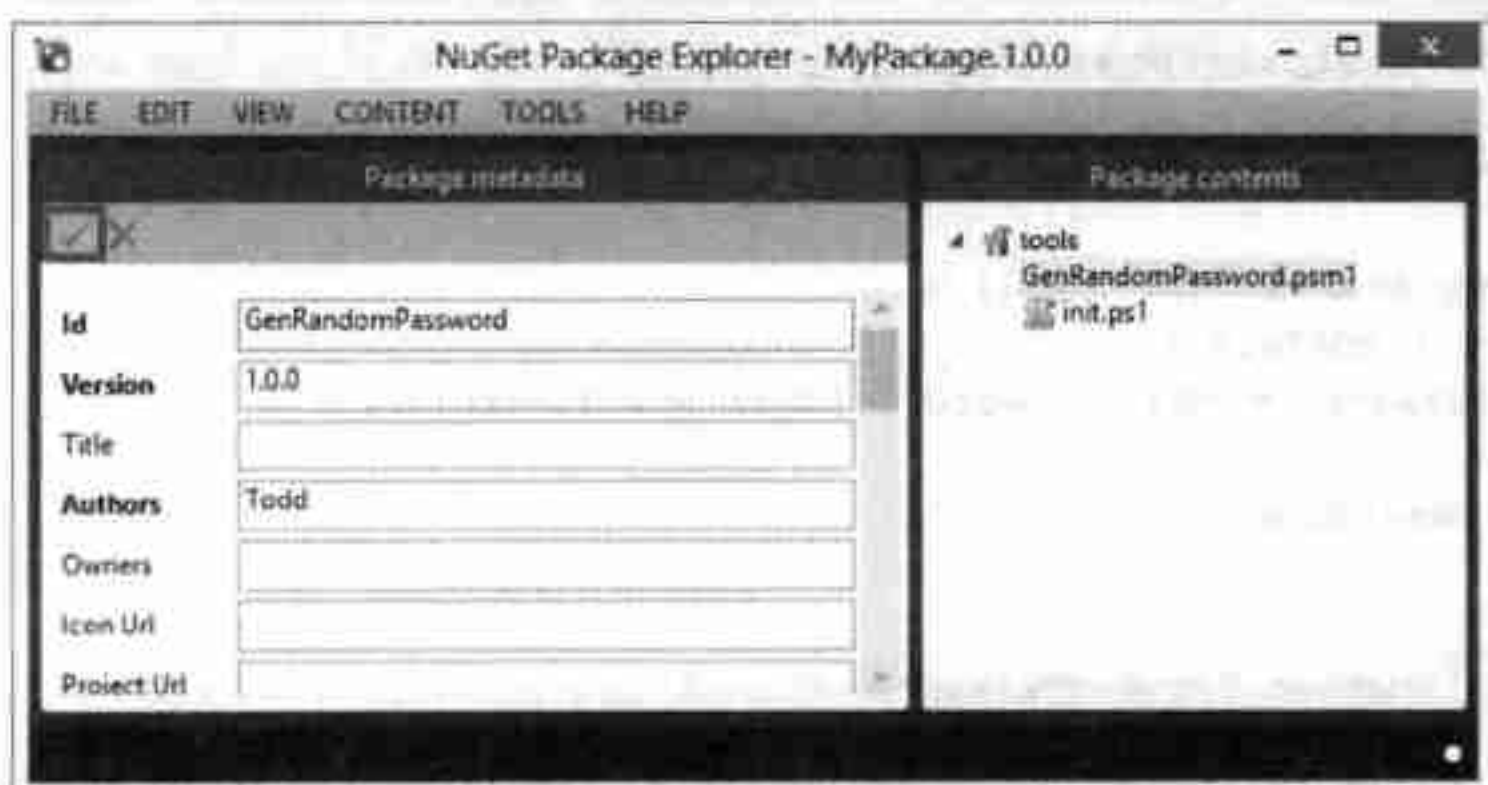


图 G-28

编辑完元数据后，单击图 G-28 中 Package metadata 面板左上角的复选标记。

G.4.4 部署 NuGet 包

现在可以发布 NuGet 包了。为了把 NuGet 包发布到驻留的 NuGet 库中，可以使用 File | Publish 菜单项。但测试包的最佳方式是把包保存到本地或文件共享 NuGet 库中。选择 NuGet Package Explorer 菜单中的 File | Save，把.nupkg 文件保存到前面使用的本地文件夹或文件共享中。

打开 Package Manager Console，把 Package source 设置为本地库。输入下面的命令以安装 NuGet 包 GenRandomPassword：

```
Install-Package GenRandomPassword
```

安装完包后，测试在 NuGet 包中添加到控制台的命令。下面的命令会生成 12 个字符的随机字符串：

```
Get-RndPassword 12
```

图 G-29 显示了安装 NuGet 包、执行 NuGet 包添加的命令后的 Package Manager Console。

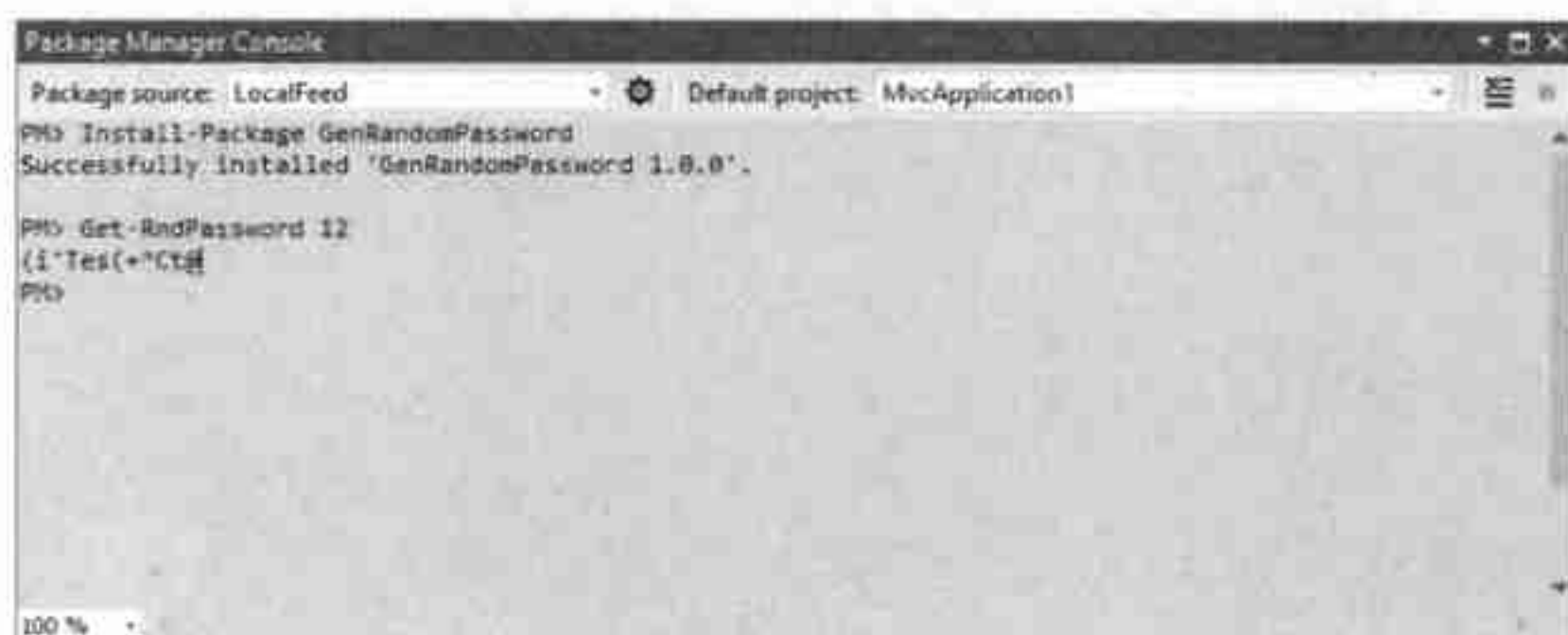


图 G-29

G.5 小结

过去，很难查找、安装和配置.NET 项目中的库。有了 NuGet，给项目添加库的复杂度即使没有完全消除，也大大降低了。利用 NuGet 资源库提供的集中化种子，查找提供所需功能的各种库也容易了许多。

如果开发人员创建库是为了供他人使用，创建 NuGet 包就已成为发布库的一种简单方式，这些库很容易发现和安装。使用 NuGet 查找和安装库已经成为开发人员的事实标准。如果开发小组希望或需要更多地控制对库和分组标准化的访问，可以使用 NuGet 资源库的各种宿主选项。这些选项包括创建私有资源库，这些库只能由小组或公司的某些开发人员访问。

NuGet 还可以使用 PowerShell 命令提供 Visual Studio 扩展。NuGet 继续发展，为开源开发人员提供了发布项目的简单方式。NuGet 与 Visual Studio 的集成也便于给项目扩展可重用的功能，节省了时间和资源。