

Java 语言的产生及其特点

1.1 Java 历史与现状

Java 来自于 Sun 公司的一个叫 Green 的项目,其原先的目的是为家用消费电子产品开发一个分布式代码系统,这样我们可以

把 E-mail 发给电冰箱、电视机等家用电器,它们进行控制,和它们进行信息交流。开始,准备采用 C++,但 C++太复杂,安全性差,

最后基于 C++开发一种新的语言 Oak(Java 的前身),Oak 是一种用于网络的精巧而安全的语言,Sun 公司曾依此投标一个交互式电视

项目,但结果是被 SGI 打败。可怜的 Oak 几乎无家可归,恰巧这时 Mark Andreessen 开发的 Mosaic 和 Netscape 启发了 Oak 项目组成员

,他们用 Java 编制了 HotJava 浏览器,得到了 Sun 公司首席执行官 Scott McNealy 的支持,触发了 Java 进军 Internet。Java 的取

名也有一个趣闻,有一天,几位 Java 成员组的会员正在讨论给这个新的语言取什么名字,当时他们正在咖啡馆喝着 java(爪哇)咖啡,有一个人灵机一动说就叫 Java 怎样,得到了其他人的赞赏,于是,Java 这个名字就这样传开了。

Java 的产生最早可以追溯到 1991 年,是 Sun Microsystem 公司的 Jame Gosling、Bill Joe 等人组成的 Geen 开发小组

设计开发的,其目的是为了开发消费类电子产品,如个人数字助手(PDA)、电子翻译器、电视游戏机以及交互式有线电视控制

盒等的市场。他们发现已有的 C 和 *++在安全性和对硬件的依赖性不适合家用类电子产品,因而开发了一个简单实用的名为

Oak 的小软件,但是并没有引起人们的重视。到 1994 年,随着 WWW(万维网)在 Internet 快速的发展,Sun 公司遂将其用于 Internet 网

络,并改名为 Java。

Sun 公司于 1995 年正式发表了 Java 语言。Java 语言一推出,便以其网络上编程的独特优势赢得许多著名公司的青睐,得到

Netscape, IBM, Microsoft Oracle 等大公司的支持,Java 迅速得到推广。

1996 年, Sun 公司成立了维护 Java 的 Javasoft 分公司。

Java 的开发环境有不同的版本,如 Sun 公司的 Java Development Kit(JDK)。目前 Microsoft 公司也推出了支持 Java 规

范的 Microsoft Visual J++ 开发环境,借助 Microsoft 的强大优势迅速推广,与 Sun 公司平分秋色。

1.2 Java 与 Internet 关系

随着国际互连网络的发展,给人们带来了极大的方便,并实现了资源共享。资源共享是 Internet 的一个重要特点,Internet 的一个

重要功能是由于当今火热的电子货币、电子商务。电子商务使商业流通更迅速,减少了中间环节,给消费者以足不出户的服务,

这也是 Internet 在应用上的一大突破。然而传统的编程语言却难以胜任电子商务系统,它要求程序代码具有基本的要求:安全、可靠,

同时要求能与运行于不同平台机器的全世界客户开展业务。Java 凭借它在语言上无法比拟的优势成为网络编程语言,并以其强安

全性、平台无关性、硬件结构无关性、语言简洁同时集面向对象等特性,应用于电子商务。

Java 根植于网络,网络的发展促进

了 Java 的规范,在 1.4 节将详细介绍 Java 的特点。Internet 的服务种类丰富,应用最广泛

的有 WWW(world Wide Web) 世界万维网服务, Gopher 服务, 文件传输服务, 远程终端服务, E-mail 电子邮件服务, 网络论坛, 各电子公告栏信息服务, 网上购物等。

Web 页是发布消息、相互交流的重要方式之一。Web 华丽的页面, 不可思议的超链接功能, 人机交互的功能, 生动的动画效果等, 都深深地吸引着访问者。Web 页由网络浏览器装载。由 Java 编写的程序代码可以嵌入 Web 页在浏览器上运行, 可以轻松地实现动画、人机对话和事件处理等功能, 于是 Java 便成为编写电子商务系统的首选语言。

Java 与 Web 联系十分紧密, Java 在 Web 上充分显现出它的强大功能。网络服务离不开网络服务器, 为此, 下面将介绍网络基本的服务器, 以帮助读者了解因特网的基本常识、Web 有关的 HTML(超文本)及 Web 服务器, 让读者更清楚地认识 Java 并用好它。

1.3 超文本 HTML 与 Java

超文本自从它诞生以来, 就一直得到运用和发展。最初的超文本只能进行单调的文本链接, 也即是进行方便的资料检索。

它是图形界面, 点击鼠标的操作一目了然, 迅速被大众接受。随后, 它的概念得到了发展, 功能得到极大的扩充。现在, 它具有动画和多媒体功能, 通过它为中介, 读者可以获得播放音乐、收看电影等网上服务, 但无论如何这些先进的功能都离不开 Java。

正是 Java 的小应用程序 Applet 嵌在 HTML 文档上, 实现了引人入胜的多媒体功能。如果读者想要知道 Java 是如何实现这个功能的, 首先, 了解 HTML 的实质及简单的制作是必要的。

◎ 超文本的实质

例如 Web 上的水平直线, 在超文本上的符号为: <hr> 在文本编辑器上是不易作出的。文本编辑器则可以容易地实现标签

的编辑。我们可以称 HTML 语言为标签语言。下面介绍超文本的语法及标签。

◎超文本的标签

超文本有两种标签: 第一种是单个标签, 格式为: <HTML 标签>; 例如:<p>标签表示另取一段。

第二种是成队标签, 格式为: <HTML 标签...文本..HTML 标签>; 超文本标签有的需要成对出现, 有的可单独出现, 视具体标签而定。

◎超文本的制作

1.手工制作(例子)

```
<head>
<html>
<title>Java 程序设计语言入门与提高</title>
<body>
<center>
<h2>Java 程序设计语言入门与提高</h2>
</center>
欢迎学习 Java 程序设计语言,在这本书中你将领略到.....<p>
...<p>
```

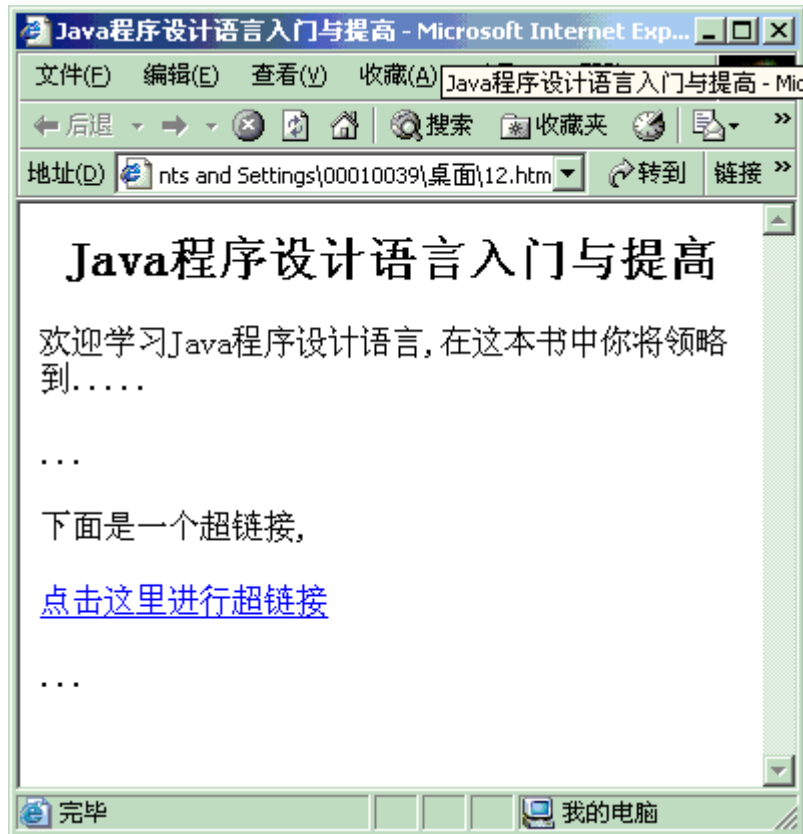
下面是一个超链接,<p>

点击这里进行超链接<p>

...

</body>

</html>



1.4 Java 的特点

随着 Internet 的迅猛发展,Java 应运而生。当 WWW 兴起时它就被众人看好,1995 年被评为十大优秀科技产品之一,一度

掀起学习 Java 的热潮。Java 与传统的编程语言相比,有如下几大特点:

1.平台无关性

平台无关性就是指 Java 能运行于不同的平台。Java 引进虚拟机原理,并运行于虚拟机,实现不同平台的 Java 接口之间。

使用 Java 编写的程序能在世界范围内共享。Java 的数据类型与机器无关,Java 虚拟机 (Java Virtual Machine) 是建立在硬件

和操作系统之上,实现 Javae 代码的解释执行功能,提供于不同平台的接口的。

2.安全性

Java 的编程类似于 C++,学习过 C++的读者将很快能掌握 Java 的精髓。Java 舍弃了 C++的指针对存储器地址的直接操作,

程序运行时,内存由操作系统分配,这样可以避免病毒通过指针侵入系统。Java 对程序提供了安全管理器,防止程序的非法访

问。

3.面向对象

Java 吸取了 C++面向对象的概念,将数据封装的简洁性和便于维护性。类的封装性、

继承性等有关对象的特性,使程序

代码只需一次编译,然后通过上述特性到达反复利用上。**Java** 提供了众多的一般对象的类,通过继承即可使用父类的方法。

Java 已有的类的能强大,如输入输出类,进行普通输入输出和文件设计图形界面的常用部件,图形用户界面的设计上给程序员很大的方便。

4.简单性

Java 舍弃 C++ 的头文件,没有全局变量,这样可节省维护头文件是不符合网络编程的。头文件中有全局变量的定义,这对可维护性和安全性都是不可取的。**Java** 舍弃了 C++ 的多重继承,引进了垃圾管理上绞尽脑汁去设计。

5.动态特性

Java 源程序经过编译后生成的二进制进制码存于网络计算机中。当 **Java** 运行的时候,动态地加载,即当程序运行到所需类时,便在网上寻找,下载到本地盘上,便于在网络上运行。

6.分布性

Java 允许将编译后的 M 进制代码分布存于网络上。应用程序可以通过 URL 统一资源定位来寻找应用程序所需的类,跟访问本地机一样。

7.多线程

多线程是 **Java** 的一大特点,能够在程序中实现多任务操作。传统的程序设计语言的程序只能单任务操作,效率非常低,

例如程序往往在接收数据输入时被阻塞,只有等到程序获得数据后才能够继续运行。而多线程程序可以创建一个线程来进行输

入输出操作,创建另一个线程在后台进行数据处理,输入输出的线程在接收数据时阻塞,而另一个线程仍然在运行。这样,多

线程程序大大提高了运行的效率和处理能力。

Java 提供了有关线程的操作、线程的创建、线程的管理、线程的废弃等处理。**Java** 虚拟机也是一个多线程程序,虚拟机

启动后,时刻在运行一个线程,该线程的优先级最低,在后台负责不用对象的垃圾处理工作。

多线程使程序能够处理多任务,

具有非常广阔的发展前景。

第二讲:认识 **Java** 程序语言

Java 是有 **SUN** 公司开发的新一代编程语言,它可以用在各种不同的机器、操作系统的网络环境中进行开发。不论你使用哪种

浏览器或者使用哪种操作系统(**Windows**、**Unix** 等等),只要浏览器支持 **JAVA**,你就可以看到生动的主页。**JAVA** 正在逐步成为

Internet 应用的主要开发语言,它彻底改变了应用软件的开发模式,为迅速发展信息世界增添了新的活力。所以作为 **Internet**

应用的开发技术人员不可不看 **JAVA**,而 **JAVA** 程序不可不先从基础学起,这正是 **JAVA** 程序员必读:认识 **Java** 程序语言 的用意。

2.1 **Java** 的几个基本概念

◎ 面向对象编程概念

这里将教会你面向对象编程的核心概念。

对象 消息 对象的生命周期 类和继承 接口和包

◎ 编程中的一些共同的问题

问题一：编译器找不到类。

解决方法：

1. 确保你已经导入了类或者它的包。
2. 如果对 `CLASSPATH` 环境变量有进行设置，要重新复位。
3. 确保类名的拼写跟定义的一样，要注意大小写问题。
4. 如果类在包中，要确保它们处在正确的子目录中。

同时，一些程序员从 `.java` 文件名字为类使用不同的名字。要确保你是使用类名字而不是文件名。实际上，使类名和文件名相同就不会出现这个错误了。

问题二：注释器不能找到其中一个类

解决方法：

1. 确保你指定的是类名而不是类的文件名。
2. 如果对 `CLASSPATH` 环境变量有进行设置，要重新复位。
3. 如果类在包中，要确保它们处在正确的子目录中。
4. 确保你从 `.class` 文件所在的目录中调用这个注释器。

问题三：程序不能工作？究竟出了什么错误？

以下是 `JAVA` 新手犯的公共错误，注意以下的各条：

1. 你是否忘记在在 `switch` 语句中的每一个 `case` 语句使用 `break`？
2. 你是否在应该使用比较运算符 `==` 的时候使用了赋值运算符 `=`？
3. 在循环语句中的终止条件是否正确？确保你没有过早或者过迟终止循环。也就是说，确保正确使用 `<` 或 `<=` 或 `>` 或 `>=` `a` 运算符。
4. 记住数组的索引是从 `0` 开始的，因此数组的循环应该是：

```
for (int i = 0; i < array.length; i++)
```

...

5. 你是否在比较浮点型数使用了 `==`？大于号和小于号 (`>` 和 `<`) 运算符在对浮点数的条件逻辑中更合适。
6. 你是否对封装、继承或者其它面向对象编程和设计概念理解有问题？
7. 确保语句块圈在大括号 `{}` 和 `}` 中间。下面的代码块看起来好象是对的，因为它采用缩进的编写，但是你仔细看这里缺少了

`{}`：

```
for (int i = 0; i < arrayOfInts.length; i++)
```

```
arrayOfInts[i] = i;
```

```
System.out.println("[i] = " + arrayOfInts[i]);
```

8. 你是否正确使用条件运算符？要确保理解 `&&` 和 `||` 以及正确使用它们。
9. 你是否使用了否定运算符 (`!`)？尽量不要使用它。这样会减少错误的发生。

10. 你是否使用了 **do-while** 语句。如果有，你知道 **do-while** 语句至少执行一次吗？它跟 **while** 循环语句不一样的，它可以连一次都不执行。

11. 你是否想从方法中改变参数的数值？在 **JAVA** 中的参数是由参数来传递的，它不能在方法中改变。

12. 你是否无意地增加一个分号（;）来过早终止语句？比如：

```
for (int i = 0; i < arrayOfInts.length; i++);  
arrayOfInts[i] = i
```

2.2 类和继承

◎ 什么是类

类实际上是对某种类型的对象定义变量和方法的原型。具体的说类就是数据和方法的封装，数据可以确定对象的属性，方法

使对象能执行一定的动作。

在现实世界中，你经常看到相同类型的许多对象。比如，你的自行车只是现实世界中许多自行车的其中一辆。使用面向

对象技术，我们可以说你的自行车是自行车对象类的一个实例。通常，自行车有一些状态（当前档位、两个轮子等等）以及行为

（改变档位、刹车等等）。但是，每辆自行车的状态都是独立的并且跟其它自行车不同。

类的特点：1. 类的封装性(Encapsulation) 2. 类的继承性(Inheritance) 3. 类的多态性(Polymorphism)

当厂家制造自行车的时候，厂商利用了自行车共有的特性来根据相同的蓝图制造许多自行车。如果制造一辆自行车就要

产生一个新蓝图，那效率就太低了。在面向对象软件中，同样地，可以让相同种类地许多对象来共有一些特性，比如矩形、雇员

记录、视频夹等等。就象自行车制造商人，你可以利用相同种类的对象是相似的事实并且你可以为这些对象创建一个蓝图。对

象的软件蓝图叫做类。自行车的类需要定义一些实例变量来包括当前档位、当前速度等等。这个类将为实例方法定义和提供实施

方法，它允许骑车者改变档位、刹车以及改变脚踏板的节奏。

当你创建了自行车类以后，你可以从这个类创建任意个自行车对象。当你创建了一个类的实例后，系统将为这个对象和

的实例变量分配内存。每个实例将给所有实例变量的副本定义在类中。

除了实例变量，类还要定义类的变量。类变量包含了被类所有实例共享的信息。比如，假设所有的自行车有相同的档位

数。在本例子中，要定义一个实例变量来容纳档位数。每一个实例都会有变量的副本，但是在每一个实例中数值都是相同的。在

这样的情况下，你可以定义一个类变量来包含档位数，这样所有的类的实例都共享这个变量。如果一个对象改变了变量，它就为

改变那个类的所有对象。类同样可以定义类方法。你可以直接从类中调用类方法，然而你必须在特定的实例中调用实例方法。

◎ 理解实例和类成员(1)

下面详细讨论一下实例和类成员，具体涉及变量和方法以及类变量和方法：

你这样声明一个成员变量，比如在类 `MyClass` 中有一个 `float` 型的 `aFloat`:

```
class MyClass {  
  
    float aFloat;  
  
}
```

这样你就声明一个实例变量。每次你创建一个类的实例的时候，系统就为实例创建了类的每一个实例变量的副本。你可以从对象中访问对象的实例变量。

实例变量跟类变量是不一样的，类变量是使用静态修改量来声明的。不管类创建了多少个实例，系统为每个类变量分配了类变量。系统为类变量分配的内存是在它第一次调用类的时候发生的。所有的实例共享了类的类变量的相同副本。你可以通过实例或者通过类本身来访问类变量。它们的方法是类似的：你的类可以有实例方法和类方法。实例方法是对当前对象的实例变量进行操作的，而且访问类变量。另外一个方法，类方法不能访问定义在类中的实例变量，除非它们创建一个新的对象并通过对象来访问它们。同样，类方法可以在类中被调用，你不必需要一个实例来调用一个类方法。

缺省地，除非其它的成员被指定，一个定义在类中成员就是一个实例成员。这个在下面定义类有一个实例变量，有一个整型的 `x`，两个实例方法 `x` 和 `setX`，它们设置其它对象以及查询 `x` 的数值。

```
class AnIntegerNamedX {  
  
    int x;  
  
    public int x() {  
  
        return x;  
  
    }  
  
    public void setX(int newX) {  
  
        x = newX;  
  
    }  
  
}
```

每次你从一个类实例化一个新的对象，你可以得到每个类的实例变量的副本。这些副本都是跟新对象有关系的。因此，每次你从这个类实例化一个新的 `AnIntegerNamedX` 对象的时候，

你得以得到跟新的 `AnIntegerNamedX` 对象有关的新副本。

一个类的所有实例共享一个实例方法的相同的实行；所有的 `AnIntegerNamedX` 实例都共享 `x` 和 `setX` 的相同执行。这里注意，两个方法 `x` 和 `setX` 是指对象的实例变量 `x`。但是，你可能会问：如果所有 `AnIntegerNamedX` 共享 `x` 和 `setX` 的相同执行，会不会造成模棱两可的状态？答案当然是：不是。在实例方法中，实例变量的名字是指当前对象的实例变量，假如实例变量不是由一个方法参数来隐藏的。这样在 `x` 和 `setX` 中，`x` 就等价于这个 `x`，而不会造成混乱。

对于 `AnIntegerNamedX` 外部的对象如果想访问 `x`，它必须通过特定的 `AnIntegerNamedX` 的实例来实现。假如这个代码片

段处在其它对象的方法中。它创建了两两种不同类型的 `AnIntegerNamedX`，它设置了 `x` 为不同的数值，然后显示它们：

```
AnIntegerNamedX myX = new AnIntegerNamedX();
```

```
AnIntegerNamedX anotherX = new AnIntegerNamedX();
```

```
myX.setX(1);
```

```
anotherX.x = 2;
```

```
System.out.println("myX.x = " + myX.x());
```

```
System.out.println("anotherX.x = " + anotherX.x());
```

这里注意，代码使用 `setX` 来为 `myX` 设置 `x` 的值，而直接给 `anotherX.x` 指定一个数值。不管用什么方法，代码是在操作两个不同的 `x` 副本：一个包含在 `myX` 对象中一，另外一个包含在 `anotherX` 对象中。其输出是用以下代码片段来实现的：

```
myX.x = 1
```

```
anotherX.x = 2
```

上面代码显示了类 `AnIntegerNamedX` 的每一个实例有自己实例变量 `x` 的副本以及每个 `x` 有自己的数值。

你可以在声明成员变量的时候，指定变量是一个类变量而不是一个实例变量。相似地，你可以指定方法是一个类方法而不是一个实例方法。系统在第一次调用类来定义变量的时候创建了一个类变量的副本。所有的类实例共享了类变量的相同副本。类方法可以只操作类变量，它们不能访问定义在类中的实例变量。

◎ 理解实例和类成员 (2)

为了指定一个成员变量为一个类变量，你可以使用 `static` 关键字。比如，我们可以修改一下上面的 `AnIntegerNamedX` 类，使得 `x` 变量现在是一个类变量：


```
class AnIntegerNamedX {  
  
    static int x;  
  
    public int x() {  
  
        return x;  
  
    }  
  
    public void setX(int newX) {  
  
        x = newX;  
  
    }  
  
}
```

现在设置它们的 **x** 数值并显示不同的输出：

```
myX.x = 2
```

```
anotherX.x = 2
```

这次的输出不同，是因为 **x** 现在是一个类变量，所以就只有这个变量的副本，它是被 **AnIntegerNamedX** 的所有实例所共享的，包括 **myX** 和 **anotherX**。当你在其它实例中调用 **setX** 的时候，你可以为所有的 **AnIntegerNamedX** 的所有实例改变 **x** 的数值。

同样，当我们声明一个方法的时候，你可以指定方法为类方法而不是实例方法。类方法只可以在类变量中进行操作，并且不能访问定义在类中的所有实例变量。

为了指定方法为类方法，你可以在方法声明处使用 **static** 关键字。下面，我们再次来修改 **AnIntegerNamedX** 类，使它的成员变量 **x** 为一个实例变量，以及它的两个方法为类方法：

```
class AnIntegerNamedX {  
  
    int x;  
  
    static public int x() {  
  
        return x;  
  
    }  
  
}
```

```

static public void setX(int newX) {

    x = newX;

}

}

```

当你想编译这个版本的 `AnIntegerNamedX`，编译器就会显示如下的错误：
`AnIntegerNamedX.java:4: Can't make a static reference to`

`nonstatic variable x in class AnIntegerNamedX.`

```

return x;

```

```

^

```

出现这些错误的原因是类方法不能访问实例变量，除非方法先创建 `AnIntegerNamedX` 的一个实例并且通过它来访问变量。

◎ 理解实例和类成员 (3)

下面我们修改一下 `AnIntegerNamedX`，让 `x` 变量成为类变量：

```

class AnIntegerNamedX {

    static int x;

    static public int x() {

        return x;

    }

    static public void setX(int newX) {

        x = newX;

    }

}

```

现在为 `x` 设置数值，并打印出 `x` 数值：

```

myX.x = 2

```

```
anotherX.x = 2
```

再一次，我们通过 `myX` 来改变 `x`，并将它改变为 `AnIntegerNamedX` 的其它实例。

实例成员和类成员之间的另外一个差别是类成员可以从类本身进行访问。你不必实例化类来访问它的类成员。下面让我们编写一段代码来直接从 `AnIntegerNamedX` 类中访问 `x` 和 `setX`:

```
...
```

```
AnIntegerNamedX.setX(1);
```

```
System.out.println("AnIntegerNamedX.x = " + AnIntegerNamedX.x());
```

```
...
```

值得一提的是，你现在已经不用再创建 `myX` 和 `anotherX` 了。你可以设置 `x` 并直接 `AnIntegerNamedX` 类中检索 `x`。你不能利用实例成

员来处理它，你只能从一个对象来调用实例方法并且只可以从对象中访问实例变量。而你可以从类的实例或者从类本身来访问类变量和方法。

2.3 函数

在 C 和 C++ 程序中，函数是普遍应用的，定义函数的格式为：

```
[函数返回值的类型] 函数名 ([参数类型参数名 1, 参数类型参数名 2, ...])
{[局部变量类型局部变量 1;]
 [局部变量类型局部变量 2;]
 ...
 [执行语句或被调函数;]
 [...]
 [return;]
}
```

[]: 符号表示该内容可有可无;

[函数返回值类型]: 函数可以有返回值，也可没有返回值，有返回值需要说明其类型，如整型（用关键字 `int` 声明），没

有返回值则用 `void` 代替 [函数返回值类型];

函数名: 函数都有函数名，使用函数名来对函数操作，如调用函数进行数值计算，调用函数进行一定的动作，进行输入输

出操作等等;

(): 表示此是一个函数，函数可以有参数，也可以没有参数。利用参数可以实现主调函数对被函数的控制，也可以通过

参数传递来将要参加计算的数传给函数去进行计算。

{...}: 为函数体, 要执行的语句放入函数体中, 每一语句以分号“;”作为结尾的标志。

例如, 定义一个函数使其在屏

幕上输出 “hello world “这句话:

```
void print()
{
    printf("hello world");
}
```

该函数无需返回值, 因而设为 void 类型。

函数体内将 printf()函数作为一条语句, 用以实现对系统输出函数的调用, “hello world”是 printf()函数的参数, 该语句同样要以 “。” 结尾。

又如定义一个函数计算两数的积:

```
int yunsuan(int x,int) /*实现乘积运算的函数（功能模块）*/
{
    int xx; /*声明局部变量存储 x, y 的乘积*/
    xx=x*y.
    return(xx) /*将值返回*/
}
```

/*...*/ 为注释语句, 可以用 “/*...*/” 进行块注释, 也可用 “//” 进行注释; 该函数的 返回值类型为整型 nit, 声明两个整型参数 x,y。

2.4 方法

当函数定义在类外称为函数, 定义在类中, 我们就称该函数为方法, 而不再称为函数。这个概念是很重要的, 举一个例子来说明:

```
class myfirstclass
{
    void printee() //定义在类中, 称为方法
    {
        System.out.println(mystring).
    }
}
```

方法是引进面向对象的概念后, 在类的定义中使用的称呼; 称之为方法而非沿用函数的称谓, 这是面向对象的概念的要求

。方法与函数的区别是显而易见的, 方法定义在类中, 是类的一部分; 而函数定义在类的外面, 不属于任何类。在 Java 中, 只有定义在类中的函数, 称为方法; 没有定义在类外面的函数。因此 java 只有方法没有函数, 类是组成 Java 程序的基本单元。在 C++中, 既有定义在类外不属于类的函数, 因此 C++既有函数的又有方法。C++程序是由类和函数组成的。

◎ 本讲实例

1.说明 C（或 C++）入口的一个例子

```
#include<stdio.h>
int yunsuan(int x,y)
{int xx;
xx=x*y;
return(xx);
}

main()
{ int a;
int b;
int c;
a=10;
b= 100;
c=yunsuan(a,b);

printf("a=10,b=100,a 乘以 b 等于",c);
}

//包含 C 语言的输入输出头文件
//实现乘积运算的函数（功能模块）
//主函数 main（）c 程序的入口点
//C 程序的变量声明
//变量赋值
//调用 yUnsuau（）函数（功能模块）进行运算
//“a=10,b=100,a 乘以 b 等于”及变量 c 为函数 printf()的参数，调用 printf()将结果输出
```

2.说明 Java 程序入口的一个例子

```
class myfirstclass //自定义的一个类 class
{
String mystring=new String("This is my first class");//定义 myclass 类的一个成员 mystring
void printee() //定义 myclass 类的一种方法 printee()
{
System.out.println(mystring);
}
//以上是自己定义的第一个类，用来生成一个实例对象
//以下为该程序的主类
class helloworld //声明主类 helloworld
{ public static void main(String args[]) //声明主方法 mian(),作为程序的入口点
{
one rr=new one(); //声明一个实例对象
```

```

rr.printee(); //调用对象 rr 的方法 printee()
System.out.println("hello world");
}
}

```

现在我们知道，一个 java 应用程序至少需要由以下组成部分：

第一，至少要含有一个主类，存储该类的文件名应为主类类名

第二，主类中应该有主方法 main()。

我们知道 Java 是由对象组,对象由类来生成,类也就成了 Java 编程的主要集中点,通过编写类来实现功能各异的实例对

象。举一个例子,类相当于汽车零件生产模具,可以生产许多相同型号的零件;主类相当于一个装配车间,将各零件对象组装成

一有机统一体即汽车。扩充 Java 程序就是设计编写类,然后将其对象在主类的主方法中装配。

习 题

1. 叙述一下目前自己对面向对象这一概念的深刻理解。

2. 简述一下以下几个概念, 便加以举例:

*对象 *类

3. 类具有_____、_____、_____等特点。

4. 定义一个时间的类, 并用该定义生成一个对象 day1。

5. 定义一个函数 add, 计算整型变量 X 与 Y 的和, 并将结果赋值给变量 XY。要求加以必要的注释。

6. Java 应用程序由_____、_____等部分组成。

第三讲:Java 的基本语法

3.1 Java 语言的基本组成

Java 语言主要由以下五种元素组成: 标识符、关键字、文字、运算符和分隔符。这五种元素有着不同的语法含义和组成规则,

它们互相配合, 共同完成 Java 语言的语意表达。下面我们分别予以讲解。

3.1.1 标识符

变量, 以及我们后面将要讲到的类和方法都需要一定的名称, 我们将这种名称叫做标识符。什么是有效的标识符呢? 在 Java

中, 所有的标识符都必须以一个字母、下划线_或美元符\$作开头。后面的字符包括数字 (0-9), 所有从 A-Z 的大写字母和 a-

z 的小写字母, 以及所有在十六进制 0xc0 前的 ASCII 码。

以上只是标识符命名的基本规则, 以下是一个正误对照表, 通过它会对标识符的命名规则有一个具体的了解:

注: boolean 为关键字

3.1.2 关键字

关键字是 Java 语言本身使用的标识符, 它有其特定的语法含义。所有的 Java 关键字不能被用作标识符, 如: for、while、

boolean 等都是 Java 语言的关键字。本课件后附有 Java 语言关键字列表。

3.2 变量

为了在 Java 中存储一个数据，必须将它容纳在一个变量之中。定义一个变量的两个基本要素是：类型和标识符，通常你可以用如下语法去说明变量：

`type identifier[,identifier];` 该语句告诉编译器用“type”的类型和以“identifier”为名字建立一个变量，这里的分号将告诉编译器这是一个说明语句的结束；方格中的逗号和标识符表示你可以把几个类型相同的变量放在同一语句进行说明，变量名中间用逗号分隔。

在你创建了一个变量以后，你可以给它赋值，或者用运算符对它进行一些运算。如同上一节内容中所提到的一样，类型将决定变量所代表的不同种类的数据，在 Java 语言中有两种变量。最基本的是简单类型变量，他们不建立在任何其他类型上，整数、浮点、布尔和字符类型都是这类型；另外 Java 可以定义构造另一种变量类型：类，这些类型建立在简单类型之上，它包括数值、变量和方法，是一种数据与代码相结合的复合结构。

3.2.1 整型变量的说明

整型变量按所占内存大小的不同可分为四种不同的类型，最短的整型是 `byte`，它只有八位长，然后是短整型 `short`，它有 16 位，`int` 类型有 32 位，长整型 `long` 是 64 位，下面是这些整型变量的说明示例。

```
byte a;
short row;
int numberOfSportor;
long treeCount;
int x,y,z;
```

3.2.2 浮点变量的说明

浮点类型可用关键字 `float` 或 `double` 来说明，`float` 型的浮点变量用来表示一个 32 位的单精度浮点数，而 `double` 型的浮点变量用来表示一个 64 位的双精度浮点数。`double` 型所表示的浮点数比 `float` 型更精确。

```
float alpha;
double speed;
```

3.2.3 字符变量说明

Java 使用 16 位的 Unicode 字符集。因此 Java 字符是一个 16 位的无符号整数，字符变量用来存放单个字符，它不是完整的字符串。示例如下：

```
char a;
a='c';
```

3.2.4 布尔变量说明

布尔型有真和假两个逻辑值，另外，逻辑运算符也将返回布尔类型的值，示例如下：

```
boolean mouseOn;
mouseOn=true;
```

布尔型是一个独立的类型，它不像 C 语言中的布尔型代表 0 和 1 两个整数，由于这个原因，Java 中的布尔类型不能转换成数字。

3.2.5 变量的使用范围

当你说明了一个变量后，它将被引入到一个范围当中，也就是说，该名字只能在程序的特定范围内使用。变量的使用范围是从它被说明的地方到它所在那个块的结束处，块是由两个大括号所定义的，例如：

```
class Example
```

```
public static void main(String args[])
```

```
    int i;
```

```
    .....
```

```
public void function()
```

```
    char c;
```

```
    .....
```

整型变量 `i` 在方法 `main` 中说明，因为 `main` 的块不包括 `function` 块，所以任何在 `function` 块中对 `i` 的引用都是错误的。对字符型变量 `c` 也同样如此。

在某一个特定情形中，变量能被别的变量所隐藏，如：在一个块中说明一个变量，而在这个块中建立一个新块并且在其中定义相同名字的变量，这样在第二个块中，程序对该变量的使用均是指第二次定义的那个变量。这样我们说，第一个变量被隐藏了。变量隐藏的示例如下：

```
class Example
```

```
public static void main(String args[])
```

```
    int i; // * * *
```

```
    boolean try=true;
```

```
    while(try)
```

```
        int i; //以下对变量 i 的引用均指这里定义的 i
```

```
        .....
```

```
        //以下对变量 i 的引用均指* * *处定义的 i
```

```
        .....
```

当你定义一个变量时，首先必须明确它的活动范围，并根据它的实际功能来命名，此外还应尽量使用详细的注释，这些办法可以使你能够清晰地区分变量，变量被隐藏的问题也会大大减少。

3.2.6 类型转换

系统方法 `System.in.read` 返回一个整型数值，但你却常常想要把它当作一个字符来使用。现在的问题是，当有一个整数而你需把变成一个字符时应当去做些什么呢？你需要去做一个类型转换为一个字符。从一种类型转换到另一种类型可以使用下面的语句：

```
    int a;
```

```
    char b;
```

```
    a=(int)b;
```

加括号的 `int` 告诉编译器你想把字符变成整型并把它放在 `a` 里，另一方面，如果你想做相反的转换，你可以使用：

```
    b=(char)a;
```

记住整型和字符型变量位长不同是非常重要的，整型是 32 位长，字符型是 16 长，所以当你从整型转换到字符型可能会丢失信息。同样，当你把 64 位的长整型数转换为整型时，

由于长整型可能有比 32 位更多的信息，你也很可能会丢失信息。即使两个量具有相同的位数，比如整和浮点型（都是 32 位），你在转换小数时也会丢失信息，Java 不象 c/c++那样允许自动类型转换，当你进行类型转换要注意使目标类型能够容纳原类型的所有信息，不会丢失信息的类型转换有：

原始类型	目标类型
byte	short,char,int,long,float,double
short	int,long,float,double
char	int,long,float,double
int	long,float,double
long	float,double
float	double

表 3-3 不会丢失信息的类型转换列表

需要说明的是，当你执行一个这里并未列出的类型转换时可能并不总会丢失信息，不过进行这样一个理论上并不安全的转换总是很危险的。

3.3 运算符

任何语言都有自己的运算符，Java 语言也不例外，如+、-、*、/等都是运算符，运算符的作用是与一定的运算数据组成表达式来完成相应的运算。对不同的数据类型，有着不同的运算符，在你定义了的变量之后，通常都要对变量赋值、改变变量的值和使用变量进行计算，这都是运算符的作用。

3.3.1 优先级

表 3-4 按从高到低的优先级列出了运算符。同一行中的运算符优先级相同。

.	[]	()	
++	--	!	~
*	/	%	
+	-		
<<	>>	>>>	
<	>	<=	>=
==	!=		
&			
^			
&&			
?:			
=			

表 3-4 运算符的优先级

优先级是指同一式子中多个运算符被执行和次序，同一级里的操作符具有相同的优先级，例如对于表达式：

$a=b+c*d/(c^d)$

Java 处理时将会按照表 3-4 从最高优先级到最低的次序进行，在本例中，因为括号优先级最高，所以先计算 c^d ，接着是 $c*d$ ，然后除以 c^d ，最后，把上述结果与 b 的和存储到变量 a 中，不论任何时候，当你一时无法确定某种计算的执行次序时，可以使用加括号的方法明确为编译器指定运算顺序，这也是提高程序可读性的一个重要方法。

3.3.2 整型运算符

整型运算符按操作数的多少可分为一元和二元两类，一元运算符一次对一个变量进行操作，二元运算符一次对两个变量进行操作。对于运算来说，如果有一个变量或操作数是长整型的，那么结果就肯定是长整型的，否则即使操作数还没有确定是字节型、短整型或字符型，运算结果都是整型，表 3-5 是一元运算符的列表。

运算符	实际操作	例子
-	改变整型数符号	-i
~	位运算：非	~i
++	加 1	i++
--	减 1	i--

表 3-5 一元运算符列表

注：一元取反运算符（ \sim ）用来改变整数的正负号，逐位求反把所变量所有是 1 的位变成 0，是 0 的位变成 1，加加和减减（ $++$ ， $--$ ）把变量的值加 1 或减 1。

以下是一个具体例子：

```
int i=0;
int j=1;
for (i=1;i<10;i++)

    j--;
    System.out.println(i+" "+j+"");

    这个程序的一行执行加运算，另一行执行减运算，注意“++”和“--”的使用，每次遇到“++”或“--”的时候，系统就把操作数的值相应地加 1 或减 1，一元运算符执行的方式是改变它们所作用的变量的值，对一元取反和逐位求反来说，变量的值并不改变，而对加和减运算（ $++$ ， $--$ ）来说，就量的值被改变了，下面的代码给出了这种工作方式的一个例子：
    int i=10,j=10,k=10,l=10;
    System.out.println(i+" "+j+" "+k+" "+l+"");
    j++;
    i--;
    ~k;
    -l;
    System.out.println(i+" "+j+" "+k+" "+l+"");
```

注意 j 和 i 被改变并打印出它们的新值，可是 k 和 l 仍然为原来的数值，当在复合表达式中使用一元取反和逐位求余运算时，你实际上是在用一个临时存贮操作数的新值。

++ 和 -- 既是前置运算符也是后置运算符，这就是说，它们既可以放在操作数前面(++x)，也可以放在后面 (x++)，如果它们被用在复合语句中如：

```
i=x++; 或 i=++x;
```

那么第一个语句中 x 把值赋给 i 以后再加 1，而第二个语句先把 x 加 1，再把新的 x 值赋给 i。

整型运算符的第二种类型是二元运算符，这种运算并不改变操作数的值，而是返回一个必须赋给变量的值，表 3-6 列出了二元运算符。

运算符	实际操作	例子
+	加运算	a+b
-	减运算	a-b
*	乘运算	a*b
/	除运算	a/b
%	取模运算	a%b
&	与运算	a&b
	或运算	a b
^	异或运算	a^b
<<	左移	a<>	右移	a>>b
>>>	右移	a>>>b

表 3-6 二元运算符列表

注意：>>与>>>的基本功能都是右移，但>>是用符号位来填充右移后所留下的空位，而>>>则是用零来填充右移后所留下的空位。

还有一类运算，它们形如：j-=i，这里-=是二元运算-和赋值运算=的复合运算，它等价于j=j-i，这种复合方式适用于所有的二元运算符。

这里有点关于整型运算进一步说明，首先，整数除法向靠近 0 的方向取整；其次，如果你除以 0 或者对 0 取模，程序就会在运行时被强行中止，如果你的运算结果超出最小极限，或者说是下溢，结果将为 0，如果超出了最大极限，就将会导致结果的回绕。

3.3.3 布尔运算符

另外还有一些运算符能产生布尔类型的结果，我们把它们称为布尔运算符，表 4-7 列出了这些运算符。

运算符	实际操作	例子
<	小于	a	大于	a>b

<=	小于等于	a<=b
>	大于等于	a>=b
==	等于	a==b
!=	不等于	a!=b

表 3-7 产生布尔类型结果的运算符

在这里要指出的是相等运算符==可能会给你带来许多麻烦,很多人在比较两个值时往往错用了等号=,以致于变成了赋值操作,所以在编程中一定要注意你在比较数值时用的是==。布尔类型本身还有几个逻辑运算符,它们可以对布尔类型的数据进行运算:

!,&&,//

其实如果你把布尔类型的值 true 等效地看成 1, false 看成 0, 那么对于相同的运算符, 整型运算的法则对布尔类型也是适用的。

布尔运算符中还有一种三元运算符, 它的格式如下:

Operand ? Statement1:Statement2

在这个式子中, 先计算 Operand 的真假, 若为真, 则执行 Statement1, 若为假, 则执行 Statement2。下面的代码给出了这种运算的一个例子:

(a>b)?a:b;

这个表达式将返回 a 和 b 中较大的那个数值。

3.3.4 浮点型运算符

传统的浮点型二元运算除了赋值运算符(+=,-,*,./)取模(%) and 取模赋值(%=0)使浮点数值等价于整数除法, 同样, ++和--使变量的值增加或减小 1。

同整型变量相似, 结果被放在最大长度的类型里, 如果运算包括两个浮点类型, 结果也是个浮点类型, 如果有一个或几个是双精度浮点型, 结果就是双精度浮点型, 当你使用关系运算符: >、<、>=、<=、==和!=时, 注意浮点变量与整型的不同, 只有 a==b 成立并不能说明 a<b 和 a>b 不成立, 这是因为浮点类型数值次序同整型不同, 当你写程序, 对浮点型数值进行假设时要小心, 浮点型数值也包括 inf, 代表无穷大, 溢出产生无穷大 inf, 下溢产生 0。

3.3.5 字符串运算符

+运算符可以把字符串并置起来, 如果哪一个操作数不是字符串, 在并置之前它会被转换成字符串, 另外, +=运算符把两个字符串并置的结果放进第一个字符串里, 在前面的例子里, 当你想把几项打印在同一行里时使用 +运算符, 试着在解释器里用 System.out.println 和+运算符来造成不同的组合输出。

3.3.6 数组

◎ 说明数组

数组是 Java 语言中的特殊类型。它们存放能通过索引来引用的一系列对象, 另外, 你可以定义数组的数组, 下面是一些示例:

```
int i[];
char c[];
float f[][];
```

◎ 数组运算符

同 C/C++不同, 数组的分配是通过使用 new 运算符建立数组然后把它赋给变量, 如:

```
int a[]=new int[10];
```

前面这个例子建立了一个包括 10 个整型变量的数组并把它赋给 a，你将得到按数字顺序的变量 a[0]，a[1]，.....，a[8]，a[9]，注意下标是从第一个元素的 0 开始，到数组个数减 1。

数组的使用与变量相同，每一个数组成员都可以被用在同类变量被使用的地方，Java 也支持多维数组。

```
char c[][]=new char[10][10];
```

```
float f[][]=new float[5][];
```

请注意在第二个说明中只有一维的尺度被确定，Java 要求在编译时（即在源代码中）至少有一维的尺度被确定了，其余维的尺度可以在以后分配。

数组主要用于你有大量相关数据想要存贮在一起而且能够简单地通过数字访问它们，数组是非常强有力的，在后面有章节中我们还会进一步讨论它。

3.4 Java 程序语句

前面四节我们已经学习了 Java 语言的基本组成，下面三节，我们将学习如何运用这些基本元素去构成 Java 程序。

首先我们学习最简单和最常用的循环语句。

Java 的循环语句分为三种：for、while 和 do，它们的语法定义如下：

```
for:          for(expression1;expression2;expression3)
               statement;

while:        while(boolean)statement;

do-while:     do statement while(boolean);
```

3.4.1 for 循环

Java 语言的 for 语句与 C 语言的用法完全一致，这一点通过下面的讲解可以看出。

for 循环结构在实现顺序递增直到达到某一极限的循环时是一个强有力的工具，for 语句的格式要求你把一个变量和一个确定的极限作比较，当达到极限时，中止循环，如：

```
for(i=0;i<10;i++).....;
```

下面解释变元的格式，在(ex1;ex2;ex3)中，第一个表达式指出 for 循环的变量初值，在这个例子中变量 i 被设为 0；第二个表达式，i<10，指出循环何时结束，在这个例子中，对 0-9 的数值表达式都为真(这就是说循环应该继续)；最后的表达式指出循环每次对变量做什么，在例子中，循环使 i 从 0 到 9，你可以使用这样的语句来顺序访问数组的各个成员：

```
int i,a[]=new int[10];
for (i=0,i<10;i++)a[i]=0;
```

这段代码把整型数组 a 中的所有元素都赋成 0。

你可以在 for 循环的头部说明你的变量，而且最后一个表达式可以省略，不过要确定在语句中对变量的值有所改变，如：

```
for(int i=10;i>=0;)i--;
```

下面 for 循环的例子将按 5 度的增量打印出一个从摄氏度到华氏度的转换表：

```
class TempConversion
```

```
public static void main (String args[])
```

```
int fahr,cels;
System.out.println("Celsius Fahrenheit");
for(cels=0;cels<=100;cels+=5)
```

```
fahr=cels*9/5+32;
System.out.println(cels+""+fahr);
```

3.4.2 while 和 while--do 循环结构

while 循环检查表达式的值是否为真，若为真，则执行给定语句，直到表达式的值为假，另一个 while--do 循环则执行给定的语句，再检查表达式，若表达式值为真则跳出循环，前面你已经使用过 while 循环来表示加法运算，你应该记得你一定改变了循环判断中的表达式的值，否则的话，如果它的值为真，你将进入一个死循环，因为它总是真，正如下面的例子：

```
boolean test=true;
while(test)

    System.out.println("Hey!.get me out of here!:);
```

你可以试着运行一下这些代码，这过千万不要等它结束(Ctrl-C 可以中断程序运行)，否则它会一直循环下去的。有些情况下，不管条件表达式的值是是真还是假，你都希望把指定的语句至少执行一次，那么就应使用 do--while 循环，下面是个例子：

```
boolean test=false;
do

    .....

while(test);
```

这种控制并不是很常用，但有时却非常重要，使用时注意结尾处 while 语句后的分号。

3.4.3 条件控制:if 和 switch

◎ if 语句

if 语句通常都与 else 语句配套使用，所以我们一般都把它叫做 if--else 语句，它的语法结构如下：

```
if--else:      if(boolean)statement1
                else statement2
```

当表达式 boolean 为真时，执行 statement1，否则执行 statement2。

下面是一个用 if--else 语句构造多分支程序的例子：

```
if(a>5) b=1
else if(a>4) b=2
else if(a>3) b=3
```

```
...  
else b=-1;
```

3.4.4 switch 分支结构

switch 分支结构实际上也是一种 **if--else** 结构，不过它使你在编码时很容易写出判断条件，特别是有很多选项的时候，它把括号里变量的值同每种情况列出的值做比较，如果相等，就执行后面的语句，如果不等，就执行 **default** 语句。在 **switch** 语句中，你通常在每一种 **case** 情况后都应使用 **break** 语句，否则，第一个相等情况后面所有的语句都会被执行，这种情况叫做落空。你可以试着分别加上和去掉 **break** 语句来执行下面的例子：

```
class Switch Test  
  
public static void main(String args[])  
throws java.io.IOException  
  
    char a;  
    System.out.println("Enter a number from 1--3:");  
    a=(char)System.in.read();  
    switch(a)  
  
        case'1':System.out.println("win a Car!");break;  
        case'2':System.out.println("picked the goat");break  
        case'3':System.out.println("get to keep your 100");  
        break;  
        default:System.out.println("entry");
```

在代码中加 **break** 语句后，你应明确知道程序将会发生的变化，并要确认程序没有转移到你不执行的代码上。

掌握了以上这些基本语句之后，你可以写更加实际点的程序了，下面这个程序计算从终端输入的数字和字符的数量。

```
class SwitchTest  
  
public static void main (String args[])  
throws java.io.IOException  
  
    int numberOfDigits=0,numberOfSpaces=0,numberOfOthers=0;  
    char c;  
    while((c=(char)System.in.read())!="")  
  
        switch(c)
```

```

case'0':
case'1':
...
...
case'8':
case'9':numberOfDigits++;break;
case' ':numberOfSpaces++;break;
default:numberOfOthers++;break;

```

```

System.out.println(":");
System.out.println("Number of digits="+numberOfDigits+""");
System.out.println("Number of spaces="+numberOfSpaces+""");
System.out.println("Number of others="+numberOfOthers+""");

```

代码的 `while` 循环从键盘读入字符直至输入一个回车符，循环内部的 `switch` 语句先把它和数字比较，当发现相等时，它就使对应的统计变量的值加 1，然后 `break` 语句结束 `switch` 语句，程序回到等待键盘输入的状态。程序中，在 `default` 语句中的 `break` 是不必要的，不过加上它可以使程序风格保持一致。

3.4.5 `break` 和 `label`

◎`break`

`break` 语句提供了一种方便的跳出循环的方法。

```

boolean test=true;
int i=0;
while(test)

    i++;
    if(i>=10) break;

```

执行这段程序时，尽管 `while` 条件表达式始终为真，全循环只运行 10 次。

3.4.6 标号 `label`

标号提供了一种简单的 `break` 语句所不能实现的控制循环的方法，当在循环语句中遇到 `break` 时，不管其它控制变量，都会终止。

但是，当你嵌套在几层循环中想退出循环时又会怎样呢？正常的 `break` 只退出一重循环，你可以用标号标出你想退出哪一个语句。

```

char a;
outer://this is the label for the outer loop
for(int i=0;i<10;i++)

    for(int j=0;j<10;j++)

```



```

a=(char)System.in.read();
if(a=='b')

    breakouter;

if(a=='c')

    continue outer;

```

在这个例子中，循环从键盘接受 100 个输入字符，输入“b”字符时，breakouter 语句会结束两重循环，注意 continue outer 语句，它告诉计算机退出现在的循环并继续执行 outer 循环。

读者可能会注意到 Java 语言没有 goto 语句，这对于那些习惯使用它的程序员来讲可能会有些不适应，但有经验的程序员一定会明白，goto 语句实际上是程序结构混乱的重要根源之一，它最终所引起的问题比它可能带来的好处要大得多。它一般用于比较特殊的问题，但却使你在日后花大量时间去找出一个程序是干什么的。正由于这些原因，Java 的设计者果断地删去了它。

习 题

- 下面标识符不合法的是（ ）。
 (A) abcd (B) \$0bf_A (C) %76bh (D) AAaa\$
- 下面标识符合法的是（ ）。
 (A) abstract (B) 1myclass (C) myclass (D) -37bc
- 整型变量数据类型 short 值范围为（ ）
 (A) -256---255 (B) -255---255 (C) -32767--32766 (D) -32768--32767
- 以下是长整型常量的是（ ）
 (A) 0x7fffffffL (B) 0x7ffff (C) 21478 (D) EE67
- 在 JAVA 语言中，表示回车的转义字符是（ ）
 (A) \ ' (B) \r (C) \f (D) \t
- 在 JAVA 的基本数据类型中 float 的预设值为（ ）
 (A) 0.0 (B) 0.0f (C) 0 (D) 0.0d
- 二维数组定义如下：
 int a[][]={{1,2,3},{4,5,6},{7,8,9}},则 a[1][1]=（ ）
 (A) 1 (B) 4 (C) 7 (D) 5
- 表达式 int a=10,b=100,c;
 c=a>b?a:b;
 最后的结果为（ ）

(A) 0 (B) 1 (C) 10 (D) 100

9、看以下逻辑运算：

```
boolean flag1=true;
boolean flag2=false;
boolean flag3
flag3=flag1&&flag2;
```

问 flag3 的值为 ()

(A) 1 (B) 0 (C) false (D) true

10、byte 型整数 24 的补码为_____，-24 的补码为_____，如果 24 是 short 型整数，则 24 的补码为_____，-24 的补码为_____。

11、op1 和 op2 为 BYTE 型整数，op1=24,op2=10,则 op1&op2=_____。

12、在屏幕上打印出“hello world!” Java 语句为_____。

13、用 if 语句编写一小程序：已知 a=10,b=20,如果 a<b 的话，a 与 b 进行交换，否则将 a 赋值于 b。

14、用 if...else、do...while,while 语句编写程序段：求和 1+2+3+4...+48+50+...100；

第四讲:Java 开发与运行环境

4.1 JDK 环境

Java 不仅提供了一个丰富的语言和运行环境，而且还提供了一个免费的 Java 开发工具集(Java Developers Kits，简称 JDK)。编程人员和最终用户可以利用这个工具来开发 java 程序或调用 Java 内容。JDK 包括以下工具：javac——Java 语言编译器，输出结果为 Java 字节码文件 class；Java——Java 字节码解释器；javap——Java 字节码分解程序，本程序返回 Java 程序的成员变量及方法等信息；javadoc——资源分析工具，用于分析 Java 程序在运行过程中调用了哪些资源，包括类和方法的调用次数和时间，以及各数据类型的内存使用情况等；javah——C 代码处理工具，用于从 Java 类调用 C++代码；AppletViewer——小应用程序浏览工具，用于测试并运行 Java 小应用程序。

Java 开发环境还包括 Java 类库(包括 I/O 类库、用户界面类库、网络类库等)和 HotJava WWW 浏览器。其中，HotJava 浏览器提供了在 WWW 环境下运行 Java 代码的一个运行系统，而且还为 WWW 开发人员提供了一个 Java 开发框架。Java 解释器是面向 Java 程序的一个独立运行系统，它可以一种稳定、高性能方式运行那些独立于平台的 Java 字节码，Java 编译器则用于生成这些字节码。

一 Java 源程序的编译

Java 源程序的编译程序是 javac.exe。javac 命令将 Java 源程序编译成字节码文件，然

后你可用 `java` 解释器 `java` 命令来解释执行这些 Java 字节码文件。Java 程序源码必须存放在后缀为 `.java` 的文件里。Java 源程序里的每一个类，`javac` 都将生成与类相同名称但后缀为 `.class` 文件。编译器把 `.class` 文件放在与 `.java` 文件的同一个目录里，除非你用了 `-d` 选项。当你引用到某个自己定义的类时，必须指明它们的存放目录，这就需要利用环境变量参数 `CLASSPATH`。环境变量 `CLASSPATH` 是由一组被分号隔开的路径名组成。如果传递给 `javac` 编译器的源文件里引用到的类定义在本文件和传递的其它文件中找不到，则编译器会按 `CLASSPATH` 定义的路径来搜索。例如：

```
CLASSPATH = .;C:\java\classes
```

则编译器先搜索当前目录，如果没搜索到，则继续搜索 `C:\java\classes` 目录。注意，系统总是将系统类的目录缺省地加在 `CLASSPATH` 后面，除非你用 `-classpath` 选项来编译。`javac_g` 是一个用于调试的未优化的编译器，功能与用法和 `javac` 一样。`javac` 的用法如下：

```
javac [-g][-O][-debug][-depend][-nowarn][-verbose][-classpath path][-nowrite][-d dir] file.java...
```

以下是每个选项的解释：

`-classpath path` 定义 `javac` 搜索类的路径。它将覆盖缺省的 `CLASSPATH` 环境变量的设置。路径是由一组由逗号隔开的路径名组成，一般格式如下：`.;<your_path>` 例如：`.;C:\java\doc\classes;C:\tools\java\classes` 表示编译器遇到一个新类，它先在本文件中查找它的定义，如果没有，则在本文件所处目录下其它文件中查找它的定义，如果还没有，则继续搜索 `C:\java\doc\classes` 目录中的所有文件，以此类推。

`-d directory` 指明类层次的根目录，格式如下：

```
javac -d <my_dir> MyProgram.java
```

这样将 `MyProgram.java` 程序里生产的 `.class` 文件存放在 `my_dir` 目录里。

`-g` 带调试信息编译，调试信息包括行号与使用 `java` 调试工具时用到的局部变量信息。如果编译没有加上 `-O` 优化选项，只包含行号信息。

`-nowarn` 关闭警告信息，编译器将不显示任何警告信息。

`-O` 优化编译 `static`，`final`，`private` 函数，注意你的类文件可能更大。

`-verbose` 让编译器与解释器显示被编译的源文件名和被加载的类名。

二 Java 程序的执行

`java -java` 语言解释器 `java` 命令解释 `java` 字节码

语法：`java [options] classname <args> java_g [options] classname <args>`

描述：`java` 命令由 `java` 编译器 `javac` 输出的 Java 字节码。

`classname` 参数是要执行的类名称。注意任意在类名称后的参数都将传递给要执行类的 `main` 函数。

`java` 执行完 `main` 函数后推出，除非 `main` 函数创建了一个或多个线程。如果 `main` 函数创建了其它线程，`java` 总是等到最后一个线程退出才退出。

选项：

`-cs`，`-checksource` 当一个编译过的类调入时，这个选项将比较字节码更改时间与源文件更改时间，如果源文件更改时间靠后，则重新编译此类并调入此新类。

`-classpath path` 定义 `javac` 搜索类的路径。它将覆盖缺省的 `CLASSPATH` 环境变量的设置。路径是由一组由逗号隔开的路径名组成，一般格式如下：

```
.;<your_path>
```

例如：`.;C:\java\doc\classes;C:\tools\java\classes`

表示解释器遇到一个新类，它先在本文件中查找它的定义，如果没有，则在本文件所处目录下其它文件中查找它的定义，如果还没有，则继续搜索 `C:\java\doc\classes` 目录

中的所有文件，以此类推。

-mx x 设置最大内存分配池，大小为 x，x 必须大于 1000bytes。缺省为 16 兆。

-ms x 设置垃圾回收堆的大小为 x，x 必须大于 1000bytes。缺省为 1 兆。

-noasyncgc 关闭异步垃圾回收功能。此选项打开后，除非显式调用或程序内存溢出，垃圾内存都不回收。本选项不打开时，垃圾回收线程与其它线程异步同时执行。

-ss x 每个 Java 线程有两个堆栈，一个是 java 代码堆栈，一个是 C 代码堆栈。-ss 选项将线程里 C 代码用的堆栈设置成最大为 x。

-oss x 每个 Java 线程有两个堆栈，一个是 java 代码堆栈，一个是 C 代码堆栈。-oss 选项将线程里 java 代码用的堆栈设置成最大为 x。

-v, -verbose 让 java 解释器在每一个类被调入时，在标准输出打印相应信息。

第五讲:Java 的类与对象

5.1.1 类的定义

定义一个类：

```
class line
{
    int color;
    float startpointx;
    float startpointy;
    float endpointx;
    float endpointy;
    void drawline ()
    { }
}
```

该类是关于一条直线的特征而定义的一个类，这个类的实例就是一个直线对象，通过该直线的 drawline()方法，可以将这条直线在屏幕上作出。类的定义由类名说明和类体说明两部分组成。上例的类定义中，class Line 就是类名说明部分。下面我们分别介绍定义类的两个部分。首先介绍类名部分的说明。

* 类名部分说明

类名说明的完整格式如下：

```
[modifiers] class classname [extends superclassname] [implements interfacename list]
{
    .....
}
```

modifiers:是类修饰符，对所定义的类加以修饰。类修饰符有如下几种：

(1) 权限修饰符：public,VJ++的修饰符没有 private;

(2) 最终类修饰符: **final**;

(3) 抽象类修饰符: **abstract**.

modifiers 为上述修饰符中任一个或它们的某种组合, 定义类时可以有也可以没有 **modifiers** 说明。在定义类时若没有用它说明, 则默认为非抽象的、非最终的、非公有的;

class:是定义类用的关键字, 要定义一个类必须用到 **class** 关键字;

classname:是所定义的类的类名, 为合法的标识符;

extends superclassname: 该说明项是继承关系说明项, 说明所定义的类是继承名为 **supername** 类而得来的。定义一个类可以特别说明是从哪一个父类继承而来, 也可以不加以说明。当没有用 **extends superclassname** 特别说明所继承的父类时, 则所定义的类的父类是 **Object**;

implements interfacenamelist: 该说明项说明所定义的类要实现的接口。要实现的接口可以是一个也可以是多个即一系列的接口, 定义的一个可以实现接口也可以不实现接口。若不用 **implements interfacenamelist** 加以说明, 则所定义的类没有实现任何接口。

* 类体说明

前面在介绍程序的人口点时我们知道了类结构, 对类体的组成有了一个形象的认识。类体由成员变量和成员方法组成, 一般成员变量在成员方法的前面说明, 也可以在方法后说明。成员变量有普通成员变量和静态成员变量, 我们就称普通成员变量为类的成员变量。静态成员变量为类变量。类体的说明可以表示如下:

```
class classname
```

```
{
```

```
    variabledeclare;
```

```
    functiondeclare;
```

```
}
```

类体的说明就是类的数据成员和成员方法的说明。关于成员变量和成员方法的具体说明, 将在下面的几节中分别说明。

5.1.2 类的访问权限

Java 的类都属于一个包, 一般将相关功能的类组织在同一个包中, 同一个包中的类往往能相互使用访问。对类的访问要检查要访问的类的权限, 若试图去使用一个没有访问权限的类, 就会出现编译错误。**Java** 用类的使用权限来管理类, 确定该类的有效位用范围。保证类得到正确安全的使用。下面介绍 **Java** 类的权限。定义类时可以用权限修饰符来说明类的权限。在 **VJ++** 中, 类权限修饰符为 **Public**; 当没有用权限修饰符明确说明时, 则类的隐含权限为 **friendly**, **friendly** 不是 **java** 的关键字, 不能用来说明类的权限。类的权限在类的定义时加以说明, 根据类定义的格式我们可以定义如下形式的类:

```
[rightmodifier] class classname
```

```
{
```

```
.....
```

```
}
```

rightmodifier:为类权限修饰符。类权限修饰符为 **public**.当没有用类权限修饰符说明时, 类的权限为 **friendly**;

若用 **Public** 说明, 则该类可以被包中其它类、对象及包外的类和对象访问;

当没有明确定义类的权限时，为 **friendly** 权限，只能被与它在同一包中的类访问；

5.1.3 成员变量的声明和初始化

成员变量的类型为 **Java** 的数据类型，包括简单数据类型和复合数据类型。成员变量声明的完整格式为：

```
class classname
{

[variablemodifiers] type variablename;
.....

}
```

variablenodifiers 是变量修饰符，**variablemodifiers** 修饰符有如下几种：

第一种：访问权限修饰符：**public,protected,private** 三种。这里 **VJ++** 不支持 **private,protected** 权限修饰符；

第二种：静态变量（又称类变量）修饰符：**static**,说明一个变量是共享变量即类变量；

第三种：常量说明符：**final**,作用是将以变量说明为一个值不变的常量。

5.1.4 成员变量的访问权限

类中的成员数据都有自己的权限，要访问类的成员变量，首先检验类的成员数据的权限。没有权限的对象或者类访问编译程序报错，编译通不过，保证了数据的安全性。成员变量的权限和类的权限一样，通过权限修饰符说明。成员变量的修饰符有：**private,protected,public**。

根据成员变量声明的语法格式，声明如下一个类：**class classname**

```
{

public int publicvariable;
protected int protectedvariable;
private int privatevariable;
.....

}
```

公共 **public** 成员变量能由包内和包外的类访问；

保护 **protected** 成员变量能由定义它的类本身及定义它的类的子类（必须在同一个包中）与它在同一个包中的其他的类访问。

私有 **private** 成员变量只能由定义它的类本身访问。

当没有权限说明，则称变量的权限隐含为 **friendly**。具有 **friendly** 权限的成员变量能够被与它在同一个包中的其他的类访问。

5.2 Java 的包

Java 程序编译的类被放在包内，要访问类就要给出类所属的包名，来指明类是在哪一个包中，以便能够找到该类。一个包中有许多类，同时还可以有子包。如我们会在应用程序中经常用到 **System.out.println()** 方法来讲结果输出。参看 **Java** 的包，我们知道 **System** 是一个类，

它属于 lang 包；同时，lang 又属于 java 包。指明类的位置是，当没有用 import 语句，要访问 System 类就首先要指明在哪个包中(用 VJ++编写程序时，每一个源文件都默认用 import 语句将 java.lang 包中的所有类引入，所以可以直接用类名 System 来访问类)。Java 用小圆点 "." 来说明包的这种包含关系，例如：

lang.System 表示 System 类属于子包 lang,java.lang 表示子包 lang 是属于 java 包，java 包是最外层包即根包。这样 java.lang.System 已经完全表明了 System 包的层次关系，根据这种关系，就能够找到 System 类并访问它了。 java.lang.System 称为访问类 System 的全限定名，用全限定名就可以访问类了，要访问类需要完全说明包的层次关系，例如 java 是最外层包，lang.System 不是类 System 的全限定名，不能用它来访问类 System。

当类有其静态成员变量和静态成员方法时，静态变量和静态方法能够被类直接访问，out 是 System 类的静态成员，out 有一个方法 println(),下面介绍怎样访问类的静态成员。

能够用类的全限定名访问类，同样可以通过类成员的全限定名访问类的成员，如 System 的成员 out 是一个对象，out 的全限定名为 java.lang.System.out;同时，对象 out 又有 println() 方法，则 println()方法的全限定名为：java.lang.System.out.println(),通过该名可以访问 println() 方法。

下面的程序是使用全限定名来访问方法的实例：

```
mainclass
{
public static void main(String arg[])
{

java.lang.String hello=new java.lang.String("hello world!");
java.lang.System.out.println(hello);

}

}
```

类的全限定名与类在文件系统中的存储结构即目录有对应关系，例如 java.lang.System 表示类 System 存储在 lang 目录中，lang 是 java 的子目录，java 是根目录。java.lang 对应的目录为：\java\lang

当包的层次很多，而类处于较内层的包时，则类的全限定名较长,例如有下面的层次的包：school. department. class. group

有名为 fly 的类在包 group 中，model 类在包 department 中，则：

fly 的全限定名为：school. department. class. group. fly

model 的全限定名为：school. department. model

显然用这样的名字来操作类，将是非常麻烦的。因此，Java 提供了 import 语句。下面介绍包操作语句 import 和 package。

5.2.1 包操作语句 import

指明将要访问的类所在的包，以便在当前目录找不到时，在 import 语句指明的包中寻找，若找不到类将出现编译错误。

包语句 import 的一般格式：

import 类名;

我们知道，访问类可以用全限定名来访问，当要访问的类在同一个包中时，这时可以直接用要访问的类的类名代替全限定名进行操作。例如，我们定义了两个类 `firstclass`、`secondclass`，它们都属于 `mylopbag.smallbag` 包，在 `secondclass` 类中可以直接用 `firstclass` 代替全限定名 `mylopbag.smallbag.firstclass` 来访问 `firstclass` 类。

当所要访问的类不在同一个包中时，就需要使用全限定名。全限定名往往很长，为此用 `import` 语句来减短访问类时用的名字。下面介绍 `import` 语句的两种用法。

用 `import` 语句指明要访问的类位置。我们知道，类 `System` 在子包 `lang` 中，子包 `lang` 在最外层包即根包 `java` 中，我们引入如下语句

第一种用法：`import java.lang.*;`

表示引入 `java.lang` 包内的所有类，当要访问包中的类时，直接用类名，如直接用类名为 `System` 访问类 `System`。同时，类 `System` 中定义了 `println()` 方法，用 `System.println()` 就可以调用 `println()` 方法。

第二种用法：`import java.lang.System;`

表示只引入 `java.lang` 包中的 `System` 类，访问包 `java.lang` 中的类时，只有 `System` 类能用类名来访问，包中的其他类需要全限定名来访问。

`System.println()` 调用 `println()` 方法，其它类的成员则需要用成员的全限定名来访问。用该语句能减少访问类时搜索的路径，提高运行的效率。

要注意的是 `import` 语句引入的是类，不能是子包，例如

`import java.lang.*;`

该语句在编译时会出错，因为 `lang` 是 `java` 的一个子包，不是类，而 `JVM` 认为 `lang` 是一个类，这样会出现找不到 `lang` 类的提示。用户在编程时，系统默认在每个源文件头加入了下面的 `import` 语句：

`import java.lang.*;`

因此在用到这条语句时，编程时无需用说明。`import` 语句在源文件的前面，一个源文件可以有多条 `import` 语句。`import` 语句在 `package` 语句后面。`import` 语句与 C 语言的 `include` 语句有本质的区别。`import` 语句只指明要用到的类所在的位置，以便能在用到时可以加载；而 C 语言用 `include` 语句将要用的文件包含在源文件中，作为源文件编译成一个模块。这体现了 Java 语言的特点，用户只需要将模块编译一次。当用户在编写另一个模块用到已经编译的模块时，只要告诉编译程序它的位置，如用全限定名或者用 `import` 语句，编译程序无需再一次编译已经编译的模块，就能够将源文件编译通过。

5.2.2 包操作语句 Package

`Package` 语句必须是源文件的第一条语句，且一个源文件只能有一条该语句。`Package` 语句的格式为：

`Package` 包名;

例如：

`Package myrootbag;`

也可以是：

`Package myrootbag.mysubbag;`

.....

5.3 java 对象

类名可以作为变量的类型来使用，如果一个变量的类型是某个类，那么它将指向这个类的实例，称为对象实例。所有对象实例和它们的类型都是相容的。就象可以把 `byte` 型的值赋给 `int` 型的变量一样，你可以把 `Object` 的子类的任何实例赋给一个 `Object` 型的变量。一个实例是类模板的单独的拷贝，带有自己的称为实例变量的数据集。每个实例也可以作为一个对象。当你定义一个变量的类型是某个类时，它的缺省值是 `null`，`null` 是 `Object` 的一个实例。对象 `null` 没有值，它和整数 `0` 不同。下面这个例子中，声明变量 `u` 的类型是类 `University`。

```
University u;
```

这里，变量 `u` 的值是 `null`。

对象通常由下面几个生成过程，首先是对象创建，然或是对象的活动期，对象的清除。

5.3.1 对象的创建

要创建一个对象，必须先定义一个类。定义一个类后，就可以创建该类的对象了。创建对象通过运算符 `new` 和与类同名的类构造方法来进行。创建对象的格式为：

```
new 构造方法名;
```

例如，创建一个字符串对象即 `String` 类的实例对象：

```
new String("hello");
```

我们可以直接使用创建的对象来访问对象的成员变量和方法。

```
int a;
```

```
a=(new String("hello")).length();
```

可以看出，直接用创建的对象访问对象的方法是不方便的。我们可以用类名来声明一个该类类型的变量，使用该变量来访问创建的对象。例如：

```
String stringvariable;
```

```
Stringcariable=new String("hello"); /*使变量引用实例对象称为对
```

变量的实例化*/

```
int a;
```

```
a=stringvariable.length();
```

一个对象可以同时被多个变量引用，这些变量指向同一个实例对象。使用任何一个变量都可以操作该实例对象。下面我们介绍创建对象以及对对象进行引用的常用格式：

第一种：

```
classname objectname;
```

```
objectname=new classname();
```

`classname`：是已经定义的类的类名；

`objectname`：声明的对象名，通过对象名来使用对象。

第二种：

```
classname objectname=new classname();
```

第三种：

```
classname objectnaem1;
```

```
classname objectnaem2;
```

```
objectname1=new classname();
```

```
objectname2=objectname1;
```

5.3.2 对象的使用

创建对象的目的就是能够使用对象，一个实例对象具有它的属性和方法。属性就是它的成员数据，通过对成员数据的访问，我们就知道对象的属性如它的状态，位置等信息；通过对对象方法的访问，我们就能让对象完成一定的功能，这正是对象设计的目的所在。在面向对象的程序设计中，程序对对象的访问是通过向对象发送消息，Java 对象之间的通信也是通过消息机制。例如，有如下类：

```
class compute
{
    int speed=133;
    int crt=14;

    void PrintInformation()
    {
        System.out.println("speed is"+speed);
        System.out.println("crt's size is"+crt);
    }
}
```

我们可以用这个类来创建它的实例对象，如下：`computer computer1=new computer();` 这样我们创建了 `computer` 的一个实例。通过向对象发送消息，对象接收到消息就会执行相应的动作。我们想要 `computer1` 对象将它的信息打印出来，我们便向它发送一条打印的消息，表示如下：`computer1.PrintInformation();` 当对象接收到消息，分析这条消息是要它调用 `PrintInformation()` 方法执行打印动作，然后执行该动作，并将对象的信息打印出来。向对象发送消息的一般语法格式为：`object.objectmember;` `object`：是创建的实例对象的名字，以便确定是向谁发送消息；`objectmember`：发送的消息内容，以便让对象知道是执行什么样的操作。`objectmember` 可以是成员方法名，也可以是成员的变量名。

5.3.3 对象的释放

Java 采用垃圾自动回收机制，它不需要程序员去设计存储管理器。若没有自动垃圾回收机制，程序员必须要自己为对象分配内存，对对象进行跟踪和标志对象，以便将无用的对象释放。当创建的对象较多时，这样的工作是繁琐而复杂的。Java 的自动垃圾回收能很好地委托程序员管理内存，当对象在使用时，在对象上加上标志；若一个对象长久没有作用，将失去标志，则自动把它当作垃圾回收，其所占的存储空间供其他对象使用。Java 的自动垃圾回收机制使程序能安全稳定地运行。

5.4 方法

定义一个方法，包括方法名部分的说明和方法体部分的说明，这两部分组合起来就可以定义一个方法。我们仍然以方法的两个组成部分来介绍方法的定义。首先介绍方法名部分的说明。

1. 方法名部分说明

```
[modifiers] returntype methodname([type variable[,type variable2[,...]]])  
{  
.....  
}
```

modifiers: 方法修饰符，可以有以下几种：

- 权限修饰符：**public**, **private**, **protected**
- 最终方法修饰符：**final**;
- 静态方法（类方法）修饰符：**static**。

returntype: 方法的返回值类型，为前面介绍的简单类型和复合类型。

methodname: 方法的名字，为合法的标识符；

(): 其内的内容为方法的参数列表，方法可以有参数也可以没有参数，参数都有相应的类型。参数可以是一个也可以是多个，多个参数之间以逗号“,”隔开。

type variable: 是方法参数说明。**type** 是参数的类型，**variable** 是参数名，其它的参数定义一样。下面是定义一个方法的例子：

```
class sample  
{  
  
    int calculation(int a,int b)  
    {  
        int c;  
        c=a*b;  
        return c;  
    }  
  
}
```

定义的方法名为 **calculation**，该函数有两个参数，参数的类型都为整型 **int**，返回值为整型 **int**。

2. 方法体部分的说明

方法体由{}括起来，紧跟在方法名的后面，是方法行为发生的场所。方法体通常由变量的声明语句、要在方法体中执行的语句以及用来调用其它方法的语句组成。下面是对上面例子中的方法体的解释。

int c;

是方法体内的局部变量的声明，临时分配内存来存储表达式 **a*b** 的值；

c=a*b;

是方法体中的执行语句，执行语句还可以是对方法的调用；

return c :

将 **c** 的值返回。

在方法体中，可以没有局部变量，也可以有，局部变量可以是多个。**Java** 是动态地分配内存，局部变量的说明可以在执行语句之前事先说明，也可以在需要局部变量的时候说明，需要的时候分配内存。例如：

```

class test
{
int cal (int a, int b, int c)
{
int d;
c=a*b;
int e;
e=c-b;
return e;} }

```

局部变量 **e** 在需要时才加以说明，然后为其分配临时内存。局部变量的类型为 **Java** 的数据类型，包括简单数据类型和复合数据类型。当方法的返回值为 **void** 类型时，没有 **reurn** 语句，否则出错；若不是 **void** 类型，则必须有 **return** 语句指明返回值。返回值的类型应与说明的类型匹配或者符合由小到大的转换规则。在方法体中说明的变量为局部变量，它的生命周期很短。局部变量仅在方法的运行时有效，方法返回后则将局部变量分配的内存收回；局部变量不再存在。例如，下面在方法 **functionname()** 的方法体中定义的变量 **bb** 为

```

局部变量： class classname { void functionname ( ) { int bb; }
}

```

方法的权限和变量的访问权限的说明一样，这里不再详细介绍。

习 题

- 1.创建对象的格式为：_____.
- 2.类的定义包括____和_____.
- 3.定义类的权限是可以用权限修饰符来说明。类的权限修饰符有那三种？
- 4.当没有权限说明，成员变量的权限隐含为____,该成员变量____被与他在同一个包中的其他类访问。
- 5.调用方法的一般格式为：_____.
- 6.java 中有一个特殊的引用值____,它可以代表任何方法中的当前对象。
- 7.包的定义？
- 8.抽象类指的是包含_____的类，从抽象类派生出的类必须实现所有的_____.
- 9.方法的修饰符有权限修饰符：____、____、____;最终修饰符____;静态方法修饰符：_____.
- 10.类是____的模板，定义了对象的_____和_____.

第六讲:Java 的例外处理

6.1 例外处理机制

程序在运行过程中不可避免要产生一些例外，例如出现内存不够，文件找不到，除数为 0 等情况。这里的例外，与程序出现的普通的逻辑错误、编译错误是不同的。例外是不可预料的，而错误是必然的。一般高级语言都有例外处理机制，当产生例外时对例外进行处理，然后再返回产生例外的语句继续执行。**Java** 提供例外处理机制，以保证程序顺利地运行。一些语句执行时可能产生例外，必须进行例外处理，否则编译将不能够通

过。下面我们来看一个例外处理的例子。

```
class mainclass
{
    public void main(String args[])
    {
        try
        {
            System.in.read();
        }
        catch(IOException e)
        {
            System.out.println(e.toString());
        }
    }
}
```

上面的例外处理语句是 **try-catch** 语句。**try-catch** 必须配套使用，一个 **try** 可以有多个 **catch()** 子句。其中 **try** 关键字

下面的 **{}** 为可能产生例外的语句，称为 **try** 语句组，**catch()** 为捕获例外子句，**catch** 子句的括号 **()** 的 **IOException e** 为

捕获的例外类型，相当于方法的参数。一个 **catch()** 子句可以有多个，每一个 **catch()** 子句都对应有一个例外处理语

句组。当某一个 **catch()** 子句捕获到例外，该例外将作为 **catch()** 的参数，传入 **catch()** 语句，我们可以通过该例外对

象获得例外信息，将信息输出。**catch()** 捕获例外相当于调用方法时传入参数，然后执行 **catch()** 的例外处理语句组。

在该例中，调用 **System.in.read()** 方法可能产生一个 **IOException** 例外，该例外必须被声明捕获，否则编译将无法过。

因此将它放入 **try** 语句组，在 **catch()** 语句组中对例外进行处理。下面我们介绍例外处理机制，以便进行例外处理。

例外也是一个实例对象。当程序执行到某一条语句时，由于某些突发的异常（如文件被破坏，系统无法找到

指定的文件等等），导致系统不知如何继续进行而抛出一个例外对象。该例外对象被抛出后，首先在产生例外

语句所在的方法内寻找例外处理语句。如果该方法中没有声明对该语句产生的例外进行处理，或者没有找到与

该例外对象类型相匹配的例外处理语句 **catch()** 语句，则该例外抛向上一级调用栈（即抛向调用产生例外的方法

的方法）。如果上一级调用栈声明对该产生例外的方法进行例外处理，且有相匹配的例外处理语句，于是处理

该语句，继续程序的执行；如果还没有例外处理，则继续抛向上一级，最后抛向运行平台。

例外有如图 6—1 所示的调用关系。

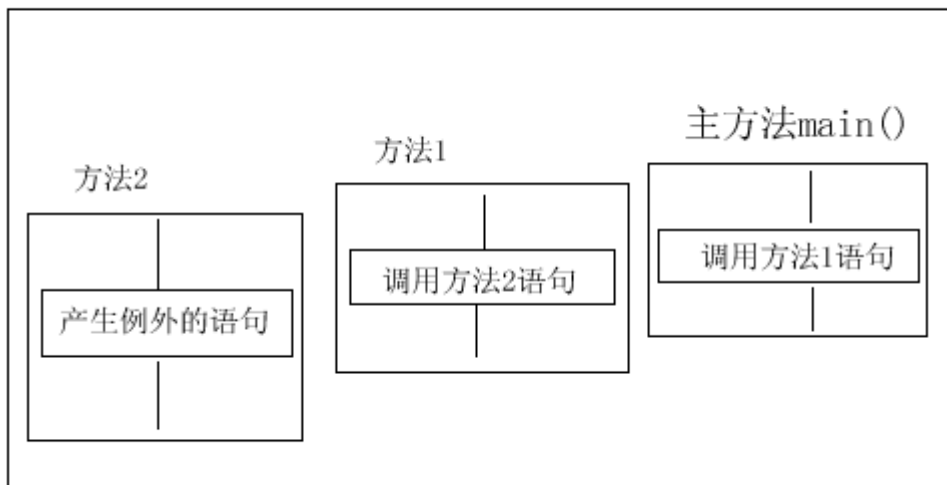


图 6-1 方法调用关系

图 6-1 中，主方法 `main()` 的一条语句是调用方法 1，方法 1 的一条语句调用方法 2。执行方法 2 的一条语句时产生例外，停止执行下一条语句，系统抛出一个例外对象。在该方法中，如果对该条语句声明进行例外处理用放入 `try` 语句组中则产生例外时，便到该 `try` 语句配套的 `catch()` 子句中寻找。如果 `catch()` 子句中的例外类型与产生该例外的类型一样或者为该例外的类型的父类时，则该 `catch()` 子句捕获该例外并处理之。继续执行方法 2 中 `try-catch()` 下面的语句。如果方法 2 中没有进行例外处理，则该例外对象抛向方法 1，相当于在方法 1 中，执行调用方法 2 的语句时产生例外。如果在方法 1 中声明了进行例外处理并处理了例外，则执行方法 2 中的处理该例外的 `try-catch()` 语句的下一条语句；如果方法 1 没有对该例外进行处理，则将该例外继续向上一级调用栈 `main()` 方法抛出。在 `main()` 方法中，相当于在执行调用方法 1 的语句时产生一个例外。如果在 `main` 方法中，该调用语句声明了进行例外处理，则寻找对应的 `catch()` 子句所能捕获的例外类型。如果 `catch()` 的例外类型与方法 2 抛出的例外一样或者为它的父类，就对抛出的例外进行处理。处理后执行处理例外语句 `try-catch` 的下一条语句，否则该例外继续向上一级调用栈抛出，交给 Java 解释器处理，Java 解释器处理完该例外，程序便终止。

例外都是例外类的实例对象，要捕获例外对象，需要知道抛出的例外对象是属于哪一个例外类。可以在程序中用程序语句创建一个例外类的实例对象，还可以通过 `throw` 语句抛出这个例外对象。为此我们要知道例外类以及例外类的层次关系，才能够正确地创建和捕获例外对象。

6.2 例外类的层次

系统抛出的例外是例外类的对象，同时我们还可以定义自己的例外类来扩充例外的种类，处理更特殊情况。

下面我们介绍例外类的继承关系。

所有例外类都是 **Throwable** 类的子类或者其子类的子类，例外类有两大类：**Error** 类和 **Exception** 类，其中 **Error** 是 **Throwable** 类的直接子类。

Error 类的例外包括动态链接库链接失败等异常，Java 程序一般不必抛出这种类型的例外，也不必捕获该类型的例外。**Exception** 类的例外是我们在编写 Java 程序需要处理的例外。

Exception 类的例外有两种类型：

RuntimeException 例外类型（运行时例外）及除 **RuntimeException** 类型外的其它例外类型（非运行时例外）。考虑处理所有的这些例外会使程序的可读性差，其中的 **RuntimeException** 类型的例外程序可以不声明捕获处理。

例如，**IndexOutOfBoundsException** 例外是 **RuntimeException** 的子类，对产生该类型的例外程序语句可以不做声明，编译能够通过。**Exception** 类除了 **RuntimeException** 子类，对其它于类类型的例外，程序必须声明对该例外进行处理。

例如，我们调用 **System.read()** 方法可能会产生 **IOException** 类的例外，为此，我们必须声明。

对该例外用 **try-catch** 语句进行处理，或者调用 **system.read()** 方法的方法声明将例外抛出。下面介绍例外处理。

6.3 例外处理语句及使用

我们知道例外类的层次，知道哪些例外必须处理，哪些可以不处理后，就可以视具体的情况通过例外处理语句来进行例外处理。例外处理语句有如下两种。

6.3.1 try-catch 语句

语法格式为：

```
try
{
...
}
catch (exception1 e)
{
...
}
catch (exception2 e)
{
...
}
catch (exception3 e)
{
...
```

```
...
}
statement;
...
```

(1) `try {...}` 捕获例外的范围，将要产生例外的语句放入 `try` 的`{}`中；

(2) `catch()`语句可以有多个；`exceptionN e` 为 `catch ()` 的参数，当 `try` 中的语句产生例外对象时，由上至下寻找，调用首先找到的与例外对象类型一样或者为该例外对象的父类的 `catch()`子句进行例外处理。例外对象就作为。`catch()`的参数，执行完该 `catch()`子句，跳过其他的 `catch()`子句，同时跳出 `try{}语句组`，执行 `try—catch ()` 语句下面的一条语句：`statement`。

从 Java 的例外捕获过程可以看出，父类可以捕获子类的例外，当我们不知道抛出的是什么类型的例外或者不关心抛出的例外类型，而只想捕获该例外时，可以将例外的一般类型 `Exception` 类型作为 `catch ()` 子句的参数，用该 `catch` 子句来捕获所有的例外，如：

`catch(Exception e)`。

通常我们需要知道产生例外的具体情况如 `IOException` 类的例外信息，需要用更具体的子类来作为 `catch()`的参数，以便捕获时获得更具体的信息，为处理错误提供很好的帮助。

6.3.2 try—catch-finally

`try-catch` 语句后面还可以跟 `finally` 语句组，构成 `try—catch—finally` 例外处理语句，其语法格式为：

```
try
{
...
}
catch (exception1 e)
{
...
}
catch (exception2 e)
{
...
}
catch (exception3 e)
{
...
}
finally
{
...
}
statement;
...
```

不论 `try` 语句组是否产生例外，`finally{}语句组`都会被执行，这样为例外处理提供一个统一的出口。`try-catch` 部分的工作原理同上。

6.3.3 throws 语句

throws 语句用在方法定义时声明该方法要抛出的例外类型，如果抛出的是 **Exception** 例外类型，则该方法被声明为抛出所有的例外。**throws** 语句的语法格式为：

```
methodname throws Exception1,Exception2,...,ExceptionN
{
...
}
```

方法名后的 **throws Exception1,Exception2,...,ExceptionN** 为声明要抛出的例外列表。当方法由该例外列表时，方法将不对这些类型及其子类类型的例外作处理，而抛向调用该方法的方法，由他去处理。**{...}**为方法体。

6.3.4 throw 语句

我们知道，例外是例外类的实例对象，我们可以创建例外类的实例对象通过 **throw** 语句抛出。该语句的语法格式为：

```
throw new exceptionname;
```

例如抛出一个 **IOException** 类的例外对象：

```
throw new IOException;
```

要注意的是，**throw** 抛出的只能够是可抛出类 **Throwable** 或者其子类的实例对象。下面的操作是错误的：

```
throw new String("exception");
```

这是因为 **String** 不是 **Throwable** 类的子类。

6.4 例外处理举例

每一个例外类都有相应的操作方法，通过这些方法来获得系统提供的错误信息，通常的方法是通过 **System.out.println()**方法将例外对象打印出来。所有例外类的子类中只定义了构造方法，没有定义其他的操作方法。操作方法定义在所有例外类的父类即可抛出类 **Throwable** 中，所有子类（包括子类的子类）通过继承 而拥有这些方法，例如 **Exception** 类。下面介绍 **Throwable** 类中定义的这些方法。

6.4.1 Throwable 类

所有 **Throwable** 的子类都继承了该类的方法。

含义	方法调用格式
构造方法	public Throwable()
	public Throwable(String message)
可以得到一个例外可以重新抛出一个例外	public Throwable fillInStackTrace
获得例外信息	public String getMessage()
将例外信息打印到指定的 PrintStream 流中 (如文件中或屏幕上)	public void printStackTrace(PrintStream s)
得到例外对象的描述	public String toString()
在屏幕上打印例外信息	public void printStackTrace()

除了系统为我们提供的例外类外，我们可以根据需要定义自己的例外处理类。例外处理类必须是继承 **Throwable** 类的子类或者是继承其子类的子类。下面介绍自定义例外处理类。

6.4.2 自定义例外类

例外类与其他类所不同的是例外类必须是继承可抛出类 `Throwable`。`Throwable` 是所有例外类的最高父类。因此要定义一个例外类，必须继承可抛出类 `Throwable` 或者其子类。只有例外类的实例对象才可以用 `throw` 语句抛出。通常，Java 的 `Exception` 的子类已经很具体，例如 `FileNotFoundException` 例外类，可以很具体的提供错误信息，没有必要再定义一个继承 `FileNotFoundException` 的子类。通常我们定义一个更一般的例外类型 `Exception` 子类，下面我们举例介绍自定义例外类。

```
class myexception extends Exception
{
public myexception(String str)
{
super(str)
}
public void putinfo()
{
System.out.println("this exception class designed by wsq,5.20.2002");
}
}
class mainclass
{
public static void throwexception() throws myexception
{
throw new myexception("sample of myexception");
}
public static void main(String args[])
{
try
{
throwexception();
}
catch(myexception e)
{
e.putinfo();
System.out.println("exception information:");
System.out.println(e);
}
}
}
```

程序运行结果为：

```
D:\javaprogram\exception>jview mainclass
this exception class designed by wsq,5.20.2002
exception information:
myexception: sample of myexception
```

第六讲:Java 的例外处理

习 题

- 1.例外处理语句有哪些?
- 2.throw 语句的语法格式是什么?
- 3.throw 抛出的必须是可抛出类 Throwable 或者是其子类的实例对象么?
- 4.例外类与其他类不同的是什么?
- 5.java 通过五个关键字来管理异常处理: _____, _____, _____, _____, _____.
6. 在 java 中, ____ 语句用来监视异常; ____ 语句用来捕获异常; ____ 用来在出现错误时产生异常。
- 7.异常的定义是什么?
- 8.例外类有两大类: _____ 和 _____.
- 9.throw 抛出的只能是 _____ 或者 _____.
- 10.例外类都是 _____ 类的子类或者其子类的子类。

第七讲:Java 的接口与包

7.1 接口

我们知道,所有的类都直接或者间接继承 Object 类,该继承关系是单一的。这种继承关系构成一棵倒挂的树, Object 类位于树根,其他类在这棵树上都有一个相应的位置。接口和抽象类相似,但接口不是类,独立与类的继承关系之外,是另外一种数据类型。接口中定义的都是抽象方法,需要提供一个类,在该类中实现所有的借口中的抽象方法。

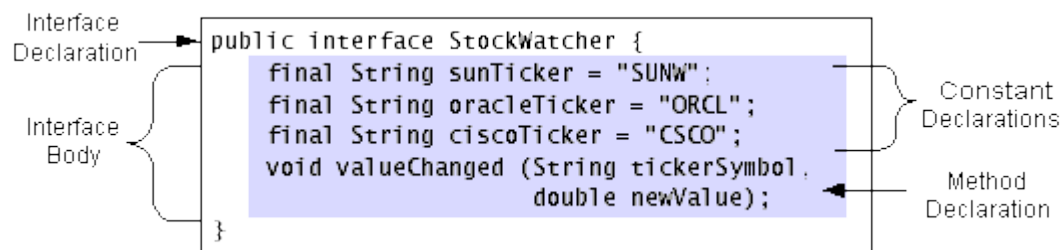
7.1.1 接口的定义

定义一个接口跟创建一个新类是相似的。接口定义需要两个组件:接口定义和接口实体。

```
interfaceDeclaration {  
    interfaceBody  
}
```

interfaceDeclaration 声明了各种关于接口的属性,比如它的名字和是否扩展其它的接口。这个 interfaceBody 包含了在接口中常量和方法声明。

如下图所示给出了接口定义有两个组件:接口声明和接口实体。接口声明定义了各种关于接口的属性,比如它的名字和是否扩展其它的属性;接口实体包含了常数和用于接口的方法声明。



StockWatcher 接口和接口定义的结构为:

```
public interface StockWatcher {  
    final String  
    sunTicker = "SUNW";  
    final String oracleTicker = "ORCL";  
    final String ciscoTicker = "CSCO";  
}
```

```
void valueChanged(String tickerSymbol, double newValue);
}
```

接口定义三个常量，它们是 **watchable** 股票的股票行情自动收集器的符号。这个接口也定义了 **valueChanged** 方法，但是没有执行它。执行这个接口的类为方法提供了执行。

下面讲讲接口的声明：

如图给出了接口声明的所有可能组件：

public	Makes this interface public.
interface <i>InterfaceName</i>	This is the name of the interface.
Extends <i>SuperInterfaces</i>	This interface's superinterfaces.
<pre>{ <i>InterfaceBody</i> }</pre>	

在接口定义中需要两个元素：**interface** 关键字和接口的名字。**Public** 指示了接口可以在任何的包中任何的类中使用。如果你没有指定接口为 **public**，那么接口就只能在定义接口的包中类使用了。

接口定义可以有另外一个组件：**superinterfaces** 系列。一个接口可以扩展另外的接口，这跟类可以扩展一样。但是，类只能扩展一个另外的类，而接口可以扩展任意个接口。**Superinterfaces** 系列以逗号分隔的所有接口，这些接口可以由新的接口扩展。

这时候，你也许会问：那接口实体怎么说呢？别急，下面就会解释了：接口实体为所有包含在接口中的方法包含了方法声明。在接口中的方法声明可以紧跟着一个逗号，因为接口不为定义在它上面的方法提供执行。所有定义在接口中的方法可以隐含地为 **public** 和 **abstract**。

接口可以包含常量 **s** 声明以及方法声明。所有定义在接口中的常量可以是 **public**、**static** 和 **final**。定义在接口中的成员声明不允许使用一些声明修饰语，比如你不能在接口中的成员声明中使用 **transient**、**volatile** 或者 **synchronized**。同样你不能在声明接口的成员的时候使用 **private** 和 **protected** 修饰语。

7.1.2 接口的执行

为了使用接口，你要编写执行接口的类。如果一个类可以执行一个接口，那么这个类就提供了执行定义在接口中的所有方法的方法。

一个接口定义了行为的协议。一个类可以根据定义在接口中的协议来执行接口。为了声明一个类执行一个接口，要包括一条执行语句在类的声明中。你的类可以执行多个接口（因为 **JAVA** 平台支持接口的多个继承），因此可以在 **implements** 后面列出由类执行的接口系列，这些接口是以逗号分隔的。

以下是一个 **applet** 的部分例子，它执行 **StockWatcher** 接口：

```
public class StockApplet extends Applet implements StockWatcher {
...

public void valueChanged(String tickerSymbol, double newValue) {
if (tickerSymbol.equals(sunTicker)) {
...
} else if (tickerSymbol.equals(oracleTicker)) {
...
} else if (tickerSymbol.equals(ciscoTicker)) {
```

```
...
}
}
}
```

这里注意，这个类引用了定义在 `StockWatcher.sunTicker` 的常量，如 `oracleTicker` 等等。执行接口的类继承了定义在接口中的常量。因此这些类可以使用简单的名字来引用常量。你可以象下面的语句，使其它任何类使用接口常量：

`StockWatcher.sunTicker`

从本质上讲，当类执行一个接口的时候，就签定了一个契约。所有的类必须执行所有定义在接口以及它的 `superinterfaces` 中的方法，以及类必须定义为 `abstract`。这个方法签名（名字和在类中参数类型的数目）必须匹配方法的签名。`StockApplet` 执行 `StockWatcher` 接口，因此 `applet` 提供了 `valueChanged` 方法。这个方法公开地更新了 `applets` 的显示或者使用这个信息。

7.2 Java 的类包

前面我们介绍了类的操作，知道了关于类的语法现象。我们可以自定义实现一定功能的类，同时 `Java` 为我们提供了有丰富功能的类。这些类都组织在 `Java` 的类包中，我们称这些包为系统工具包。`Java` 系统提供的系统工具包为：

`java.lang`, `java.io`, `java.util`, `java.net`, `java.awt`, `java.awt.peer`, `java.awt.image`, `java.applet`.

而这些包又提供了功能强大的类和接口，这里我们就不一一介绍。

在 `Java` 程序设计中，正确熟练地使用类，首先要知道类的位置，他在什么目录下，是属于哪一个包。更重要的一点是，要知道类的定义（即该类对使用者提供的接口）。我们只有知道类的定义，才能创建类的实例对象，才能知道对象的属性，才能正确地调用对象的方法；同时，我们还需知道类的继承关系，即类在继承树中的位置。我们知道继承能够实现代码的复用，在继承树中，子类继承父类。子类除了继承父类中自定义的可以继承的方法外，还继承它的父类所继承的方法，这样子类就拥有父类中继承来的自定义的方法，同时还拥有它的父类的可以继承的方法。可见，清楚地知道类的继承关系，能更充分地利用类的功能。下按我们举例说明：

```
class superfather
{
    public void method1()
    {
        ...
    }
}
class father extends superfather
{
    public void method2()
    {
        ...
    }
}
```

```

}
class sub extends father
{
...
}

```

子类 **sub** 继承了父类 **father**,父类 **father** 继承了子类 **sub** 的祖父类 **superfather**(即父类 **father** 的父类 **superfather**).相对于子类来说,在祖父类(**superfather**)中定义了方法 **method1()**,父类 (**father**)继承祖父类(**superfather**)的方法 **method1**,拥有该方法;在父类(**father**)中,又定义了方法 **method2()**,这样父类拥有了方法 **method1** 和方法 **method2()**,子类(**sub**)继承父类(**father**),子类(**sub**)继承了父类(**father**)的方法 **method1()**和方法 **method2()**,子类也就拥有了他们。使用子类继承而来的方法就像使用子类中自定义的方法一样。**Java** 的父类为我们定义了许多普遍的基本方法。从上面的继承功能可见,这些方法对下面的子类/子类的子类(**supersub**)都是非常有用的。

7.3 最高类 **Object** 和系统类 **System**

7.3.1 **Object**

Object 类属于 **java.lang** 包。在 **Java** 中,所有的类都是直接或者间接的继承 **Object** 类而来的,也就是除了 **Object** 雷,所有的类都是继承父类而来的,**Object** 没有父类。因此,**Java** 所有的类的继承关系就构成了一棵倒立的树,**Object** 处于树根的位置,其他的类在继承树上也有相应的位置。

7.3.2 **System**

System 类属于 **java.lang** 包。**System** 类是直接继承最高类 **Object**.系统类定义的成员变量和成员方法都是静态的,可以用类名直接调用类的成员变量和成员方法,静态的成员相当于全局量。功能有:进行标准输入输出,获取系统当前时间,获得系统的属性,程序的强制终止,装载动态链接库,设置系统的属性等。系统类定义三个流:

下面介绍系统类的应用。

一、标准输入输出功能

* 输入

用基本输入流 **in** 进行输入操作,输入流 **in** 的输入方法有:

int read(): 从键盘只读入一个字符,得到的是字符的 **ASCII** 码;

int read(byte b[]): 从键盘读入字符,存于 **b[]**数组中;

int read(byte b[],int off,int len): 从键盘输入的字符序列中,读入指定起始字符和指定个数的字符,存在数组 **b[]**中,**off** 是要读入的字符的起始位置,**len** 是要读入的字符的个数。

上面的输入方法艘是读入单个字符,得到字符的 **ASCII** 码,不能用他们来读入一个整数。

*输出用输出流 **out** 进行输出操作,

对上面的方法的说明:

- 1 使用 **write()**方法输出的是一个字节,并将字节所代表的 **ASCII** 码在标准输出设备上(一般为屏幕)显示。
- 2 使用方法 **print()**和方法 **println()**都是输出表示变量值的字符串,在标准输出设备上显示的

是变量的值。

3 有关标准输入输出的其他方面可以参见标准流。

二：系统属性操作功能

用系统属性来定义当前运行环境的特性，运行时系统被初始化，运行环境和系统属性也被初始化。

1.系统属性的获取

我们可以通过系统类 `System` 下面的方法来得到系统的属性：

```
public static Properties getProperties()  
public static String getProperty(String key)  
public static String getProperty(String key,String def)
```

Java 将系统属性封装在属性类 `Properties` 中，属性类 `Properties` 属于 `java.util` 包。我们可以通过系统类 `System` 的方法 `getProperties()` 可以得到当前系统的属性对象，通过属性对象的方法 `list()` 可以将属性显示在屏幕上。属性类 `Properties` 定义了下面的方法操作属性对象，见下表：

2.系统属性的设置

设置系统属性的方法为：

```
public static void setProperties(Properties props)
```

该方法的参数为一个属性对象，将属性对象中的所有属性项的属性值设置为系统的属性，系统原来的属性值将被取代。因此，作为参数的属性对象应该事先装载要设置的属性值。要为属性对象装载属性有两种方式：

`public Properties(Properties defaults)`:通过构造方法在创建属性对象的时候为属性对象初始化；

`public void load(InputStream in)`:通过属性对象的方法在属性对象创建后装载，该方法将标准输入流中的属性装载于属性对象。

3.获取系统时间

用系统类的方法 `currentTimeMillis()` 可以得到自 1970 年 1 月 1 日到系统的当前时刻之间的时间，时间的单位为毫秒。在操作开始和结束时分别记下系统时间，这样可以知道该系统操作所需的时间。

4.数组的拷贝功能

使用系统类的方法：

```
public static void arraycopy(Object src,int src_position,Object dst,int dst_position,int length)
```

参数 `Object src` 为源数组，参数 `int src_position` 为源数组中的起始元素，参数 `Object dst` 为目标数组，参数 `int dst_position` 为目标数组中的起始元素，参数 `int length` 为要拷贝的元素的个数。

5.安全管理

Java 是一门网络编程语言，提供安全管理器来进行程序的操作管理。安全管理器是一个 `SecurityManager` 对象在系统运行期起作用，任何对安全危险的操作都必须得到 `SecurityManager` 对象的许可，得不到认可的操作将产生一个 `SecurityException` 的对象。

7.4 其他类

7.4.1 字符串处理类 String 和 StringBuffer

字符串类 **String** 和 **StringBuffer** 都是对字符串进行处理的工具。**String** 处理的是不可变的字符串，即 **String** 的实例对象创建后就不可变，不能在 **String** 的实例中插入新的字符，不能替换字符串中的字符，不能在字符串后面追加字符或者字符串等改变 **String** 对象的操作。**StringBuffer** 可以处理可变的字符串，可以替换字符等。

7.4.2 Math 类

Math 类为我们提供了许多数学运算方法，这些方法都是静态的，用类名 **Math** 访问。

7.4.3 简单数据类型封装类

Java 语言对一些简单数据类型做了封装，程序员可以直接使用这些数据类型，不必从头去定义。下面简单介绍一下。有整型数据的封装 **Integer**，长整型数据封装类 **Long**，浮点型数据的封装类 **Float**、**Double** 类，**Boolean** 类，**Character** 类。

第七讲:Java 的接口与包

习 题

- 1.java 系统提供的系统工具包有哪些？
- 2.protected void finalize()的含义是什么？
- 3.系统类定义的成员变量和成员方法都是静态的还是动态的？
- 4.基本输入流 **in** 的输入方法有那三种？
- 5.设置系统属性的方法为：_____。
- 6.我们知道为属性对象装载属性有两种方式：第一种是：**public Properties(Properties props);** 第二种是：_____。
- 7.设计 Java 接口的目的是什么？
- 8.接口的定义由_____和_____组成。
- 9.Java 的类库中包含了一个非正式的并可在其他 Java 核心包中使用的工具类集合，这些类保存在_____和_____。
- 10.字符串类 **String** 和 **StringBuffer** 的区别和联系？

第八讲:输入输出操作

8.1 Java 输入和输出

可将 Java 库的 I/O 类分割为输入与输出两个部分，这一点在用 Web 浏览器阅读联机 Java 类文档时便可知道。通过继承，从 **InputStream**（输入流）衍生的所有类都拥有名为 **read()** 的基本方法，用于读取单个字节或者字节数组。类似地，从 **OutputStream** 衍生的所有类都拥有基本方法 **write()**，用于写入单个字节或者字节数组。然而，我们通常不会用到这些方法；它们之所以存在，是因为更复杂的类可以利用它们，以便提供一个更有用的接口。因此，我们很少用单个类创建自己的系统对象。一般情况下，我们都是将多个对象重叠在一起，提供

自己期望的功能。我们之所以感到 Java 的流库（Stream Library）异常复杂，正是由于为了创建单独一个结果流，却需要创建多个对象的缘故。

很有必要按照功能对类进行分类。库的设计者首先决定与输入有关的所有类都从 `InputStream` 继承，而与输出有关的所有类都从 `OutputStream` 继承。

8.1.1 `InputStream` 流

`InputStream` 的作用是标志那些从不同起源地产生输入的类。这些起源地包括（每个都有一个相关的 `InputStream` 子类）：

- (1) 字节数组
- (2) `String` 对象
- (3) 文件
- (4) “管道”，它的工作原理与现实生活中的管道类似：将一些东西置入一端，它们在另一端出来。
- (5) 一系列其他流，以便我们将其统一收集到单独一个流内。
- (6) 其他起源地，如 Internet 连接等（将在本书后面的部分讲述）。

除此以外，`FilterInputStream` 也属于 `InputStream` 的一种类型，用它可为“破坏器”类提供一个基础类，以便将属性或者有用的接口同输入流连接到一起。这将在以后讨论。

`InputStream` 流是一个抽象类，在该类中实现了基本的输入方法这些方法见下表：

表 8-1 <code>InputStream</code> 流的方法	
含 义	方法的调用格式
构造方法，子类调用	<code>public InputStream()</code>
得到可以从流中读取的字节数	<code>public int available()</code>
关闭流	<code>public void close()</code>
在输入流的当前读取位置标记流，从该流的位置读取 <code>readlimit</code> 所指定的字节数后该标记失效	<code>public void mark(int readlimit)</code>
判断流是否支持 <code>mark()</code> 方法	<code>public boolean markSupported()</code>
一个抽象的方法，需要子类实现	<code>public abstract int read()</code>
从输入流中读入 <code>b.length</code> 个字节的数据写入数组 <code>b</code> 中，方法返回读取的字节数	<code>public int read(byte b[])</code>
从输入流的当前位置读取 <code>len</code> 指定长度的数据，写入数组 <code>b</code> 中 <code>off</code> 下标开始的位置，返回读取的字节数	<code>public int read(byte b[], int off, int len)</code>
重置流的位置为 <code>mark()</code> 方法标记的位置	<code>public void reset()</code>
从流的当前位置跳过 <code>n</code> 指定的字节数，返回跳过的字节数	<code>public long skip(long n)</code>

8.1.2 `OutputStream` 流

这一类别包括的类决定了我们的输入往何处去：一个字节数组（但没有 `String`；假定我们可用字节数组创建一个）；一个文件；或者一个“管道”。除此以外，`FilterOutputStream` 为“破坏器”类提供了一个基础类，它将属性或者有用的接口同输出流连接起来。这将在以后讨论。`OutputStream` 流是一个抽象类，该类中实现了输出操作的基本方法，其子类的流继承该父类中的这些方法，同时子类需要扩充 `OutputStream` 流中的方法，来实现更特殊的功能。

8.3 内存读写

I/O 流的子类 `ByteArrayInputStream` 流、`ByteArrayOutputStream` 流和 `StringBufferInputStream` 流和 `StringBufferOutputStream` 流可以对内存进行输入输出操作。

`ByteArrayInputStream` 流是从数组中读入字节。该流的构造方法为：

`public ByteArrayInputStream(byte bf[])`: 用指定的数组初始化输入流；
`public ByteArrayInputStream(byte buf[],int offset,int length)` : 用数组中从指定下标开始 len 长度的数据初始化输入流。

`ByteArrayStream` 流重写了 `InputStream` 流的 `read()`,`available()`,`reset()`,`skip()` 方法来实现特殊的功能。

`ByteArrayInputStream` 流的 `reset()` 方法是将流的当前位置置于输入流的起始位置，即字节数组的起始位置。

`ByteArrayOutputStream` 流是向数组中写入字节数据。通过该流的构造方法构造一个输出流，该流有一个缓冲区，当我们用构造方法；

`public ByteArrayOutputStream(int size)` 构造流时，用 size 指定流缓冲区的大小，初始值是 32 字节，缓冲区的长度可以随数据的写入自动增加。用 `public int size()` 方法可以得到输出流中的有效字节数。用方法 `public void write(int b)` 来将数据写入输出流中，再用方法 `public void write(byte b[],int off,int len)` 将输出流中的数据写入数组，同时可以通过方法 `public byte[] toByteArray()` 来得到输出流中缓冲区的有效数据，写入一个数组中。`public void writeTo(OutputStream out)` 方法可以将流中的数据输出到参数指定的输出流中。`public void reset()` 方法清除缓冲区中的内容，以便该缓冲区能够重用。

`StringBufferInputStream` 流，`StringBufferOutputStream` 流的操作跟 `ByteArrayInputStream` 流，`ByteArrayOutputStream` 流的操作基本相似，不同的是这两个流是对字符串进行操作的，它读写的是 16 位字符。

8.4 管道流

我们可能接触到 DOS 操作系统提供的管道命令，应对管道有一个具体的了解。管道的作用是将一个程序或者线程的输出作为另一个程序或者线程的输入。Java 流中也引入了管道的概念，提供了两个管道类：`PipedInputStream` 流和 `PipedOutputStream` 流。管道输入流作为数据接收端，输出流作为数据的发送端。管道输入流和输出流必须连接才能实现数据的流动。

管道输入流的构造方法为：

`public PipedInputStream()`: 创建一个没有连接输出流的输入流；
`public PipedInputStream(PipedOutputStream src)`: 构造一个连接参数指定的输出流的输入流。

管道输出流的构造方法为：

`public PipedOutputStream()`: 创建一个没有连接输入流的输出流；
`public PipedOutputStream(PipedInputStream snk)` 创建一个连接由参数指定的输入流的输出流。

`PipedInputStream`，`PipedOutputStream` 流中都提供了连接管道的方法，`PipedInputStream` 提供的方法为：

```
public void connect(PipedInputStream snk);
```

PipedOutputStream 流提供的方法:

```
public void connect(PipedOutputStream src)
```

在管道流中, 为完成管道的输入输出, 分别重写了 read()、write()方法。

第八讲:输入输出操作

习 题

- 1.所有的输入流的最高父类是_____.
- 2.I/O 流的子类_____, _____、_____, _____可以对内存进行输入输出操作。
- 3.管道的作用是什么?
- 4.管道输入流的构造方法是_____,管道输出流的构造方法是_____.
- 5.Java 提供了_____和_____来进行文件的输入输出流。
- 6.FileInputStream 流的构造方法是什么? FileOutputStream 流的构造方法又是怎样?
- 7.Java 提供了随机访问文件的类_____流, 该流的构造方法为:_____或_____.
8. 使用连接流_____可以将多个流顺序地连接成一个流, 它的构造方法为: _____或_____.
- 9._____是 I/O 包中唯一代表磁盘文件的对象。
- 10._____实现了所有格式化输出功能。

第九讲:多线程程序设计

9.1 线程的机制与运行机理

多线程程序是 Java 程序一个很重要的特点。在一个程序中可以同时运行多个相对独立的线程, 系统将 CPU 分成非常小的时间片, 分别分配给每一个线程, 这样这些线程的运行跟并行运行一样, 在一个程序中同时可以处理多个任务, 这是非常有用的, 例如我们 用一个线程来接收输入, 而另一个线程进行其它的数据处理, 这样就大大地提高了程序执行的效率。线程与进程是两个不同的概念, 进程是一个程序的执行序列, 在多任务的操作系统中, 由操作系统调度, 线程是一个进程中的一个子序列, 线程由程序负责管理, 而进程由操作系统负责管理。在多线程的环境下, 线程共用相同的地址空间, 线程之间的通信 是非常方便的, 它们共同构成一个进程, 而进程之间有不同的地址空间。进程和线程都是动态的概念。进程与程序的区别是: 进程是当程序执行时, 操作系统的一个调度, 是动态的, 程序是静态的。当程序执行时, 得到操作系统的调度, 形成操作系统的一个进程, 依照程序的指令序列运行。线程可以由程序负责管理, 我们可以在程序中创建线程, 可以启动线程, 挂起线程, 终止线程, 将线程从内存中清除等等操作。Java 是基于操作系统级的多线程环境之上设计的, Java 解释器也是依靠多线程来处理任务的, 所有类库的设计都考虑到多线程的机制。下面我们来介绍有关 Java 的线程机制。

◎ 线程的机制

Java 的解释器本身也是一个多线程程序，例如，不用对象的垃圾回收是通过一个时时运行着的优先级比较低的线程来完成的

。线程的优先级是用 0—10 的整数表示的，数字越大优先级越高。当一个程序中只运行一个线程时，此时并不能够显现出线程优先

级的作用，这是因为始终只有一个线程在运行。优先级是用于决定多个线程之间的切换顺序和每一个线程所占 CPU 的时间的多少

，通常，优先级高的线程得到 CPU 的次数多一些，总共占 CPU 的时间长一些，该线程处理的任务效率高一些。一个线程也可以自愿

放弃对 CPU 的控制而把它让给其它的线程。

当多个线程在等待获得 CPU 时间时，优先级高的线程优先抢占到 CPU 时间，同一优先级的线程按照队列获得 CPU 时间。

一个程序可以有多个线程，线程之间可以共享数据。当线程以异步方式访问共享数据时，这是不安全的。例如当线程处于异步工作

方式时，一个线程读数据，一个线程处理数据，若线程还没有读完数据另一个线程就去处理数据，这样必然会得到错误的结果。若

我们将这两个线程同步，等到第一个线程将数据读完后第二个线程才能够处理数据，这样就不会产生异步那样的错误。Java 用关

键字 `synchronized` 来同步对共享数据操作的方法，在一个对象中，用 `synchronized` 声明的方法统称为同步方法。Java 提供了一个同

步模型—监视器。每一个对象都有一个监视器，负责管理线程对对象的同步方法的访问，对象有唯一的一把钥匙，多个线程同时进

入对象，只有取得该对象钥匙的线程才可以访问同步方法，其它的线程在该对象中等待，直到该线程用 `wait()` 放弃该对象的唯

一的这把钥匙，让其它的线程可以取得该钥匙，从而得到访问同步方法。这就是监视器的管理原理。监视器的管理原理也可以叙述

为：当多个线程进入了对象，想要访问该对象的同步方法时，同一个时刻只能有一个线程取得访问对象声明同步的方法，其它线

程在该对象中等待，直到该线程用 `wait()` 方法来放弃对象的钥匙给其它等待的线程，抢占到该钥匙后便能够继续线程的执行，

没有取得钥匙的线程仍然被阻塞在该对象中等待。线程同步机制的监视器原理可以用图形象地表示，如图 9.1 所示。

Java 程序中的多个线程是通过消息来实现相互的联系，提供了两个方法来实现线程之间的消息发送：`Wait()`，`notify()`。例如一个对象的

`synchronized` 方法，同一时刻只能有一个线程访问该对象的同步方法。当一个线程进入了该对象的 `notify()`方法，其它的线程被阻塞，可以通过 `s`

`notify()`方法来唤醒其它的线程。同时通过 `wait()`方法来使线程处于阻塞状态，等待由其它的线程用 `notify()`方法唤醒。

上面我们介绍了有关线程的原理和运行机制，Java 都有对应的实现方法，下面我们就具体介绍程序中线程的实现和操作方法。

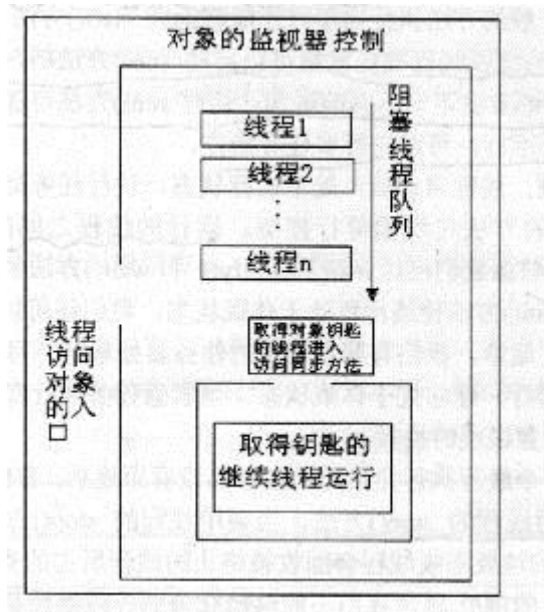


图 9.1 监视器的工作原理图

9. 2 线程的创建与优先级

9.2.1 线程的创建方法

我们知道 Java 是面向对象的程序语言，用 Java 进行程序设计就是设计和使用类。Java 为我们提供了线程类 Thread 来创建

线程。线程就是 Thread 类或者其子类的实例对象。下面我们首先介绍一个创建线程的例子。

例子如下：

```
class mainclass
```

```
{ public static void main(String args[])
```

```
{ Thread sample=new Thread();
```

```
//启动一个线程,系统为其分配内存
```

```
sample.run();
```

```
//启动该线程,由于没有重写 Thread 类的 run()方法,启动的线程没有任何操作
```

```
System.out.println(sample);
```

```
//打印创建的线程的信息
```

```
}
```

```
}
```

上面的例子完成了线程的创建和启动，创建线程使用线程的构造方法。Thread 类的构方法列于表 9—2。由表 9—2 且可以看

出，我们可以有多种构造方法创建线程。我们可以直接通过构造方法队 Thread（）构造一个线程，但是该线程没有任何操作，

通常不能够满足程序设计的要求。

含 义	方法的调用格式
创建一个空线程	public Thread()
利用实现 Runnable 接口的类的对象为参数创建一个线程，实现 Runnable 接口的类通常是实现 Runnable 接口的 run()方法，线程启动时运行实现的 run()方法	public Thread(Runnable target)
创建一个以实现 Runnable 类的对象为参数，以 name 字符串命名的线程	public Thread(Runnable target, String name)
创建一个以 name 字符串命名的线程	public Thread(String name)
创建一个属于一个线程组的线程	public Thread(ThreadGroup group, Runnable target)
创建一个有名字的属于一个线程组的线程	public Thread(ThreadGroup group, Runnable target, String name)
创建一个属于线程组的有名字的线程	public Thread(ThreadGroup group, String name)

表 9-2

上面的例子中，我们就是通过 Thread（）构造方法创建了一个线程，并且启动了这个线程。这个线程没有任何操作，这是因为 hrad 类中的 run（）方法没有任何操作语句。启动线程，也就是启动线程的 run（）方法。因此，要使线程能够实现预定的功能，必须实现自己的 run（）方法。

◎ 通过 Thread 类的子类创建线程

定义继承 Thread 类的子类，在于类中重写 Thread 类的方法 run（）来设计自己的线程体。创建该子类的实例对象就创建了一个线程，当线程启动后，启动的就是子类中重写的 run()方法。举例介绍如下：

```
class mythread1 extends Thread
{
public void run()
{ while(true)
System.out.println("this is mythread,it is runing...")
}
}
```

该例定义一个 Thread 类的子类，重写了父类的 run（）方法，实现了自己的 run（）方法（线程体）。创建该类线程体的语

句为： mythreadl threadone=new mythreadl（）；

启动线程的语句为：

threadone.start（）；

线程启动后就运行 mythreadl 类的线程体。

◎ 创建线程的例子

下面我们来介绍一个创建上面两种创建线程的实际例子！

```

class mythread1 extends Thread
{
public void run ()
{
while (true)
System.out.println (“this is mythread1 running...”);
}
}
class mythread Implements Runnable
{
public void run ()
{
while (true)
System.out.println (“this is mythread2 running...”);
}
}
class mainclass
{
Public static void main (String args [])
{
mythread1 threadone=new mythread1 ();
Thread threadtwo=new Thread(new mythread2());
threadone.start();
threadtwo.start();
}
}

```

程序运行的部分过程为:

```

this is mythread2 running ...
this is mythread2 running ...
this is mythread1 running ...
this is mythread1 running ...
this is mythread2 running ...
this is mythread2 running ...

```

由上面的线程的运行可以看出，线程 threadone 与线程 threadtwo 是交替占有 CPU，并行运行的。总之，线程是 Thread 类或者其子类的实例对象。

9.2.2 线程的优先级

我们提及线程的优先级，线程的优先级用整数 *n* 表示，线程的最高优先级是 10，最低为 1。线程类为我们定义了关于线程的优先级的静态成员变量，列于表 9-2。

含 义	方法的调用格式
<code>public final static int MAX_PRIORITY</code>	<code>public final static int MAX_PRIORITY = 10</code>
<code>public final static int MIN_PRIORITY</code>	<code>public final static int MIN_PRIORITY = 1</code>
<code>public final static int NORM_PRIORITY</code>	<code>public final static int NORM_PRIORITY = 5</code> 缺省为此优先级

表 9-2

我们可以通过 `Thread` 类名来直接使用这些成员变量，例如，`Thread.MIN_PRIORITY` 即代表整数 1。当我们创建一个线程没有设置线程的优先级时，该线程的优先级为 `NORM_PRIORITY`，即优先等级为 5。从理论上讲，高优先级的线程比低优先级的线程具有更多的机会获得运行。一个低优先级的线程在运行时，当一个高优先级的线程被唤醒时，将会立即抢占到 CPU 而得到执行。通常线程在进行 I/O 操作时会被阻塞而转向睡眠状态，当线程访问同步方法被阻塞时，也会转向睡眠状态，这样，这些线程会将 CPU 让给其它的线程，使其它的线程有更多的机会得到运行。下面我们介绍线程优先级的设置。

设置线程优先级通过调用线程对象的 `setPriority()` 方法。语句为：

```
Thread threadone=new Thread;
threadone.set.Priority();
```

可以在线程启动前调用该方法来设置线程的优先级，也可以在启动后进行优先级的设置。下面我们举例说明线程优先级的设置及作用。下面的程序明确地创建了两个线程进行信息输出，同时还有一个启动线程的主线程，在主线程中用 `ystem.in.read()` 读入一个字符。这样一共运行三个进行 I/O 操作的线程，这三个线程将分别抢占 CPU 来获得运行。

◎ 具体例子程序

```
import java.io. *;
class mythread1 extends Thread
{
public void run()
{
while(true)
System. out. println("mythread1 running... ");
}
}
class mythread2 extends Thread
{
public void run()
{
```



```

while(true)
System.out.println("thread2 running... ");
}
}
class mainclass
{
public static void main(String args[])
{
mythread1 thread1=new mythread1();
mythread2 thread2=new mythread2();
thread1.setPriority(7);
thread2.setPriority(3);
thread1.start();
thread2.start();
try
{
System.out.println( 'please input a char:' );
System.in.read();
}
catch(IOException (e)
{
System.out.println(e);
}
thread1.stop();
thread2.stop();
}
}

```

程序的运行过程:

mythread1 running ...

mythread1 running ...

mythread1 running ...

mythread1 running ...

mythread1 running ...

mythread1 running ...

mythread1 running ...

mythread1 running ...

mythread1 running ...

mythread1 running ...

mythread1 running ...

mythread1 running ...

mythread1 running ...

mythread1 running ...

mythread1 running ...

mythread1 running. ..

thread2 runing ...

可以看出, 优先级为 7 的线程占有大部分的 CPU 时间; 优先级为 3 的线程占有的 CPU 时间是非常少的。但是它们仍然是交替运行的。

9.3 Daemon 线程与线程组

◎ 创建服务线程----Daemon 线程

Daemon 线程是一个服务线程, 它的优先级是最低的, 为其它的线程提供服务。例如我们知道, 垃圾回收运行的是一个线程, 该线程就是一个 Daemon 线程。当系统中运行的只有 Daemon 线程, 则系统会自动清除 Daemon 线程, 退出 Java 解释器。

Daemon 线程也有优先级, 同样, 在线程启动前后都可以设置该线程的优先级。对于一个线程, 我们可以通过方法 `public`

`final boolean isDaemon()` 来判断一个线程是否是 Daemon 线程。

要使一个线程成为 Daemon 线程, 可以通过线程的 `public final void setDaemon(boolean on)` 方法来设置该线程

为 Daemon 线程, 该方法必须在线程启动之前调用。Daemon 线程是服务线程, 通常它的线程体是一个无限循环。

★ 举例介绍服务线程的操作

◎ 线程组

线程组提供对线程的管理, 例如, 我们可以通过线程组启动或者停止线程组中所有的线程。线程都是线程组的成员,

当我们创建线程时, 可以明确地指明该线程是某一个线程组的线程; 同时我们也可以不明确地指明该线程所属的线程组, 这时

, 线程属于 main 线程组, 这个 main 线程组称为缺省线程组。当 Java 的应用程序 Application 启动后, 系统会创建这个 main 线程组, 没有指明所属线程组的线程属于这个缺省线程组。Applet 运行时 也会相应地生成缺省线程组。

线程组中的 `uncaughtException (Thread t, Throwable e)` 方法是非常重要的。当一个线程没有用处时, 想终止线

程的执行以回收该线程所占的系统资源, 需要调用线程的 `Stop ()` 方法。调用 `stop ()` 方法会产生线程例外 `ThreadDeath`,

线程组的 `uncaughtException (Thread t, Throwable e)` 就是对该例外进行处理的方法, 它释放掉该线程所占的系统资源

以便使线程真正处于 `death` 状态而真正清除线程。因此, 我们不应该捕获对 `()` 的例外, 否则该线程没有释放掉资源, 仍然处

于 `alive` 状态。上面 `uncaughtException (Thread t, Throwable e)` 处理的过程是自动进行的, 不需要程序干预。

一个线程加入一个线程组后就永久地属于这个线程组, 不可改变。我们可以通过方法 `public final ThreadGroup`

`getThreadGroup ()` 来得到线程所属的线程组。同时, 一个线程组可以属于另一个线程组。

线程组的构造方法如下：

```
public ThreadGroup (String name) public ThreadGroup (ThreadGroup parent, String name)
```

线程组的方法介绍如下：

我们可以通过方法 `public final ThreadGroup getParent()` 来得到它所在的线程组。我们可以通过方法

`public final void setDaemon(boolean daemon)` 来设置线程组中的线程为服务线程。用 `public final void`

`setMaxPriority(int pri)` 方法来设置线程组的优先级。如果线程组中的某一个线程的优先级比 `pri` 大，则该线程的优先级不

变。可以通过方法 `public final void stop()` 来终止一个线程组中的线程的运行；通过方法 `public final void`

`resume()`、`public final void suspend()` 来唤醒线程组或者使线程组处于休眠状态；通过方法

`public int`

`activeCount()` 来得到线程组中所有的线程包括活动线程的数目；通过方法 `public int`

`activeGroupCount()`

可以得到线程组中的活动线程组数目；通过方法 `public final int getMaxPriority()`、`public final`

`String getName()` 分别获得线程组的最大优先级和线程组的名字。通过方法 `public final`

`boolean isDaemon()`

判断是否是一个 `Daemon` 线程组；通过方法 `public final boolean parentOf(ThreadGroup g)` 来判

断线程组是否是线程组 `g`，或

者是否是 `g` 的父线程组。另外还有一组方法用来得到线程组中的所有活动线程和活动线程组：

```
public int enumerate(Thread list[])
```

```
public int enumerate(Thread list[], boolean recurse)
```

```
public int enumerate(ThreadGroup list[])
```

```
public int enumerate(ThreadGroup list[], boolean recurse)
```

我们可以将一组相关的线程加入同一个线程组中，以便通过线程组的设置同时设置线程的优先级，同时设置线程其它

的状态等。我们可以通过名字来知道线程或者线程组操作的是哪一个线程。

9.4 多线程及线程同步

前面我们知道，一个程序中的多线程是交替执行的，它们的运行是无序的。当某一个线程在访问一个对象时，另一个线程

同时也可以访问该对象。例如我们有一个仓库，生产车间将产品生产出来存于仓库中，销售部门将产品从仓库中提走产品。

这个过程中，销售部门必须在仓库中有产品时才能够提货。当仓库中没有产品时，销售部门必须等待。我们可以将一个线程模

拟销售部门，另一个线程模拟生产部门，同时构造一个仓库 `buffer`。运行这样一个多线程程序，就仿真上面的实际过程。我们

根据上面的模型来用普通多线程模拟。

9.4.1 一般多线程

在下面的程序中，定义了一个 `buffer` 类，称它为仓库类。用该类就可以生成一个 `buffer` 的对象，称为仓库。该仓库类中

定义了两个成员方法：一个是 `get()`，用来模拟销售者从仓库中提走产品；另一个成员方法

为 `put()`，用来模拟产品制造者往仓库中添加产品。然后定义两个线程类：一个是 `customer` 类，在其中的 `run()` 方法中通过调用 `buf.get()` 从仓库中取走产品，模拟销售者；另一个是 `producer` 类，在其中的 `run()` 方法中通过调用 `buf.put()` 往仓库中添加产品，模拟产品制造者。这两个线程的构造方法都是以 `buffer` 类的对象为参数的。在主类中，我们首先创建了一个仓库 `buf`，然后都以 `buf` 作为构造方法的参数创建三个线程。我们知道，以对象作为方法的参数，则参数就指向了该对象，方法中对参数的操作也就是对对象的操作。所以，创建的三个线程操作同一个对象 `buf`，也就是保证在同一个仓库添加或者取走产品。

★ 具体示例程序代码

◎ 一般多线程具体示例

```
class buffer ()
{
    int i;
    public int gethuf()
    {
        return i;
    }
    public buffer(int initb)
    {
        i=initb;
    }
    public void Put()
    {
        i++;
    }
    public void get()
    {
        i--;
    }
}
class customer extends Thread
{
    buffer buf;
    public customer(buffer)
    {
        this.buf=buf;
    }
    public void run()
    {
        while(true)
        {
```

```

    bufget();
    System.out.println("take l,now total in buf="+bufgethuf());
}
}
}
class producer extends Thread
{
    buffer buf;
    Public Producer(buffer buf)
    {
        this.buf=buf;
    }
    public void run()
    {
        while(true)
        {
            bufput();
            System. out. println (" make l, now total in buf=" +buf getbuf() );
        }
    }
}
class mainclass
{
    public static void main(String args[])
    {
        buffer buf=new buffert0);
        customer custom1=new customer(buf);
        customer custom2=new customer(buf);
        producer produce=new Producer(buf);
        produce.start();
        custom1 .start();
        custom2.start();
        try
        {
            Thread.sleep(2000);
        }
        catch(ExcePhon e)
        {
            System.out.println(e);
        }
        custom1.stop();
        custom2.stop();
        produce.stop();
    }
}

```

```
}
```

程序运行部分结果为:

```
make1 ,now total in buf=-207
```

```
make l,now total in buf=-212
```

```
take l,now tOtal in buf=-210
```

```
take 1,now total in buf=-212
```

```
take l,now total in buf=-213
```

```
take l,now total in buf=-213
```

```
take l,now total in buf=-215
```

可以看出,库存的产品数最后为负值,这是由于我们没有控制这些线程的执行过程。使用线程同步将可以克服这样的问题,

下面我们将介绍线程的同步控制。

9. 4.2 线程同步控制

当我们引进线程同步后,就可以控制只能够有一个线程访问对象的同步方法。线程的同步原理我们前面已经介绍过了,每

一个对象都有唯一的一把钥匙。要访问对象的同步方法,必须首先取得对象的这把钥匙。钥匙只有一把,因此,一次只能有一个线程能够访问对象的同步方法。其它要访问该对象的同步方法的线程将被阻塞,直到占据该对象的线程放弃访问对象的同步方法。通常是调用 `ObjWCt` 中定义的 `Wait()` 方法来放弃对线程的同步方法的访问。调用方法 `nohb()` 可以唤醒等待访问同步方法的线程。唤醒线程意味着能够立即访问到同步方法。当一个线程放弃对同步方法的访问后,被阻塞的方法同时去抢占,一般优先级高的能够优先抢占到。

我们用上面的生产销售的程序的例子来说明同步的具体体现。当将仓库类 `buffer` 中的方法 `get()` 和 `put()` 方法用关键字

`synchronized` 声明为同步方法时,则 `get()` 和 `put()` 方法就被同步。这两个方法被声明同步后,就控制了同一个时刻只有一

个线程能够访问对象中的同步方法。也就是说,当一个线程访问 `get()` 方法时,其它的线程将不能够访问 `put()` 方法,必须等

到这个线程调用 `Wait()` 方法放弃钥匙,其它的线程才能够访问 `put()` 方法。

◎ 同步方法与同步语句块

用关键字 `SynCtwzed` 在方法定义时来声明方法为同步方法,同步方法有类同步方法和实例同步方法两种。当一个方法为静

态方法(即类方法)同时又声明为同步方法,则该方法为类同步方法,线程要访问类同步方法必须取得类的钥匙;当一个方法为

非静态方法被声明为同步方法时,该方法为实例对象同步方法,线程要访问实例对象的同步方法必须得到实例对象的钥匙。例如

,我们声明 `classbump()` 方法为一个公有的静态的无返回值的类同步方法:

```
public stactic synchronized void classBump ()
```

当一个类具有同步方法时,则创建的该类的实例对象就具有同步方法。线程在访问实例对象的同步方法时,需要取得该对象的钥

匙。因此,在类的定义时,声明同步必须知道,当访问该方法时线程需要取得的是哪一个对象的钥匙。对于声明一个类的成员方

法为同步方法,则访问该方法需要取得钥匙为该类的对象的钥匙。例如,下面的声明成员方法为同步方法:

```

class demo

{

synchronOnized void get()

{
}

synchronized void set()

{
}
}

```

同时，我们可以声明一个语句块为同步语句块。用关键字 `synchronollized` 在方法体声明方法的一个语句块为一个同步语句

块。对于同步块，在声明时，必须在 `synchronized` 后的括号内指明取得的是哪一个实例对象的钥匙。例如，下面的程序段说明怎

样声明一个同步语句块。在 `get（）` 方法内声明一个同步语句块，其中 `synchronized(this)` 冲的 `es` 表示线程要访问同步语句块，

需要取得的 `this` 代表的类的实例对象的钥匙，`synchronized(this)` 下面的 `{ }` 的语句是同步语句。

★ 同步语句块的例子

```

class test

{
public void get()

{
synchronized(this) //声明 get()方法中的语句为同步语句块

{
synchronizedstatement1;
synchronizedstatement2;
}
}

static void sample()
{
try
{
synchronized（Class. forNametest（test）） //指明线程访问时要取得是类 test 的钥匙
{
.....

```

```

}

}
catch (ClassNotFoundException e)

{
}
}
public synchronized void put ( ) //声明 Put ( ) 方法为同步方法
{
statement1;
statement2;
}
}

```

上面的程序段中,方法 `put()`与方法 `get()`中的同步语句块 `synchronized(this){ }`声明为同步, 线程要访问它们需要取得 `test` 对象的钥匙。一个对象只有一把钥匙, 因此, 一个线程在访问被声明为同步的 `put()` 同步方法时, 其它的线程将不能够访问该对象中的其它被声明为同步的即 `synchronized(this){ }`中的语句块, 当然也不能够访问 `put ()` 方法。这就是同步的具体体现。可以看出, 同步方法和同步语句块可以实现相同的同步功能。

例如, 下面的两种声明同步的方式其同步的功能是一样的:

同步方式一: 将方法声明同步

```

class test
{
public synchronized void get()
{
...
}
public synchronized void put()
{
...
}
}

```

同步方式二: 将方法声明同步

```

class test
{
public void get ( )
{

synchronized(this)
{

```



```

.....
}
}
public void put()
{
synchronized(this)
{
...
}
}
}
}
}

```

◎ wait()、notify()及 notifyAll()

当一个线程在访问某一个对象的同步方法完成时，可以调用 `nohb()` 来唤醒一个处于等待访问该对象同步方法的线程，也

可以调用方法 `notify()` 来唤醒所有其它处于等待访问同步方法状态的线程，然后调用 `Wait()` 方法来使该线程处于等待状态。

当一个线程被唤醒且得到钥匙后，将从它调用 `Wait()` 语句被中断的地方开始执行。一个线程调用 `Wait()` 方法将从调用 `Wait()` 的地方使线程处于等待状态。

◎ 例子：使用同步机制来实现上面的生产和销售同步

◎ 线程死锁

假设有两个线程分别访问两个对象的同步方法，在访问同步方法中，线程 1 要访问给线程 2 正在访问的对象的同步方法，同时线程 2 也要访问线程 1 正在访问的方法。这时，线程 1 在访问线程 2 正在访问的对象的同步方法时被阻塞，等待线程 2 放弃；同时线程 2 在访问线程 1 正在访问的对象的同步方法时被阻塞。这样造成两个线程都处于阻塞状态，而停止运行，称为线程死锁。因此，在线程同步时要避免线程死锁。

实际问题中死锁的例子是很多的，例如某一个工人要同时取得两把螺丝刀时才能够工作，否则处于等待状态。现在一共只有两把螺丝刀，有两个工人各有一把螺丝刀，都想要对方的一把螺丝刀来得到工作但同时谁也不想让出自己的螺丝刀，这样就造成一个无限等待的状态，即死锁。

第九讲:多线程程序设计

习 题

1.Java 是用关键字 () 来同步对共享数据操作的方法，该方法成为同步方法。

(A) Thread (B) synchronized (C) wait (D) notify

2.Java 是通过 () 方法来使进程处于阻塞状态。

(A) Thread() (B) notify() (C) synchronized() (D) wait()

3.Java 调用线程用哪种方法 ()。

(A) start() (B) run() (C) wait() (D) rusunme()

4.Java 中, 优先级最低的线程是 ()。

(A) wait (B) Daemon 线程 (C) run (D) main 线程

5.创建线程所有的方法为 ()。

(A) Thread() (B) stop() (C) wait() (D) main()

6.自己动手写一个创建线程的程序。(可参考课件里面的例子)

7.说明以下类定义的作用。

```
class mythread1 extends Thread
{
public void run()
{
while(true)
system.out.println("this is mythread1,it is running;");
}
}
```

8.简述如何会发生进程死锁。

第十讲: Applet 基础

Applet 是 java 程序进口中最具有吸引力的地方。Applet 则出运行于浏览器上, 它可以产生生动的图形用户界面, 进行

友好的人机访问, 同时它还具有处理图象、声音、动画等多媒体的功能。Applet 的强大功能在于它运行于互联网上, 可以将代

码从服务器上下载到本地机上运行。同时 Applet 还有一套安全机制用来限制 Applet 对本地的操作, 这样使客端可以不必担心诸

如病毒侵入、修改本地机文件和 Applet 运行本地机重要程序等不安全的操作。

10.1 第一个 Applet 小应用程序

我们首先来学习制作一个简单的 Applet, 并将 Applet 嵌入 HTML 页在浏览器中运, 下面是 Applet 的一个例子。

```
import java.awt.*;
import java.applet.*;
public class firstapplet extends Applet
{String mystring=new String("hello world!");
public void paint(Graphics g)
{
g.drawString(mystring,30,40);
}
}
```

该程序定义了一名为 firstapplet 的类, 它继承了 Applet 类而来。Applet 程序的主类(入口类) 必须继承 Applet 类

。在主类中创建了 `mystring` 字符串对象，重写父类的 `paint()` 方法 以便在浏览器窗口中输出 “hello world!”。用 `VJ++` 将该程序编译成 `.class` 文件。下面编写一个超文 HTML 文件，并将 Applet 嵌入， 超文本文件如下：

```
<html>
<head>
<title>firstapplet</title>
</head>
<body>
<hr>
<applet
code=firstapplet.class
width=320
height=240>
</applet>
<hr>
</body>
</html>
```

上面的文本中，Applet 程序嵌入 `<applet...></applet>` 即 Applet 标签中，`code=firstapplet.class` 用来指明要运行我们上面编译过的 `firstapplet.class` 文件。`Width=320`，`height=240` 指明 Applet 程序区域的大小。

将该超文本文件与上面编译的 `firstapplet.class` 文件存于同一个目录下。启动浏览器 IE 或者 Netscape，打开上面的超文本，就会在浏览器窗口中看到 “hello world!”。在浏览器中运行的结果如图所示：

10.1.2 Applet 的运行平台浏览器

Applet 必须依靠浏览器的解释才能运行。本节介绍浏览器平台的基本知识，以及 Applet 与浏览器的结合：HTML 如何嵌入 Applet，Applet 如何取得浏览器的信息。

◎ 浏览器

浏览器是 WWW 服务——客户程序。WWW 服务器是采用超文本（Hypertext）方式来贮存信息的，通过浏览器我们可以打开超文本文件来检索网上信息。关于 WWW 的基本知识参看第 1 章 Java 概述。早期的浏览器（如 Mosaic）只能够支持静态文本和图画，没有交互的功能。协议是内置的，不易扩充。当一种新的协议和数据类型产生时，必须将浏览器升级，否则不能够使用。传统的浏览器支持的协议也比较少，表 10—1 说明传统浏览器支持的协议，图 10—2 说明传统浏览器的运行方式。

表10-1 传统浏览器支持的协议

支持协议		
Http	html	
gif	ftp	

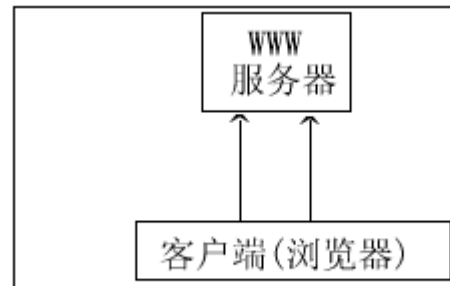


图10-2 传统浏览器的工作方式

Internet 的发展非常迅速，新的协议的出现和新的数据类型的产生使传统的浏览器的缺陷日益明显。1995 年末推出的

用 Java 语言编写的 HotJava 浏览器，体现了 Internet 发展的要求。HotJava 的协议是非内置的，它除了支持传统的协议外，还可

以根据需求动态地扩充，它是一个开放式的浏览器。表 10.2 说明 HotJava 支持的协议。

表10-2 HotJava支持的协议

Http	html	url	gif
ftp	ftp	根据需要扩充	HotJava协议

Java 语言的出现，使浏览器的功能得到了扩展，在 Web 页上可以实现动画和人机交互，这就是 Java 的 Applet 程序在浏

览器上运行的结晶。Java 语言在 Internet 上的成功，使许多大公司都支持 Java。目前支持 Java 的 Applet 程序的浏览器除了

Sun 公司的 HotJava 外，还有 Netscape 浏览器，微软公司的 Internet Explorer 浏览器，另外还有其它支持 Java 规范的浏览器。

◎ 超文本中的 Applet 标签

要运行 Applet，必须要将 Applet 嵌入 HTML 文档。关于 HTML 文档的建立已在第 1 章介绍，这里要介绍如何将 Applet 嵌

入 HTML 文档。我们知道，HTML 通过标签标识文档的各个部分。当浏览器加载 HTML 文档时，对标签进行解释，执行相应的动作，如

使用字体标签使文档的字体改变，将某些标签解释为直线等等。支持 Java 的浏览器提供了 Applet 标签，用来标识出 Applet 块

，以便浏览器将 Applet 标签内的内容解释做相应的处理。下面介绍 HTML 的 Applet 标签。

Applet 标签的格式为：

```
<applet
```

```
  [codebase=codeurl]
```

```
  code=codename
```

```
  [alt=alternatetext]
```

```

[ name= appletinstancename]
width=pixels heigh=pixels
[align=alignment] >

[<param name=paramname1 value=paranamevalue1>]
[<param name=Paramname2 value=paranamevalue2>]
[<param name=paramname3 value=Paranamevalue3>]
.....
[necessny html document being displayed]

</applet>

```

接下来，我们对该标签的格式进行一些必要的解释：

◆ **codebase=codeurl**

用该项来指明要运行的 Applet 程序所在的目录。该项是可选的，当没有用该项说明时，将默认为 HTML 文档所在目录。

◆ **code=codename**

用来指明存储主类的文件名，以便运行 Applet 程序。如果没有用 codebase 指明 Applet 程序所在的目录，则该文件

是相对于 HTML 文档所在的目录。该选项是必须的。

◆ **alt=alternatetext**

用来输出简短信息，以便不能运行 Applet 的浏览器可以通过该信息告诉用户进行的操作。该选项是可选的。

◆ **name=appletinstancename**

用来指明正在运行的 Applet 实例对象的名字，以便同一页面上的多个 Applet 实例对象通过用该语句指定的名字进行

相互的访问。该项也是可选的。

◆ **wide=pixels height=pixels**

用来指明 Applet 区域的大小。我们还可以在 Applet 程序中重新设置 Applet 区域的大小。**该项是必须的。**

◆ **align=alignment**

用来指明 Applet 在浏览器中的排列方式。Alignment 的值可以为：left、right、top、texttop、middle、

absmiddle、baseline、bottom、absbottom。该项是可选的。

◆ **param name=paramname value=paranamevalue**

参数行。参数行可以有多行，每一行是一个参数，通过参数行来将参数传递给 Applet 程序。Applet 程序通过方法

getParameter()来取这些参数。

10.2 Applet 运行机理

设计 Applet 程序，必须有一个主类，该主类必定要继承 Applet 类。Applet 有它的生命周期，Applet 的基本生命过程为：

Applet 对象创建、初始化、启动、运行、休眠和 Applet 的释放。与 Applet 生命活动密切相关的方法有 init()、start()、

paint()、stop()及 destroy()这些方法在父类中都有定义，有默认的执行动作，我们可以在子类中重写这些方法，在方法

中加入代码，来实现特定的功能。当测浏览器加载 Applet 程序时，会创建一个主类的实例，主类是继承 Applet 而来，通常就称

主类的实例为 Applet 对象，简称 Applet。

在 Applet 活动期间，当 Applet 区域被其它窗口覆盖时，AWT 会调用 paint()方法来对屏幕进行重画。当调用

repaint()方法时，repaint()方法首先调用 update()方法，然后 update()方法调用 paint()方法。有时系统处于繁忙

状态，也可能不调用 update()方法，而直接调用 repaint()方法。

接下来，我们介绍一下 Applet 生命过程中的主要方法。

◆ init() 方法

当浏览器加载 Applet 程序时，首先，创建一个 Applet 的实例对象，分配空间；接着，调用 init()方法初始化 Applet 对

象 Jinit()方法在 Applet 加载时首先被调用，且在整个 Applet 生命周期中只调用一次。

◆ start() 方法

Init()方法初始化 Applet 后，调用 start()方法启动诸如线程等动作。同时，当用户离开 Applet 所在的页面时，可以

停止不必在后台运行的一些线程来加快浏览器的运行速度；当又回到该页面时，调用该方法来重新启动被 Stop()方法

停止的线程。

◆ paint() 方法

当 start()方法执行完后，调用 paint()方法来绘制浏览器中的 Applet 区域。同时，awt 系统可以监视 Applet。如果

Applet 区域被覆盖，将调用 paint()方法来重新绘制被覆盖的部分。

◆ stop() 方法

当用户离开该 Applet 对象所在的 HTML 时，调用 stop()方法对 Applet 进行处理，如停止一些不必要在后台运行的线

程；当用户又回到 Applet 所在的页面时，调用 Start()方法来重新启动线程。

◆ destroy() 方法

当用户关闭浏览器时，调用 destroy 方法来做最后的处理，释放掉 Applet 所占的资源。

◆ repaint() 方法

我们可以通过调用该方法来对窗口进行重画。当调用 repaint()方法时，方法 repaint()首先调用 update()来填充背景

，然后由 lupdate()调用 paint()方法进行图形绘制。通常在程序中调用 repaint()方法来对屏幕强制进行重画，以便

及时更新屏幕。我们在程序中不断产生新的信息需要在屏幕上输出，而 AWT 屏幕监视系统只能够在屏幕被覆盖时才调用

repaint()方法对屏幕进行更新，屏幕没有覆盖则系统不会自动调用 repaint()方法来更新屏

幕。为此我们可以将在程序中定期地调用 `repaint()` 方法，就可以即使重画屏幕显示最新信息。

◆ `update()` 方法

当我们没有重写 `update()` 方法时，父类 `Applet` 的 `update` 缺省的执行情况是先用默认的背景颜色填充 `Applet` 区域，然后

再调用 `paint` 方法。如果 `paint()` 方法使用的背景颜色与缺省颜色不同，就会看到窗口有闪烁。因此，一般重写父类

`Applet` 的该方法，将要绘图的工作由 `update()` 方法完成，同时重写 `paint()` 方法，让 `paint()` 方法调用该方法。

`Applet` 子类继承了 `Applet` 的所有能够继承的方法，上面是类 `Applet` 的部分方法。`Applet` 还有其它方法，调用它们可以实现其它的功能。

10.3 `Applet` 的程序结构

`Applet` 与 `Application` 都是 Java 程序，都遵循前面的语法。类是构成 Java 程序的基本单元，在 `Application` 中，类的

操作是丰富多彩的，如继承、方法重载、方法隐藏等等。同样，我们可以在 `Applet` 程序中自定义需要的类来创建有特定功能的

实例对象，同时通过类的特性如继承、方法隐藏以及方法重载来实现类的灵活应用。在 `Application` 中的语法现象适用于

`Applet`。

我们知道，`Application` 运行于 Java 的解释器平台上，`Applet` 运行于浏览器窗口中。`Applet` 有自己的运行机理，与之相

对应的是程序的主类的基本结构。`Application` 程序必须有一个主类，主类中有一个主方法。由主类和主方法作为程序执行的

入口点。`Applet` 也同样有一个主类，该类必须是 `Applet` 类的子类。子类就继承了 `Applet` 类的方法，通过重写继承而来的方法

实现特定的功能。例如本章第一节的 `firstapplet` 中重写 `paint()` 方法来在浏览器中输出“hello world!”同时，`Applet` 程序

也可以定义除主类外其它的实现不同功能的类，关于这方面的知识，在前面类的操作一章已做了介绍。主类是 `Applet` 运作的核

心，下面将介绍 `Applet` 程序的主类的一个基本结构，读者可以根据提示键入相应的代码，便可以编写一个功能较强大的 `Applet`

程序。

◎ `Applet` 程序的主类的一个基本结构:

```
import java.applet.*;
import java.awt.*;

//=====
//Main Class for applet appletname
//
//=====
Public class appletname extends Applet
{
```

```

//下面是类的构造方法
//-----
public ww()
{
//TODO:在构造方法中加入代码
}

//当 Applet 是第一次被加载或者被重新加载时，AWT 将调用 init（）方法
//重写该方法，可以进行如对数据结构的初始化，加载图像，
//设置字体，创建窗口，设置布局管理器，加入图形组件等操作
//-----
public void init()
{
resize(320,240);

//TODO:在这里加入初始化 aPplat 代码

}

//当 Applet 要被终止和释放时，destroy()被调用
// 通常该方法可以用父类的方法，而不必重写
//-----
public void destroy()
{
//TODO:加入清理 Applet 的代码
}

//绘图处理方法
//-----
public void paint(Graphics g)
{
g.drawString("Created with Microsoft Visual J++ Version 1.1",10,20);

//可以加入绘图语句，在 Applet 区域绘制所要的图形
}

//当 Applet 所在的页面第一次出现在屏幕上，或者用户离开该页面后又回到该页面时，调用
Staft（）方法；
//可以用该方法来启动线程。也可以执行其他的语句
//-----
public void start()
{
// 这里加入要运行的代码
}

```


//当用户离开 Applet 所在的页面时，调用 stop（）方法，
//通常，通过该方法来停止线程的运行，
//也可以加入其它一些处理语句，此时 Applet 并没有被释放,而是处于后台休眠状态

//-----

```
public void stop()
```

```
{
```

//TODO:这里加入要运行的代码

```
}
```

//下面是对鼠标事件的处理：

//当鼠标在 Applet 区域按下键时，mouseDown（）方法被调用

//-----

```
public boolean mouseDown(Event evt,int x,int y)
```

```
{
```

//里可以加入处理鼠标键按下的事件处理代码

```
return true;
```

```
}
```

//下面是对鼠标事件的处理

//当鼠标在 Applet 区域放开键时，mouseUp（）方法被调用

//-----

```
public boolean mouseUp(Event evt,int x,int y)
```

```
{
```

//TODO:在这里可以加入处理放开鼠标键事件的处理代码

```
return true;
```

```
}
```

//下面是对鼠标事件的处理

//当在 Applet 区域按下鼠标键并拖动时，将调用 mouseDrag（）方法

//-----

```
public boolean mouseDrag(Event evt,int x,int y)
```

```
{
```

//TODO:在这里加入对鼠标拖动事件的处理代码

```
return true;
```

```
}
```

//鼠标事件处理

//当鼠标没有按下任何键而在 Applet 区域中移动时，mouseMove()方法将被调用

//-----

```
public boolean mouseMove(Event evt,int x,int y)
```

```

{
//TODO:加入鼠标移动事件的处理代码
return true;

}

//鼠标事件处理
//当鼠标移入 Applet 区域时， mouseEnter()方法将被调用
//-----
public boolean mouseEnter(Event evt,int x,int y)
{
//TODO:这里加入鼠标移入 Applet 区域事件的处理代码
return true;

}

//鼠标事件处理
//当鼠标移出 Applet 区域时， mouseExit（）方法将被调用
//-----
public boolean mouseExit(Exit evt,int x,int y)
{
//TODO:这里加入对鼠标移出事件的处理代码
return true;

}

```

//TODO:另外，在这里你可以自定义一定功能的方法，以便用来被上面的方法调用

//这个框架中添入代码，便可以设计出一个具有较强功能的 Applet 小应用程序

接下来，我们对 Applet 主类作几点说明：

- ◆在该主类中，重写了父类的 init（）方法来初始化 Applet，该程序中调用 resize()方法来设置 Applet 在浏览器中的大小。

- ◆重写 Start（）方法来启动一些操作，如启动一个或者多个线程。

- ◆重写 paint（）方法，在浏览器中输出信息，作图等。

- ◆重写 destroy（）方法，以便用户在关闭浏览器时，先对 Applet 进行必要的处理后，再关闭浏览器。

10.4 Applet 程序设计

前面我们介绍了 Applet 的生命周期，知道 Applet 程序运行的过程。Applet 程序与 Application 程序的区别在于：

Applet 程序的主类必须要继承 Applet 类，当浏览器加载 Applet 程序时，创建 Applet 对象，Applet 对象就在它的使用寿命中有规律活动，执行相应的方法。因此，我们重写这些方法，在方法中加入我们要执行的代码，当 Applet 的这些方法被调用时，加入的代码便得到执行。而 Application 程序的主类必须拥有 main () 方法，在 main () 方法加入代码，顺序地执行程序语句，直到 main () 方法返回，程序运行结束。

我们知道，Nplet 的生命活动是通过有规律地对方法 init ()、start ()、paint ()、stop () 和 destory () 的调用来体现的。另外，相关的方法还有 repaint () 方法和 update () 方法，它们在屏幕被覆盖时 AWT 会自动调用 repaint () 方法，在 repaint () 方法中调用 update () 方法，update () 做屏幕的背景处理后会调用 paint () 方法。下面我们重写上面的方法，在这些方法中加入代码，设计 Applet 程序。

本章前面的几节主要讨论的是 Applet 主类。主类是程序的核心，程序的引擎。可以看出，Applet 主类的运行机理与 Application 不同，即它们的引擎不同，这是运行平台决定的。他们都是 Java 程序，都遵从共同的语法（如：它们的程序结构是相同的，都是由类组成），它们是统一的，这在程序既可以作为 Applet 又可以作为 Application 方面可以得到体现。因此前面的 Java 语法学习是进行 Applet 设计的基础，后面介绍的 AWT 图形用户设计其实就是介绍 AWT 类，介绍类的功能，从而能够使用类。熟练地掌握类的操作，是进行功能强大的程序设计的必然要求。一个程序既可以作为 Applet 程序也可以作为 Application 程序，其标志是看程序中是否有它们的主类。若程序中既有 Applet 主类又有 Application 主类，则这个程序既可以作为 Applet 在浏览器中运行，又可以在解释器上运行。下面我们分别具体地介绍 Applet 程序的设计。

◎ 基本 Applet 程序

在 Applet 程序设计中一般包括类的引入、主类的定义，也可以定义附加类，用附加类在 Applet 中创建具有一定功能的实例对象，完成一定的功能。Applet 的主类是必不可少的，必要的时候可以用 package 语句将源文件中定义的存于指定的包。我们在 Applet 程序设计中经常要用到 java.awt 包中的类，通常需要将下面两条语句引入源文件：

```
import java.awt.*;
import java.applet.Applet;
```

当我们用到其它包的类时，一般也应引入。

在 Java 的程序设计中，类是基本单元。一个 Applet 除了必须定义主类外，可以定义其它的附加类。Java 程序完整的结构可以表示为：

```
[package...;]
[import...;]
import java.applet.Applet;
```

```
[class myone //在 Applet 类中定义一个附加类 myone
{
...
}]
```

```
[class mytwo //在 Applet 类中定义另一个附加类 mytwo
{
...

}]
```

[其它附加的类定义]

```
class mainclass extends Applet //定义主类，主类必须继承 Applet 类
{
[myone one=new myone();] //在主类中创建附加类对象示例
[mytwo tWoonew mytWo();]
...
}
```

```
Public void init() //我们可以在其它的方法中创建附加类
{
}
}
```

计算 1,2,3,4...,99,100 的和

```
import java.awt.*;
import java.applet.Applet;
```

```
class cal
{
public int getResult()
{
int result=0;
for(int i=1;i<=100;i++)
{
result=result+i;
}
return result;
}
}
```

```

class mainapplet extends Applet
{
    cal calone;
    public void init()
    {
        setBackground (Color. white);
        calone=new cal();
    }
    Public void paint(GraPhics g)
    {
        g.setColor(Color.blue);
        g.setFont(new Font("TimesRoman",Font.ITALIC, 20) );
        g.drawString("The result of the problem is ",20,80);
        g.drawString("1+2+3+...+99+100= "+calone.getresult(),30,110);
    }
}

```

上面程序运行需要的 HTML 文档如下：

```

<html>
<head>
<title>mainapplet</title>
</head>
<body>
<hr>
<applet

code=mainapplet
weidth=340
height=200>

</applet>
<hr>
</body>
</html>

```

一个既作为 Applet 又可作为 Application 的程序

在本节第一个例子中，定义了两个类：一个主类，一个附加类。用附加类来创建一个对象，用该对象来计算 $1+2+\dots+99+100$ 的值。

下面我们举例说明一个既作为 Applet 又可以作为 Application 的 Java 程序。

例 2. 一个既作为 Applet 又可作为 Application 的程序

该程序的功能与例 1 相同，都是计算 $1+2+\dots+99+100$ 的值。该程序中定义了三个类：Cal 类、Application 类、appletmain 类。其中 applicationmain1 为 Application 程序的主类，Appletmain 为 Applet 程序的主类。当我们要在解释器

上运行时，主类应是 applicationmain，命令行格式为：

```
jview.exe applicationmain
```

当我们在浏览器上运行时，主类为 Appletmain，Applet 标签中的 code 行对应的是 Applet 主类的类名。该程序作为 Applet 程序的 Applet 标签为：

```
<applet
```

```
code=appletmain
```

```
weidth=340
```

```
height=200>
```

```
</applet>
```

程序代码如下：

```
import java.awt.*;
```

```
import java.applet.Applet;
```

```
class cal
```

```
{
```

```
Public int getResult()
```

```
{
```

```
int resuled;
```

```
for(int i=1;i<= 100;i++)
```

```
{
```

```
result=result+i;
```

```
}
```

```
return result;
```

```
}
```

```
}
```

```
class applicationmain
```

```
{
```

```
Public stactic void main(String args[])
```

```
{
```

```
cal calone=new cal();
```

```
System.out.Println(" 1+2+3+...+99+100= "+calone.getResult());
```

```
}
```

```
}
```

```
class appletmain extends App1et
```

```
{
```

```
cal calone;
```

```
Public void init()
```

```
{
```

```
setBackground(Color.white);
```

```

calone=new cal();
}
public void (Graphics g)
{
g.setColor(Color.blue);
g.setFont(new Font("TimesRoman",Font.ITALIC, 20));
g.drawString(" 1+2+3+...+99+100)= "+calone.getresult(),30, 1 10);
}
}

```

习 题

- 1.Applet 的强大功能在于它运行于 () 上。
(A) Frontpage 2002 (B) Dreamweaver (C) HTML (D) 互联网
- 2.用 VJ++ 可将 Applet 程序编译成 () 文件。
(A) .bat (B) .class (C) .html (D) .App
- 3.Applet;在 HTML 语言中应嵌套在 () 之间。
(A) <body>...</body> (B) <applet>...</applet>
(C) <HTML>...</HTML> (D) <Script>...</Script>
- 4.下面用 Java 语言编写的浏览器是 ()
(A) IE 6.0 (B) Netscap (C) HotJava (D) 没有
- 5.在 Applet 活动期间,当 applet 区被其它窗口覆盖时,AWT 用调用 () 方法对屏幕进行重画。
(A) wait() (B) update() (C) repaint() (D) paint()
- 6.当用户关闭浏览器时,调用 () 来释放 Applet 所占的资源。
(A) stop() (B) init() (C) update() (D) destroy()
- 7.在 Applet 程序中经常用到的类是 ()
(A) class (B) package.awt (C) Applet (D) java.awt
- 8.当调用 repaint()方法时,先调用 () 方法,再对屏幕进行重画。
(A) update() (B) paint() (C) stop() (D) init()
- 9.下列浏览器中没有交互功能的是 ()。
(A) IE 6.0 (B) Mosaic (C) Netscap (D) HotJava
- 10.我们通常用 () 方法来放弃对线程的同步方法访问。
(A) wait() (B) stop() (C) put() (D) get()
- 11.在 Applet 中用到 Java.awt 包中的类时,所必须引用的两条语句为:
_____, _____。
- 12.要取得 name 为 red 的参数的 Value 值,其操作语句为:
_____。
- 13.与 Applet 生命过程中密切相关的方法有_____, _____, _____,
_____, _____, _____等。
- 14.简述一下 Applet 中提供的安全机制。
- 15.请指出 Applet 标签格式中,codebase=codeurl,code=codename,alt=alternatetext 等语句的功能。
- 16.将本章中第一个 Applet 程序进行编译(可用任何一种编译工具),然后嵌入到 HTML 中,

看看效果。

第十一讲：多媒体技术基础

11.1 引言

在进行多媒体设计时，我们经常要用到下面的类：Font、FontMetrics、Color、Graphics、Dimension、Image、

Inset、MediaTracker、Point、Polygon、Rectangle、Toolkit，这些类都属于 java.awt 包。这些类的继承关系如图 11.1

所示。

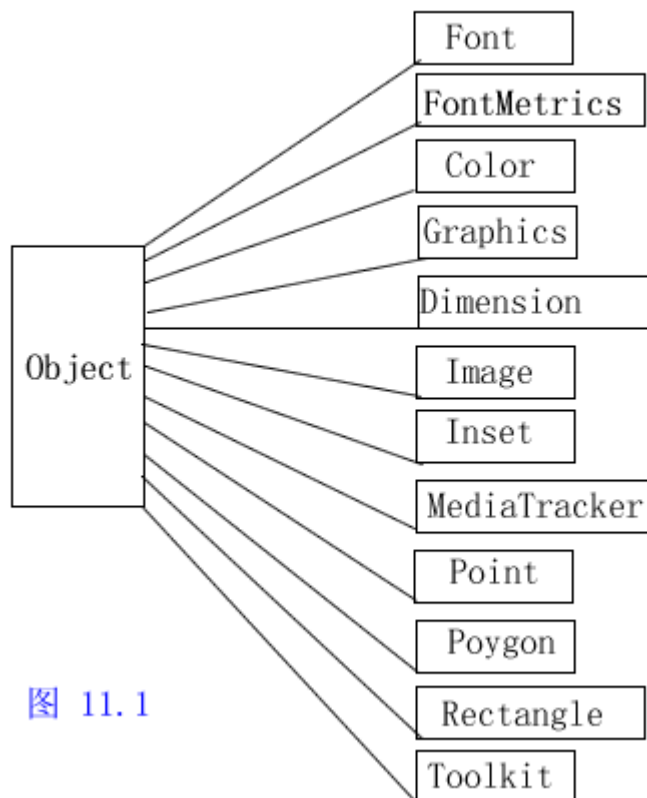


图 11.1

从继承关系图 11.1 得知，这些类都是继承 Java 的最高父类 Object，这些子类继承父类的方法，拥有父类的方法。为此，我们介绍最高父类 Object。最高父类属于 java.lang 包。

11.2 对象的属性

我们知道，Java 是面向对象的程序设计语言，对象有它的属性（例如文本区组件对象有文本的字体、颜色、背景等

等属性)，对象的属性通常是用对象成员变量的值来表示。要改变对象的属性，可以有两种方式：第一种直接访问对象的成

员变量。这种方式既不便于使用，同时也不安全，在类的设计时，通常避免这种直接对对象成员变量的访问，一般用下面的

第二种方法来改变对象的属性。第二种是通过调用对象的方法来改变对象的属性，这是一种安全方便的比较好的方式。

◎ 下面的例子说明上面两种改变对象的属性的方法：

```
import java.awt.*;
import java.applet.Applet;

class sample
{
    int i=100;
    int j=10;

    Setting(int i, int j)
    {
        this.i=i;
        this.j=j;
    }
}

class myapplet extends Applet
{
    sample mysample;

    Public void init()
    {
        mysample=new sample(); // 创建 sample 类的实例对象

        mysamp1e.i=50; // 直接访问对象 mysample 的成员变量 i，改变它的值

        mysample.j=5; // 直接访问对象 sample 的成员变量 j，改变它的值

        mysample.setting(50,5); // 通过对象 mysample 的方法 setting（）来访问对象成员变量 i,j
    }
}
```

上面我们说明了通过访问对象的成员变量改变对象的属性的方法。在 AWT 界面设计中，改变对象的属性的方法一般是

第二种：通过调用对象的方法，每一个对象都有自己的属性如颜色、字体等。不同的组件通过该组件的方法来设置组件自己

的属性（字体、颜色等），因此我们调用对象的方法设置的是被调用的对象的属性而不是其它对象的属性。组件有它的设置

字体的方法，如绘图环境有绘图环境设置字体的方法。**Applet** 是一个容器组件，它有字体、背景颜色、前景颜色等属性。设

置组件的属性的方法在 **Component** 类中有定义，组件都是继承 **Component** 类而来，因此可以通过 **Component** 类中定义的方法来

改变组件的属性。绘图环境（实现抽象类 **Graphics** 的子类的实例，由绘图环境产生）也有字体、背景颜色、前景颜色等属性

，设置这些属性的方法在 **Graphics** 类中都有定义，因此可以通过 **Graphics** 类中定义的方法接口来设置对象的属性。

11.3 字体与颜色的设置

Java 提供了对字体的封装 **Font** 类。我们可以通过 **Font** 类创建 **Font** 的对象，然后通过 **setFont()**方法来设置字体。

11.3.1 Font 类

接下来,我们来看看 **Font** 类的定义:

```
public class java.awt.Font
extends java.lang.Object
{

    // Fields

    protected String name;
    protected int size;
    protected int style;

    // style has the following bit masks

    public final static int BOLD;
    public final static int ITALIC;
    public final static int PLAIN;

    // Constructors

    public Font(String name, int style, int size);

    // Methods

    public boolean equals(Object obj);
    public String getFamily();
    public static Font getFont(String nm);
    public static Font getFont(String nm, Font font);
```

```

public String getName();
Public int getSize();
public int getStyle();
public int hashCode();
public boolean isBold();
Public boolean isItalic();
public boolean isPlain();
public String toString();
}

```

接下来,我们对该类的定义做一些必要的解释:

◎ 通过 Font (string name, int style, int size) 构造方法来创建 Font 对象。创建 Font 对象需要三个参数:

String name: 要创建的字体对象的字体名称, 我们通过 `getFontList()` 方法可以得到支持的字体。Font 类提供对如下几种类字体的支持: Dialog、Helvetica、TimeRoman、Courier、DiagInput 等。

int style: 字体的样式, 字体的样式为 BOLD、IC、PLAIN;

int size: 字体的大小, 根据需要可以设置要显示的字体的大小。

◎ 通过 `getFont(string nm)` 方法可以得到当前使用的字体对象。另外, 我们还可以通过 Graphics 类或者组件的方法 `getFont()` 得到使用的字体。

◎ 可以通过 `getName()` 方法来获得当前使用的字体的名字。

◎ 可以通过 `getSize()` 方法来获得当前使用的字体的尺寸。

◎ 可以通过 `isBold()` 方法判断当前使用的字体的样式是否为粗体。

◎ 可以通过 `isItalic()` 方法判断当前使用的字体的样式是否为斜体。

◎ 可以通过 `isPlain()` 方法判断当前使用的字体的样式是否为正常字体。

11.3.2 字体 (Font)对象的创建

通过上面的介绍, 我们要创建 Font 字体, 需要给定构造方法中的三个参数, 而后再通过构造方法 Font (string name, int style, int size) 来创建 Font 对象。例如我们创建一个字体对象, 该对象的字体的名为 TimesRoman,

字体的样式为 ITALIC, 字体的大小为 20, 创建格式如下:

```
Font boldfont=new Font("TimesRoman",ITALIC,20);
```

通过 Graphics 的方法 `setFont()` 来设置输出的字符的字体。如: `g.setFont(boldfont);` 同时还可以通过其它的组件的 `SetFont()` 方法来设置字体。

上面我们介绍了 Font 类的有关操作和字体的创建, 下面我们将介绍怎样进行绘图颜色的设置。

11.3.3 设置字体

在 AWT 界面设计中, 组件和绘图环境都有设置字体的方法, 我们举例说明。下面是获得 AWT 支持的字体, 并将其显示

于 Applet 区域的 Applet 源文件。在程序中，我们通过方法 `getToolkit().getFontList()` 来获得 AWT 支持的所有字体，通过 `Graphics` 类的 `setFont()` 方法来设置字体。

★ 设置字体例子演示

```
import java.applet.Applet;
import java.awt.*;

class appletfonts extends Applet
{
    String[] fontarray;
    Public void init()
    {
        resize(640,480);
        fontarray=getToolkit().getFontList();
    }
    Public void paint(Graphics g)
    {
        Font sfont=getFont();
        g.drawString("fontstyle=BOLD",20,20);
        for(int i=0;i<fontarray.length-1;i++)
        {
            Font font1=new Font(fontarray[i],Font.BOLD,20);
            g.setFont(font1);
            g.drawString(fontarray[i],i+20,i * 50+60);
            font1=null;
        }
        g.setFont(sfont);
        g.drawString("fontstyl=ITALIC", 170,20);
        for(int i=0; i<=fontarray.length - 1; i++ )
        {
            Font font1=new Font(fontarray[i],Font.ITALIC,20);
            g.setFont(font1);
            g.drawString(fontarray[i], i+170,i * 50+60);
            font1=null;
        }
        g.setFont(sfont);
        g.drawString("fontstyle=PLAIN", 300,20);
        for(int i=0;i<=fontarray.length-1;i++)
        {
            Font font1=new Font(fontarray[i],Font.PLAIN, 20);
            g.setFont(font1);
            g.drawString(fontarray[i], i+300,i * 50+60);
            font1=null;
        }
    }
}
```

```
}  
}  
}
```

下面是嵌入了上面源程序编译后的 applet. class 代码的 HTML 文档:

```
<html>  
<head>  
<title>appletfonts</title>  
</head>  
</body>  
<hr>  
<applet  
code=aopletfonts  
width=640  
height=480>  
</applet>  
<hr>  
</body>  
</html>
```

将上面的 HTML 文档与 appletfonts. class bytecode 文件存于同一个目录下, 用 IE 浏览器打开上面的 HTML 文件。

11.3.3 颜色的设置

Java 提供对颜色的封装 Color 类。要设置颜色, 首先要创建颜色实例对象。下面分别介绍。

◎ Color 类的定义如下:

```
public final class java.awt.Color extends java.lang.Object  
{
```

```
// Fields
```

```
public final static Color black;  
public final static Color blue;  
public final static Color cyan;  
public final static Color darkGray;  
public final static Color gray;  
public final static Color green;  
public final static Color lightGray;  
public final static Color magenta;  
public final static Color orange;  
public final static Color pink;  
public final static Color red;  
public final static Color white;  
public final static Color yellow;
```

```

//Constructors
public Color(float r, float g, float b);
public Color(int rgb):
public Color(int r, int g, int b);

//Methods
public Color brighten();
public Color darken();
public boolean equals(Object obj);
public int getBlue();
public static Color getColor(String nm);
public static Color getColor(String nm, Color v);
public static Color getColor(String nm, int v);
public int getGreen();
public static Color
getHsbColor(float h, float s, float b);
public int getRed();
public int getRGB();
public int hashCode();
public static int HSBtoRGB(float hue, float saturation, float brightness);
public static float[] RGBtoHSB(int r, int g, int b, float[] hsbvals);
public String toString();
}

```

接下来我们对 Color 的定义进行一些必要的解释。

★ 我们可以通过构造方法 Color (float r, float g, float b)、Color (int rgb) 和 public Color(int r, int

g, int b) 来创建自定义的 Color 对象，以调配不同的颜色。

★ Color 类为我们提供了基本颜色的静态对象，当我们要用到这些颜色时，可以直接用这些对象来设置颜色，而省去自

定义的工作。这些颜色对象为 black、blue、cyan、darkGray、gray、green、lightGray、magenta、orange、pink、red、

white、yellow。这些对象是 Color 的静态成员，引用类的静态成员可以通过类名直接引用，例如引用 red 对象 Color.red。

★ 通过 getBlue () 得到当前使用的颜色的蓝色成份，通过 getGreen () 方法来得到当前使用的颜色的绿色成份，通

过 getRed () 方法得到红色成份。上面介绍了 Color 的基本操作，下面介绍怎样创建颜色对象。

◎ 颜色对象的创建

要创建一个颜色对象，我们根据构造方法给出相应的参数，这里以构造方法 Color(int r, int g, int b)为例进行

介绍。根据这个构造方法，我们应给出红 (r)、绿 (g)、蓝(b)则三种颜色成份值 (整型)，取值范围为：0~255。我们取值

为:r=100, g=20, b=20 创建一个颜色对象 mycolor 如下：

```
Color mycolor=new Color(100,20,20);
```

可以通过 Graphics 类的 setColor () 方法来设置颜色，也可以通过其它组件的 setColor()方法来设置颜色。下面介绍有关作图的操作。

◎ 颜色的设置

前面我们介绍过 Color 类，其中封装了一些静态的颜色成员对象，当我们需要用这些颜色对象时无需重新定义。下面的程序是将这些颜色显示在 Applet 区域。

★ 颜色设置的具体实例

具体源程序如下：

```
import java.applet.Applet;
import java.awt. *;
class appletcolor extends Applet
{
    Color[][] Colormay=new Color[3][5];
    String[][] Stringarray=new String[3][5];
    Public void init()
    {
        setBackground(Color. white);
        Colorarray [0] [0] =Color. red;
        Stringarray[0][0]="red";
        Colorarray[0][1]= Color.green;
        Stringarray[0][1]="green";
        Colorarray[0][2]= Color.blue;
        Stringarray [0] [2] ="blue";
        Colorarray[0][3]= Color.yellow;
        Stringarray [0] [3] ="yellow";
        Colorarray[0][4]= Color.white;
        Stringmy [0] [4] = "white";
        Colorarray[1] [0]= Color.orange;
        Stringarray [1] [0] = "orange";
        Colormay[1][1]= Color.pink;
        Stringarray[1][1]="pink";
        Colorarray[1][2]= Color.magenta;
        Stringmy[1] [2] ="magenta";
        Colorarray[1][3]= Color.cyan;
        Stringarray[1][3]="cyan";
        Colorarray[1][4]= Color.darkGray;
        Stringarray [1] [4] = "darkGray";
        Colorarray[2][0]= Color.gray;
        Stringmp [2] [0] ="gray";
        Colorarray[2][1]= Color.lightGray;
```

```

Stringarray[2][1]="lightGray";
Colorarray[2][2]= Color.black;
Stringarray [2] [2] ="black";
resize(640,480);
}
Public void paint(Graphics g)
{
setBackground(Color. white );
g.setFont(new Font(g. getFont() .getName(),g. getFon() .getStyle(),20));
for(int i=();i<=2;i++)
{
for(int j=0;j<=4;j++)
{
if(i==2&&j>=3)
continue;
g. setColor(Coloraarray [i][j] );
g.fillRect(40+j*50, 40+i*100, 50,50);
g.drawString(Stringarray[i][j],40+j*50,i*100+120);

}
}
}
}
}

```

下面是上面的 bytecode 要嵌入的 HTML 文档，它与 byteCode 存于同一个目录中。

```

<html>
<head>
<title > app1etcolor</title>
</head>
<body>
<hr>
<applet
code=appletco1or
width=640
height=480>
</applet>
<hr>
</body>
</html>

```

在上面的程序中，我们分别用不同的前景颜色填充了 13 个方块，并且每一个方块下面用同样颜色的字注明了该颜色的名称。我们将背景颜色设置成白色，所以第一行最右边的白色方块被白色的背景隐藏了。对上面的程序的几点说明：

我们可以通过组件的 setBackground（）来设置背景颜色。Applet 继承了父类的

`setBackground()` 方法，`Applet` 的子类的实例对象也是一个组件，可以通过 `setBackground()` 方法来设置背景颜色。上面的程序中我们正是通过该方法来设置背景颜色的，同时可以用 `Graphics` 类的 `setColor()` 方法来设置绘图颜色。在 HTML 文档中设置 `Applet` 区域大小，以便使 `Applet` 区域能够充分地显示程序的输出结果。该程序帮助我们复习怎样使用类的静态成员，同时也可以帮助我们领会方法继承的强大功能。

`Applet` 子类从父类继承了许多方法，通过这些方法可以实现许多功能，例如上面我们通过继承而的方法 `setBackground()` 来设置背景颜色。关于 `Applet` 的继承关系，可以查看 Java 的继承图或者 Java 的类包。

11.4 Graphic 类与图象处理

11.4.1 Graphics 类

`Graphics` 类主要用于绘图，它提供了绘制、填充各种几何图形，获取颜色、字体信息等方法。

◎ `Graphics` 类的定义及主要方法

`Graphics` 定义如下：

```
public abstract class java.awt.Graphics
extends java.lang.Object
{
// Constructors
    protected Graphics();
// Methods
    public abstract void clearRect(int x, int y, int width, int height);

    public abstract void clipRect(int x, int y, int width, int height);

    public abstract void
    copyArea(int x, int y, int width, int height, int dx, int dy);

    public abstract Graphics create();
    public Graphics create(int x, int y, int width, int height);
    public abstract void dispose();
    public void draw3DRect(int x, int y, int width, int height, boolean raised);

    public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle);

    public void drawBytes(byte data[], int offset, int length, int x, int y);

    public void drawChars(char data[], int offset, int length, int x, int y);

    public abstract boolean
    drawImage(Image img, int x, int y, Color bgcolor, ImageObserver observer);
```

```
public abstract boolean  
drawImage(Image img, int x, int y, ImageObserver observer);
```

```
public abstract boolean  
drawImage(Image img, int x, int y, int width, int height, Color bgcolor,  
ImageObserver observer);
```

```
public abstract boolean  
drawImage(Image img, int x, int y, int width, int height, ImageObserver observer);
```

```
public abstract void drawLine(int x1, int y1, int x2, int y2);
```

```
public abstract void drawOval(int x, int y, int width, int height);
```

```
public abstract void  
drawPolygon(int xPoints[], int yPoints[], int nPoints);  
public void drawPolygon(Polygon p);  
public void drawRect(int x, int y, int width, int height);
```

```
public abstract void drawArc(int x, int y, int width, int height, int arcWidth, int arcHeight);
```

```
public abstract void  
drawString(String str, int x, int y);
```

```
public void fill3DRect(int x, int y, int width, int height, boolean raised);
```

```
public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle);
```

```
public abstract void fillOval(int x, int y, int width, int height);
```

```
public abstract void fillPolygon(int xPoints[], int yPoints[], int nPoints);
```

```
public void fillPolygon(Polygon p);
```

```
public abstract void  
fillRect(int x, int y, int width, int height);
```

```
public abstract void  
fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight);
```

```
public void finalize();  
public abstract Rectangle getClipRect();  
public abstract Color getColor();
```

```

public abstract Font getFont();
public FontMetrics getFontMetrics();
public abstract FontMetrics getFontMetrics(Font f);
public abstract void setColor(Color c);
public abstract void setFont(Font f);
public abstract void setPaintMode();
public abstract void setXORMode(Color cl);
public String toString();
public abstract void translate(int x, int y);
}

```

上面是 `Graphics` 类中定义的方法，我们可以使用其中的方法来进行图形绘制，分别说明如下。

1. 图形绘制方法

- ◎ 作立体四边形方法 `draw3DRect()`;
- ◎ 作弧方法 `drawArc()`;
- ◎ 将图画显示在 Applet 窗口中 `drawImage()`
- ◎ 作直线方法 `drawLine()`;
- ◎ 作椭圆方法 `drawOval()`;
- ◎ 作多边形方法 `drawPolygon()`;
- ◎ 作矩形方法 `drawRect()`;
- ◎ 作圆角矩形方法 `drawRoundRect()`;
- ◎ 显示字符串方法 `drawString()`;

2. 填充方法

- ◎ 填充指定的矩形区域方法 `fill3DRect()`;
- ◎ 填充指定圆弧区域方法 `fillArc()`;
- ◎ 填充椭圆区域方法 `fillOval()`;
- ◎ 填充多边形区域方法 `fillPolygon()`;
- ◎ 填充矩形区域方法 `fillRect()`;
- ◎ 填充圆角矩形方法 `fillRoundRect()`;

3. 该类中提供的其它方法

- ◎ 可以用方法 `getColor()` 得到当前使用的颜色对象;
- ◎ 可以用方法 `getFont()` 得到当前使用的字体;
- ◎ 可以用方法 `setColor(Color c)` 来设置颜色;
- ◎ 可以用方法 `setFont(Font font)` 设置字体。

当我们要调用这些方法时，需要给出方法的相应参数。这些方法及参数都在上面的类中定义了，使用时查看上面类中的

方法即可。如：我们要调用 `drawLine()` 方法，直上面的 `Graphics` 类可以确切地知道该方法的具体定义为：

```
public abstract void drawLine (int x1, int y1, int x2,int y2)
```

因此，要调用该方法，就必须给出直线的两点的坐标：`x1, y1, x2, y2`。上面介绍了 `Graphics` 类及其部分关于绘图的

方法的功能，下面介绍怎样使用 `Graphics` 类及其方法。

◎ 绘图方法的具体使用

我们知道，抽象类是不能够创建实例对象的。Awt 能够生成一个实例，可以通过 Graphics 类的变量引用，该实例具有上面的方法。这个实例可以作为方法 paint()、update () 的参数，因此可以在方法 Paint ()、update () 中操作 AWT 产生的这个实例。例如：

```
public void update(Graphics g)
{
    g.drawString("glad to see you ",10,20);
    g. draw3DRect(50, 50, 20,20, true);
    .....
}
```

◎ 利用 Graphics 封装的方法进行绘图

在下面的程序中我们用类 Graphics 的绘图方法进行基本图形的绘制，关于该类的其它方法可以参见上面的

Graphics 类：

```
import java.applet.Applet;
import java.awt.*;
class paintPicture extends Applet
{
    public void init()
    {
        setBackground(Color. white );
    }
    public void paint(Graphics g)
    {
        g.setColor(Color.blue);
        g.setFont(new Font("TimesRoman",Font. BOLD, 20 ));
        for(int j =20;j <= 170;j =j +50)
        {
            g.drawLine(20,j, 420,j);
        }
        for(int i=20;i<=420;i=i+1(X))
        {
            g.drawLine(i,20,i,170);
        }.
        g.setColor(Color.red);
        g.drawRect(40,30,60,30);
        g.fillRect(40,80,6(),30);
        g. MString("ReCt",40, 165 );
        g. drawRoundedRect(140, 30,6(), 3 0, 8, 8 );
        g. fillRoundedRect(140,80,6(), 30, 8, 8);
        g. drawString("ThundRect", 120, 165 );
        g. for3DRect(240, 30,ed,30,true);
    }
}
```

```

g.fill3DRect(240, 80,60, 30, true);
g.drawString("3Drect T",220,165);
g.draw3DRect(340, 30,60, 30,false);
g.fill3DRect(340, 80,60, 30,false);
g.drawString("3Drect F",320,165);
}
}

```

嵌入的 HTML 文档如下：

```

<html>
<head>
<title>paintpicture</title>
</head>
<body>
<hr>

<applet>
code=paintpicture
width=640
height=480>

</applet>
<hr>
</body>
</html>

```

11.4.2 图象的处理

◎ 普通图象的装载

要显示图像，首先要将图片数据从网络或者磁盘上装载于内存中，装载图像数据也就是创建一个 Image 对象。创建

Image 对象通常不用它的构造方法，而是通过调用 Applet 类对象下面方法的返回值获得：

```
public Image getImage(URL ur)
```

```
public Image getImage(URL, String name)
```

上面的两个方法都可以返回 Image 对象，将要显示图像的数据装入内存。其中 URL 为图像文件的资源定位地址，通常我们

可以通过下面的 Applet 方法来获得图像文件的相对地址：

```
public URL getCodeBase () //获得正在运行的 ByteCode 代码的地址
```

```
public URL getDocumentBase () //获得嵌有运行该 Java 程序的 HTML 的地址
```

创建了 Image 对象后，就可以将图像绘制在屏幕上了。通常我们通过绘图环境方法：

```
boolean drawImage(Image img, int x, int y, ImageObserver observer);
```

```
boolean drawImage(Image img, int x, int y, Color bgcolor, ImageObserver observer);
```

```
boolean drawImage(Image img, int x, int y, int width, int height, Color bgcolor, ImageObserver
```

```
observer);
```

```
boolean drawImage(Image img, int x, int y, int width, int height, ImageObserver observer);
```

将图像绘制到屏幕上。其中 ImageObserver observer 为图形控制对象，我们将在下一节介绍。

可以通过下面的方法来

获取 Image 对象的高度和宽度：

```
public abstract int getHeight (ImageObserver observer)
```

```
public abstract int getWidth(ImageObserver observer)
```

接下来我们举一个例子：

```
import java.awt. *;
```

```
import java.applet.Applet;
```

```
import java.net. *;
```

```
class myimage extends Applet
```

```
{
```

```
Image img;
```

```
public void init()
```

```
{
```

```
img=getImage(getCodeBase(), "zy1.jpg");
```

```
resize(640,480),
```

```
}
```

```
public void paint(Graphics g)
```

```
{
```

```
int x=0,y=0;
```

```
g.drawImage (img, x, y,Color.white, this);
```

```
}
```

```
}
```

◎ 使用缓冲技术装载图像

前面我们看到，直接将图像绘制到屏幕上会使人感到有一种抖动感。这是因为图像装载需要时间，装载一部分后就在屏

幕上绘制一部分，而不是整幅图像装载完后才绘制到屏幕上，这样就产生抖动。采用缓冲技术的思想是，首先在内存中创建一个

绘图区，将图像绘制到该内存的绘图区，当图像完全绘制到内存区后再绘制到屏幕上。

Java 具体的实现方法为：通过方法 `createImage (d.width, d.height)` 获得一个 Image 的对象（即内存的绘图区），

然后通过该对象的 `getGraphics ()` 方法获取绘图环境，通过该对象的绘图环境的方法

`drawImage ()` 绘制一幅图（将图像先绘

制到内存中），这样就创建了一个在内存中的图像即完成了图像在内存中的绘制。而后，通过屏幕的绘图环境 `g` 的方法

`drawImage ()` 将创建的图像对象（即内存中的图像）绘制到屏幕上。因为图像数据都事先 装载在内存中，所以整副图像几乎同

时显现出来，这就是图像绘制的缓冲技术。下面我们举例实现缓冲技术。

★ 实现缓冲技术具体例子演示

```
import java.awt.*;
import java.applet.Applet;
import java.net.*;

class myimage extends Applet implements Runnable
{
    Image img,buf;
    Dimension d=null;
    Graphics bufg;
    int x=(),y=();
    boolean loaded=false;
    Thread mythread;
    public void init()
    {
        img=getImage(getCodeBase(), "zy1.jpg" );

        mythread=new Thread(this);
        resize(640,480);
    }
    public void start()
    {
        mythread.start();
    }
    public void update(Graphics g)
    {
        if(d==null)
        {
            d=this.getSize();
            buf=createImage(d.getWidth(),d.getHeight());
            bufg=buf.getGraphics();
        }
        loaded=buf.drawImage(img,x,y,Color.white,this);
        if(!loaded)
            g.drawImage(buf,x,y,Color.white,this);
    }
    public void paint(Graphics g)
    {
        loaded=g.drawImage(buf,x,y,Color.white,this);
    }
    public void run()
    {
        while(!loaded)
        {
```

```

rePaint();
try
{
Thread.sleep(10);
}
catch(Exception e)
{
System.out.println(e);
}
}
}
}
}

```

程序运行后，整幅图像几乎会在同一时刻显现出来，消除了图像的抖动，这就是缓冲技术的优点。

11.5 图形与文本

Java 语言的内置类库对多媒体技术的支持能力相当强，尤其是对文本、图形、图像、声音等媒体的处理与极其方便而又丰富的接口。更令人兴奋不已的是，综合运用这些媒体所编制出来的一个个 Java 小应用程序(Applet) 冰冰的静态的 Web 主页(Homepage)上居然展现出一番热热闹闹的动态的新景观，这便是著名的 Java 动画。在它们就将进入到 Java 多姿多彩的多媒体世界中去。

11.5.1 图形的绘制

Java 语言的类库中提供了丰富的绘图方法(method)，其中大部分对图形、文本、图像(image)的操作方法都定义在 Graphics 类中。我们已经知道，Graphics 类又是 java.awt 程序包的一部分，因此，每当我们要进行图形、文本、图像的绘制时，不要忘了在 Java 源文件的头部先写上：import java.awt.Graphics; 在这里要特别指出的是，当我们想要在屏幕上绘制图形、文本及图像的绘制操作，就可以在屏幕上看到所显示的结果。

1. 图形坐标系

为了将某一图形在屏幕上绘制出来，我们首先要碰到的问题也许就是“画在哪个位置”，为了解决这个问题，我们引入一个精确的图形坐标系来将该图形定位。与大多数其它计算机图形系统所采用的二维坐标系一样，Java 的坐标系以屏幕的左上角为原点，水平向右为 X 轴的正方向，竖直向下为 Y 轴的正方向，每个坐标点的值表示屏幕上的一个象素点的位置，所有坐标点的值都取整数。图 4—1 表示用此图形坐标系统在屏幕上绘制一个矩形。

2. 画线

在 Java 的 Graphics 类中提供画线功能的是 drawLine() 方法，其调用格式如下：drawLine(int x1,int y1,int x2,int y2); 需要设置四个参数，其中 x1,y1 表示线段的一个坐标点，x2,y2 表示线段的另一个坐标点。如下面这段程序画出的显示结果如图 4—2 所示。

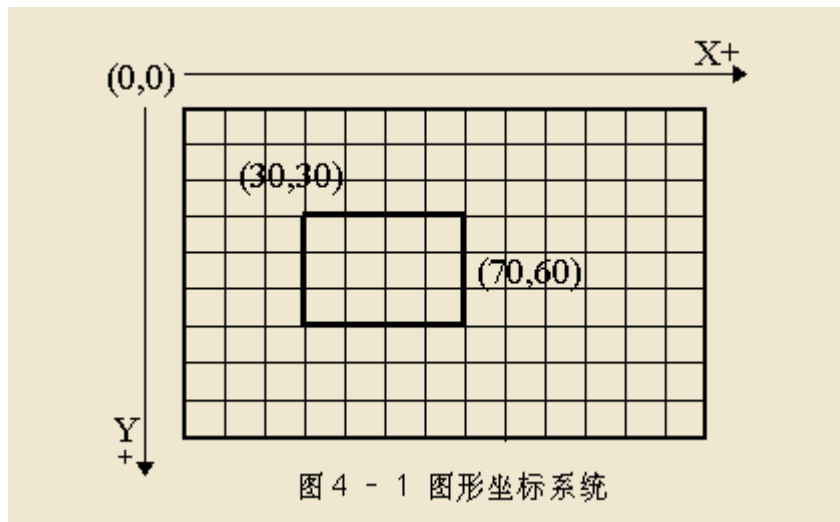


图 4—1 图形坐标系

```

1: import java.awt.Graphics;
2: public class Lines extends java.applet.Applet{
3: public void paint(Graphics g){
4: g.drawLine(30,30,70,70);
5: g.drawLine(60,50,60,50);
6: }
7: }

```

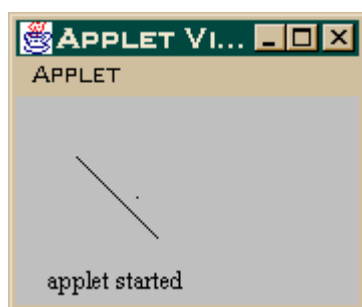


图 4—2 一条线段与一个点

由于 `Graphics` 类不专门提供画点的方法，所以程序中第 5 行将线段的两个点的坐标均设为(60,50)，因而就画了一个点。

3. 矩形

`Graphics` 类中提供了三种类型的矩形，它们分别是普通矩形、圆角矩形和立体矩形。而每一种矩形都提供了的方法，一种是仅画出矩形的边框；另一种是不仅画出 边框，并且还用相同的颜色将整个矩形区域填满。

(1)普通矩形 画普通矩形需调用 `drawRect()`或 `fillRect()`方法，它们的调用格式如下：

`drawRect(int x, int y, int width, int height)` //边框型风格

`fillRect(int x, int y, int width, int height)` //填充型风格

其中头两个参数分别表示矩形左上角的 `x` 坐标和 `y` 坐标，后两个参数分别表示矩形的宽度和高度。如下面画出两个矩形，其显示结果如图 4—2 所示。

```

public void paint(Graphics g){
g.drawRect(40,20,60,40);
g.fillRect(120,20,60,40); }

```

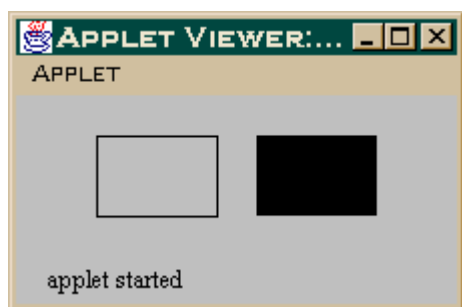


图 4-3 普通矩形的例子

(2)圆角矩形 圆角矩形，也就是矩形的四个顶角呈圆弧状，每个圆弧其实是由四分之一的椭圆弧所构成。画圆角矩形的两个方法的调用格式如下：

`drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)` `fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)`

我们可以看出，它们除了具有和普通矩形含义相同的前四个参数外，还多了两个用来描述圆角性质的参数。`arcWidth` 代表了圆角弧的横向直径；`arcHeight` 代表了圆角弧的纵向直径。例如如图 4-4 中左边一个圆角矩形所设的圆角参数为(`arcWidth=80`, `arcHeight=30`)，其效果就相当于该圆角弧存在于一个长 40 宽 15 的小矩形中；而右边一个圆角矩形所设的圆角参数为(`arcWidth=100`, `arcHeight=60`)，并且和整个圆角矩形的 `width` 和 `height` 参数值相等，因而该圆角矩形实际变成

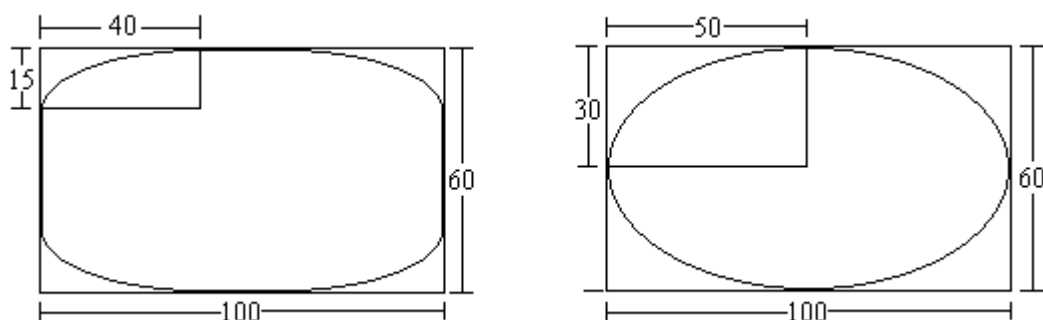


图 4-4 圆角矩形的参数设置

下面的 `paint()` 方法画出三个圆角矩形，显示效果如图 4-5 所示。我们不难发现，随着 `arcWidth` 和 `arcHeight` 参数的增大，矩形的圆角就越圆。

```
public void paint(Graphics g){
    g.drawRoundRect(20,20,80,60,20,20);
    g.fillRoundRect(120,20,80,60,40,30);
    g.drawRoundRect(220,20,80,60,60,40);}

```

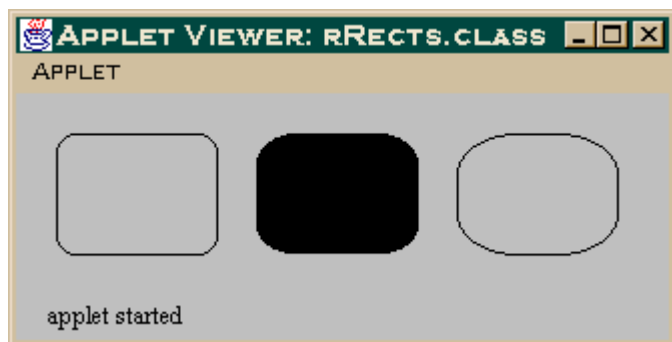


图 4-5 圆角矩形的例子

(3)立体矩形 立体矩形也可以说是三维矩形。其实，Java 中的立体矩形并非真正的三维图形，而仅仅是在矩形上加一点阴影，使矩形看上去相对表平面好象有凸出或凹下的效果，其调用方法的格式如下：

```
draw3DRect(int x, int y, int width, int height, boolean raised)
```

```
fill3DRect(int x, int y, int width, int height, boolean raised)
```

这两个方法中的前四个参数与 `drawRect()` 方法中所用的参数含义是一样的，第五个参数 `raised` 便是定义该矩形有凸出（值为 `true`）还是凹下（值为 `false`）的效果。例如，下面的 `paint()` 方法中，分别画了一个凸出的和一个凹下的，其显示效果如图 4-6 所示。确实，由于 Java 立体矩形中的阴影实在太薄，立体效果当然也就不太明显，图 4-6 是一个放大的凹角形状。

```
public void paint(Graphics g){  
g.draw3DRect(20,20,80,60,true);  
g.fill3DRect(120,20,80,60,false); }
```

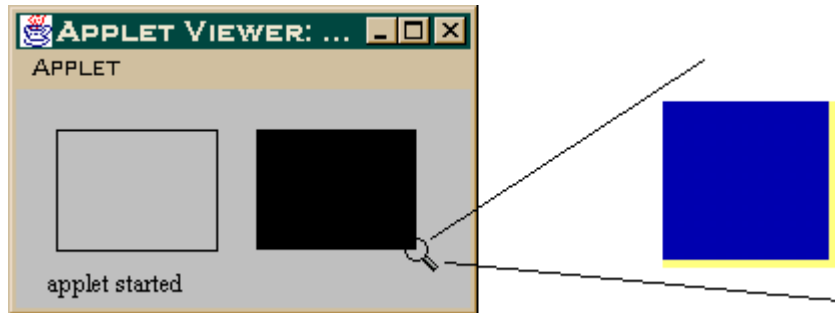


图 4-6 立体矩形的例子

4. 多边形

多边形的画法通常是给出一组坐标点，再用直线段将这些点依次连接起来。`Graphics` 类中也提供两个方法，一个是边框型 `drawPolygon()` 方法，另一个是填充型 `fillPolygon()` 方法，并且每一种方法都有两种不同的参数类型。其调用格式为：

```
drawPolygon(int xPoints[],int yPoints[],int nPoints)
```

```
fillPolygon(int xPoints[],int yPoints[],int nPoints)
```

其中 `xPoints` 参数是一个整数数组，用以存放多边形坐标点的 X 坐标值，`yPoints` 参数存放相应的一组 Y 坐标值，`nPoints` 则表示共有几个坐标点。如下面的 `paint()` 方法分别画了一个边框型和一个填充型的多边形，其显示效果如图 4-7 所示。

```
public void paint(Graphics g){  
int Poly1_x[]={30,63,115,72,67};  
int Poly1_y[]={40,20,95,74,106};  
int Poly1_pts=Poly1_x.length;  
int Poly2_x[]={180,213,265,222,217};  
int Poly2_y[]={40,20,95,74,106};  
int Poly2_pts=Poly2_x.length;  
g.drawPolygon(Poly1_x,Poly1_y, Poly1_pts);  
g.fillPolygon(Poly2_x,Poly2_y, Poly2_pts); }
```

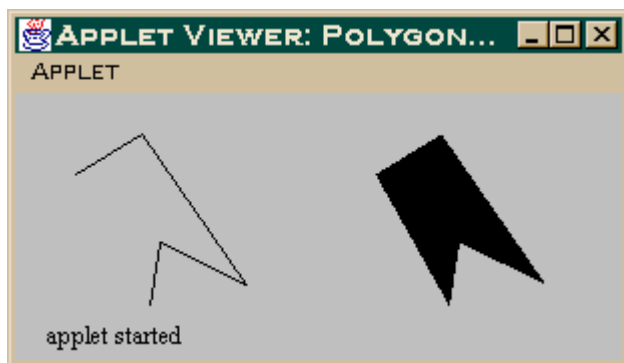


图 4—7 多边形的例子

可以看出，边框型多边形并不自动关闭多边形的最后一条边，而仅是一段开放的折线。所以，若想画封闭形，不要忘了在数组的尾部再添上一个起始点的坐标。上述两个画多边形方法的第二种参数传递形式为：

```
drawPolygon(Polygon p)
```

```
fillPolygon(Polygon p)
```

其中 Polygon 是定义在 java.awt 中的一个类，它的构造方法也有两种不同的参数传递形式，一种与 drawPolygon 第一种调用格式一样：

```
Polygon(int xPoints[],int yPoints[],int nPoints)
```

另一种调用格式则是创建一个空的多边形（无参数）： Polygon()

那为什么要另外创建 Polygon 对象？生成一个空的 Polygon 对象又有何用呢？原来，Polygon 类中提供了一系列方法，可以方便的进行与多边形相关的操作，象其中的 addPoint()方法可将多边形的坐标点动态地增加到 Polygon 对象中。下面列出的 paint()方法所执行的结果与图 4—7 所示的结果是一样的。

```
public void paint(Graphics g){
int Poly1_x[]={30,63,115,72,67};
int Poly1_y[]={40,20,95,74,106};
int Poly1_pts=Poly1_x.length;
Polygon poly1= new Polygon(Poly1_x,Poly1_y, Poly1_pts);
Polygon poly2= new Polygon();
poly2.addPoint(180,40);
poly2.addPoint(213,20);
poly2.addPoint(265,95);
poly2.addPoint(222,74);
poly2.addPoint(217,106);
g.drawPolygon(poly1);
g.fillPolygon(poly2); }
```

5. 椭圆

在 Java 中绘制椭圆的方法是给出该椭圆的外接矩形作为参数，其调用格式与画普通矩形的方法相似：

```
drawOval(int x, int y, int width, int height) //边框型风格
```

```
fillOval(int x, int y, int width, int height) //填充型风格
```

这里要特别注意：x 和 y 不是椭圆的圆心坐标，而是该椭圆外接矩形的左上角。因此，画椭圆时，把它的外接矩形将有助于在坐标系统中定位。另外，Graphics 类不专门 提供画圆的方法，而只需将 width 与 height 参数置为相等即可。例如，下面的 paint()方法，画出一个圆和一个用颜色填充的椭圆，其显示效果如图 4—8 所示。

```
public void paint(Graphics g){
g.drawOval(30,20,60,60);
g.fillOval(130,20,80,60); }
```

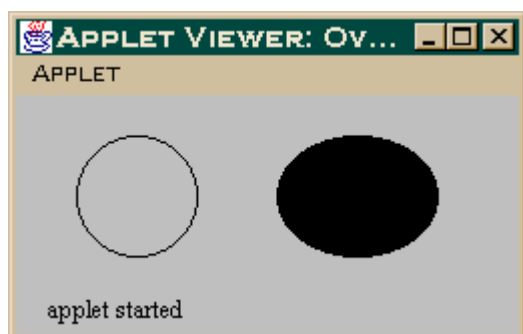


图 4-8 椭圆的例子

6. 画弧

弧是椭圆的一部分，因而画弧的方法就相当于先画一个椭圆，而后取该椭圆中所需要的一部分。它们的调用方法如下：

`drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)` //边框型风格

`fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)` //填充型风格

其中前四个参数的含义与画椭圆一样，因此也必须用矩形的观点来确定弧在坐标系统中的位置。后两个参数定义椭圆的一部分：`startAngle` 参数表示该弧从什么角度开始，`arcAngle` 参数表示从 `startAngle` 开始转了多少度。弧度坐标系所示，水平向右表示 0° ，逆时针方向为正角度值，顺时针方向为负角度值。

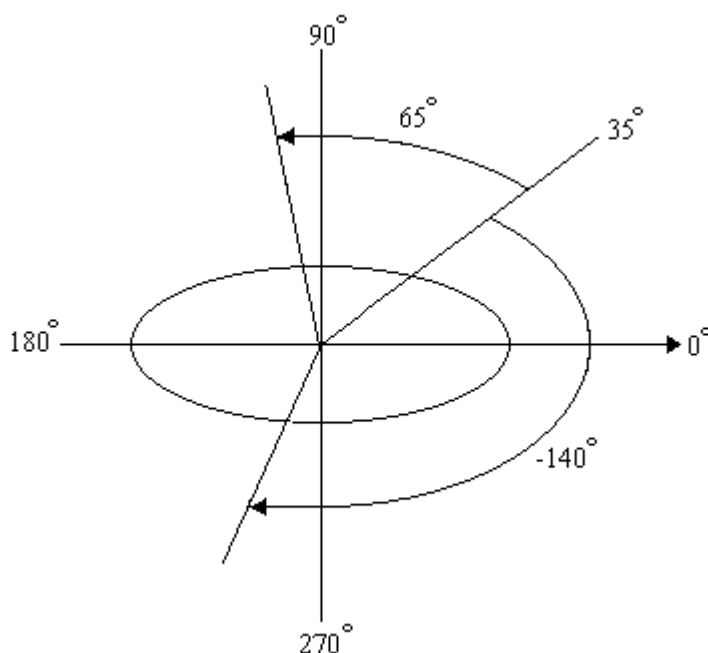


图 4-9 弧度坐标系

如果 `startAngle` 和 `arcAngle` 中有任一值大于 360° 的话，都会被转换为 0 到 360° 之间的数值。因此若要画 370° 的弧，`arcAngle` 需设为 360 的整数倍，若设为 370 度则相当于只画了 10 度。另外 `fillArc()` 方法的效果并不是填充弧的连线所围的区域，而是填充弧的两端点与圆心连线所围的扇形区域，象一个饼图。下面的 `paint()` 方法画了图 4-10 所示的弧，其显示效果如图 4-10 所示。

```
public void paint(Graphics g){
    g.drawArc(10,20,100,60,35,65);
    g.drawArc(110,20,100,60,35,-140);
    g.fillArc(210,20,100,60,35,65);
    g.fillArc(310,20,100,60,35,-140); }
```

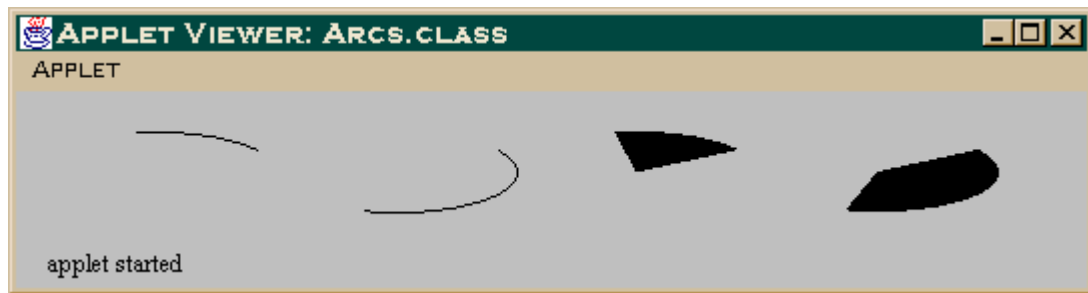


图 4—10 弧的例子

7. 复制与清除图形

当我们需要在屏幕上重复绘制一些相同的图形时，也可采用 `Graphics` 类中的 `copyArea()` 方法，它可将屏幕域里的内容复制到屏幕的另一区域。其调用格式如下：

```
copyArea(int x, int y, int width, int height, int dx, int dy)
```

前四个参数我们应该是相当熟悉了，它定义了要被复制的屏幕的矩形区域。最后两个参数则表示新区域与的偏移距离：若 `dx, dy` 为正值，则表示新区域相对于原区域的右方及下方所偏移的像素值；反之，它们取负值对左方及上方的偏移量。若要清除屏幕的某一矩形区域所画的内容，就要选用 `clearRect()` 方法，它用当前的背景色清除整个矩形区域。其调用格式为：

```
clearRect(int x, int y, int width, int height)
```

可以看出这四个参数定义了所要清除的矩形区域。例如，想要清除整个 applet 区域，可先调用 `Applet` 类的 `getSize()` 方法得到整个 applet 的宽度和高度（该方法没有参数，返回值是一个 `Dimension` 对象，该对象具有 `width` 和 `height` 属性），然后调用 `clearRect()` 方法就可以了：

```
g.clearRect(0, 0, size().width, size().height);
```

11.5.2 文本与字体

`Graphics` 类也提供了在屏幕上显示文本的方法，但若要使文本的显示更具特色，让它满足某种字体、某种大小、某种颜色的要求，就需要用字体类 `Font` 来定义。

1. 设置文本信息

当我们想要在屏幕上输出文本信息时，首先要确定的就是采用何种字体，例如中文的“宋体”、“楷体”，或是英文的“TimesRoman”体、“Courier”体等等，接着再决定该字体输出时采用哪种风格，是斜体型还是粗体型等等，最后再决定字体的大小尺寸。所有这些都由 `Font` 类来定义，我们不难猜出其构造方法的调用格式：

```
Font(String name, int style, int size)
```

不错，它的三个参数就是我们先前所说的字体名、字体风格和尺寸大小。并且 `Font` 类中已定义了类变量来表示不同的字体风格，如 `Font.BOLD`（表示粗体）、`Font.ITALIC`（表示斜体）、`Font.PLAIN`（表示普通体）。由于它们被定义为整数常量，所以可以进行相加运算来生成复合 style，例如想让 style 即是粗体又是斜体，可以这样写：

```
Font fn = new Font("TimesRoman", Font.BOLD+Font.ITALIC, 28);
```

虽然我们定义了所需的字体，但其显示结果有时也并非如愿。因为运行该 applet 的客户端系统有可能并未安装我们指定的字体，这时 Java 就会以缺省字体来替代它。因此，不妨先查看一下客户端系统目前究竟支持哪些字体，这就要用到 `Graphics` 类中的 `getFontList()` 方法，它返回系统目前可用的字体列表，然后就可决定到底选用哪种字体。例如：

```
Toolkit systk = Toolkit.getDefaultToolkit();
```

```
String fonts = systk.getFontList();
```

2. 显示文本

创建了 `Font` 对象以后，我们就可以利用 `Graphics` 类中提供的 `drawString()`、`drawChars()` 等方法来显示字符串。首先还要用 `setFont()` 方法，将所创建的 `Font` 对象设为当前所用的字体。下面就是 `Graphics` 类中这三个方法的调用格式：

```
setFont(Font font);
```

```
drawString(String str, int x, int y)
```

```
drawChars(char data[], int offset, int length, int x, int y)
```

其中 `setFont()` 方法的参数就是一个创建好的 `Font` 对象,表明系统当前选用哪个 `Font` 对象所定义的字体信息。方法中的 `str` 即是要显示的字符串, `x,y` 指明字符串显示的起始位置坐标, 具体的说, `x` 表示第一个字符的左边, `y` 表示第一个字符串的基线 (baseline, 见图 4-12) 位置坐标。因此, 这里的坐标并不是通常意义上的矩形区域的左上角。方法则是用来显示多个字符的, 也就是从给定的字符数组中抽取连续的一部分显示在屏幕上。其中 `data` 参数就是字符数组, `offset` 表示从第几个字符位置开始显示, `length` 表示共显示几个字符, `x` 与 `y` 参数的含义与 `drawString` 代表显示在屏幕上的起始位置。如下面的程序显示了一些不同的文本字体, 其显示结果如图 4-11 所示。

```
import java.awt.Graphics;
import java.awt.Font;
public class Fonts extends java.applet.Applet{
    public void paint(Graphics g){
        Font ftp20 = new Font("TimesRoman",Font.PLAIN,20);
        Font fai15 = new Font("Arial",Font.ITALIC,15);
        Font fcb24 = new Font("Courier",Font.BOLD,24);
        Font fsib30 = new Font("宋体",Font.ITALIC+Font.BOLD,30);
        g.setFont(ftp20);
        g.drawString("Font name TimesRoman , style plain , size 20",10,20);
        g.setFont(fai15);
        g.drawString("Font name Arial , style italic , size 15",10,50);
        g.setFont(fcb24);
        g.drawString("Font name Courier , style bold , size 24",10,80);
        g.setFont(fsib30);
        g.drawString("字体名 宋体, 风格 斜体+粗体, 尺寸 30",10,120);
    }
}
```

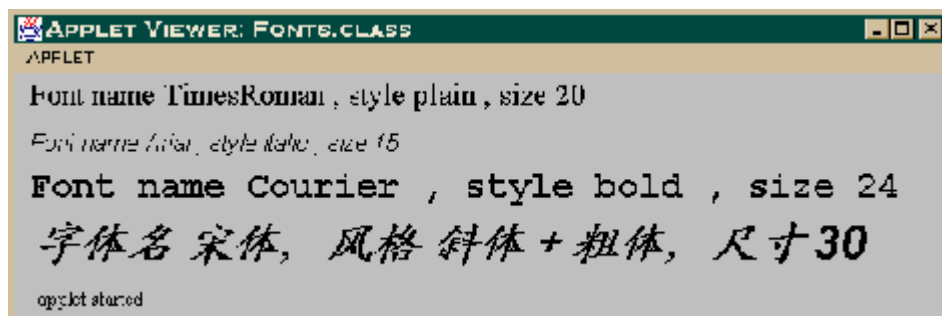


图 4-11 各种字体的例子

3. 获取字体信息

(1)获取基本信息 如果不清楚系统当前所用的字体信息, 可以先调用 `Graphics` 类中的 `getFont()` 方法, 该方法返回系统当前所用的 `Font` 对象, 然后就可以调用 `Font` 类中提供的几个方法来获取该字体的基本信息。表 4-1 列出了 `Font` 类中的这些方法。表 4-1 `Font` 类提供的一些主要方法

(2)获取详细信息 有时候, 我们为了在屏幕上更精确地定位文本, 还需要了解所选字体的更详细的信息, 例如到底有多高, 有多宽, 两行字符串的间隙有多少等等。这时, 我们需要用到一个新的类 `FontMetrics` 来提供这些信息。可以调用 `Graphics` 类中的 `getFontMetrics()` 方法来获取关于当前字体的 `FontMetrics` 对象 (该方法也无参数)。然后, 可以调用表 4-2 所示的 `FontMetrics` 类中所提供的方法来获取更详细的字体信息。图 4-12 中给出了字体中关于 `Ascent` 和 `Leading` 等概念的示意图。

11.5.3 颜色的设置

现在, 让我们改变一下总是在灰色背景上用黑色绘图以及显示文本的习惯, 而给我们的 applet 增添一些五彩斑斓的颜色。与设置字体信息相似, 要设置新的颜色, 必须先创建 `Color` 对象, 然后再调用 `Graphics` 类中设置颜色的方法来将

对象设为当前所用的绘图颜色。

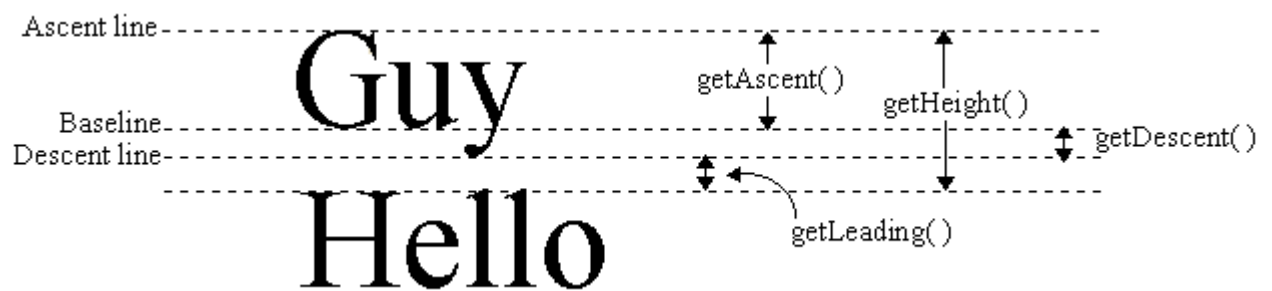


图 4-12 字体

1. 创建 Color 类

Java 中每一种颜色都看成是由红 (R)、绿 (G)、蓝 (B) 三原色组合而成的。因此 Color 类的构造方法采用

`Color(int r, int g, int b)`

其中每个参数的值都在 0 到 255 之间，数值越大就表明这种颜色的成份越重。例如(0,0,0)代表黑色，(255,0,0)代表红色。当然最终在屏幕上是否能显示所定义的颜色还取决于客户端系统的调色板所支持的颜色种类的多少。若客户端并不支持当前所定义的颜色值，就会在调色板中挑选最接近的颜色来代替。Color 类中还定义了一些标准颜色，存储在类变量中，使的这些标准颜色的引用显得更为方便。这些类变量如表 4-3 所示。

2. 设置当前颜色

为了能使用刚才生成好的 Color 对象来显示文本及绘制图形，还需调用 Graphics 类中的 `setColor()` 方法把该系统当前所用的绘画颜色，其调用格式为：

`setColor(Color c)`

例如，想要用蓝色来显示文本，最简单的办法是直接引用标准色的类变量：

`setColor(Color.blue);`

另外，Java 还提供了设置整个 applet 的背景和前景的方法，它们分别是：`setBackground()` 方法和 `setForeground()` 方法。它们都被定义在 `java.awt.Component` 类中，因此该方法能被其子类（包括 Applet 类及 Applet 类的子类）自动继承。调用格式与 `setColor()` 方法一样：

`setBackground(Color c)`

`setForeground(Color c)`

其中 `setForeground()` 方法将影响到 applet 中所有已经用其它任何颜色所绘制的图形及显示的文本，把它们都设置成该方法所定义的前景颜色，而不需用该颜色重新一一绘制。有“set”必有相应的“get”，Java 中还提供了 `getColor()` 方法（在 `Component` 类中）、`getBackground()` 方法和 `getForeground()` 方法（在 `Component` 类中）来分别获取当前的绘图颜色、applet 的背景颜色、applet 的前景颜色。下面，我们写一段程序来显示一排用随机定义的颜色所填充的小方块，它们的显示效果如图 4-13 所示。

```
import java.awt.Graphics;
import java.awt.Color;
public class Colors extends java.applet.Applet{
    public void paint(Graphics g){
        int red,green,blue;
        for (int i=10;i<400;i+=40){
            red=(int)Math.floor(Math.random()*256); "
            green=(int)Math.floor(Math.random()*256); "
            blue=(int)Math.floor(Math.random()*256); "
            g.setColor(new Color(red,green,blue));
            g.fillRect(i,20,30,30);
        }
    }
}
```




图 4—13 使用绘图颜色的例子

第十二讲:Applet 图形用户设计

Applet 图形用户界面设计主要依赖于 Java 的图形界面设计工具集 AWT。AWT 是 Java 提供的抽象界面设计工具包，用

该工具包可以设计常用的图形用户界面。该工具包能够在不同结构的机器上和不同的平台上实现用户界面的一致性，这样便于用户使用，减少学习的工作量。

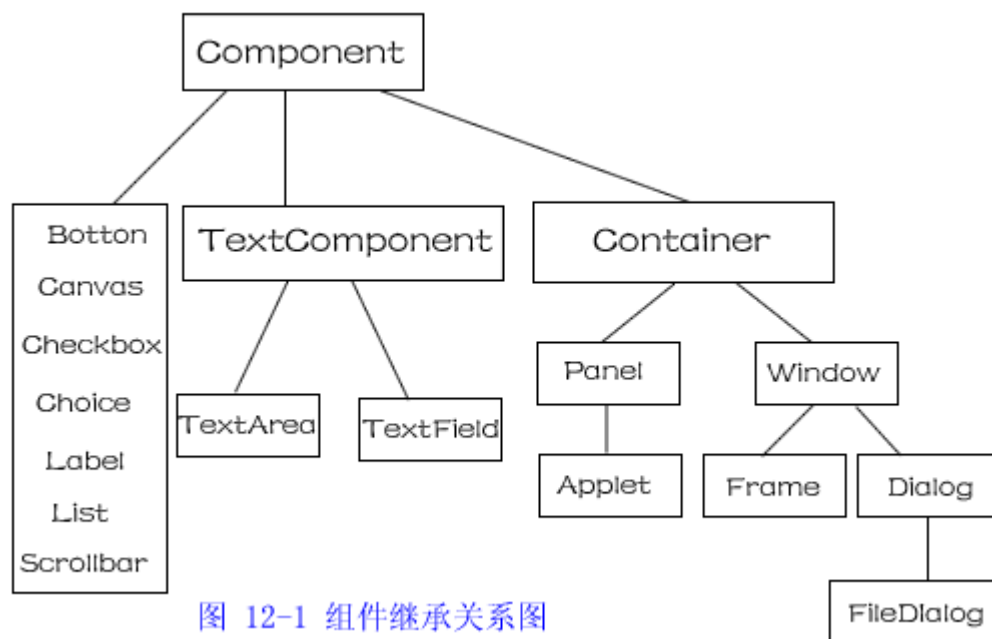
12.1Java 图形界面设计工具集 AWT

AWT 是一个类包，用该类包进行界面设计需要学习和熟练使用类，要知道 AWT 类继承的层次关系和组织结构。

© AWT 工具包的类的继承关系

AWT 继续关系图 12—1 所示。Component 类是所有组件的父类，在该类中实现了一些基本的方法。Component 类直接继承

Object，属于 java.awt 包。



◎ AWT 界面的组成

上面的类对应于图形界面上的组件，因此我们也将对应的类称为组件。AWT 的界面是由组件构成，所有的组件都是继承

Component 组件而来。组件有两种：一种是容器组件，一种是基本组件。

(1) 基本组件：按钮 (Button)，标签 (Label)，文本区 (TextArea)，文本框 (TextField)，作图区 (Canvas)，滑动条 (Scrollbar)，列表 (List)，核选框 (Checkbox)，选择框 (Choice)，菜单 (Menu)，

菜单条 (MenuBar)。通过这些组件来与用户进行交互。

(2) 容器组件：窗口组件 Window、Dialog、FileDialog、Frame 和面板 Panel。容器组件可以容纳 AWT 组件，同时面

板组件 Panel 还可以包含在 AWT 容器中。容器的最高父类为 Container 类。

上面介绍了组件和容器，我们将组件放入容器便能够构成基本的用户界面。基本组件在容器中是有布局的。如果没有指

定组件在该容器中的布局方式，AWT 会为该容器提供一个缺省的布局方式。

(3) AWT 容器的布局方式：BorderLayout、GridBagLayout、GridLayout、CardLayout、FlowLayout。

通过上面的组件、容器及布局方式，可以根据要求确定界面的层次结构。下面将具体地介绍组件的应用。所有的组件都

是从最高父类 Component 继承而来，Component 类中定义了组件的基本方法，组件使用这些继承而来的方法可以进行一些操作

，如将一个组件加入到容器中，设置组件的布局方式等。下面我们讨论 Component 类中与界面设计有关的方法。

12.2 图形界面基本组件

图形界面的基本组件，包括按钮（Button）、标签（Label）、核选框（Checkbox）、文本区（TextArea）、文本框（TextField）、滑动条（Scrollbar）、圆板（Canvas）、列表框（List）、下拉列表（Choice）、容器等，下面逐一进行介绍。

12.2.1 按钮 Button

每一个按钮上有一个按钮名作标志。操作按钮可以激活相应的事件。下面举例介绍按钮的创建。

```
import java.applet.Applet;
import java.awt.*;
class myfirstbutton extends Applet
{
    Button button1,button2,button3,button4;
    public void init()
    {
        setLayout(new GridLayout(2,2)),
        setBackground(Color. white);
        button1=new Button("Ok");
        button2=new Button("Cancel");
        button3=new Button("Help");
        button4=new Button("disable button");
        add(button1);
        add(button2);
        add(button3);
        button4.disable();
        add(button4);
    }
}
```

下面对程序作几点说明：

(1)上面我们用到了 add（）方法来将生成的 button1， button2， button3， button4 按钮显示在 Applet 区域。自定义

义类 Myfirstbutton 类继承 Applet 类，Applet 类继承了 Panel 类，Panel 类有 Add（）方法可以将组件加入容器中。因此，

Myfirstbutton 也是一个容器，继承了父类的 add()方法，通过该方法来将组件加入该容器中，同时使用继承而来的方法

setLayout（）来设置 myfirstbutton 容器的布局。

（2）程序中关于容器的布局方式将在后面具体介绍。

12.2.2 标签 Label

标签通常用来显示一些提示信息，下面举例说明。

```
import java.applet.Applet;  
import java.awt.*;
```

```
class firstlabel extends Applet  
{  
    Label label1,label2,labe13,label4;  
    public void init()  
    {  
        label1=new Label("label is usually used to display information");  
        label2=new Label("label2");  
        1abel3=new Label("labe13");  
        label4=new Labe1("label4");  
        add(label1);  
        add(label2);  
        add(label3);  
        add(label4);  
    }  
}
```

HTML 文件为:

```
<html>  
<head>  
<title>firstlabel</title>  
</head>  
<body>  
<hr>  
<applet  
code=firstlabel  
width=320  
height=240>  
</applet>  
<hr>  
</body>  
</html>
```

12.2.3 核选框 Checkbox

◎一般核选框 Checkbox

核选框是 Checkbox 类的实例对象，创建核选框需要通过 Checkbox 类提供的构造方法。Checkbox 类提供的构造方法有如下三种：

(1) public Checkbox();

(2) public Checkbox (String label);

(3) `public Checkbox (String label, CheckboxGroup group, boolean state)`。

(4) `Checkbox ()` 构造方法创建一个没有标记的核选框；

(5) `Checkbox (String label)` 构造方法用来创建一个用 `label` 字符串名字的核选框；

(6) `Checkbox (String label, CheckboxGroup group, boolean state)` 构造方法用来创建一个属于核选框组的核选框。一个核选框组中的核选框只能有一个核选框处于选中的状态。关于核选框组的作用，将在下面介绍。我们首先创建不属于任何核选框组的核选框。

下面是核选框使用的例子。

```
import java.applet.Applet;
import java.awt.*;
class checkbox extends Applet
{
    Checkbox checkbox1,checkbox2,checkbox3;
    public void init()
    {
        setFont(new Font("TimesRoman",Font. BOLD, 20));
        setBackground(Color. white);
        checkbox1=new Checkbox();
        checkbox2=new Checkbox("the second checkbox");
        checkbox3=new Checkbox("the third checkbox");
        add(checkbox1);
        add(checkbox2);
        add(checkbox3);
        checkbox1 .setLabel("the name is created by method setLabel");
        checkbox1 .setState(true);
    }
    public void paint(Graphics g)
    {
        g.setColor(Color.blue);
        g.drawString("these are samples of creating checked",20, 150);
    }
}
```

下面是 HTML 文档中的 Applet 标签。

```
<Applet
    code=checkbox
width=500
height=200>
</applet>
```

`Checkbox` 类还提供如下方法来设置和读取核选框：

```
public void setLabel(String label)
```

```
public void setState(boolean state)
```

```
public String getLabel()
```

```
public boolean getState()
```

我们可以用方法 `public void setLabel (String label)` 在程序中重新设置创建的 `Checkbox` 的名字，同时我们也

可以用方法 `public void setstate (boolean state)` 来重新设置 `Checkbox` 的状态（选中与没有被选中）。当核选框被创

建并显示在屏幕上时，我们可以用鼠标来操作核选框使其处于选中状态或者取消选中状态。

我们可以通过 `Checkbox` 的方法

`public String getLabel ()` 来得到该核选框对象的名字，通过 `public boolean getstate ()` 方法来取得核选框对象的状态（选中或者没有选中）。当鼠标操作核选框时，会产生相应的事件，该事件会激活相应的方法来处理该事件，程序就会响应

我们的操作。

我们的操作。

12.2.4 文本区 (TextArea)创建

文本区可以接收用户的文本输入，同时对文本区的操作将会产生相应的事件，通过 `AWT` 的时间处理机制可以响应对文

本区的操作。要操作文本区，首先要创建文本区对象，文本区类 `TextArea` 犯。为我们提供了如下的构造方法来创建文本区对

象：

```
public TextArea ();
```

```
public TextArea (int rows, nit cols);
```

```
public TextArea (String text);
```

```
public TextArea (String text, int rows, int cols);
```

通过 `TextArea()` 构造方法可以创建一个不含有初始文本的空文本区；用构造方法 `TextArea (int rows, int`

`cols)` 可以创建一个在文本区只显示指定行数和指定列数的文本的文本区，该文本区可以有

多行，且每一行同样可以输入多个

字符，该方法就是用来指定文本区在 `Applet` 区域所占的空间；用构造方法 `TextArea(string`

`text, int rows, int`

`cols)` 可以创建一个指定显示行列数目且含有初始文本的文本区。

知道了创建文本区的构造方法，我们便可以进行文本区的创建，下面举例说明：

```
import java.awt. *;
```

```
import j ava. applet.Applet;
```

```
class mytextarea extends Applet
```

```
{
```

```
TextArea text1,text2,text3;
```

```
Public void init()
```

```
{
```

```
text1=new TextArea(" text1 has this text at first,\n \
```

```
text2 takes 10 cows 20 rols,\n\
```

```
but no text at first, \n\
```

```

text3 takes 10 cows 20 ro1s and \n\
has the text--\"text3\".");
text2=new TextArea(10,20);
text3=new TextArea("text3", 10,20),
text1l.setFont(new Font(' 'TimesRoman", Font. ITALIC, 15 ) );
text1. setForeground(Color. blue );
text2.setFont(new Font("TimesRoman", Font. ITALIC, 15 ) ),
text3.setFont(new Font(' 'Times Roman",Font. ITALIC, 15 ));
add(text1);
add(text2);
add(text3);
}
}

```

下面是嵌入 HTML 的 Applet 标签。

```

<applet
code=mytextarea
width=500
height=300
</applet>

```

◎ 方法 `appendText(String str)`

方法地 `appendText (String sir)` 用来追加文本区的文本；方法 `getColunns ()` 用来取得文本区对象在 applet 区域所占的列数；方法 `insertText (String sir, int pos)` 可以在指定的位置（文本行的第几列）插入文本；方法 `replaceText (String sir,int start, int end)` 可以用指定的文本替代文本区中文本的指定部分。参数 `Start` 用来指定文本区中要被取代的文本的起始点（文本行的第几列）；参数 `end` 用来指定要被取代的文本的终止点（文本行的第几列）

◎ 方法 `appendText(String str)` 应用举例

```

import java.awt.*;
import java.applet.Applet;
class textareause extends Applet implements Runnable
{
    TextArea mytextarea;
    Thread mythread,
    public void init()
    {
        mythread=new Thread(this);
        setForeground (Color. red);
        setB ackground(Color.white);
        mytextarea=new TextArea("this is the original text.",5,45),
    }
}

```

```

mytextarea.setFont(new Font("TimesRoman",Font. ITALIC,20));
mytextarea.setBackground(Color. orange );
mytextarea.setForeground (Color. b1ue);
add(mytextarea);
mytextarea. insertText' 'THIS IS INSERTED TEXT",4);
}
public void start()
{
mythread.start();
}
public void paint(Graphics g)
{
int cols,rows;
cols =mytextarea.getColumns();
rows=mytextarea. getRows();
g.drawString("Columns is "+cols+" ,"+"rows is "+rows,20,20());
}
public void run()
{
try
{
    Thread.sleep(5000);
}
catch(Exception e)
{}
mytextarea. replaceText("THIS IS THE REPLACED TEXT", 8, 10);
repaint();
}
}

```

下面是要嵌入 HTML 文档的 Applet 标签。

```

<applet
code=textareause
width=500
height=250>
</applet>

```

当打开嵌有上面 Applet 标签的 HTML 文档后，会发现插入文本 THIS IS INSERTEDTEXT 在文本区的第五列开始插入。

当延时 5 秒钟后，将用文本 THIS IS THE REPLACED TEXT 取代文本区中已有的文本行的第九列到第十一列的文本。上面介绍

了文本区的创建和使用，在文本区中我们可以输入编辑和文本。文本区对象也提供对文本的编辑方法，我们可以在程序中对文

本进行编辑。还有获取文本区对象特征的方法，同时文本区也是继承超类 Component 而来，有超类的方法。我们可以根据需要

选用这些方法来编辑文本，对文本进行处理，对文本区对象进行操作。在本地机上，我们可

以进行文件操作，将文本区的内容存于磁盘。下面我们介绍与文本区相似的文本框。

12.2.5 文本框（TextField）的创建

文本框与文本区功能相似，但文本框用法更灵活。例如，在文本框中用其它的字符显示来取代用键盘输入的

文本，使真实文本不可见，可以作为口令输入框。同时文本框没有滑动条，这样非常适合作为登录时用的输入框。下面介绍

文本框的创建和使用。

创建一个文本框可以用下面的构造方法：

- (1) `public TextField();`
- (2) `public TextField(int cols);`
- (3) `public TextField(String text);`
- (4) `public TextField(String text, int cols);`

用上面的构造方法创建对象跟文本区基本一样，下面我们举例说明文本框。

```
import java.awt. *;
import java.applet.Applet;
class textfield extends Applet
{
    TextField mytextfield1,mytextfield2,mytextfield3,mytexifield4;
    public void init()
    {
        setBackground(Color. white);
        setForeground(Color. blue);
        setFont(new Font(' 'TimesRoman",Font. ITALIC, 20) );
        mytextfield1=new TextField();
        mytextfield2=new TextField(10),
        mytextfield3=new TextFie1d("textfield3");
        mytextfield4=new TextFeld("textfield4", 10),
        add(mytexifield1);
        add(mytextfield2);
        add(mytextfield3);
        add(mytextfield4);
    }
    public void paint(Graphics g)
    {
        g.setColor(Color.red);
        g. draw String("mytextfield 1 has no text at first;", 20, 180),
        g. draw S tring("mytexifield2 takes 10 co1umns,has no text at first;", 20, 200);
        g. draw String("mytextfield 3 has the text- \"mytextfield3;", 20, 220);
        g. draw S tring ("mytextfield4 take s 10 columnns,, has the text- \"mytextfie1d4.", 20, 240);
    }
}
```

下面是 Applet 的标签:

```
<applet  
code=textfield  
width=600  
height==250>  
</applet>
```

12.2.6 滑动条 (Scrollbar) 的创建

在图形窗口中我们几乎都要用到滑动条, 通常用滑动条来显示屏幕外的文本或者图 像。滑动条是一个带有长度指

示器的长条, 滑动条由三部分组成: 滑块指示器、滑轨和左右两边的箭头。滑动条上有一个可以滑动的滑块指示器。滑动条都

有一个值, 且有最大值和最小值。滑块指示器所在的位置对应滑动条的一个值, 改变滑块指示器的位置, 滑动条的值也相应地

改变。滑块指示器只能够在最大值和最小值之间滑动。滑动条有竖直滑动条和水平滑动条, 它们的用法是一样的。对于水平滑

动条, 当滑块指示器滑动到最左边时, 滑动条的值为最小值; 当它滑动到最右边时, 滑动条的值为最大。对于竖直滑动条, 滑

块指示器在最上面时, 滑动条取最小值; 在最下面, 滑动条取最大值。滑块指示器的宽度

(visible 属性)、最大值、最小值都可以 进行设置。上面示意中的滑动条的宽度 (visible) 值为 64, 最小值 (linimum) 为 0, 最大值 (maximun) 为 255。

当点击滑轨时, 滑动条的值有较大的改变; 点击箭头时, 有较小的改变。改变的步长可以在程序中设置。下面我们介绍滑动条的创建。

创建滑动条对象的构造方法如下:

```
public Scrollbar();
```

```
public Scrollbar(int orientation);
```

```
public Scrollbar(int orientation, int value,int visible, int ndnimum, int maximum);
```

构造方法 Scrollbar (用来创建一个空的竖直滑动条。用该方法创建滑动条, 还需要使用滑动条对象封装的方法来

设置滑动条的属性才可用; 构造方法 Scrollbar (int orientation) 用来创建一个空的水平或者竖直滑动条; 构造方法

Scrollbar (int orientation, nit value, int visible, int minimum, int maximum)用来创建水平或者竖直的滑动条

, 在创建的时候, 可以为滑动条设置这些属性: 滑动条的值 (value), 滑动条的上的滑块长度值 (visible), 滑动条的最

小值 (minimum), 滑动条的最大值, 该方法可以创建一个完整的滑动条。下面我们来创建滑动条。举例说明如下:

```
import java.awt *;
```

```
import java.applet.App1et;
```

```
c1ass scrollbar extends Applet implements Runnable
```

```

{
Thread mythread;
Scrollbar scrollbar1, scrollbar2, scrollbar3, scrollbar4;
Label label1,label2,label3,label4
public void init()
{
label1=new Label("scrollbar1");
label2=new Label("scrollbar2");
label3=new Label("scrollbar3");
label4=new Label("scrollbar4"),
mythread=new Thread(this);
setBackground(Color. white);
setForeground(Color.blue);
setFont(new Font(' 'TimesRoman",Font. ITALIC, 20));
scrollbar1=new Scrollbar();
scrollbar2=new Scrollbar(SwingConstants.HORIZONTAL);
scrollbar3=new Scrollbar(SwingConstants.VERTICAL );
scrollbar4=new Scrollbar(SwingConstants.HORIZONTAL,0,10,100);
add(label1);
add(scrollbar1),
add(label2);
add(scrollbar2),
add(label3);
add(scrollbar3),
add(label4),
add(scrollbar4);
}
public void start()
{
mythread.start();
}
public void paint(Graphics g)
{
g.drawString("The value of scrollbar4 is" +scrollbar4. getValue(), 20, 150 ),
g.drawString("The jumping Page-step of scrollbar4 is" \
"+ scrollbar4. getPageIncrement(), 20, 180);
g.drawString("the jumping Line-step of scrollbar4 is" \
"+scrollbar4. getLineIncrement(), 20, 210 );
}
public void run()
{
while(true)
{
repaint();

```

```

    trs
    {
        Thread.sleep(500);
    }
    catch(Exception e)
    {}
    }
    }
    }
}

```

在上面的程序中，我们创建了四个标签（Label），四个滑动条（Scrollbar）。滑动条有一个当前值，当我们滑动滑动条时，滑动条的值也跟着增加或者减少。在程序中，用方法 `getValue()` 来得到条的值，用方法 `getPageIncrement()` 来得到当点击滑动条上的滑轨时滑动条值增加或者减少的步长（page-step），用方法 `getLineIncrement()` 来得到点击滑动条上的箭头时滑动条值增加或者减少的步长（line-step），并将这 H 个值显示于屏幕上。

12.2.7 列表框 List

列表框类（List）提供如下的构造方法来创建列表框：

```
public List ();
```

```
public List (int rows, boolean multipleSelections);
```

构造方法 `List ()` 可以创建一个空的列表框，用该方法创建的列表框的大小为四行高，

构造方法 `List (int rows, boolean multipleSelections)` 用来创建一个指定显示行数和是否能够多选 of 列表框

。当列表框中的项目多于列表框能够显示的行数时，列表框将会出现滑动条，通过滑动条滑动来显示被隐藏的项目。下面举例

说明列表框的创建。

```

import java.awt.*;
import java.applet.Applet;
class mylist extends Applet
{
    List mlist1,mlist2,
    public void init()
    {
        setBackground (Color. white);
        setFont(new Font("Times Roman",Font. ITALIC, 20 ));
        mlist1=new List();
        mlist2=new List(5,true);
        mlist1 .addItem("class one");
        mlist1 .addItem("class two"),
        mlist1 .addItem("class three");
        mlist1 .addItem("class four");
        mlist1 .addItem("class five");
        mlist1 .addItem("class six"),

```

```

Inlist2.addItem("one");
ndist2.addItem("two");
nili st. addItem(' 'three" );
mlist2.addItem("four");
nolist2.addItem("five");
nolist2.addItem("six"),
add(mlist1);
add(mlist2);
}
}

```

该程序运行需要的 Applet 标签为:

```

<applet
code=mylist
width=400
height=300>
</applet>

```

对于列表框的其它操作方法，如表 12-1 所示：

表 12-1 列表框 List 的常用操作方法

方法的调用格式	含 义
public void addItem(String item)	向列表框加入一项
public void addItem(String item, int index)	
public boolean allowsMultipleSelections()	判断是否可以多选
public int countItems()	获得列表框中的项数
public void delItem(int position)	删除指定项
public void delItems(int start, int end)	删除指定起止间的项
public void deselect(int index)	取消对指定项的选择
public String getItem(int index)	得到索引的对应项
public int getRows()	得到列表框在屏幕上所占的行数
public int getSelectedIndex()	得到总共选择的项数
public String getSelectedItem()	得到选择的项
public String[] getSelectedItems()	得到所选择的所有的项
public boolean isSelected(int index)	判断某一项是否被选中
public void makeVisible(int index)	使某一项处于可见状态
public void replaceItem(String newValue, int index)	用新的一项取代列表框中的指定项
public void select(int index)	选择指定的项
public void setMultipleSelections(boolean v)	设置可以选择多项

12.2.8 下拉列表 Choice

下拉列表的功能与列表框的功能相似，但下拉列表所占的空间小，结构紧凑。创建下拉列表的构造方法如下：

```
public Choice ( )
    下面举例介绍下拉列表的创建：
import java.applet.Applet;
import java.awt. *;
class mychoice extends Applet
{
    Choice mchoice;
    public void init()
    {
        setFont(new Font(' 'ITALIC', Font. ITALIC, 15 ) ),
        mchoice=new Choice();
        mchoice. addItem("one" );
        mchoice. addItem(' 'two" ),
        mchoice. addItem(' 'tbree");
        add(mchoice);
    }
}
```

该程序运行的标签如下：

```
<applet
code=mychoice
Width=300
height=2(X)>
</applet>
```

下拉列表的操作方法如表 12-2 所示。

表 12-2 下拉列表 Choice 的常用操作方法

方法的调用格式	含 义
public void addItem(String item)	增加一项
public int countItems()	总共的项数
public String getItem(int index)	获得索引对应的项
public int getSelectedIndex()	得到选择项的索引
public String getSelectedItem()	得到选择的项
protected String paramString()	获得调试信息
public void select(int pos)	选择指定位置的项
public void select(String str)	选择指定名字的项

12.2.9 容器

作为容器，它首先具有容器的特性，前面我们创建的基本组件都需要用方法 add（）将组件加入到容器中以便在屏幕

上显示，这个容器就是 **Applet** 容器。**Applet** 类继承了容器类 **panel**，因此 **Applet** 是一个容器，要显示的组件除了需要用构造方法创建外，还需要使用 **Applet** 子类也即 **Applet** 程序的主类继承而来的方法 **add()**，将组件直接或者间接加入 **Applet**。例如我们前面在 **Applet** 中加入过按钮、标签等组件。容器类都是继承容器父类 **Container** 而来。**Container** 类中定义容器操作的一般共同的方法。上面的 **add()** 方法就是在 **Container** 中定义的，其它的常用方法列于表 12-3。

表 12-3 **Container** 类中的常用方法

方法的调用格式	含 义
<code>Public Component add(Component comp)</code>	加入组件到当前容器中
<code>Public Component add(Component comp, int pos)</code>	
<code>Public Component add(String name, Component comp)</code>	
<code>Public int countComponents()</code>	获得容器中组件的数目
<code>Public Component getComponent(int n)</code>	取得容器中的指定组件
<code>Public Component[] getComponents()</code>	取得容器中的所有组件
<code>Public LayoutManager getLayout()</code>	取得容器的布局管理
<code>Public Insets insets()</code>	设置容器的边界
<code>Public Component locate(int x, int y)</code>	将组件放入容器的指定位置
<code>Public void paintComponents(Graphics g)</code>	在容器中所有的组件上绘图
<code>Public void remove(Component comp)</code>	移走容器中的一个指定组件
<code>Public void removeAll()</code>	移走容器中的所有组件
<code>Public void setLayout(LayoutManager mgr)</code>	设置容器的布局管理器

12.3 组件的布局方式

布局管理是对容器对象而言的。在 **Container** 类中定义了与布局有关的方法，如取得容器的布局管理方法：

`public LayoutManager getLayout()`，设置容器的布局管理器方法：`Public void setLayout (LayoutManager mgr)`

，其它方法参看上面介绍的 **Container** 的方法。

12.3. 1 **BorderLayout** 方式

BorderLayout 将容器分成 5 区域：东、西、南、北、中。其构造方法为：

`public BorderLayout()`

`Public BorderLayout (int hgap, int vgap)` **hgap** 表示水平间距，**vgap** 表示竖直间距。我们在加入组件时

用 **Component** 类中定义的 `add (String name, Component comp)` 方法，其中 **name** 对于 **BorderLayout** 布局来说，可为：“

North”，“**South**”，“**West**”，“**East**”，“**Center**” “**North**”表示将该组件放入北方。

下面我们举例说明：

`import java.awt.*;`

```

import java.applet.Applet;
class buttondir extends Applet
{
public void init()
{
setLayout(new BorderLayout());
add("North", new Button("North"));
add("South", new Button("South"));
add("East", new Button("East"));
add("West", new Button("West"));
add("Center", new Button("Center"));
}
}

```

其它的布局方式还有：

CardLayout 方式 FlowLayout 方式

GridLayout 方式 GridBagLayout 方式

◎ CardLayout 方式

CardLayout 布局方式是多个组件共享同一块显示区域的布局方式。例如，有两个面板同时在一块区域，这时，

一个面板就在另一个面板上，同一时刻只能显示一个面板。CardLayout 布局方式可以在这两个面板之间切换，显示要显示的面板。CardLayout 布局方式可以使多个面板（也可以是其它的组件，通常使用的是面板）共享一个显示区域，在这些组件之间切换，显示要显示的组件。该方式的构造方法为：

public CardLayout ()

public CardLayout (int hgap, int vgap) hgap 为水平边界距离，vgap 为竖直边界距离。

常用的操作方法有：

public void next (Container target) 显示下一种方式

public void previous (Container target) 显示前一种方式

public void show (Container target, String name) 显示指定的方式，name 为方式名

◎ FlowLayout 方式

该方式为缺省方式，即一个容器没有设置布局方式时，这种方式就是该容器的布局方式。它简单地从左到右布置组

件，并使组件处于行的中间，如果一行容不下，必要时开始新一行。

◎ GridLayout 方式

该布局方式是将容器划分为指定的行和列的网格，每一个网格所占的空间一样，该容器的组件就填充于这些网格中

。每一个组件占据一个网格并且充满网格。使用网格构造方法来创建该布局方式，然后用方法 setLayout () 设置，构造方法为：

public GridLayout (int rows, int cols)


```
public GndLayout (Int rows, Int cds, Int hgap, Int vgcy)
```

◎ GridLayout 方式举例介绍

```
import java.awt.*;  
import java.applet.Applet;  
class grid extends Applet  
public void init()  
{  
    setLayout(new GridLayout(3,3));  
    add(new Button("b1"));  
    add(new Button("b2")),  
    add(new Button("b3")),  
    add(new Button("b4")),  
    add(new Button("b5")),  
    add(new Button("b6")),  
    add(new Button("b7")),  
}  
}
```

下面我们举例介绍。

◎ GridBagLayout 方式

该布局方式相对复杂，其基本操作是：首先用 `GridBagConstraints` 类的构造方法创建一个约束即一个实例对象：

用 `GridBagConstraints` 类的方法设置约束，创建一组件；用 `GridBagLayout` 类的方法

`setConstraints (Component comp,`

`GridBagConstraints constraints)` 将该组件与 `GridBagLayout` 布局相联系；用 `add ()` 方法加入设置了 `GridBagLayout`

的容器中，就完成这一个组件在该容器中的布局。

◎ GridBagLayout 布局方式的一个例子

```
import java.awt.*;  
import java.util.*;  
import java.applet.Applet;  
public class GridBagEx1 extends Applet  
{  
    protected void makebutton(String name,  
        GridBagLayout gridbag,GridBagConstraints c )  
    {  
        Button button = new Button(name);  
        gridbag.setConstraints(button, c);  
        add(button);  
    }  
    public void init()
```

```

{
GridBagLayout gridbag = new GridBagLayout();
GridBagConstraints c = new GridBagConstraints();
setFont(new Font("Helvetica", Font.PLAIN, 14));
setLayout(gridbag);
c.fill = GridBagConstraints.BOTH;
c.weightx = 1.0;
makebutton("Button1", gridbag, c);
makebutton("Button2", gridbag, c);
makebutton("Button3", gridbag, c);
// end row
c.gridwidth = GridBagConstraints.REMAINDER;
makebutton("Button4", gridbag, c);
c.weightx = 0.0; // reset to the default
makebutton("Button5", gridbag, c); // another row
// next-to last in row
c.gridwidth = GridBagConstraints.RELATIVE;
makebutton("Button6", gridbag, c);
c.gridwidth = GridBagConstraints.REMAINDER; // end row
makebutton("Button7", gridbag, c);
c.gridwidth = 1; // reset to the default
c.gridheight = 2;
c.weighty = 1.0;
makebutton("Button8", gridbag, c);
c.weighty = 0.0; // reset to the default
// end row
c.gridwidth = GridBagConstraints.REMAINDER;
c.gridheight = 1; // reset to the default
makebutton("Button9", gridbag, c);
makebutton("Button10", gridbag, c);
resize(300, 100);
}

public static void main(String args[])
{
Frame f = new Frame("GridBag Layout Example");
GridBagExl exl = new GridBagExl();
exl.init();
f.add("Center", exl);
f.pack();
f.show()
}
}

```

12.4 事件处理

当我们在组件上用鼠标点击，AWT 就会产生一个事件，每一个组件都可以处理该组件上发生的事件。一个组件不处理该事件，它可以将事件向上一级即所在容器抛出，由该容器调用相应方法处理。这就是事件处理机制。

我们知道所有图形元件都是继承 Component 类而得来的，在 Component 类中定义了事件处理的方法。这些事件处理方法列于表 12.7。

表 12-7 Component 中定义的事件处理方法	
方法调用格式	含 义
public void deliverEvent(Event evt)	向该组件传递一个事件，该方法会调用 postEvent(Event evt)方法
public boolean postEvent(Event evt)	该方法会调用组件的 handleEvent(Event evt)来处理传递过来的事件
public boolean handleEvent(Event evt)	当组件上产生任何事件时，AWT 都会自动调用该方法去处理事件
public boolean gotFocus(Event evt, Object what)	组件得到焦点处理方法，需要由 handleEvent(Event evt)调用，默认时 handleEvent(Event evt)会调用该方法处理失去焦点事件
public boolean lostFocus(Event evt, Object what)	组件失去焦点处理方法，需要由 handleEvent(Event evt)调用，默认时 handleEvent(Event evt)会调用该方法
public boolean keyDown(Event evt, int key)	键盘事件处理方法，需要由方法 handleEvent(Event evt)调用，默认时 handleEvent(Event evt)会自动调用它们来处理键盘事件
public boolean keyUp(Event evt, int key)	
public boolean action(Event evt, Object what)	
public boolean mouseDown(Event evt, int x, int y)	鼠标事件，需要由 handleEvent(Event evt)调用，默认时，handleEvent(Event evt)会自动调用这些方法来处理相应的鼠标事件
public boolean mouseDrag(Event evt, int x, int y)	
public boolean mouseEnter(Event evt, int x, int y)	
public boolean mouseExit(Event evt, int x, int y)	
public boolean mouseMove(Event evt, int x, int y)	
public boolean mouseUp(Event evt, int x, int y)	

当在一个组件上点击或者进行其它的操作时，该组件会产生任意一个事件，这时 AWT 都会调用 handleEvent (Event evt) 方法来处理。如果该方法成功地处理完，将会返回 false；若不处理，一般需要调用父类的 handleEvent (Event evt)，由父类去处理。若我们没有重写 handleEvent (Event evt) 方法，该方法会自动调用上面具体的事件处理方法，见表 12-7，例如：keyDown (Event evt, int key)、mouseDown (Event evt, int x, int y)。

要处理事件，需要重写处理事件的方法，通常会重写如鼠标事件处理方法、键盘事件处理方法。我们已经介绍

过，组件上任何发生的事件都会调用 `handleEvent (Eventevt)`，它是处理事件的总入口，由它去调用相应的具体事件处

理方法去处理具体的事件,默认时，它会自动调用上面表 12—7 介绍的事件处理方法。一般我们可以不重写该方法，而是重写

具体的事件处理方法，例如 `MouseEnter (Even evt, int x, int y)` 方法。当上面的方法不能够满足需求时，可以定义自

己的具体事件处理方法，这时需要重写 `handleEvent (Even evt)` 方法，以便在该方法中调用自己的具体事件处理方法。

习 题

1.Component 类直接继承于 ()，属于 java.awt。

(A) Container (B) Object (C) Frame (D) Window

2.()构造方法用来创建一个用 Label 字符串名字的核选框。

(A) Checkbox(String label) (B) Checkbox() (C) CheckboxLabel() (D) CheckLabel()

3.文本区是继承超类 () 而来，有超类的方法。

(A) Object (B) Container (C) Component (D) Window

4.List 的常用方法中，`public int countItems()`代表的意义是 ()。

(A) 想列表框中加入一项 (B) 得到总共选择的项目 (C) 判断是否可以多选 (D) 获得列表框中的项数

5.在容器中，创建面板的构造方法为：

(A) `public Window(Frame parent)` (B) `public Frame()`

(C) `public Panel()` (D) `public Frame(String title)`

6.AWT 容器的布局方式有：_____、_____、
_____、_____等。

7.Checkbox 类提供的构造方法有：_____、_____、
_____。

8.创建一个指定显示行列数目且含有初始文本的文本区的构造方法为：

_____。

9.在创建滑动条的创建时，如果我们要创建水平的滑动条是，其构造方法为：

_____。

10.在组件布局方式中，GridLayout 方式的构造方法有：_____、
_____。

11.阅读下面的程序，回答出其运行结果。

```
import java.applet.Applet;
import java.awt.*;
class firstlabel extends Applet
{
Label label1,label2,labe13,label4;
```

```

public void init()
{
label1=new Label("欢迎来到忻州师范学院信息网络中心！");
label2=new Label("中心机房");
label3=new Label("东工作机房");
label4=new Label("西工作机房");
add(欢迎来到河南大学软件技术实验室！);
add(中心机房);
add(东工作机房);
add(西工作机房);
}
}

```

HTML 文件为：

```

<html>
<head>
<title>firstlabel</title>
</head>
<body>
<hr>
<applet
code=firstlabel
width=320
height=240>
</applet>
<hr>
</body>
</html>

```

写出运行结果：

12.编写一个程序，在屏幕上显示一个画板 Canvas,要求：底色为黑色，文字字体为宋体，加黑，字体大小为 20。

显示的文字内容为：HELLO WORLD！

第十三讲:Java 网络通信功能

网络通信是 Java 程序的一个重要功能，java. net 中包含网络通信所需要的类。Java 支持 Internet 的 TCP、UDP 协

议，拥有网络操作的 I/O 特性。Java 程序的网络通信有两种基本的方式，数据包方式和插座方式。数据包方式是服务程序将要

传递的数据打包，分成一个个小的数据包。每一个数据包都有它要传送到的计算机的地址，一旦数据包发送，就不能够保证

它一定能够到达目的地址。同样，在数据的传递过程中，也不能够保证数据不被破坏或者发

送方能够得到应答。因此，这种

方式中服务主机跟客户机不是时时连接的，对于重要数据的传递是不太适用的。

插座方式则是服务主机与客户机时时连接的，服务程序等待客户程序的连接，一旦建立客户程序与服务程序之间的

插座连接，就可以通过流操作的方式来实现发送和接受数据的双向数据传递。在 Internet 上要传递数据，必须通过

InetAddress 地址来指明数据要达到的目的地和服务方的地址。下面我们具体地介绍网络数据传递的相关知识和上面提到的

两种网络数据传递方式的实现。

13.1 Internet 地址

下面我们介绍 Java 提供的类 InetAddress 来进行有关 Internet 地址的操作。

InetAddress 类没有构造方法，要创建该类的实例对象，可以通过该类的静态方法获得该对象。这些静态方法列于下：

```
public static InetAddress getLocalHost ()
```

```
public static InetAddress getByName (String host)
```

```
public static InetAddress [] getAllByName (String host)
```

方法 getLocalHost () 获得本地机的 InetAddress 对象；方法 getByName (String host) 获得由 host 指定的

InetAddress 对象，host 是计算机的域名（也就是名字），其作用跟 IP 地址一样，只不过域名标识计算机比 IP 标识计算机更易于记忆。我们知道，在 Internet 上不允许多台计算机共用一个名字（或者说是 IP 地址），但是在 Web 中，可以用相同的名字代表一组计算机，通过方法 InetAddress [] getAllByName (String host) 可以获得具有相同名字的一组 InetAddress 对象。

InetAddress 对象有它的常用的操作方法，这些方法列于下：

1. public byte[] getAddress()

2. public String getHostName()

3. public String toString()

方法 getAddress() 得到 IP 地址，方法 getHostName () 得到主机名字，方法 toString() 得到主机名和 IP 地址的字符串

。下面举例介绍。

★ 获取主机 IP 地址的例子

```
import java.net.*;
class Internet
{
    public static void main(String args[])
    {
        try
        {
            InetAddress iads;
            iads= InetAddress .getByName("192.168.35.73" );
```

```

System.out.println("hostname="+iads.getHostName());
System.out.println(iads.toString());
}
catch(Exception e)
{
System.out.println(e);
}
}
}
}

```

程序的运行结果为：

```

D:\javaprogram\window>jview internet
hostname=192.168.35.73
192.168.35.73/192.168.35.73

```

13.2 URL 资源定位

Java 将 URL 封装成 URL 类，URL 的组成部分 Java 的类为我们提供了不同的构造方法：

```

public URL(String spec)
public URL(String protocol,String host,int port,String file)
public URL(String protocol, String host, String file)
public URL( context, String spec)
URL (String spec) 构造方法用指定的 URL 来创建一个 URL 对象；URL (String protocol, String
host,
int Port, String file) 构造方法用指定的协议、主机名、端口号、文件路径及 文件名来创建一
个 URL 对象；URL (
String protocol, String host, String file) 构造方法用指定的协议、主机名、路径及文件名来
创建 URL 对象；URL (
URL context, String spec) 构造方法用已存在的 URL 对象来创建 URL 对象。
public final Object getContent()
public String getFile();
public String getHost()
public int getPort()
public String getProtocol()
public String toExternalForm();
public String toString()
public URLConnection openConnection()

```

接下来,我们举例介绍 URL 对象的创建及使用。

★ URL 对象的创建及使用

```

import java.net.*;
class myurl
{

```

```

public static void main(String args[])
{
    try
    {
        URL url=new URL("http://www. tsinghua. edu .cn: 80/home/homepage.htm")
        System.out.println("the Protocol: "+url.getProtocol());
        System.out.println("the hostname: "+url.getHost());
        System.out.println("the port: "+url.getPort());
        System.out.println("the file: "+url.getFile());
        System.out.println("the ext: "+url.toExternalForm());
        System.out.println(url.toString());
    }
    catch(Exception e)
    {
        System.out.println(e);
    }
}

```

该程序运行后的结果如下:

```

the protocol:http
the hostname: www. tsinghua. edu. cn
the port:80
the file: /home/homepage. html
the ext: http://www.tsinghua. edu. cn: 80/home/homepage .html

```

13.3 URLConnection 对象连接 URL 服务器

我们可以直接通过构造方法 `URLConnection(URL url)` 来创建一个 `URLConnection` 对象, 同时我们在上面介绍了可以通过 `URLConnection` 对象的 `openConnection()` 来得到一个 `URLConnection` 对象。下面我们使用 `URLConnection` 对象的操作方法:

```
public int getLength() //获得文件的长度
```

```
public String getContentType() //获得文件的类型
```

```
public long getDate() //获得文件创建的时间
```

```
public long getLastModified() //获得文件最后修改的时间
```

```
public InputStream getInputStream() //获得输入流, 以便读取文件的数据
```


接下来,我们用上面的方法从 **Web** 服务器上读取文件的信息,将文件的信息打印到屏幕并同时写入本地机。

★ **URLConnection** 对象连接 **URL** 服务器

```
import java.net. *;
import java.io. *;
class urlcon
{
public static void main(String args[])
{
InputStream is;
OutputStream os;
int b;
try
{
url=new URL('http://192.168.35.73');
URLConnection uric= url.openConnection();
System.out.println("the length:"+uric.getContentLength());
System.out.println("the type:"+uric.getContentType());
System.out.println("the date: "+uric.getDate());
System.out.println("the lastmodified:"+uric.getLastModified());
os=new FileOutputStream("d:\\down.html" );
if(uric. getContentLength()>= 1 )
{
is=uric.getInputStream();
while((b=is.read())!=-1)
{
os.write(b);
System.out.write(b);
}
}
else
{
System.out.println("no content");
}
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

程序运行的开始, 打印 **Web** 服务器的 **http://192.168.35.73** 的文件的信息如下:

```
D:\javaprogram>window>jview urclon
the length:2113
the type:text/html
the date:925218413000
the lastmodified: 922472682000
```

接着会在屏幕上打印文件中的内容，同时写入 d:\down.html 文件中。

13.4 客/服务器通信

13.4.1 数据包方式实现服务器与客户通信

使用数据包方式首先将数据打包，Java. net 包中的 DatagramPacket 类用来创建数据包。数据包有两种，一种用来

传递数据包，该数据包有要传递到的目的地址；另一种数据包用来接收传递过来的数据包中的数据。要创建发送的数据包，通过 DatagramPacket 类的方法构造：Public DatagramPacket (byte ibuf[], int ilength)

ibuf[]为接受数据包的存储数据的缓冲区的长度，ilength 为从传递过来的数据包中读取的字节数。

创建发送数据包的构造方法为：

```
public DatagramPacket (byte ibuf[], int ilength, InetAddress iaddr,int port)
```

iaddr 为数据包要传递到的目标地址，ipport 为目标地址的程序接受数据包的端口号（即目标地址的计算机上运行的客户程序是在哪一个端口接收服务器发送过来的数据包）。

数据包也是对象，也有操作方法用来获取数据包的信息，这是很有用的，列于下：

```
1.public InetAddress getAddress () //获得数据包要发送的目标地址
```

```
2.Public byte [] getData () //获得数据包中的数据
```

```
3.public int getLength () //获得数据包中数据的长度
```

```
4.Public int getPort () //获得数据包中的目标地址的主机端口号
```

发送和接收数据包还需要发送和接收数据包的插座，即 DatagramSocket 对象，通过构造方法：

```
1.public DatagramSocket () //用本地机上任何一个可用的端口创建一个插座
```

```
2.Public DatagramSocket (int port) //用一个指定的端口创建一个插座
```

插座对象也有相应的方法，例如插座发送数据包的方法、插座接收数据包的方法，介绍如下：

```
1.Public void close () //当我们创建一个插座的同时打开了插座，用该方法来关闭插座
```

```
2. public int getLocalPort () //得到本地机的端口
```

3. public void receive (datagrampacket p) //接收数据包

4. public void Send (DatagramPacket p) //发送数据包

★ 服务程序向客户程序传递一个数据包

有了上面的方法，下面我们举例介绍数据包方式的服务器和客户机之间的通信。这个程序演示服务程序向客户程序

传递一个数据包，读者可以扩充功能，用循环方式使服务程序可以发送多个数据包，同时使客户程序可以接收多个数据包。

程序被编译后，将会生成两个主类，一个主类 **server** 在服务器端运行，一个主类 **client** 在客户机端运行，以实现数据的传送功能。

服务器端运行的程序代码为：

```
import java.net.*;
import java.io.*;
class server
{
    public static void main(String args[])
    {
        int indexco;
        int serverPort=100;
        int clientport=200;
        int b_size=6();
        byte[] buf=new byte[b_size];
        try
        {
            InetAddress iadd= InetAddress. getByName(" 192. 168. 23. 253 ");
            DatagramSocket ds=new DatagramSocket(serverport);
            int c;
            while((c=System. in .read())!='\n',)
            {
                switch(c)
                case '\r':break;
                default: buf[index]=(byte)c;
                index++;
            }
        }
        DatagramPacket dp=new DatagramPacket(buf,index,iadd,clientport);
        ds.send(dp);
    }
    catch(Exception e)
    {
        System.out.println(e);
    }
}
```

```
}
}
```

上面为服务器端发送数据程序的源代码，下面是客户端获取数据的程序源代码：

```
import java.io.*;
class c1ient
{
public static void main(String args[])
{
int clientPort=200;
int b_size=60;
byte[] buf=new byte[b_size];
ny
{
DatagramPacket dp=new DatagramPacket(buf, b_size );
DatagramSocket ds=new DatagramSocket(clientport);
ds.receive(dp);
System.out.println(new String (dp.getData(),0,0, dp. getLength() ) );
}
catch(Exception e)
{ }
}
}
```

为了能够在本地机上进行演示，上面程序中的 `InetAddress. getByName("192. 168. 23. 253")` 中的 192.

168. 23. 253 应改为本地机的 IP 地址，服务器程序与客户程序源代码分别编译后，生成服务器端程序和客户端可执行程序。

13.4.2 流连接方式实现客户端与服务器之间的通信

◎ 客户端插座 Socket

创建客户端插座就会建立与服务器端插座之间的连接，如果没有服务器，则会产生一个错误。连接建立，就可以

获取插座的建立与服务器端相连的输入输出流和进行其它的操作，用输入输出流来进行数据双向传递了。这些操作通过

Socket 对象提供的方法进行，

例如 Socket 对象提供了方法 `public InputStream getInputStream ()`、`public OutputStream getOutputStream ()` 来获取与插座相连的输入输出流。

创建客户端插座可以通过 Socket 的构造方法，介绍如下：

1. `public Socket (String host, int port)`

2. `public Socket (InetAddress address, int port)` // 用该方法创建一个与指定远程主机 IP 地址及其端口 IP 建立的连接；

3. `public Socket (String host, int port, boolean stream)`

4. public socket(InetAddress address, Int port, boolean stream)

当 stream 值为 true，则创建插座时建立一个流，否则，创建插座时，建立一个数据包插座（即 Datagramsocket 的对象）。

Socket 对象的操作方法介绍如下：

1. public void close () //关闭 Socket

2. Public InetAddress getInetAddress () //得到远程主机 IP 地址的 InetAddress 对象

3. public InputStream getInputStream () //得到 Socket 建立的输入流

4. public int getLocalPort() //得到与远程机连接的本地机的端口号

5. pubic OutputStream getOutputStream () //得到 Socket 建立的输出流

6. public int getport () //得到与之建立连接的远程主机的端口号

★ 创建一个客户端程序的例子

★ 创建一个客户端程序的例子

```
import java.net.*;
import java.io.*;
class client
{
public static void main(String args[1]) throws Exception
{
int remotelP=666;
String remotelP="192.168.23.253";
Socket s=new Socket(remotelP, remoteport);
DataOutPutStream dos=new DataOutPutStream(s.getOutPutStream() );
DataInPutStream dis=new DataInPutStream System.in);
String str;
System.out.print("client quest:");
sfordis.readLine();
while(str!=null)
{
System.out.print("client quest:");
dos.writeBytes(str);
dos.writeChar('\n');
if(str.equals("bye!"))
break;
str=dis.readLine();
}
```

```

}
dis.close();
dis.close();
s.close();
}
}

```

另外还有一种与 `GridLayout` 相似更为灵活的布局方式 `GridBagLayout`，该方式允许组件有不同的高度和宽度。

◎ 服务器端插座 `ServerSocket`

服务器端的插座等待客户端插座的连接。它与客户端的插座不同，客户端的插座把没有连接的情况当作一种错误操作。

服务器端插座可以通过 `ServerSocket` 的构造方法来创建，当该对象创建后就表示愿意与客户端建立连接。

`ServerSocket` 的构造方法为：

```
public ServerSocket (int port)
```

指定端口号来创建 `ServerSocket` 对象，该端口为端主机的端口；

```
public ServerSocket (int port, int count)
```

用指定的端口号创建 `ServerSocket` 对象，如果该端的端口正在使用，只等待 `count` 毫秒。

下面介绍 `ServerSocket` 对象的操作方法：

1. `public Socket accept ()` // 获取与客户端连接的 `Socket` 对象
2. `public void close ()` // 关闭端的 `ServerSocket` 对象
3. `public InetAddress getInetAddress ()` // 得到与端相连的客户端的 `InetAddress` 对象
4. `public int getLocalPort ()` // 得到服务器在监听的端口号

★ 与前面客户端程序相对应的服务端的程序举例

★ 与前面客户端程序相对应的服务端的程序举例

```

import java.net.*;
import java.io. *;
class server
{
public static void main(String args[]) throws Exception
{
int serverport=666,
ServerSocket ss=new ServerSocket(serverport);
socket s=ss.accept();
DataInputStream dis=new DataInputStream(s.getInputStream ());
String str;
str=dis.readLine();

```

```

while(str!=null)
{
    System.out.println(" Answer: "+str);
    System.out.flush();
    if(str.equals("bye!"))
        break;
    str=dis.readLine();
}
dis.close();
s.close();
ss.close();
}
}

```

为了演示客户端和端的通信，在客户程序中，将远程端的 `remoteIP` 地址设置成客户程序运行所在机器的 IP 地址，以便端和客户端程序能够在一台计算机实现通信，编译上面的端和客户端程序。

第十三讲:Java 网络通信功能

习 题

1. 获得本地机的 `InetAddress` 对象的方法是 ()。
(A) `getByName()` (B) `getLocalHost()` (C) `getAllByName()` (D) `getName()`
2. 在互联网中用于统一资源定位，并且是 World Wide Web 上的基本构件，它是 ()。
(A) `URL` (B) `HTML` (C) `Java` (D) `Applet`
3. 在 `URLConnection` 对象中，用于获得输入流，方便读取数据的方法是 ()。
(A) `getContentLength()` (B) `getContentType()` (C) `getLastModified()` (D) `getInputStream()`
4. 在数据报方式实现服务器与客户通信中，用于得到本地机的端口的的方法是 ()。
(A) `DatagramSocket()` (B) `InetAddress getAddress()`
(C) `getLocalPort()` (D) `DatagramSocket(int port)`
5. 服务器插座通过 () 方法来创建。
(A) `Socket` (B) `ServerSocket` (C) `DatagramSocket()` (D) `DatagramPacket()`
6. 我们可以通过类的静态方法获得实例对象，这些方法有：_____、_____、_____。
7. 阅读下面的程序，填空。

```

import java.net.*;
class Internet
{
    public static void main(String args[])
    {
        try
        {
            InetAddress iads;

```

```

iads= InetAddress .getByName("172.20.20.100" );
System. out. println("hostname="+iads. getHostName());
System. out. println(iads. toString() );
}
catch(Exception e)
{
System.out.Println(e);
}
}
}
}

```

(1) 该程序的作用是：_____。

(2) 用命令 D:\javaprogram\window>jview internet 运行该程序，写出结果为：

_____、_____。

8.阅读下面的程序，写出运行结果。

```

import java.net. *;
class myurl
{
public static void main(String args[])
{
try
{
URL url=new [URL("http://www.henu.edu.cn:80/home/homepage.htm")
System.out.println("the Protocol: "+url.getPrtOCol());
System.out.Println(',the hostname: tt +url .getHost());
System.out.println("the port' "+url.getPort());
System.out.prinhn("the file:' +url.getFle());
System.out.printin("the ext: "+url.toExtersmalForm)),
System. out. println(url. toString());
}
catch(Exception e)
{
System.out.println(e);
}
}
}
}

```

运行结果为：

the protocol:_____。

the hostname:_____。

the port:_____。

the file:_____。

the ext:_____。