

全国计算机技术与软件专业技术资格（水平）考试指定用书

软件设计师教程

（第2版）

陈平 褚华 主编

全国计算机技术与软件专业技术资格（水平）考试办公室组编

清华大学出版社



2007版

全国计算机技术与软件专业资格(水平)考试真题及答案

[2008年下半年试题分析与解答 软考指定用书 清华出版](#)(含各科)

[2008年上半年试题分析与解答 软考指定用书 清华出版](#)(含各科)

[2007年下半年试题分析与解答 软考指定用书 清华出版](#)(含各科)

[2007年上半年试题分析与解答 软考指定用书 清华出版](#)(含各科)

[2006年下半年试题分析与解答 软考指定用书 清华出版](#)(含各科)

[2006年上半年试题分析与解答 软考指定用书 清华出版](#)(含各科)

[2009年计算机技术与软件水平考试各科目考试大纲汇总](#)

[全国计算机技术与软件专业资格\(水平\)考试真题及答案汇总](#)

[\[软考视频\]计算机技术与软件专业资格考试推荐视频教程下载汇总](#)

教材及同步辅导见下页。

计算机技术与软件专业技术(水平)考试指定教材及同步辅导

软考初级:

[程序员教程\(第二版\)2007 版 软考指定用书 高清PDF版](#)

[程序员考试辅导: 全国计算机技术与软件专业技术资格\(水平\)考试辅导用书](#)

[网络管理员教程\(第 2 版\)2007 版 软考指定用书 高清PDF版](#)

[网络管理员考试同步辅导\(计算机与网络基础知识篇\) 软考指定辅导用书](#)

[网络管理员考试同步辅导\(网络系统管理与维护篇\) 软考指定使用辅导用书](#)

软考中级:

[网络工程师教程\(第 2 版\) 2007 版 软考指定用书 高清PDF版](#)

[网络工程师教程 软考指定用书 高清PDF版](#)

[网络工程师考试同步辅导: 计算机与网络知识篇 软考指定用书](#)

[网络工程师考试同步辅导\(网络系统设计与管理篇\) 软考指定辅导用书](#)

[软件设计师教程\(第 2 版\) 2007 版 软考指定用书 高清PDF版](#)

[软件设计师考试同步辅导\(下午科目\) 高清PDF版](#)

[软件设计师考试同步辅导\(上午科目\) 高清PDF版](#)

[软件设计师考试考点分析与真题详解\(软件设计技术篇\)](#)

[软件设计师考试辅导：考点精讲、例题分析、强化训练 冶金工业出版](#)

[数据库系统工程师教程 软考指定用书 高清PDF版](#)

[软件评测师教程 软考指定教材 高清PDF版](#)

[信息系统管理工程师教程 软考指定用书 高清PDF版](#)

[信息系统监理师教程 软考指定用书 高清PDF版](#)

软考高级：

[系统分析师教程 软考指定教材 高清PDF版](#)

[系统分析师考试辅导\(2007 版\) 软考指定辅导用书 高清PDF版](#)

[系统分析师教程 PDF文字版](#)

[系统分析师经典教材 Word版](#)

[信息系统项目管理师教程 软考指定教材 高清PDF版](#)

[信息系统项目管理师辅导教程\(上下册\)](#)

[计算机专业英语教程 PDF文字版](#)

更多计算机资料请访问：[大家论坛计算机专区](#)

内 容 简 介

本书按照人事部、信息产业部全国计算机技术与软件专业技术资格(水平)考试要求编写,内容紧扣《软件设计师考试大纲》,阐述软件设计师考试必备的知识和技能的要点。

全书共 12 章,分别对计算机系统知识、程序语言、操作系统、系统开发与运行、网络基础知识、多媒体基础知识、数据库技术、数据结构、常用算法设计、面向对象技术、标准化基础知识和知识产权基础知识进行了详尽的讲解。

本书是软件设计师考试应试者的必读教材,也可以作为各类计算机信息技术培训和辅导的教材,还可以作为广大工程技术人员学习计算机信息技术基础知识的参考书。

本书扉页为防伪页,封面贴有清华大学出版社防伪标签,无上述标识者不得销售。

版权所有,侵权必究。侵权举报电话:010-62782989 13501256678 13801310933

图书在版编目(CIP)数据

软件设计师教程 / 陈平, 褚华主编. —2 版. —北京: 清华大学出版社, 2006.6

(全国计算机技术与软件专业技术资格(水平)考试指定用书)

ISBN 978-7-302-12957-8

I. 软… II. ①陈…②褚… III. 软件设计-工程技术人员-资格考核-自学参考资料 IV. TP311.5

中国版本图书馆 CIP 数据核字(2006)第 043270 号

责任编辑: 柴文强 刘 霞

责任印制: 孟凡玉

出版发行: 清华大学出版社 地 址: 北京清华大学学研大厦 A 座

<http://www.tup.com.cn> 邮 编: 100084

c-service@tup.tsinghua.edu.cn

社 总 机: 010-62770175 邮购热线: 010-62786544

投稿咨询: 010-62772015 客户服务: 010-62776969

印 装 者: 清华大学印刷厂

经 销: 全国新华书店

开 本: 185 × 230 印张: 40.25 防伪页: 1 字数: 906 千字

版 次: 2006 年 6 月第 2 版 印 次: 2007 年 4 月第 3 次印刷

印 数: 25001 ~ 30000

定 价: 60.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题,请与清华大学出版社出版部联系调换。联系电话: (010)62770177 转 3103 产品编号: 021636-02/TP

序

在国务院鼓励软件产业发展政策的带动下，我国软件业一年一大步，实现了跨越式发展，销售收入由 2000 年的 593 亿元增加到 2003 年的 1633 亿元，年均增长速度 39.2%；2000 年出口软件仅 4 亿美元，去年则达到 20 亿美元，三年中翻了两番多；全国“双软认证工作体系”已经规范运行，截止 2003 年 11 月底，认定软件企业 8582 家，登记软件产品 18287 个；11 个国家级软件产业基地快速成长，相关政策措施正在落实；我国软件产业的国际竞争力日益提高。

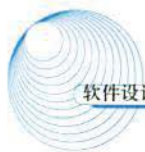
在软件产业快速发展的带动下，人才需求日益迫切，队伍建设与时俱进，而作为规范软件专业人员技术资格的计算机软件考试已在我国实施了十余年，累计报考人数超过一百万，为推动我国软件产业的发展作出了重要贡献。

软件考试在全国率先执行了以考代评的政策，取得了良好的效果。为贯彻落实国务院颁布的《振兴软件产业行动纲要》和国家职业资格证书制度，国家人事部和信息产业部对计算机软件考试政策进行了重大改革：考试名称调整为计算机技术与软件专业技术资格（水平）考试；考试对象从狭义的计算机软件扩大到广义的计算机软件，涵盖了计算机技术与软件的各个主要领域（5 个专业类别、3 个级别层次和 20 个职业岗位资格）；资格考试和水平考试合并，采用水平考试的形式（与国际接轨，报考不限学历与资历条件），执行资格考试政策（各用人单位可以从考试合格者中择优聘任专业技术职务）；这是我国人事制度改革的一次新突破。此外，将资格考试政策延伸到高级资格，使考试制度更为完善。

信息技术发展快，更新快，要求从业人员不断适应和跟进技术的变化，有鉴于此，国家人事部和信息产业部规定对通过考试获得的资格（水平）证书实行每隔三年进行登记的制度，以鼓励和促进专业人员不断接受新知识、新技术、新法规的继续教育。考试设置的专业类别、职业岗位也将随着国民经济与社会发展而动态调整。

目前，我国计算机软件考试的部分级别已与日本信息处理工程师考试的相应级别实现了互认，以后还将继续扩大考试互认的级别和国家。

为规范培训和考试工作，信息产业部电子教育中心组织一批具有较高理论水平和丰富实践经验的专家编写了全国计算机技术与软件专业技术资格（水平）考试的教材和辅导用书，按照



考试大纲的要求,全面介绍相关知识与技术,帮助考生学习和备考。

我们相信,经过全社会的共同努力,全国计算机技术与软件专业技术资格(水平)考试将会更加规范、科学,进而对培养信息技术人才,加快专业队伍建设,推动国民经济和社会信息化作出更大的贡献。

信息产业部副部长 娄勤俭

前言

(第2版)

全国计算机软件专业技术资格(水平)考试实施至今已经历了近二十年,在社会上产生了很大的影响,对我国软件产业的形成和发展做出了重要的贡献。为了适应我国计算机信息技术发展的需求,国家人事部和信息产业部决定将考试的级别拓展到计算机信息技术行业的各个方面,以满足社会上对各种计算机信息技术人才的需要。

编者受信息产业部计算机软件专业技术资格(水平)考试办公室委托,在《软件设计师教程》一书的基础上进行修编。在考试大纲中,要求考生掌握的知识面很广,一个条目在大学里可能是一学期的课程,因此编写的难度很高。考虑到参加考试的人员已有一定的基础,所以本书中只对考试大纲中所涉及到的知识领域的要点加以阐述,限于篇幅不能详细地展开,请读者谅解。

全书共分12章,由陈平、褚华担任主编。第1章计算机系统知识由李伯成、褚华编写,第2章程序语言基础知识由张淑平编写,第3章操作系统知识由王亚平编写,第4章系统开发和运行知识由褚华编写,第5章网络基础知识由张凤琴编写,第6章多媒体基础知识由刘强编写,第7章数据库技术基础由王亚平编写,第8章数据结构由张淑平编写,第9章常用算法设计方法由褚华编写,第10章面向对象技术由褚华、胡圣明、陈平编写,第11章标准化基础知识和第12章知识产权基础知识由刘强编写,全书由褚华统稿。

在本书的编写过程中,参考了许多相关的书籍和资料,编者在此对这些参考文献的作者表示感谢。同时感谢清华大学出版社在本书出版过程中所给予的支持和帮助。

因水平有限,书中难免存在错漏和不妥之处,望读者指正,以利改进和提高。

编者

2006年2月于西安电子科技大学

目 录

第 1 章 计算机系统知识1

1.1 计算机体系结构1

1.1.1 计算机体系结构的发展1

1.1.2 存储系统3

1.1.3 CISC/RISC13

1.1.4 输入/输出技术14

1.1.5 流水线操作20

1.1.6 总线结构22

1.1.7 多处理机与并行处理23

1.2 安全性、可靠性与系统性能

评测基础知识28

1.2.1 计算机安全概述28

1.2.2 加密技术30

1.2.3 认证技术34

1.2.4 计算机可靠性38

1.2.5 计算机系统的性能评价41

1.2.6 计算机故障诊断与容错45

第 2 章 程序语言基础知识48

2.1 程序语言概述48

2.1.1 程序语言的基本概念48

2.1.2 程序设计语言的种类和特点49

2.1.3 程序语言的基本成分53

2.2 语言处理程序基础59

2.2.1 汇编语言基本原理59

2.2.2 编译程序基本原理62

2.2.3 解释程序基本原理90

第 3 章 操作系统知识93

3.1 操作系统基础知识93

3.1.1 操作系统的定义与作用93

3.1.2 操作系统的特征与功能94

3.1.3 操作系统的类型95

3.2 处理机管理98

3.2.1 基本概念98

3.2.2 进程的控制102

3.2.3 进程间的通信104

3.2.4 管程109

3.2.5 进程调度111

3.2.6 死锁112

3.2.7 线程115

3.3 存储管理116

3.3.1 基本概念117

3.3.2 分页存储管理118

3.3.3 虚拟存储管理120

3.4 设备管理125

3.4.1 设备管理概述125

3.4.2 I/O 软件127

3.4.3 通道、DMA 与缓冲技术130

3.4.4 Spooling 技术132

3.4.5 磁盘调度133

3.5 文件管理134

3.5.1 文件与文件系统134

3.5.2 文件的结构和组织136

3.5.3 文件目录139

3.5.4 存取方法和存储空间的管理141

3.5.5 文件的使用142

3.5.6 文件的共享和保护142

3.5.7 系统的安全与可靠性144

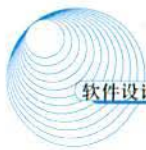
3.6 作业管理146

3.6.1 作业管理和作业控制146

3.6.2 作业调度147

3.6.3 用户界面148

3.7 网络操作系统和嵌入式操作



系统基础知识	149	4.5.2 系统评价	233
3.7.1 网络操作系统	149	第5章 网络基础知识	235
3.7.2 嵌入式操作系统	151	5.1 网络概述	235
3.8 操作系统实例	152	5.1.1 计算机网络的概念	235
3.8.1 UNIX 操作系统	152	5.1.2 计算机网络的分类	238
3.8.2 Windows 2000/XP 操作系统	159	5.1.3 网络的拓扑结构	239
第4章 系统开发和运行知识	165	5.2 ISO/OSI 网络体系结构	241
4.1 软件工程基础知识	165	5.3 网络互连硬件	244
4.1.1 软件工程概述	165	5.3.1 网络的设备	244
4.1.2 软件需求分析	169	5.4.2 网络的传输介质	247
4.1.3 软件开发项目管理	171	5.3.3 组建网络	249
4.1.4 软件配置管理	176	5.4 网络的协议与标准	252
4.1.5 软件工具与软件开发环境	178	5.4.1 网络的标准	253
4.1.6 软件过程管理	181	5.4.2 局域网协议	254
4.1.7 软件质量管理与质量保证	183	5.4.3 广域网协议	258
4.2 系统分析基础知识	191	5.4.4 Internet 协议	263
4.2.1 系统分析概述	191	5.5 Internet 及应用	268
4.2.2 结构化分析方法	193	5.5.1 Internet 概述	268
4.2.3 系统分析报告	199	5.5.2 Internet 地址	269
4.3 系统设计知识	200	5.5.3 Internet 服务	273
4.3.1 系统设计的内容和步骤	200	5.6 WindowsNT 系统及管理	278
4.3.2 系统设计的基本原理	202	5.6.1 Windows NT 概述	279
4.3.3 系统总体结构设计	204	5.6.2 WindowsNT 系统管理	283
4.3.4 结构化设计方法	208	5.7 网络安全	285
4.3.5 面向数据结构的设计方法	210	5.7.1 网络安全概述	285
4.3.6 系统详细设计	212	5.7.2 网络的信息安全	287
4.4 系统实施知识	217	5.7.3 防火墙技术	291
4.4.1 系统实施概述	217	第6章 多媒体基础知识	298
4.4.2 程序设计	218	6.1 多媒体的基本概念	298
4.4.3 系统测试与调试	221	6.1.1 媒体的分类	298
4.4.4 测试策略和测试方法	222	6.1.2 多媒体的特征	299
4.4.5 调试	226	6.2 音频	300
4.4.6 系统文档	227	6.2.1 数字声音基础	300
4.4.7 系统转换	228	6.2.2 波形声音	302
4.5 系统运行和维护知识	229	6.2.3 声音合成	304
4.5.1 系统维护概述	229	6.2.4 MIDI	306

6.2.5	声音文件格式	307
6.3	图形和图像	308
6.3.1	彩色与图像基础	308
6.3.2	计算机中的图形数据表示	310
6.3.3	图像的获取	311
6.3.4	图像的属性	312
6.3.5	图形图像转换	313
6.3.6	图像的压缩编码	314
6.3.7	多媒体数据压缩编码的 国际标准	316
6.3.8	图形、图像文件格式	317
6.4	动画和视频	319
6.4.1	动画	319
6.4.2	模拟视频	322
6.4.3	数字视频	323
6.4.4	数字视频标准	324
6.4.5	视频压缩编码	325
6.4.6	视频文件格式	327
6.5	多媒体网络	328
6.5.1	超文本与超媒体	329
6.5.2	流媒体的基本概念	330
6.5.3	互联网上获取声音和 影视的方法	330
6.6	多媒体计算机系统	332
6.6.1	多媒体计算机硬件系统	333
6.6.2	多媒体软件系统	335
6.7	虚拟现实的概念	338

第7章 数据库技术基础

7.1	基本概念	342
7.1.1	数据库与数据库管理系统	342
7.1.2	DBMS 的功能	343
7.1.3	DBMS 的特征及分类	344
7.1.4	数据库的三级模式结构	345
7.2	数据模型	348
7.2.1	数据模型的基本概念	348
7.2.2	数据模型的三要素与 常用的数据模型	349

7.2.3	E-R 模型	349
7.2.4	层次模型	356
7.2.5	网状模型	357
7.2.6	关系模型	358
7.3	关系代数	359
7.3.1	关系数据库的基本概念	359
7.3.2	5 种基本的关系代数运算	364
7.3.3	扩展的关系代数运算	366
7.4	关系数据库 SQL 语言简介	374
7.4.1	SQL 数据库体系结构	374
7.4.2	SQL 的基本组成	376
7.4.3	SQL 数据定义	376
7.4.4	SQL 数据查询	380
7.4.5	SQL 数据更新	389
7.4.6	SQL 的访问控制	390
7.4.7	嵌入式 SQL	392
7.5	关系数据库规范化	393
7.5.1	函数依赖	393
7.5.2	规范化	394
7.5.3	模式分解及分解 应具有的特性	397
7.6	数据库的控制功能	402
7.6.1	事务管理	402
7.6.2	数据库的备份与恢复	403
7.6.3	并发控制	404
7.6.4	安全性和授权	406

第8章 数据结构

8.1	线性结构	412
8.1.1	线性表	412
8.1.2	栈和队列	416
8.1.3	串	430
8.2	数组、矩阵和广义表	435
8.2.1	数组	435
8.2.2	矩阵	437
8.2.3	广义表	438
8.3	树	439
8.3.1	树的定义及基本运算	439

8.3.2	二叉树的定义及基本运算	441	9.2.1	迭代法	506
8.3.3	二叉树的性质	441	9.2.2	穷举搜索法	506
8.3.4	二叉树的存储结构	442	9.2.3	递推法	507
8.3.5	二叉树的遍历	444	9.3	递归法	508
8.3.6	线索二叉树	446	9.4	分治法	512
8.3.7	二叉树的应用: 最优二叉树	448	9.4.1	分治法的基本思想	512
8.3.8	树和森林	452	9.4.2	分治法的典型实例	513
8.4	图	454	9.5	动态规划法	517
8.4.1	图的定义	455	9.5.1	动态规划法的基本思想	517
8.4.2	图的存储结构	456	9.5.2	动态规划法的典型实例	518
8.4.3	图的遍历	458	9.6	回溯法	522
8.4.4	生成树及最小生成树	461	9.6.1	回溯法的算法框架	523
8.4.5	拓扑排序和关键路径	463	9.6.2	回溯法的典型实例	525
8.4.6	最短路径	466	9.7	贪心法	532
8.4.7	图的应用	469	9.8	分支限界法	536
8.5	查找	471	9.9	概率算法简介	538
8.5.1	查找的基本概念	471			
8.5.2	静态查找表	472	第10章 面向对象技术		539
8.5.3	动态查找表	476	10.1	面向对象的基本概念	539
8.5.4	哈希表及其查找	486	10.2	面向对象程序设计	541
8.6	排序	489	10.2.1	面向对象的好处	542
8.6.1	排序的基本概念及运算	489	10.2.2	面向对象程序设计语言	542
8.6.2	简单排序	490	10.2.3	程序设计语言中的 OOP 机制	545
8.6.3	希尔排序	492	10.2.4	面向对象的程序	550
8.6.4	快速排序	493	10.3	面向对象开发技术	553
8.6.5	堆排序	494	10.3.1	面向对象分析	553
8.6.6	归并排序	497	10.3.2	面向对象设计	555
8.6.7	基数排序	498	10.3.3	面向对象测试	555
8.6.8	内部排序方法的比较和选择	499	10.4	面向对象分析与设计方法	556
8.6.9	外部排序	500	10.4.1	Peter Coad 和 Edward Yourdon 的 OOA 和 OOD 方法	556
			10.4.2	Booch 的 OOD 方法	558
			10.4.3	OMT 方法	558
			10.4.4	UML 概述	561
第9章 常用算法设计方法		504	10.5	设计模式	570
9.1	算法和算法设计基本概念	504	10.5.1	设计模式的要素	570
9.1.1	算法	504			
9.1.2	算法设计	504			
9.1.3	算法效率的度量	505			
9.1.4	算法的存储空间需求	506			
9.2	迭代法、穷举搜索法、递推法	506			

10.5.2	创建型设计模式	571
10.5.3	结构型设计模式	572
10.5.4	行为设计模式	573
第 11 章	标准化基础知识	577
11.1	标准化的基本概念	577
11.1.1	标准、标准化的概念	577
11.1.2	标准化的范围和对象	577
11.1.3	标准化的实质	578
11.1.4	标准化的目的	579
11.2	标准化过程模式	579
11.2.1	标准的制定	579
11.2.2	标准的实施	580
11.2.3	标准的更新	580
11.3	标准的分类	581
11.3.1	根据适用范围分类	581
11.3.2	根据标准的性质分类	583
11.3.3	根据标准化的对象和作用分类	584
11.3.4	根据法律的约束性分类	586
11.4	标准的代号和编号	586
11.5	国际标准和国外先进标准	588
11.5.1	国际标准	588
11.5.2	国外先进标准	588
11.5.3	采用国际标准和国外先进标准	589
11.5.4	采用程度的概念	589
11.5.5	采用国际标准和国外先进标准的原则	590
11.6	信息技术标准化	591
11.6.1	信息编码标准化	591
11.6.2	条码标准化	592
11.6.3	汉字编码标准化	592
11.6.4	软件工程标准化	592
11.7	标准化组织	594
11.7.1	国际标准化组织	594
11.7.2	区域标准化组织	595
11.7.3	行业标准化组织	596

11.7.4	国家标准化组织	596
11.8	ISO9000 标准简介	597
11.8.1	ISO9000 标准	597
11.8.2	ISO9000: 2000 系列标准文件结构	598
11.8.3	ISO9000: 2000 核心标准简介	598
11.8.4	ISO9000: 2000 系列标准确认的八项原则	599
11.9	能力成熟度模型 CMM 简介	602
11.10	ISO/IEC 15504 过程评估标准简介	604

第 12 章	知识产权基础知识	607
12.1	知识产权的概念与特点	607
12.1.1	知识产权的概念	607
12.1.2	知识产权的特点	608
12.1.3	我国保护知识产权的法规	610
12.2	计算机软件著作权的主体与客体	610
12.2.1	计算机软件著作权的主体	610
12.2.2	计算机软件著作权的客体	611
12.3	计算机软件受著作权法保护的条件	612
12.4	计算机软件著作权的权利	613
12.4.1	计算机软件的著作人身权	613
12.4.2	计算机软件的著作财产权	613
12.4.3	软件合法持有人的权利	614
12.4.4	计算机软件著作权的行使	614
12.4.5	计算机软件著作权的保护期	615
12.5	计算机软件著作权的归属	615
12.5.1	软件著作权归属的基本原则	615
12.5.2	职务开发软件著作权的归属	615
12.5.3	合作开发软件著作权的归属	617
12.5.4	委托开发软件著作权的归属	617
12.5.5	接受任务开发软件著作权的归属	618
12.5.6	计算机软件著作权主体变更后软件著作权的归属	618
12.6	计算机软件著作权侵权的鉴别	620
12.6.1	计算机软件著作权侵权行为	620



12.6.2 不构成计算机软件侵权的 合理使用行为	622	12.9.1 专利权的保护对象与特征	627
12.6.3 计算机著作权软件 侵权的识别	622	12.9.2 授予专利权的条件	628
12.7 软件著作权侵权的法律责任	623	12.9.3 专利的申请	629
12.8 计算机软件的商业秘密权	625	12.9.4 专利权的行使	630
12.8.1 商业秘密的概念	625	12.9.5 专利权的限制	631
12.8.2 计算机软件商业秘密的侵权	626	12.9.6 专利侵权行为	632
12.8.3 计算机软件商业秘密 侵权的法律责任	626	12.10 企业知识产权的保护	632
12.9 专利权概述	627	12.10.1 知识产权管理	632
		12.10.2 知识产权的保护和利用	633
		12.10.3 建立经济约束机制规范 调整各种关系	634

第 1 章 计算机系统知识

1.1 计算机体系结构

1.1.1 计算机体系结构的发展

1. 计算机系统结构概述

计算机系统结构又称为计算机体系结构,就是计算机的属性及功能特征,即计算机的外特性。尽管不同的使用者所了解的计算机的属性有所不同,就通用计算机系统来说,计算机系统结构的属性应包括:

- 硬件所能处理的数据类型。
- 所能支持的寻址方式。
- CPU 的内部寄存器。
- CPU 的指令系统。
- 主存的组织与主存的管理。
- 中断系统的功能。
- 输入输出设备及连接接口。
- 计算机体系结构类型。

2. 计算机体系结构分类

(1) Flynn 分类法

1966 年 Flynn 提出了如下定义:

- 指令流 (Instruction Stream): 机器执行的指令序列。
- 数据流 (Data Stream): 由指令流调用的数据序列,包括输入数据和中间结果。
- 多倍性 (Multiplicity): 在系统最受限制的元件上同时处于同一执行阶段的指令或数据的最大可能个数。

按指令流和数据流的不同组织方式,把计算机体系结构分为如下 4 类:单指令流单数据流 (SISD);单指令流多数据流 (SIMD);多指令流单数据流 (MISD);多指令流多数据流 (MIMD)。

(2) 冯式分类法

1972 年冯泽云提出用最大并行度来对计算机体系结构进行分类。所谓最大并行度 P_m 是指计算机系统在单位时间内能够处理的最大的二进制位数。设每一个时钟周期 Δt_i 内能处理的二进制位数为 P_i , 则 T 个时钟周期内平均并行度为: $P_a = (\sum P_i) / T$, 其中 i 为 $1, 2, \dots, T$ 。平均并行度取决于系统的运行程度, 与应用程序无关。所以系统在周期 T 内的平均利用率为 $\mu = P_a / P_m = (\sum P_i) / (T \times P_m)$ 。

图 1-1 所示为用最大并行度对计算机体系结构的分类。用平面直角坐标系中的一点表示一个计算机系统, 横坐标表示字宽 (N 位), 即在一个字中同时处理的二进制位数; 纵坐标表示位片宽度 (M 位), 即在一个位片中能同时处理的字数, 则最大并行度 $P_m = N \times M$ 。由此得出 4 种不同的计算机结构:

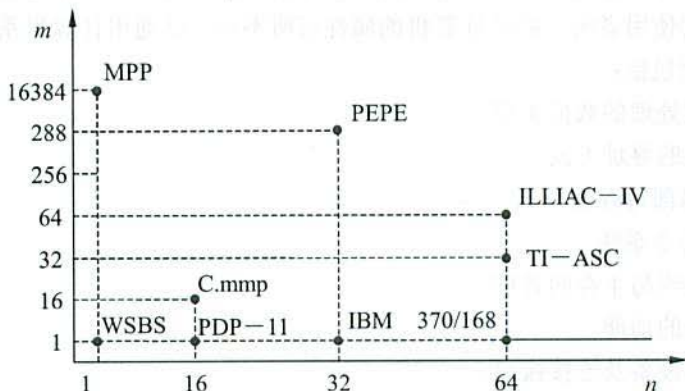


图 1-1 冯式分类法

- 字串行、位串行 (简称 WSBS)。其中 $N=1, M=1$ 。
- 字并行、位串行 (简称 WPBS)。其中 $N=1, M>1$ 。
- 字串行、位并行 (简称 WSBP)。其中 $N>1, M=1$ 。
- 字并行、位并行 (简称 WPBP)。其中 $N>1, M>1$ 。

3. 计算机系统结构与计算机组成的区别

计算机系统结构所解决的问题是计算机系统总体上、功能上需要解决的问题, 而计算机组成要解决的是逻辑上如何具体实现的问题。

比如说指令系统的确定属于计算机系统结构, 而指令的具体实现则属于计算机组成。指令系统中要不要设置乘、除法指令是计算机系统结构要解决的问题, 而一旦决定设置, 具体用什么方法来实现乘、除法指令就属于计算机组成应解决的问题。

主存容量及编址方式的确定属于计算机系统结构,而具体如何构成主存则属于计算机组成。可以想象,即使是系统结构相同的计算机,但具体按此系统结构构成的计算机在实现方法、性能及价格上会有很大差别。

4. 系统结构中并行性的发展

(1) 并行性

并行性包括两个方面:同时性和并发性。同时性是指两个或两个以上的事件在同一时刻发生;并发性是指两个或两个以上的事件在同一时间间隔内连续发生。

充分利用并行性实现计算机的并行处理,可以提高计算机的处理速度。

(2) 分类

从计算机信息处理的步骤和阶段的角度看,并行处理可分为:

- 存储器操作并行。
- 处理器操作步骤并行(流水线处理机)。
- 处理器操作并行(阵列处理机)。
- 指令、任务、作业并行(多处理机、分布处理系统、计算机网络)。

(3) 并行性的发展

从20世纪80年代开始,计算机系统结构有了很大发展,相继出现了精减指令集计算机(RISC)、指令级上并行的超标量处理机、超级流水线处理机、超长指令计算机、多微处理机系统、数据流计算机等。

20世纪90年代以来,计算机系统结构最主要的发展是大规模并行处理(MPP),其中多处理机系统和多计算机系统是研究开发的热点。

1.1.2 存储系统

1. 存储器的层次结构

存储体系结构包括不同层次上的存储器,通过适当的硬件、软件有机地组合在一起形成计算机的存储体系结构。现在大多数人都将高性能计算机的存储体系结构描述成如图1-2所示的3层存储器层次结构。

3层存储器结构是高速缓存(Cache)、主存储器(MM)和辅助存储器(外存储器)。也有人将存储器层次分为4层,即将CPU内部的寄存器也看作是存储器的一个层次。

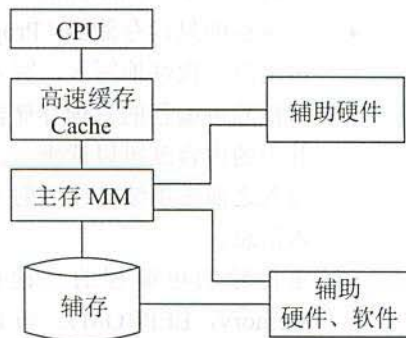
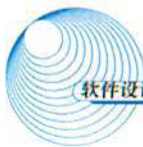


图 1-2 存储器层次结构示意图



有一些简单的计算机没有高速缓存(Cache),则这样的计算机的存储体系就剩下主存和辅存两个层次。

2. 存储器的分类

1) 按存储器所处位置分

按存储器所处的位置可分为内存和外存。

(1) 内存:也称为主存,设在主机内或主机板上,用来存放机器当前运行所需要的程序和数据,以便向CPU提供信息。相对于外存,其特点是容量小速度快。

(2) 外存:也称为辅存,如磁盘、磁带、光盘等,用来存放当前不参加运行的大量信息,在需要时,可把需要的信息调入内存。相对于内存,外存的容量大、速度慢。

2) 按存储器的构成材料分

按构成存储器的材料可分为磁存储器、半导体存储器和光存储器。

(1) 磁存储器:是用磁性介质做成的,如磁芯、磁泡、磁膜、磁鼓、磁带及磁盘等。

(2) 半导体存储器:根据所用元件又可分为双极型和MOS型;根据数据是否需要刷新,又可分为静态(Static Memory)和动态(Dynamic Memory)两类。

(3) 光存储器:如光盘(Optical Disk)存储器。

3) 按存储器的工作方式分

按存储器的工作方式可分为读写存储器和只读存储器。

(1) 读写存储器:既能读取数据也能存入数据的存储器。

(2) 只读存储器:根据数据的写入方式,这种存储器又可细分为ROM、PROM、EPROM、EEPROM等类型。

- 固定只读存储器(Read Only Memory, ROM):这种存储器是在厂家生产时就写好数据的,其内容只能读出,不能改变。一般用于存放系统程序BIOS和用于微程序控制。
- 可编程的只读存储器(Programmable Read Only Memory, PROM):其中的内容可以由用户一次性地写入,写入后不能再修改。
- 可擦除可编程的只读存储器(Erasable Programmable Read Only Memory, EPROM):其中的内容既可以读出,也可以由用户写入,写入后还可以修改。改写的方法是,写入之前先用紫外线照射15~20 min以擦去所有信息,然后再用特殊的电子设备写入信息。
- 电擦除的可编程的只读存储器(Electrically Erasable Programmable Read Only Memory, EEPROM):与EPROM相似,EEPROM中的内容既可以读出,也可以进行改写。只不过这种存储器是用电擦除的方法进行数据的改写。
- 闪速存储器(Flash Memory):简称闪存,闪存的特性介于EPROM和EEPROM之

间, 类似于 EEPROM, 闪存也可使用电信号进行信息的擦除操作。整块闪存可以在数秒内删除, 速度远快于 EPROM。

4) 按访问方式分

按访问方式可分为按地址访问的存储器和按内容访问的存储器。

5) 按寻址方式分

按寻址方式可分为随机存储器、顺序存储器和直接存储器。

(1) 随机存储器 (Random Access Memory, RAM): 这种存储器可对任何存储单元存入或读取数据, 访问任何一个存储单元所需的时间是相同的。

(2) 顺序存储器 (Sequentially Addressed Memory, SAM): 这种存储器访问数据所需要的时间与数据所在的存储位置相关, 磁带是典型的顺序存储器。

(3) 直接存储器 (Direct Addressed Memory, DAM): 是介于随机存取和顺序存取之间的一种寻址方式。磁盘是一种直接存取存储器, 它对磁道的寻址是随机的, 而在一个磁道内, 则是顺序寻址。

3. 相联存储器

相联存储器是一种按内容访问的存储器。其工作原理就是把数据或数据的某一部分作为关键字, 将该关键字与存储器中的每一单元进行比较, 找出存储器中所有与关键字相同的数据字。

相联存储器的结构如图 1-3 所示, 各部件的功能如表 1-1 所示。

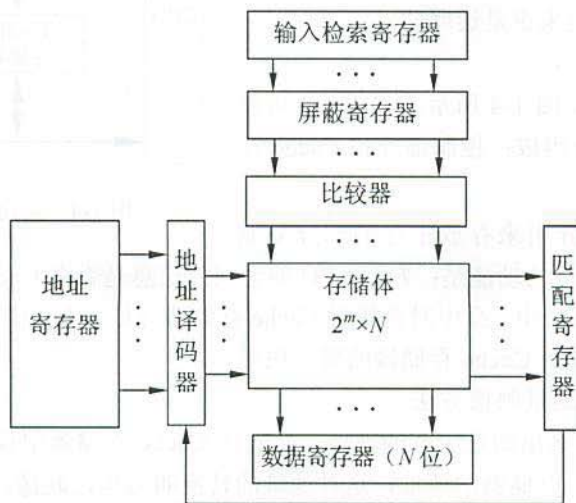


图 1-3 相联存储器的结构框图

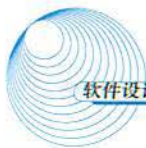


表 1-1 部件功能说明

部件	功能
输入检索寄存器	用来存放要检索的内容(关键字)
屏蔽寄存器	用来屏蔽那些不参与检索的字段
比较器	将检索的关键字与存储体的每一单元进行比较。为了提高速度,比较器的数量应很大。对于位比较器,应每位对应一个,应有 $2^m \times N$ 个。对于字比较器应有 2^m 个
存储体	用于存放信息
匹配寄存器	用来记录比较的结果。它应有 2^m 个二进制位,用来记录 2^m 个比较器的结果,1 为相等(匹配),0 为不相等(不匹配)
数据寄存器	用来存放存储体中某个单元的内容
地址寄存器、地址译码器	使相联存储器具有按地址查找的功能

相联存储器的可用在高速缓冲存储器中;在虚拟存储器中用来作段表、页表或快表存储器;用在数据库和知识库中。

4. 高速缓存 Cache

高速缓存 Cache 是用来存放当前最活跃的程序和数据,作为主存局部域的副本。其特点是:容量一般在几 KB 到几 MB 之间;速度一般比主存快 5~10 倍,由快速半导体存储器构成;其内容是主存局部域的副本,对程序员来说是透明的。

1) 高速缓存的组成

高速缓存的组成如图 1-4 所示。由图 1-4 可以看到,Cache 由两部分组成:控制部分和 Cache 存储器部分。

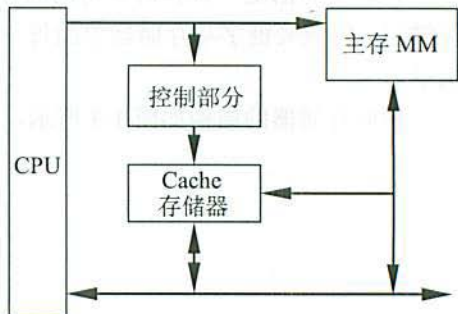


图 1-4 高速缓存的构成框图

Cache 存储器部分用来存放主存的部分复制(副本)信息。控制部分的功能是:判断 CPU 要访问的信息是否在 Cache 存储器中,若在即为命中,若不在则没有命中。命中时直接对 Cache 存储器寻址。未命中时,要按照替换原则,决定主存的一块信息放到 Cache 存储器的哪一块里。

2) 高速缓存中的地址映像方法

在 CPU 工作时,送出的是主存的地址,而应从 Cache 存储器中读写信息。这就需要将主存地址转换成 Cache 存储器的地址,这种地址的转换叫做地址映像。Cache 的地址映像有 3 种方法。

(1) 直接映像。直接映像是指主存的块与 Cache 中块的对应关系是固定的。如图 1-5 所示。

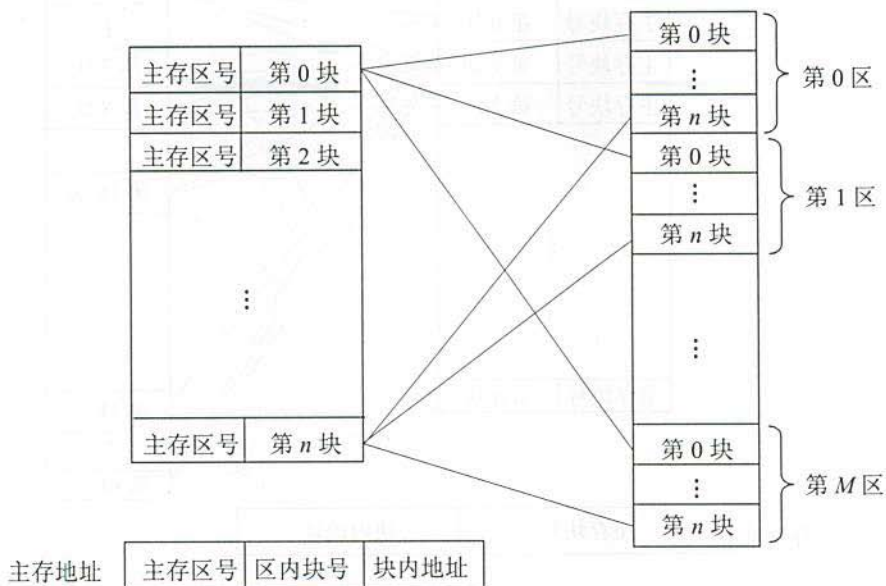


图 1-5 直接映像示意图

在这种映像方式下, 由于主存中的块只能存放在 Cache 存储器的相同块号中。因此, 只要主存地址中的主存区号与 Cache 中的主存区号相同, 则表明访问 Cache 命中。一旦命中, 以主存地址中的区内块号立即可得到要访问的 Cache 存储器中的块。而块内地址就是主存地址中给出的低位地址, 见图 1-5。

直接映像这种方式的优点是地址变换很简单, 只要主存地址中主存区号找到了, 则主存地址中后面的区内块号和块内地址立刻就找到了该块及块内的存储单元。缺点是灵活性差。例如不同区号中块号相同的块无法同时调入 Cache 存储器, 即使 Cache 存储器中有空着的块也只能空着。

(2) 全相联映像。全相联映像的示意图如图 1-6 所示。同样, 主存与 Cache 存储器均分成容量相同的块。这种映像方式允许主存的任一块可以调入 Cache 存储器的任何一个块的空间中。

在地址变换时, 利用主存地址高位表示的主存块号与 Cache 中的主存块号进行比较, 若相同即为命中。这时可根据块号所对应的块就知道要访问的是哪一块。Cache 存储器的块找到后, 块内地址就是主存的低位地址。这便可以读写 Cache 块中的内容。在变换时当找到主

存块号命中时,还必须知道主存的这一块存到了 Cache 的哪一块里面。

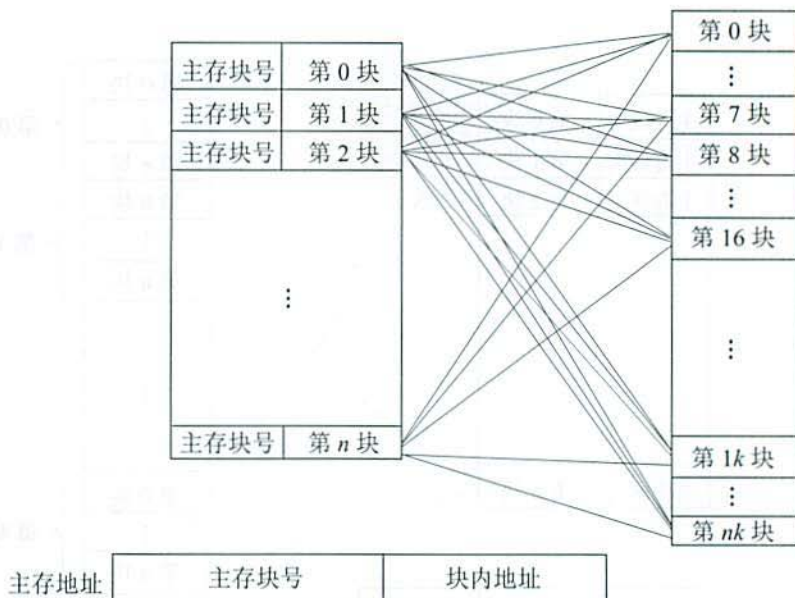


图 1-6 全相联映像示意图

全相联映像的主要优点是主存的块调入 Cache 的位置不受限制,十分灵活。其主要缺点是无法从主存块号中直接获得 Cache 的块号,变换比较复杂,速度比较慢。

(3) 组相联映像。这种方式是前面两种方式的折中。具体做法是将 Cache 中的块再分成组。例如,假定 Cache 有 16 块,再将每两块分为 1 组,则 Cache 就分为 8 组。主存同样分区,每区 16 块,再将每两块分为 1 组,则每区就分为 8 组。

组相联映像就是规定组采用直接映像方式而块采用全相联映像方式。也就是说主存任何区的 0 组只能存到 Cache 的 0 组中,1 组只能存 1 组,依次类推。组内的块则采用全相联映像方式,即一组内的块可以任意存放。也就是说主存一组中的任一块可以存入 Cache 相应组中任一块中。

这种方式下,通过直接映像方式来决定组号,在一组内再用全相联映像方式来决定 Cache 中的块号。由主存地址高位决定主存区号与 Cache 中区号比较可决定是否命中。主存后面的地址即为组号。但组块号要根据全相联映像方式由记录可以决定组内块号。

3) 替换算法

替换算法的目标就是使 Cache 获得最高的命中率。常用算法有:

(1) 随机替换算法。就是用随机数发生器产生一个要替换的块号,将该块替换出去。

(2) 先进先出算法。就是将最先进入 Cache 的信息块替换出去。此法简单但并不是说最先进入的并不经常用。

(3) 近期最少使用算法。这种方法是将近期最少使用的 Cache 中的信息块替换出去。该算法较先进先出算法要好一些。但此法也不能保证过去不常用将来也不常用。

(4) 优化替换算法。这种方法必须先执行一次程序,统计 Cache 的替换情况。有了这样的先验信息,在第二次执行该程序时便可以用最有效的方式来替换,达到最优的目的。

4) 高速缓存的性能分析

若 H 为 Cache 的命中率, t_c 为 Cache 的存取时间, t_m 为主存的访问时间。则 Cache 存储器的等效访问时间 t_a 为:

$$t_a = Ht_c + (1 - H)t_m$$

使用 Cache 存储器比不使用 Cache 存储器 CPU 的访问存储器的速度,提高的倍数 r 可用下式求得:

$$r = t_m / t_a$$

5. 虚拟存储器

虚拟存储器是由主存、辅存、存储管理单元及操作系统中存储管理软件组成的存储系统。在程序员使用该存储系统时,可以使用的内存空间可以远远大于主存的物理空间。但实际上并不存在那么大的主存,故称其为虚拟存储器。虚拟存储器分为:

(1) 页式虚拟存储器,以页为信息传送单位的虚拟存储器。通常一页为几百字节到几 K 字节。为实现页式管理,需建立虚页与实页间的关系表,称为页表;在页表及变换软件的控制下,可将程序的虚拟地址变换为主存的实地址。页式管理的优点是:页表硬件少,查表速度快;主存零头少。其缺点是:分页无逻辑意义,不利于存储保护。

(2) 段式虚拟存储器,以程序的逻辑结构形成的段(如某一独立程序模块、子程序等)作为主存分配依据的一种段式虚拟存储器的管理方法。为实现段式管理,需建立段表;在段地址变换机构及软件的控制下,可将程序的虚拟地址变换为主存的实地址。段式管理的优点是:段的界限分明;支持程序的模块化设计;易于对程序段的编译、修改和保护;便于多道程序的共享。主要缺点:因为段的长度不一,主存利用率不高,产生大量内存碎片,造成浪费;段表庞大,查表速度慢。

(3) 段页式虚拟存储器,页式虚拟存储器和段式虚拟存储器两者相结合的一种管理方式。在这种虚拟存储器中,程序按逻辑结构分段,每一段再分成若干大小固定的页。程序的调入调出是按页进行的,而程序又可按段实现保护。因此,这种管理方式兼有前两者的优点,只是地址变换速度比较慢。

从以上的描述可以看到,虚拟存储器将容量的外存也纳入存储器的管理范围。但在具

体执行程序时需判断程序是否在内存中,若不在(可认为不命中),则需从辅存中调入。这种思路与前面描述的 Cache 中的替换一样。因此,虚拟存储器中的替换算法与前面所述的一样,此处不再说明。

6. 外存储器

外存储器用来存放暂时不用的程序和数据,并且以文件的形式存储。CPU 不能直接访问外存中的程序和数据,只有将其以文件为单位调入主存方可访问。外存储器由磁表面存储器(如磁盘、磁带)及光盘存储器构成。下面介绍两种常用的外存储器。

1) 磁盘存储器

在磁表面存储器中,磁盘的存取速度较快,且具有较大的存储容量,故是目前广泛使用的外存储器。

(1) 磁盘存储器的构成:磁盘存储器由盘片、驱动器、控制器和接口组成。盘片用来存储信息;驱动器用于驱动磁头沿盘面径向运动以寻找目标磁道位置,驱动盘片以额定速率稳定旋转,并且控制数据的写入和读出。控制器接受主机发来的命令,将它转换成磁盘驱动器的控制命令,并实现主机和驱动器之间数据格式的转换及数据传送,以控制驱动器的读/写操作。一个控制器可以控制一台或多台驱动器。接口是主机和磁盘存储器之间的连接逻辑。在微机系统中,通常将磁盘控制器与接口制作在一块插件上,称为磁盘适配卡。

另外,在软盘驱动器中,盘片可以随便拆卸,因而盘片可与驱动器分离。但在多数硬盘存储器中,盘片常密封于驱动器之中,不可分离。

(2) 磁盘存储器的种类:磁盘存储器有两种。一种是以软质聚酯塑料薄片为基体,在基体上涂敷氧化铁磁性材料作为记录介质,称为软盘;另一种是采用硬质基体,在基体上生成一层很薄但很均匀的记录磁层,称为硬盘。

① 软盘:盘片的直径和记录密度可能不同,目前使用的是 3.5 英寸高密度软盘。软盘的盘面封装在一个保护套内,保护套内有一层无纺布,用来防尘,消除静电,保护盘片表面以免划伤。保护套中心的大圆孔是装卡孔,起定位作用。保护套上还有一个读写口、写保护口及索引孔。盘片旋转时磁头通过读写口对盘片进行读/写操作。若要防止软盘中的文件不被修改和删除,应让软盘处于写保护。

为了正确存储信息,将盘片划成许多同心圆,称为磁道,从外到里编号,最外一圈为 0 道,往内道号依次增加。沿径向的单位距离的磁道数称为道密度,单位为 tpi (每英寸磁道数)。将一个磁道沿圆周等分为若干段,每段称为一个扇段或扇区,每个扇区内可存放一个固定长度的数据块,如 512B。磁道上单位距离可记录的比特数称为位密度,单位为 bpi (每英寸比特数)。因为每条磁道上的扇区数相同,而每个扇区的大小又一样,所以每条磁道都记录同

样多的信息。又因为里圈磁道圆周比外圈磁道的圆周小，所以里圈磁道的位密度要比外圈磁道的位密度高。最内圈的位密度称为最大位密度。

访问软盘时应给出软盘驱动器号（A 盘或 B 盘）、磁头号、磁道号、扇区号、交换量（指文件占有扇区数）等寻址信息。

软盘的记录格式是指磁盘表面上信息的存储格式。为了盘结构的互换性和简化系统设计，采用统一的标准记录格式，ISO 已确定将 IBM 记录格式作为国际标准。图 1-7 为 IBM34 系统磁道格式。

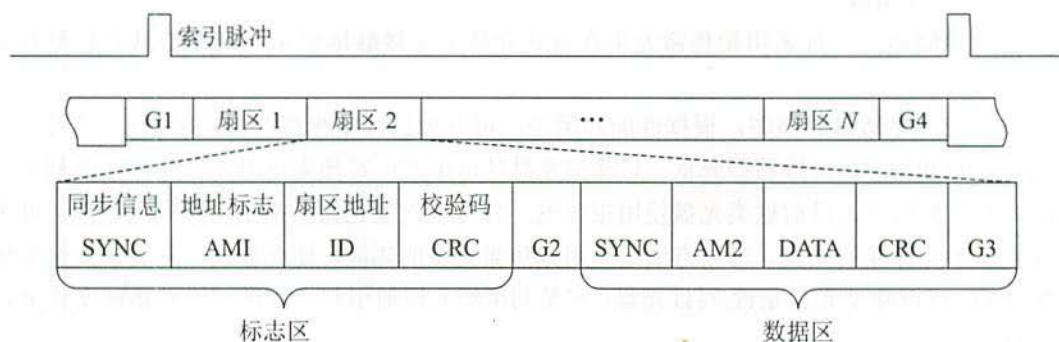


图 1-7 IBM34 系统磁道格式

有的操作系统允许用户使用非标准记录格式，以保护自己软件的产权，防止非法复制。

一般新买来的软盘，在使用前要用磁盘命令进行格式化。格式化分为物理格式化（低级格式化）和逻辑格式化（高级格式化）。物理格式化把磁道化分为若干扇区，每个扇区又化分为标志区和数据区，并将有关信息写入磁盘，但 DATA 段空着。逻辑格式化完成：建立文件目录表、磁盘扇区分配表、磁盘参数表等。

实际上有两种磁盘容量指标。一种是非格式化容量，它是指一个磁盘所能存储的总位数；另一种是格式化容量，它是指各扇区中 DATA 区容量总和，计算公式分别如下：

非格式化容量=面数×（磁道数/面）×内圆周长×最大位密度

格式化容量=面数×（磁道数/面）×（扇区数/道）×（字节数/扇区）

② 硬盘：按盘片是否固定、磁头是否移动等指标，硬盘可分为移动磁头固定盘片的磁盘存储器、固定磁头的磁盘存储器、移动磁头可换盘片的磁盘存储器和温彻斯特磁盘存储器（简称温盘）。按盘片的直径，分为 14 英寸、8 英寸、5.25 英寸、3.5 英寸几种。

一个硬盘驱动器内可装有多片盘片，组成盘片组，每个盘片都配有一个独立的磁头。所有记录面上相同序号的磁道构成一个圆柱面，其编号与磁道编号相同。文件存储在硬盘上时尽可能放在同一圆柱面上，或者放在相邻柱面上，这样可以缩短寻道时间。

硬盘的寻址信息由硬盘驱动号、圆柱面号、磁头号(记录面号)、数据块号(或扇区号)以及交换量组成。

软盘和硬盘在结构和性能上的区别:

- 硬盘转速高、存取速度快;软盘转速低、存取速度慢。
- 硬盘有固定头、固定盘、盘组等结构;软盘都是活动头、可换盘片结构。
- 硬盘是浮动磁头读写,磁头不接触盘片;软盘磁头是接触式读写。
- 硬盘系统价格较贵;软盘价格低、盘片保存使用简便。

2) 光盘存储器

光盘存储器是一种采用聚焦激光束在盘式介质上非接触地记录高密度信息的新型存储装置。

(1) 光盘存储器的类型:根据性能和用途,可分为只读型光盘(CD-ROM)、只写一次型光盘(WORM)和可擦除型光盘。只读型光盘是由生产厂家预先用激光在盘片上蚀刻不能再改写的各种信息,目前这类光盘使用很普遍。只写一次型光盘是指由用户一次写入、可多次读出但不能擦除的光盘。写入方法是利用聚焦激光束的热能,使光盘表面发生永久性变化而实现的。可擦除型光盘是读/写性光盘,它是利用激光照射引起介质的可逆性物理变化来记录信息。

(2) 光盘存储器的组成及特点:光盘存储器由光学、电学和机械部件等组成。其特点是记录密度高;存储容量大;采用非接触式读/写信息(光头距离光盘通常为2mm);信息可长期保存(其寿命达10年以上);采用多通道记录时数据传输率可超过200MB/s;制造成本低;对机械结构的精度要求不高;存取时间较长。

光盘存储器与磁盘存储器的比较:

- 光盘是非接触式读写信息,比磁盘的头盘间距大1万倍左右,所以光盘的耐用性高,使用寿命长。
- 光盘可靠性高,对使用环境要求不高,机械振动上的问题较少,不需要特殊的防震与除尘设备。
- 光盘的记录密度为磁盘的10~100倍,但取数时间慢于磁盘,其读/写速度只有磁盘的几分之一。
- 光盘易于更换,可做成自动换盘装置。

7. 磁盘阵列技术

磁盘阵列是由多台磁盘存储器组成的,一个快速大容量高可靠的外存子系统。现在常见的称为廉价冗余磁盘阵列(RAID)。目前,RAID分为6级,如表1-2所示。

表 1-2 廉价冗余磁盘阵列

RAID 级别	说明
RAID0	0 级廉价冗余磁盘阵列是一种不具备容错能力的阵列。由 N 个磁盘存储器组成的 0 级阵列, 其平均故障间隔时间 (MTBF) 是单个磁盘存储器的 N 分之一, 但数据传输率是单个磁盘存储器的 N 倍
RAID1	1 级廉价冗余磁盘阵列是采用镜像容错改善可靠性的一种磁盘阵列
RAID2	2 级廉价冗余磁盘阵列是采用汉明码作错误检测的一种磁盘阵列
RAID3	3 级廉价冗余磁盘阵列这种磁盘阵列减少了用于检验的磁盘存储器的台数, 从而提高了磁盘阵列的有效容量。一般只有一个检验盘
RAID4	4 级廉价冗余磁盘阵列是一种可独立地对组内各磁盘进行读写的磁盘阵列, 该阵列也只用一个检验盘
RAID5	5 级廉价冗余磁盘阵列该阵列是对 RAID4 的一种改进, 它不设置专门的检验盘。同一台磁盘上既记录数据, 也记录检验信息。这就解决了前面多台磁盘机争用一台检验盘的问题

除此之外, 目前还有 RAID6、RAID7、RAID10 等, 它们均是对前者的改进, 此处不再说明。

1.1.3 CISC/RISC

1. 指令系统的发展

CISC 的含义是复杂指令集计算机。众所周知, 早期的计算机指令系统比较简单, 通常只有十几到几十条指令。随着计算机的发展, 指令系统的也随之发展, 不仅指令条数增加, 而且指令的功能也有了许多增加。其目的是利用增加指令功能和复杂程度缩小汇编与高级语言的差距。另外为了使新老型号的系列 CPU 的指令系统向上兼容, 就需要在新的 CPU 中既要保持老的指令系统, 又要增加新的指令。例如, 80X86 系列 CPU, 每更新一次, 必然增加一些指令。

由于上述原因使得 CPU 的指令系统越来越庞大, 越来越复杂。这就是复杂指令集计算机 (CISC)。由于指令系统的复杂性, 使 CPU 硬件也变得十分复杂, 同时也限制了 CPU 的运行速度。因此, 复杂指令集计算机 (CISC) 在性能上的提高遇到了很大的困难。

2. 精简指令集计算机 (RISC)

人们对典型的 CISC 执行程序中指令使用频度进行统计发现, 指令系统中只有大约 20% 的指令被经常使用, 其使用频度达 80%, 而且这些指令都是一些加、传送、转移等最简单的指令。也就是说大多数的复杂指令只有 20% 的使用概率。

若只保留 20% 的最简单的指令, 使指令尽可能简单, 从而设计一种硬件结构十分简单、



执行速度很高的 CPU。这就是精简指令集计算机 (RISC)。

3. RISC 的特点

RISC 简化了 CPU 的控制器,同时提高了处理速度,它的特点列如下:

- (1) 指令种类少。一般只有十几到几十条简单的指令。
- (2) 指令长度固定,指令格式少。这可使指令译码更加简单。
- (3) 寻址方式少。适合于组合逻辑控制器,便于提高速度。
- (4) 设置最少的访内指令。访问内存比较花时间,尽量少用。
- (5) 在 CPU 内部设置大量的寄存器。使大多数操作在速度很快的 CPU 内部进行。
- (6) 非常适合流水线操作。由于指令简单,并行执行就更易实现。

1.1.4 输入/输出技术

1. 微型计算机中最常用的内存与接口的编址方式

尽管在微型计算机中存在着许多种内存与接口地址的编址方法。但最常见到的是下面将要描述的两种,而且理解了这两种编址方法,再遇到其他的编址方法也就不难理解了。

1) 内存与接口地址独立的编址方法

这种编址方法也有人称为内存与接口地址隔离的编址方法。即在微型机中,内存地址和接口地址是完全独立的两个地址空间,它们是完全独立的并且是相互隔离的。

在使用中,这种编址方式地址很清楚,内存就用于存放程序和数据,而接口就用于寻址外设。所使用的指令也完全不同,用于接口的指令只用于接口读写,其余的指令全都是用于内存的。因此,在编程序或读程序中很易使用和辨认。

这种编址方法的缺点就是用于接口的指令太少、功能太弱。

2) 内存与接口地址统一的编址方法

这种编址方法也有人称为内存与接口地址混合的编址方法。在这种编址方法里,内存地址和接口地址统一在一个公共的地址空间里。也就是说内存和接口共用这些地址。在这些地址空间里拿出某一些地址分配给接口使用,而剩下的就可以归内存所用。那也就是说,地址空间里的每一个地址都可以分配给接口也可以分配给内存使用。但是,分配给内存的只能用于内存,接口绝不允许使用。同样,分配给接口的地址内存也绝不能再使用。

这种编址方法的优点是原则上用于内存的指令全都可以用于接口。这就大大地增强了对接口的操作功能。而且在指令上也不再区分内存或接口指令,也就是说两者用的指令全都是一样的。

该编址方法的缺点就在于整个地址空间被分成两部分,其中一部分分配给接口使用,剩

余的为内存所用，这经常会导致内存地址不连续。再就是用于内存的指令和用于接口的指令是完全一样的，这在读程序时就需根据参数定义表仔细加以辨认。

2. 直接程序控制

在完成外设数据的输入/输出中，整个输入/输出过程是在 CPU 执行程序的控制下完成的。这种方式分为两种情况：

1) 无条件传送

在此情况下，外设总是准备好的，它可以无条件地随时接收 CPU 发来的输出数据，也能够无条件地随时向 CPU 提供需要输入的数据。

2) 程序查询方式

在这种方式下，利用查询方式进行输入/输出，就是通过 CPU 执行程序查询外设的状态，判断外设是否准备好了接收数据或向 CPU 输入的数据。根据这种状态，CPU 有针对性地为外设的输入/输出服务。

通常，一个计算机系统中可以存在着多种不同的外设，如果这些外设是用查询方式工作，则 CPU 应对这些外设逐一进行查询，发现哪个外设准备就绪就对该外设服务。其过程如图 1-8 所示。

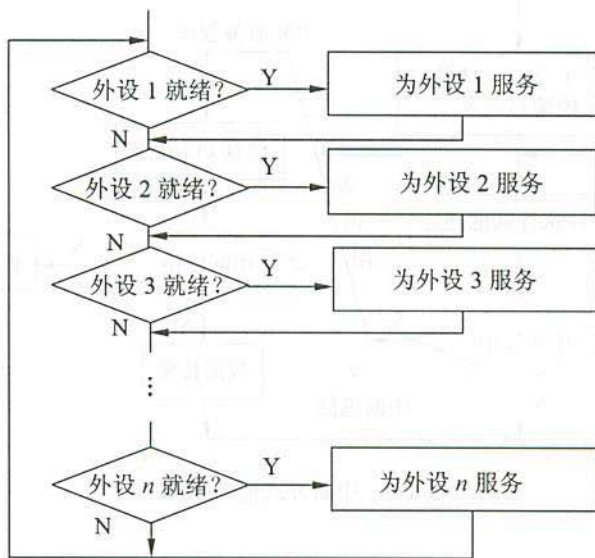


图 1-8 多台外设查询工作

上面对查询方式的描述可以看到，这种工作方式有两大缺点：

(1) 降低了 CPU 的效率。在这种工作方式下, CPU 不做别的事, 只是不停地对外设的状态进行查询。在实际的工程应用中, 对那些慢速的外设在不影响到外设工作时, CPU 可以执行其他任务。

(2) 对外部的突发事件无法做出实时响应。

查询方式的优点在于这种思想很易理解, 同时实现这种工作方式也很容易。

3. 中断方式

由程序控制 I/O 的方法其主要缺点在于 CPU 必须等待 I/O 系统完成数据传输任务, 在此期间 CPU 需定期地查询 I/O 系统的状态, 以确认传输是否完成。因此整个系统的性能严重下降。

为了克服该缺陷, 把中断机制引入到 I/O 传输过程中。CPU 利用中断方式完成数据的输入/输出: 当 I/O 系统与外设交换数据时, CPU 无须等待也不必去查询 I/O 的状态, 而可以抽身出来处理其他任务。当 I/O 系统完成了数据传输后则以中断信号通知 CPU。CPU 保存正在执行程序的现场, 转入 I/O 中断服务程序, 完成与 I/O 系统的数据交换。然后再返回原主程序继续执行。与程序控制方式相比, 中断方式因为 CPU 无须等待而提高了效率。中断方式输入/输出如图 1-9 所示。

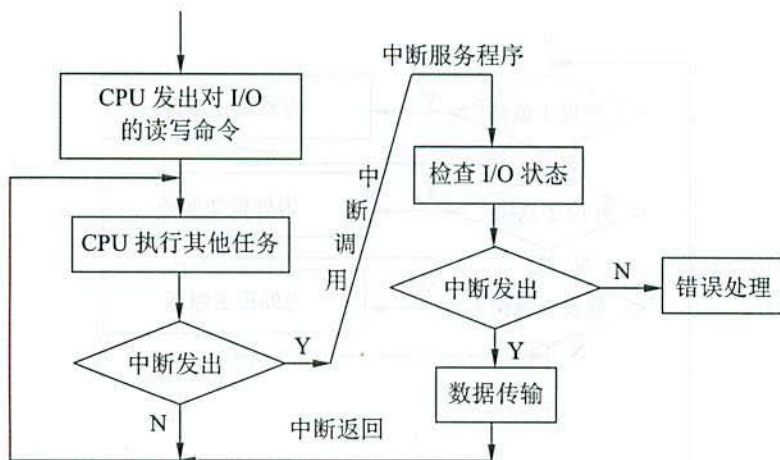


图 1-9 中断方式输入/输出

1) 中断处理方法

在系统中具有多个中断源的情况下, 常用的处理方法有: 多中断信号线法 (Multiple Interrupt Lines)、中断软件查询法 (Software Poll)、菊花链法 (Daisy Chain)、总线仲裁法和中断向量表法。

(1) 多中断信号线法: 每个中断源都有属于自己的一根中断请求信号线向 CPU 提出中断请求。

(2) 中断软件查询法: 当 CPU 检测到一个中断请求信号以后, 即转入到中断服务程序去轮询每个中断源, 以确定是谁发出了中断请求信号。对各个设备的响应优先级由软件设定。如图 1-10 所示。

(3) 菊花链法: 软件查询的缺陷在于花费时间太多。菊花链法实际上是一种硬件查询法。所有的 I/O 模块共享一根共同的中断请求线, 而中断确认信号则以链式在各模块间相连。当 CPU 检测到中断请求信号, 则发出中断确认信号。中断确认信号依次在 I/O 模块间传递, 直到发出请求的模块, 该模块则把它的 ID 送往数据线由 CPU 读取, 如图 1-11 所示。

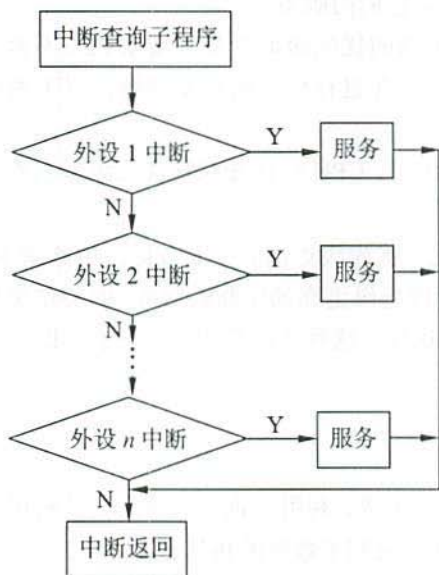


图 1-10 中断软件查询法

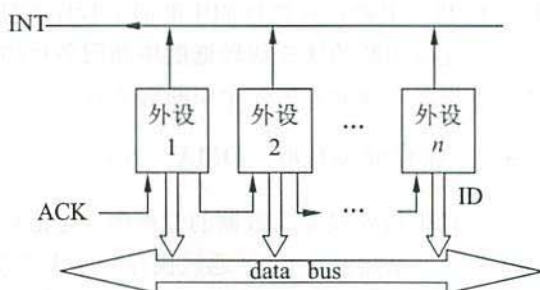


图 1-11 菊花链法

(4) 总线仲裁法: 一个 I/O 设备在发出中断请求之前, 必须先获得总线控制权。所以可由总线仲裁机制来裁定谁可以发出中断请求信号。当 CPU 发出中断响应信号后, 该设备即把自己 ID 发往数据线。

(5) 中断向量表法: 中断向量表用来保存各个中断源的中断服务程序的入口地址。当外设发出中断请求信号 (INTR) 以后, 由中断控制器 (INTC) 确定其中断号, 并根据中断号查找中断向量表来取得其中断服务程序的入口地址, 同时 INTC 把中断请求信号提交给 CPU, 如图 1-12 所示。中断源的优先级由 INTC 来控制。

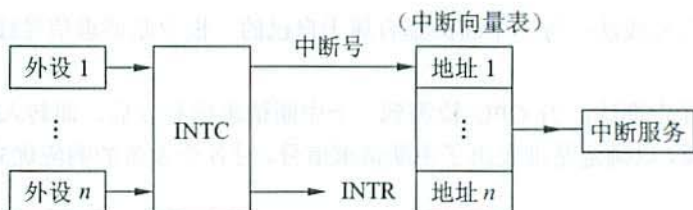


图 1-12 中断向量表法

2) 中断优先级控制

在具有多个中断源的计算机系统中，这些中断源对服务的要求紧迫程度可能不同。在这样的计算机系统中就需要按中断源的轻重缓急来安排对它们的服务。

在中断优先级控制系统中给最紧迫的中断源分配高的优先级而给那些要求相对不紧迫（例如几百 μs 到几 ms ）的中断源分配低一些的优先级。在进行优先级控制时解决以下两种情况。

(1) 当不同优先级的多个中断源同时提出中断请求时，CPU 应优先响应优先级最高的中断源。

(2) 当 CPU 正在对某一个中断源服务时，又有比它优先级更高的中断源提出中断请求，CPU 应能暂时中断正在执行的中断服务程序而转去对优先级更高的中断源服务，服务结束后再回到原先被中断的优先级较低的中断服务程序继续执行。这种情况称为中断嵌套，即一个中断服务程序中嵌套着另一个中断服务程序。

4. 直接存储器存取（DMA）方式

在计算机与外设交换数据的过程中，无论是无条件传送、利用查询方式传送还是利用中断方式传送，都需要由 CPU 通过执行程序来实现。这就限制了数据的传送速度。

直接内存存取（Direct Memory Access, DMA），是指数据在内存与 I/O 设备间的直接成块传送，即在内存与 I/O 设备间传送一个数据块的过程中，不需要 CPU 的任何干涉，只需要 CPU 在过程开始启动（即向设备发出“传送一块数据”的命令）与过程结束（CPU 通过轮询或中断得知过程是否结束和下次操作是否准备就绪）时的处理，实际操作由 DMA 硬件直接执行完成，CPU 在此传送过程中可做别的事情。

DMA 传送的一般过程如图 1-13 所示。

下面对这一过程分别予以介绍

(1) 外设向 DMA 控制器（DMAC）提出 DMA 传送的请求。

(2) DMA 控制器向 CPU 提出请求，其请求信号通常加到 CPU 的保持请求输入端 HOLD 上。

(3) CPU 在完成当前的总线周期后立即对此请求做出响应，CPU 的响应包括两个方面

的内容：一方面 CPU 将有效的保持响应信号 HLDA 输出加到 DMAC 上，告诉 DMAC 它的请求已得到响应；另一方面 CPU 将其输出的总线信号置为高阻，这就意味着 CPU 放弃了对总线的控制权。

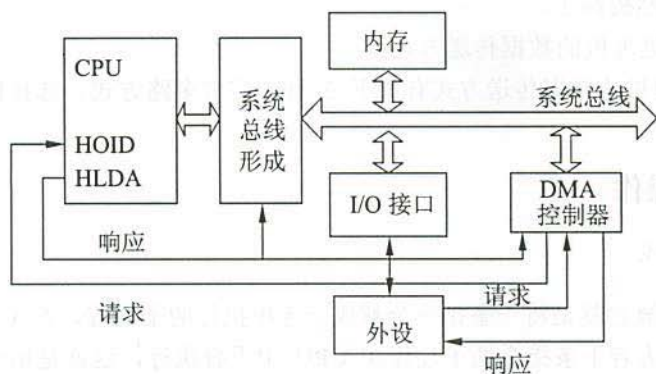


图 1-13 DMA 过程示意图

(4) 此时，DMAC 获得了对系统总线的控制权，开始实施对系统总线的控制。同时向提出请求的外设送出 DMAC 的响应信号，告诉外设其请求已得到响应，现在起准备开始进行数据的传送。

(5) DMAC 送出地址信号和控制信号，实现数据的高速传送。

(6) 当 DMAC 将规定的字节数传送完时，它就将 HOLD 信号变为无效并加到 CPU 上，撤销对 CPU 的请求。CPU 检测到无效的 HOLD 就知道 DMAC 已传送结束，CPU 就送出无效的 HLDA 响应信号，同时重新获得系统总线的控制权，接着 DMA 前的总线周期继续执行下面的总线周期。

在此再强调说明，在 DMA 传送过程中无需 CPU 的干预，整个系统总线完全交给了 DMAC，由它控制系统总线完成数据传送。在 DMA 传送数据时要占用系统总线，根据占用总线的方法不同，DMA 可以分为中央处理器停止法，总线周期分时法和总线周期挪用法等。无论采用哪种方法，在 DMA 传送数据期间，中央处理器不能使用总线。

5. 输入/输出处理机 (IOP)

上面所描述的输入/输出方式适合于外设不多，速度不很高的小型或微型机中，实现起来不是很复杂。在大型计算机中，外设很多，要求计算机的速度很高。采用程序传送、查询、中断或 DMA 均会因输入/输出而造成过大的开销，影响计算机的整体性能。为此，提出采用输入/输出处理机。

1) 输入/输出处理机的功能

输入/输出处理机是一个专用处理机,接在主计算机上,主机的输入/输出操作由它来完成。它根据主机的 I/O 命令,完成对外设数据的输入/输出。既然输入/输出由 IOP 来完成了,主机的工作效率必然提高了。

2) 输入/输出处理机的数据传送方式

输入/输出处理机的数据传送方式有如下 3 种:字节多路方式、选择传送方式和数组多路方式。

1.1.5 流水线操作

1. 指令流水线

指令流水线的概念就是将一条指令分解成一连串执行的子过程,在 CPU 中变一条指令的串行执行子过程为若干条指令的子过程在 CPU 中重叠执行,这就是指令流水线的思路。如果能做到每条指令均分解为 m 个子过程,且每个子过程的执行时间都一样。则利用此条流水线可将一条指令的执行时间由原来的 T 缩短为 T/m 。

1) 流水的基本概念

流水线技术是将一个重复的时序分解成若干个子过程,而每一个子过程都可有效地在其专用功能段上与其他子过程同时执行。流水线技术应用于计算机系统结构的各个方面,在此我们以指令的执行过程为例介绍流水线技术。

前面已举例介绍过,可将指令的执行过程分解成取指令、分析指令和执行指令 3 个子过程。早期指令的执行是顺序方式,即现行指令执行完毕后才开始读取后继指令,这种处理方式控制简单,且比较直观,但在时间安排上不能充分利用各部件。为了提高工作速度,现在的大多数计算机都在不同程度上采取重叠处理方式,重叠的程度取决于存储体与运算部件的多少及控制指令部件的工作方式。

在一次重叠处理时,可将指令的执行过程粗分为分析和执行两个子过程,当第 I 条指令在指令部件中分析完毕后,将进入执行部件去实现指令的操作。此时指令分析部件处于“空闲”状态,若利用这个“空闲”状态对第 $I+1$ 条指令进行分析,使其与第 I 条指令的执行同步进行,也即两条指令在时间上存在着重叠,如图 1-14 所示。

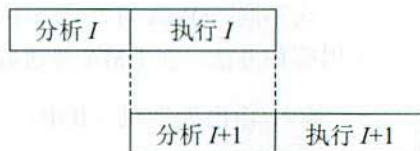


图 1-14 一次重叠处理

若把指令的执行过程进一步细分为取指令、指令译码、取操作数和执行 4 个子过程,并改进运算器的结构以加快其执行子过程,则得到图 1-15 所示的流水处理的时空图,其中的 1, 2, 3, 4, 5 表示要处理的 5 条指令。一次重叠与流水

的区别在于前者将一条指令的执行分为两个子过程,而后者却分为4个或多个子过程。一次重叠可同时执行两条指令,而流水方式则可同时执行多条指令。

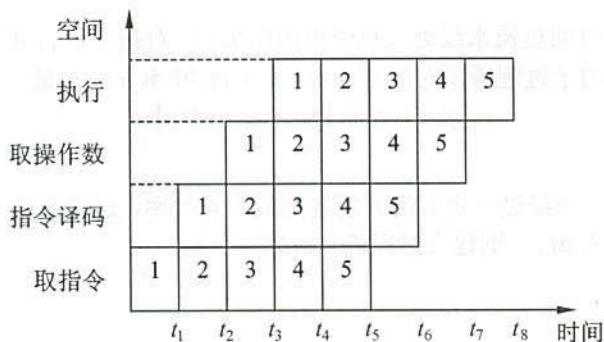


图 1-15 流水处理的时空图

2) 流水技术的特点

- 流水线可分成若干个相互联系子过程。
- 实现子过程的功能所需时间尽可能相等。
- 形成流水处理,需要一段准备时间。
- 指令流发生不能顺序执行时,会使流水过程中断,再形成流水过程则需要时间。

3) 流水线结构的分类

流水线结构的类型众多,并且分类方法各异,常见的几种分类方法如下所述:

(1) 按完成的功能分类

- 单功能流水线:只完成一种固定功能的流水线,如只能实现浮点加。
- 多功能流水线:同一流水线上可有多种连接方式来实现多种功能,如 ASC 运算器中的 8 个可并行工作的功能块,可按不同的连接方式实现浮点加、减或定点乘法运算。

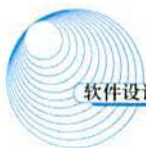
(2) 按同一时间内各段之间的连接方式分类

- 静态流水线:同一时间流水线上的所有功能块只能按同一种运算的连接方式工作。如 ASC 运算器中的 8 个可并行工作的功能块,或都按浮点加、减运算连接,或都按定点乘法运算连接。

- 动态流水线:同一时间流水线上的所有功能块可按不同种运算的连接方式工作。

(3) 按数据表示分类

- 标量流水处理机:只能对标量数据进行流水处理。
- 向量流水处理机:它具有向量指令,可对向量的各元素进行流水处理。



2. 流水线处理机的主要指标

1) 吞吐率

吞吐率是指单位时间里流水线处理机流出的结果数。对指令而言就是单位时间里执行的指令数。如果流水线的子过程所用时间不一样长,则吞吐率 p 应为最长子过程的倒数,即:

$$p = 1 / \max \{ \Delta t_1, \Delta t_2, \dots, \Delta t_m \}$$

2) 建立时间

流水线开始工作,须经过一定时间才能达到最大吞吐率,这就是建立时间。若 m 个子过程所用时间一样,均为 Δt_0 ,则建立时间 $T_0 = m\Delta t_0$ 。

1.1.6 总线结构

1. 总线的定义与分类

广义地讲,任何连接两个以上电子元器件的导线都可以称为总线。通常分为4类:

- (1) 芯片内总线。用于在集成电路芯片内部各部分的连接。
- (2) 元件级总线。用于一块电路板内各元器件的连接。
- (3) 内总线,又称系统总线。用于构成计算机各组成部分(CPU、内存、接口等)的连接。
- (4) 外总线,又称通信总线。用计算机与外设或计算机与计算机的连接或通信。

2. 内总线

内总线有专用内总线和标准内总线。内总线的性能直接影响到计算机的性能。自计算机发明,尤其是微型机诞生以来,内总线的标准已超过百条。常见的内总线标准有:

1) ISA 总线

ISA 是工业标准总线。它向上兼容更早的 PC 总线,在 PC 总线 62 个插座信号的基础上,再扩充另一个 36 个信号的插座构成 ISA 总线。

ISA 总线主要包括 24 个地址线,16 条数据线,控制总线(内存读写、接口读写、中断请求、中断响应、DMA 请求、DMA 响应等), $\pm 5V$ 、 $\pm 12V$ 电源、地线等。

2) EISA 总线

该总线是在 ISA 总线的基础上发展起来的 32 位总线。该总线定义 32 位地址线,32 位数据线,以及其他控制信号线、电源线、地线等共 196 个接点。总线传输速率达 33Mb/s。该总线利用总线插座与 ISA 总线相兼容,插板插在上层为 ISA 总线信号;将插板插在下层便是 EISA 总线。

3) PCI 总线

PCI 总线是目前微型机上广泛采用的内总线。PCI 总线有适于 32 位机的 124 个信号的标

准和适于 64 位机的 188 个信号的标准。PCI 总线的传输速率至少为 133Mb/s, 64 位 PCI 总线的传输速率为 266Mb/s, 具有很高的传输速率。PCI 总线的工作与处理器的工作是相互独立的, 也就是说 PCI 总线时钟与处理器时钟是独立的、非同步的。PCI 总线上设备是即插即用的。

3. 外总线

外总线的标准有七八十种之多, 此处仅介绍下面几种。

1) RS-232C

RS-232C 是一条串行外总线, 其主要特点是: 所需传输线比较少, 最少只需 3 条线(一条发、一条收、一条地线)即可实现全双工通信。传送距离远, 用电平传送为 15m, 电流环传送可达 1km。有多种可供选择的传送速率。采用非归零码负逻辑工作, 电平 $\leq -3V$ 为逻辑 1, 而电平 $\geq +3V$ 为逻辑 0, 具有较好的抗干扰性。

2) SCSI 总线

小型计算机系统接口(SCSI)是一条并行外总线, 广泛用于连接软硬磁盘、光盘、扫描仪等。该接口总线早期是 8 位的, 后来发展到 16 位。传输速率由 SCSI-1 的 5Mb/s 到 16 位的 Ultra2 SCSI 的 80Mb/s。今天的传输速率已高达 320Mb/s。该总线上最多可接 63 种外设。传输距离可达 20m(差分传送)。

3) USB

通用串行总线 USB, 近几年得到了十分广泛的应用。USB 由 4 条信号线组成, 其中两条用于传送数据, 另外两条传送 +5V 容量为 500mA 的电源。可以经过集线器 HUB 进行树状连接, 最多可达 5 层。该总线上可接 127 个设备。USB1.0 有两种传送速率: 低速 1.5Mb/s, 高速为 12Mb/s。USB2.0 的传送速率为 480Mb/s。USB 总线最大的优点还在于它支持即插即用并支持热插拔。

4) IEEE-1394

这是另一条串行外总线, 近几年同样得到十分广泛的应用。IEEE-1394 由 6 条信号线组成, 其中两条用于传送数据, 两条传送控制信号, 另外两条传送 8~40V 容量为 1500mA 的电源。资料上介绍, IEEE-1394 总线上可接 63 个设备。IEEE-1394 的传送速率从 400Mb/s, 800Mb/s, 1600Mb/s 直到 3.2Gb/s。而这种总线最大的优点也在于它支持即插即用并支持热插拔。

1.1.7 多处理机与并行处理

1. 阵列处理机

1) 阵列处理机的概念

在前面已经提到有关并行处理的概念, 在这里专门介绍阵列处理机。阵列处理机又称并

行处理机, 它将重复设置的多个处理单元 (PU) 按一定方式连成阵列, 在单个控制部件 (CU) 控制下, 对分配给自己的数据进行处理, 并行地完成一条指令所规定的操作。这是一种单指令流多数据流 (SIMD) 计算机, 通过资源重复实现并行性。如图 1-16 所示。

2) SIMD 计算机的互连网络

SIMD 计算机的互连网络的设计目标: 结构简单、灵活; 处理单元间信息传送的步骤尽可能少。

(1) 立方体单级互连网络。对于具有 N 个节点的立方体单级互连网络共有 $n = \log_2 N$ 种互联函数, 即

$$\text{Cube}_i(P_{n-1} \cdots P_i \cdots P_1 P_0) = P_{n-1} \cdots \overline{P_i} \cdots P_1 P_0$$

其中 P_i 为用二进制编号的第 i 位, 且 $0 \leq i \leq n-1$, 也就是说, 每一个处理单元只能与其二进制编号的某一位取反编号的处理单元相连接。

(2) PM2I 单级互连网络。这就是“加减 2^i ”单级互连网络, 能实现与 j 号处理单元相连接的处理单元号为 $j \pm 2^i$ 。其互联函数为:

$$\text{PM2}_{+i}(j) = j + 2^i \bmod N$$

$$\text{PM2}_{-i}(j) = j - 2^i \bmod N$$

(3) 混洗交换单级互连网络。这种互联方式的互联函数为:

$$\text{Shuffle}(P_{n-1} P_{n-2} \cdots P_1 P_0) = (P_{n-2} \cdots P_1 P_0 P_{n-1})$$

可见, 将处理单元的二进制编号循环左移一位便是要连接的处理单元的二进制编号。

2. 多处理机

多处理机系统是有多台处理机组成的系统, 每台处理机有属于自己的控制部件, 可以执行独立的程序, 共享一个主存储器和所有的外部设备。它是多指令流多数据流计算机。在多处理机系统中, 机间的互连技术决定着多处理机的性能。多处理机之间的互连, 要满足高频带、低成本、连接方式的多样性以及在不规则通信情况下连接的无冲突性。

1) 多处理机按其构成的分类

(1) 异构型 (非对称型) 多处理机系统: 是由多个不同类型或可完成不同功能的处理机组成, 按照作业要求的顺序, 利用时间重叠技术, 依次对它们的多个任务进行处理, 各自完成规定的功能操作。

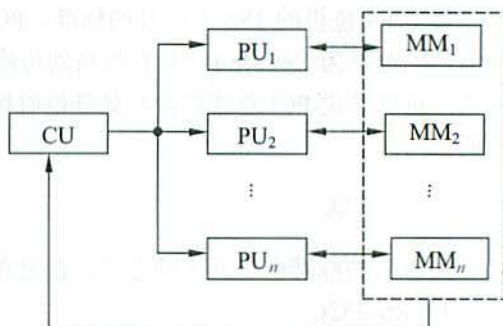


图 1-16 SIMD 计算机

(2) 同构型(对称型)多处理机系统:是由多个同类型或可完成同等功能的处理机组成,同时处理同一作业中能并行执行的多个任务。

(3) 分布式处理系统:是把若干台具有独立功能的处理机相互连接起来,在操作系统的控制下,统一协调地工作,是最少依赖集中的程序、数据或硬件的系统。

2) 多处理机系统的结构

按机间的互连结构,有如下4种多处理机结构。

(1) 总线式结构:总线结构是一种最简单的结构形式,它把处理机与I/O之间的通信方式引入到处理机之间。总线式结构中有单总线结构、多总线结构、分级式总线、环式总线等多种。在此只介绍单总线结构和多总线结构。

在单总线结构中,处理机和外部设备通过自身的接口用一套总线相连,如图1-17所示。在单总线结构中,同一时间内只允许一对处理机或设备之间进行信息的传送,当处理机的个数较多时,系统性能会受到严重影响,为此,必须采用改进方案——多总线结构。

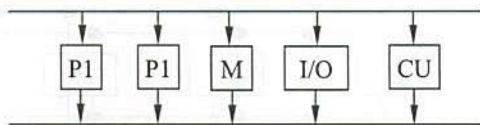


图 1-17 单总线结构

在多总线结构中,可以设置多套总线,如处理机总线、存储器总线、I/O 总线等,以增加处理机之间的通信线路,提高处理机之间的传送效率,如图1-18所示,其中,SBC为总线控制器,I/O C为I/O通道控制器。

(2) 交叉开关结构:交叉开关结构是设置一组织横开关阵列,把横向的处理机P及I/O通道与纵向的存储器M连接起来,如图1-19所示。

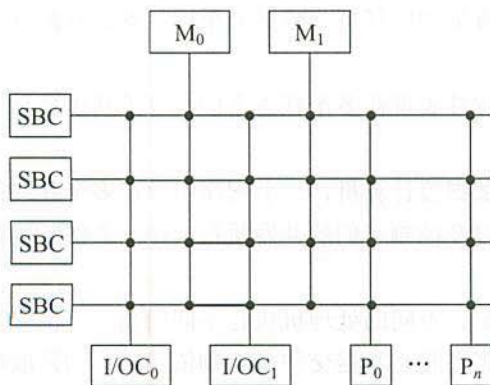


图 1-18 EPOS 多总线系统结构

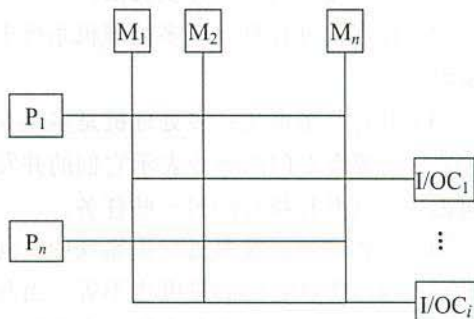


图 1-19 交叉开关式结构

(3) 多端口存储器结构:在多端口存储器结构中,将多个多端口存储器的对应端口连在一起,每一个端口负责一个处理机P及I/O通道的访问存储器的要求。如图1-20为四端口存

储器的系统结构。

(4) 开关枢纽式结构：在开关枢纽式结构中，有多个输入端和多个输出端，在它们之间切换，使输入端有选择地与输出端相连。因为有多多个输入端，所以存在互连要求上的冲突。为此加入一个具有分解冲突的部件，称为仲裁单元。仲裁单元与在一个输入端和多个输出端之间进行转换的开关单元一起构成一个基本的开关枢纽。任何互连网络都是由一个或多个开关枢纽组成，如图 1-21 所示。

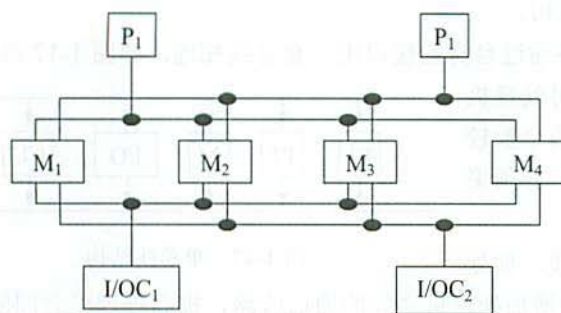


图 1-20 四端口存储器系统结构

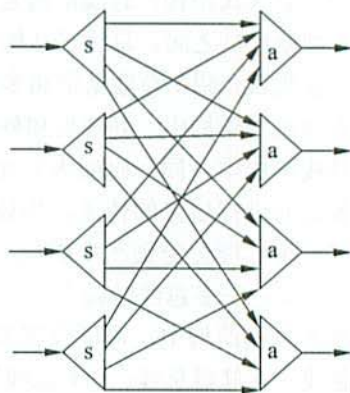


图 1-21 总线和交叉开关的枢纽形式

3) 多处理机系统的特点

(1) 结构灵活性：多处理机系统可以把要并行处理的任务、数据乃至标量都进行并行处理，所以具有较强的通用性及灵活性。

(2) 程序的并行性：在多处理机系统中，并行性表现在多个任务之间，可以利用多种途径实现并行。

(3) 并行任务派生：多处理机是多指令流多数据流计算机，一个程序中存在多个并发的程序段，需要有专门的指令表示它们的并发关系以及控制它们的并发执行，使一个任务执行时可派生与其并行执行的另一些任务。

(4) 进程同步：在多处理机系统中，同一时刻，不同的处理机执行不同的指令。由于执行时间不等，所以它们的进度也不等。当并发程序之间有数据交往或控制依赖时，则采取特殊的同步措施，使它们所包含的指令之间保持程序要求的正确顺序。

(5) 资源分配和任务调度：因为多处理机执行并发任务，所需处理机的数目不固定，各个处理机进入或退出任务时所需资源的情况也很复杂，所以资源分配和任务调度的好坏直接影响整个系统的效率。

3. 并行处理机

与采用流水结构的单机系统都是单指令流多数据流计算机，但它们也有区别，并行处理机采用资源重复技术，而采用流水结构的单机系统则采用时间重叠技术。

1) 并行处理机的两种典型结构

(1) 具有分布存储器的并行处理机结构

具有分布存储器的并行处理机结构(如图 1-22 所示)中有两类存储器：一类存储器是附属于主处理机，主处理机实现整个并行处理机的管理，在其附属的存储器内常驻操作系统。另一类是分布在各个处理单元 PE 上的存储器(即 PEM)，这类存储器用来保存程序和数据。每个处理单元只与附属于自身的存储器直接相连，而各处理单元之间的通信则采用互网络 ICN 交换数据。

说明：图中 PE 为处理单元；CU 为控制部件；PEM 为局部存储器；ICN 为互网络。

(2) 具有共享存储器的并行处理机结构

具有共享存储器的并行处理机结构(如图 1-23 所示)中，将若干个存储器构成统一的并行处理机存储器，通过互网络 ICN 为整个并行系统的所有处理单元共享。

说明：图中 PE 为处理单元；CU 为控制部件；M 为共享存储器；ICN 为互网络。

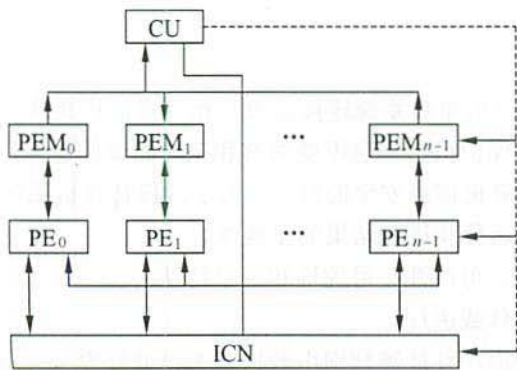


图 1-22 具有分布存储器的并行处理机结构

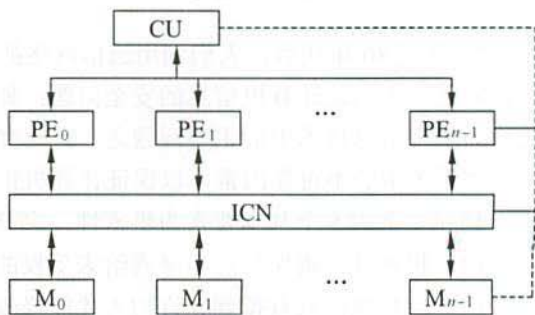


图 1-23 具有共享存储器的并行处理机结构

上述两种结构的共同特点是在整个系统中设置多个处理单元，各个处理单元按照一定的连接方式交换信息，在统一的控制部件作用下，各自对分配来的数据并行地完成同一条指令所规定的操作。

2) 并行处理机的特点

(1) 资源重复：并行处理机中的各个处理单元可以对向量所包含的各个分量同时进行运

算;另外,每个处理单元可以承担多种处理功能。所以,增加处理单元的个数,以提高并行处理机的运算速度。

(2) 连接模式:并行处理机的各个处理单元之间通过互连网络交换数据,互连网络的不同拓扑结构直接决定了并行处理机的结构。

(3) 专用性:并行处理机直接与一定的算法相联系,所以它具有专用性。

(4) 复合性:并行处理机的效率体现在向量数组处理上,所以整个系统是由三部分复合起来的一个多机系统:多个处理单元组成阵列并行地处理向量;功能极强的控制部件是一台标量处理机;系统的管理功能则由高性能单处理机完成。

1.2 安全性、可靠性与系统性能评测基础知识

1.2.1 计算机安全概述

计算机安全是指计算机资源的安全,是要保证这些计算机资源不受自然和人为的有害因素的威胁和危害。计算机资源是由系统资源和信息资源两大部分组成。系统资源包括硬件、软件、配套设备设施、有关文件资料,还可以包括有关的服务系统和业务工作人员。信息资源包括计算机系统中存储、处理和传输的大量各种各样的信息。

1. 信息安全的基本要素

20 世纪 80 年代后,人们利用通信网络把孤立的单机系统连接起来,相互通信和共享资源。随之而来的是计算机信息的安全问题:如何保护机密信息不受黑客和工业间谍的入侵,已成为未来信息技术中的主要问题之一。研究计算机信息安全的目的是:为了改善计算机系统和应用系统中的不可靠因素,以保证计算机正常运行和运算结果的正确性。

信息安全的 5 个基本要素为机密性、完整性、可用性、可控性和可审计性。

(1) 机密性:确保信息不暴露给未授权的实体或进程。

(2) 完整性:只有得到允许的人才能修改数据,并能够判别出数据是否已被篡改。

(3) 可用性:得到授权的实体在需要时可访问数据。

(4) 可控性:可以控制授权范围内的信息流向及行为方式。

(5) 可审查性:对出现的安全问题提供调查的依据和手段。

计算机安全是一个涵盖非常广的课题,既包括硬件、软件和技术,还包括安全规划、安全管理和安全监督。计算机安全可包括安全管理、通信与网络安全、密码学、安全体系及模型、容错与容灾、涉及安全的应用程序及系统开发、法律、犯罪及道德规范等领域。

其中安全管理是非常重要的,作为信息系统的管理部门应根据管理原则和该系统处理数

据的保密性,制定相应的管理制度或规范。例如,根据工作的重要程度确定系统的安全等级,根据确定的安全等级确定安全管理的范围,制定相应的机房管理制度、操作规程、系统维护措施以及应急措施等。

2. 计算机的安全等级

计算机系统三类安全性指技术安全性、管理安全性、政策法律安全性。但是,一个安全产品的购买者如何知道产品的设计是否符合规范,是否能解决计算机网络的安全问题,不同组织机构各自都制定了一套安全评估准则。一些重要的安全评估准则有:

(1) 美国国防部和国家标准局推出的《可信计算机系统评估准则》(TCSEC)。

(2) 加拿大的《可信计算机产品评估准则》(CTCPEC)。

(3) 美国制定的《联邦(最低安全要求)评估准则》(FC)。

(4) 欧洲英、法、德、荷4国国防部门信息安全机构联合制定的《信息技术安全评估准则》(ITSEC),该准则事实上已成为欧盟各国使用的共同评评标准。

(5) 美国制定的《信息技术安全通用评估准则》(简称CC标准),国际标准组织(ISO)于1996年批准CC标准以“ISO/IEC 15408-1999”为名称正式列入国际标准系列。

其中美国国防部和国家标准局的《可信计算机系统评测标准》TCSEC/TDI将系统划分为4组7个等级,如表1-3所示。

表 1-3 安全性的级别

组	安全级别	定义
1	A1	可验证安全设计:提供B3级保护同时给出系统的形式化隐秘通道分析,非形式化代码一致性验证
2	B3	安全域:该级的TCB必须满足访问监控器的要求,提供系统恢复过程
	B2	结构化安全保护:建立形式化的安全策略模型,并对系统内的所有主体和客体实施自主访问和强制访问控制
	B1	标记安全保护:对系统的数据加以标记,并对标记的主体和客体实施强制存取控制
3	C2	受控访问控制:实际上是安全产品的最低档次,提供受控的存取保护,存取控制以用户为单位
	C1	只提供了非常初级的自主安全保护,能实现对用户和数据的分离,进行自主存取控制,数据的保护以用户组为单位
4	D	最低级别,保护措施很小,没有安全功能

3. 安全威胁

随着信息交换的激增,安全威胁所造成的危害越来越被受到重视,因此对信息保密的需

求也从军事、政治和外交等领域迅速扩展到民用和商用领域。所谓安全威胁是指某个人、物、事件对某一资源的机密性、完整性、可用性或合法性所造成的危害。某种攻击就是威胁的具体实现。安全威胁分为两类：故意（如黑客渗透）；偶然（如信息发往错误的地址）。

典型的安全威胁举例如表 1-4 所示。

表 1-4 典型的安全威胁

威胁	说明
授权侵犯	为某一特权使用一个系统的人却将该系统用作其他未授权的目的
拒绝服务	对信息或其他资源的合法访问被无条件地拒绝，或推迟与时间密切相关的操作
窃听	信息从被监视的通信过程中泄漏出去
信息泄露	信息被泄漏或暴露给某个未授权的实体
截获/修改	某一通信数据项在传输过程中被改变、删除或替代
假冒	一个实体（人或系统）假装成另一个实体
否认	参与某次通信交换的一方否认曾发生过此次交换
非法使用	资源被某个未授权的人或者未授权的方式使用
人员疏忽	一个授权的人为了金钱或利益或由于粗心将信息泄露给未授权的人
完整性破坏	通过对数据进行未授权的创建、修改或破坏，使数据的一致性受到损坏
媒体清理	信息被从废弃的或打印过的媒体中获得
物理入侵	一个入侵者通过绕过物理控制而获得对系统的访问
资源耗尽	某一资源（如访问端口）被故意超负荷地使用，导致其他用户的服务被中断

4. 影响数据安全的因素

影响数据安全的因素有内部和外部两类。

（1）内部因素。可采用多种技术对数据加密；制定数据安全规划；建立安全存储体系，包括容量、容错数据保护、数据备份等；建立事故应急计划和容灾措施；重视安全管理，制定数据安全管理制度。

（2）外部因素。可将数据分成不同的密级，规定外部使用人员的权限。设置身份认证、密码、口令、指纹、声纹笔迹等多种认证。设置防火墙，为计算机建立一道屏障，防止外部入侵破坏数据。建立入侵检测、审计和追踪，对计算机进行防卫。同时，也包括计算机的物理环境的保障、防辐射、防水、防火等外部防灾措施。

1.2.2 加密技术

加密技术是最常用的安全保密手段，数据加密技术的关键在于：加密/解密算法和密钥管理。加密技术包括两个元素：算法和密钥。数据加密的基本过程就是对原来为明文的文件或数据按某种加密算法进行处理，使其成为不可读的一段代码，通常称为“密文”。

只能在输入相应的密钥之后才能显示出原来内容, 通过这样的途径来达到保护数据不被窃取。

数据加密和数据解密是一对逆过程, 数据加密是用加密算法 E 和加密密钥 K_1 , 将明文 P 变换成密文 C , 记为:

$$C = E_{K_1}(P)$$

数据解密是数据加密的逆过程, 解密算法 D 和解密密钥 K_2 , 将密文 C 变换成明文 P , 记为:

$$P = D_{K_2}(C)$$

在安全保密中, 可通过适当的密钥加密技术和管理机制来保证网络的信息通信安全。密钥加密技术的密码体制分为对称密钥体制和非对称密钥体制两种。相应地, 对数据加密的技术分为两类, 即对称加密(私人密钥加密)和非对称加密(公开密钥加密)。对称加密采用了对称密码编码技术, 它的特点是文件加密和解密使用相同的密钥, 即加密密钥也可以用作解密密钥, 这种方法在密码学中叫做对称加密算法, 对称加密算法使用起来简单快捷, 密钥较短, 且破译困难。对称加密以数据加密标准(Data Encryption Standard, DES)算法为典型代表。非对称加密通常以 RSA(Rivest Shamir Adleman)算法为代表。对称加密的加密密钥和解密密钥相同, 而非对称加密的加密密钥和解密密钥不同, 加密密钥可以公开而解密密钥需要保密。

1. 对称加密技术

对称加密的体制模型如图 1-24 所示。

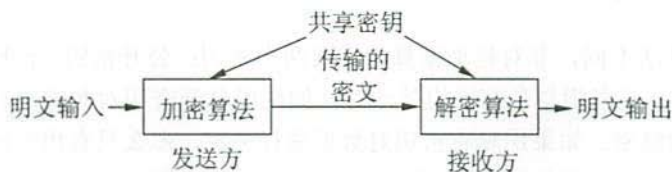


图 1-24 对称加密体制模型

目前常用的对称加密算法有:

(1) 数据加密标准算法

DES 作为联邦信息处理第 46 号标准(FIPS PUB 46), 1994 年美国国家标准和技术局(NIST)再次肯定 DES 以 FIPS PUB 46-2 的名义共联邦再使用 5 年。DES 主要采用替换和移位的方法加密。它用 56 位密钥对 64 位二进制数据块进行加密, 每次加密可对 64 位的输入

数据进行 16 轮编码, 经一系列替换和移位后, 输入的 64 位原始数据转换成完全不同的 64 位输出数据。DES 算法仅使用最大为 64 位的标准算术和逻辑运算, 运算速度快, 密钥生产容易, 适合于在当前大多数计算机上用软件方法实现, 同时也适合于在专用芯片上实现。

(2) 三重 DES (3DES, 或称 TDEA)

在 DES 的基础上采用三重 DES, 即用两个 56 位的密钥 K1、K2, 发送方用 K1 加密, K2 解密, 再使用 K1 加密。接收方则使用 K1 解密, K2 加密, 再使用 K1 解密, 其效果相当于将密钥长度加倍。1999 年 TDEA 合并到加密标准中, 文献号为 FIPS PUB46-3。

(3) RC-5 (Rivest Cipher 5)

RC-5 是由 Ron Rivest (公钥算法的创始人之一) 在 1994 年开发出来的。RC-5 是在 RCF2040 中定义的, RSA 数据安全公司的很多产品都使用了 RC-5。

(4) 国际数据加密算法 (International Data Encryption Adleman, IDEA)

除了数据加密标准 (DES), 另一个对称密钥加密系统是国际数据加密算法 (IDEA), 它比 DES 的加密性好, 而且对计算机功能要求也没有那么高。IDEA 是瑞士的著名学者提出的, 它在 1990 年正式公布并在以后得到增强。开始该算法命名为 PES (Proposed Encryption Standard), 1992 年被命名为 IDEA, 目前, IDEA 被认为是相对安全的分组密码算法。

这种算法是在 DES 算法的基础上发展出来的, 类似于三重 DES。发展 IDEA 也是因为感到 DES 具有密钥太短等缺点, 已经过时。IDEA 的密钥为 128 位, 这么长的密钥在今后若干年内应该是安全的。类似于 DES, IDEA 算法也是一种数据块加密算法, 它设计了一系列加密轮次, 每轮加密都使用从完整的加密密钥中生成的一个子密钥。IDEA 加密标准由 PGP (Pretty Good Privacy) 系统使用。

2. 非对称加密技术

与对称加密算法不同, 非对称加密算法需要两个密钥: 公开密钥 (publickey) 和私有密钥 (privatekey)。公开密钥与私有密钥是一对, 如果用公开密钥对数据进行加密, 只有用对应的私有密钥才能解密; 如果用私有密钥对数据进行加密, 那么只有用对应的公开密钥才能解密。因为加密和解密使用的是两个不同的密钥, 所以这种算法叫作非对称加密算法。

非对称加密有两个不同的体制, 如图 1-25 所示。

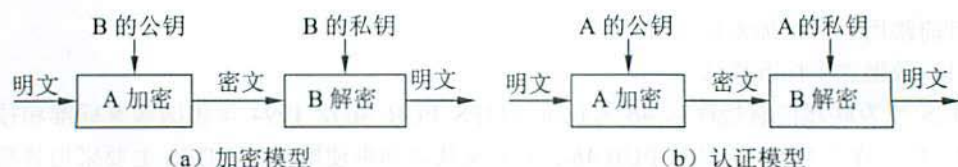


图 1-25 非对称加密体制模型

非对称加密算法实现机密信息交换的基本过程是:甲方生成一对密钥并将其中的一把作为公用密钥向其他方公开;得到该公用密钥的乙方使用该密钥对机密信息进行加密后再发送给甲方;甲方再用自己保存的另一把专用密钥对加密后的信息进行解密。甲方只能用其专用密钥解密由其公用密钥加密后的任何信息。

非对称加密算法的保密性比较好,它消除了最终用户交换密钥的需要,但加密和解密花费时间长、速度慢,它不适合于对文件加密而只适用于对少量数据进行加密。

3. 密钥管理

密钥是有生命周期的,它包括密钥和证书的有效时间,以及已撤销密钥和证书的维护时间等。密钥既然要求保密,这就涉及到密钥的管理问题,管理不好,密钥同样可能被无意识地泄露,并不是有了密钥就高枕无忧,任何保密也只是相对的,是有时效的。密钥管理主要是指密钥对的安全管理,包括密钥产生、密钥备份、密钥恢复和密钥更新等。

1) 密钥产生

密钥对的产生是证书申请过程中重要的一步,其中产生的私钥由用户保留,公钥和其他信息则交于 CA 中心进行签名,从而产生证书。根据证书类型和应用的不同,密钥对的产生也有不同的形式和方法。对普通证书和测试证书,一般由浏览器或固定的终端应用来产生,这样产生的密钥强度较小,不适合应用于比较重要的安全网络交易。而对于比较重要的证书,如商家证书和服务器证书等,密钥对一般由专用应用程序或 CA 中心直接产生,这样产生的密钥强度大,适合于重要的应用场合。

另外,根据密钥的应用不同,也可能会有不同的产生方式。比如签名密钥可能在客户端或 RA 中心产生,而加密密钥则需要在 CA 中心直接产生。

2) 密钥备份和恢复

在一个 PKI (Public Key Infrastructure, 公开密钥体系) 系统中,维护密钥对的备份至关重要,如果没有这种措施,当密钥丢失后,将意味着加密数据的完全丢失,对于一些重要数据,这将是灾难性的。所以,密钥的备份和恢复也是 PKI 密钥管理中的重要一环。换句话说,即使密钥丢失,使用 PKI 的企业和组织必须仍能够得到确认,受密钥加密保护的重要信息也必须能够恢复。当然,不能让一个独立的个人完全控制最重要的主密钥,否则将引起严重后果。

企业级的 PKI 产品至少应该支持用于加密的安全密钥的存储、备份和恢复。密钥一般用口令进行保护,而口令丢失则是管理员最常见的安全疏漏之一。所以,PKI 产品应该能够备份密钥,即使口令丢失,它也能够让用户在一定条件下恢复该密钥,并设置新的口令。

3) 密钥更新

如果用户可以一次又一次地使用同样密钥与别人交换信息,那么密钥也同其他任何密码一样存在着一定的安全性,虽然说用户的私钥是不对外公开的,但是也很难保证私钥长期的

保密性, 很难保证长期以来不被泄露。如果某人偶然地知道了用户的密钥, 那么用户曾经和另一个人交换的每一条消息都不再是保密的了。另外使用一个特定密钥加密的信息越多, 提供给窃听者的材料也就越多, 从某种意义上讲也就越不安全了。

对每一个由 CA 颁发的证书都会有有效期, 密钥对生命周期的长短由签发证书的 CA 中心来确定, 各 CA 系统的证书有效期限有所不同, 一般大约为 2~3 年。当用户的私钥被泄漏或证书的有效期快到时, 用户应该更新私钥。这时用户可以废除证书, 产生新的密钥对, 申请新的证书。

4) 多密钥的管理

假设在某机构中有 100 个人, 如果他们任意两人之间可以进行秘密对话, 那么总共需要多少密钥呢? 每个人需要知道多少密钥呢? 也许很容易得出答案, 如果任何两个人之间要不同的密钥, 则总共需要 4950 个密钥, 而且每个人应记住 99 个密钥。如果机构的人数是 1000、10000 人或更多, 这种办法就显然过于愚蠢了, 管理密钥将是一件非常困难的事情。为此需要研究并开发用于创建和分发密钥的加密安全的方法。

Kerberos 提供了一种解决这个较好方案, 它是由 MIT 发明的, 使保密密钥的管理和分发变得十分容易, 但这种方法本身还存在一定的缺点。为能在因特网上提供一个实用的解决方案, Kerberos 建立了一个安全的、可信任的密钥分配中心 (Key Distribution Center, KDC), 每个用户只要知道一个和 KDC 进行会话的密钥就可以了, 而不需要知道成百上千个不同的密钥。

1.2.3 认证技术

1. 认证技术概述

认证技术主要解决网络通信过程中通信双方的身份认可。认证的过程涉及到加密和密钥交换。通常, 加密可使用对称加密、不对称加密及两种加密方法的混合方法。认证方一般有账户名/口令认证、使用摘要算法认证、基于 PKI (Public Key Infrastructure, 公钥架构) 的认证。

一个有效的 PKI 系统必须是安全的和透明的, 用户在获得加密和数字签名服务时, 不需要详细地了解 PKI 的内部运作机制。在一个典型、完整和有效的 PKI 系统中, 除证书的创建和发布, 特别是证书的撤销, 一个可用的 PKI 产品还必须提供相应的密钥管理服务, 包括密钥的备份、恢复和更新等。没有一个好的密钥管理系统, 将极大影响一个 PKI 系统的规模、可伸缩性和在协同网络中的运行成本。在一个企业中, PKI 系统必须有能力为一个用户管理多对密钥和证书; 能够提供安全策略编辑和管理工具, 如密钥周期和密钥用途等。

PKI 是一种遵循既定标准的密钥管理平台, 它能够为所有网络应用提供加密和数字签名

等密码服务及所必需的密钥和证书管理体系。简单来说, PKI 就是利用公钥理论和技术建立的提供安全服务的基础设施。PKI 技术是信息安全技术的核心, 也是电子商务的关键和基础技术。PKI 的基础技术包括加密、数字签名、数据完整性机制、数字信封、双重数字签名等。完整的 PKI 系统必须具有权威认证机构(CA)、数字证书库、密钥备份及恢复系统、证书作废系统、应用接口(API)等基本构成部分。

(1) 认证机构: 即数字证书的申请及签发机关, CA 必须具备权威性的特征。

(2) 数字证书库: 用于存储已签发的数字证书及公钥, 用户可由此获得所需的其他用户的证书及公钥。

(3) 密钥备份及恢复系统: 如果用户丢失了用于解密数据的密钥, 则数据将无法被解密, 这将造成合法数据丢失。为避免这种情况, PKI 提供备份与恢复密钥的机制。但须注意, 密钥的备份与恢复必须由可信的机构来完成。并且, 密钥备份与恢复只能针对解密密钥, 签名私钥为确保其唯一性而不能够做备份。

(4) 证书作废系统: 证书作废处理系统是 PKI 的一个必备的组件。与日常生活中的各种身份证件一样, 证书有效期以内也可能需要作废, 原因可能是密钥介质丢失或用户身份变更等。为实现这一点, PKI 必须提供作废证书的一系列机制。

(5) 应用接口: PKI 的价值在于使用户能够方便地使用加密、数字签名等安全服务, 因此一个完整的 PKI 必须提供良好的应用接口系统, 使得各种各样的应用能够以安全、一致、可信的方式与 PKI 交互, 确保安全网络环境的完整性和易用性。

PKI 采用证书进行公钥管理, 通过第三方的可信任机构(认证中心, 即 CA), 把用户的公钥和用户的其他标识信息捆绑在一起, 其中包括用户名和电子邮件地址等信息, 以在 Internet 网上验证用户的身份。PKI 把公钥密码和对称密码结合起来, 在 Internet 网上实现密钥的自动管理, 保证网上数据的安全传输。

因此, 从大的方面来说, 所有提供公钥加密和数字签名服务的系统, 都可归结为 PKI 系统的一部分, PKI 的主要目的是通过自动管理密钥和证书, 为用户建立起一个安全的网络运行环境, 使用户可以在多种应用环境下方便地使用加密和数字签名技术, 从而保证网上数据的机密性、完整性、有效性。数据的机密性是指数据在传输过程中, 不能被非授权者偷看; 数据的完整性是指数据在传输过程中不能被非法篡改; 数据的有效性是指数据不能被否认。

PKI 发展的一个重要方面就是标准化问题, 它也是建立互操作性的基础。目前, PKI 标准化主要有两个方面: 一是 RSA 公司的公钥加密标准 PKCS (Public Key Cryptography Standards), 它定义了许多基本 PKI 部件, 包括数字签名和证书请求格式等; 二是由 Internet 工程任务组 IETF (Internet Engineering Task Force) 和 PKI 工作组 PKIX (Public Key Infrastructure Working Group) 所定义的一组具有互操作性的公钥基础设施协议。在今后很长的一段时间内, PKCS 和 PKIX 将会并存, 大部分的 PKI 产品为保持兼容性, 也将会对这两

种标准进行支持。

2. Hash 函数与信息摘要 (Message Digest)

Hash (哈希) 函数提供了这样一种计算过程: 输入一个长度不固定的字符串, 返回一串定长度的字符串, 又称 Hash 值。单向 Hash 函数用于产生信息摘要。Hash 函数主要可以解决以下两个问题: 在某一特定的时间内, 无法查找经 Hash 操作后生成特定 Hash 值的原文; 也无法查找两个经 Hash 操作后生成相同 Hash 值的不同报文。这样在数字签名中就可以解决验证签名和用户身份验证、不可抵赖性的问题。

信息摘要简要地描述了一份较长的信息或文件, 它可以被视为一份长文件的“数字指纹”。信息摘要用于创建数字签名, 对于特定的文件而言, 信息摘要是唯一的。信息摘要可以被公开, 它不会透露相应文件的任何内容。MD2, MD4 和 MD5 (MD 表示信息摘要) 是由 Ron Rivest 设计的专门用于加密处理的, 并被广泛使用的 Hash 函数, 它们产生一种 128 位信息摘要, 除彻底地搜寻外, 没有更快的方法对其加以攻击, 而其搜索时间一般需要 1025 年之久。

3. 数字签名

1) 数字签名的主要过程

- (1) 信息发送者使用一单向散列函数 (Hash 函数) 对信息生成信息摘要。
- (2) 信息发送者使用自己的私钥签名信息摘要。
- (3) 信息发送者把信息本身和已签名的信息摘要一起发送出去。
- (4) 信息接收者通过使用与信息发送者使用的同一个单向散列函数 (Hash 函数) 对接收的信息本身生成新的信息摘要, 再使用信息发送者的公钥对信息摘要进行验证, 以确认信息发送者的身份和信息是否被修改过。

2) 数字加密的主要过程

- (1) 当信息发送者需要发送信息时, 首先生成一个对称密钥, 用该对称密钥加密要发送的报文。
- (2) 信息发送者用信息接收者的公钥加密上述对称密钥。
- (3) 信息发送者将第一步和第二步的结果结合在一起传给信息接收者, 称为数字信封。
- (4) 信息接收者使用自己的私钥解密被加密的对称密钥, 再用此对称密钥解密被发送方加密的密文, 得到真正的原文。

数字签名和数字加密的过程虽然都使用公开密钥体系, 但实现的过程正好相反, 使用的密钥对也不同。数字签名使用的是发送方的密钥对, 发送方用自己的私有密钥进行加密, 接收方用发送方的公开密钥进行解密, 这是一个一对多的关系, 任何拥有发送方公开密钥的人

都可以验证数字签名的正确性。数字加密则使用的是接收方的密钥对,这是多对一的关系,任何知道接收方公开密钥的人都可以向接收方发送加密信息,只有唯一拥有接收方私有密钥的人才能对信息解密。另外,数字签名只采用了非对称密钥加密算法,它能保证发送信息的完整性、身份认证和不可否认性,而数字加密采用了对称密钥加密算法和非对称密钥加密算法相结合的方法,它能保证发送信息保密性。

4. SSL 安全协议

SSL 安全协议最初是由 Netscape Communication 公司设计开发的,又叫“安全套接层(Secure Sockets Layer)协议”,主要用于提高应用程序之间数据的安全系数。SSL 协议的概念可以被总结为:一个保证任何安装了安全套接字的客户和服务端间事务安全的协议,它涉及所有 TC/IP 应用程序。

1) SSL 安全协议主要提供的服务

(1) 用户和服务器的合法性认证。认证用户和服务器的合法性,使得它们能够确信数据将被发送到正确的客户机和服务器上。客户机和服务器都是有各自的识别号,这些识别号由公开密钥进行编号,为了验证用户是否合法,安全套接层协议要求在握手交换数据时进行数字认证,以此来确保用户的合法性。

(2) 加密数据以隐藏被传送的数据。安全套接层协议所采用的加密技术既有对称密钥技术,也有公开密钥技术。在客户机与服务器进行数据交换之前,交换 SSL 初始握手信息,在 SSL 握手信息中采用了各种加密技术对其加密,以保证其机密性和数据的完整性,并且用数字证书进行鉴别,这样就可以防止非法用户进行破译。

(3) 保护数据的完整性。安全套接层协议采用 Hash 函数和机密共享的方法来提供信息的完整性服务,建立客户机与服务器之间的安全通道,使所有经过安全套接层协议处理的业务在传输过程中能全部完整准确无误地到达目的地。

2) SSL 安全协议实现的过程

SSL 安全协议是一个保证计算机通信安全的协议,对通信对话过程进行安全保护,其实现过程主要经过如下几个阶段。

(1) 接通阶段:客户机通过网络向服务器打招呼,服务器回应。

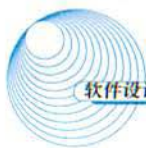
(2) 密码交换阶段:客户机与服务器之间交换双方认可的密码,一般选用 RSA 密码算法,也有的选用 Diffie-Hellman 和 Fortezza-KEA 密码算法。

(3) 会谈密码阶段:客户机与服务器间产生彼此交谈的会谈密码。

(4) 检验阶段:客户机检验服务器取得的密码。

(5) 客户认证阶段:服务器验证客户机的可信度。

(6) 结束阶段:客户机与服务器之间相互交换结束的信息。



当上述动作完成之后,两者间的资料传送就会加密,另外一方收到资料后,再将编码资料还原。即使盗窃者在网络上取得编码后的资料,如果没有原先编制的密码算法,也不能获得可读的有用资料。

发送时信息用对称密钥加密,对称密钥用非对称算法加密,再把两个包绑在一起传过去。接收的过程与发送正好相反,先打开有对称密钥的加密包,再用对称密钥解密。

在电子商务交易过程中,由于有银行参与,按照 SSL 协议,客户的购买信息首先发往商家,商家再将信息转发银行,银行验证客户信息的合法性后,通知商家付款成功,商家再通知客户购买成功,并将商品寄送客户。

5. 数字时间戳技术

数字时间戳技术就是数字签名技术一种变种的应用。在电子商务交易文件中,时间是十分重要的信息。在书面合同中,文件签署的日期和签名一样均是十分重要的,防止文件被伪造和篡改的关键性内容。数字时间戳服务(Digital Time Stamp Service, DTS)是网上电子商务安全服务项目之一,能提供电子文件的日期和时间信息的安全保护。

时间戳(time-stamp)是一个经加密后形成的凭证文档,它包括3个部分:

- (1) 需加时间戳的文件的摘要(digest)。
- (2) DTS 收到文件的日期和时间。
- (3) DTS 的数字签名。

一般来说,时间戳产生的过程为:用户首先将需要加时间戳的文件用 Hash 编码加密形成摘要,然后将该摘要发送到 DTS, DTS 在加入了收到文件摘要的日期和时间信息后再对该文件加密(数字签名),然后送回用户。

书面签署文件的时间是由签署人自己写上的,而数字时间戳则不然,它是由认证单位 DTS 来加的,以 DTS 收到文件的时间为依据。

1.2.4 计算机可靠性

1. 计算机可靠性概述

计算机系统的硬件故障通常是由元器件的失效引起的。对元器件进行寿命试验并根据实际资料统计得知,元器件的可靠性可分成3个阶段:开始阶段器件工作处于不稳定期,失效率较高;第二阶段器件进入正常工作期,失效率最低,基本保持常数;第三阶段元器件开始老化,失效率又重新提高;这就是所谓的“浴盆曲线”。因此应保证在计算机中使用的元器件处于第二阶段。在第一阶段对元器件应进行老化筛选,而到了第三个阶段,则淘汰该计算机。

计算机系统的可靠性是指从它开始运行($t=0$)到某时刻 t 这段时间内能正常运行的概率,用 $R(t)$ 表示。所谓失效率是指单位时间内失效的元件数与元件总数的比例,以 λ 表示,当 λ 为常数时,可靠性与失效率的关系为:

$$R(t) = e^{-\lambda t}$$

典型的失效率与时间的关系曲线如图 1-26 所示。

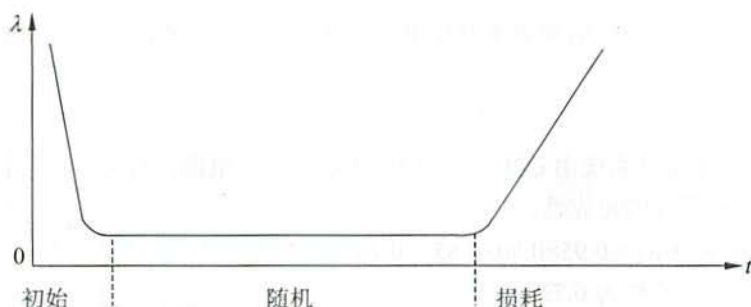


图 1-26 失效率特性

两次故障之间系统能正常工作的时间的平均值称为平均无故障时间 (MTBF):

$$MTBF = 1/\lambda$$

通常用平均修复时间 (MTRF) 来表示计算机的可维修性,即计算机的维修效率,指从故障发生到机器修复平均所需要的时间。计算机的可用性是指计算机的使用效率,它以系统在执行任务的任意时刻能正常工作的概率 A 来表示:

$$A = \frac{MTBF}{MTBF + MTRF}$$

计算机的 RAS 技术,就是指用可靠性 R 、可用性 A 和可维修性 S 这 3 个指标衡量一个计算机系统。但在实际应用中,引起计算机故障的原因除了元器件以外还与组装工艺、逻辑设计等因素有关。因此不同厂家生产的兼容机,即使采用相同的元器件,其可取性及 MTBF 也可能会相差很大。

2. 计算机可靠性模型

计算机系统是一个复杂的系统,而且影响其可靠性的因素也非常繁复,很难直接对其进行可靠性分析;但通过建立适当的数学模型,把大系统分割成若干子系统,可以简化其分析过程。常见的系统可靠性数学模型有以下 3 种:

(1) 串联系统:假设一个系统由 N 个子系统组成,当且仅当所有的子系统都能正常工作时,系统才能正常工作,这种系统称为串联系统,如图 1-27 所示。设系统各个子系统的可靠

性分别用 R_1, R_2, \dots, R_N 来表示, 则系统的可靠性 R 可由下式求得:

$$R = R_1 R_2 \cdots R_N$$

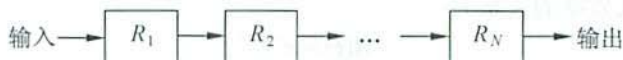


图 1-27 串联系统的可靠性模型

如果系统的各个子系统的失效率分别用 $\lambda_1, \lambda_2, \dots, \lambda_N$ 来表示, 则系统的失效率 λ 可由下式求得:

$$\lambda = \lambda_1 + \lambda_2 + \cdots + \lambda_N$$

【例 1.1】 设计计算机系统由 CPU、存储器、I/O 三部分组成, 其可靠性分别为 0.95, 0.90 和 0.85。求计算机系统的可靠性。

解: $R = R_1 \cdot R_2 \cdot R_3 = 0.95 \times 0.90 \times 0.85 = 0.73$

计算机系统的可靠性为 0.73。

(2) 并联系统: 假如一个系统由 N 个子系统组成, 只要有一个子系统正常工作, 系统就能正常工作, 这样的系统称为并联系统, 如图 1-28 所示。设每个子系统的可靠性分别以 R_1, R_2, \dots, R_N 表示, 整个系统的可靠性只可由下式来求得:

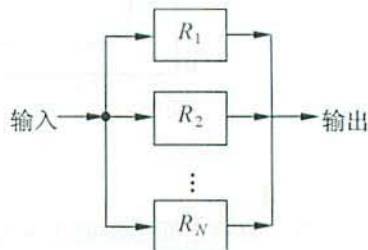


图 1-28 并联系统的可靠性模型

$$R = 1 - (1 - R_1)(1 - R_2) \cdots (1 - R_N)$$

假如所有的子系统的失效率均为 λ , 则系统的失效率 μ 为:

$$\mu = \frac{1}{\frac{1}{\lambda} \sum_{j=1}^N \frac{1}{j}}$$

在并联系统中只有一个子系统是真正需要的, 其余 $N-1$ 个子系统称为冗余子系统, 随着冗余子系统数量的增加, 系统的平均无故障时间也增加了。

【例 1.2】 设一个系统由 3 个相同子系统构成, 其可靠性为 0.9, 平均无故障时间为 10000 小时, 求系统的可靠性和平均无故障时间。

解: $R_1 = R_2 = R_3 = 0.9$ $\lambda_1 = \lambda_2 = \lambda_3 = 1/10000 = 1 \times 10^{-4}$ (小时)

系统可靠性 $R = 1 - (1 - R_1)^3 = 0.999$

系统平均无故障时间 MTBF 为:

$$MTBF = \frac{1}{\mu} = \frac{1}{\lambda} \sum_{j=1}^3 \frac{1}{j} = \frac{1}{\lambda} \times \left(1 + \frac{1}{2} + \frac{1}{3} \right) = 18333 \text{ (小时)}$$

(3) N 模冗余系统: N 模冗余系统由 N 个 ($N=2n+1$) 相同的子系统和一个表决器组成, 表决器把 N 个子系统中占多数相同结果的输出作为系统的输出, 如图 1-29 所示。

在 N 个子系统中, 只要有 $n+1$ 个或 $n+1$ 个以上子系统能正常工作, 系统就能正常工作, 输出正确的结果。假设表决器是完全可靠的, 每个子系统的可靠性为 R_0 , 则 N 模冗余系统的可靠性为:

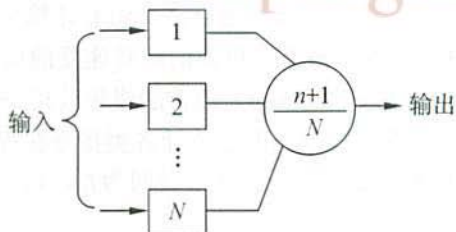


图 1-29 N 模冗余系统

$$R = \sum_{i=n+1}^N \binom{N}{i} \times R_0^i (1-R_0)^{N-i}$$

其中, $\binom{N}{i}$ 表示从 N 个元素中取 i 个元素的组合数。

提高计算机的可靠性一般采取两项措施:

- 提高元器件质量, 改进加工工艺与工艺结构完善电路设计。
- 发展容错技术, 使得在计算机硬件有故障的情况下, 计算机仍能继续运行, 得出正确的结果。

1.2.5 计算机系统的性能评价

无论是生产计算机的厂商还是使用计算机的用户都需要有某种方法来衡量计算机的性能, 作为设计、生产、购买和使用的依据。但是由于计算机系统是一个极复杂的系统, 其体系结构、组成和实现都有若干种策略; 而且其应用领域也千差万别, 所以很难找到统一的规则或标准去评测所有的计算机。

1. 性能评测的常用方法

(1) 时钟频率: 计算机的时钟频率在一定程度上反映了机器速度, 一般来讲, 主频越高, 速度越快。但是相同频率、不同体系结构的机器, 其速度可能会相差很多倍, 因此还需要用其他方法来测定机器性能。

(2) 指令执行速度: 在计算机发展的初期, 曾用加法指令的运算速度来衡量计算机的速度, 速度是计算机的主要性能指标之一。因为加法指令的运算速度大体上可反映出乘法、除法等其他算术运算的速度, 而且逻辑运算、转移指令等简单指令的执行时间往往设计成与加法指令相同, 因此加法指令的运算速度有一定代表性。当时表征机器运算速度的单位是 KIPS (千条指令/秒), 后来随着机器运算速度的提高, 计量单位由 KIPS 发展到 MIPS (百万条指令/秒)。

(3) 等效指令速度法: 随着计算机指令系统的发展, 指令的种类大大增加, 用单种指令的 MIPS 值来表征机器的运算速度的局限性日益暴露, 因此很快就出现了改进的办法, 称之为吉普森 (Gibson) 混合法或等效指令速度法。

等效指令速度法统计各类指令在程序中所占比例, 并进行折算。设某类指令 i 在程序中所占比例为 w_i , 执行时间为 t_i , 则等效指令的执行时间为:

$$T = \sum_{i=1}^n (w_i \times t_i)$$

其中 n 为指令的种类数。

(4) 数据处理速率 (Processing Data Rate, PDR) 法: 因为在不同程序中, 各类指令的使用频率是不同的, 所以固定比例方法存在着很大的局限性; 而且数据长度与指令功能的强弱对解题的速度影响极大。同时这种方法也不能反映现代计算机中高速缓冲存储器 (Cache)、流水线、交叉存储等结构的影响。具有这种结构的计算机的性能不仅与指令的执行频率有关, 而且也与指令的执行顺序和地址分布有关。

PDR 法采用计算 PDR 值的方法来衡量机器性能, PDR 值越大, 机器性能越好。PDR 与每条指令和每个操作数的平均位数以及每条指令的平均运算速度有关, 其计算方法如下:

$$\text{PDR} = L / R$$

其中: $L = 0.85G + 0.15H + 0.4J + 0.15K$

$$R = 0.85M + 0.09N + 0.06P$$

式中: G 是每条定点指令的位数; M 是平均定点加法时间; H 是每条浮点指令的位数; N 是平均浮点加法时间; J 是定点操作数的位数; P 是平均浮点乘法时间; K 是浮点操作数的位数。

此外, 还作了如下规定: $G > 20$ 位, $H > 30$ 位; 从主存取一条指令的时间等于取一个字的时间; 指令与操作数存放在主存, 无变址或间址操作; 允许有并行或先行取指令功能, 此时选择平均取指令时间。PDR 值主要对 CPU 和主存储器的速度进行度量, 但不适合衡量机器的整体速度, 因为它没有涉及 Cache、多功能部件等技术对性能的影响。

(5) 核心程序法: 上述性能评价方法主要是针对 CPU (有时包括主存), 它没有考虑诸如 I/O 结构、操作系统、编译程序的效率等系统性能的影响, 因此难以准确评价计算机的实际工作能力。

核心程序法是研究较多的一种方法, 它把应用程序中用得最频繁的那部分核心程序作为评价计算机性能的标准程序, 在不同的机器上运行, 测得其执行时间, 作为各类机器性能评价的依据。机器软硬件结构的特点能在核心程序中得到反映, 但是核心程序各部分之间的联系较小。由于程序短, 所以访问存储器的局部性特征很明显, 以致 Cache 的命中率比一般程

序高。

2. 基准测试程序

基准程序法(Benchmark)是目前一致承认的测试性能的较好方法,有多种多样的基准程序,如主要测试整数性能的基准程序、测试浮点性能的基准程序等。

(1) 整数测试程序: Dhrystone 是一个综合性的基准测试程序。它是为了测试编译器和 CPU 处理整数指令和控制功能的有效性,人为地选择一些“典型指令”综合起来形成的测试程序。

用 C 语言编写的 Dhrystone 基准程序用了 100 条语句,由下列操作组成:各种赋值语句;各种数据类型和数据区;各种控制语句;过程调用和参数传递;整数运算和逻辑操作。

Dhrystone 程序测试的结果由每秒多少个 Dhrystones 来表示机器的性能,这个数值越大性能越好。VAX11/780 的测试结果为每秒 1757Dhrystones,为便于比较,人们假设 1VAX MIPS=每秒 1757Dhrystones,将被测机器的结果除以 1757,就得到被测机器相对 VAX11/780 的 MIPS 值。有些厂家在宣布机器性能时就用 Dhrystone MIPS 值作为各自机器的 MIPS 值。

不过不同的厂家在测试 MIPS 值时,使用的基准程序一般是不一样的,因此不同厂家机器的 MIPS 值有时虽然是相同的,但其性能却可能差别很大,那是因为各厂家在设计计算机时针对不同的应用领域:如科学和工程应用、商业管理应用、图形处理应用等,而采用了不同的体系结构和实现方法。同一厂家的机器,采用相同的体系结构,用相同的基准程序测试,得到的 MIPS 值越大,一般说明机器速度越快。

(2) 浮点测试程序:在计算机科学和工程应用领域内,浮点计算工作量占很大比例,因此机器的浮点性能对系统的应用有很大的影响。有些机器只标出单个浮点操作性能,如浮点加法、浮点乘法时间。而大部分工作站则标出用 Linpack 和 Whetstone 基准程序测得的浮点性能。Linpack 主要测试向量性能和高速缓存性能。Whetstone 是一个综合性测试程序,除测试浮点操作外,还测试整数计算和功能调用等性能。

① 理论峰值浮点速度:巨型机和小巨型机在说明书中经常给出“理论峰值速度”的 MFLOPS 值。它不是机器实际执行程序时的速度,而是机器在理论上最大能完成的浮点处理速度。它不仅与处理机时钟周期有关,而且还与一个处理机里能并行执行操作的流水线功能部件数目和处理机的数目有关。多个 CPU 机器的峰值速度是单个 CPU 的峰值速度与 CPU 个数的乘积。

② Linpack 基准测试程序:Linpack 基准程序是一个用 FORTRAN 语言写成的子程序软件包,称为基本线性代数子程序包,此程序完成的主要操作是浮点加法和浮点乘法操作。测量计算机系统的 Linpack 性能时,让机器运行 Linpack 程序,测量运行时间,将结果用 MFLOPS 表示。

当解 n 阶线性代数方程组时, n 越大, 向量化程度越高。其关系如表 1-5 所示。

表 1-5 矩阵的向量化程度

矩阵规模	100×100	300×300	1000×1000
向量化百分比	80%	95%	98%

向量化百分比指的是含向量成分的计算量占整个程序计算量的百分比。在同一台机器中, 向量化程度越高。机器的运算速度越快, 因为不管 n 的大小, 求解方程时花在非向量操作的时间差不多是相等的。

③ Whetstone 基准测试程序: Whetstone 是用 FORTRAN 语言编写的综合性测试程序, 主要由执行浮点运算、整数算术运算、功能调用、数组变址、条件转移和超越函数的程序组成。Whetstone 的测试结果用 Kwips 表示, 1Kwips 表示机器每秒钟能执行 1000 条 Whetstone 指令。

(3) SPEC 基准程序 (SPEC Benchmark): SPEC 是 System Performance Evaluation Cooperation 的缩写, 是由几十家世界知名的计算机大厂商所支持的非盈利的合作组织组成, 旨在开发共同认可的标准基准程序。

SPEC 基准程序是由 SPEC 开发的一组用于计算机性能综合评价的程序。以对 VAX11/780 机的测试结果作为基数, 其他计算机的测试结果以相对于这个基数的比率来表示。SPEC 基准程序能较全面地反映机器性能, 有很高的参考价值。

SPEC 1.0 是 1989 年 10 月宣布的, 是一套复杂的基准程序集, 主要用于测量与工程和科学应用有关的数字密集型的整数和浮点数方面的计算。源程序超过 15 万行, 包含 10 个测试程序, 使用的数据量比较大, 分别测试应用的各个方面。

SPEC 基准程序测试结果一般以 SPECmark (SPEC 分数)、SPECint (SPEC 整数) 和 SPECfp (SPEC 浮点数) 来表示。其中 SPEC 分数是 10 个程序的几何平均值, SPEC 整数是 4 个整数程序的几何平均值, SPEC 浮点数是 6 个浮点程序的几何平均值。

1992 年在原来 SPECint89 和 SPECfp89 的基础上又增加了 2 个整数测试程序和 8 个浮点数测试程序, 因此 SPECint92 由 6 个程序组成, SPECfp92 由 14 个程序组成。这 20 个基准程序是基于不同的应用写成的, 主要测 32 位 CPU、主存储器、编译器和操作系统的性能。

已有大约 30 个计算机软件和硬件公司参加了这个组织, 它们都承认这种测试, 并同意作为它们生产的机器或系统的测试标准。SPEC 基准程序的测试结果获得了普遍的认可。

参加这个组织的主要成员有: IBM, AT&T, BULL, CDC, DG, DEC, Fujitsu, HP, Intel, MIPS, Motorola, SGI, SUN 和 Unisys 等。1995 年这些厂商又共同推出了 SPECint95 和 SPECfp95 作为最新的测试标准程序。

(4) TPC 基准程序: TPC 是 Transaction Processing Council (事务处理委员会) 的缩写,

TPC 基准程序是由 TPC 开发的评价计算机事务处理性能的测试程序,用以评测计算机在事务处理、数据库处理、企业管理与决策支持系统等方面的性能。TPC 已经推出了 4 套基准程序: TPC-A、TPC-B、TPC-C 和 TPC-D。其中 A 和 B 已经过时,不再使用了。TPC-C 是在线事务处理(OLTP)的基准程序,TPC-D 是决策支持的基准程序。TPC 即将推出 TPC-E,作为大型企业信息服务的基准程序。

1.2.6 计算机故障诊断与容错

1. 计算机故障诊断技术

1) 计算机的故障

根据计算机故障表现出的特点,可以分为永久性、间歇性及瞬时性 3 类。

(1) 永久性故障:永久性故障表现出稳定性及持续性的特征,如元器件的损坏、电路的断线或短路、程序编写的错误等,它的特点是故障可以重复出现。

(2) 间歇性故障:间歇性故障表现出不稳定性及对系统状态具有依赖性的特征,此时可能表现出机器时好时坏的现象。

(3) 瞬时性故障:瞬时性故障是由偶然原因引起的短暂故障,一般无须修复就能恢复正常。但若频繁出现,也会影响工作;所以需要查出故障原因,以消除影响。

2) 故障诊断方法

故障诊断包括故障检测和故障定位两个方面:故障检测是测试并确定计算机系统有无故障的过程;故障定位是判定故障发生在某个子系统、功能块或器件的过程。

通常,故障诊断的主要方法有下述 3 种。

(1) 对电路直接进行测试的故障定位测试法:将被测试的系统划分成若干个测试域,并向这些域发送一系列调试码,然后收集并分析被调试区域的返回码,以确定故障位置或找出产生故障的元器件。

这种按线路内部的电路结构逐个进行逻辑关系测试的方法,将随着电路规模的增大而急剧地增加复杂性。因此对计算机系统往往是先采用下面提到的按功能测试的方法诊断出某个出故障的功能部件,然后再对这一部件的电路进行测试,或者干脆把这一产生故障的功能部件整个替换掉。

(2) “检查诊断程序”法:用机器语言写的“检查诊断程序”来进行诊断的方法是一种功能测试法。它利用机器指令的功能来对系统的某些部件进行测试。但由于一条指令的正确执行,往往涉及许多部件,因此故障定位所需的诊断时间较长,而且要求系统必须有能力保证诊断程序的正确执行,否则计算机连程序都不能运行,更谈不上诊断了。

(3) 微诊断法:在微程序控制的计算机中用微指令来对系统进行诊断叫做微诊断法。由

微指令组成的微诊断程序存放在控制存储器中或者先存放在外存储器中,诊断时再调入控制存储器。设计这种可写入的控制存储器称为动态微程序设计。微诊断法也是一种功能测试法。为了进行测试而必须保证工作只涉及较少部件,因此故障分辨得很细,诊断程序运行的时间也较短。

不论采用指令或微指令进行诊断,都是采用滚雪球的方法。先对系统的某一部分进行测试,如果没有发现错误,就在这部分的基础上对其他未测试部分进行测试。如同滚雪球般,越滚越大,直到全部测试完成。

2. 计算机容错技术

容错是采用冗余方法来消除故障影响。针对硬件,有时间冗余和元器件冗余两种方法。时间冗余是对同一计算进行重复运算,并对结果进行比较,或进行验算等,这种方法对解决偶然性故障比较有效。元器件冗余即是利用附加的硬件来保证在局部有故障的情况下系统能正常工作。

容错的技术在不断进步,从最初的简单双机冗余到越来越高级的容错系统设计,保障了系统的可取性和稳定性。

(1) 简单的双机备份:在20世纪60年代主要采用双处理机或双机的方法来达到容错的目的。例如把关键的元件(处理机、存储器等)或整个计算机设置两套:一套是系统运行时使用,另一套用作备份。根据系统的工作情况又可分为:热备份和冷备份两种。

① 热备份(双重系统):两套系统同时同步运行,当联机子系统检测到错误时,退出服务进行检修,而由“热备份”子系统接替工作。

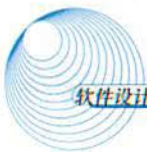
② 冷备份(双工系统):处于冷备份的子系统平时停机或者运行与联机系统无关的运算,当联机子系统产生故障时,人工或自动进行切换,使冷备份系统成为联机系统。在冷备份时,不能保证从程序端点处精确地连续工作,因为备份机不能取得原来机器上当前运行的全部数据。

(2) 操作系统支持的双机容错:在20世纪70年代中期出现了软件和硬件结合的容错方法。该方法在操作系统的层次上,支持联机维修,即故障部分退出运行、进行维修并重新投入运行都不影响正在运行的应用程序。该结构的特点是系统内包括双处理器、双存储器、双输入/输出控制器、不间断工作的电源以及与之适应的操作系统等。因此上述硬件的任一部分发生故障都不会影响系统的继续工作。系统容错是在操作系统控制下进行的,在每个处理机上都保持了反映所有系统资源状态的表格以及本机和它机的工作进程。

为了正常工作,必须保持双处理机工作同步,及时比较操作结果;并且提供检测机制以定位出故障的处理器(因为双机表决结果为1:1,不能判定哪台机器出故障)。

3. 故障处理步骤及方法

- (1) 故障封闭：防止故障扩散。
- (2) 检错：利用冗余代码，提供出错信息，例如奇偶检验码、海明检验码、CRC 校验码等。
- (3) 重复执行：利用软件重新执行运算，以清除瞬时异常的影响。
- (4) 诊断：对于非瞬时异常，确定故障性质及地点。
- (5) 系统重构及恢复：对故障部件从系统中退出，重建系统结构，然后恢复运行。
- (6) 修复：对故障部件进行在线修复或脱机修复，一般为后者。
- (7) 重入：故障部件修复后，重新投入系统运行。



第2章 程序语言基础知识

程序语言是为了书写计算机程序而人为设计的符号语言,用于对计算过程进行描述、组织和推导。程序设计语言的广泛使用始于1957年出现的FORTRAN,经过四十多年的发展,目前世界上流行的程序语言有上百种之多,程序语言的演化已经超越了运行它们的机器。

2.1 程序语言概述

本节主要介绍了程序语言的基本概念、基本成分和一些有代表性的程序语言的特点及其适用范围。

2.1.1 程序语言的基本概念

1. 低级语言和高级语言

计算机的硬件只能识别由0、1字符串组成的机器指令序列,即机器指令程序,因此机器指令程序是最基本的计算机语言。由于机器指令是特定的计算机系统所固有的、面向机器的语言,所以用机器语言进行程序设计时,需要对机器结构有较多的了解。另外,用机器语言编制出来的程序可读性很差,程序也难以理解、修改和维护。为了提高程序设计的效率,人们就用容易记忆的符号代替0、1序列,来表示机器指令,例如,用ADD表示加法、SUB表示减法等。用符号表示的指令称为汇编指令,汇编指令的集合被称为汇编语言。虽然使用汇编语言编写程序的效率和程序的可读性有所提高,但汇编语言和机器语言十分接近,它的书写格式在很大程度上取决于特定计算机的机器指令,因此它仍然是一种面向机器的语言,人们称机器语言和汇编语言为低级语言。在这个基础上,人们开发了功能更强、抽象级别更高的语言以支持程序设计,于是就产生了面向各类应用的程序语言,称为高级语言。目前已开发出许多高级语言,常见的有Fortran、Cobol、Pascal、C、Ada、C++、Java等。这类语言与人们使用的自然语言比较接近,大大提高了程序设计的效率。

2. 编译程序和解释程序

目前,尽管人们可以借助高级语言与计算机进行交互,但是计算机仍然只能理解和执行由0、1序列构成的机器语言,因此高级程序语言需要翻译,担负这一任务的程序称为“语言处理程序”。由于应用的不同,语言之间的翻译也是多种多样的。它们大致可分为:汇编程序、解

释程序和编译程序。

用某种高级语言或汇编语言编写的程序称为源程序,源程序不能直接在计算机上执行。如果源程序是用汇编语言编写的,则需要一个称为汇编程序的翻译程序将其翻译成目标程序后才能执行。如果源程序是用某种高级语言编写的,则需要对应的解释程序或编译程序对其进行翻译,然后在机器上运行。

解释程序也称为解释器,它或者直接解释执行源程序,或者将源程序翻译成某种中间表示形式后再加以执行;而编译程序(编译器)则是将源程序翻译成目标语言程序,然后在计算机上运行目标程序。两种语言处理程序的根本区别是:在编译方式下,机器上运行的是与源程序等价的目标程序,源程序和编译程序都不再参与目标程序的执行过程;而在解释方式下,解释程序和源程序(或某种等价表示)要参与到程序的运行过程中,运行程序的控制权在解释程序。解释器翻译源程序时不生成独立的目标程序,而编译器则将源程序翻译成独立的目标程序。

3. 程序设计语言的定义

一般地,程序设计语言的定义都涉及语法、语义、语用等3个方面。

(1) 语法是指由程序语言的基本符号组成程序中的各个语法成分(包括程序)的一组规则,其中由基本符号构成的符号(单词)书写规则称为词法规则,由符号(单词)构成语法成分的规则称为语法规则。程序语言的语法可通过形式语言进行描述。

(2) 语义是程序语言中按语法规则构成的各个语法成分的含义,可分为静态语义和动态语义。静态语义指编译时可以确定的语法成分的含义;而运行时刻才能确定的含义是动态语义。一个程序的执行效果说明了该程序的语义,它取决于构成程序的各个组成部分的语义。

(3) 语用表示了构成语言的各个记号和使用者的关系,涉及符号的来源、使用和影响。

语言的实现则有个语境问题。语境是指理解和实现程序设计语言的环境,这种环境包括编译环境和运行环境。

2.1.2 程序设计语言的种类和特点

1. 程序语言的发展概述

程序语言有交流算法和计算机实现的双重目的,现在的程序语言种类繁多,它们在应用上各有不同的侧重面。

若一种程序语言不依赖于机器硬件,则称为高级语言;若程序语言能够应用于范围广泛的问题求解过程中,则称之为通用的程序设计语言。

Fortran 是第一个被广泛用来进行科学计算的高级语言。一个 Fortran 程序由一个主程序或一个主程序与若干个子程序组成。主程序及每一个子程序都是独立的程序单位,称为一个程序

模块。在 Fortran 中,子程序是实现模块化的有效途径。

ALGOL 60 主导了 20 世纪 60 年代程序语言的发展。它有严格的文法规则,采用巴科斯范式 BNF 来描述语言的语法。ALGOL 是一个分程序结构的语言。一般来说,一个 ALGOL 程序本身就是一个分程序。每个分程序由 begin 和 end 括起来,以说明分程序的范围和它所管辖的名字的作用域。分程序的结构可以是嵌套的,也就是说,分程序内可以包含别的分程序。过程也可以称为一个分程序。同一个名字在不同的分程序中可以代表完全不同的实体。如果一个名字在若干层嵌套分程序中多次被说明,则程序中该名字的使用由离使用点最近的内层说明决定,即“最近嵌套原则”。此外,ALGOL 还提供了数组的动态说明和过程的递归调用。

因为一个分程序只有在执行时才需要数据空间,执行完成后就释放所占用的空间。因此,分程序结构的主要优点是可以非常有效地使用存储器。由于分程序结构的嵌套性,用一个栈来组织和管理整个程序运行时的数据空间是非常方便的。

COBOL (COmmon Business-Oriented Language) 是一种面向事务处理的高级语言。在企业管理中,数值计算并不复杂,但数据处理的信息量却很大。1959 年,由美国的一些计算机用户组织设计了专用于商务处理的计算机语言 COBOL,并于 1961 年由美国数据系统语言协会公布,经不断修改、丰富、完善和标准化,已发展了多种版本。

COBOL 语言使用了 300 多个英语保留字,大量采用普通英语词汇和句型。COBOL 语言的语法规则很严格。目前 COBOL 语言主要应用于情报检索、商业数据处理等管理领域。

Pascal 语言是一种结构化程序设计语言,由瑞士苏黎世联邦工业大学的沃斯(N.Wirth)教授研制,于 1971 年正式发表。Pascal 是从 ALGOL60 衍生的,但功能更强且容易使用。Pascal 语言在高校计算机软件教学中一直处于主导地位。后来的 Pascal 语言中添加了并发控制结构,产生了并发 Pascal。在 Pascal 语言中分程序和过程这两个概念合二为一,统一为过程。而一个 Pascal 程序本身可看成是一个操作系统所调用的过程。Pascal 过程可以是嵌套和递归的。

C 语言是 20 世纪 70 年代发展起来的一种通用程序设计语言,它提供了一个丰富的运算符集合以及比较紧凑的语句格式。C 语言的主要特色是兼顾了高级语言和汇编语言的特点,简洁、丰富、可移植。C 与 UNIX 操作系统紧密相关,UNIX 操作系统及其上的许多软件都是用 C 编写的。C 提供了高效的执行语句并且允许程序员直接访问操作系统和底层硬件,这使得 C 在系统应用和实时处理应用的开发中成为主要语言。

C++是在 C 语言的基础上于 20 世纪 80 年代发展起来的,与 C 兼容。在 C++中,最主要的是增加了类机制,使其成为一种面向对象的程序设计语言。

Java 产生于 20 世纪 90 年代,其目的是用于开发网络浏览器的小应用程序,但是作为一种通用的程序设计语言,Java 也得到了广泛的应用。Java 保留了 C++的基本语法、类和继承等概念,删掉了 C++中一些不好的特征,因此与 C++相比,Java 更简单,其语法和语义更合理。

各种程序语言都在不断地发展之中。目前,程序设计语言及编程环境正向面向对象及可视

化编程环境方向发展,出现了许多新的语言及其开发工具。例如,微软公司的 Visual 系列编程工具及 PowerBuilder 等,已经得到了广泛的应用。

2. 程序语言的种类和特点

程序语言的分类没有统一的标准,这里根据设计程序的方法将程序语言大致分为命令式程序设计语言、面向对象的程序设计语言、函数式程序设计语言和逻辑型程序设计语言等类型。

1) 命令式程序设计语言

命令式语言是基于动作的语言,在这种语言中,计算被看成是动作的序列。命令式语言族开始于 Fortran, Pascal 和 C 语言体现了命令式程序设计的关键思想。

2) 面向对象的程序设计语言

面向对象的程序设计在很大程度上应归功于从模拟领域发展起来的 Simula, Simula 提出了对象和类的概念。C++、Java 和 smaltalk 是面向对象程序设计语言的代表。

一般认为,面向对象程序语言主要包含下面几个概念。

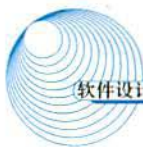
- 对象:对象是人们要进行研究的任何事物,它具有状态和操作。面向对象语言把状态和操作封装于对象实体之中,并提供一种访问机制,使对象的“私有数据”仅能由这个对象的操作来执行。用户只能通过向允许公开的操作提出要求(或发送消息),才能查询和修改对象的状态。这样,对象状态的具体表示和操作的具体实现都被隐藏起来了。
- 类:类是面向对象语言必须提供的由用户定义的数据类型,它将具有相同状态、操作和访问机制的多个对象抽象成一个对象类。在定义了类以后,属于这种类的一个对象叫作类实例或类对象。类代表一般,而该类的一个对象代表具体。
- 继承:继承是面向对象语言的另一个基本要素。在客观世界中,存在着整体和部分、一般和特殊的关系。继承实现了一般与特殊的关系,解决了软件的重用性和扩充性的问题。类与类之间可以组成继承层次,一个类的定义可以定义在另一个已定义类的基础上,前者称为子类,后者称为父类。子类可以继承父类中的属性和操作,也可以定义自己的属性和操作;从而使内部表示上有差异的对象可以共享与它们结构中的共同部分有关的操作,达到概念复用和代码重用的目的。

3) 函数式程序设计语言

函数式语言是一类以 λ -演算为基础的语言,其基本概念来自于 LISP,这是一个在 1958 年为了人工智能应用而设计的语言。函数是一种对应规则(映射),它使定义域中每个元素和值域中唯一的元素相对应。例如:

函数定义 1: $\text{Square}[x] := x * x$

函数定义 2: $\text{Plustwo}[x] := \text{Plusone}[\text{Plusone}[x]]$



函数定义 3: $\text{fact}[n] := \text{if } n=0 \text{ then } 1 \text{ else } n * \text{fact}[n-1]$

在函数定义 2 中使用了函数复合,即将一个函数调用嵌套在另一个函数定义中。在函数定义 3 中,函数被递归定义。由此可见,函数可以看成是一种程序,其输入就是定义中左边括号中的量,它也可将输入组合起来产生一个规则,组合过程中可以使用其他函数或该函数本身。这种用函数和表达式建立程序的方法就是函数式程序设计。函数式程序设计语言的优点之一就是对表达式中出现的任何函数都可以用其他函数来代替,只要这些函数调用产生相同的值。

函数式语言的代表 LISP 在许多方面与其他语言不同,其中最为显著的是,该语言中的程序和数据的形式是等价的,这样数据结构就可以作为程序执行,同样程序也可以作为数据修改。在 LISP 中,大量地使用递归。

4) 逻辑型程序设计语言

逻辑型语言是一类以形式逻辑为基础的语言,其代表是建立在关系理论和一阶谓词理论基础上的 PROLOG。PROLOG 代表“Programming in Logic”。PROLOG 程序是一系列事实、数据对象或事实间的具体关系和规则的集合。通过查询操作把事实和规则输入数据库。用户通过输入查询来执行程序。在 PROLOG 中,关键操作是模式匹配,通过匹配一组变量与一个预先定义的模式并将该组变量赋给该模式来完成操作。以值集合 S 和 T 上的二元关系 R 为例, R 实现后,可以询问:

- 已知 a 和 b , 确定 $R(a, b)$ 是否成立;
- 已知 a , 求所有使 $R(a, y)$ 成立的 y ;
- 已知 b , 求所有使 $R(x, b)$ 成立的 x ;
- 求所有使 $R(x, y)$ 成立的 x 和 y 。

逻辑型程序设计具有与传统的命令型程序设计完全不同的风格。PROLOG 数据库中的事实和规则是一些 Hore 子句。Hore 子句的形式为“ $P:-P_1, P_2, \dots, P_n$ ”,其中 $n \geq 0$, $P_i (1 \leq i \leq n)$ 为形如 $R_i(\dots)$ 的断言, R_i 是关系名。该子句表示规则:若 P_1, P_2, \dots, P_n 均为真(成立),则 P 为真。当 $n=0$ 时, Hore 子句变成“ P ”,这样的子句称为事实。

一旦有了事实与规则后,我们就可以提出询问。测试用户询问 A 是否成立时,采用归结方法:

- 如果程序中包含事实“ P ”,且 P 和 A 匹配,则 A 成立。
- 如果程序中包含 Hore 子句“ $P:-P_1, P_2, \dots, P_n$ ”,且 P 和 A 匹配,则 PROLOG 转而测试“ P_1, P_2, \dots, P_n ”;只有当 P_1, P_2, \dots, P_n 都成立时才能断言 P 成立。当求解某个 P_i 失败时,则返回到前面的某个成功点并尝试另一种选择,也就是进行回溯。
- 只有当所有可能情况都已穷尽时,才能推导出 P 失败。

PROLOG 有很强的推理功能,适用于书写自动定理证明、专家系统、自然语言理解等问题的程序。

2.1.3 程序语言的基本成分

程序语言的基本成分包括数据、运算、控制和传输等。

1. 程序语言的数据成分

程序语言的数据成分指的是一种程序语言的数据类型。数据对象总是对应着应用系统中某些有意义的东西,数据表示则指示了程序中值的组织形式。数据类型用于代表数据对象,还用于在基础机器中完成对值的布局,同时还可用于检查表达式中对运算的应用是否正确。

数据是程序操作的对象,具有存储类、类型、名称、作用域和生存期等属性,使用时要为其分配内存空间。数据名称由用户通过标识符命名,标识符是由字母、数字和下划线“_”组成的标记;类型说明数据占用内存的大小和存放形式;存储类说明数据在内存中的位置和生存期;作用域则说明可以使用数据的代码范围;生存期说明数据占用内存的时间范围。从不同角度可将数据进行不同的划分。

1) 常量和变量

按照程序运行时数据的值能否改变,将数据分为常量和变量。程序中的数据对象可以具有左值和(或)右值。左值指存储单元(或地址、容器),右值是值(或内容)。变量具有左值和右值,在程序运行过程中其右值可以改变;常量只有右值,在程序运行过程中其右值不能改变。

2) 全局量和局部量

按数据的作用域范围,可分为全局量和局部量。系统为全局变量分配的存储空间在程序运行的过程中一般是不改变的,而为局部变量分配的存储单元是动态改变的。

3) 数据类型

按照数据组织形式的不同可将数据分为基本类型、用户定义类型、构造类型及其他类型。

C(C++)的数据类型如下:

- 基本类型:整型(int)、字符型(char)、实型(float、double)和布尔类型(bool)。
- 特殊类型:空类型(void)。
- 用户定义类型:枚举类型(enum)。
- 构造类型:数组、结构、联合。
- 指针类型: `type *`。
- 抽象数据类型:类类型。

其中,布尔类型和类类型是C++语言在C语言的基础上扩充的。

2. 程序语言的运算成分

程序语言的运算成分指明允许使用的运算符号及运算规则。大多数高级程序语言的基本运

算可以分成算术运算、关系运算和逻辑运算,有些语言如 C (C++) 还提供位运算。运算符的使用与数据类型密切相关。为了明确运算结果,运算符要规定优先级和结合性,必要时还要使用圆括号。

3. 程序语言的控制成分

控制成分指明语言允许表述的控制结构,程序员使用控制成分来构造程序中的控制逻辑。理论上已经证明可计算问题的程序都可以用顺序、选择和重复这 3 种控制结构来描述。

1) 顺序结构

顺序结构用来表示一个计算操作序列。计算过程从所描述的第一个操作开始,按顺序依次执行后续的操作,直到序列的最后一个操作,如图 2-1 所示。顺序结构内也可以包含其他控制结构。

2) 选择结构

选择结构提供了在两种或多种分支中选择其中一个的逻辑。基本的选择结构是指定一个条件 P,然后根据条件的成立与否决定控制流走程序块 A 还是走程序块 B,从两个分支中选择一个执行。如图 2-2 (a) 所示。选择结构中的程序块 A 或程序块 B 还可以包含顺序、选择和重复结构。程序语言中通常还提供简化了的选择结构,也就是没有程序块 B 的分支结构,如图 2-2 (b) 所示。

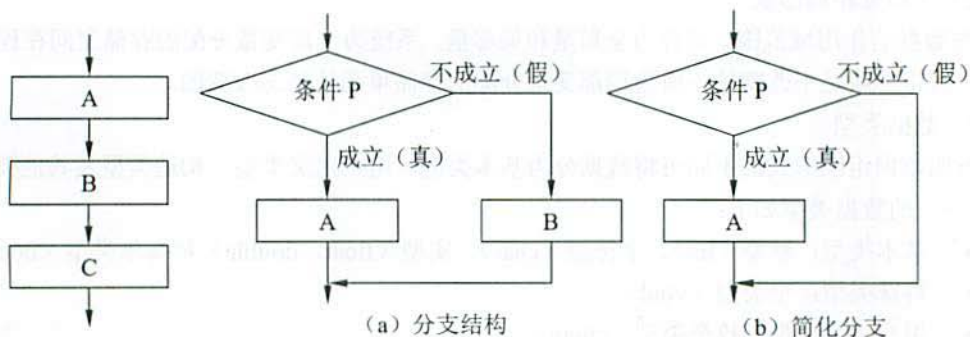


图 2-1 顺序结构示意图

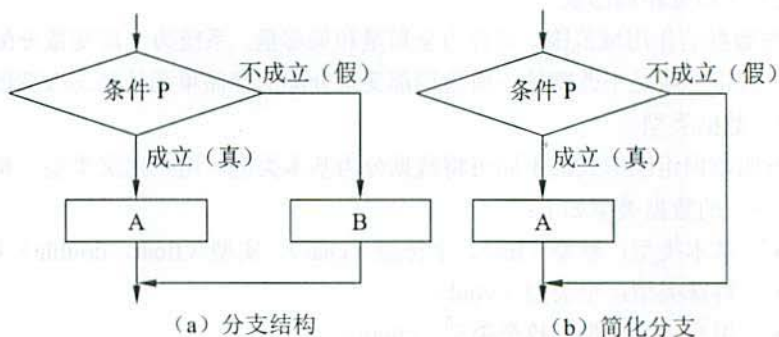


图 2-2 选择结构示意图

3) 循环结构

循环结构描述了重复计算的过程,通常由 3 部分组成:初始化、需要重复计算的部分和重复的条件,其中初始化部分有时在控制的逻辑结构中不进行显式的表示。重复结构主要有两种形式: while 型重复结构和 do-while 型重复结构。while 型结构的逻辑含义是先判断条件 P,若成立,则执行需要重复的计算 A,然后再去判断重复条件;否则控制流就退出重复结构,如图

2-3 (a) 所示。do-while 型结构的逻辑含义是先执行需要重复进行的计算 A, 然后再判断条件 P, 若成立则继续执行计算 A 的过程并判断条件; 否则控制流就退出重复结构, 如图 2-3 (b) 所示。

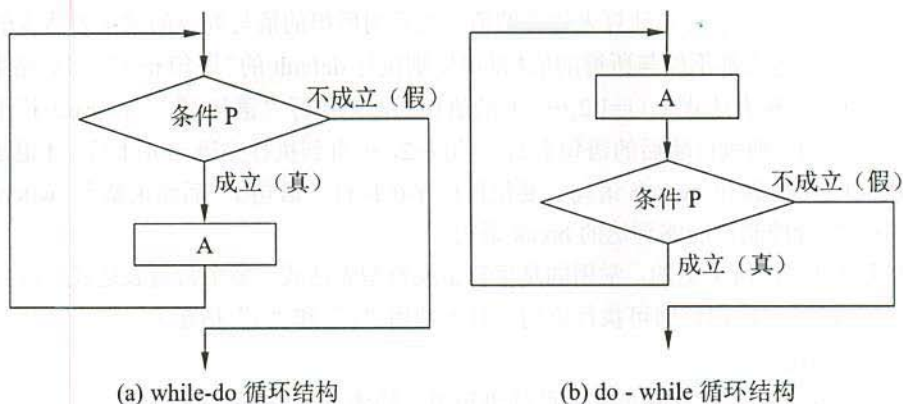


图 2-3 循环结构示意图

4) C (C++) 提供的控制结构语句

(1) 复合语句

复合语句用于描述顺序控制结构。复合语句是一系列用“{”和“}”括起来的声明和语句, 其主要作用是将多条语句组成一个可执行单元。语法上能出现语句的地方都可以使用复合语句。复合语句是一个整体, 要么全部执行, 要么一条语句也不执行。

(2) if 语句和 switch 语句

① if 语句实现的是双分支的选择结构, 其一般形式为:

```
if (表达式) 语句 1; else 语句 2;
```

其中语句 1 和语句 2 可以是任何合法的 C (C++) 语句, 当语句 2 为空语句时, 可以简化为: if (表达式) 语句 1;

使用 if 语句时, 需要注意的是 if 和 else 的匹配关系。C 语言规定, else 总是与离它最近的尚没有 else 的 if 相匹配。

② switch 语句描述了多分支的选择结构, 其一般形式为:

```
switch (表达式) {
    case 常量表达式 1: 语句 1;
    case 常量表达式 2: 语句 2;
    ...
    case 常量表达式 n: 语句 n;
```


default: 语句 n+1;

}

执行 switch 语句时,首先计算表达式的值,然后用所得的值与列举的常量表达式值依次比较,若任一常量表达式都不能与所得的值相匹配,则执行 default 的“语句 n+1”,然后结束 switch 语句。若此值与常量表达式 I ($i=1,2,\dots,n$) 的值相同,则执行“语句 i”,当“case i”的语句 i 中无 break 语句时,则执行随后的语句 i+1,语句 i+2, …直到执行完语句 n+1 后,才退出 switch 语句;或者以 break 跳出 switch 语句。要使得程序在执行“语句 i”后结束整个 switch 语句,则语句 i 中应包含控制流能够到达的 break 语句。

表达式可以是任何类型的,常用的是字符型或整型表达式。多个常量表达式可以共用一个语句组。语句组可以包括任何可执行语句,且无须用“{”和“}”括起来。

(3) 循环语句

C (C++) 语言提供了 3 种形式的循环语句用于描述重复型的控制结构。

① while 语句。while 语句描述了先判断条件再执行重复计算的控制结构, while 语句的一般形式是:

while (条件表达式) 循环体语句;

其中,循环体语句是内嵌的语句,当循环体语句多于一条时,应使用“{”和“}”括起来。执行 while 语句时,先计算条件表达式的值,当值为非 0 时,就执行循环体语句,然后重新计算条件表达式的值后再进行判断,否则就结束 while 语句的执行过程。

② do-while 语句。do-while 语句描述了先执行需要重复的计算再判断条件的控制结构,其一般格式是:

do

循环体语句;

while (条件表达式);

执行 do-while 语句时,先执行内嵌的循环体语句,然后再计算条件表达式的值,若值为非 0,则再一次地执行循环体语句和计算条件表达式并进行判断,直到条件表达式的值为 0 时,才结束 do-while 语句的执行过程。

③ for 语句。for 语句的基本格式是:

for (表达式 1; 表达式 2; 表达式 3) 循环体语句;

可用 while 语句等价地表示为:

表达式 1:

```
while (表达式 2) {
    循环体语句;
    表达式 3;
}
```

for 语句的使用是很灵活的,其内部的 3 个表达式都可以省略,但用于分隔 3 个表达式的分号“;”不能遗漏。

实际上,在 for 语句的语义中,表达式 1 只计算一次,其作用相当于进行 for 语句中一些变量的初始化,因此可以在 for 语句中省略,将其功能放在 for 语句之前实现即可。表达式 2 的值是判断是否执行循环体语句的依据,如果默认,则认为该表达式的值为非 0,for 语句将无法终止,除非在循环体语句中加入终止循环的 break 语句。表达式 3 用于修改可以改变表达式 2 的值的变量,若省略,其功能可放在循环体语句部分实现。

C 语言中还提供了实现控制流跳转的 goto 语句,由于它会破坏程序的逻辑结构,因此不提倡使用。

程序语言的传输成分指明语言允许的数据传输方式,如数据的输入和输出等。

4. 函数

C 程序由一个或多个函数组成,每个函数都有一个名字,其中有且仅有一个名字为 main 的函数,作为程序运行时的起点。函数是程序模块的主要成分,它是一段具有独立功能的程序。函数的使用涉及 3 个概念:函数定义、函数声明和函数调用。

1) 函数定义

函数的定义包括两部分:函数首部和函数体。函数的定义描述了函数做什么和怎么做。函数定义的一般格式是:

```
返回值的类型  函数名(形式参数表)  //函数首部
{
    函数体;
}
```

函数首部说明了函数返回值的数据类型、函数的名字和函数运行时所需的参数及类型。函数所实现的功能在函数体部分进行描述。

C (C++) 程序中所有函数的定义都是独立的。在一个函数的定义中不允许定义另外一个函数,也就是不允许函数的嵌套定义。

2) 函数声明

函数应该先声明后引用。如果程序中对一个函数的调用在该函数的定义之前进行,则应该



在调用前对被调用函数进行声明。函数原型用于声明函数。函数声明的一般形式为:

返回值类型 函数名 (参数类型表);

使用函数原型的目的在于告诉编译器传递给函数的参数个数、类型以及函数返回值的类型,参数表中仅需要依次列出函数定义时参数的类型。函数原型可以使编译器更彻底地检查源程序中对函数的调用是否正确。

3) 函数调用

当在一个函数(称为主调函数)中需要使用另一个函数(称为被调函数)实现的功能时,便以名字进行调用,称为函数调用。在使用一个函数时,只要知道如何调用就可以了,并不需要关心被调用函数的内部实现。因此,主调函数需要知道被调函数的名字、返回值和需要向被调函数传递的参数(个数、类型、顺序)等信息。函数调用的一般形式为:

函数名(实参表);

在C程序的执行过程中,通过函数调用实现了函数定义时描述的功能。函数体中若调用自己,则称为递归调用。

C和C++通过传值方式将实际参数传递给形式参数。调用函数和被调用函数之间交换信息的方法主要有两种:一种是由被调用函数把返回值返回给主调函数,另一种是通过参数带回信息。函数调用时实参与形参间交换信息的方法有值调用和引用调用两种。

(1) 值调用 (call by value): 若实现函数调用时实参向形式参数传递相应类型的值,则称为是传值调用。这种方式下形式参数不能向实际参数传递信息。

在C语言中,要实现被调用函数对实际参数的修改,必须用指针作形参。即调用时需要先对实参进行取地址运算,然后将实参的地址传递给指针形参。本质上仍属于传值调用。这种方式实现了间接内存访问。

(2) 引用调用 call by reference: 引用是C++中增加的数据类型,当形式参数为引用类型时,函数中对形参的访问和修改实际上就是针对相应实际参数所作的访问和改变。例如:

```
void swap(int &x, int &y) { /*交换 x 和 y*/  
    int temp;  
    temp=x;  x=y;  y=temp;  
}
```

函数调用: `↓ swap(a, b);`

引用调用时将实际参数的地址传递给形式参数,使得形参的地址就是对应实参的地址。在实现调用 `swap(a, b)` 时, `x`、`y` 就是 `a`、`b` 的别名,因此,函数调用完成后,交换了 `a` 和 `b` 的值。

2.2 语言处理程序基础

语言处理程序是一类系统软件的总称,其主要作用是将高级语言或汇编语言编写的程序翻译成某种机器语言程序,使程序可在计算机上运行。语言处理程序主要分为汇编程序、编译程序和解释程序等3种基本类型。

2.2.1 汇编语言基本原理

1. 汇编语言

汇编语言是为特定的计算机或计算机系统设计的面向机器的符号化的程序设计语言。用汇编语言编写的程序称为汇编语言源程序。因为计算机不能直接识别和运行符号语言程序,所以要用专门的翻译程序——汇编程序进行翻译。用汇编语言编写程序要遵循所用语言的规范和约定。

汇编语言源程序由若干条语句组成,一个程序中可以有3类语句:指令语句、伪指令语句和宏指令语句。

1) 指令语句

指令语句又称为机器指令语句,将其汇编后能产生相应的机器代码,这些代码能被CPU直接识别并执行相应的操作。基本的指令如ADD、SUB、AND等,书写指令语句时必须遵循指令的格式要求。

指令语句可分为传送指令、算术运算指令、逻辑运算指令、移位指令、转移指令和处理机控制指令等类型。

2) 伪指令语句

伪指令语句指示汇编程序在汇编源程序时完成某些工作,比如给变量分配存储单元地址,给某个符号赋一个值等。伪指令语句与指令语句的区别是:伪指令语句经汇编后不产生机器代码,而指令语句经汇编后要产生相应的机器代码;另外,伪指令语句所指示的操作是在源程序被汇编时完成的,而指令语句的操作必须在程序运行时完成。

通常汇编语言都应设立下面的一些伪指令:

(1) 常数定义伪指令语句:例如,IBM360/370的汇编语言中定义常数语句的格式是:

名字 DC x,y,...

其中,DC是语句的记忆码,x,y等规定了用户定义的常数。汇编程序应把所定义的常数按规定依次装入以名字为起始地址的一系列单元中。虚拟机COMET上的汇编语言CASL也设立了DC伪指令来指定和存储常数。



(2) 存储定义伪指令语句: IBM360/370 的汇编语言中用 DS 语句来定义内存单元, 例如:

```
TAB1 DS 100C
```

其中, 100 表示指定存储区域能存储的字节数, TAB1 代表被分配的存储区域的首地址。CASL 语言也设立了定义存储单元的伪指令 DS。

(3) 开始伪指令语句: 开始语句用来指出源程序段的开头; IBM360/370 汇编语言和 CASL 语言中的伪指令 START 就是这样的语句。

(4) 结束伪指令语句: 结束语句用来指出源程序段的结束; IBM360/370 汇编语言和 CASL 语言都设立了 END 伪指令表示源程序结束。

每一条汇编指令语句被划分成 4 个区, 依次是标号区、操作码区、操作数区和注解区; 各个区域之间用确定的符号分隔开。对于标号区中的标号, 程序员用它来指示一条汇编指令语句, 它实际上代表该指令的内存单元地址。操作码区是该语句的指令助记符, 它可以是机器指令助记符、伪指令码等。操作数区指出本条汇编指令所操作的运算对象, 用寻址方式指定操作数的来源, 常用的是寄存器操作数和内存单元操作数。

3) 宏指令语句

在汇编语言中, 还允许用户将多次重复使用的程序段定义为宏。宏的定义必须按照相应的规定进行, 每个宏都有相应的宏名。在程序的任意位置, 若需要使用这段程序, 只要在相应的位置使用宏名, 即相当于使用了这段程序。因此, 宏指令语句就是宏的引用。

CASL 汇编语言提供的宏指令用于实现输入、输出和程序结束的功能, 在用户程序中直接引用即可。

2. 汇编程序

汇编程序的功能是将汇编语言所编写的源程序翻译成机器指令程序。汇编程序的基本工作包括: 将每一条可执行汇编语句转换成对应的机器指令; 处理源程序中出现的伪指令。由于汇编指令中形成操作数地址的部分可能出现后面才会有定义的符号, 所以汇编程序一般需要两次扫描源程序才能完成翻译过程。

第一次扫描的主要工作是定义符号的值并创建一个符号表 ST。ST 记录了汇编时所遇到的符号的值。另外, 有一个固定的机器指令表 MOT1, 其中记录了每条机器指令的记忆码和指令的长度。在汇编程序翻译源程序的过程中, 为了计算各汇编语句中标号的地址, 需要设立一个位置计数器或单元地址计数器 LC (Location Counter), 其初值一般为 0。在扫描源程序时, 每处理完一条机器指令或与存储分配有关的伪指令 (如定义常数语句、定义储存语句), LC 的值就增加相应的长度。这样, 在汇编过程中, LC 的内容就是下一条被汇编的指令的偏移地址。若正在汇编的语句是有标号的, 则该标号的值就取 LC 的当前值。

此外,在第一次扫描中,还需要对与定义符号值有关的伪指令进行处理。为了叙述方便,不妨设立伪指令表 POT1。POT1 表的每一个元素只有两个域:伪指令助记符和相应的子程序入口。下面的步骤①~⑤描述了汇编程序第一次扫描源程序的过程。

- ① 单元计数器 LC 置初值 0;
- ② 打开源程序文件;
- ③ 从源程序中读入第一条语句;
- ④ while (若当前语句不是 END 语句){
 - if (当前语句有标号) 则将标号和单元计数器 LC 的当前值填入符号表 ST;
 - if (当前语句是可执行的汇编指令语句) 则查找 MOT1 表获得当前指令的长度 K, 并令 $LC = LC + K$;
 - if (当前指令是伪指令) 则查找 POT1 表并调用相应的子程序;
 - if (当前指令的操作码是非法记忆码) 则调用出错处理子程序;
 - 从源程序中读入下一条语句;
- ⑤ 关闭源程序文件。

第二次扫描的任务是产生目标程序。除了使用前一次扫描所生成的符号表 ST 外,还要使用机器指令表 MOT2,该表中的元素有:机器指令助记符、机器指令的二进制操作码(binary-code),格式(type)和长度(length)。此外,还要设立一个伪指令表 POT2,供第二次扫描时使用,POT2 的每一元素仍有两个域:伪指令记忆码和相应的子程序入口。与第一次扫描的不同之处是:在第二次扫描中,伪指令有着完全不同的处理。

在第二次扫描中,可执行汇编语句应被翻译成对应的二进制代码机器指令。这一过程涉及两个方面的工作:一是把机器指令助记符转换成二进制机器指令操作码,这可通过查找 MOT2 表来实现;二是求出操作数区各操作数的值(用二进制表示)。在此基础上,就可以装配出用二进制代码表示的机器指令。从求值的角度看,第二部分工作并不复杂。由于形成操作数地址的各个部分都以表达式形式出现,只要定义一个过程 eval-expr(index, value),其功能是:通过 index 给定一个表达式在汇编语句缓冲区 S 的开始位置,该过程就用 value 返回此表达式的值。例如,虚拟计算机 COMET 的机器指令可归属于“X”型指令,其汇编语句为:

```
OP R1,N2,X2
OP R1,N2
```

可以写出下面处理“X”型指令的程序段(假定 index 已指向操作数在缓冲区 S 的首地址):

```
eval-expr(index,R1);
index:=index+1;
```



```

eval-expr(index,N2);
if S[index]=',' then
begin
    index:=index+1;
    eval-expr(index,X2);
end
else
    X2:=0;

```

类似地可以写出其他类型指令处理操作数的程序段。设当前可执行汇编语句的操作助记符在 MOT2 表的索引值为 i, 则整个可执行汇编语句的处理可以描述如下:

```

OP:=MOT2[i].binary-code;
TYPE:=MOT2[i].type;
case TYPE of
    'X': 求 X 型指令操作数各个部分值, 然后按规定字节形成指令;
    ...
end;
将形成的指令送往输出区;

```

在第二次扫描中, 根据伪指令助记符, 调用 POT2 表相应元素所规定的子程序。DS 伪指令的主要目的是预留存储空间。不妨设一个工作单元 K, 用以累计以字节为单位的存储空间大小, 初值为 0。从 DS 伪指令的操作数区求出 K 的大小后, 就向输出区送 K 个空格以达到保留所规定存储单元的目的。DC 伪指令处理的结果是向输出区送出转换得到的常量。最后, 开始伪指令工作是输出目标程序开始的标准信息, 而结束伪指令则是输出目标程序结束的标准信息, 这些信息都是为装配程序提供的。

2.2.2 编译程序基本原理

1. 编译过程概述

编译程序的功能是把某高级语言书写的源程序翻译成与之等价的目标程序(汇编语言或机器语言)。编译程序的工作过程可以分为 6 个阶段, 如图 2-4 所示, 实际的编译器中可能会将其中的某些阶段结合在一起进行处理。

下面简要介绍各阶段实现的主要功能。

1) 词法分析阶段

源程序可以简单地被看成是一个多行的字符串。词法分析阶段是编译过程的第一阶段, 这个阶段的任务是对源程序从前到后(从左到右)逐个字符地扫描, 从中识别出一个个“单词”符号。“单词”符号是程序设计语言的基本语法单位, 如: 关键字(或称保留字)、标识符、常

数、运算符和分隔符(如标点符号、左右括号)等。词法分析程序输出的“单词”常以三元组的方式输出,即单词种别和单词自身的值。

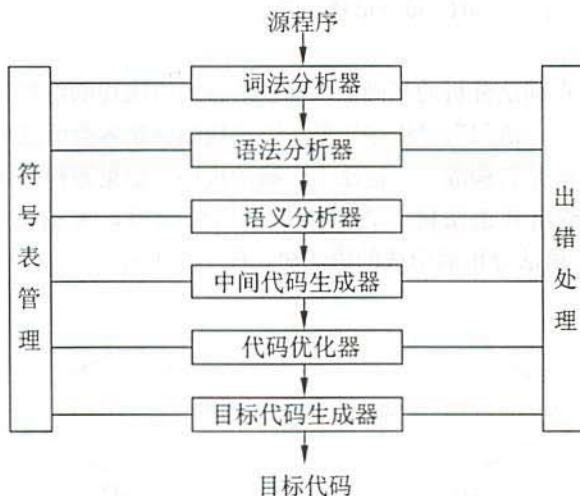


图 2-4 编译器的工作阶段示意图

词法分析过程依据的是语言的词法规则,即描述“单词”结构的规则。例如对于某 PASCAL 源程序中的一条声明语句和赋值语句:

```
VAR X,Y,Z: real;
X:=Y+Z*60;
```

词法分析阶段将构成这条语句的字符串分割成如下的单词序列:

- | | | | |
|----------|-----|-----------|------|
| (1) 保留字 | VAR | (2) 标识符 | X |
| (3) 逗号 | , | (4) 标识符 | Y |
| (5) 逗号 | , | (6) 标识符 | Z |
| (7) 冒号 | : | (8) 标准标识符 | real |
| (9) 分号 | ; | (10) 标识符 | X |
| (11) 赋值号 | := | (12) 标识符 | Y |
| (13) 加号 | + | (14) 标识符 | Z |
| (15) 乘号 | * | (16) 整常数 | 60 |
| (17) 分号 | ; | | |

对于标识符 X、Y、Z,其单词种别都是 id(标识符),字符串“X”、“Y”和“Z”都是单词的值;而对于单词 60,整常数是该单词的种别,60 是该单词的值。这里,我们用 id1、id2



和 id3 分别代表 X、Y 和 Z, 强调标识符的内部标识由于组成该标识符的字符串不同而有所区别。经过词法分析后, 声明语句“VAR X,Y, Z: real;”表示为“VAR id1,id2,id3:real;”; 赋值语句“X:=Y+Z*60;”表示为“id1:=id2+id3*60;”。

2) 语法分析阶段

语法分析的任务是在词法分析的基础上, 根据语言的语法规则将单词符号序列分解成各类语法单位, 如“表达式”、“语句”、“程序”等。语法规则就是各类语法单位的构成规则。通过语法分析确定整个输入串是否构成一个语法上正确的程序。如果源程序中没有语法错误, 语法分析后就能正确地构造出其语法树; 否则就指出语法错误, 并给出相应的诊断信息。对“id1:=id2+id1*60”进行语法分析后形成的语法树如图 2-5 所示。

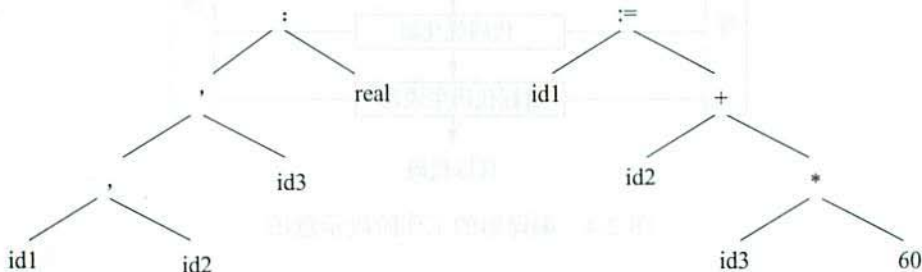


图 2-5 语法树示意图

词法分析和语法分析本质上都是对源程序的结构进行分析。

3) 语义分析阶段

语义分析阶段主要检查源程序是否包含语义错误, 并收集类型信息供后面的代码生成阶段使用。只有语法和语义都正确的源程序才能被翻译成正确的目标代码。

语义分析的一个主要工作是进行类型分析和检查。程序语言中的一个数据类型一般包含两个方面的内容: 类型的载体及其上的运算。例如: 整除取余运算符只能对整型数据进行运算, 若其运算对象中有浮点数就认为是类型不匹配的错误。

在确认源程序的语法和语义之后, 就可对其进行翻译并改变源程序的内部表示。对于声明语句, 需要记录所遇到的符号的信息, 所以应进行符号表的填查工作。在图 2-6 所示的符号表中, 每一行存放一个符号的信息。第一行存放标识符 X 的信息, 其类型为 real, 为它分配的地址是 0。第二行存放 y 的信息, 它的类型是 real, 为它分配的地址是 4。因此, 在这种语言中, 为一个 real 型数据分配的存储空间是 4 个存储单元。对于可执行语句, 则检查结构合理的表达式是否有意义。对“id1:=id2+id1*60”进行语义分析后的语法树如图 2-6 所示, 其中增加了一个语义处理节点 intto real, 该运算用于将一个整型数转换为浮点数。

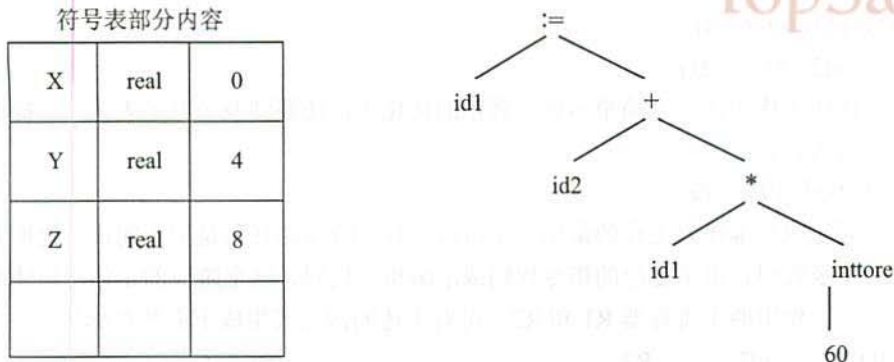


图 2-6 语义分析后的符号表和语法树示意图

4) 中间代码生成阶段

中间代码生成阶段的工作是根据语义分析的输出生成中间代码。“中间代码”是一种简单且含义明确的记号系统，可以有若干种形式，它们的共同特征是与具体的机器无关。中间代码的设计原则主要有两点：一是容易生成，二是容易被翻译成目标代码。最常用的一种中间代码是与汇编语言的指令非常相似的三地址码，其实现方式常采用四元式。四元式的形式为：(运算符，运算对象 1，运算对象 2，运算结果)。

例如对语句“X:=Y+Z*60”，可生成以下四元式序列：

- ① (inttoreal, 60, -, t1)
- ② (*, id3, t1, t2)
- ③ (+, id2, t2, t3)
- ④ (:=, t3, -, id1)

其中 t_i ($i=1, 2, 3$) 是编译程序生成的临时变量，用于存放临时的运算结果。

语义分析和中间代码生成所依据的是语言的语义规则。

5) 代码优化阶段

优化是一个编译器的重要组成部分，由于编译器将源程序翻译成中间代码的工作是机械的、按固定模式进行的，因此，生成的中间代码往往在时间上和空间上有很大的浪费。当需要生成高效的目标代码时，就必须进行优化。优化过程可以在中间代码生成阶段进行，也可以在目标代码生成阶段进行。由于中间代码是不依赖于具体机器的，此时所作的优化一般建立在对程序的控制流和数据流分析的基础之上，与具体的机器无关。优化所依据的原则是程序的等价变换规则。例如，在生成“X:=Y+Z*60”的四元式后，60 是编译时已知的常数，把它转换为 60.0 的工作可以在编译时完成，没有必要生成一个四元式，同时 t_3 仅仅用来将其值传递给 id_1 ，也可以被化简掉，因此上述的中间代码可转优化成下面的等价代码：

① (*, id3, 60.0, t1)

② (+, id2, t1, id1)

这只是优化工作中的一个简单示例,真正的优化工作还要涉及公共子表达式的提取、循环优化等更多的内容。

6) 目标代码生成阶段

目标代码生成是编译器工作的最后一个阶段。这一阶段的任务是把中间代码转换成特定机器上的绝对指令代码、可重定位的指令代码或汇编指令代码,这个阶段的工作与具体的机器密切相关。例如,使用两个寄存器 R1 和 R2,可对上述的四元式生成下面的目标代码:

① MOVF id3, R2

② MULF #60.0, R2

③ MOVF id2, R1

④ ADDF R2, R1

⑤ MOV R1, id1

这里用#表明 60.0 为常数。

7) 符号表管理

符号表的作用是记录源程序中各个符号的必要信息,以辅助语义的正确性检查和代码生成,在编译过程中需要对符号表进行快速有效地查找、插入、修改和删除等操作。符号表的建立可以始于词法分析阶段,也可以放到语法分析和语义分析阶段,但符号表的使用有时会延续到目标代码的运行阶段。

8) 出错处理

用户编写的源程序不可避免地会有一些错误,这些错误大致可分为静态错误和动态错误。动态错误也称动态语义错误,指程序中包含的逻辑错误,它们发生在程序运行时,例如变量取零时作除数、引用数组元素下标错误等。静态错误是指编译阶段发现的程序错误,可分为语法错误和静态语义错误,如单词拼写错误、标点符号错误、表达式中缺少操作数、括号不匹配等有关语言结构上的错误称为语法错误,而语义分析时发现的运算符与运算对象类型不合法等错误属于静态语义错误。

在编译时发现程序中的错误后,编译程序应采用适当的策略修复它们,使得分析过程能够继续下去,以便在一次编译过程中尽可能多地找出程序中的错误。

对于编译器的各个阶段,在逻辑上可以把它划分为前端和后端两部分。前端包括从词法分析到中间代码生成各阶段的工作,后端包括中间代码优化和目标代码的生成、优化等阶段。这样,以中间代码为分水岭,把编译器分成了与机器有关的部分和与机器无关的部分。如此以来,对于同一种程序语言的编译器,开发出一个前端之后,就可以针对不同的机器开发相应的后端,前后端有机结合后就形成了该语言的一个编译器。当语言有改动时,只会涉及前端部分

的维护。对于不同的程序语言,分别设计出相应的前端,然后将各个语言的前端与同一个后端相结合,就可得到各个语言在某种机器上的编译器。

2. 文法和语言的形式描述

1) 字母表、字符串、字符串集合及运算

- 字母表 Σ : 是元素的非空有穷集合。例如 $\Sigma = \{a, b\}$ 。
- 字符: 字母表 Σ 中的一个元素。例如 Σ 上的 a 或 b 。
- 字符串: Σ 中的字符组成的有穷序列。例如 a, ab, aaa 都是 Σ 上的字符串。
- 字符串的长度: 指字符串中的字符个数。如 $|aba|=3$ 。
- 空串 ε : 由零个字符组成的序列, $|\varepsilon|=0$ 。
- 连接: 字符串 S 和 T 的连接是指将串 T 接续在串 S 之后,表示为 $S \cdot T$, 连接符号 “ \cdot ” 可省略。显然对于字母表 Σ 上的任意字符串 S , $S \cdot \varepsilon = \varepsilon \cdot S = S$ 。
- 空集: 用符号 ϕ 表示。
- Σ^* : 是指包括空串 ε 在内的 Σ 上所有字符串的集合。例如, 设 $\Sigma = \{a, b\}$, $\Sigma^* = \{\varepsilon, a, b, aa, bb, ab, ba, aaa, \dots\}$ 。
- 字符串的方幂: 把字符串 α 自身连接 n 次得到的串,称为字符串 α 的 n 次方幂,记为 α^n 。
 $\alpha^0 = \varepsilon, \alpha^n = \alpha \alpha^{n-1} = \alpha^{n-1} \alpha \quad (n > 0)$ 。
- 字符串集合的运算: 设 A, B 代表字母表 Σ 上的两个字符串集合。
或(合并): $A \cup B = \{\alpha \mid \alpha \in A \text{ 或 } \alpha \in B\}$ 。
积(连接): $AB = \{\alpha\beta \mid \alpha \in A \text{ 且 } \beta \in B\}$ 。
幂: $A^n = A \cdot A^{n-1} = A^{n-1} \cdot A \quad (n > 0)$, 并规定 $A^0 = \{\varepsilon\}$ 。
正则闭包+: $A^+ = A^1 \cup A^2 \cup A^3 \cup \dots \cup A^n \cup \dots$ 。
闭包*: $A^* = A^0 \cup A^+$ 。显然, $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots \cup \Sigma^n \cup \dots$ 。

2) 文法和语言的形式描述

语言 L 是有限字母表 Σ 上有限长度字符串的集合,这个集合中的每个符号串都是按照一定的规则生成的。下面从产生语言的角度出发,给出文法和语言的形式定义。所谓产生语言,是指指定出有限个规则,借助它们就能产生此语言的全部句子。

(1) 文法的定义: 描述语言语法结构的形式规则称为文法。文法 G 是一个四元组,可表示为: $G = (V_N, V_T, P, S)$, 其中 V_T 是一个非空有限集,其中的每个元素称为一个终结符; V_N 是一个非空有限集,其每个元素称为非终结符。 $V_N \cap V_T = \phi$, 即 V_N 和 V_T 不含公共元素。令 $V = V_N \cup V_T$, 称 V 为文法 G 的词汇表, V 中的符号称为文法符号,包括终结符和非终结符。 $S \in V_N$, 称为开始符号; 它至少要在一条产生式中作为左部出现。 P 是产生式的有限集合,每

一个产生式是形如“ $\alpha \rightarrow \beta$ ”的规则,其中, α 称为产生式的左部, $\alpha \in V^+$ 且 α 中至少含有一个非终结符; β 称为产生式的右部且 $\beta \in V^*$ 。若干个产生式 $\alpha \rightarrow \beta_1, \alpha \rightarrow \beta_2, \dots, \alpha \rightarrow \beta_n$ 的左部相同时,可简写为 $\alpha \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$;称 $\beta_i (1 \leq i \leq n)$ 为 α 的一个候选式。

(2) 文法的分类:乔姆斯基(Chomsky)把文法分成4种类型,即0型、1型、2型和3型。这4类文法之间的差别在于对产生式施加不同的限制。若文法 $G = (V_N, V_T, P, S)$ 的每个产生式 $\alpha \rightarrow \beta$,均有 $\alpha \in (V_N \cup V_T)^*$, α 至少含有一个非终结符,且 $\beta \in (V_N \cup V_T)^*$,则称 G 为0型文法。对0型文法的每条产生式分别施加以下的第 i 条限制,则可得 i 型文法:

- G 的任何产生式 $\alpha \rightarrow \beta$ ($S \rightarrow \varepsilon$ 除外)均满足 $|\alpha| \leq |\beta|$ ($|x|$ 表示 x 中文法符号的个数);
- G 的任何产生式形如 $A \rightarrow \beta$,其中 $A \in V_N$, $\beta \in (V_N \cup V_T)^*$;
- G 的任何产生式形如 $A \rightarrow a$ 或 $A \rightarrow aB$ (或者 $A \rightarrow Ba$),其中 $A, B \in V_N$, $a \in V_T$ 。

0型文法也称为短语文法,其能力相当于图灵机,任何0型语言都是递归可枚举的;反之,递归可枚举集也必定是一个0型语言。1型文法也称为上下文有关文法,这种文法意味着对非终结符的替换必须考虑上下文,并且一般不允许替换成 ε 串。例如,若 $\alpha A \beta \rightarrow \alpha \gamma \beta$ 是1型文法的产生式, α 和 β 不全为空,则非终结符 A 只有在左边是 α ,右边是 β 的上下文中才能替换成 γ 。2型文法就是上下文无关文法,非终结符的替换无须考虑上下文。3型文法等价于正规式,因此也被称为正规文法或线性文法。

(3) 句子和语言:设有文法 $G = (V_N, V_T, P, S)$,下面引入一些概念:

- 推导与直接推导:推导就是从文法的开始符号 S 出发,反复使用产生式,将产生式左部的非终结符替换为右部的文法符号序列(展开产生式用 \Rightarrow 表示),直到产生一个终结符的序列时为止。若有产生式 $\alpha \rightarrow \beta \in P, \gamma, \delta \in V^*$,则 $\gamma \alpha \delta \Rightarrow \gamma \beta \delta$ 称为文法 G 中的一个直接推导,并称 $\gamma \alpha \delta$ 可直接推导出 $\gamma \beta \delta$ 。显然,对 P 中的每一个产生式 $\alpha \rightarrow \beta$ 都有 $\alpha \Rightarrow \beta$ 。若在文法中存在一个直接推导序列,即 $\alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n (n > 0)$,则称 α_0 可推导出 α_n , α_n 是 α_0 的一个推导,并记为 $\alpha_0 \xRightarrow{+}_G \alpha_n$ 。我们用记号 $\alpha_0 \xRightarrow{+}_G \alpha_n$ 表示 $\alpha_0 = \alpha_n$ 或者 $\alpha_0 \xRightarrow{+}_G \alpha_n$ 。
- 直接归约和归约(推导的逆过程):若文法 G 中有一个直接推导 $\alpha \Rightarrow \beta$,则称 β 可直接归约成 α ,或 α 是 β 的一个直接归约。若文法 G 中有一个推导 $\gamma \xRightarrow{*}_G \delta$,则称 δ 可归约成 γ ,或 γ 是 δ 的一个归约。
- 句型 and 句子:若文法 G 的开始符号为 S ,那么,从开始符号 S 能推导出的符号串称为文法的一个句型,即 α 是文法 G 的一个句型,当且仅当有如下推导 $S \xRightarrow{*}_G \alpha, \alpha \in V^*$ 。若

X 是文法 G 的一个句型, 且 $X \in V_T^*$, 则称 X 是文法 G 的一个句子, 即仅含终结符的句型是一个句子。

- 语言: 从文法 G 的开始符号出发, 所能推导出的句子的全体称为文法 G 产生的语言, 记为 $L(G)$, 它可以表示成: $L(G) = \{X \mid S \xRightarrow{*}_G X, X \in V_T^*\}$ 。

(4) 文法的等价: 若文法 G_1 与文法 G_2 产生的语言是相同的, 即 $L(G_1) = L(G_2)$, 则称这两个文法是等价的。

在形式语言理论和编译理论中, 文法等价是一个很重要的概念, 根据这一概念, 我们可对文法进行等价改造, 以得到所需形式的文法。

3. 词法分析

语言中具有独立含义的最小语法单位是符号(单词), 如标识符、无符号常数与界限符等。词法分析的任务是把构成源程序的字符串转换成单词符号序列。词法规则可用 3 型文法(正规文法)或正规表达式描述, 它产生的集合是语言基本字符集 Σ (字母表) 上的字符串的一个子集, 我们称为正规集。

1) 正规表达式和正规集

对于字母表 Σ , 其上的正规式及其表示的正规集可以递归定义如下:

- (1) ε 是一个正规式, 它表示集合 $L(\varepsilon) = \{\varepsilon\}$;
- (2) 若 a 是 Σ 上的字符, 则 a 是一个正规式, 它所表示的正规集为 $\{a\}$;
- (3) 若正规式 r 和 s 分别表示正规集 $L(r)$ 和 $L(s)$, 则
 - $r|s$ 是正规式, 表示集合 $L(r) \cup L(s)$;
 - $r \cdot s$ 是正规式, 表示集合 $L(r)L(s)$;
 - r^* 是正规式, 表示集合 $(L(r))^*$;
 - (r) 是正规式, 表示集合 $L(r)$;

仅由有限次地使用上述 3 个步骤定义的表达式才是 Σ 上的正规式, 其中运算符 “|”、“ \cdot ”、“ $*$ ” 分别称为 “或”、“连接” 和 “闭包”。在正规式的书写中, 连接运算符 “ \cdot ” 可省略。运算符的优先级从高到低顺序排列为: “ $*$ ”、“ \cdot ”、“|”。

设 $\Sigma = \{a, b\}$, 表 2-1 列出了 Σ 上的一些正规式和相应的正规集。

若两个正规式表示的正规集相同, 则认为二者等价。两个等价的正规式 U 和 V 记为 $U = V$ 。例如, $b(ab)^* = (ba)^*b$, $(a|b)^* = (a^*b^*)^*$ 。设 U , V 和 W 均为正规式, 正规式的代数性质如表 2-2 所示。



表 2-1 正规式与正规集

正规式	正规集
ab	符号串 ab 构成的集合
$a b$	符号串 a 、 b 构成的集合
a^*	由零个或多个 a 构成的符号串集合
$(a b)^*$	所有字符 a 和 b 构成的串的集合
$a(a b)^*$	以 a 为首字符的 a 、 b 字符串的集合
$(a b)^*abb$	以 abb 结尾的 a 、 b 字符串的集合

表 2-2 正规式的代数性质

公理	公理
$U V=V U$	$(UV)W=U(VW)$
$(U V) W=U (V W)$	$\varepsilon U=U\varepsilon=U$
$U(V W)=UV UW$	$V^*=(V^* \varepsilon)$
$(U V)W=UW VW$	$V^{**}=V^*$

2) 有限自动机

有限自动机是一种识别装置的抽象概念,它能准确地识别正规集。有限自动机分为两类:确定的有限自动机和不确定的有限自动机。

(1) 确定的有限自动机(Deterministic Finite Automata, DFA)。一个确定的有限自动机是个五元组 (S, Σ, f, s_0, Z) , 其中:

- S 是一个有限集, 其每个元素称为一个状态;
- Σ 是一个有穷字母表, 其每个元素称为一个输入字符;
- f 是从 $S \times \Sigma \rightarrow S$ 上的单值部分映像。 $f(A, a)=Q$ 表示当前状态为 A 、输入为 a 时, 将转换到下一状态 Q 。我们称 Q 为 A 的一个后继状态;
- $s_0 \in S$, 是唯一的一个开始状态;
- Z 是非空的终止状态集合, $Z \subseteq S$ 。

一个 DFA 可以用两种直观的方式表示: 状态转换图和状态转换矩阵。状态转换图简称为转换图, 它是一个有向图。DFA 中的每个状态对应转换图中的一个节点, DFA 中的每个转换函数对应图中的一条有向弧, 若转换函数为 $f(A, a)=Q$, 则该有向弧从节点 A 出发, 进入节点 Q , 字符 a 是弧上的标记。

【例 2.1】 已知有 DFA $M1 = (\{s_0, s_1, s_2, s_3\}, \{a, b\}, f, S, \{s_3\})$, 其中 f 为:

$$f(s_0, a)=s_1, f(s_0, b)=s_2, f(s_1, a)=s_3, f(s_1, b)=s_2, f(s_2, a)=s_1, f(s_2, b)=s_3, f(s_3, a)=s_3$$

与 DFA $M1$ 对应的状态图如图 2-7(a)所示, 其中, 双圈表示的节点是终态节点。状态转换矩阵可以用一个二维数组 M 表示, 矩阵元素 $M[A, a]$ 的行下标表示状态, 列下标表示输入字符,

$M[A, a]$ 的值是当前状态为 A 、输入为 a 时, 应转换到的下一状态。与 DFA M_1 对应的状态转换矩阵如图 2-7(b)所示。在转换矩阵中, 一般以第一行的行下标所对应的状态作为初态, 而终态则需要特别指明。

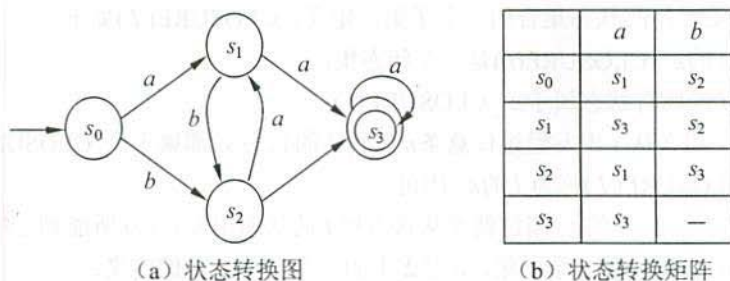


图 2-7 DFA 的状态转换图和转换矩阵

对于 Σ 中的任何字符串 ω , 若存在一条从初态节点到某一终止状态节点的路径, 且这条路径上所有弧的标记符连接成的字符串等于 ω , 则称 ω 可由 DFA M 识别 (接受或读出)。若一个 DFA M 的初态节点同时又是终态节点, 则空字 ε 可由该 DFA 识别 (或接受)。DFA M 所能识别的语言 $L(M) = \{\omega | \omega \text{ 是从 } M \text{ 的初态到终态的路径上的弧上标记所形成的串}\}$ 。

Σ 上的一个字符串集合 V 是正规的, 当且仅当存在 Σ 上的一个 DFA M , 且 $V = L(M)$ 。

(2) 不确定的有限自动机 (Nondeterministic Finite Automata, NFA)。一个不确定的有限自动机也是一个五元组, 它与确定有限自动机的区别是:

- f 是从 $S \times \Sigma \rightarrow 2^S$ 上的映像。对于 S 中的一个给定状态及输入符号, 返回一个状态的集合。即当前状态的后继状态不一定是唯一确定的。
- 有向弧上的标记可以是 ε 。

【例 2.2】 已知有 NFA $N = (\{s_0, s_1, s_2, s_3\}, \{a, b\}, f, S, \{s_3\})$, 其中 f 为:

$f(s_0, a) = s_0, f(s_0, a) = s_1, f(s_0, b) = s_0, f(s_1, b) = s_2, f(s_2, b) = s_3$

与 NFA M_2 对应的状态转换图和状态转换矩阵如图 2-8 所示。

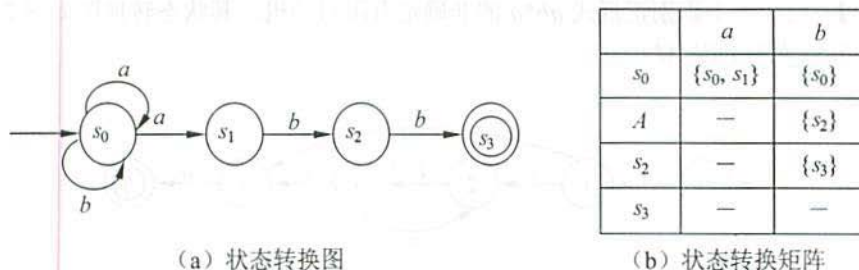


图 2-8 NFA 的状态转换图和转换矩阵

显然, DFA 是 NFA 的特例。实际上, 对于每个 NFA M , 都存在一个 DFA N , 且 $L(M)=L(N)$ 。对于任何两个有限自动机 $M1$ 和 $M2$, 如果 $L(M1)=L(M2)$, 则称 $M1$ 和 $M2$ 是等价的。

3) NFA 到 DFA 的转换

(1) 若 I 是 NFA N 的状态集合的一个子集, 定义 $\varepsilon_CLOSURE(I)$ 如下:

- 状态集 I 的 $\varepsilon_CLOSURE(I)$ 是一个状态集;
- 状态集 I 的所有状态属于 $\varepsilon_CLOSURE(I)$;
- 若 $s \in I$, 那么从 s 出发经过任意条 ε 弧到达的状态 s' 都属于 $\varepsilon_CLOSURE(I)$ 。

状态集 $\varepsilon_CLOSURE(I)$ 称为 I 的 ε 闭包。

由以上的定义可知, I 的 ε 闭包就是从状态集 I 的状态出发, 经 ε 所能到达的状态的全体。

假定 I 是 N 的状态集的一个子集, a 是 Σ 中的一个字符, 我们定义:

$$I_a = \varepsilon_CLOSURE(J)$$

其中, J 是所有那些可从 I 中的某一状态节点出发经过一条 a 弧而到达的状态节点的全体。

在定义了 $\varepsilon_CLOSURE$ 后, 我们就可以用子集法将一个非确定的有限自动机转化为一个确定的有限自动机。

(2) NFA 转换为 DFA。设 NFA $N = (S, \Sigma, f, s_0, Z)$, 与之等价的 DFA $M = (S', \Sigma, f', q_0, Z')$, 用子集法将非确定的有限自动机确定化的算法步骤如下:

- ① 求出 DFA M 的初态 q_0 , 即令 $q_0 = \varepsilon_CLOSURE(\{s_0\})$, 此时 S' 仅含初态 q_0 , 并且没有标记;
- ② 对于 S' 中尚未标记的状态 $q_i = \{s_{i1}, s_{i2}, \dots, s_{im}\}, s_{ij} \in S (j=1, \dots, m)$ 进行以下处理:

- 标记 q_i ;
- 对于每个 $a \in \Sigma$, 令 $T = f(\{s_{i1}, s_{i2}, \dots, s_{im}\}, a)$, $q_j = \varepsilon_CLOSURE(T)$;
- 若 q_j 尚不在 S' 中, 则将 q_j 作为一个未加标记的新状态添加到 S' , 并把状态转换函数 $Sf'(q_i, a) = q_j$ 添加到 DFA M 中;

③ 重复进行步骤②, 直到 S' 中不再有未标记的状态时为止。

④ 令 $Z' = \{q \mid q \in S' \text{ 且 } q \cap Z \neq \emptyset\}$

【例 2.3】 已知一个识别正规式 ab^*a 的非确定有限自动机, 其状态转换图如图 2-9 所示, 用子集法将其转换为 DFA M 。

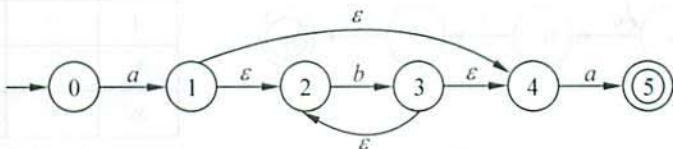


图 2-9 ab^*a 的 NFA 状态转换图

根据 $\varepsilon_CLOSURE$ 的定义,可以求出 $\varepsilon_CLOSURE(\{0\})=\{0\}$,将 $\{0\}$ 记为DFA的初态 q_0 ;然后根据题中所给的状态转换图以及算法步骤中的②求解DFA M 的各个状态的过程如下:

$\varepsilon_CLOSURE(q_0,a)=\{1,2,4\}$,将 $\{1,2,4\}$ 记为 q_1 ; $\varepsilon_CLOSURE(q_0,b)=\{\}$; 标记 q_0 ;

$\varepsilon_CLOSURE(q_1,a)=\{5\}$,将 $\{5\}$ 记为 q_2 (终态);

$\varepsilon_CLOSURE(q_1,b)=\{2,3,4\}$,将 $\{2,3,4\}$ 记为 q_3 ; 标记 q_1 ;

$\varepsilon_CLOSURE(q_2,a)=\{\}$; $\varepsilon_CLOSURE(q_2,b)=\{\}$; 标记 q_2 ;

$\varepsilon_CLOSURE(q_3,a)=\{5\}$,即 q_2 ;

$\varepsilon_CLOSURE(q_3,b)=\{2,3,4\}$,即 q_3 ; 标记 q_3 ;

当 S' 中不再有未标记的状态时,算法即可终止,所得的DFA M 如图2-10所示。

(3) DFA的最小化。从NFA转换得到的DFA不一定是最简化的,可以通过等价变换将DFA进行最小化处理。

对于有限自动机中的任何两个状态 t 和 s ,若从其中一个状态出发接受输入字符串 ω ,而从另一状态出发不接受 ω ,或者从 t 和 s 出发到达不同的接受状态,则称 ω 对状态 s 和 t 是可区分的。若状态 s 和 t 不可区分,则称其为可以合并的等价状态。

对图2-10所示的自动机进行化简所得的DFA如图2-11所示。

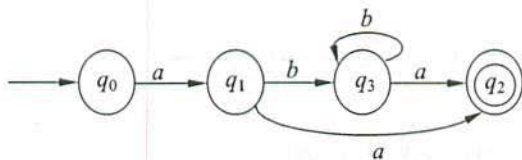


图 2-10 识别 ab^*a 的 DFA 示意图

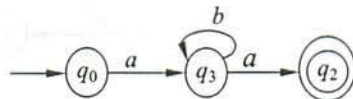


图 2-11 识别 ab^*a 的最小化 DFA 示意图

4) 正规式与有限自动机之间的转换

(1) 对于 Σ 上的NFA M ,可以构造一个 Σ 上的正规式 R ,使得 $L(R)=L(M)$ 。

我们把状态转换图的概念拓宽,令每条弧可用一个正规式作标记。为 Σ 上的NFA M 构造相应的正规式 R ,分为两步:

① 在 M 的状态转换图中加两个节点,一个 x 节点,一个 y 节点。从 x 节点到NFA M 的初始状态节点引一条弧并用 ε 标记,从NFA M 的所有终态节点到 y 节点引一条弧并用 ε 标记。形成一个与 M 等价的 M' , M' 只有一个初态 x 和一个终态 y 。

② 按下面的方法逐步消去 M' 中除 x 和 y 的所有节点。在消除节点的过程中,用正规式来标记弧,最后节点 x 和 y 之间的弧上的标记就是所求的正规式。消除节点的规则如图2-12所示。

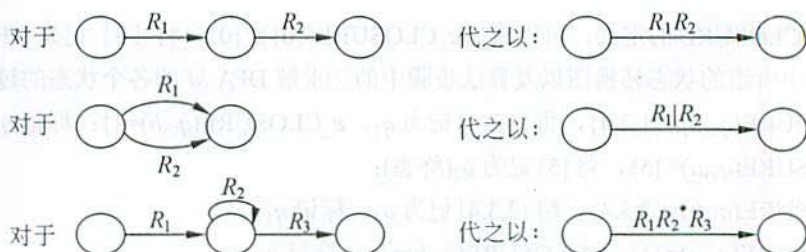


图 2-12 有限自动机到正规式的转换规则示意图

(2) 同样地, 对于 Σ 上的每个正规式 R , 可以构造一个 Σ 上的 NFA M , 使得 $L(M)=L(R)$ 。

① 对于正规式 R , 可用图 2-13 所示的拓广状态图表示。

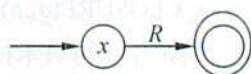


图 2-13 拓广状态图

② 通过对正规式 R 进行分裂并加入新的节点, 逐步把图转变成每条弧上的标记是 Σ 上的一个字符或 ε , 转换规则如图 2-14 所示。

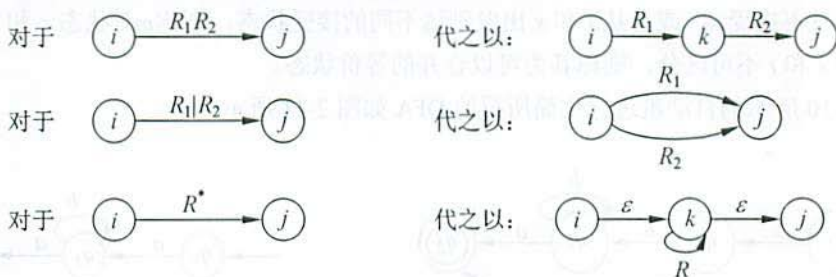


图 2-14 正规式到有限自动机的转换规则示意图

最后所得的图即为一个 NFA M , x 为初态节点, y 为终态节点。显然, $L(M)=L(R)$ 。

5) 词法分析器的构造

有了正规式和有限自动机的理论基础后, 就可以构造出编译程序的词法分析模块。构造词法分析器的一般步骤如下:

- ① 用正规式描述语言中的单词构成规则;
- ② 为每个正规式构造一个 NFA, 它识别正规式所表示的正规集;
- ③ 将构造出的 NFA 转换成等价的 DFA;
- ④ 对 DFA 进行最小化处理, 使其最简;
- ⑤ 从 DFA 构造词法分析器。

4. 语法分析

语法分析的任务是根据语言的语法规则, 分析单词串是否构成短语和句子, 即表达式、语

句和程序等基本语言结构,同时检查和处理程序中的语法错误。程序设计语言的绝大多数语法规则可以采用上下文无关文法进行描述。语法分析方法有多种,根据产生语法树的方向,可分为自底向上和自顶向下两类。

1) 上下文无关文法

上下文无关文法属于乔姆斯基(Chomsky)定义的2型文法,被广泛地用于表示各种程序设计语言的语法规则。对于上下文无关文法 $G[S] = (V_N, V_T, P, S)$, 其产生式的形式都是: $A \rightarrow \beta$, 其中 $A \in V_N$, $\beta \in (V_N \cup V_T)^*$ 。

若不加特别说明,下面我们用大写英文字母 A, B, C 等表示非终结符,小写英文字母 a, b, c 等表示终结符号, u, v, w 等表示终结符号串,小写希腊字母 $\alpha, \beta, \gamma, \delta$ 等表示终结符和非终结符混合的文法符号串。由于一个上下文无关文法的核心部分是其产生式集合,所以文法可以简写为其产生式集合的描述形式。

(1) 规范推导(最右推导):如果在推导的任何一步 $\alpha \Rightarrow \beta$ (其中 α, β 是句型),都是对 α 中的最右(最左)非终结符进行替换,则称这种推导为最右(最左)推导。最右推导常称为规范推导。

(2) 短语、直接短语和句柄:设 $\alpha\delta\beta$ 是文法 G 的一个句型,即 $S \xRightarrow{*} \alpha\delta\beta$, 且满足 $S \xRightarrow{*} \alpha A \beta$ 和 $A \xRightarrow{+} \delta$, 则称 δ 是句型 $\alpha\delta\beta$ 相对于非终结符 A 的短语。特别地,如果有 $A \Rightarrow \delta$, 则称 δ 是句型 $\alpha\delta\beta$ 相对于产生式 $A \rightarrow \delta$ 的直接短语。一个句型的最左直接短语称为该句型的句柄。

【例 2.4】 对于简单算术表达式,我们可以用下面的文法 $G[E]$ 进行描述。

$$G[E] = (\{E, T, F\}, \{+, *, (,), id\}, P, E)$$

$$P = \{E \rightarrow T \mid E + T, T \rightarrow F \mid T * F, F \rightarrow (E) \mid id\}$$

可以证明 $id+id*id$ 是该文法的句子。下面我们用最右推导的方式从文法的开始符号出发推导出该句子。为了表示推导过程中相同符号的不同出现,我们给符号加一个下标。

$$\begin{aligned} E &\Rightarrow E_1 + T_1 \Rightarrow E_1 + T_2 * F_1 \Rightarrow E_1 + T_2 * id_3 \Rightarrow E_1 + F_2 * id_3 \Rightarrow E_1 + id_2 * id_3 \\ &\Rightarrow T_3 + id_2 * id_3 \Rightarrow F_2 + id_2 * id_3 \Rightarrow id_1 + id_2 * id_3 \end{aligned}$$

该推导过程可以用树型结构进行描述,如图 2-15 所示。

由于 $E \Rightarrow F_3 + id_2 * id_3$, 且 $F_3 \Rightarrow id_1$, 所以 id_1 是句型 $id_1 + id_2 * id_3$ 相对于非终结符 F 的短语,也是相对于产生式 $F \rightarrow id$ 的直接短语。

由于 $E \Rightarrow E_1 + T_2 * id_3$, 且 $T_2 \Rightarrow id_2$, 所以 id_2 是句型 $E_1 + id_2 * id_3$ 的相对于非终结符 T 的短语。

由于 $E \Rightarrow E_1 + T_1$, 且 $T_1 \Rightarrow id_2 * id_3$, 所以 $id_2 * id_3$ 是句型 $E_1 + id_2 * id_3$ 的相对于非终结符 T 的短语。

由于 $E \Rightarrow id_1 * id_2 + id_3$, 所以 $id_1 * id_2 + id_3$ 是句型 $id_1 * id_2 + id_3$ 的相对于非终结符 E 的短语。

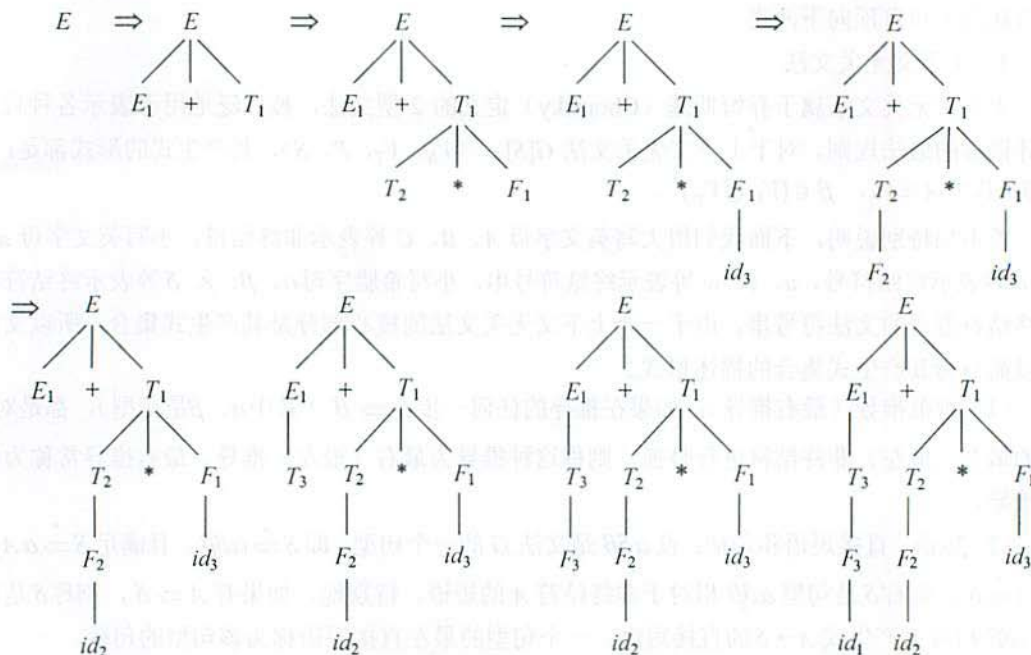


图 2-15 推导过程示意图

实际上 id_1 , id_2 , id_3 , $id_1 * id_2$ 和 $id_1 * id_2 + id_3$ 都是句型 $id_1 * id_2 + id_3$ 的短语, 而且 id_1 、 id_2 、 id_3 均是直接短语, 其中 id_1 是最左直接短语, 即句柄。

2) 自顶向下分析方法

自顶向下分析法的基本思想是: 对于给定的输入串 ω , 从文法的开始符号 S 出发进行最左推导, 直到得到一个合法的句子或者发现一个非法结构。在推导的过程中试图用一切可能的方法, 自上而下、从左到右地为输入串 ω 建立语法树。整个分析过程是一个试探的过程, 是反复使用不同产生式谋求与输入序列匹配的过程。若输入串是给定文法的句子, 则必能成功, 反之必然出错。

文法中存在下述产生式时, 自顶向下分析过程中会出现下面的问题:

- 若文法中存在形如 $A \rightarrow \alpha\beta | \alpha\beta$ 的产生式, 即 A 产生式中有多于一个候选项的前缀是相同的 (称为公共左因子, 简称左因子), 则可能导致分析过程中的回溯处理;
- 若文法中存在形如 $A \rightarrow A\alpha$ 的产生式, 由于采取了最左推导, 可能会造成分析过程陷入

死循环的情况,产生式的这种形式被称为左递归。

因此,需要对文法进行改造,消除其中的左递归,以避免分析陷入死循环;提取左因子,以避免回溯。

(1) 消除文法的左递归

① 消除直接左递归。一般而言,假定非终结符号 A 的产生式如下:

$$A \rightarrow A\alpha | \beta$$

其中, α 不等于 ε , β 不以 A 开头,那么就对 A 的产生式进行如下的改造:

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A' | \varepsilon$$

由于 $A \Rightarrow A\alpha^* \Rightarrow \beta\alpha^*$, 可知改造前后关于 A 的产生式是等价的。将上述结果推广到更一般的情况,得到消除文法直接左递归的算法如下:

- 输入: 文法 G 中所有的 A 产生式;
- 输出: 等价的不含直接左递归的文法 G' 。

方法: 首先, 将非终结符 A 的产生式整理为如下形式:

$$A \rightarrow A\alpha_1 | A\alpha_2 | \dots | A\alpha_m | \beta_1 | \beta_2 | \dots | \beta_n$$

其中, $\alpha_i (1 \leq i \leq m)$ 都不等于 ε , $\beta_j (1 \leq j \leq n)$ 都不以 A 开头, 然后用下述产生式代替 A 的产生式。

$$A \rightarrow \beta_1 A' | \beta_2 A' | \dots | \beta_n A'$$

$$A' \rightarrow \alpha_1 A' | \alpha_2 A' | \dots | \alpha_m A' | \varepsilon$$

② 消除文法中一切左递归的算法。消除文法中所有左递归的方法是: 对产生式的右部进行代换, 将所有的非直接左递归产生式改造为直接的左递归产生式, 然后消除文法中的直接左递归产生式。

【例 2.5】对于文法

$$G[E] = (\{E, T, F\}, \{+, *, (,), id\}, P, E)$$

$$P = \{E \rightarrow TE + T, T \rightarrow FT * F, F \rightarrow (E) | id\}$$

消除左递归后的文法为:

$$G'[E] = (\{E, E', T, T', F\}, \{+, *, (,), id\}, P', E)$$

$$P' = \{E \rightarrow TE', E' \rightarrow +TE' | \varepsilon, T \rightarrow FT', T' \rightarrow *FT' | \varepsilon, F \rightarrow (E) | id\}$$

(2) 提取公共左因子。假定关于 A 的产生式为: $A \rightarrow \alpha\beta_1 | \alpha\beta_2 | \dots | \alpha\beta_n$, 那么称 α 为 A 的候选式的公共左因子 (简称左因子)。如果 A 的产生式有左因子, 推导过程中会出现无法确定用 A 产生式的哪个候选式替换 A 的情况, 这时可以重写 A 产生式来推迟这种决定, 直到看见足够的输入, 能正确做出选择时为止, 因此可将 A 的产生式改写如下:

$$A \rightarrow \alpha A'$$

$$A' \rightarrow \beta_1 | \beta_2 | \cdots | \beta_n$$

经反复提取,就能使每个非终结符(包括新引进的)的任意两个候选式不含有公共前缀。

(3) LL(1)文法。我们首先定义两个集合: FIRST(a)和 FOLLOW(A)。

① 文法符号序列 α 的 FIRST 集合定义如下:

$\text{FIRST}(\alpha) = \{a | \alpha \Rightarrow a \cdots, \text{且 } a \in V_T\}$, 若 $\alpha \Rightarrow \varepsilon$, 约定 $\varepsilon \in \text{FIRST}(\alpha)$ 。

求解一个文法符号的 FIRST 集合时,先将每一个文法符号 X ($X \in V_N \cup V_T$) 的 FIRST (X) 置空,然后应用以下规则:

- 若 $X \in V_T$, 则将 $\text{FIRST}(X) = \{X\}$;
- 若 $X \in V_N$, 则考查其产生式:
 - ◆ 若有 $X \rightarrow \varepsilon$, 则将 ε 加入 $\text{FIRST}(X)$;
 - ◆ 若有 $X \rightarrow a \cdots$, 且 $a \in V_T$, 则将 a 加入 $\text{FIRST}(X)$;
 - ◆ 若有 $X \rightarrow Y_1 Y_2 \cdots Y_k$, 且 $Y_1 \in V_N$, 则将 $\text{FIRST}(Y_1)$ 中除 ε 之外的所有元素加入 $\text{FIRST}(X)$; 若 $Y_1 Y_2 \cdots Y_{i-1} \Rightarrow \varepsilon$, 则将 $\text{FIRST}(Y_i)$ 中除 ε 之外的所有元素加入 $\text{FIRST}(X)$; 特别地, 若 $\varepsilon \in \text{FIRST}(Y_j)$, $j=1, 2, \cdots, k$, 则将 ε 加入 $\text{FIRST}(X)$;
- 反复执行①和②两步,直到每个文法符号的 FIRST 集合不再扩大时为止。

任意的文法符号序列 $X_1 X_2 \cdots X_k$ 的 FIRST 集合的计算方法可从上面的计算过程中推出。

② 非终结符 A 的 FOLLOW 集合定义如下:

$\text{FOLLOW}(A) = \{a | S \Rightarrow \cdots A a \cdots, \text{且 } a \in V_T\}$, 若 A 是某个句型的最右符号, 则约定 $\#$ 属于 $\text{FOLLOW}(A)$ 。

求解一个文法中非终结符 A 的 FOLLOW 集合时,先将 $\text{FOLLOW}(A)$ 置空,然后应用以下规则:

- 若 A 是文法的开始符号, 则将 $\#$ 加入 $\text{FOLLOW}(A)$, $\#$ 是输入结束标记;
- 若有产生式 $A \rightarrow \alpha B \beta$, 则将 $\text{FIRST}(\beta)$ 中除 ε 之外的所有元素加入 $\text{FOLLOW}(B)$ 中;
- 若有产生式 $A \rightarrow \alpha B$, 或 $A \rightarrow \alpha B \beta$ 且 $\varepsilon \in \text{FIRST}(\beta)$, 则将 $\text{FOLLOW}(A)$ 中的全体元素加入 $\text{FOLLOW}(B)$ 中。

非形式地讲,文法符号序列 α 的 FIRST 集合,就是从 α 出发可以推导出的所有以终结符号开头的序列中的开头终结符号。而一个非终结符的 FOLLOW 集合,就是从文法开始符号可以推导出的所有含 A 句型中紧跟在 A 之后的终结符号。

【例 2.6】对文法

$$G[E] = (\{E, E', T, T', F\}, \{+, *, (,), id\}, P, E)$$

$$P = \{E \rightarrow TE', E' \rightarrow +TE' | \varepsilon, T \rightarrow FT', T' \rightarrow *FT' | \varepsilon, F \rightarrow (E) | id\}$$

对于终结符号 a , 显然 $\text{FIRST}(a) = \{a\}$, 所以下面只列出文法中非终结符号的 FIRST 和

FOLLOW 集合的计算结果。

$\text{FIRST}(E) = \text{FIRST}(T) = \text{FIRST}(F) = \{ (, \text{id})$

$\text{FIRST}(T) = \{ *, \varepsilon \}$

$\text{FIRST}(E') = \{ +, \varepsilon \}$

$\text{FOLLOW}(E) = \text{FOLLOW}(E') = \{ \#, \} \}$

$\text{FOLLOW}(T) = \text{FOLLOW}(T') = \{ \#,), + \}$

$\text{FOLLOW}(F) = \{ \#,), +, * \}$

一个文法 G 是 $LL(1)$ 的, 当且仅当 G 的任何两个产生式 $A \rightarrow \alpha | \beta$ 满足下面的条件:

- 对任何终结符 a , α 和 β 不能同时推导出以 a 开始的文法符号序列;
- α 和 β 最多有一个可以推导出 ε ;
- 若 $\beta \Rightarrow \varepsilon$, 则 α 不能推导出以 $\text{FOLLOW}(A)$ 中的终结符开始的任何文法符号序列。

(4) 递归下降分析法。递归下降分析法要求文法是 $LL(1)$ 文法, 它直接以程序的方式模拟产生式产生语言的过程, 其基本思想是: 为每一个非终结符构造一个子程序, 每个子程序的过程体按该产生式候选项分情况展开, 遇到终结符即进行匹配, 而遇到非终结符则调用相应的子程序。该分析法从调用文法开始符号的子程序开始, 直到所有非终结符都展开为终结符并得到匹配为止。若分析过程可以达到这一步, 则表明分析成功, 否则表明输入串中有语法错误。对于规模较小的语言, 递归下降分析法是一种有效的方法, 其优点是简单且易于构造, 缺点是程序与文法直接相关, 对文法的任何改变都需要在程序中进行相应的修改。

(5) 预测分析法。预测分析法是另一种自顶向下的分析方法, 其基本模型如图 2-16 所示。

一个 $LL(1)$ 文法的预测分析表可以用一个二维数组 M 表示, 其元素 $M[A, a] (A \in V_N, a \in V_T \cup \#)$ 存放关于 A 的产生式, 表明当遇到输入符号为 a 且用 A 进行推导时, 所应采用的产生式; 若 $M[A, a]$ 为 error, 则表明推导时遇到了不该出现的符号, 应进行出错处理。构造一个文法的预测分析表的过程如下:

- ① 对文法的每个产生式 $A \rightarrow \alpha$, 执行②和③;
- ② 对 $\text{FIRST}(\alpha)$ 的每个终结符 a , 加入 $A \rightarrow \alpha$ 到 $M[A, a]$;
- ③ 若 $\varepsilon \in \text{FIRST}(\alpha)$, 则对 $\text{FOLLOW}(A)$ 的每个非终结符 b (包括 $\#$), 加入 $A \rightarrow \alpha$ 到 $M[A, b]$ 中;
- ④ M 中其他没有定义的条目置为 error。

预测分析法的工作过程是: 初始时, 将 “ $\#$ ” 和文法的开始符号依次压入栈中; 在分析过

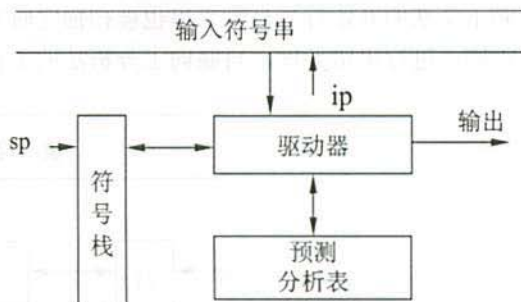


图 2-16 预测分析模型示意图

程中, 根据输入串中的当前输入符号 a 和当前的栈顶符号 X 进行处理:

若 $X=a=\#$, 则分析成功, 若 $X=\#$ 且 $a \neq \#$, 则出错;

若 $X \in V_T$ 且 $X=a$, 则 X 退栈, 并读入下一个符号 a , 若 $X \in V_T$ 且 $X \neq a$, 则出错;

若 $X \in V_N$ 且 $M[X, a] = 'A \rightarrow \alpha'$, 则 X 退栈, α 中的符号从右到左依次进栈(ε 无需进栈);
若 $M[X, a] = 'error'$, 则调用出错程序进行处理。

根据文法 $G[E] = \{E \rightarrow TE', E' \rightarrow +TE' | \varepsilon, T \rightarrow FT', T' \rightarrow *FT' | \varepsilon, F \rightarrow (E) | id\}$ 构造的预测分析表为:

	id	+	*	()	#
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		

3) 自底向上分析方法

自底向上分析方法也称移进-归约分析法, 它的基本思想是对输入序列 ω 自左向右进行扫描, 并将输入符号逐个移进一个栈中, 边移进边分析, 一旦栈顶符号串形成某个句型的可归约串时, 就用某个产生式的左部非终结符来替代, 这称为一步归约。重复这一过程, 直至栈中只剩下文法的开始符号且输入串也被扫描完时为止, 确认输入串 ω 是文法的句子, 表明分析成功; 否则, 进行出错处理。自底向上分析法的工作模型如图 2-17 所示。

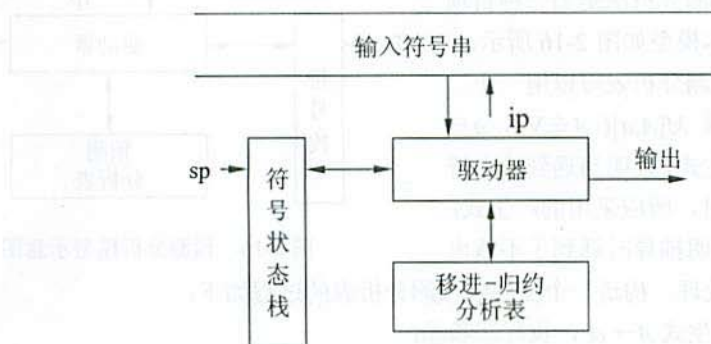


图 2-17 移进-归约分析模型

移进-归约分析法的数学模型是下推自动机。若模型中采用算符优先分析表, 用“最左素短语”来刻画“可归约串”, 则相应的分析器称为算符优先分析器; 若采用 LR 分析表, 用“句

柄”来刻画“可归约串”，则相应的分析器称为 LR 分析器。下面我们简单介绍 LR 分析器的原理。

LR 分析法是一种规范归约分析法。规范归约是规范推导（最右推导）的逆过程，下面举例说明规范归约的过程。

【例 2.7】 设文法 $G[S]=\{S \rightarrow aAcBe, A \rightarrow b, A \rightarrow Ab, B \rightarrow d\}$ ，下面对输入串 $\#abbcd\#$ 进行分析。先设一个符号栈，并把句子左括号“ $\#$ ”放入栈底，其分析过程如下：

步骤	符号栈	输入符号串	动作
①	#	abbcd#	移进
②	#a	bbcd#	移进
③	#ab	bcd#	归约 ($A \rightarrow b$)
④	#aA	bcd#	移进
⑤	#aAb	cde#	归约 ($A \rightarrow Ab$)
⑥	#aA	cde#	移进
⑦	#aAc	de#	移进
⑧	#aAcd	e#	归约 ($B \rightarrow d$)
⑨	#aAcB	e#	移进
⑩	#aAcBe	#	归约 ($S \rightarrow aAcBe$)
⑪	#S	#	接受

说明：在第③步中栈顶符号串 b 是句型 $abbcd$ 的句柄，用产生式 $A \rightarrow b$ 进行归约；第⑤步中 Ab 是句型 $aAbcd$ 的句柄，用相应产生式 $A \rightarrow Ab$ 进行归约；第⑧步和第⑩步是同样的道理。上述分析过程也可看成是自底向上构造语法树的过程。

LR 分析法根据当前分析栈中的符号串（通常以状态表示）和向右顺序查看输入串的 k 个 ($k \geq 0$) 符号，就可唯一确定分析器的动作是移进还是归约，以及用哪条产生式进行归约，因而也就能唯一地确定句柄。当 $k=1$ 时，已能满足当前绝大多数高级语言编译程序的需求。常用的 LR 分析器有 LR(0), SLR(1), LALR(1) 和 LR(1) 4 种。

一个 LR 分析器由 3 个部分组成。

(1) 驱动器：或称驱动程序。对所有 LR 分析器，驱动程序都是相同的。

(2) 分析表：不同的文法具有不同的分析表。同一文法采用不同的 LR 分析器时，分析表也不同。分析表又可分为动作表 (ACTION) 和状态转换表 (GOTO) 两个部分，它们都可用二维数组表示。

(3) 分析栈：包括文法符号栈和相应的状态栈。

分析器的动作由栈顶状态和当前输入符号决定 (LR(0) 分析器不需向前查看输入符号)，LR 分析器的模型如图 2-18 所示。

其中 SP 为栈顶指针, S_i 为状态, X_i 为文法符号。ACTION[S_i, a]= S_j 规定了栈顶状态为 S_i 且遇到输入符号 a 时应执行的动作。状态转换表 GOTO[S_i, X]= S_j 表示当状态栈顶为 S_i 且文法符号栈顶为 X 时应转向状态 S_j 。

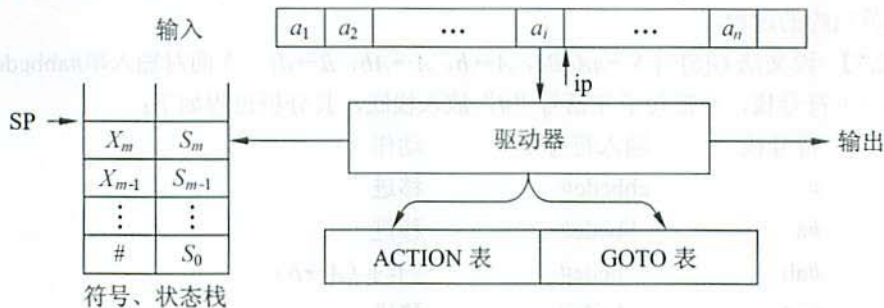


图 2-18 LR 分析器的模型示意图

LR 分析器的工作过程以格局的变化来反映。格局的形式为：(栈，剩余输入，动作)。分析是从某个初始格局开始的，经过一系列的格局变化，最终达到接受格局，表明分析成功；或者达到出错格局。表明发现一个语法错误。因此，开始格局的剩余输入应该是全部的输入序列，而接受格局中的剩余输入应该为空，任何其他格局或者出错格局中的剩余输入应该是全部输入序列的一个后缀。

在 LR 分析过程中，改变格局的动作有以下 4 种：

- (1) 移进 (shift)：当 ACTION[S_i, a]= S_j 时，把 a 移进文法符号栈并转向状态 S_j 。
- (2) 归约 (reduce)：当在文法符号栈顶形成句柄 β 时，把 β 归约为相应产生式 $A \rightarrow \beta$ 的非终结符 A 。若 β 的长度为 r (即 $|\beta|=r$)，则弹出文法符号栈顶的 r 个符号，然后将 A 压入文法符号栈中。
- (3) 接受 (accept)：当文法符号栈中只剩下文法的开始符号 S ，并且输入符号串已经结束时 (当前输入符是“#”)，分析成功。
- (4) 报错 (error)：当输入串中出现不该有的文法符号时，就报错。

LR 分析器的核心部分是分析表的构造，这里不再详述。

5. 语法制导翻译和中间代码生成

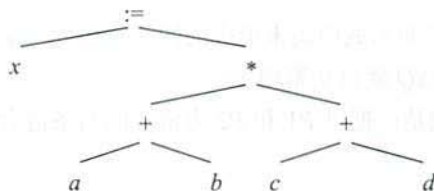
程序语言的语义分为静态语义和动态语义。描述程序语义的形式化方法主要有属性文法、公理语义、操作语义和指称语义等，其中属性文法是对上下文无关文法的扩充。目前应用最广的静态语义分析方法是语法制导翻译，其基本思想是将语言结构的语义以属性的形式赋予代表此结构的文法符号，而属性的计算以语义规则的形式赋予文法的产生式。在语法分析的推导或归约的步骤中，通过语义规则实现对属性的计算，以达到对语义的处理。本节主要讨论以属性为基础，程序的静态语义分析和中间代码的生成。

1) 中间代码

从原理上讲,源程序在进行了语义分析之后就可以直接生成目标代码,但由于源程序与目标代码的逻辑结构往往差别很大,特别是考虑到具体机器指令系统的特点,要使翻译一次到位很困难,而且用语法制导方式机械生成的目标代码往往是繁琐和低效的,因此有必要设计一种中间代码,将源程序首先翻译成中间代码表示形式,以利于进行与机器无关的优化处理。由于中间代码实际上也起着编译器前端和后端分水岭的作用,所以使用中间代码也有助于提高编译程序的可移植性。常用的中间代码表示形式有后缀式、三元式、四元式和树形等。

(1) 后缀式(逆波兰)。逆波兰式是波兰逻辑学家卢卡西维奇(Lukasiewicz)发明的一种表示表达式的方法。他把运算符写在运算对象的后面,例如,把 $a+b$ 写成 $ab+$, 所以也称为后缀式。这种表示法的优点是根据运算对象和算符的出现次序进行计算,不需要使用括号。用栈结构实现后缀式的计算是很方便的,一般的方法是:自左至右扫描后缀式,遇到运算对象时就将其压入栈中,遇到 k 元运算符时就从栈中弹出 k 项进行运算,并将结果压入栈中,当表达式被扫描完时,栈顶元素就是表达式的运算结果。对于表达式 $x := (a+b) * (c+d)$, 其后缀式为 “ $xab+cd+*:=$ ”。

(2) 树形表示。我们也可以用树形数据结构来表示一个表达式或语句。表达式 $x := (a+b) * (c+d)$ 的树形表示为:



(3) 三元式表示。三元式是由运算符 OP、第一运算对象 ARG1 和第二运算对象 ARG2 组成的。三元式出现的先后顺序和表达式各部分的计值顺序是一致的。表达式 $x := (a+b) * (c+d)$ 的三元式表示为:

- ① (+, a, b)
- ② (+, c, d)
- ③ (*, ①, ②)
- ④ (:=, ③, x)

(4) 四元式表示。四元式是一种普遍采用的中间代码形式,其组成成分为运算符 OP、第一运算对象 ARG1、第二运算对象 ARG2 和运算结果 RESULT。其中,运算对象和运算结果有时指用户自定义的变量,有时指编译程序引入的临时变量,RESULT 总是一个新引进的临时变量,用来存放运算结果。表达式 $x := (a+b) * (c+d)$ 的四元式表示为:

- ① (+, a, b, t1)

② $(+, c, d, t_2)$

③ $(*, t_1, t_2, t_3)$

④ $(:=, t_3, _, x)$

2) 常见语法单位的翻译

常见的语法单位主要有：算术表达式、布尔表达式、赋值语句、控制语句（分支、循环）等。对于不同的结构，有不同的处理方法，但翻译程序的构造原理是相似的。

对于各种语法单位的语法制导翻译，一般是在相应的语法规则中加入适当的语义处理，下面先说明翻译过程中要使用的语义变量和语义过程。

- $\text{Entry}(\text{id})$ 的功能是：在符号表中查找标识符 id 以获取它在表中的位置（入口）。
- $S.\text{code}$ ：语句 S 被翻译后形成的代码序列。
- $E.\text{place}$ ：与非终结符 E 相联系的语义变量，表示存放 E 值的变量名在符号表的入口或整数码（若此变量是临时变量）。
- $E.\text{tc}$ ：当表达式 E 的值为真时控制流转向的语句标号（四元式的地址编号）。
- $E.\text{fc}$ ：当表达式 E 的值为假时控制流转向的语句标号（四元式的地址编号）。
- $\text{GEN}(\text{OP}, \text{ARG1}, \text{ARG2}, \text{RESULT})$ 的功能是：把四元式 $(\text{OP}, \text{ARG1}, \text{ARG2}, \text{RESULT})$ 填进四元式表中。
- NXQ ：表示下一个将要形成但尚未形成的四元式地址（编号）。 NXQ 的初值为 1，每执行一次 $\text{GEN}()$ ， NXQ 就自动累增 1。
- $\text{Merg}(P1, P2)$ 的功能是：把以 $P1$ 和 $P2$ 为链首的两条链合并为一条链，并返回合并后的链首指针。
- $\text{Backpatch}(p, t)$ 的功能是：把 p 所链接的每条四元式的第 4 项都以 t 作为值进行填充。
- Newtemp ：生成一个新的临时存储单元。

拉链与回填的基本思想是当一个四元式中存在尚不确定的转向地址时，将所有转向同一地址的四元式链接成一个链表，一旦转向地址被确定，则沿此链向所有的四元式回填该地址。

下面简单说明程序语言中常见结构的语法制导翻译方法。

(1) 赋值语句及简单算术表达式的翻译

下面的产生式描述了由简单变量、算术加、乘、取负运算和圆括号构成的简单算术表达式，以及将一个算术表达式赋值给一个简单变量的赋值语句的形式规则。

$A \rightarrow \text{id} := E$

$E \rightarrow E + E \mid E * E \mid -E \mid (E) \mid \text{id}$

为该文法编写的语义规则如下：

$A \rightarrow \text{id} := E \quad \{ \text{GEN}(\text{:=}, E.\text{place}, _, \text{Entry}(\text{id})) \}$

$E \rightarrow E^{(1)} + E^{(2)} \quad \{ E.\text{place} := \text{Newtemp}; \text{GEN}(+, E^{(1)}.\text{place}, E^{(2)}.\text{place}, E.\text{place}) \}$

$E \rightarrow E^{(1)} * E^{(2)} \quad \{E.place := \text{Newtemp}; \text{GEN}(*, E^{(1)}.place, E^{(2)}.place, E.place)\}$
 $E \rightarrow -E^{(1)} \quad \{E.place := \text{Newtemp}; \text{GEN}(@, E^{(1)}.place, _, E.place)\}$
 $E \rightarrow (E^{(1)}) \quad \{E.place := E^{(1)}.place\}$
 $E \rightarrow id \quad \{E.place := \text{Entry}(id)\}$

(2) 布尔表达式的翻译

布尔表达式常用于表示控制结构中的条件,其计算过程可采用直接计算和短路计算两种形式。直接计算是指布尔表达式中的每个因子都进行运算,而短路计算指只要表达式的值能够确定下来,就停止计算,并非表达式中所有的运算对象都进行运算。下面以短路计算方式为例介绍布尔表达式作为控制条件时的翻译方法。

布尔表达式的形式定义(文法)为:

$E \rightarrow E \text{ and } E \mid E \text{ or } E \mid \text{not } E \mid (E) \mid id \mid id \text{ relop } id$

其中, relop 代表关系运算符(<、>、≤、≥、=、≠),运算符的优先级和结合律遵照通常的习惯。 $E \rightarrow E^{(1)} \text{ and } E^{(2)}$, $E \rightarrow E^{(1)} \text{ or } E^{(2)}$ 的代码结构如图 2-19(a)和(b)所示。

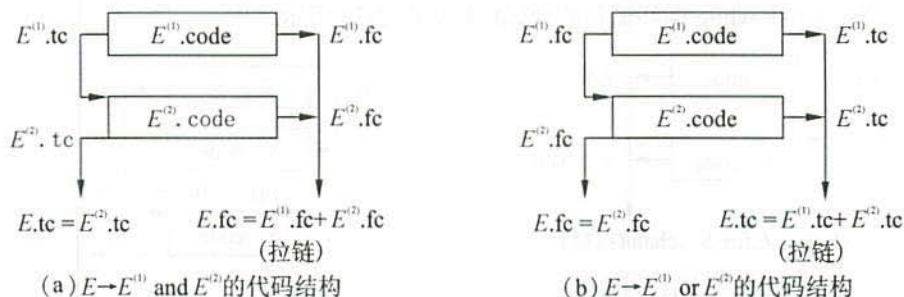


图 2-19 and 与 or 的代码结构示意图

改写文法并编写语义子程序如下所示。

- ① $E \rightarrow E^{(1)} \text{ and } E^{(2)}$, 为记住 $E^{(2)}$ 的第一条四元式的地址, 需改写为:

$E^{\wedge} \rightarrow E^{(1)} \text{ and} \quad \{ \text{Backpatch}(E^{(1)}.tc, \text{NXQ}); \quad // \text{回填真出口}$
 $\quad \quad \quad E^{\wedge}.fc := E^{(1)}.fc; \quad // \text{传假出口}$
 $\quad \quad \quad \}$
 $E \rightarrow E^{\wedge} E^{(2)} \quad \{ E.tc := E^{(2)}.tc; \quad // \text{传真出口}$
 $\quad \quad \quad E.fc := \text{Merg}(E^{\wedge}.fc, E^{(2)}.fc) \quad // \text{合并假出口}$
 $\quad \quad \quad \}$

- ② $E \rightarrow E^{(1)} \text{ or } E^{(2)}$, 为记住 $E^{(2)}$ 的第一条四元式的地址, 需改写为:

$E^{\vee} \rightarrow E^{(1)} \text{ or} \quad \{ \text{Backpatch}(E^{(1)}.fc, \text{NXQ}); \quad // \text{回填假出口}$
 $\quad \quad \quad E^{\vee}.tc := E^{(1)}.tc; \quad // \text{传真出口}$


```

    }
    E → E∨ E(2)      { E.tc := Merg(E∨.tc, E(2).tc); //合并真出口
                        E.fc := E(2).fc //传假出口
    }
    E → not E(1)      { E.tc := E(1).fc; E.fc := E(1).tc } //交换真、假出口
    E → (E(1))         { E.tc := E(1).tc; E.fc := E(1).fc } //传真、假出口
    E → id            { E.tc := NXQ; E.fc := NXQ+1;
                      GEN(jnz, Entry(id), -, 0); GEN(j, -, -, 0); }
    E → id(1) relop id(2) { E.tc := NXQ; E.fc := NXQ+1;
                          GEN(jrelop, Entry(id(1)), Entry(id(2)), 0);
                          GEN(j, -, -, 0); }

```

(3) 常见语句的翻译

这里只列举条件语句、while 循环语句和赋值语句的翻译方法，其形式定义为：

$S \rightarrow A | \text{if } E \text{ then } S^{(1)} \mid \text{if } E \text{ then } S^{(1)} \text{ else } S^{(2)} \mid \text{while } E \text{ do } S^{(1)}$

用语义变量 $S.\text{chain}$ 记录语句结束后的转向地址。回填工作将在处理 S 的外层环境的某个时刻完成。条件语句和 while 循环语句的代码结构如图 2-20 所示。

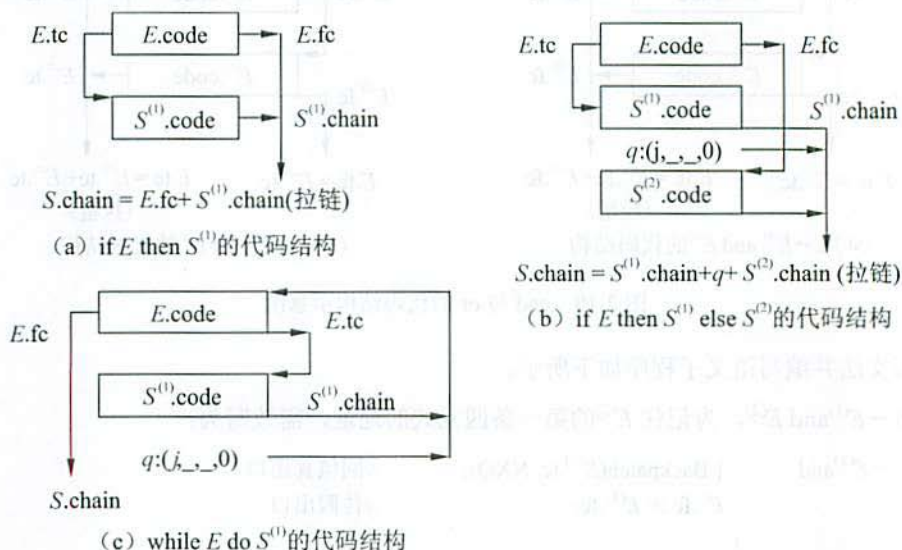


图 2-20 常见控制语句的代码结构

```

C → if E then      { Backpatch(E.tc, NXQ); //填真出口
                    C.chain := E.fc; //传假出口 }
S → CS(1)          { S.chain := Merg(C.chain, S(1).chain); } //拉链
Tp → CS(1) else   { q := NXQ; GEN(j, -, -, 0);

```

```

    Tp.chain:= Merg(S(1).chain,q);           //拉链
    Backpatch(C.chain, NXQ);                 //填假出口
  }
  S→TpS(1)    { S.chain:= Merg(Tp.chain, S(2).chain); } //拉链
  W→while    { W.quad:= NXQ; }                     //记录 while 语句的首地址
  Wd→W E do  { Backpatch(E.tc, NXQ);               //填真出口
                Wd.chain := E.fc;                  //传假出口
                Wd.quad:= W.quad;                  //传首地址 }
  S→WdS(1)    { Backpatch(S(1).chain, Wd.quad);      //S(1)执行完后转向 E
                GEN(j, -, -, Wd.quad);
                S.chain:= Wd.chain;                //S 的出口
  }
  S→A        { S.chain:=0 }                        //赋值语句不含其他非常规出口 (无出口链)

```

【例 2.8】 设 NXQ 的初值为 1，每产生一条四元式，NXQ 的值就累增 1，将语句 if $a < b$ then while $a < b$ do $a := a + b$ 翻译成四元式的主要步骤如下：

步骤	产生式	语义处理及结果	四元式
①	$E \rightarrow a < b$	$E.tc := (1)$ $E.fc := (2)$ 产生两条四元式	1: $(j <, a, b, 0)$ 2: $(j, -, -, 0)$
②	$C \rightarrow \text{if } E \text{ then}$	用 NXQ 值(=3)回填 E.tc 所记录的四元式(1), $C.chain := (2)$	1: $(j <, a, b, 3)$
③	$W \rightarrow \text{while}$	$W.quad := 3$	
④	$E \rightarrow a < b$	$E.tc := (3)$ $E.fc := (4)$ 产生两条四元式	3: $(j <, a, b, 0)$ 4: $(j, -, -, 0)$
⑤	$W^d \rightarrow W E \text{ do}$	用 NXQ 值(=5)回填 E.tc 所记录的四元式(3), $W^d.chain := (4)$ $W^d.quad := 3$	3: $(j <, a, b, 5)$
⑥	$E \rightarrow a + b$	生成临时存储单元 t1, 产生一条四元式	5: $(+, a, b, t1)$
⑦	$A \rightarrow id := E$	产生一条四元式	6: $(:=, t1, -, a)$
⑧	$S \rightarrow A$	$S.chain := (0)$	
⑨	$S \rightarrow W^d S^{(1)}$	用 $W^d.quad$ 值(=3)回填 $S^{(1)}.chain$ 所记录的四元式(0), 产生一条四元式 $S.chain := (4)$	7: $(j, -, -, 3)$
⑩	$S \rightarrow CS^{(1)}$	合并 C.chain 和 $S^{(1)}.chain$ 两条链, 并将合并后的链赋值给 $S.chain := (4, 2)$	

最后产生的四元式序列如下所示，其中 2、4 两条四元式有待于回填：

1: $(j <, a, b, 3)$

- 2: (j,-,-,0)
- 3: (j<,a,b,5)
- 4: (j,-,-,0)
- 5: (+,a,b,t1)
- 6: (:=,t1,-,a)
- 7: (j,-,-,3)
- 8:

(4) 动态存储分配和过程调用的翻译

过程(函数)说明和过程(函数)调用是程序中一种常见的语法结构,绝大多数语言都含有这方面的内容。过程说明和调用语句的翻译,有赖于形式参数与实际参数结合的方式以及数据空间的分配方式。

由于各种语言的不同特点,在目标程序运行时,对存储空间的分配和组织有不同的要求,在编译阶段应产生相应的目标来满足不同的要求。需要分配存储空间的对象有基本数据类型(如整型、实型和布尔型等)、结构化数据类型(如数组和记录等)和连接数据(如返回地址、参数等)。分配的依据是名字的作用域和生存期的定义规则。分配的策略按语言的特点有静态和动态两大类。

如果在编译时就能确定目标程序运行时所需的全部数据空间的大小,则在编译时就安排好目标程序运行时的全部数据空间,并确定每个数据对象的存储位置。这种分配策略称为静态存储分配。FORTRAN 语言可以完全采用静态存储分配策略。

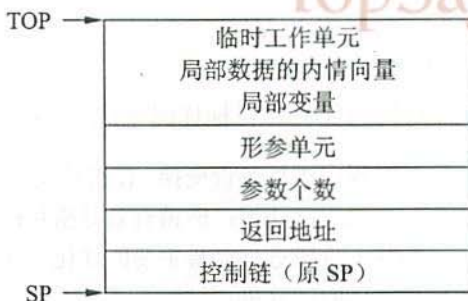
如果一个程序语言允许递归过程和可变数据结构,那么,就需采用动态存储分配技术。动态存储分配策略的实现有栈分配方式和堆分配方式两种。在栈式动态存储分配技术中,将程序的数据空间设计为一个栈,每当调用一个过程时,它所需的数据空间就分配在栈顶;每当过程工作结束时,就释放这部分空间。栈式存储分配适用于 PASCAL、C、ALGOL 等语言。如果一个程序语言为用户提供自由地申请数据空间和退还数据空间的机制(如 C++ 中的 new/delete),或者不仅有过程而且有进程的程结构,也就是说空间的使用未必服从“先申请后释放”的原则,那么栈式的动态存储分配方式就不适用了,这种情况下通常使用堆分配技术。下面仅就一个简单的栈式分配,说明过程调用和过程说明的翻译。

考虑一种简单的程序语言结构:没有分程序结构,过程定义不嵌套,但允许过程递归调用。

通常称过程(函数)的一次运行为一个活动。活动是一个动态的概念,除了设计为永不停机的过程(例如操作系统等),或者是因为设计错误而出现死循环的情况外,任何过程的活动均有有限的生存期。每个活动在运行时的环境就称为它的活动记录。一般情况下,一个活动记录的内容如图 2-21(a)所示,活动记录内容的一种安排方式如图 2-21(b)所示。其中,SP 指向当前活动记录在控制栈中的起始位置, TOP 指向栈顶位置。

1. 参数与返回地址
2. 控制链(可选)
3. 访问链(可选)
4. 保存的机器状态
5. 本地(局部)数据
6. 临时变量

(a) 活动记录的内容



(b) 活动记录的一种安排方式

图 2-21 过程的活动记录

设 SP 总是指向现行过程活动记录的起点, TOP 始终指向栈顶单元, 则过程调用语句 call P (T_1, T_2, \dots, T_n) 的四元式序列是:

```
para  T1
para  T2
...
para  Tn
call  P, n
```

① 在运行时, para 和 call 应产生传递参数和进行调用的代码:

$(i+3)[TOP] := T_i$ (传递参数值) 或 $(i+3)[TOP] := \text{addr}(T_i)$ (传递参数地址)

$1[TOP] := SP$ (保护现行 SP)

$3[TOP] := n$ (传递参数个数)

JSR P (转子程序指令, 即控制转向过程 P 的第一条指令)

② 转入过程 P 后, 就进入过程说明的代码:

$SP := TOP + 1$ (定义新的 SP)

$1[SP] := \text{返回地址}$ (保留返回地址)

$TOP := TOP + L$ (定义新的 TOP, L 是过程 P 的活动记录所需要的单元数, 编译时可确定)

③ 过程说明中数组说明的代码, 调整 TOP 的代码:

④ 过程体内执行语句的代码 (用 SP 变址方式访问对象):

⑤ 退出过程语句时 return(E)的代码序列为:

$R := E.\text{place}$ (把 E 的值放到特定的寄存器 R 中, 调用段将从中获得被调用过程的结果值):

$TOP := SP - 1$ (恢复 TOP)

$SP := [SP]$ (恢复 SP)

PC:=2[TOP](取返回地址)

JMP PC

6. 中间代码优化和目标代码生成

优化就是对程序进行等价变换,使得从变换后的程序能生成更有效的目标代码。所谓等价,是指不改变程序的运行结果;所谓有效是指目标代码运行时间较短,占用的存储空间较少。优化可在编译的各个阶段进行。最主要的优化是在目标代码生成以前,对中间代码进行的,这类优化不依赖于具体的计算机。

目标代码的生成由代码生成器实现。代码生成器以经过语义分析或优化后的中间代码为输入,以特定的机器语言或汇编代码为输出。代码生成所需考虑的主要问题有以下几个方面。

1) 中间代码形式

中间代码有多种形式,其中树与后缀表示形式适用于解释器,而编译器多采用与机器指令格式较接近的四元式形式。

2) 目标代码形式

目标代码可以分为两大类:汇编语言形式和机器指令形式。机器指令形式的目标代码又可以根据需求的不同分为绝对机器指令代码和可再定位机器代码。绝对机器代码的优点是可以立即执行,一般应用于一类称为 load-and-go 形式的编译模式,即编译后立即执行,不形成存在外存上的目标代码文件。可再定位机器代码的优点是目标代码可以被任意链接并装入内存的任意位置,是编译器采用较多的代码形式。汇编语言作为一种中间输出形式,便于进行分析和测试。

3) 寄存器的分配

由于存取寄存器的速度远快于存取内存单元的速度,所以总是希望尽可能多地使用寄存器存储数据,而寄存器的个数是有限的,因此,如何分配及使用寄存器,是目标代码生成时需要着重考虑的。

4) 计算次序的选择

代码执行的效率会随计算次序的不同有较大的差别。在生成正确目标代码的前提下,适当地安排计算次序并优化代码序列,也是生成目标代码时要考虑的重要因素之一。

2.2.3 解释程序基本原理

解释程序是另一种语言处理程序,在词法、语法和语义分析方面与编译程序的工作原理基本相同,但是在运行用户程序时,它直接执行源程序或源程序的内部形式。因此,解释程序不产生源程序的目标程序,这是它和编译程序的主要区别。图 2-22 显示了解释程序实现高级语言的 3 种方式。

源程序被直接解释执行的处理方式如图 2-22 中的标记 A 所

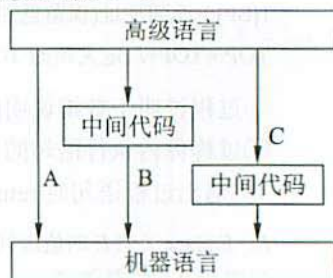


图 2-22 解释器类型示意图

示。这种解释程序对源程序进行逐个字符的检查,然后执行程序语句规定的动作。例如,如果扫描到字符串序列:“GOTO L”,解释程序就开始搜索源程序中标号 L 后面紧跟冒号“:”的出现位置。这类解释程序通过反复扫描源程序来实现程序的运行,运行效率很低。

解释程序也可以先将源程序翻译成某种中间代码形式,然后对中间代码进行解释来实现用户程序的运行,这种翻译方式如图 2-25 中的标记 B 和 C 所示。通常,在中间代码和高级语言的语句间存在一一对应的关系。APL 和 SNOBOL4 的很多实现就采用这种方法。解释方式 B 和 C 的不同之处在于中间代码的级别,在方式 C 下,解释程序采用的中间代码更接近于机器语言。在这种实现方案中,高级语言和低级中间代码间存在着“1:n”的对应关系。PASCAL-P 解释系统是这类解释程序的一个实例,它在词法分析、语法分析和语义基础上,先将源程序翻译成 P-代码,再由一个非常简单的解释程序来解释执行这种 P-代码。这类系统具有比较好的可移植性。

1. 解释程序的结构

这类系统通常可以分成两部分:第一部分是分析部分,包括通常的词法分析、语法分析和语义分析程序,经语义分析后把源程序翻译成中间代码,中间代码常采用逆波兰表示形式。第二部分是解释部分,用来对第一部分产生的中间代码进行解释执行。下面简要介绍第二部分的工作原理。

设用数组 MEM 模拟计算机的内存,源程序的中间代码和解释部分的各个子程序都存放在数组 MEM 中。全局变量 PC 是一个程序计数器,它记录了当前正在执行的中间代码的位置。这种解释部分的常见结构可以由下面两部分组成:

- (1) $PC := PC + 1$;
 - (2) 执行位于 `opcode-table[MEM[PC]]` 的子程序(解释子程序执行后返回到前面)。
- 用一个简单例子来说明其工作原理。设两个实型变量 A 和 B 进行相加的中间代码是:

start: Ipush

A

Ipush

B

Iaddreal

其中,中间代码 Ipush 和 Iaddreal 实际上都是 opcode-table 表的索引值(即位移),而该表的单元中存放的值就是对应的解释子程序的起始地址, A 和 B 都是 MEM 中的索引值。解释部分开始执行时, PC 的值为 start-1。

```
opcode-table[Ipush]=push
opcode-table[Iaddreal]=addreal
```


解释部分可表示如下:

```

interpreter-loop:  PC:=PC+1;
                  goto opcode-table[MEM[PC]];
push:             PC:=PC+1;
                  stackreal(MEM[MEM[PC]]);
                  goto interpreter-loop;
address:          stackreal(popreal()+popreal());
                  goto interpreter-loop;
... (其余各解释子程序)

```

其中, `stackreal()` 表示把相应值压入栈中, 而 `popreal()` 表示取得栈顶元素值并弹出栈顶元素。

2. 高级语言编译与解释程序的比较

对于高级语言的编译和解释工作方式, 可以从以下几个方面进行比较。

(1) 效率。编译比解释方式可能取得更高的效率。

一般情况下, 在解释方式下运行程序时, 解释程序可能需要反复扫描源程序。例如, 每一次引用变量都要进行类型检查, 甚至需要重新进行存储分配, 从而降低了程序的运行速度。在空间上, 以解释方式运行程序需要更多的内存, 因为系统不但需要为用户程序分配运行空间, 而且要为解释程序及其支撑系统分配空间。

在编译方式下, 编译程序除了对源程序进行语法和语义分析外, 还要生成源程序的目标代码并进行优化, 所以这个过程比解释方式工作需要更多的时间。虽然与仔细写出的机器程序相比, 一般由编译程序创建的目标程序运行的时间更长, 需要占用的存储空间更多。但源程序只需要被编译程序翻译一次, 就可以多次运行。因此总体来讲, 编译方式比解释方式可能取得更高的效率。

(2) 灵活性。由于解释程序需要反复检查源程序, 这也使得解释方式能够比编译方式更灵活。当解释器直接运行源程序时, “在运行中” 修改程序就成为可能, 例如增加语句或者修改错误等。另外, 当解释器直接在源程序上工作时, 它可以对错误进行更精确地定位。

(3) 可移植性。解释器一般也是用某种程序设计语言编写的, 因此只要对解释器进行重新编译, 就可以使解释器运行在不同的环境中。

由于编译和解释的方法各有特点, 因此现有的一些编译系统既提供编译的方式, 也提供解释的方式, 甚至将两种方式进行结合在一起。例如在 Java 虚拟机上发展的一种 “compiling-just-in-time” 新技术, 就是在代码第一次运行时进行编译, 在其后的运行中就不再进行编译了。

第3章 操作系统知识

3.1 操作系统基础知识

计算机系统软件极为丰富，通常分为系统软件和应用软件两大类。应用软件是指计算机用户利用计算机的软件、硬件资源为某一专门的应用目的而开发的软件。例如：科学计算、工程设计、数据处理、事务处理、过程控制等方面的程序，以及文字处理软件、表格处理软件、辅助设计软件（CAD）、实时处理软件等。系统软件是计算机系统的一部分，由它支持应用程序的运行。为用户开发应用系统提供一个平台，用户可以使用它，但不能随意修改它。一般常用的系统软件有：操作系统、语言处理程序、连接程序、诊断程序、数据库管理系统等。操作系统是计算机系统中的核心软件，其他软件是建立在操作系统的基础上，并在操作系统的统一管理和支持下运行的。

3.1.1 操作系统的定义与作用

计算机系统的硬件资源包括中央处理机（CPU）、存储器（包括主存与外存）和输入/输出设备等物理设备；计算机系统的软件资源是以文件形式保存在存储器上的程序和数据等信息。操作系统能有效地组织和管理系统中的各种软、硬件资源，合理地组织计算机系统工作流程，控制程序的执行，并且向用户提供一个良好的工作环境和友好的接口。操作系统有两个重要的作用：

1. 通过资源管理，提高计算机系统的效率

操作系统是计算机系统的资源管理者，它含有对系统软、硬件资源实施管理的一组程序。其首要作用就是通过 CPU 管理、存储管理、设备管理和文件管理，对各种资源进行合理的分配，改善资源的共享和利用程度，最大限度地发挥计算机系统的工作效率，提高计算机系统在单位时间内处理工作的能力（即系统的“吞吐量” throughput）。

2. 改善人机界面，向用户提供友好的工作环境

操作系统不仅是计算机硬件和各种软件之间的接口，也是用户与计算机之间的接口。试想如果不安装操作系统，用户将要面对的是 0 和 1 组成的代码和一些难懂的机器指令，通过按钮或按键来操作计算机，这样既笨拙，又费时。一旦安装操作系统后，用户面对的不再是笨拙的裸机，而是操作便利、服务周到的操作系统，从而明显改善用户界面，提高用户的工作效率。

3.1.2 操作系统的特征与功能

1. 操作系统的特征

操作系统的4个特征是并行性、共享性、虚拟性和不确定性。

(1) 并行性 (concurrency): 指在计算机系统中存在着许多并发执行的活动。对计算机系统而言, 并发是指宏观上看系统内有多道程序同时运行, 微观上看是串行运行。因为在大多数计算机系统中一般只有一个CPU, 在任意时刻只能有一道程序占用CPU。

(2) 共享性 (sharing): 系统中各个并发活动要共享计算机系统中的各种软、硬件资源, 因此操作系统必须解决在多道程序间合理的分配和使用资源问题。

(3) 虚拟性 (virtual): 虚拟是操作系统中的重要特征, 所谓虚拟是指把物理上的一台设备变成逻辑上的多台设备。例如: 在操作系统中采用了spooling技术, 可以利用快速、大容量可共享的磁盘作为中介, 模拟多个非共享的低速的输入输出设备, 这样的设备称为虚拟设备。

(4) 不确定性 (Non-Determinacy): 通常一个程序的初始条件相同时, 无论何时运行结果必然相同。但由于程序的并发执行系统内的各种进程错综复杂, 与这些进程有关的事件, 如从外部设备来的中断、输入/输出请求、各种运行故障等, 发生的时间都不可预测, 如果处理不当将导致系统出错, 这种不确定性所带来错误是很难查找的。

2. 操作系统的功能

操作系统具有5大管理功能。

1) 进程管理

进程管理实质上是对处理机执行“时间”的管理, 采用多道程序等技术将CPU真正合理地分配给每个任务。其功能包括:

- (1) 进程控制: 创建、撤销、挂起、改变运行优先级等。
- (2) 进程同步: 协调并发进程之间的推进步骤, 以协调资源共享。
- (3) 进程通信: 进程之间传送数据, 以协调进程间的协作。
- (4) 进程调度: 作业和进程的运行切换, 以充分利用处理机资源和提高系统性能。

2) 文件管理

文件管理又称为信息管理。其功能包括。

- (1) 文件存储空间管理: 解决如何存放信息, 以提高空间利用率和读写性能。
- (2) 目录管理: 解决信息检索问题。
- (3) 文件的读写管理和存取控制: 解决信息安全问题, 可通过系统设置用户口令、对用户分类、设置文件权限。

(4) 软件管理：软件的版本、相互依赖关系、安装和拆除等。

3) 存储管理

存储管理是对主存储器“空间”进行管理。其功能包括：

(1) 存储分配与回收：操作系统为每个程序分配存储空间，当程序运行完毕回收存储空间。

(2) 存储保护：保证进程间互不干扰、相互保密，如：访问合法性检查、甚至要防止从“垃圾”中窃取其他进程的信息。

(3) 地址映射（变换）：进程逻辑地址到主存物理地址的映射。

(4) 主存扩充（覆盖、交换和虚拟存储）：提高主存利用率、扩大进程的主存空间。

4) 设备管理

设备管理实质是对硬件设备的管理，其中包括对输入/输出设备的分配、启动、完成和回收。操作系统中含有许多设备驱动程序，用户和应用程序使用外部设备时并不需要知道外部设备的具体特性，不需要对设备的使用专门编程，对设备的具体操作都由设备驱动程序完成。这样不但简化了程序设计，而且程序运行不依赖于具体硬件配置，即与“硬件无关”。

5) 作业管理

作业管理包括任务、界面管理、人机交互、图形界面、语音控制和虚拟现实等。操作系统提供系统命令一级的接口：供用户用于组织和控制自己的作业运行，如命令行、菜单式或 GUI “联机”、命令脚本“脱机”。操作系统还提供编程一级接口：供用户程序和系统程序调用操作系统功能，如系统调用和高级语言库函数。

3.1.3 操作系统的类型

操作系统分为批量处理操作系统（简称批处理）、分时操作系统、实时操作系统、网络操作系统、分布式操作系统、微机操作系统和嵌入式操作系统。

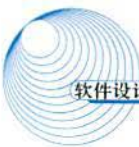
1. 批处理操作系统

在早期的计算机系统中，程序的每一次运行都需要人工干预，操作过程繁琐，占用很多人工等待的时间，也很容易产生错误，可真正执行程序的时间却很短。而且程序在执行的过程中，要独占系统的全部硬件资源，利用率很低，为此引入了批处理操作系统。批处理操作系统分为单道批处理和多道批处理两种。

在批处理系统中的作业由用户程序、数据和作业说明书（作业控制语言）3部分组成。

(1) 批处理系统的优点：同一批内各作业的自动依次执行，改善了主机 CPU 和 I/O 设备的使用效率，提高了吞吐量。

(2) 批处理系统的缺点：磁带或磁盘需要人工装卸，作业需要人工分类，监督程序易遭到用户程序的破坏（由人工干预才可恢复）；其次，由于一次要处理一批作业，在该批作业处理



过程中,任何用户都不能与计算机进行交互。即使发现了某个作业的程序有错,也要等该批作业全部结束后才能修改。这对与软件开发人员来说,不能不说是严重的缺陷。正是这一问题,导致了分时操作系统应运而生。

2. 分时系统

分时操作系统将 CPU 的工作时间划分为许多很短的时间片,轮流为各个终端的用户服务。分时系统的特点如下所述。

(1) 多路性:允许在一台主机上同时连接多台联机终端,系统按分时原则为每个用户服务。宏观上是多个用户同时工作,共享系统资源,而微观上则是每个用户作业轮流运行一个时间片。多路性即同时性,它提高了资源利用率,从而促进了计算机更广泛的应用。

(2) 独立性:每个用户各占一个终端,彼此独立操作,互不干扰。因此,用户会感觉到就像他一人独占主机。

(3) 交互性:用户可通过终端与系统进行人机对话。用户可以请求系统提供多方面服务,如文件编辑、数据处理和资源共享等。

(4) 及时性:用户的请求能在很短时间内获得响应,此时间间隔是以人们所能接受的等待时间确定的,通常为 1~2 s。响应时间是分时系统的重要指标,它是用户发出终端命令到系统做出响应的时间间隔。系统的响应时间主要是根据用户所能接受的等待时间确定的。分时系统中时间片的选择是一个复杂和关键的任务,如时间片选得过大,造成响应时间不变时会导致用户数减少,或造成响应时间过长;当时间片选择过小时,在一个时间片内切换开销相对增加,一个进程相对要花费更多的时间片才能结束,一个进程在系统中的周转时间大大增长。最佳的时间片值应既能使分时用户得到好的响应时间,同时又要使在一个时间片内切换开销相对较小(可忽略)。

UNIX 系统是典型多用户、多任务的分时操作系统。

3. 实时系统

实时是指计算机对于外来信息能够以足够快的速度进行处理,并在被控对象允许的时间范围内做出快速反应。实时系统对交互能力要求不高,但要求可靠性有保障。为了提高系统的响应时间,对随机发生的外部事件做出及时响应并对其进行处理。

1) 实时系统的类型

实时系统分为实时控制系统和实时信息处理系统。

(1) 实时控制系统:主要用于生产过程的自动控制、实验数据自动采集、武器的控制(包括火炮自动控制、飞机自动驾驶、导弹的制导系统)。

(2) 实时信息处理系统:主要用于实时信息处理,如飞机订票系统、情报检索系统。

2) 实时系统的主要特点

(1) 快速的响应时间：实时系统是为了提高系统响应时间而设计的操作系统，特别是实时控制系统，对外部事件的响应要十分及时。外部事件往往以中断方式通知系统，系统有较强的中断处理能力，实时系统的设计也以“事件驱动”方式来设计。

(2) 有限的交互能力：实时系统（如实时信息处理系统）一般是专用系统，它能提供人机交互方式，但用户只能访问系统中某些特定的专用服务程序，不能像分时系统那样向终端用户提供多方面的服务。

(3) 高可靠性：批处理系统和分时系统虽也要求系统可靠，相比之下，实时系统则要求系统高度可靠。因此，实时系统中往往都采用双机系统和多级容错措施来保证系统和数据的安全。

3) 实时系统与分时系统的区分

实时系统与分时系统除了应用的环境不同，主要区分如下：

(1) 系统的设计目标不同：分时系统是设计成一个多用户的通用系统，交互能力强；而实时系统大都是专用系统。

(2) 交互性的强弱：分时系统是多用户的通用系统，交互能力强，而实时系统是专用系统，仅允许操作并访问有限量的专用程序，不能随便修改其中的程序，且交互能力差。

(3) 响应时间的敏感程度不同：分时系统是以人能接收的等待时间为系统的设计依据，而实时系统是以被测物体所能接受的延迟为系统设计依据。因此，实时系统对响应时间的敏感程度强，而分时系统的敏感程度弱。

4. 网络操作系统

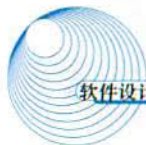
网络操作系统是使联网的计算机能方便而有效地共享网络资源，为网络用户提供所需各种服务的软件和有关协议的集合。因此，网络操作系统的功能主要包括高效、可靠的网络通信；对网络中共享资源（在 LAN 中有硬盘、打印机等）进行有效的管理；提供电子邮件、文件传输、共享硬盘、打印机等服务；网络安全管理；提供互操作能力。

5. 分布式操作系统

分布式计算机系统是由多个分散的计算机经网络连接而成的计算机系统，系统中的计算机无主次之分，任意两台计算机可以通过通信交换信息。为分布式计算机配置的操作系统就是“分布式操作系统”。

分布式操作系统能直接对系统中各类资源进行动态的分配和调度、任务划分、信息传输协调工作，并为用户提供一个统一的界面和标准的接口，用户通过这一界面实现所需要的操作以及使用系统资源，使系统中若干台计算机相互协作完成共同的任務，有效地控制和协调诸任务的并行执行，并向系统提供统一的、有效的接口软件集合。

分布式操作系统是网络操作系统的更高级形式，它保持网络系统所拥有的全部功能，同时



又具有透明性、可靠性、高性能等。网络操作系统与分布式操作系统虽然都属于管理分布在不同地理位置的计算机,但最大的差别是:网络操作系统工作时必须确认网址,而分布式系统用户则不必知道计算机的确切地址;分布操作系统负责整个系统的资源分配,通常能够很好地隐藏系统内部的实现细节,如对象的物理位置、并发控制、系统故障等。这些对用户都是透明的。

6. 微机操作系统

微型计算机拥有巨大的使用量和广泛的用户。配置在微型计算机上的操作系统称为微机操作系统。常用的微机操作系统有 MS-DOS、MS Windows、OS/2、SCO UNIX、Linux 等。其中:Microsoft 公司开发的单用户单任务操作系统 MS-DOS 是首先在 IBM-PC 机上使用的微机操作系统,MS-DOS 操作系统是事实上的 16 位微机单用户单任务操作系统的标准。

多任务操作系统 Windows98/NT/2000/XP 是 Microsoft 公司开发的一个图形用户界面的多任务、多线程、全 32 位的操作系统。多用户多任务操作系统 SCO UNIX 是 SCO 公司将运行于大、中、小型机上 UNIX 操作系统移植到微机上的多用户多任务操作系统。

Linux 操作系统是一个遵循标准操作系统界面的免费操作系统,具有 UNIX BSD 和 UNIX SYS V 的扩展特性,它的创造者是芬兰籍的 Linus B.Torvalds 和其他开发人员。Linux 推行公共许可证(General Public License, GPL),保证用户有使用上的自由,获得源程序的自由、修改的自由,以及复制和推广的自由。

7. 嵌入式操作系统

嵌入式操作系统是运行在嵌入式智能芯片环境中,对整个智能芯片及其控制的各种部件装置等资源进行统一协调、处理、指挥和控制的系统软件。

3.2 处理机管理

在多道程序批处理系统和分时系统中,有多个并发执行的程序,此时用程序这个静态的概念已经不能描述系统中程序动态变化的过程,所以引入了进程概念。进程是资源分配和独立运行的基本单位。研究操作系统的进程,实质上是研究系统中诸进程之间的并发特性以及进程之间的相互制约性。

3.2.1 基本概念

1. 程序与进程

1) 程序的顺序执行

一个较大的程序通常都由若干个程序段组成,程序在执行时,各程序段必须按照先后次序

逐个执行。程序各程序段先后执行的次序关系可用前趋图表示。

前趋图是一个有向无循环图,图由节点和节点间的有向边组成,节点代表各程序段的操作,而节点间的有向边表示两程序段操作之间存在的前趋关系(“→”)。两程序段 P_i 和 P_j 的前趋关系表示成 $P_i \rightarrow P_j$, P_i 是 P_j 的前趋, P_j 是 P_i 的后继。图 3-1 为 3 个程序段,第一个程序段输入是第二个程序段计算的前趋,第二个程序段计算是第三个程序段输出的前趋。



图 3-1 3 个节点的前趋图

程序顺序执行时的特征如下:

- (1) 顺序性: 程序各程序段严格按照规定的顺序执行。
- (2) 封闭性: 程序运行时机内各资源只受该程序控制而改变, 执行结果不受外界因素影响。
- (3) 可再现性: 只要程序执行环境和初始条件相同, 程序多次执行, 可获得相同结果。

2) 程序并发执行的特征

若在计算机系统中采用多道程序设计技术, 则主存中的多道程序可处于并发执行状态。对于上述有 3 个程序段的作业, 虽然每个作业有前趋关系的各程序段不能在 CPU 和输入/输出各部件并行执行, 但是同一个作业内没有前趋关系的程序段或不同作业的程序段可以分别在 CPU 和各输入/输出部件上并行执行。例如, 某系统中有一个 CPU、一台输入设备和一台输出设备, 每个作业具有 3 个程序段输入 I_i , 计算 C_i 和输出 $P_i(i=1,2,3)$ 。图 3-2 为 3 个作业的各程序段并发执行的前趋图。

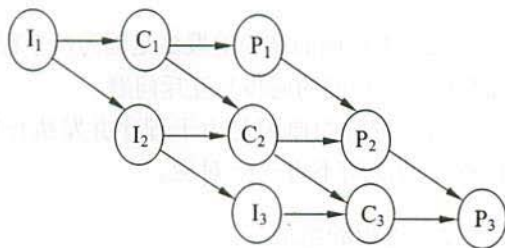


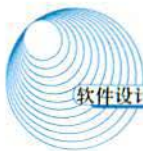
图 3-2 程序并发执行的前趋图

从图 3-2 中可以看出, I_2 与 C_1 并行执行; I_2 、 C_2 与 P_1 并行执行; C_3 与 P_2 并行执行。其中: I_2 、 I_3 受到 I_1 的间接制约, C_2 、 C_3 受到 C_1 的间接制约, P_2 、 P_3 受到 P_1 的间接制约。同理, C_1 、 P_1 受到 I_1 的直接制约等。

程序并发执行时的特征如下:

- (1) 失去了程序的封闭性。
- (2) 程序和机器的执行程序的活动不再一一对应。
- (3) 并发程序间的相互制约性。

例如, 两个并发执行的程序段完成交通流量的统计, 其中“观察者” P_1 识别通过的车辆数,



“报告者”P2 定时将观察者的计数值清“0”。程序实现如下:

```
P1                                P2
L1:  if 有车通过 then            L2:  PRINT COUNT;
COUNT:=COUNT+1;                COUNT:=0;
GOTO L1;                          GOTO L2;
```

对于上例, 由于程序可并发执行, 所以可能有以下 3 种执行序列:

```
COUNT:=COUNT+1; PRINT COUNT; COUNT:=0
PRINT COUNT; COUNT:=0; COUNT:=COUNT+1
PRINT COUNT; COUNT:=COUNT+1; COUNT:=0
```

假定 COUNT 的某个循环的初值为 n , 那么对这 3 种执行序列得到的 COUNT 结果不同, 如表 3-1 所示:

表 3-1 程序并发执行的结果

执行序列	①	②	③
COUNT 打印的值	$n+1$	n	N
COUNT 执行后的值	0	1	0

这种不正确的结果的发生是因为两个程序共享变量 COUNT 引起的, 为了解决这一问题, 需要研究进程间的同步与互斥问题。

引入进程的原因是由于程序并发执行破坏了程序的封闭性和可再现性, 使得程序和执行程序的活动不再一一对应。

2. 进程的组成

进程是程序的一次执行, 该程序可以和其他程序并发执行。进程通常是由程序、数据及进程控制块 (PCB) 组成的。

(1) 程序: 进程的程序部分描述了进程需要完成的功能。假如一个程序能被多个进程同时共享执行, 那么这一部分就应该以可再入 (纯) 码的形式编制, 它是程序执行时不可修改的部分。

(2) 数据: 进程的数据集合部分包括了程序在执行时所需的数据及工作区。这部分只能为一个进程所专用, 是进程的可修改部分。

(3) 进程控制块: 是进程的描述信息和控制信息, 是进程动态特性的集中反映, 也是进程存在的唯一标志。进程控制块主要包含的内容如表 3-2 所示。

表 3-2 PCB 的内容

信息	含义
进程标识符	标明系统中的各个进程
状态	说明进程当前的状态
位置信息	指明程序及数据在主存或外存的物理位置
控制信息	参数、信号量、消息等
队列指针	链接同一状态的进程
优先级	进程调度的依据
现场保护区	将处理机的现场保护到该区域以便再次调度时能继续正确运行
其他	因不同的系统而异

3. 进程的状态及其状态间的切换

1) 三态模型

在多道程序系统中, 进程在处理器上交替运行, 状态也不断地发生变化, 因此进程一般有 3 种基本状态: 运行、就绪和阻塞。图 3-3 显示了进程基本状态及其转换, 也称三态模型。

(1) 运行: 当一个进程在处理器上运行时, 则称该进程处于运行状态。显然对于单处理机系统, 处于运行状态的进程只有一个。

(2) 就绪: 一个进程获得了除处理机外的一切所需资源, 一旦得到处理机即可运行, 则称此进程处于就绪状态。

(3) 阻塞: 也称等待或睡眠状态, 一个进程正在等待某一事件发生(例如请求 I/O 而等待 I/O 完成等)而暂时停止运行, 这时即使把处理机分配给进程也无法运行, 故称该进程处于阻塞状态。

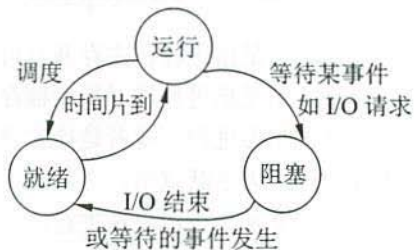


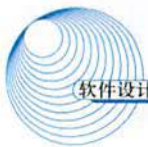
图 3-3 进程的三态模型

2) 五态模型

事实上, 对于一个实际的系统, 进程的状态及其转换将更复杂。例如, 引入新建态和终止态构成了进程的五态模型。如图 3-4 所示。

其中, 新建态对应于进程刚刚被创建时没有被提交的状态, 并等待系统完成创建进程的所有必要信息。因为创建进程时分为两个阶段, 第一个阶段为一个新进程创建必要的管理信息, 第二个阶段让该进程进入就绪状态。由于有了新建态, 操作系统往往可以根据系统的性能和主存容量的限制推迟新建态进程的提交。

类似地, 进程的终止也可分为两个阶段, 第一个阶段等待操作系统进行善后处理, 第二个阶段释放主存。



3) 具有挂起状态的进程状态及其转换

由于进程的不断创建,系统资源特别是主存资源已不能满足所有进程运行的要求。这时,就必须将某些进程挂起,放到磁盘对换区,暂时不参加调度,以平衡系统负载;进程挂起的原因可能是系统出现故障,或者是用户调试程序,也可能是需要检查问题。图 3-5 是具有挂起状态的进程状态及其转换。

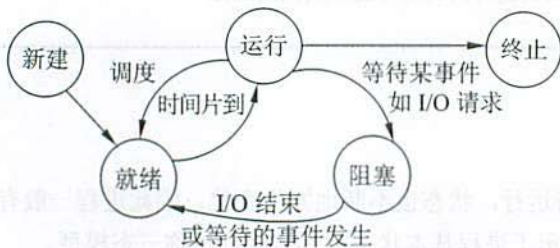


图 3-4 进程的五态模型

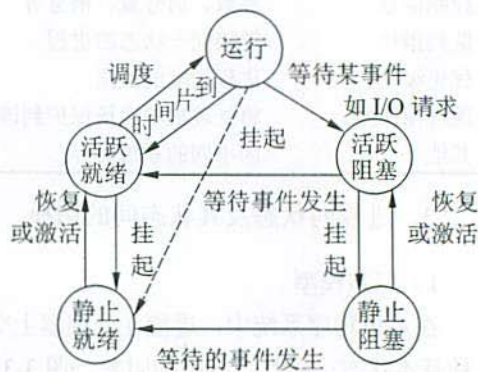


图 3-5 细分进程状态及其转换

活跃就绪是指进程在主存并且可被调度的状态。

静止就绪是指进程被对换到辅存时的就绪状态,是不能被直接调度的状态,只有当主存中没有活跃就绪态进程,或者是挂起就绪态进程具有更高的优先级,系统将把挂起就绪态进程调回主存并转换为活跃就绪。

活跃阻塞是指进程已在主存,一旦等待的事件产生便进入活跃就绪状态。

静止阻塞是指进程对换到辅存时的阻塞状态,一旦等待的事件产生便进入静止就绪状态。

3.2.2 进程的控制

进程的控制就是对系统中所有进程从创建到消亡的全过程实施有效的控制。这意味着不仅要控制正在运行的进程,而且还要能创建新的进程,撤销已完成的进程。

为了对进程进行有效的控制,操作系统必须设置一套控制机构。进程的控制机构是由操作系统内核实现的。通常将与硬件密切相关的模块放在紧挨硬件的软件层中,并使它们存储在主存,以便提高操作系统的运行效率。例如中断处理程序、常用的设备驱动程序、运行频率较高的模块(如时钟中断、进程调度许多模块公用的基本模块)。通常将这部分称之为操作系统的内核,它为系统实现进程控制和存储器管理提供了有效的控制机制。不同的操作系统内核所包含的功能不同,但大多数操作系统的内核包含支撑功能和资源管理的功能。

1. 支撑功能

(1) 中断处理：操作系统的各种重要的活动最终都将依赖于中断。如各种类型的系统调用、键盘命令的输入、设备驱动及文件系统等都依赖于中断。通常内核只对中断进行“有限次处理”，然后转入有关进程继续处理。这不仅可以减少中断处理的时间，也可以提高程序的并发性。

(2) 时钟管理：操作系统的许多活动要用到它。如分时系统时间片调度算法中，当时间片用完时，由时钟管理产生一个中断信号，通知调度程序重新调度。在实时系统中的截止时间控制、批处理系统中的最长运行时间的控制等。

(3) 原语操作：内核在执行某些基本操作时，往往是通过原语操作来实现的。原语是由若干条机器指令构成的，用于完成特定功能的一段程序。原语在执行的过程中是不可分割的。内核中所包含的原语主要有进程控制、进程通信、资源管理以及其他方面的原语。

2. 资源管理功能

(1) 进程管理：进程管理的大部分或全部功能都放在内核中。如进程的调度与分配、进程的创建和撤销。

(2) 存储器管理：如主存的分配和回收、主存保护和对换等功能模块都放在内核中。

(3) 设备管理：如设备驱动程序、缓冲区管理、设备分配等功能模块都放在内核中。

3. 进程控制原语

进程控制原语主要有进程创建原语、进程撤销原语、进程挂起原语、进程激活原语、进程阻塞原语以及进程唤醒原语。

1) 创建原语与撤销原语

(1) 创建原语：创立一个进程实际上就是先为其创建一个 PCB，设置好 PCB 的各项参数，其次装入该进程的实体（程序和数据）。通常有两种情况需要创建进程：在系统生成时就建立起一些系统进程，如系统的输入读进程和输出写进程；通过创建原语产生进程，通常用创建原语创建的主要是非常驻主存的系统进程和用户进程。

(2) 撤销原语：一旦操作系统发现了要求终止进程的事件后，便调用进程撤销原语终止一个指定的进程。撤销原语将读出进程的状态，若被终止的进程处于“运行”状态，应立即中断其执行，保护 CPU 现场，设置重新调度标志；若有子孙进程，还应终止所有的子孙进程，以免其成为不可控的；收回所有的资源，或者归还父进程，或者归还系统；将被终止的进程从队列中移出。

引起进程终止的事件有正常结束、异常结束（如越界错误、保护错、特权指令错、运行超时、等待超时及 I/O 故障）、外界干预（操作员或操作系统干预、父进程请求、父进程终止）等。



2) 挂起原语与激活原语

(1) 挂起原语: 当出现了引起进程挂起的事件时, 挂起原语将检查被挂起进程的状态。调用挂起原语的进程只能挂起它自己或它的子孙, 而不能挂起别的族系的进程。挂起原语的执行过程是: 检查要挂起进程 PCB 的现行状态, 若正处于活动就绪态, 便将它改为静止就绪态; 如是活动阻塞态则改为静止阻塞态。如是运行态, 则将它改为静止就绪态, 并调用进程调度程序重新分配处理机。为了方便用户或父进程考察该进程的运行情况, 需把该进程的 PCB 复制到主存指定区域。

(2) 激活原语: 当发生了激活进程的事件时, 如用户进程或父进程请求激活指定的进程, 激活原语将检查被激活进程的状态, 若进程处于静止就绪, 则置为活动就绪; 若进程处于静止阻塞, 则置为活动阻塞。

3) 阻塞原语与唤醒原语

当正在执行的进程请求操作系统提供服务时, 由于请求的设备或资源得不到满足, 此时只有用阻塞原语将其阻塞起来, 一旦等待的资源空闲, 则用唤醒原语唤醒该进程。引起进程阻塞的原因主要有:

(1) 启动某种操作: 如某进程启动某种操作后, 必须等待操作结束才能继续运行, 如启动 I/O 设备, 当 I/O 操作结束了, 由中断处理进程将其唤醒。

(2) 新数据尚未到达: 当两个相互合作的进程, 一个进程必须等待另一个进程的结果, 此时必须阻塞一个进程, 当等待的事件发生了, 再将其唤醒。

(3) 无新工作可做: 在系统中往往设置一些具有特定功能的系统进程, 每当任务完成后, 这些进程将自己阻塞起来, 当有新的任务再将其唤醒。

3.2.3 进程间的通信

1. 同步与互斥

在计算机系统中, 多个进程可以并发执行, 因此进程间必然存在共享资源和相互合作的问题。

1) 进程间的同步

一般来说, 一个进程相对于另一个进程的运行速度是不确定的, 也就是说进程是在异步环境下运行的, 每个进程都有各自独立的、不可预知的速度向前推进。但是进程相互合作的进程需要在某些确定点上协调它们的工作, 当一个进程到达了这些点后, 除非另一进程已经完成了某些操作, 否则就不得不停下来等待这些操作结束。这就是进程间的同步。

2) 进程间的互斥

在多道程序系统中, 各进程可以共享各类资源, 但有些资源一次只能共一个进程使用, 称为临界资源 (Critical Resource, CR), 如打印机、公共变量、表格等。

通常, 同步是进程间的直接制约问题, 互斥是进程间的间接制约问题。

3) 临界区管理的原则

临界区 (Critical Section, CS) 是进程中对临界资源实施操作的那段程序。对互斥临界区管理的原则是:

(1) 有空即进: 当无进程处于临界区时, 允许进程进入临界区, 并且只能在临界区运行有限的时间。

(2) 无空则等: 当有一进程在临界区时, 其他欲进入临界区的进程必须等待, 以保证进程互斥地访问临界资源。

(3) 有限等待: 对要求访问临界资源的进程, 应保证进程能在有限时间进入临界区, 以免陷入“饥饿”状态。

(4) 让权等待: 当进程不能进入自己的临界区时, 应立即释放处理机, 以免进程陷入“忙”状态。

2. 信号量机制

1965 年, 荷兰学者 Dijkstra 提出的信号量机制是一种卓有成效的进程同步与互斥的工具。目前信号量机制有了很大的发展, 主要有: 整型信号量、记录型信号量、信号量集机制。

1) 整型信号量与 PV 操作

信号量是一个整型变量, 根据控制对象的不同赋不同的值。信号量分为两类。

- 公用信号量: 实现进程间的互斥, 初值=1 或资源的数目。
- 私用信号量: 实现进程间的同步, 初值=0 或某个正整数。

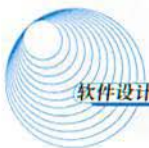
信号量 S 的物理意义是 $S \geq 0$ 表示某资源的可用数, $S < 0$ 其绝对值表示阻塞队列中等待该资源的进程数。

对系统中的每一个进程, 其工作的正确与否不仅取决于它自身的正确性, 而且与它在执行中能否与其他相关进程正确的实施同步互斥有关。PV 操作是实现进程同步与互斥的常用方法。PV 操作是低级通信原语, 在执行期间不可分割。其中, P 操作表示申请一个资源, V 操作表示释放一个资源。

P 操作定义: $S := S - 1$, 若 $S \geq 0$, 则执行 P 操作的进程继续执行; 否则若 $S < 0$, 则置该进程为阻塞状态 (因为无可用资源), 并将其插入阻塞队列。

P 操作可用如下过程表示:

```
Procedure P (Var S: Semaphore);  
Begin  
    S := S - 1;  
    If S < 0 then W(S)      {执行 P 操作的进程插入等待队列}  
End;
```

V 操作定义: $S := S + 1$, 若 $S > 0$, 则执行 V 操作的进程继续执行; 否则若 $S \leq 0$, 则从阻塞状态唤醒一个进程, 并将其插入就绪队列, 然后执行 V 操作的进程继续。

V 操作可用如下过程表示:

```
Procedure V(Var S:Semaphore);
Begin
  S:=S+1;
  If S≤0 then R(S)      {从阻塞队列中唤醒一个进程}
End;
```

2) 利用 PV 操作实现进程的互斥

令信号量 mutex 的初值为 1, 当进入临界区时执行 P 操作, 退出临界区时执行 V 操作。这样进入临界区的代码段如下:

```
P(mutex)
临界区
V(mutex)
```

【例 3.1】 将交通流量统计程序改写如下:

P1	P2
L1: if 有车通过 then	L2: begin
begin	P(mutex)
P(mutex)	PRINT COUNT;
COUNT:=COUNT+1;	COUNT:=0;
V(mutex)	V(mutex)
end	end
GOTO L1;	GOTO L2;

3) 利用 PV 操作实现进程的同步

进程的同步是由于进程间合作引起的相互制约的问题, 要实现进程的同步可用一个信号量与消息联系起来, 当信号量的值为“0”时表示希望的消息未产生, 当信号量的值为非“0”时表示希望的消息已经存在。假定用信号量 S 表示某条消息, 进程可以通过调用 P 操作测试消息是否到达, 调用 V 操作通知消息已准备好。最典型的是单缓冲区的生产者和消费者的同步问题。

【例 3.2】 生产者进程 P_1 不断地生产产品送入缓冲区, 消费者进程 P_2 不断地从缓冲区中取产品消费。为了实现 P_1 与 P_2 进程间的同步问题, 需要设置一个信号量 S_1 , 且初值为 1, 表示缓冲区空, 可以将产品送入缓冲区; 设置另一个信号量 S_2 , 且初值为 0, 表示缓冲区有产品。其同步过程如图 3-6 所示。

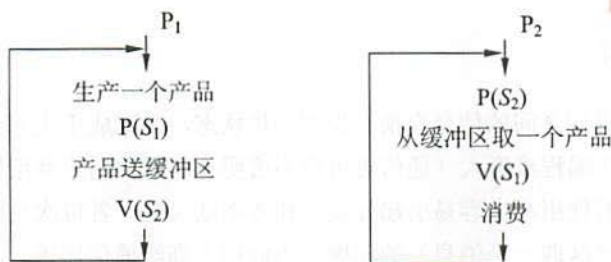


图 3-6 单缓冲区的同步举例方法 1

若信号量 S_1 的初值可以设为 0, 此时同步过程图 3-7 所示。

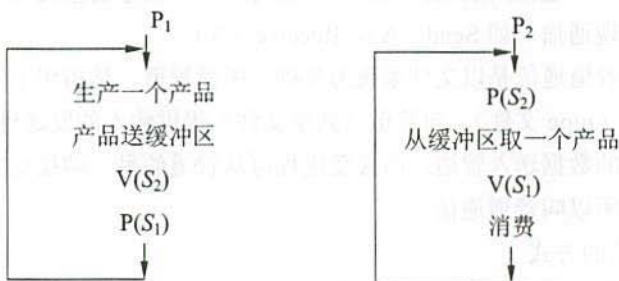


图 3-7 单缓冲区的同步举例方法 2

【例 3.3】 一个生产者和一个消费者, 缓冲区可存放 n 件物品。生产者不断地生产产品, 消费者不断地消费产品。如何用 PV 操作实现生产者和消费者的同步。可以通过设置 3 个信号量 S 、 S_1 和 S_2 , 其中, S 是一个互斥信号量初值为 1, 因为缓冲区是一个互斥资源, 所以需要进行互斥控制; S_1 表示是否可以将物品放入缓冲区, 初值为 n ; S_2 : 表示缓冲区是否存有物品, 初值为 0。同步过程如图 3-8 所示。

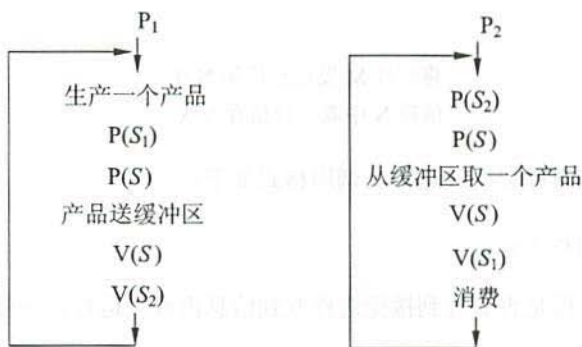


图 3-8 缓冲区容量为 n 时的同步举例



3. 高级通信原语

进程间通信是指进程之间的信息交换,少则一种状态,多则成千上万个信息,若用前面介绍的PV操作实现存在编程难度大(通信对用户不透明,即需要用户利用低级通信工具实现进程间的同步与互斥。若使用不当容易引起死锁)和效率低(生产者每次只能向缓冲区放一个消息,消费者只能从缓冲区取一个消息)的问题。为此引入高级通信原语。

1) 进程高级通信的类型

(1) 共享存储系统:相互通信的进程共享某些数据结构或存储区实现进程之间的通信。

(2) 消息传递系统:进程间的数据交换以消息为单位,程序员直接利用系统提供的一组通信命令(原语)来实现通信。如 Send(A), Receive(A)。

(3) 管道通信:管道通信是以文件系统为基础。所谓管道,是指用于连接两个进程之间的一个打开的共享文件(pipe文件)。向管道(共享文件)提供输入的发送进程(即写进程),以字符流的形式将大量的数据送入管道;而接受进程可从管道的另一端接受大量的数据。由于他们通信是采用管道,所以叫管道通信。

2) 进程高级通信的方式

(1) 直接通信:直接通信是将消息直接发送给指定进程,因此,Send和Receive原语中应指出进程名字。其调用格式如下:

Send(Who, Message)	发送消息给指定进程或一组进程
Receive(Who, Message)	从约定进程接受消息

(2) 间接通信:这种通信是以信箱为媒体来实现通信的,只要接收信件的进程设立一个信箱,这样,若干个进程都可以向同一个进程发送信件。因此,Send和Receive原语中应给出信箱名。其调用格式如下:

Send(N, M)	将信件M发送到信箱N中
Receive(N, X)	信箱N中取一封信存入X

但有的系统还提供带标记的发送,其调用格式如下:

Send(Who, Message, tag)

用Tag可以指定进程是否要等到接受进程取到信息再继续运行。一般接收进程总是要等待消息到达后才继续运行。

3.2.4 管程

1. 管程的引入

用信号量和 PV 操作来解决进程的同步与互斥问题, 需在程序中适当位置安排 PV 操作, 否则会造成死锁错误, 为了解决分散编程带来的困难, 汉森 (Brinsh Hansen) 和霍尔 (Hoare) 在 1974 年和 1975 年提出了另一种同步机制——管程。其基本思路是采用资源集中管理的方法, 将系统中的资源用某种数据结构抽象地表示出来。由于临界区是访问共享资源的代码段, 可以建立一个管程管理进程提出的访问请求。

采用这种方式对共享资源的管理就可以借助数据结构及在其上实施操作的若干过程来进行; 对共享资源的申请和释放可以通过过程在数据结构上的操作来实现。

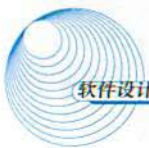
管程是由一些共享数据、一组能为并发进程所执行的作用在共享数据上的操作的集合、初始代码以及存取权组成的。管程提供了一种可以允许多进程安全有效地共享抽象数据类型的机制, 管程实现同步机制的基础是“条件结构”(condition construct)。为实现进程互斥同步, 必须定义一些条件变量, 例如: `var notempty, notfull: condition`。这些条件变量只能被 `wait` 和 `signal` 操作所访问。`notfull.wait` 操作意味着调用该操作的进程将被挂起, 使另一个进程执行; 而 `notfull.signal` 操作仅仅是启动一个被挂起的进程, 如无挂起进程则 `notfull.signal` 操作相当于空操作, 不改变 `notfull` 状态, 这不同于 V 操作。

2. 管程的结构

每一个管程都要有一个名字以供标识, 用语言来写一个管程的形式如下所示:

```

Type <管程名>=monitor
  <管程变量说明>;
  define <(能被其他模块引用的) 过程名列表>;
  procedure <过程名>(<形式参数表>)
    begin
      <过程体>;
    end;
    ...
  procedure <过程名>(<形式参数表>)
    begin
      <过程体>;
    end;
  begin
    <管程的局部数据初始化语句>;
  end
  
```

3. 利用管程解决生产者-消费者问题

【例 3.4】 建立一个管程 Producer-Consumer 它包括两个过程 put (item) 和 get (item), 它们分别执行将生产的信息放入缓冲池和从缓冲池取出信息的操作, 设置变量 count 表示缓冲池已存消息的数目。管程描述如下所示:

```
Type Producer-Consumer=monitor
var  buffer : array [ 0 , ... ,n-1] of item ; {定义缓冲区}
in , out , count : integer ; {in out 为存取指针, count 缓冲区产品个数}
notfull , notempty : condition ; {条件变量}
procedure entry put (item)
begin
    if count >= n then notfull.wait ; {缓冲区已满}
    buffer ( in ) := nextp ;
    in := (in+1) mod n ;
    count = count + 1 ;
    if notempty.queue then notempty.signal ; {唤醒等待者}
endprocedure entry get ( item)
begin
    if count <= 0 then notempty.wait ; {缓冲区已空}
    nextc := buffer ( out ) ;
    out := (out+1) mod n ;
    count := count - 1 ; {减少一个产品}
    if notfull.queue then notfull.signal ; {唤醒等待者}
end
begin in := out := 0 ; count := 0 ; end {初始化}
```

利用管程解决生产者-消费者问题, 其中生产者和消费者程序如下:

```
cobegin
    producer : begin
        repeat
            produce an item in nextp ;
            Producer-Consumer.put ( item ) ;
        until false ;
    end
    consumer : begin
        repeat
            Producer-Consumer.get ( item ) ;
```

```
consume the item in nextc ;  
until false ;  
end;
```

```
coend
```

3.2.5 进程调度

1. 操作系统的3级调度

在某些操作系统中,一个作业从提交到完成需要经历高、中、低3级调度。

(1) 高级调度:又称“长调度”或“作业调度”或“接纳调度”。它决定处于输入池中的哪个后备作业可以调入主系统做好运行的准备,成为一个或一组就绪进程。系统中一个作业只需经过一次高级调度。

(2) 中级调度:又称“中程调度”或“对换调度”。它决定处于交换区中的就绪进程哪个可以调入主存,以便直接参与对CPU的竞争。在主存资源紧张时,为了把进程调入主存,必须将主存中处于阻塞状态的进程调至交换区,以便为调入进程腾出空间。这相当于使处于主存的进程和处于交换区的进程互换了位置。

(3) 低级调度:又称“短程调度”或“进程调度”。它决定处于主存中的就绪进程哪个可以占用CPU。是操作系统中最活跃、最重要的调度程序,对系统的影响很大。

2. 进程调度方式与算法

1) 进程调度方式

调度方式是指当有更高优先级的进程到来时如何分配CPU。调度方式分为可剥夺和不可剥夺两种。可剥夺式是指当有更高优先级的进程到来时,强行将正在运行进程占用的CPU分配给高优先的进程;不可剥夺式是指当有更高优先级的进程到来时,必须等待正在运行进程自动释放占用的CPU,然后将CPU分配给高优先的进程。

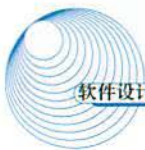
2) 进程调度算法类型

常用的进程调度算法有先来先服务、时间片轮转、优先数调度和多级反馈调度算法。

(1) 先来先服务(FCFS):是按照作业提交或进程变为就绪状态的先后次序,分配CPU。即每当进入进程调度时,总是将就绪队列队首的进程投入运行。FCFS算法主要用于宏观调度,其特点是:比较有利于长作业,而不利于短作业;有利于CPU繁忙的作业,而不利于I/O繁忙的作业。

(2) 时间片轮转(Round Robin):其基本思路是通过时间片轮转,提高进程并发性和响应时间特性,提高资源利用率。时间片轮转算法主要用于微观调度,其设计目标是提高资源利用率。

(3) 优先级调度:分为静态优先级和动态优先级两种。



① 静态优先级：进程的优先级是在创建时就已确定好了，直到进程终止都不会改变。

② 动态优先级：在创建进程时赋予一个优先级，在进程运行过程中还可以改变，以便获得更好的调度性能。

(4) 多级反馈调度算法：是在时间片轮转算法和优先级算法的基础上改进的。其优点是：照顾了短进程以提高系统吞吐量、缩短了平均周转时间；照顾 I/O 型进程以获得较好的 I/O 设备利用率和缩短响应时间；不必估计进程的执行时间，动态调节优先级。

3) 进程调度算法实现

(1) 设置多个就绪队列：队列 1，队列 2，…，队列 n ，分别赋予不同的优先级，队列 1 的优先级 \geq 队列 2 的优先级 $\geq \dots \geq$ 队列 n 的优先级。每个队列执行时间片的长度也不同，规定优先级越低则时间片越长，如逐级加倍。

(2) 新进程进入内存后，先投入队列 1 的末尾，按 FCFS 算法调度；若某进程在队列 1 的一个时间片内未能执行完，则降低投入到队列 2 的末尾，同样按 FCFS 算法调度；如此下去，当进程降低到最后的队列，则按“时间片轮转”算法调度直到完成。

(3) 仅当较高优先级的队列为空，才调度较低优先级的队列中的进程执行。如果进程执行时有新进程进入较高优先级的队列，则抢先执行新进程，并把被抢先的进程投入原队列的末尾。

3. 进程优先级的确定

过程优先级确定需要考虑如下情况：

(1) 对 I/O 型进程：让其进入最高优先级队列，以便能及时响应需要 I/O 交互的进程。通常执行一个小的时间片，在该时间片内，要求可处理完一次 I/O 请求的数据，然后转入到阻塞队列。

(2) 对计算型进程：每次都执行完时间片，进入更低级队列。最终采用最大时间片来执行，减少调度次数。

(3) 对 I/O 次数不多而主要是 CPU 处理的进程：在 I/O 完成后，放回优先 I/O 请求时离开的队列，以免每次都回到最高优先级队列后再逐次下降。

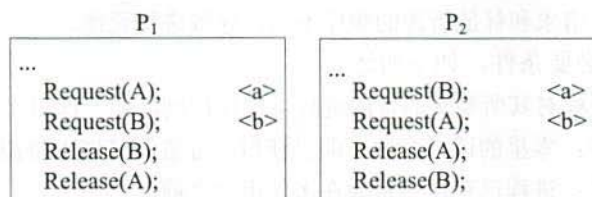
(4) 为适应一个进程在不同时间段的运行特点，I/O 完成时，提高优先级；时间片用完时，降低优先级。

3.2.6 死锁

在计算机系统中有许多互斥资源（如磁带机、打印机、绘图仪等）或软件资源（如进程表、临界区等），若两个进程同时使用打印机，或同时进入临界区必然会出现问题。所谓死锁是指两个以上的进程互相都因请求对方已经占有的资源，导致无法继续运行下去的现象。

1. 死锁举例

【例 3.5】 进程推进顺序不当引起的死锁。设系统中有一台读卡机 A，一台打印机 B，它们被进程 P_1 和 P_2 共享，两个进程并发执行并按下列顺序请求和释放资源：



假如按 $P_1<a> P_2<a> P_1 P_2$ 的次序执行，则系统发生死锁。因为进程 $P_1<a>$ 时，由于读卡机未被占用，所以请求可以得到满足；进程 $P_2<a>$ 时，由于打印机未被占用，所以请求也可以得到满足；接着进程 P_1 时，由于打印机被占用，所以请求得不到满足， P_1 等待；进程 P_2 时，由于读卡机被占用，所以请求得不到满足， P_2 也等待。导致互相在请求对方已占有的资源，系统发生死锁。

【例 3.6】 同类资源分配不当引起死锁。若系统中有 m 个资源被 n 个进程共享，当每个进程都要求 k 个资源，而 $m < nk$ 时，即资源数小于进程所要求的总数时，可能会引起死锁。例如， $m=5$ ， $n=3$ ， $k=3$ ，若系统采用的分配策略是轮流地为每个进程分配，则第一轮系统先为每个进程分配 1 台，还剩下 2 台；第二轮系统再为 2 个进程各分配 1 台，此时，系统中已无可供分配的资源，使得各个进程都处于等待状态导致系统发生死锁。

【例 3.7】 PV 操作使用不当引起的死锁。对于图 3-9，当信号量 $S_1=S_2=0$ 时，将发生死锁。

图 3-9 中， P_2 进程从缓冲区取产品前，先执行 $P(S_2)$ ，由于 $S_2=-1$ 故 P_2 等待； P_1 进程将产品送到缓冲区后，执行 $P(S_1)$ ，由于 $S_1=-1$ 故 P_1 等待。这样， P_1 、 P_2 进程都无法继续运行下去，导致系统死锁。

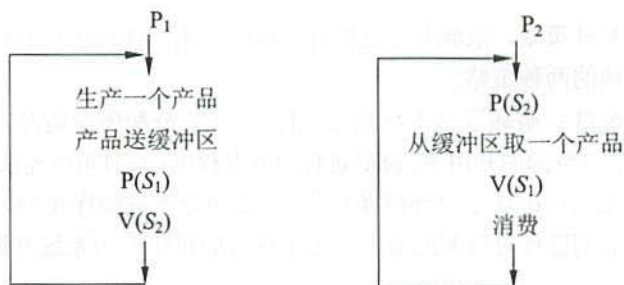


图 3-9 PV 操作引起的死锁



2. 死锁产生的原因及条件

可以看出,产生死锁的原因为竞争资源及进程推进顺序非法。当系统中有多个进程所共享的资源,不足以同时满足它们的需求时,引起它们对资源的竞争导致死锁。进程推进顺序非法,进程在运行的过程中,请求和释放资源的顺序不当,导致进程死锁。

产生死锁有4个必要条件,如下所述。

- (1) 互斥条件:进程对其所要求的资源进行排他性控制,即一次只允许一个进程使用。
- (2) 请求保持条件:零星的请求资源,即已获得部分资源又请求资源被阻塞。
- (3) 不可剥夺条件:进程已获得的资源在未使用完之前,不能被剥夺,只能在使用完时由自己释放。
- (4) 环路条件:当发生死锁时,在进程资源有向图中必构成环路,其中每个进程占有了下一个进程申请的一个或多个资源,如图3-10所示。

进程资源有向图由方框、圆圈和有向边3部分组成。其中方框表示资源,圆圈表示进程。请求资源:○→□箭头由进程指向资源;分配资源:□→○箭头由资源指向进程。

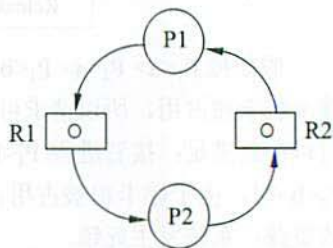


图3-10 进程资源有向图

3. 死锁的处理

死锁的处理策略主要有4种。

- 鸵鸟策略:即不理睬策略;
- 预防策略:破坏死锁的4个必要条件之一;
- 避免策略:精心地分配资源,动态地回避死锁;
- 检测与解除死锁:一旦发生死锁,系统不但能检测出,还能解除。

1) 死锁预防

死锁预防是采用某种策略,限制并发进程对资源的请求,使系统在任何时刻都不满足死锁的必要条件。预防死锁的两种策略:

- (1) 预先静态分配法:破坏了“不可剥夺条件”。预先分配所需资源,保证不等待资源。该方法的问题是降低了对资源的利用率,降低进程的并发程度;有时可能无法预先知道所需资源。
- (2) 资源有序分配法:破坏了“环路条件”。把资源分类按顺序排列,保证不形成环路。该方法存在的问题是限制进程对资源的请求;由于资源的排序占用系统开销。

2) 死锁避免

死锁预防是设法破坏产生死锁的4个必要条件之一,严格防止死锁的产生;死锁避免则不那么严格地限制产生死锁的必要条件。最著名的是Dijkstra提出的银行家算法。死锁避免算法

需要很大的系统开销。

(1) 安全状态：在避免死锁的方法中，允许进程动态地申请资源，系统在进行资源分配之前，先计算资源分配的安全性。若此次分配不会导致系统进入不安全状态，便将资源分配给进程；否则，进程等待。

所谓安全状态，是指系统能按某种顺序（如 $\langle P_1, P_2, \dots, P_n \rangle$ ）为每个进程分配其所需资源，直至最大需求，使每个进程都可顺序完成。通常称 $\langle P_1, P_2, \dots, P_n \rangle$ 序列为安全序列。若系统不存在这样一个安全序列，则称系统处于不安全状态。

虽然并非所有不安全状态都是死锁状态，但当系统进入不安全状态后，便有可能进入死锁状态；反之，只要系统处于安全状态，系统便可避免进入死锁状态。因此，避免死锁的实质在于如何使系统不进入不安全的状态。

(2) 银行家算法：该算法对于进程发出的每一个系统可以满足的资源请求命令加以检测，如果发现分配资源后，系统进入不安全状态，则不予分配；若分配资源后系统仍处于安全状态，则实施分配。与死锁预防策略相比提高了资源的利用率，但增加了系统开销。

3) 死锁检测

解决死锁的另一条途径是死锁检测方法，这种方法对资源的分配不加限制，即允许死锁产生。但系统定时地运行一个死锁检测程序，判断系统是否发生死锁，若检测到有死锁，则设法加以解除。

4) 死锁解除

死锁解除通常采用资源剥夺法和撤销进程法。

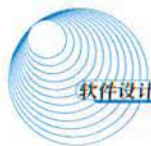
(1) 资源剥夺法：从一些进程那里强行剥夺足够数量的资源分配给死锁进程。

(2) 撤销进程法：根据某种策略逐个地撤销死锁进程，直到解除死锁为止。

3.2.7 线程

自 20 世纪 60 年代推出进程的概念后，在操作系统中都是以进程作为独立运行的基本单位。直到 20 世纪 80 年代中期，人们又推出了比进程更小的能独立运行的基本单位——线程。线程能提高系统内程序并发执行的程度，从而可进一步提高系统的吞吐量。目前不仅在新推出的操作系统中引入了线程的概念，而且在新推出的数据库管理系统和一些应用软件中，也纷纷引入了线程来改善系统的性能。

传统的进程有两个基本属性：可拥有资源的独立单位；可独立调度和分配的基本单位。由于在进程的创建、撤销和切换中，系统必须为之付出较大的时空开销。因此，在系统中所设置的进程数目不宜过多，进程切换的频率不宜太高，这就限制了并发程度的提高。引入线程后，将传统进程的两个基本属性分开，线程作为调度和分配的基本单位，进程作为独立分配资源的单位。用户可以通过创建线程来完成任务，以减少程序并发执行时付出的时空开销。



例如,在文件服务进程中,可设置多个服务线程。当一个线程受阻时,第二个线程可以继续运行,当第二个线程受阻时,第三个线程可以继续运行……从而显著地提高文件系统的服务质量及系统的吞吐量。

这样,对拥有资源的基本单位,不用频繁对其切换,从而进一步提高系统的并发程度。需要说明的是:线程是进程中的一个实体,是被系统独立分配和调度的基本单位。线程基本上不拥有资源,只拥有一点运行中必不可少的资源(如程序计数器、一组寄存器和栈)。它可与同属一个进程的其他线程共享进程所拥有的全部资源。

线程可创建另一个线程,同一个进程中的多个线程可并发执行。线程也具有就绪、运行、阻塞 3 种基本状态。线程具有许多传统进程所具有的特性,故称为轻型进程(Light-Weight Process);传统进程称之为重型进程(Heavy-Weight Process),它相当于只有一个线程的任务。

在引入了线程的操作系统中,通常一个进程都有若干个线程。可从 4 个方面来比较线程与进程。

- (1) 调度:将线程作为调度和分配的基本单位,进程作为资源拥有的基本单位。
- (2) 并发性:不仅进程之间可并发执行,而且在同一个进程中的多个线程之间也可并发执行。
- (3) 拥有资源:进程拥有资源的一个独立单位,线程不拥有系统资源,但可访问隶属进程的资源。
- (4) 系统开销:在创建或撤销进程时,由于系统都要为之分配和回收资源,导致系统的开销将明显地大于创建或撤销线程时的开销。

线程分为用户级线程(User-Level Threads)和内核支持线程(Kernel-Supported Threads)两类。用户级线程不依赖于内核,该类线程的创建、撤销和切换都不利用系统调用来实现;某些系统中实现的是内核支持线程,这种线程依赖于内核,即无论是在用户进程中的线程,还是在系统中的线程,它们的创建、撤销和切换都利用系统调用来实现,因为该类线程是与内核有关的;还有一些系统同时实现了两种类型的线程。

与线程不同的是,不论是系统进程还是用户进程,在进行切换时,都要依赖于内核中的进程调度。因此,不论是什么进程都是与内核有关的,是在内核支持下进行切换的。尽管程序和进程表面上看起来相似,但它们在本质上是不同的。程序是静态的指令序列,而进程是为执行该程序的线程而保留的资源集。

3.3 存储管理

存储器管理的对象是主存储器(简称主存或称内存)。存储器是计算机系统中的关键性资源,是存放各种信息的主要场所。特别是近几年来,系统软件、应用软件在功能及其所需存储

空间等方面都在急剧增加, 如何对存储器实施有效的管理, 不仅直接影响到存储器的利用率, 而且还对系统性能有重大的影响。虽然主存容量在不断扩大, 但主存仍是宝贵资源, 如何提高主存的利用率, 扩充主存, 对主存信息实现有效保护是存储器管理的主要任务。

3.3.1 基本概念

1. 存储器的结构

存储器的功能是保存数据, 存储器的发展方向是高速、大容量和小体积。例如, DRAM、SDRAM、SRAM 等体现在主存访问速度方面的提高; 硬盘技术在大容量方面的发展主要体现在接口标准、存储密度等。

存储组织的功能是在存储技术和 CPU 寻址技术许可的范围内组织合理的存储结构, 使得各层次的存储器都处于均衡的繁忙状态(如提高缓存命中率正好使主存读写保持繁忙), 其依据是访问速度是否匹配、容量要求和价格等。一般存储器的结构有“寄存器-主存-外存”结构和“寄存器-缓存-主存-外存”结构, 如图 3-11 所示。

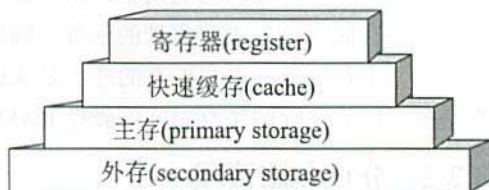


图 3-11 存储器的层次结构

(1) 虚拟地址: 对程序员来说, 数据的存放地址是由符号决定的, 故称符号名地址, 或者称为名地址, 而把源程序的地址空间叫做符号名地址空间或者名空间。它是从 0 号单元开始编址, 并顺序分配所有的符号名所对应的地址单元, 所以它不是主存中的真实地址, 故称为相对地址、程序地址、逻辑地址或称虚拟地址。

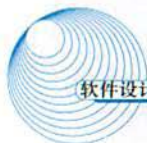
(2) 地址空间: 我们把程序中由符号名组成的空间称为地址空间。源程序经过汇编或编译后再经过链接编辑程序加工形成程序的装配模块, 即转换为相对地址编址的模块, 它是以 0 为基址顺序进行编址的。相对地址也称为逻辑地址或虚地址, 把程序中由相对地址组成的空间叫做逻辑地址空间。相对地址空间通过地址再定位机构转换到绝对地址空间, 绝对地址空间也叫物理地址空间。

(3) 存储空间: 简单说来, 逻辑地址空间(简称地址空间)是逻辑地址的集合, 物理地址空间(简称存储空间)是物理地址的集合。

2. 地址重定位

(1) 重定位: 程序的逻辑地址被转换成主存的物理地址的过程称为地址重定位。在可执行文件装入时需要解决可执行文件中地址(指令和数据)和主存地址的对应关系。由操作系统中的装入程序 Loader 和地址重定位机构来完成。地址重定位分为静态地址重定位和动态地址重定位。

(2) 静态重定位: 是指在程序装入主存时已经完成了逻辑地址到物理地址的变换, 在程序



的执行期间将不会再次发生变化。静态地址重定位的优点是无需硬件地址变换机构的支持,它只要求程序本身是可重定位的,它只对那些要修改的地址部分具有某种标识,由专门设计的程序来完成。在早期的操作系统中多数都采用这种方法。动态重定位的缺点是必须给作业分配一个连续的存储区域,在作业的执行期间不能扩充存储空间,也不能在主存中移动,多个作业也难以共享主存中的同一程序副本和数据。

(3) 动态重定位:是在程序运行期间完成逻辑地址到物理地址的变换。其实现机制要依赖硬件地址变化机构,如基地址寄存器 BR。动态地址再定位的优点是程序在执行期间可以换入和换出主存,可以解决主存紧张的问题;可以在主存中移动,把主存中的碎片集中起来,可以充分利用空间;不必给程序分配连续的主存空间,可以较好地利用较小的主存块;可以实现共享。

3. 存储管理的功能

(1) 主存储器的分配和回收:主分配的主要任务是为每一道程序分配主存空间。

(2) 提高主存储器的利用率:减少碎片(也称零头),允许多道程序动态共享主存。

(3) 存储保护:主存保护的任务是确保每道程序都在自己的主存空间运行,互不干扰。

(4) 主存扩充:主存扩充的任务是从逻辑上来扩充主存容量,使用户认为系统所拥有的主存空间远比其实际的主存空间(硬件 RAM)大得多。

3.3.2 分页存储管理

尽管分区管理方案是解决多道程序共享主存的可行方案,但是该方案的主要问题是用户程序必须装入连续的地址空间中,若无满足用户要求的连续空间时,需要进行分区靠拢操作,这是以耗费系统时间为代价的。为此引入了分页存储管理方案。

1. 纯分页存储管理

(1) 分页原理:是将一个进程的地址空间划分成若干个大小相等的区域,称为页。相应地,将主存空间划分成与页相同大小的若干个物理块,称为块或页框。在为进程分配主存时,将进程中若干页分别装入多个不相邻接的块中。

(2) 地址结构:分页系统的地址结构如图 3-12 所示,它由两部分组成。前一部分为页号 P;后一部分为偏移量 W,即页内地址。图中的地址长度为 32 位,其中 0~11 位为页内地址(每页的大小为 4KB),12~31 位为页号,所以允许地址空间的大小最多为 1MB 个页。



图 3-12 分页地址结构

(3) 页表：在将进程的每一页离散地分配到主存的多个物理块中后，系统应保证能在主存中找到每个页面所对应的物理块。为此，系统为每个进程建立了一张页面映射表，简称页表，如图 3-13 所示。每个页在页表中占一个表项，记录该页在主存中对应的物理块号。进程在执行时，通过查找页表，就可以找到每页所对应的物理块号。可见，页表的作用是实现从页号到物理块号的地址映射。

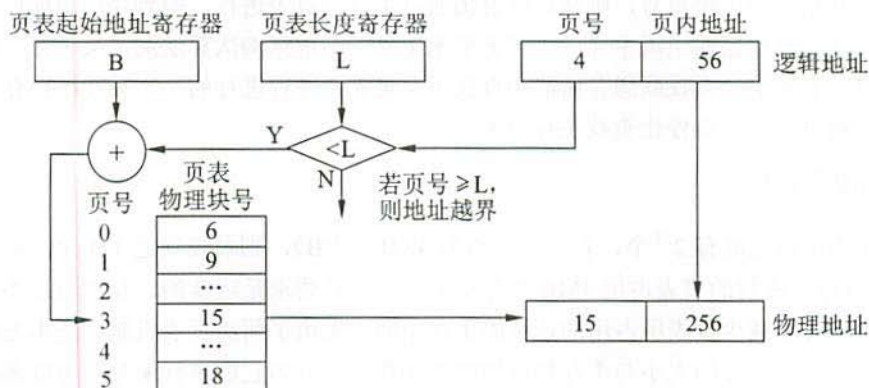


图 3-13 页式存储管理的地址映射

(4) 地址变换机构：基本任务是利用页表把用户程序中的逻辑地址变换成主存中的物理地址，实际上就要将用户程序中的页号变换成主存中的物理块号。为了实现地址变换功能，在系统中设置页表寄存器，用来存放页表的起始地址和页表的长度。在进程未执行时，每个进程对应的页表的起始地址和长度存放在进程的 PCB 中，当该进程被调度时，就将它们装入页表寄存器。在进行地址变换时，系统将页号与页表长度进行比较，如果页号大于等于页表寄存器中的页表长度 L（页号从 0 开始），则访问越界，产生越界中断。若未出现越界，则根据页表寄存器中的页表始址和页号计算出该页在页表项中的位置，得到该页的物理块号，将此物理块号装入物理地址寄存器中。与此同时，将有效地址（逻辑地址）寄存器中页内地址直接装入物理地址寄存器的块内地址字段中，这样便完成了从逻辑地址到物理地址的变换。

2. 快表

从地址映射的过程可以发现，页式存储管理至少需要两次访问主存。例如，第一次是访问页表，得到的是数据的物理地址；第二次是存取数据。若数据是间接地址，还需要再进行地址变换。为了提高访问主存的速度，可以在地址映射机构中增加一组高速寄存器，用来保存页表。这种方法需要大量的硬件开销，经济上是不可行的。另一种方法是在地址映射机构中增加一个小容量的联想存储器，联想存储器由一组高速存储器组成，称之为快表，用来保存当前访问频



率高的少数活动页的页号及相关信息。在快表中,除了逻辑页号、物理块号外,还要增加特征位,表示该行是否为空;增加访问位,表示最近该页是否被访问过,目的是为了淘汰那些长时间未访问过或用得很少或不用的那些页面。

联想存储器存放的只是当前进程最活跃的少数几页,因此,用户程序要访问数据时,根据该数据所在逻辑页号,在联想存储器中找出对应的物理页号,然后与页内地址拼接形成物理地址;若找不到相应的逻辑页号,则地址映射仍通过主存的页表进行,得到物理地址后,须将物理块号填入联想存储器的空闲单元中,若无空闲单元,则根据淘汰算法淘汰某一行,再填入新得到的页号。事实上,查找联想存储器和查找主存页表是并行进行的,一旦在联想存储器中找到相符的逻辑页号时,就停止查找主存页表。

3. 两级页表机制

80386 的逻辑地址有 2^{32} 个,若页面大小为 4KB (2^{12} B),则页表项达 1M 个,每个页表项占用 4B,故每个进程的页表占用 4MB 主存空间,并且还要求是连续的,显然这是不现实的。在 80386 中,为了减少页表所占用的连续的主存空间,采用了两级页表机制。基本方法是将页表进行分页,每个页面的大小与主存物理块的大小相同,并为它们进行编号,可以离散地将各个页面分别存放在不同的物理块中,为此需要建立一张页表,称为外层页表(页表目录),即第一级页表,其中的每个表目是存放某个页表的物理地址。第二级是页表,其中的每个表目所存放的是页的物理块号。

两级页表的逻辑地址结构和两级页表的地址变换机构如图 3-14 所示。

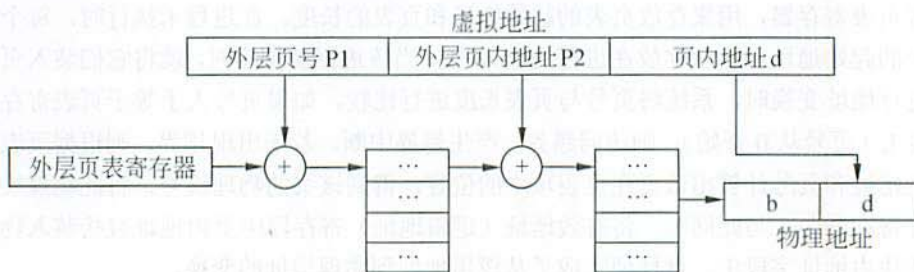


图 3-14 两级页表的地址变换机构

3.3.3 虚拟存储管理

1. 虚拟存储器的引入

1) 局部性原理

早在 1968 年 P. Denning 就指出过,程序在执行时将呈现出局部性规律,即在一段时间内,

程序的执行仅局限于某个部分；相应地，它所访问的存储空间也局限于某个区域内。程序的局限性表现在时间局限性和空间局限性两个方面。

(1) 时间局限性：如果程序中的某条指令一旦执行，则不久的将来该指令可能再次被执行；如果某个存储单元被访问，则不久以后该存储单元可能再次被访问。产生时间局限性的典型原因是在程序中存在大量的循环操作。

(2) 空间局限性：一旦程序访问了某个存储单元，则在不久的将来，其附近的存储单元也最有可能被访问。即程序在一段时间内所访问的地址，可能集中在一定的范围内，因为程序是按顺序执行的。

2) 虚拟存储器的定义

根据局部性原理，一个作业在运行之前，没有必要把作业全部装入主存，而仅将那些当前要运行的那部分页面或段先装入主存便可启动运行，其余部分暂时留在磁盘上。

程序在运行时如果它所访问的页（段）已调入主存，便可继续执行下去；但如果程序所要访问的页（段）尚未调入主存（称为缺页或缺段），此时程序应利用 OS 所提供的请求调页（段）功能，将它们调入主存，以使进程能继续执行下去。

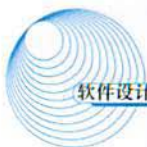
如果此时主存已满，无法再装入新的页（段），则还须再利用页（段）的置换功能，将主存中暂时不用的页（段）调出至磁盘上，腾出足够的主存空间后再将所要访问的页（段）调入主存，使程序继续执行下去。这样，便可使一个大的用户程序在较小的主存空间中运行；也可使主存中同时装入更多的进程并发执行。从用户角度看，该系统所具有的主存容量，将比实际主存容量大得多，人们把这样的存储器称为虚拟存储器。

虚拟存储器是具有请求调入功能和置换功能，能仅把作业的一部分装入主存便可运行作业的存储器系统，它能从逻辑上对主存容量进行扩充的一种虚拟的存储器系统。其逻辑容量由主存和外存容量之和以及 CPU 可寻址的范围来决定，其运行速度接近于主存速度。可见，虚拟存储技术是一种性能非常优越的存储器管理技术，故被广泛地应用于大、中、小型机器和微型机中。

3) 虚拟存储器的实现

(1) 请求分页系统：它是在分页系统的基础上，增加了请求调页功能和页面置换功能所形成的页式虚拟存储系统。它允许只装入若干页的用户程序和数据（而非全部程序），就可以启动运行，以后再通过调页功能和页面置换功能，陆续把将要使用的页面调入主存，同时把暂不运行的页面置换到外存上，置换时以页面为单位。

(2) 请求分段系统：它是在分段系统的基础上，增加了请求调段和分段置换功能所形成的段式虚拟存储系统。它允许只装入若干段（而非全部段）的用户程序和数据，就可以启动运行，以后再通过调段功能和置换功能将不运行的段调出，同时调入将要运行的段，置换是以段为单位。



(3) 请求段页式系统: 它是在段页式系统的基础上, 增加了请求调页和页面置换功能所形成的段页式虚拟存储系统。

4) 虚拟存储器的特征

(1) 离散性: 指在主存分配时采用离散的分配方式, 它是虚拟存储器的最基本的特征。

(2) 多次性: 指一个作业被分成多次调入主存运行, 即在作业运行时没有必要将其全部装入, 只需将当前要运行的那部分程序和数据装入主存即可。多次性是虚拟存储器最重要的特征。

(3) 对换性: 指允许在作业的运行过程中在主存和外存的对换区之间换进、换出。

(4) 虚拟性: 指能够从逻辑上扩充主存容量, 使用户所看到的主存容量远大于实际主存容量。

2. 请求分页中的硬件支持

请求分页是在纯分页系统的基础上, 增加了请求调页功能、页面置换功能所形成的页式虚拟存储系统, 是目前常用的一种虚拟存储器的方式。

1) 请求分页的页表机制

它是在纯分页的页表机制上形成的, 由于只将应用程序的一部分调入主存, 还有一部分仍在磁盘上, 故需在页表中再增加若干项, 如状态位、访问字段、辅存地址等供程序(数据)在换进、换出时参考。

2) 缺页中断机构

在请求分页系统中, 每当所要访问的页面不在主存时, 便要产生缺页中断, 请求 OS 将所缺页调入主存。与一般中断的主要区别在于: 缺页中断在指令执行期间产生和处理中断信号, 而一般中断在一条指令执行完后检查和处理中断信号。缺页中断返回到该指令的开始重新执行该指令, 而一般中断返回到该指令的下一条指令执行。一条指令在执行期间, 可能产生多次缺页中断。

3) 地址变换机构

请求分页系统中的地址变换机构, 是在分页系统的地址变换机构的基础上, 再为实现虚拟存储器而增加了某些功能形成的, 如产生和处理缺页中断、从主存中换出一页的功能, 等等。

3. 页面置换算法

在进程运行过程中, 如果发生缺页, 此时主存中又无空闲块时, 为了保证进程能正常运行, 就必须从主存中调出一页程序或数据送磁盘的对换区。但究竟将哪个页面调出, 需要根据一定的页面置换算法来确定。置换算法的好坏将直接影响系统的性能, 不适当的算法可能会导致系统发生“抖动”(Thrashing)。即刚被换出的页很快又被访问, 需重新调入, 导致系统频繁地更换页面, 以致一个进程在运行中把大部分时间花费在完成页面置换的工作上, 这种现象称之为

系统发生了“抖动”(也称颠簸)。请求分页系统的核心问题是选择合适的页面置换算法。常用的页面置换算法如下所述。

1) 最佳(Optimal)置换算法

它是一种理想化的算法,性能最好,但在实际上难于实现。即选择那些永不使用的,或者是在最长时间内不再被访问的页面置换出去。但是要确定哪一个页面是未来最长时间内不再被访问的,是很难估计的,所以该算法通常用来评价其他算法。

【例 3.8】 假定系统为某进程分配了 3 个物理块,进程访问页面的顺序为: 0, 1, 6, 5, 7, 4, 7, 3, 5, 4, 7, 4, 5, 6, 5, 7, 6, 0, 7, 6, 如图 3-15 所示。进程运行时先将 0, 7, 6 三个页面装入主存。当进程访问页面 5 时,产生缺页中断,根据最佳置换算法,页面 0 将在第 18 次才被访问,是三页中将最久不被访问的页面,所以被淘汰。接着访问页面 7 时,发现已在主存中,而不会产生缺页中断,以此类推。从图可以看出,采用最佳置换算法,只发生了 6 次页面置换。

访问页面	0	7	6	5	7	4	7	3	5	4	7	4	5	6	5	7	6	0	7	6
物理块	0	0	0	5	5	5	5	5	5	5	5	5	5	5	5	5	5	0	0	0
		7	7	7	7	7	7	3	3	3	7	7	7	7	7	7	7	7	7	7
			6	6	6	4	4	4	4	4	4	4	4	6	6	6	6	6	6	6
缺页	×	×	×	×		×		×			×			×				×		

图 3-15 最佳置换算法

2) 先进先出(FIFO)置换算法

该算法总是淘汰最先进入主存的页面,即选择在主存中驻留时间最久的页面予以淘汰。该算法实现简单,只需把一个进程已调入主存的页面按先后次序链接成一个队列,并设置一个指针即可。它是一种最直观、性能最差的算法,它有贝来迪(Belady)异常现象:对页面访问序列 ABCDABEABCDE,当分配的物理块从 3 块增加到 4 块时,缺页次数反而增加。

【例 3.9】 假定系统中某进程访问页面的顺序为如例 3.8 所示,利用 FIFO 算法对上例进行页面置换的结果如图 3-16 所示。

访问页面	0	7	6	5	7	4	7	3	5	4	7	4	5	6	5	7	6	0	7	6
物理块	0	0	6	5	7	4	4	3	5	3	7	4	5	6	6	7	7	0	0	0
		7	0	6	5	7	7	4	3	5	3	7	4	5	5	6	6	7	7	7
			7	0	6	5	5	7	4	4	5	3	7	4	4	5	5	6	6	6
缺页	×	×	×	×	×	×		×	×		×	×	×	×		×		×		

图 3-16 先进先出 FIFO 算法



从图中可以看出,共发生了 11 次面置换,缺页次数 14 次。

3) 最近最久未使用置换算法 LRU (Least Recently Used)

该算法是选择最近最久未使用的页面予以淘汰,系统在每个页面设置一个访问字段,用以记录这个页面自上次被访问以来所经历的时间 T ,当要淘汰一个页面时,选择 T 最大的页面。但在实现时需要硬件的支持(寄存器或栈)。

【例 3.10】假定系统中某进程访问页面的顺序如例 3-8 所示,利用 LRU 算法对上例进行页面置换的结果如图 3-17 所示。

访问页面	0	7	6	5	7	4	7	3	5	4	7	4	5	6	5	7	6	0	7	6
物理块	0	0	6	5	7	4	7	3	5	4	7	4	5	6	5	7	6	0	7	6
		7	0	6	5	7	4	7	3	5	4	7	4	5	6	5	7	6	0	7
			7	0	6	5	5	4	7	3	5	5	7	4	4	6	5	7	6	0
缺页	×	×	×	×	×	×		×	×	×	×			×		×		×		

图 3-17 最近最久未使用 LRU 算法

从图中可以看出,共发生了 10 次面置换,缺页次数 12 次。

4) 最近未用置换算法 NUR (Not Used Recently)

将最近一段时间未引用过的页面换出。这是一种 LRU 的近似算法。该算法为每个页面设置一位访问位,将主存中的所有页面都通过链接指针链成一个循环队列。当某页被访问时,其访问位置“1”。在选择一页淘汰时,检查其访问位,如果是“0”,则选择该页换出;若为“1”,则重新置为“0”,暂不换出该页,在循环队列中检查下一个页面,直到访问位为“0”的页面为止。由于该算法只有一位访问位,只能用它表示该页是否已经使用过,而置换时是将未使用过的页面换出去,所以把该算法称为最近未用算法。

4. 工作集

程序在运行中所产生的缺页情况,会影响程序的运行速度及系统性能,而缺页率的高低又将是与每个进程所占用的物理块数目有关。究竟应该为每个进程分配多少个物理块,才能把缺页率保持在一个合理的水平上。否则会因为进程频繁地从辅存请求页面而出现“颠簸”(也称抖动)现象。所谓工作集是指在某段时间间隔(Δ)里,进程实际要访问的页面的集合。工作集的理论是在 1968 年由 Denning 提出来的,他认为,虽然程序只需有少量的几页在主存就可以运行,但为了使程序能够有效地运行,较少地产生缺页,就必须使程序的工作集驻留在主存中。把某进程在时间 t 的工作集记为 $w(t, \Delta)$,把变量 Δ 称为工作集“窗口尺寸”(Windows Size)。正确选择工作集窗口(Δ)的大小,对存储器的有效利用和系统吞吐量的提高,都将产生重要

的影响。

程序在运行时对页面的访问是不均匀的,即往往在某段时间内的访问仅局限于较少的若干个页面,如果能够预知程序在某段时间间隔内要访问哪些页面,并能将它们提前调入主存,将会大大地降低缺页率,从而减少置换工作,提高 CPU 的利用率。当每个工作集都已达到最小值时,虚存管理程序跟踪进程的缺页数量,根据主存中自由页面数量可以适当增加其工作集的大小。

3.4 设备管理

设备管理是操作系统中最繁杂而且与硬件紧密相关的部分。设备管理不但要管理实际 I/O 操作的设备(如磁盘机、打印机),还要管理诸如设备控制器、DMA 控制器、中断控制器、I/O 处理机(通道)等支持设备。设备管理包括各种设备分配、缓冲区管理和实际物理 I/O 设备操作,通过管理达到提高设备利用率和方便用户的目的。

3.4.1 设备管理概述

对不同的计算机系统,使用大量的不同类型的设备,这些设备的特点也完全不同,因此设备管理是操作系统设计中最杂乱无序的领域。

设备是计算机系统与外界交互的工具,具体负责计算机与外部的输入/输出工作,所以常称为外部设备(简称外设)。在计算机系统中将负责管理和 I/O 机构称为 I/O 系统。因此,I/O 系统由设备、控制器、通道(具有通道的计算机系统)、总线和 I/O 软件组成。

1. 设备的分类

现代计算机系统都配有各种各样的设备,如打印机、显示器、绘图仪、扫描仪、键盘、鼠标等。设备可以有各种不同的分类方式。

1) 按数据组织分类

(1) 块设备(Block Device):指以数据块为单位来组织和传送数据信息的设备。这类设备用于存储信息,有磁盘和磁带等。它属于有结构设备。典型的块设备是磁盘,每个盘块的大小为 512B~4KB,磁盘设备的基本特征是:传输速率较高,通常每秒钟为几兆位;它是可寻址的,即可随机地读/写任意一块;磁盘设备的 I/O 采用 DMA 方式。

(2) 字符设备(Character Device):指以单个字符为单位来传送数据信息的设备。这类设备一般用于数据的输入和输出,有交互式终端、打印机等。它属于无结构设备。字符设备的基本特征是:传输速率较低;不可寻址,即不能指定输入时的源地址或输出时的目标地址;字符设备的 I/O 常采用中断驱动方式。



2) 从资源分配角度分类

(1) 独占设备: 指在一段时间内只允许一个用户(进程)访问的设备, 大多数低速的 I/O 设备, 如用户终端、打印机等属于这类设备。因为独占设备属于临界资源, 所以多个并发进程必须互斥地进行访问。

(2) 共享设备: 指在一段时间内允许多个进程同时访问的设备。显然, 共享设备必须是可寻址的和可随机访问的设备。典型的共享设备是磁盘。共享设备不仅可以获得良好的设备利用率, 而且是实现文件系统和数据库系统的物质基础。

(3) 虚拟设备: 指通过虚拟技术将一台独占设备变换为若干台供多个用户(进程)共享的逻辑设备。一般可以利用假脱机技术(Spooling 技术)实现虚拟设备。Spooling 技术将在后续内容中介绍。

3) 按数据传输率分类

(1) 低速设备: 指传输速率为每秒钟几个字节到数百个字节的设备。典型的设备有键盘、鼠标、语音的输入等。

(2) 中速设备: 指传输速率在每秒钟数千个字节至数十千个字节的设备。典型的设备有行式打印机、激光打印机等。

(3) 高速设备: 指传输速率在数百千个字节至数兆字节的设备。典型的设备有磁带机、磁盘机、光盘机等。

4) 其他分类方法

按输入/输出对象可分为人机通信、机机通信设备。按是否可交互可分为非交互设备, 如机机通信设备、外存、卡带机等; 交互设备, 如终端。

I/O 设备的种类繁多, 从 OS 观点来看, 其重要的性能指标有数据传输速率、数据的传输单位、设备的共享属性等。

2. 设备管理的目标与任务

1) 设备管理的目标

设备管理的目标主要是如何提高设备的利用率, 为用户提供方便统一的界面。提高设备的利用率, 就是提高 CPU 与 I/O 设备之间的并行操作程度, 主要利用的技术有: 中断技术、DMA 技术、通道技术、缓冲技术。其次是为用户提供方便、统一的界面。所谓方便是指用户能独立于具体设备的复杂物理特性之外而方便地使用设备。所谓统一, 是指对不同的设备尽量使用统一的操作方式, 例如各种字符设备用一种 I/O 操作方式。这就要求用户操作的是简便的逻辑设备, 而具体的 I/O 物理设备由操作系统去实现, 这种性能常常被称为设备的独立性。

2) 设备管理的任务

设备管理的任务是保证在多道程序环境下, 当多个进程竞争使用设备时, 按一定策略分配

和管理各种设备,控制设备的各种操作,完成 I/O 设备与主存之间的数据交换。设备管理的主要功能如下所述。

(1) 动态地掌握并记录设备的状态。在设置有通道的系统中,还应掌握通道、控制器的使用状态。

(2) 设备分配和释放:设备管理程序按照一定的策略把某一个 I/O 设备及其相应的设备控制器和通道分配给某一用户(进程),以保证在 I/O 设备和 CPU 之间有传输信息的通路。将未分配到设备(包括控制器、通道)的进程,插入等待队列。

(3) 缓冲区管理:为了解决 CPU 与 I/O 之间速度不匹配的矛盾,在它们之间配置了缓冲区。这样设备管理程序要负责管理缓冲区的建立、分配和释放。

(4) 实现物理 I/O 设备的操作:对于具有通道的计算机系统,设备管理程序根据用户提出的 I/O 请求,生成相应的通道程序并提交给通道,然后用专门的通道指令启动通道,对指定的设备进行 I/O 操作,并能响应通道的中断请求。对于未设置通道的系统,设备管理程序直接驱动设备进行 I/O 操作。

(5) 提供设备使用的用户接口:包括命令接口和编程接口。

(6) 设备的访问和控制:包括并发访问和差错处理。

(7) I/O 缓冲和调度:目的是提高 I/O 访问效率。

3.4.2 I/O 软件

设备管理软件的设计水平决定了设备管理的效率。I/O 设备管理软件的结构其基本思想是分层构造,也就是说把设备管理软件组织成为一系列的层次。其中低层与硬件相关,它把硬件与较高层次的软件隔离开来。而最高层的软件则向应用提供一个友好的、清晰而统一的接口。

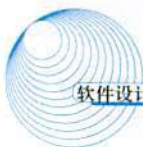
设计 I/O 软件的主要目标是设备独立性和统一命名。I/O 软件独立于设备就可以提高设备管理软件的设计效率。当输入/输出设备更新时,没有必要重新编写全部设备驱动程序。在实际应用中我们也可以看到,在常用操作系统中,只要安装了相对应的设备驱动程序,就可以很方便地安装好新的输入/输出设备。甚至不必重新编译就能将设备管理程序移到别处执行。

I/O 设备管理软件一般分为 4 层:分别是中断处理程序、设备驱动程序、与设备无关的系统软件 and 用户级软件。至于一些具体分层时细节上的处理是依赖于系统的,没有严格的划分;只要有利于设备独立这一目标,可以为了提高效率而设计不同的层次结构。

1. 中断处理程序

一旦 CPU 响应中断,转入中断处理程序,系统就开始进行中断处理。对中断处理过程如下:

(1) CPU 检查响应中断的条件是否满足。CPU 响应中断的条件是:有来自于中断源的中断请求、CPU 允许中断。如果中断响应条件不满足,则中断处理无法进行。



- (2) 如果 CPU 响应中断, 则 CPU 关中断, 使其进入不可再次响应中断的状态。
- (3) 保存被中断进程的现场。为了在中断处理结束后能使进程正确地返回到被中断点, 系统必须保存当前处理状态字 PSW 和程序计数器 PC 等的信息。这些信息通常保存在特定堆栈或硬件寄存器中。
- (4) 分析中断原因, 调用中断处理子程序。在多个中断请求同时发生时, 处理优先级最高的中断源发出的中断请求。在系统中, 为了处理上的方便, 通常都是针对不同的中断源编制有不同的中断处理子程序(陷阱处理子程序)。这些子程序的入口地址(或陷阱指令的入口地址)存放在主存的特定单元中。需要说明的是, 不同的中断源也对应着不同的处理器状态字 PSW。这些不同的 PSW 被放在相应的主存单元中, 与中断处理子程序入口地址一起构成中断向量。显然, 根据中断或陷阱的种类, 系统可由中断向量表中迅速地找到该中断响应的优先级、中断处理子程序(或陷阱指令)的入口地址和对应的 PSW。
- (5) 执行中断处理子程序。在有些系统中的陷阱是通过陷阱指令向当前执行进程发出软中断信号后, 调用对应的处理子程序执行。
- (6) 退出中断, 恢复被中断进程的现场或调度新进程占据 CPU。
- (7) 开中断, CPU 继续执行。

2. 设备驱动程序

设备驱动程序是直接同硬件打交道的软件模块。一般而言, 设备驱动程序的任务是接受来自与设备无关的上层软件的抽象请求, 进行与设备相关的处理。

设备驱动程序的最突出的特点是, 它与 I/O 设备的硬件结构密切联系。设备驱动程序中基本上是依赖于设备的代码。设备驱动程序是操作系统底层中唯一知道各种输入/输出设备的控制器细节及其用途的部分。例如, 只有磁盘驱动程序具体了解磁盘的区段、磁道、柱面、磁头、磁臂的运动、交错访问系数、马达驱动器、磁头定位次数, 以及所有保证磁盘正常工作的机制, 其他软件根本不过问这些硬件操作的细节。

不同的操作系统中, 对设备驱动程序的结构的要求是不同的。一般而言, 在操作系统的相关文档中, 都有对设备驱动程序结构方面的统一要求。可见对于某一类设备而言, 是采用通用的设备驱动程序, 还是采用专用的设备驱动程序, 取决于用户在这台输入/输出设备上追求的目标。如果把设备安装的便利性放在第一位, 那么建议考虑使用该类设备的通用驱动程序; 如果优先考虑设备的运行效率, 那么当然应该使用专门为这台设备编写的驱动程序。

3. 与设备无关的系统软件

除了一些 I/O 软件与设备相关之外, 大部分软件是与设备无关的。至于设备驱动程序与设备无关的软件之间的界限如何划分, 则随操作系统的不同而不同。具体划分原则取决于系统的

设计者怎样权衡系统与设备的独立性、驱动程序的运行效率等诸多因素。对于一些按照设备独立方式实现的功能,出于效率和其他方面的考虑,也可以由设备驱动程序实现。图 3-18 给出了常见的设备无关软件层实现的一些功能。

一般而言,所有设备都需要的 I/O 功能可以在与设备独立的软件中实现。这类软件面向应用层并提供一个统一的接口。

例如在 UNIX 系统中,像/dev/tty00 这样的设备名,唯一确定了一个特殊文件的 i 节点,这个 i 节点包含了主设备号和从设备号。主设备号用于寻找对应的设备驱动程序,而从设备号提供了设备驱动程序的有关参数,用来确定要读写的具体设备。

设备驱动程序的统一接口
设备命名
设备保护
提供一个与设备无关的逻辑块
缓冲
存储设备的块分配
独占设备的分配和释放
错误处理

图 3-18 与设备无关 I/O 软件的功能

4. 用户层 I/O 软件

通常大部分 I/O 软件都包含在操作系统中,但是用户程序仍有一小部分是与库函数连接在一起的,甚至还有在内核之外运行的程序。通常的系统调用,包括 I/O 系统调用,是由库函数实现。例如,一个用 C 语言编写的程序可含有如下的系统调用:

```
count=write(fd, buffer, nbytes);
```

在这个程序运行期间,该程序将与库函数 write 连接在一起,并包含在运行时的二进制程序代码中。显然,所有这些库函数是设备管理 I/O 系统的组成部分。通常这些库函数所做的工作主要是把系统调用时所用的参数放在合适的位置,由其他的 I/O 过程去实现真正的操作。在这里,输入/输出的格式是由库函数完成的。标准的 I/O 库包含了许多涉及 I/O 的过程,它们都是作为用户程序的一部分运行的。

例如,C 语言中的 Printf 以一个格式串和可能的一些变量作为输入,构造一个 ASCII 字符串,然后调用 WRITE 这个系统调用输出这个串。

但是,并非所有的用户层 I/O 软件都是由库函数组成的。Spooling(假脱机)系统是另一种重要的实现方法。Spooling 系统是多道程序设计系统中,模拟独占 I/O 设备完成假脱机的一种 I/O 技术。假设有一种典型的假脱机设备——行式打印机,一个进程打开了它,然后很长时间不使用,这样就导致了其他进程都无法使用这台打印机打印。

解决方法是创建一个特殊进程,称为守护(daemon)进程,以及一个特殊目录,称为 spooling 目录。当一个进程要打印一个文件时,首先要生成打印的整个文件,将其放在 spooling 目录下。然后由守护进程完成该目录下文件的打印工作,该进程是唯一一个拥有使用打印机特殊文件权



限的进程。而且,通过保护特殊文件以防止用户直接使用,可以解决进程空占打印机的问题。

需要指出的是,Spooling 技术不仅仅只适用于打印机这类输入/输出设备,还可应用到其他一些情况。图 3-19 总结了 I/O 软件的所有层次及每一层的主要功能。



图 3-19 I/O 系统的层次结构与每层的主要功能

图中的箭头给出了 I/O 部分的控制流。这里我们举一个读硬盘文件的例子。当用户程序试图读一个硬盘文件时,需要通过操作系统实现这一操作。与设备无关软件检查高速缓存中是否有要读的数据块。若没有,则调用设备驱动程序,向 I/O 硬件发出一个请求。然后,用户进程阻塞并等待磁盘操作的完成。当磁盘操作完成时,硬件产生一个中断,转入中断处理程序。中断处理程序检查中断的原因,认识到这是磁盘读取操作已经完成,于是唤醒用户进程取回从磁盘读取的信息,从而结束了此次 I/O 请求。用户进程在得到了所需的硬盘文件内容之后,继续运行。

3.4.3 通道、DMA 与缓冲技术

1. 通道

引入通道的目的是使数据的传输独立于 CPU,使 CPU 从繁琐的 I/O 工作中解脱出来。设置通道后,CPU 只需向通道发出 I/O 命令,通道收到命令后,从主存中取出本次 I/O 要执行的通道程序,并执行,仅当通道完成了 I/O 任务后才向 CPU 发出中断信号。

根据信息交换方式的不同将通道分为 3 类:

(1) 字节多路通道(Byte Multiplexor Channel):通常都含有许多非分配型子通道,每一个子通道连接一台 I/O 设备。主通道采用时间片轮转法,轮流地为各个子通道服务。只要字节多路通道扫描子通道的速率足够快,而连接到子通道的设备速率不太高时,便不会丢失信息。因此这些子通道连接的是慢速外围设备,如纸带输入机、纸带输出机、卡片输入机、卡片输出机、行式打印机等设备。

(2) 数组选择通道(Block Selector Channel):是由于字节多路通道不适于连接高速设备,所以引入数组选择通道。这种通道的传输速率高,可以连接多台高速设备,但由于该通道仅含

有一个可分配型通道, 因此, 在某一段时间内只能执行一个通道程序、为一台设备进行输入/输出。这样, 一旦某设备占用了通道, 并且会一直由它独占, 即使无数据传输而被闲置, 直到该设备自动释放通道。可见这种通道的利用率很低。

(3) 数组多路通道 (Block Multiplexor Channel): 是结合了数组选择通道传输速率高和字节多路通道能使各个子通道分时并行操作的优点。该通道中含有多个非分配型子通道, 因而该通道既具有很高的数据传输速率, 又能获得令人满意的通道利用率, 其数据传输是按数组方式进行的。

由于通道价格昂贵, 导致计算机系统通道数是有限的, 这往往会成为输入/输出的“瓶颈”问题。一个单通路的 I/O 系统中主存和设备之间只有一条通道。一旦某通道被设备占用, 即使另一通道空闲, 连接该通道的其他设备也只有等待。解决“瓶颈”问题的最有效的方法是增加设备到主机之间的通道, 使得主存和设备之间有两条以上的通道。如图 3-20 所示。

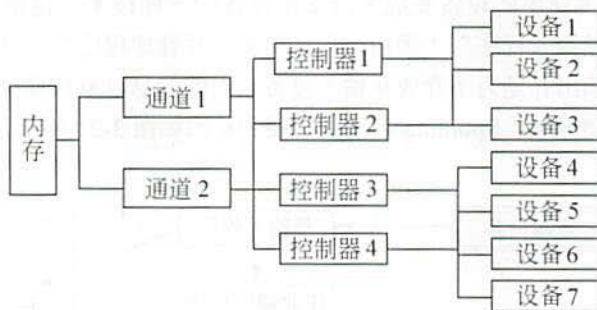


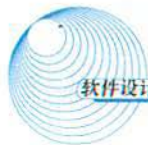
图 3-20 多通道的 I/O 系统

2. DMA 技术

直接主存取 (Direct Memory Access, DMA) 是指数据在主存与 I/O 设备间的直接成块传送, 即主存与 I/O 设备间传送一个数据块的过程中, 不需要 CPU 的任何干涉, 只需要 CPU 在过程开始启动 (即向设备发出“传送一块数据”的命令) 与过程结束 (CPU 通过轮询或中断得知过程是否结束和下次操作是否准备就绪) 时的处理, 实际操作由 DMA 硬件直接执行完成, CPU 在此传送过程中可做别的事情。例如, 在非 DMA 时, 打印 2048 字节, 至少需要执行 2048 次输出指令, 加上 2048 次中断处理的代价。而在 DMA 情况下, 若一次 DMA 可传送 512 个字节, 则只需要执行 4 次输出指令和处理 4 次打印机中断。若一次 DMA 可传送字节数大于等于 2048 个字节, 则只需要执行一次输出指令和处理一次打印机中断。

3. 缓冲技术

缓冲技术可提高外设利用率, 尽可能使外设处于忙状态。缓冲技术可以采用硬件缓冲和软



件缓冲。硬件缓冲是利用专门的硬件寄存器作为缓冲,软件缓冲是通过操作系统来管理的。引入缓冲的主要原因有以下几个方面:

- (1) 缓和 CPU 与 I/O 设备间速度不匹配的矛盾。
- (2) 减少对 CPU 的中断频率,放宽对中断响应时间的限制。
- (3) 提高 CPU 和 I/O 设备之间的并行性。

在所有的 I/O 设备与处理机(主存)之间,都使用了缓冲区来交换数据。所以操作系统必须组织和管理好这些缓冲区。缓冲可分为单缓冲、双缓冲、多缓冲和环形缓冲。

3.4.4 Spooling 技术

Spooling (Simultaneous Peripheral Operations On Line) 是外围设备联机操作的缩写,常简称为 Spooling 系统或假脱机系统。所谓 Spooling 技术实际上是用一类物理设备模拟另一类物理设备的技术,是使独占使用的设备变成多台虚拟设备的一种技术,也是一种速度匹配技术。Spooling 系统是由“预输入程序”、“缓输出程序”和“井管理程序”以及输入井和输出井组成的。其中,输入井和输出井是为了存放从输入设备输入的信息以及作业执行的结果,系统在辅助存储器上开辟的存储区域。Spooling 系统的组成和结构如图 3-21 所示。

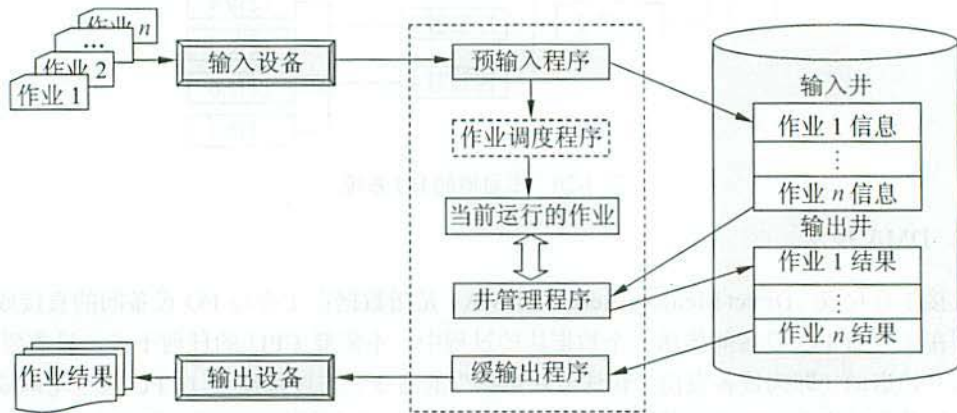


图 3-21 Spooling 系统的组成和结构

Spooling 系统的工作过程是操作系统初启后激活 Spooling 预输入程序使它处于捕获输入请求的状态,一旦有输入请求消息,Spooling 输入程序立即得到执行,把装在输入设备上的作业输入到硬盘的输入井中,并填写好作业表以便在作业执行中要求输入信息时,可以随时找到它们的存放位置。当作业需要输出数据时,可以先将数据送到输出井,当输出设备空闲时,由 Spooling 输出程序把硬盘上输出井的数据送到慢速的输出设备上。

Spooling 系统中拥有一张作业表用来登记进入系统的所有作业的作业名、状态、预输入表

位置等信息。每个用户作业拥有一张预输入表用来登记该作业的各个文件的情况,包括设备类、信息长度及存放位置等。输入井中的作业有4种状态:

- (1) 提交状态:作业的信息正从输入设备上预输入。
- (2) 后备状态:作业预输入结束但未被选中执行。
- (3) 执行状态:作业已被选中运行,运行过程中,它可从输入井中读取数据信息,也可向输出井写信息。
- (4) 完成状态:作业已经撤离,该作业的执行结果等待缓输出。

3.4.5 磁盘调度

磁盘是可被多个进程共享的设备。当有多个进程都请求访问磁盘时,为了保证信息的安全,系统每一时刻只允许一个进程启动磁盘进行 I/O 操作,其余的进程只能等待。因此,操作系统应采用一种适当的调度算法,以使各进程对磁盘的平均访问(主要是寻道)时间最小。磁盘调度分为移臂调度和旋转调度两类,并且是先进行移臂调度,然后再进行旋转调度。由于访问磁盘最耗时的是寻道时间,因此,磁盘调度的目标应是使磁盘的平均寻道时间最少。

1. 磁盘驱动调度

常用的磁盘调度算法如下:

(1) 先来先服务 FCFS (First-Come First-Served)。这是最简单的磁盘调度算法。它根据进程请求访问磁盘的先后次序进行调度。此算法的优点是公平、简单,且每个进程的请求都能依次得到处理,不会出现某进程的请求长期得不到满足的情况。但此算法由于未对寻道进行优化,致使平均寻道时间可能较长。

(2) 最短寻道时间优先 SSTF (Shortest Seek Time First)。该算法选择这样的进程,其要求访问的磁道与当前磁头所在的磁道距离最近,使得每次的寻道时间最短,但这种调度算法却不能保证平均寻道时间最短。

(3) 扫描算法 (SCAN)。扫描算法不仅考虑到欲访问的磁道与当前磁道的距离,更优先考虑的是磁头的当前移动方向。例如,当磁头正在由里向外移动时,SCAN 算法所选择的下一个访问对象应是其欲访问的磁道,既在当前磁道之外,又是距离最近的。这样由里向外地访问,直至再无更外面的磁道需要访问时才将磁臂换向,由外向里移动。这时,同样也是每次选择在当前磁道之内,且距离最近的进程来调度。这样,磁头逐步地向里移动,直至再无更里面的磁道需要访问。显然,这种方式避免了饥饿现象的出现。这种算法中,磁头移动的规律颇似电梯的运行,故又常称为电梯调度算法。

(4) 单向扫描调度算法 (CSCAN)。CSCAN 存在这样的问题:当磁头刚从里向外移动过某一磁道时,恰有一进程请求访问此磁道,这时该进程必须等待,待磁头从里向外,然后再从外



向里扫描完所有要访问的磁道后,才处理该进程的请求,致使该进程的请求被延迟。为了减少这种延迟,CSCAN 算法规定磁头作单向移动。

(5) N-Step-SCAN 算法。在 SSTF、SCAN 及 CSCAN 几种调度算法中,都可能出现磁臂停留在某位置不动的情况。例如,有一个或几个进程对某一磁道有着较高的访问频率,即它们反复地请求对某一磁道进行 I/O 访问,从而垄断了整个磁盘设备。在高密度盘上更容易出现这种情况,我们把这一现象称为磁臂“粘着”(Armstickiness)。

(6) FSCAN 算法。其实质上是 N 步 SCAN 算法的简化。它只将磁盘请求访问队列分成两个子队列。一是当前所有请求磁盘 I/O 的进程形成的队列,由磁盘调度按 SCAN 算法进行处理。另一个队列则是在扫描期间,新出现的所有请求磁盘 I/O 进程的队列,把它们插入另一个等待处理的请求队列。这样,所有的新请求都将被推迟到下一次扫描时处理。

2. 旋转调度算法

当移动臂定位后,有多个进程等待访问该柱面时,应当如何决定这些进程的访问顺序?这就是旋转调度要考虑的问题。显然系统应该选择延迟时间最短的进程对磁盘的扇区进行访问。当有若干等待进程请求访问磁盘上的信息时,旋转调度应考虑如下情况:

- (1) 进程请求访问的是同一磁道上的不同编号的扇区。
- (2) 进程请求访问的是不同磁道上的不同编号的扇区。
- (3) 进程请求访问的是不同磁道上具有相同编号的扇区。

对于(1)与(2)旋转调度总是让首先到达读写磁头位置下的扇区先进行传送操作;对于(3)旋转调度可以任选一个读写磁头位置下的扇区进行传送操作。

3.5 文件管理

随着计算机应用需求不断增长,快速、高效地处理大量的信息是计算机的首要任务之一,而这些信息通常是存储在大容量的外存储器上。然而在早期,用户要访问外存储器上的信息是很麻烦的,不仅要考虑信息在外存储器上的存放位置,而且要记住信息在外存储器的分布情况,构造 I/O 程序。稍不注意,就会破坏已存放的信息。特别是多道程序技术出现后,多个用户之间根本无法预料各个不同程序间的信息在外存储器上的是如何分配的。鉴于这些原因,引入文件系统专门负责管理外存储器上的信息,使用户可以“按名”高效、快速和方便地存储信息。

3.5.1 文件与文件系统

1. 文件

文件(File)是具有符号名的、在逻辑上具有完整意义的一组相关信息项的集合。例如,

一个源程序、一个目标程序、编译程序、一批待加工的数据、各种文档等都可以各自组成一个文件。

信息项是构成文件内容的基本单位,可以是一个字符,也可以是一个记录,记录可以等长,也可以不等长。一个文件包括文件体和文件说明。文件体是文件真实的内容;文件的说明是操作系统为了管理文件所用到的信息,包括文件名、文件内部标识、文件的类型、文件存储地址、文件的长度、访问权限、建立时间、访问时间等。

文件是一种抽象机制,它隐蔽了硬件和实现细节,提供了将信息保存在磁盘上而且便于以后读取的手段,使用户不必了解信息存储的方法、位置以及存储设备实际运作方式便可存取信息。因此,在文件管理中的一个非常关键的问题在于文件的命名。文件名是在进程创建文件时确定的,以后这个文件将独立于进程存在直到它被显式删除;其他进程要使用文件时必须显式指出该文件名,操作系统根据文件名对其进行控制和管理。不同的操作系统文件命名规则有所不同,即文件名字的格式和长度因系统而异。

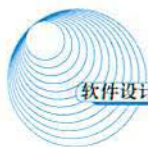
2. 文件系统

由于计算机系统处理的信息量越来越大,所以不可能将所有的信息保存到主存中。特别是在多用户系统中,既要保证各用户文件存放的位置不冲突,又要防止任一用户对外存储器(简称外存)空间占而不用;既要保证各用户文件在未经许可的情况下窃取、破坏,又要允许在特定的条件下多个用户共享某些文件。因此,需要设立一个公共的信息管理机制来负责统一管理外存和外存上的文件。

所谓文件管理系统,就是操作系统中实现文件统一管理的一组软件和相关数据的集合,专门负责管理和存取文件信息的软件机构,简称文件系统。文件系统的功能包括按名存取,即用户可以“按名存取”,而不是“按地址存取”;统一的用户接口,即在不同设备上提供同样的接口,方便用户操作和编程;并发访问和控制,即在多道程序系统中支持对文件的并发访问和控制;安全性控制,即在多用户系统中的不同用户对同一文件可有不同的访问权限;优化性能,即采用相关技术提高系统对文件的存储效率、检索和读写性能;差错恢复,即能够验证文件的正确性,并具有一定的差错恢复能力。

3. 文件的类型

- (1) 按文件性质和用途可将文件分为系统文件、库文件和用户文件。
- (2) 按信息保存期限分类可将文件分为临时文件、档案文件和永久文件。
- (3) 按文件的保护方式分类可将文件分为只读文件、读写文件、可执行文件和不保护文件。
- (4) UNIX 系统将文件分为普通文件、目录文件、设备文件(特殊文件)。
- (5) 目前常用的文件系统类型有 FAT、Vfat、NTFS、Ext2、HPFS 等。



文件分类的目的是对不同文件进行管理,提高系统效率;提高用户界面友好性。当然,根据文件的存取方法和物理结构不同还可以将文件分为不同的类型,这将在文件的逻辑结构和文件的物理结构中介绍。

3.5.2 文件的结构和组织

文件的结构是指文件的组织形式。从用户角度看到的文件组织形式称为文件的逻辑结构,文件系统的用户只要知道所需文件的文件名,就可存取文件中的信息,而无须知道这些文件究竟存放在什么地方。从实现的角度看文件在文件存储器上的存放方式,称为文件的物理结构。

1. 文件的逻辑结构

文件的逻辑结构可分为两大类:一是有结构的记录式文件,它是由一个以上的记录构成的文件,故又称为记录式文件;二是无结构的流式文件,它是由一串顺序字符流构成的文件。

1) 有结构的记录式文件

在记录式文件中,所有的记录通常都是描述一个实体集的,有着相同或不同数目的数据项,记录的长度可分为定长和不定长两类。

(1) 定长记录:它是指文件中所有记录的长度都是相同的。所有记录中的各个数据项,都处在记录中相同的位置,具有相同的顺序及相同的长度,文件的长度用记录数目表示。定长记录的特点是处理方便,开销小,是目前较常用的一种记录格式,被广泛用于数据处理中。

(2) 变长记录:它是指文件中各记录的长度不相同。这是因为:其一是一个记录中所包含的数据项数目可能不同。如书的著作者、论文中的关键词;其二是数据项本身的长度不定。例如,病历记录中的病因、病史;科技情报记录中的摘要等。但不论哪一种,在处理前每个记录的长度是可知的。

2) 无结构的流式文件

文件体为字节流,不划分记录。无结构的流式文件通常采用顺序访问方式,并且每次读写访问可以指定任意数据长度,其长度以字节为单位。对流式文件访问是利用读写指针指出下一个要访问的字符。可以把流式文件看作是记录式文件的一个特例。在 UNIX 系统中,所有的文件都被看作是流式文件。即使是有结构的文件,也被视为流式文件,系统不对文件进行格式处理。

2. 文件的物理结构

文件的物理结构是指文件的内部组织形式,即文件在物理存储设备上的存放方法。由于文件的物理结构决定了文件信息在文件存储设备上的存放位置,所以文件的逻辑块号到物理块号的转换也是由文件的物理结构决定的。根据用户和系统管理上的需要,可采用多种方法来组织

文件，下面介绍几种常见的文件物理结构。

1) 连续结构

连续结构也称顺序结构。它将逻辑上连续的文件信息（如记录）依次连续存放在连续编号的物理块上。只要知道文件的起始物理块号和文件的长度，就可以很方便地进行文件的存取。例如，文件 W.TXT 占用了 50、51、52、53 号物理块，系统只需将文件的起始块号 50 和文件的长度放在文件目录中该文件所对应的文件说明中即可，如图 3-22 所示。

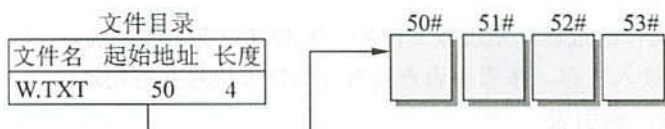


图 3-22 连续结构

连续结构的最佳应用场合是对文件的诸记录进行批量存取时，在所有逻辑文件中其存取效率是最高的。但在交互应用的场合，如果用户（程序）要求随机地查找或修改单个记录，此时系统需要逐个地查找诸记录，这样采用连续结构所表现出来的性能就可能很差，尤其是当文件较大时情况将更为严重。连续结构的另一个缺点是不便于记录的增加或删除操作。

2) 链接结构

链接结构也称串联结构，它是将逻辑上连续的文件信息（如记录）存放在不连续的物理块上，每个物理块设有一个指针指向下一个物理块。因此，只要知道文件的第一个物理块号，就可以按链指针查找整个文件。

例如，文件 W.TXT 占用了 60、86、92、103 号物理块，文件的起始块号 60 放在文件说明中，如图 3-23 所示。

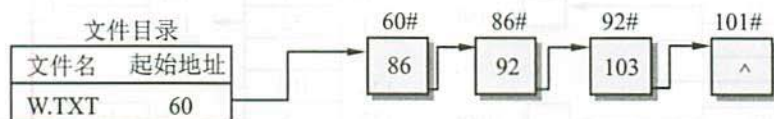


图 3-23 链接结构

3) 索引结构

采用索引结构将逻辑上连续的文件信息（如记录）存放在不连续的物理块中，系统为每个文件建立一张索引表。索引表记录了文件信息所在的逻辑块号对应的物理块号，并将索引表的起始地址放在文件对应的文件目录项中。

例如，文件 W.TXT 占用了 60、86、92、103 号物理块，文件索引表存放在 98 号物理块中，W.TXT 文件的文件目录项指向文件索引表，如图 3-24 所示。

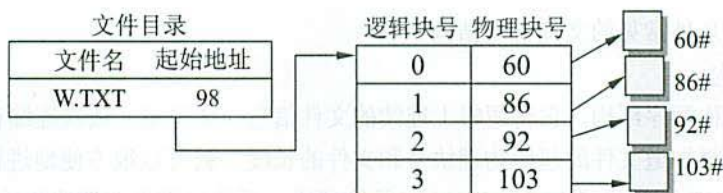
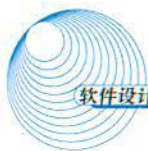


图 3-24 索引结构

访问 W.TXT 文件的过程：系统按文件名“W.TXT”查文件目录表，根据索引表的起始地址将 98# 索引表块读入主存，按索引表查找对应的物理块号并将物理块读入主存。

4) 多个物理块的索引表

索引表是在文件建立时由系统自动建立的，并与文件一起存放在同一文件卷上。一个文件的索引表根据文件大小的不同，占用物理块的个数不等，一般占一个或几个物理块。多个物理块的索引表可有两种组织方式：链接文件 and 多重索引方式。

(1) 链接文件方式：将多个索引表块按链接文件的方式串联起来。例如，每个索引表项占 4 个字节（可表示物理块号的范围从 $0 \sim 2^{32}$ ），若物理块的大小为 512B，则一个物理块可存放 127 个索引表项和一个链接字，如图 3-25 所示。

(2) 多重索引方式：具有多个物理块的索引表的另一种有效组织方式是多重索引方式。其结构如图 3-26 所示。

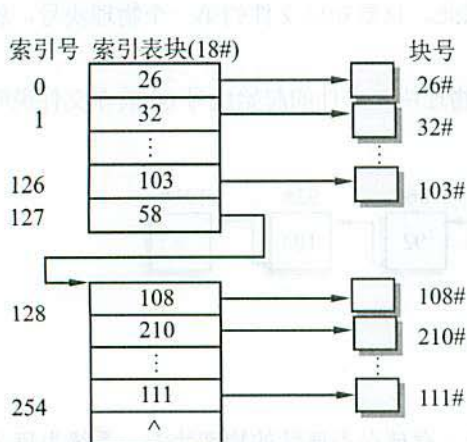


图 3-25 链接方式的索引结构

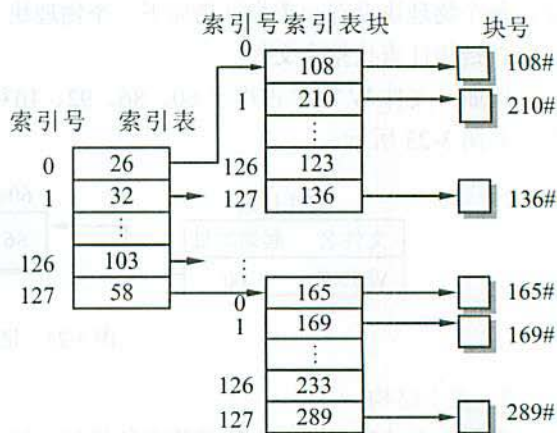


图 3-26 多重索引结构

5) UNIX 文件系统的索引结构

UNIX 文件系统采用的是三级索引结构，文件系统中 inode 是基本的构件，它表示文件系

统树型结构的节点。UNIX 有直接、一级间接、二级间接和三级间接 4 种寻址方式。

3.5.3 文件目录

为了实现“按名存取”，系统必须为每个文件设置用于描述和控制文件的数据结构，它至少应包括文件名和存放文件的物理地址，这个数据结构称为文件控制块 FCB，文件控制块的有序集合称为文件目录。换句话说，文件目录是由文件控制块组成的，专门用于文件的检索。文件控制块 FCB 也称为文件的说明或文件目录项（简称目录项）。

1. 文件控制块 FCB

文件控制块 FCB 中包含以下 3 类信息：基本信息类、存取控制信息类和使用信息类。

(1) 基本信息类：文件名、文件的物理地址、文件长度和文件块数等。

(2) 存取控制信息类：文件的存取权限，像 UNIX 用户分成文件主、同组用户和一般用户 3 类，这 3 类用户的读写执行（RWX）权限。

(3) 使用信息类：包括文件建立日期、最后一次修改日期、最后一次访问的日期；当前使用的信息，如打开文件的进程数，在文件上的等待队列等；目录文件，文件目录是由文件控制块组成的，专门用于文件的检索，文件目录可以存放在文件存储器固定位置也可以以文件的形式存放在磁盘上，我们将这种特殊的文件称之为目录文件。

2. 目录结构

文件目录结构的组织方式直接影响到文件的存取速度，关系到文件共享性和安全性，因此组织好文件的目录是设计文件系统的重要环节。常见的目录结构有 3 种：一级目录结构、二级目录结构和多级目录结构。

1) 一级目录结构

一级目录的整个目录组织是一个线性结构，在整个系统中只需建立一张目录表，系统为每个文件分配一个目录项（文件控制块）。一级目录结构简单，但缺点是查找速度慢，不允许重名和不便于实现文件共享等，因此它主要用在单用户环境中。

2) 二级目录结构

为了克服一级目录结构存在的缺点，引入了二级目录结构。二级目录结构是由主文件目录 MFD（Master File Directory）和用户目录 UFD（User File Directory）组成的。在主文件目录中，每个用户文件目录都占有一个目录项，其目录项中包括用户名和指向该用户目录文件的指针。用户目录是由用户所有文件的目录项组成。如图 3-27 所示。

二级目录结构基本上克服了单级目录的缺点，其优点如下：提高了检索目录的速度，较好

地解决了重名问题。采用二级目录结构也存在一些问题。该结构虽然能有效地将多个用户隔离开,这种隔离在各个用户之间完全无关时是一个优点;但当多个用户之间要相互合作去共同完成一个大任务时,且一用户又需去访问其他用户的文件时,这种隔离便成为一个缺点,因为这种隔离使诸用户之间不便于共享文件。

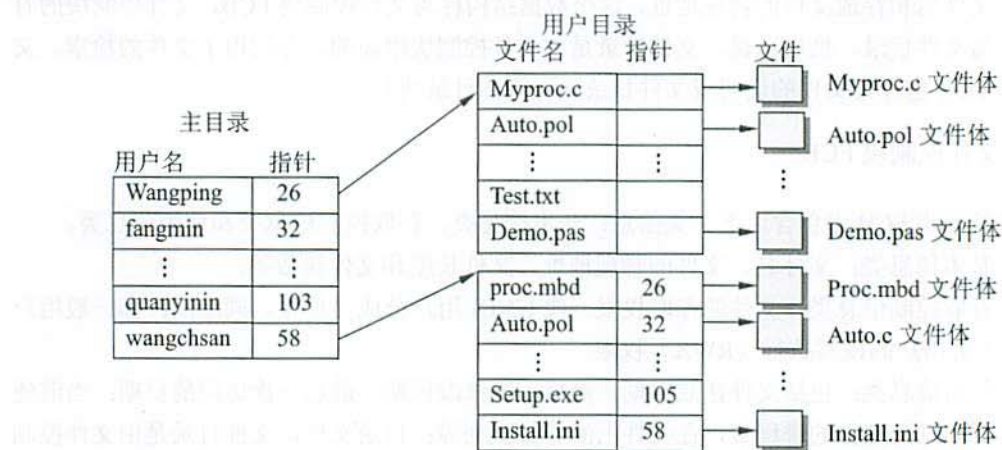


图 3-27 二级目录结构

3) 多级目录结构

为了解决以上问题,在多道程序设计系统中常采用多级目录结构,这种目录结构像一棵倒置的有根树,所以也称为树形目录结构。从树根向下,每一个节点是一个目录,叶节点是文件。MS-DOS 和 UNIX 等操作系统均采用多级目录结构。

采用多级目录结构的文件系统中,用户要访问一个文件,必须指出文件所在的路径名,路径名是从根目录开始到该文件的通路上所有各级目录名拼起来得到。各目录名之间,目录名与文件名之间需要用分隔符隔开。例如,在 MS-DOS 中分隔符为“\”,在 UNIX 中分隔符为“/”。绝对路径名(absolute path name)是指从根目录“/”开始的完整文件名,即它是由从根目录开始的所有目录名以及文件名构成的。

在多级目录中存取一个文件需要用文件全名,这就意味着允许用户在自己的目录中使用与其他用户文件相同的文件名,由于各用户使用不同的目录,虽二者使用了相同的文件名,但它们的文件全名仍不相同,这就解决了重名问题。采用多级目录结构提高了检索目录的速度,例如采用单级目录。查找一个文件最多需查遍系统目录文件中的所有文件目录项,平均也要查一半文件目录项。而多级目录查找一个文件最多只要查遍文件路径上根目录文件和子目录文件中

的目录项。

3.5.4 存取方法和存储空间的管理

1. 文件的存取方法

文件的存取方法是指读写文件存储器上的一个物理块的方法。通常分为顺序存取、随机存取。顺序存取是指对文件中的信息按顺序依次读写的方式；随机存取是指对文件中的信息可以按任意的次序随机地读写文件中的信息。

(1) 顺序存取法：在提供记录式文件结构的系统中，顺序存取法就是严格按物理记录排列的顺序依次读取。如果当前读取的是 R_i 记录，下一次要读取的记录自动地确定为 R_{i+1} 。在只提供无结构的流式文件中，顺序存取法是按读写的偏移 (offset) 从当前位置开始读写，每读完一段信息，读写偏移自动加上这段信息的长度，以便读下一段信息。

(2) 直接存取法：直接存取法允许用户随意存取文件中任意一个物理记录。对于无结构的流式文件，采用直接存取法，必须事先将读写偏移移动到待读写信息的位置上，然后再进行读写。

(3) 按键存取法：按键存取法是直接存取法的一种，它不是根据记录的编号或地址来存取文件中的记录，而是根据文件中各记录的某个数据项内容来存取记录的，这种数据项称之为“键”。因此，将这种存取法称之为按键存取法。

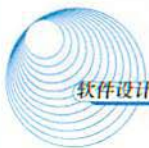
2. 文件存储空间的管理

外存具有大容量的存储空间，被多用户共享，用户执行程序经常要在磁盘上存储文件和删除文件，因此，文件系统必须对磁盘空间进行管理。外存空闲空间管理的数据结构通常称为磁盘分配表 (disk allocation table)。常用的空闲空间的管理方法有：空闲块表、位示图、空闲块链和成组链接法 4 种。

(1) 空闲区表：将外存空间上一个连续未分配区域称为“空闲区”。操作系统为磁盘外存上所有空闲区建立一张空闲表，每个表项对应一个空闲区，空闲表中包含序号、空闲区的第一块号、空闲块的块数等信息。它适用于连续文件结构。

(2) 位示图：这种方法是在外存上建立一张位示图 (bitmap)，记录文件存储器的使用情况。每一位对应文件存储器上的一个物理块，取值 0 和 1 分别表示空闲和占用。文件存储器上的物理块依次编号为：0、1、2、…。假如系统中字长为 32 位，那么在位示图中的第一个字对应文件存储器上的 0、1、2、…、31 号物理块；第二字对应文件存储器上的 32、33、34、…、63 号物理块；以下类推。

这种方法的主要特点是位示图的大小由磁盘空间的大小（物理块总数）决定，位示图的描



述能力强, 适合各种物理结构。

(3) 空闲块链: 每个空闲物理块中有指向下一个空闲物理块的指针, 所有空闲物理块构成一个链表, 链表的头指针放在文件存储器的特定位置上(如管理块中)。不需要磁盘分配表, 节省空间。每次申请空闲物理块只需根据链表的头指针取出第一个空闲物理块, 根据第一个空闲物理块的指针可找到第二个空闲物理块, 依次类推即可。

(4) 成组链接法: 在 UNIX 系统中, 将空闲块分成若干组, 每 100 个空闲块为一组, 每组的第一个空闲块登记了下一组空闲块的物理盘块号和空闲块总数, 假如一个组的第一个空闲块号等于 0 的话, 有特殊的含义, 意味着该组是最后一组, 即无下一组空闲块。

3.5.5 文件的使用

文件系统将用户的逻辑文件按一定的组织方式转换成物理文件存放到文件存储器上, 也就是说文件系统为每个文件与该文件在磁盘上的存放位置建立了对应关系。当用户使用文件时, 文件系统通过用户给出的文件名, 查出对应文件的存放位置, 读出文件的内容。在多用户环境下, 为了文件安全和保护起见, 操作系统为每个文件建立和维护关于文件主、访问权限等方面的信息。为此操作系统在操作级(命令级)和编程级(系统调用和函数)向用户提供文件的服务。

(1) 操作系统在操作级向用户提供的命令有: 目录管理类命令、文件操作类命令(如复制、删除和修改)、文件管理类命令(如设置文件权限)等。

(2) 操作系统在编程级向用户提供的系统调用主要有:

- 创建文件: 如 create (文件名, 参数表)。
- 撤销文件: 如 delete (文件名)。
- 打开文件: 如 open (文件名, 参数表)。
- 关闭文件: 如 close (文件名)。
- 读文件: 如 read (文件名, 参数表)。
- 写文件: 如 write (文件名, 参数表)。

3.5.6 文件的共享和保护

1. 文件的共享

文件共享是指不同用户进程使用同一文件, 它不仅是不同用户完成同一任务所必需的功能, 而且还可以节省大量的主存空间, 减少由于文件复制而增加的访问外存的次数。文件共享有多种形式, 采用文件名和文件说明分离的目录结构有利于实现文件共享。

常见的文件链接有硬链接和符号链接两种。

(1) 硬链接。文件的硬链接是指两个文件目录表目指向同一个索引节点的链接, 该链接也称基于索引节点的链接。换句话说硬链接是指不同文件名与同一个文件实体的链接。文件硬链接不利于文件主删除它拥有的文件, 因为文件主要删除它拥有的共享文件, 必须首先删除(关闭)所有的硬链接, 否则就会造成共享该文件的用户的目录表指针悬空。

例如, 在 UNIX 系统中的 “ln” 命令, 可以将多个文件名与一个文件体建立链接, 其格式为:

in 文件名 新文件名 或 in 文件名 目录名

ls 命令是放在/bin 子目录下, 我们可在/usr/bin 子目录下设置一个 DOS 兼容的命令 dir, 该命令实为执行 ls 命令。使用命令 “ln” 可给一已存在文件增加一个新文件名, 即文件链接数增加 1, 此种链接是不能跨越文件系统的。为了共享文件, 只是在两个不同子目录下取不同的文件名 ls 和 dir, 但它们具有相同的索引节点。UNIX 这种文件的结构称为树形带勾链的目录结构。在文件的索引节点中 di_nlink 变量表示链接到该索引节点上的链接数; 在用命令 “ls -l” 长列表显示时, 文件的第 2 项数据项表示链接数。

(2) 符号链接。符号链接建立的新的文件或目录与原来文件或目录的路径名映射。当访问一个符号链接时, 系统通过该映射找到原文件的路径, 并对其进行访问。

例如, 在 UNIX 系统中的 “ln -s” 命令建立符号链接。此时, 系统为共享的用户创建一个 link 类型的新文件, 将这新文件登录在该用户共享目录项中, 这个 link 型文件包含链接文件的路径名。该类文件在用 ls 命令长列表显示时, 文件链接数为 1。

采用符号链接可以跨越文件系统, 甚至可以通过计算机网络连接到世界上任何地方的机器中的文件, 此时只需提供该文件所在的地址以及在该机器中的文件路径。

符号链接的缺点: 其他用户读取符号链接的共享文件比读取硬链接的共享文件需要增加读盘操作的次数。因为其他用户去读符号链接的共享文件时, 系统中根据给定的文件路径名, 逐个分量地去查找目录, 通过多次读盘操作才能找到该文件的索引节点, 而用硬链接的共享文件的目录文件表目中已包括了共享文件的索引节点号。

2. 文件的保护

文件系统对文件的保护常采用存取控制方式进行, 所谓存取控制就是不同的用户对文件的访问规定不同的权限, 以防止文件被未经文件主同意的用户访问。

(1) 存取控制矩阵。理论上存取控制方法可用存取控制矩阵, 它是一个二维矩阵, 一维列出计算机的全部用户, 另一维列出系统中的全部文件, 矩阵中每个元素 A_{ij} 是表示第 i 个用户对第 j 个文件的存取权限。通常存取权限有可读、可写、可执行以及它们的组合, 如表 3-3 所示。

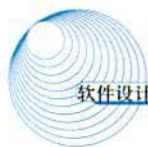


表 3-3 存取控制矩阵

文件 用户	ALPHA	BETA	REPORT	SQRT	×××
张 军	RWX	...	R-X	...	
李晓钢	R-X	...	RWX	R-X	...
王 伟	...	RWX	R-X	R-X	
赵 凌	RWX	
×××	

存取控制矩阵在概念上是简单清楚的,但实现上却有困难。当一个系统用户数和文件数很大时,二维矩阵要占很大的存储空间,验证过程也将耗费许多系统时间。

(2) 存取控制表。存取控制矩阵由于太大而往往无法实现。一个改进的办法是按用户对文件的访问权力的差别对用户进行分类,由于某一文件往往只与少数几个用户有关,所以这种分类方法可使存取控制表大为简化。UNIX 系统就是使用这种存取控制表方法。它把用户分成 3 类:文件主、同组用户和其他用户,每类用户的存取权限为可读、可写、可执行以及它们的组合。在用 ls 长列表显示时每组存取权限用 3 个字母 RWX 表示,如读、写和执行中那一样存取不允许则用“-”字符表示,用 ls -l 长列表显示 ls 文件如下:

```
-r-xr-xr-t 1 bin bin 43296 May 13 1997 /opt/K/SCO/Unix/5.0.4Eb/bin/ls
```

显示前 2~10 共 9 个字符表示文件的存取权限,每 3 个字符为一组,分别表示文件主、同组用户和其他用户的存取权限。由于存取控制表对每个文件按用户分类,所以该存取控制表可存放在每个文件的文件控制块中,对 UNIX 只需 9 位二进制来表示 3 类用户对文件的存取权限,该权限存在文件索引节点的 di_mode 中。

(3) 用户权限表。改进存取控制矩阵的另一种方法是以用户或用户组为单位将用户可存取的文件集中起来存入表中,这称为用户权限表,表中每个表目表示该用户对应文件的存取权限,这相当于存取控制矩阵一行的简化。

(4) 密码。在创建文件时,由用户提供一个密码,在文件存入磁盘时用该密码对文件内容加密。进行读取操作时,要对文件进行解密,只有知道密码的用户才能读取文件。

3.5.7 系统的安全与可靠性

1. 系统的安全

系统的安全涉及两类不同的问题:一类涉及到技术、管理、法律、道德和政治等问题;另一类涉及操作系统的安全机制。随着计算机应用范围扩大,在所有稍具规模的系统中,都从多

个级别上来保证系统的安全性。一般从系统级、用户级、目录级和文件级 4 个级别上对文件进行安全性管理。

(1) 系统级安全管理的主要任务是不允许未经授权的用户进入系统,从而也防止了他人非法使用系统中各类资源(包括文件)。系统级管理的主要措施有注册与登录。

(2) 用户级安全管理是通过对所有用户分类和对指定用户分配访问权。不同的用户对不同文件设置不同的存取权限来实现。例如,在 UNIX 系统中将用户分为文件主、组用户和其他用户。有的系统将用户分为超级用户、系统操作员和一般用户。

(3) 目录级安全管理是为了保护系统中各种目录而设计的,它与用户权限无关。为保证目录的安全规定只有系统核心才具有写目录的权利。

(4) 文件级安全管理是通过系统管理员或文件主对文件属性的设置来控制用户对文件的访问。通常可设置以下几种属性:只执行、隐含、只读、读/写、共享、系统。用户对文件的访问,将由用户访问权、目录访问权限及文件属性三者的权限所确定。或者说有效权限和文件属性的交集。例如对于只读文件,尽管用户的有效权限是读/写,但都不能对只读文件进行修改、更名和删除。对于一个非共享文件,将禁止在同一时间内由多个用户对它们进行访问。通过上述四级文件保护措施,可有效地对文件的保护。

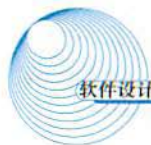
2. 文件系统的可靠性

文件系统的可靠性是指系统抵抗和预防各种物理性破坏和人为性破坏的能力。比起计算机的损坏,文件系统破坏往往后果更加严重。例如,将开水撒在键盘上引起的故障,尽管伤脑筋但毕竟可以修复;但如果文件系统破坏了在很多情况下是无法恢复的。特别是对于哪些程序文件、客户档案、市场计划或其他数据文件丢失的客户来说,这不亚于一场大的灾难。尽管文件系统无法防止设备和存储介质的物理损坏,但至少应能保护信息。

(1) 转储和恢复。文件系统中无论是硬件或软件都会发生损坏和错误。例如自然界的闪电、电压的突变、火灾和水灾等均可能引起软、硬件的破坏。为了使文件系统万无一失,应当采用相应的措施。最简单和常用的措施是通过转储操作,形成文件或文件系统的多个副本。这样一旦系统出现故障,利用转储的数据使得系统恢复成为可能。常用的转储方法有静态转储和动态转储、海量转储和增量转储。

(2) 日志文件。在计算机系统的工作的过程中,操作系统把用户对文件的插入、删除和修改的操作写入日志文件。一旦发生故障,操作系统恢复子系统利用日志文件来进行系统故障恢复,并可协助后备副本进行介质故障恢复。

(3) 文件系统的一致性。影响文件系统可靠性因素之一是文件系统的一致性问题。很多文件系统是先读取磁盘块到主存,在主存进行修改,修改完毕再写回磁盘。但如读取某磁盘块,修改后再将信息写回磁盘前系统崩溃,则文件系统就可能会出现不一致性状态。如果这些未被



写回的磁盘块是索引节点块、目录块或空闲块,那么后果是不堪设想的。通常解决方案是采用文件系统的一致性检查,一致性检查包括块的一致性和文件的一致性检查。

3.6 作业管理

作业是系统为完成一个用户的计算任务(或一次事务处理)所做的工作总和。例如,对用户编写的源程序,需要经过编译、连接装入以及执行等步骤得到结果,这其中的每一个步骤称之为作业步。操作系统中用来控制作业进入、执行和撤销的一组程序称之为作业管理程序。操作系统可以进一步为每个作业创建作业步进程,完成用户的工作。

3.6.1 作业管理和作业控制

1. 作业控制

用户作业可以采用脱机和联机两种控制方式控制作业运行。在脱机控制方式中,作业运行的过程是不需要人工干预的,因此,用户必须将自己想让计算机干什么的意图用作业控制语言(JCL)编写成作业说明书连同作业一起提交给计算机系统。在联机控制方式中,操作系统向用户提供了一组联机命令,用户可以通过终端键入命令将自己想让计算机干什么的意图告诉计算机,以控制作业的运行过程,因此整个作业的运行过程是需要人工干预。

作业由程序、数据和作业说明书3部分组成。

作业说明书包括作业基本情况、作业控制、作业资源要求的描述,它体现用户的控制意图。其中,作业基本情况包括用户名、作业名、编程语言、最大处理时间等;作业控制描述包括作业控制方式、作业步的操作顺序、作业执行出错处理;作业资源要求描述包括处理时间、优先级、主存空间、外设类型和数量、实用程序要求等。

2. 作业状态及转换

作业的状态分为:提交、后备、执行、完成4种。

(1) 提交:作业提交给计算机中心,通过输入设备送入计算机系统的过程时的状态称之为提交状态。

(2) 后备:作业通过 Spooling 系统输入到计算机系统的后备存储器(磁盘)中,随时等待作业调度程序调度时的状态。

(3) 执行:一旦作业被作业调度程序选中,为其分配了必要的资源,并为其建立相应的进程后,该作业便进入了执行状态。

(4) 完成:当作业正常结束或异常终止时,作业进入完成状态。此时有作业调度程序对该作业进行善后处理。如撤销作业的作业控制块,收回作业所占的系统资源,将作业的执行结果

形成输出文件放到输出井中, 由 Spooling 系统控制输出。

作业的状态及转换图 3-28 如下所示。

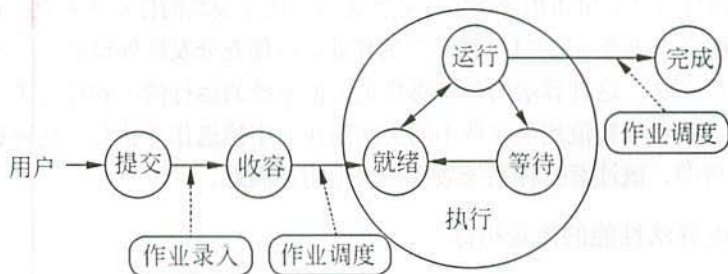


图 3-28 作业的状态及其转换

3. 作业控制块和作业后备队列

所谓作业控制块 JCB 是记录与该作业有关的各种信息的登记表。JCB 是作业存在的唯一标志, 包括用户名、作业名、状态标志等信息。

由于在输入井中有较多的后备作业, 为了便于作业调度程序调度, 通常将作业控制块排成一个或多个队列, 而这些队列称之为作业后备队列, 也就是说作业后备队列是由若干个 JCB 组成的。

3.6.2 作业调度

选择的调度算法需要考虑的因素包括与系统的整个设计目标一致、均衡使用系统资源, 以及平衡系统和用户的要求。对用户来说, 作业能“立即执行”往往难以做到, 但是应保证进入系统的作业在规定的截止时间内完成, 而且系统应设法缩短作业的平均周转时间。

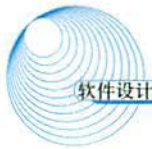
1. 作业调度算法

常用的作业调度算法:

- (1) 先来先服务: 按作业到达先后进行调度, 即启动等待时间最长的作业。
- (2) 短作业优先: 以要求运行时间长短进行调度, 即启动要求运行时间最短的作业。
- (3) 响应比高优先: 响应比高的作业优先启动。

定义响应比如下: $R_p = \frac{\text{作业响应时间}}{\text{作业执行时间}}$, 其中作业响应时间为作业进入系统后的等候时间与作

业的执行时间之和, 即 $R_p = 1 + \frac{\text{作业等待时间}}{\text{作业执行时间}}$ 。



响应比高者优先算法,在每次调度前都要计算所有被选作业(在作业后备队列中)的响应比,然后选择响应比最高的作业执行。该算法比较复杂,系统开销大。

(4) 优先级调度算法:可由用户指定作业优先级,优先级高的作业先启动。也可有系统根据作业要求的紧迫程度,或者照顾“I/O繁忙”的作业,以便充分发挥外设的效率等。

(5) 均衡调度算法:这种算法的基本思想是根据系统的运行情况和作业本身的特性对作业进行分类。作业调度程序轮流地从这些不同类别的作业中挑选作业执行。这种算法力求均衡地使用系统的各种资源,既注意发挥系统效率,又使用户满意。

2. 作业调度算法性能的衡量指标

在一个以批量处理为主的系统中,通常用平均周转时间或平均带权周转时间来衡量调度性能的优劣。假设作业 $J_i(i=1,2,\cdots,n)$ 的提交时间为 t_{si} , 执行时间为 t_{ri} , 作业完成时间为 t_{wi} , 则作业 J_i 的周转时间 T_i 和带权周转时间 W_i 分别定义为:

$$T_i = t_{wi} - t_{si} \quad (i=1,2,\cdots,n) \quad W_i = T_i / t_{ri} \quad (i=1,2,\cdots,n)$$

n 个作业的平均周转时间 T 和平均带权周转时间 W 分别定义为:

$$T = \frac{1}{n} \sum_{i=1}^n T_i, \quad W = \frac{1}{n} \sum_{i=1}^n W_i$$

站在用户的角度来说,总是希望自己的作业在提交后能立即执行,这意味着当等待时间为零时作业的周转时间最短,即 $T_i = t_{ri}$ 。但是作业的执行时间 t_{ri} 并不能直观地衡量出系统的性能,而带权周转时间 W_i 却能直观反应系统的调度性能。站在整个系统的角度来说,不可能满足每个用户的这种要求,而只能是系统的平均周转时间或平均带权周转时间最小。

3.6.3 用户界面

用户界面(User Interface)是计算机中实现用户与计算机通信的软件、硬件部分的总称。用户界面也称用户接口或人机界面。

用户界面的硬件部分包括用户向计算机输入数据或命令的输入装置及由计算机输出供用户观察或处理的输出装置。用户界面的软件部分包括用户与计算机相互通信的协议、约定、操纵命令及其处理软件。目前常用的输入/输出装置有键盘、鼠标、显示器、打印机等。常用的人机通信方法有命令语言、选项、表格填充及直接操纵等。从计算机用户界面发展过程来看,可分为若干阶段:

1. 控制面板式用户界面

这是计算机发展早期,用户通过控制台开关、按键或穿孔纸带向计算机送入命令或数据,而计算机通过指示灯及打印机输出运行情况或结果。这种界面的特点是人去适应现在看来是十

分笨拙的计算机。

2. 字符用户界面

字符用户界面是基于字符型的。用户通过键盘或其他输入设备输入字符,由显示器或打印机输出字符。字符用户界面的优点是功能强、灵活性好、屏幕开销少;缺点是操作步骤繁琐,学会操作也较费时。

3. 图形用户界面

随着文字、图形、声音、图像等多媒体技术的出现,使各种图形用户界面应运而生,用户既可使用传统的字符,也可使用图形、图像和声音同计算机进行交互,操作将更为自如,更加方便。现代界面的关键技术在超文本。超文本的“超”体现在它不仅是包括文本,还包括图像、音频、视频等多媒体信息,即将文本的概念扩充到超文本,超文本的最大特点是具有指向性。

4. 新一代用户界面

虚拟现实技术将用户界面发展推向一个新阶段:人将作为参与者,以自然的方式与计算机生成的虚拟环境进行通信。以用户为中心、自然、高效、高带宽、非精确、无地点限制等是新一代用户界面的特征。多媒体、多通道及智能化是新一代用户界面的技术支持。语音、自然语言、手势、头部跟踪、表情、视线跟踪等新的、更加自然的交互技术,将为用户提供更方便的输入技术。计算机将通过多种感知通道来理解用户的意图,实现用户的要求;计算机不仅以二维屏幕向用户输出,而且以真实感(立体视觉、听觉、嗅觉、触觉等)的计算机仿真环境向用户提供真实的体验。

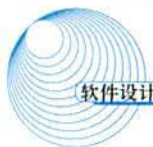
3.7 网络操作系统和嵌入式操作系统基础知识

3.7.1 网络操作系统

计算机网络系统除了硬件还需要有系统软件,二者结合构成计算机网络的基础平台。操作系统是最重要的系统软件。网络操作系统是网络用户和计算机网络之间的一个接口,它除了应具备通常操作系统应具备的基本功能外,还应有联网功能,支持网络体系结构和各种网络通信协议,提供网络互联功能,支持有效、可靠安全地数据传送。

1. 网络操作系统的特征

一个典型的网络操作系统的特征包括:



(1) 硬件独立性: 网络操作系统可以运行在不同的网络硬件上, 可以通过网桥或路由器与别的网络连接。

(2) 多用户支持: 应能同时支持多个用户对网络的访问, 应对信息资源提供完全的安全和保护功能。

(3) 支持网络实用程序及其管理功能: 如系统备份、安全管理、容错和性能控制。

(4) 多种客户端支持: 如 Windows NT 网络操作系统包括 OS/2、Windows98、UNIX 等多种客户端, 极大地方便了网络用户。

(5) 提供目录服务: 以单一逻辑的方式让用户访问位于世界范围内的所有网络服务和资源的技术。

(6) 支持多种增值服务: 如文件服务、打印服务、通信服务和数据库服务等。

2. 网络操作系统分类

(1) 集中模式: 集中式网络操作系统是由分时操作系统加上网络功能演变而来的, 系统的基本单元是由一台主机和若干台与主机相连的终端构成, 将多台主机连接起来形成了网络, 信息的处理和控制是集成的, UNIX 就是这类系统的典型例子。

(2) 客户/服务器模式: 是流行的网络工作模式。该种模式网络可分为服务器和客户。服务器是网络的控制中心, 其任务是向客户提供一种或多种服务; 服务器可有多种类型, 如提供文件/打印服务的文件服务器等。客户是用于本地处理和访问服务器的站点, 在客户中包含了本地处理软件和访问服务器上服务程序的软件接口。

(3) 对等模式 (Peer-to-Peer): 采用这种模式的操作系统网络中各个站点是对等的。它既可作为客户去访问其他站点, 又可作为服务器向其他站点提供服务, 在网络中既无服务处理中心, 也无控制中心, 或者说, 网络的服务和控制功能分布在各个站点上。可见该模式具有分布处理及分布控制的特征。

现代操作系统已把网络功能包含到操作系统的内核中, 作为操作系统核心功能的一个组成部分。微软公司的 Windows NT, AT & T 公司的 UNIX System V、Sun 公司的 SunOS、HP 公司的 HP/UX、IBM 公司的 AIX、Linux 等都已把 TCP/IP 网络功能包含在内核中。

Windows NT 的输入/输出 I/O 系统包含有 5 部分: 输入输出管理程序、文件系统、缓冲存储管理系统、设备驱动程序、网络驱动程序。

网络操作系统是整个网络的灵魂, 它决定了网络的功能, 并由此决定了不同网络的应用领域及方向。目前网络操作系统主要有 3 大阵营: UNIX、Windows NT 和 NetWare。各种网络操作系统具有不同的特点, 随着网络技术的发展, 新的网络操作系统还会不断出现, 用户可根据自己的需要进行选择, 而不要仅局限于其技术水平的高低。

3.7.2 嵌入式操作系统

在嵌入式系统中的操作系统，称为嵌入式操作系统。嵌入式操作系统是运行在嵌入式智能芯片环境中，对整个智能芯片以及它所操作、控制的各种部件装置等资源进行统一协调、调度、指挥和控制的系统软件。嵌入式系统广泛应用于各种工业控制系统、计算机外设、微波炉、洗衣机、冰箱等低端设备；也用在信息化家电、掌上电脑、机顶盒、WAP 手机、路由器等高端设备中。

1. 嵌入式操作系统的特点

一般而言，嵌入式操作系统不同于一般意义的计算机操作系统，它有占用空间小、执行效率高、方便进行个性化定制和软件要求固化存储等特点。嵌入式操作系统和其他嵌入式软件都具有如下特点：

(1) 微型化：由于硬件平台的局限性，如主存少、字长短、运行速度有限、能源少（用小型电池）、外部设备和控制对象千变万化等，因此，不论从性能还是从成本角度考虑，都不允许它占用很多资源。系统代码量少，应在保证应用功能的前提下，以微型化作为特点来设计嵌入式操作系统的结构与功能。

(2) 可定制：嵌入式操作系统的运行平台多种多样，应用更是五花八门，所以表现出专业化的特点。以减少成本和缩短研发周期考虑，要求它能运行在不同的微处理器平台上，能针对硬件变化进行结构与功能上的配置，以满足不同应用需要。

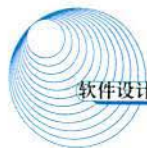
(3) 实时性：嵌入式操作系统广泛应用于过程控制、数据采集、传输通信、多媒体信息及关键要害领域需要迅速响应的场合，实时响应要求严格，因此实时性是其特点之一。

(4) 可靠性：系统构件、模块和体系结构必须达到应有的可靠性，对关键应用还要提供容错和防故障措施，以进一步提高可靠性。

(5) 易移植性：为了提高系统的易移植性，通常采用硬件抽象层 HAL (Hardware Abstraction Level) 和板级支撑包 BSP (Board Support Package) 的底层设计技术。HAL 提供了与设备无关的特性，屏蔽硬件平台的细节和差异，向操作系统上层提供统一接口，保证了系统的可移植性。而一般由硬件厂家提供的按给定的编程规范完成 BSP，则保证了嵌入式操作系统可在新推出的微处理器平台上运行。

2. 嵌入式系统开发环境

嵌入式系统开发环境通常配有源码级可配置的系统模块设计、丰富的同步原语、可选择的调度算法、可选择主存分配策略、定时器与计数器、多方式中断处理支持、多种异常处理选择、多种通信方式支持、标准 C 语言库、数学运算库和开放式应用程序接口。较著名的嵌入式操作系统有：Windows CE、VxWorks、pSOS、Palm OS、 μ C/OS-II。



3.8 操作系统实例

3.8.1 UNIX 操作系统

1. UNIX 系统的结构

UNIX 操作系统是由美国贝尔实验室发明的一种多用户、多任务的分时操作系统。现已发展成为当前使用普遍、影响深远的工业界主流的操作系统,成为重要的企业级操作平台,UNIX 广泛运行于 PC、小型机等各种环境,用于大型信息系统的关键业务服务,如数据库和 Internet 主机。UNIX 结构如图 3-29 所示。UNIX 最内层硬件提供基本服务,内核提供全部应用程序所需的各种服务。

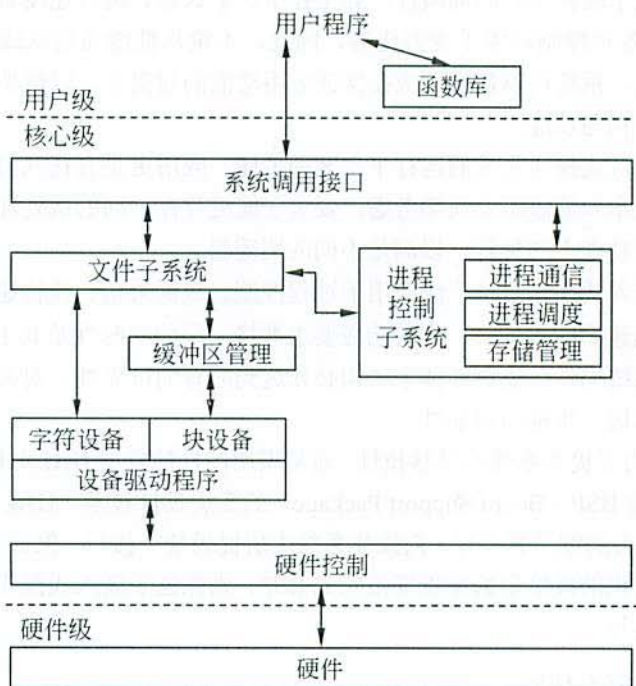


图 3-29 UNIX 系统结构

2. 文件系统

UNIX 文件系统的目录结构是树形带交叉勾连的,根目录记为“/”,非叶节点的为目录文

件，叶节点可以是目录文件、也可以是文件或特殊文件。目录是一个包含目录项的文件，在逻辑上，可以认为每个目录项都包含一个文件名，同时还包含说明该文件属性的信息。文件属性是文件类型、文件长度、文件主、文件的许可权（例如，其他用户能否访问该文件）、文件最后的修改时间等。当创建一个新目录时，系统自动创建了两个文件名：“.”（称为点）和“..”（称为点-点）。点表示当前目录，点-点表示父目录。在最高层次的根目录中，点-点与点相同。某些 UNIX 文件系统限制文件名的最大长度为 14 个字符，BSD 版本则将这种限制扩展为 255 个字符。

UNIX 文件系统具有如图 3-30 所示的结构。引导块：占据文件系统的开头，占一个物理块，包含引导代码段；超级块：描述文件系统的状态，如容量、空闲块号、空闲索引节点号（i_node）等；索引节点区：第一个索引节点是文件系统的根索引节点，当执行了 mount 命令之后，该文件系统的目录结构就可以从这个根索引节点开始进行存取了；数据存储空间：存放数据的区域。

引导块	超级块	索引节点区	数据存储空间
-----	-----	-------	--------

图 3-30 文件系统的布局

进程可以通过系统调用访问文件。例如，open（打开文件）、close（关闭文件）、write（写文件）、read（读文件）、stat（查询文件属性）、chmod（改变文件的许可权）、chown（改变文件所有者）、creat（创建一个文件）、mkdir（创建一个目录文件）、cd（改变当前目录）、link（建立连接）、unlink（删除文件连接）等。

3. 进程与存储管理

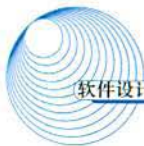
1) 进程的组成

在 UNIX 中进程由控制块 PCB、正文段和数据段组成。其中进程控制块 PCB 由常驻主存的基本进程控制块 proc 和非常驻主存的进程扩充控制块 user 结构；正文段是可供多个进程程序，为了管理可共享的正文段，UNIX 设置了一张正文表 text[]，每个正文段都占据一个表项，用来指明正文段在主存和磁盘的位置；数据段是进程执行时用到的数据段，若进程执行的程序是非共享的，则也构成数据段的一部分。

2) 进程控制

在 UNIX 中的进程控制子系统负责进程同步、进程间通信、存储管理及进程调度。控制进程的系统调用有 fork：创建一个子进程；exec：改变执行程序的映像；exit：结束一个进程的执行；wait：暂停进程的执行，用于进程之间的同步，例如父进程等待子进程执行结束；signal：控制进程对特别事件的响应；kill：发送软中断信号；msgsnd：发送消息；msgrcv：接受消息等。

3) 进程调度



UNIX 系统对进程的调度采用动态优先数调度算法,进程的优先数随进程的执行情况而变化。就绪进程是否能占用处理机的优先权取决于进程的优先数,优先数越小优先权越大。UNIX 系统中优先数确定方法有两种:设置方法和计算方法。设置方法是用于要进入睡眠的进程,当进程正在或即将转入用户态运行时,用计算方法确定优先数。UNIX 计算优先数的公式为:

$$p\text{-pri} = p\text{-cpu}/2 + PUSER + p\text{-nice} + NZERO$$

其中, $p\text{-pri}$ 表示进程的优先数; $p\text{-cpu}$ 为处理器的占用时间; $PUSER$ 和 $NZERO$ 表示偏置常数; $p\text{-nice}$ 允许用户使用 Shell 命令 `nice` 或系统调用 `nice` 修改,用户有权将其值设置为 0~39 之间的数。

4) 存储管理

UNIX 早期的版本采用“对换技术”扩充主存容量,进程可以被换出到对换区,也可以从对换区换进到主存。高版本的 UNIX 主存管理采用分页式虚拟存储机制,对换技术作为一种辅助手段。采用二次机会页面替换算法。

4. 设备管理

在 UNIX 系统中,文件等于系统中可用的任何资源。UNIX 的设计者们遵循一条这样的规则:UNIX 系统中可以使用的任何计算机资源都用一种统一的方法表示。他们选择用“文件”这个概念作为一切资源的抽象表示方法。

在 UNIX 文件系统中,尽管可以看到很多文件,但是这些文件并不一定是磁盘上的某些数据集合。实际上,UNIX 系统中的每一个设备,如鼠标、键盘、显示器、磁盘上的分区、打印机等,都在 UNIX 的文件系统中占用了索引节点。这个节点不仅有名字,而且有创建日期、访问权限等。总而言之,从表面上看来这些节点与一般的磁盘文件没有任何区别,但是它们是特殊文件(设备文件)。在编程时,可以通过一个通用的文件描述符(FileDescriptor)对它们进行访问。这种“通过文件描述符访问资源”的观念被扩展到 BSD Socket 当中,成为网络资源访问的标准方式。

UNIX 系统包括两类设备:块设备和字符设备。用户可以通过文件系统与设备接口,因为每个设备有一个文件名,可以像文件那样存取。设备文件有一个索引节点,在文件系统目录中占据一个节点,但其索引节点上的文件类型与其他文件不同,是“块”或者是“字符”特殊文件。文件系统与设备驱动程序接口是通过设备开关表。硬件与驱动程序之间的接口:控制寄存器、I/O 指令,一旦出现设备中断,根据中断矢量转相应的中断处理程序,完成用户所要求的 I/O 任务。UNIX 设备管理的主要特点:

(1) 块设备与字符设备具有相似的层次结构。这是指对它们的控制方法和所采用的数据结构、层次结构几乎相同。

(2) 将设备作为一个特殊文件，并赋予一个文件名。这样，对设备的使用类似于对文件的存取，具有统一的接口。

(3) 采用完善的缓冲区管理技术。引入“预先读”、“异步写”和“延迟写”方式，进一步提高系统效率。

在 UNIX 系统中，每类设备都有一个驱动程序，用它来控制该类设备。任何一个驱动程序通常都包含了用于执行不同操作的多个函数，如打开、关闭、启动设备、读和写等函数。为使核心能方便地转向各函数，系统为每类设备提供了一个设备开关表，开关表是每个设备驱动程序的一系列接口过程的入口表，给出了一组标准操作的驱动程序入口地址，文件系统可通过开关表中的各函数入口地址转向适当的驱动程序入口。如图 3-31 所示。图中显示了 UNIX 设备驱动程序通过相应的块设备开关表和字符设备开关表描述向上与文件系统的接口。

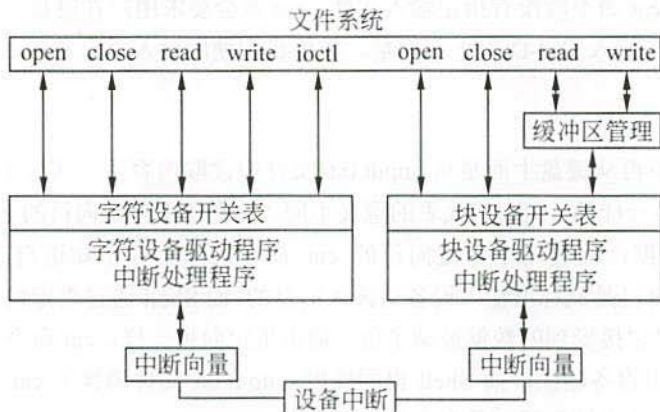


图 3-31 UNIX 系统中的设备开关表

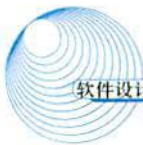
每类设备有自己的设备处理程序，但一般可分成两部分：用于启动设备的设备驱动程序和负责处理 I/O 完成工作的设备中断处理程序。

5. 输入/输出转向

Shell 即是一种命令语言，又是一种程序设计语言。在 UNIX 中，任何一个存放一条或多条命令的文件称为 Shell 程序或 Shell 过程。

1) 输入/输出转向

UNIX 系统 Shell 向用户提供了输入/输出转向命令，可以在不改变应用程序本身的情况下自由地改变其数据的输入源和输出目的地。其中，“>”、“>>”表示输出转向，“<”表示输入转向。例如，cat 命令用来将输入文件的数据显示在屏幕上：



```
cat input.txt
```

上述 `cat` 命令将 `input.txt` 文件中的内容输出到屏幕（标准输出设备）上。但是，如果我们将命令写成：

```
cat input.txt > output.txt
```

那么 `cat` 命令就会将原本输出到屏幕上的内容输入到文件 `output.txt` 中去并覆盖 `output.txt` 的内容。如果使用

```
cat input.txt >> output.txt
```

将 `input.txt` 文件的内容添加到 `output.txt` 文件的末尾。

如果希望使用 `cat` 命令时没有指定输入文件，`cat` 就会要求用户在键盘（标准输入设备）上输入数据，直到用户输入 `Ctrl-D` 为止。但是，如果我们使用输入重定向符，就可以写成：

```
cat < input.txt
```

这样，`cat` 就不再从键盘上而是从 `input.txt` 文件中读取内容了。实际上，这条命令与 `cat input.txt` 在效果上是一样的，但它们代表的意义不同。没有使用重定向符的 `cat` 命令知道自己在从文件中读取数据；而使用了重定向符的 `cat` 命令实际上并不知道自己得到的数据来自 `input.txt`，它只知道自己是从小标准输入设备上读入信息的，而 `Shell` 通过重定向耍了个花招，“骗”过了 `cat` 命令，使得它接受到的数据被调了包。输出重定向也一样，`cat` 命令在输出时也是只知道自己是向标准输出设备输出，而 `Shell` 也同样用 `output.txt` 文件调换了 `cat` 命令的标准输出设备，使得 `cat` 命令的输出转到了文件当中。

2) 管道

在 `UNIX` 中“`|`”表示管道。一个管道总是连接两个命令，将左边命令的标准输出与右边命令的标准输入相连，于是左边命令的输出结果就直接成了右边命令的输入。这个功能使得用户可以在不改动程序本身的前提下使多个程序可以通过标准输入/输出设备进行数据传递。利用管道命令可以简化命令行的写法，例如下面的 3 条命令：

```
ls > file  
sort < file1 > file2  
pr < file2
```

可以用管道命令表示为：

```
ls | sort | pr
```

例如,如果我们要统计当前目录中所有文件和目录中的数目,并将其记录在文件 output.txt 中。利用管道命令如下:

```
ls -a | wc -l > output.txt
```

ls -a 用来列出当前目录下的所有文件和目录;wc -l 负责统计 ls 输出中的行数;最后我们将输出重定向到 output.txt 中。命令十分简单,完成的工作却相当复杂。

6. Shell 程序

Shell 不但负责管理命令行界面,而且 Shell 自己也是一个编程的环境。实际上,我们可以将命令按照命令行的格式写入一个文件,再将其权限设置为可执行,就可以像普通命令一样执行它了。这个文件我们通常称为脚本(Script)。熟悉 DOS 的用户:熟悉批处理文件,Shell 脚本就相当于 DOS 的批处理文件。而且 Shell 脚本中也同样支持如 if、for、case 等程序控制流程,甚至还支持变量和函数定义。Shell 实际上是一种编程语言,利用 Shell 语言可以编写出功能很强的 Shell 程序,将程序段组合起来。

1) 正则表达式

在 UNIX 中正则表达式不仅用在 vi 中,还用在 shell 中。正则表达式用来确定字符串模式的一个规则集,是对文本字符串的一种描述,该描述能简洁而又完整地刻画文本字符串的关键特性。因此,正则表达式通常被用作字符串的匹配操作。正则表达式符号如表 3-4 所示。

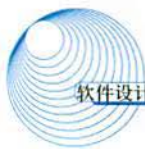
表 3-4 正则表达式符号

符号	含义
.	能与除换行符之外的行内任何字符匹配
*	匹配前一字符的零次或多次出现
[]	[]中只能匹配一个字符,若要匹配多个,则使用多个[]
^	如果方括号中的第一个字符是^,则匹配不属于[]中的字符
\$	如果出现在正则表达式末尾,则表示行尾,\$前面的正则表达式所匹配的字符串仅出现在行尾才匹配
\	转移符,用于改变特殊符号的含义,也可后跟一字符的八进制表示
" "	双引号内的字符在匹配时忽视其特殊含义
\<	字首匹配
\>	字尾匹配

【例 3.11】 匹配字符串:32787、188567、1234567890 的正则表达式。

解:这些字符串的共同特性都是数字,若要匹配任意长度的数字序列可用如下正则表达式:

[0-9][0-9]* 而不能使用[0-9]*



【例 3.12】 分别写出字符串: what 出现在行首和行尾的正则表达式。

解: /^what 当 what 出现在行首时才会被找到

/ what\$ 当 what 出现在行尾时才会被找到

若匹配文件的任意一行应用: ^.*\$。

2) Shell 变量

Shell 变量可分为 3 种类型: 用户定义变量、系统定义变量和 Shell 定义变量。

用户定义变量必须以字母或下划线开始, 可以包含字母、下划线和数字的字符序列。用户定义的 Shell 变量能用赋值语句置初值或重置值。例如: `ux=UNIX`。

系统定义变量如表 3-5 所示。

表 3-5 常用的系统定义变量

变量名	含义
HOME	用户主目录名
PATH	定义 Shell 在寻找命令时使用的查找路径
PS1	系统基本提示符, 默认为\$
PS2	系统辅助提示符, 默认为>
IFS	内部字段分割符, 默认是空格、制表符和换行符
MAIL	存放用户的邮件文件路径名
TERM	定义用户使用的终端类型
CDPATH	Cd 命令要查找的目录表
LOGNAME	用户的注册名
SHELL	Shell 程序的路径名
MANPATH	连接动态库时的搜索路径

Shell 定义变量如表 3-6 所示。

表 3-6 shell 定义变量

变量名	含义
\$0	命令名, 在 Shell 程序内可用\$0 获得调用该程序的名字
\$1-\$9	Shell 程序的位置参量
\$#	位置参数的个数, 不包括命令名
\$*	所有位置参量, 即相当于\$1,\$2,\$3,...
\$@	与\$*基本相同, 但当用双引号转义时, "\$@"还是能分解成多个参数, 但"\$*"则合并成一个参数。
\$?	上一命令的返回代码, 成功返回 0, 否则返回 1。
\$\$	当前命令的进程标识数
\$_	Shell 执行的最近后台进程标识数
\$_	Shell 标识位组成的字符串, 可由 Shell 传递来, 或由 set 命令设置

3) Shell 程序

Shell 向用户提供了许多用于简化输入的符号, 这些符号包括各种通配符、字符串定义符、转义符、变量定义符等。这些符号可以被看做是 Shell 的保留字, 通常称为“元字符”。元字符的种类和作用非常多, 它们无论在 Shell 的命令行输入还是在 Shell 程序设计中都起着非常重要的作用。

【例 3.13】 显示用户登录名、用户主目录以及当前命令的进程标识符的 Shell 程序如下:

```
echo username: $LOGNAME
echo Home directory: $HOME
echo Current shell's PID: $$
```

有趣的是, Shell 命令行本身也是一个交互式的脚本执行环境, 也就是说, 在命令行上同样可以使用脚本中的控制语句, 也可以定义变量 (实际上就是环境变量), 甚至可以定义函数。这都与脚本文件中的命令一样。但是有一点必须注意: Shell 程序有许多种, 不同的 Shell 有不同的编程命令和语法。虽然它们基本上大同小异, 但还是有许多差别。

3.8.2 Windows 2000/XP 操作系统

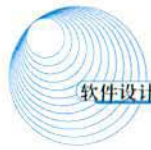
1. Windows 操作系统的体系结构

Windows 操作系统和许多操作系统一样通过硬件机制实现了核心态 (管态) 以及用户态 (目态) 两种特权状态。当处于核心态时, 可以执行任何指令, 并改变状态; 当处于用户态时, 只能执行非特权指令。用户程序一般都运行在用户态, 而操作系统中的哪些至关重要的代码运行在核心态, 以免被错误的应用程序破坏。

Windows 2000/XP 操作系统的核心组件使用了面向对象的设计原则, 例如, 它不能直接访问某个数据结构中由单个组件维护的消息, 这些组件只能使用外部接口传送参数访问或修改这些数据。Windows 2000/XP 操作系统的体系结构框架如图 3-32 所示。



图 3-32 Windows 2000/XP 操作系统体系结构



用户进程有4种类型:

(1) 系统支持进程: 例如登录进程 WINLOGO 和会话管理器 SMSS, 它们不是 Windows 2000/XP 的服务。

(2) 服务进程: 是 Windows 2000/XP 的服务, 如事件日志服务。

(3) 环境子系统: 它们向应用程序提供运行环境(操作系统功能调用接口), Windows 2000/XP 有3个环境子系统: Win32、POSIX 或 OS/21.2。

(4) 应用程序: 它们可以是 Win32、Windows3.2、MS-DOS、POSIX 或 OS/21.2 等5种类型。

从图中可以看出, 服务进程和应用程序是不能直接调用操作系统服务的, 必须通过子系统动态链接库与系统交互。

Windows 2000/XP 核心类组件包括:

(1) 核心包含了最低级的操作系统, 例如, 线程调度、中断和异常、多处理器同步等, 同时它也提供了执行体来实现高级结构的一组例程和基本对象。

(2) 执行体包含了基本的操作系统服务, 例如主存管理、进程和线程管理、安全控制、I/O 以及进程间的通信。

(3) 硬件抽象层, 将内核、设备驱动程序, 以及执行体与硬件分隔开来, 以便适应多种平台。

(4) 设备驱动程序, 包括文件系统和硬件设备驱动程序, 其中, 硬件设备驱动程序将用户的 I/O 函数调用转换位对特定硬件设备的 I/O 请求。

(5) 图形引擎包含了实现图形用户界面的基本函数。

2. 文件系统

Windows 2000/XP 支持传统的 FAT 文件系统, 该文件系统最初是针对相对较小容量的磁盘而设计的, 但随着计算机外存容量的迅速扩展, 出现了明显的不适应性。FAT 文件系统最多只能容纳 2^{12} 或 2^{16} 个簇, 单个文件卷的容量小于 2GB。如果一个簇包含的扇区数增加可以使得单个卷的容量增大, 但是文件空间的碎片很多, 浪费很大。

从 Windows 9x 和 Windows me 开始, FAT 表被扩展到了 32 位, 形成了 FAT32 文件系统, 解决了 FAT16 系统在文件容量上的问题, 可以支持 4GB 的大硬盘分区, 但由于 FAT 表的大幅度扩充, 造成文件系统效率大幅度下降。Windows 98 支持 FAT32 文件系统, 但与其同期开发的 Windows NT 不支持 FAT32, 基于 NT 构建的 Windows 2000/XP 支持 FAT32 系统。除此之外, Windows 2000/XP 支持的文件系统格式包括 CDFS、UDF、FAT12、FAT16、FAT32 和 NTFS。

NTFS 文件系统是 Windows 2000/XP 本身的文件系统格式。NTFS 使用 64 位簇进行索引, 这使得 NTFS 有能力寻址达 16EB (16×10^{18} B)。NTFS 的主要特征如下:

(1) 可恢复性: 之所以建立新 Windows 文件系统, 就是为了具备从系统崩溃和磁盘故障中恢复数据的能力。当发生故障时, NTFS 能够重建文件卷, 并使他们恢复到一个一致的状态。

(2) 安全性: NTFS 使用对象模型来实施安全机制。一个打开的文件是作为一个文件对象

来实现的, 该文件对象有一个作为该文件的一部分而存储在磁盘上的安全描述体。当进程试图打开一个文件对象的句柄之前, NTFS 安全系统会验证该进程是否具有资格。安全描述体与用户登录到系统并提供的密码的请求结合来保证除非是系统管理员或文件的所有者给定了特定的许可, 否则进程就不能访问该文件。

(3) 大磁盘和大文件: NTFS 比 FAT 以及其他大多数文件系统都能够更有效的支持非常大的磁盘和非常大的文件。

(4) 多数据流: 在 NTFS 中, 每一个与文件有关的信息单元, 如文件名、所有者、时间标记、数据等都可以作为文件属性来执行, 所以 NTFS 文件可以包含多数据流。这项技术为高端服务器应用程序提供了增强功能的新手段。

(5) 通用索引功能: NTFS 的体系结构允许在一个磁盘卷中索引文件属性, 从而可以有效地定位匹配各种标准文件。在 Windows 2000/XP 中, 这种索引机制被扩展到其他属性, 如对象 ID。

Windows 2000/XP 还提供分布式文件服务。分布式文件系统(DFS)是用于 Windows 2000/XP 服务器上的网络组件, 最初它是作为一个扩展层发布在 NT4 的, 但在功能上受到限制。在 Windows 2000/XP 中, 这种限制得到了修正。DFS 使用户更容易找到和管理网上的数据。使用 DFS, 可以更加容易地创建单目录树, 该目录树可以包括多文件服务器和组、部门或企业中的文件共享。

3. 进程

Windows 2000/XP 中进程是资源分配的单位, 并将进程作为对象来进行管理, 可以通过相应的句柄来引用对象, 操作系统提供一组控制进程对象的服务。进程对象属性包括进程标识、资源访问令牌、进程的基本优先级等。

Windows 2000/XP 的线程是内核线程, 是处理机调度的单位。线程的上下文主要包括寄存器、线程环境块、核心栈和用户栈。Windows 2000/XP 中的线程有 7 种状态, 如图 3-33 所示。

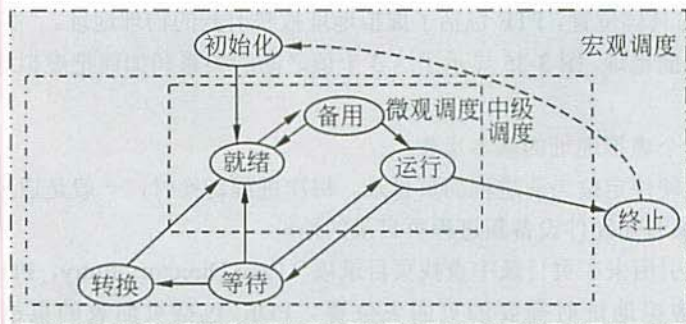
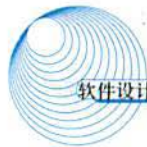


图 3-33 Windows 2000/XP 中线程的状态



- (1) 就绪状态: 已获得除处理机之外的所需资源, 正等待调度。
- (2) 备用状态: 已选择好线程的执行处理机, 正等待描述表切换, 以进入运行态。系统中每个处理机只能有一个处于备用状态的线程。
- (3) 运行状态: 正在处理机上运行的线程, 直到被抢先、或时间片用完、或线程终止或进入等待。
- (4) 等待状态: 线程正等待某对象, 以同步线程的执行。当等待时间出现时, 根据优先级进入运行或就绪状态。
- (5) 转换状态: 该状态与就绪状态类似, 但线程的内核堆栈位于外存。
- (6) 终止状态: 线程执行完毕就进入终止状态, 如执行体有一个指向线程对象的指针, 可以将处于终止状态的线程重新初始化, 并再次使用。
- (7) 初始化状态: 线程创建时的状态。

4. 存储管理

Windows 2000/XP 默认情况下使用二级页面表结构来转换物理地址和虚拟地址。32 位的虚拟地址被解释为 3 个单独的组件: 页面目录索引、页面表索引和字节索引, 并被当作描述页面映射的结构的索引, 图 3-34 所示。页大小和 PTE 宽度表明页面目录和页面表索引字段的宽度。例如, 在 x86 系统上, 因为页面为 4096B, 字节索引则为 12 位。

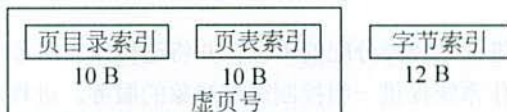


图 3-34 x86 系统中一个 32 位虚拟地址结构

“页面目录索引”用来指明虚拟地址的页目录在页表中的位置; “页面表索引”用来确定页表项 PTE 在页表中的具体位置, PTE 包括了虚拟地址被映射到的物理地址。“字节索引”在该物理页面内寻找正确的地址。图 3-35 显示了这 3 个值之间的关系和如何把虚拟地址映射到物理地址的过程。

以下是转换一个虚拟地址的基本步骤:

- (1) 主存管理硬件定位当前进程的页目录。每次进程切换时, 一般是通过操作系统设置专用的 CPU 寄存器来通知硬件设备新进程页目录的地址。

- (2) 页目录索引用来在页目录中查找页目录项 (Page Directory Entry, PDE) 的索引, 页目录项描述了映射虚拟地址时需要的页面表位置。PDE 包括页面表的页框号 (Page Frame Number, PFN)。

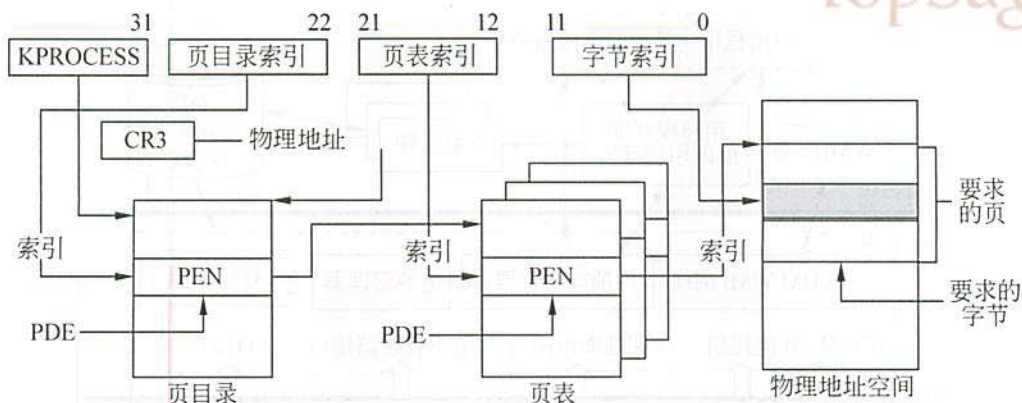


图 3-35 虚拟地址变换

(3) 页表索引用来在页表中指出 PTE 的位置，PTE 描述了虚拟页面在物理主存的位置。

(4) PTE 用来确定页框的位置。如果所需的页面有效，则页表项包含物理主存中的一个页框号 PFN。如果页面无效，则主存管理错误处理程序将查找页面并试图使页面有效。如果不能使页面有效，错误处理程序产生一个访问违例或错误检查。

(5) 当 PTE 指向一个有效的页面时，字节索引用来查找在物理页面内所需数据的地址。

5. 设备管理

Windows 2000/XP 操作系统采用和强调了软件工程中抽象的原则，在设计中全力找出各种事务的共性，用一致的模型、方法和界面来规范化，如客户/服务器模型规范各个用户进程之间的关系。尤其突出的是建立了广义的资源管理概念，并统一地用对象模型来描述和规范化。这样降低了系统的复杂性。在输入/输出设计上，建立了一个统一的高层界面：IO 设备虚拟界面，即将所有的读写数据看成直接送往虚拟文件的字节流。

Windows 2000/XP 的 I/O 系统体系结构如图 3-36 所示，它由几个可执行模块和大量设备驱动程序组成。

从图中我们可以看到，Windows 2000 的 I/O 体系的设计使用了分层结构，这有利于实现其平台无关性，也为其他目标的实现带来的便利。在整个体系的最底层也就是直接与硬件交互的层是硬件抽象层 (HAL)，它隐藏了不同硬件平台之间的差异，使得上层的驱动程序、各可执行模块的实现可以与平台无关。从具体实现的角度上讲，HAL 实际是由系统提供的许多总线设备驱动程序的集合，尽管具体的总线不同但它们向上层提供了统一的接口。

在 HAL 层之上的是设备驱动层，在这个层次上，各驱动程序利用总线驱动程序提供的 I/O 端口访问函数访问并控制对应的硬件设备。事实上，设备驱动层又可分为许多更细的层次。

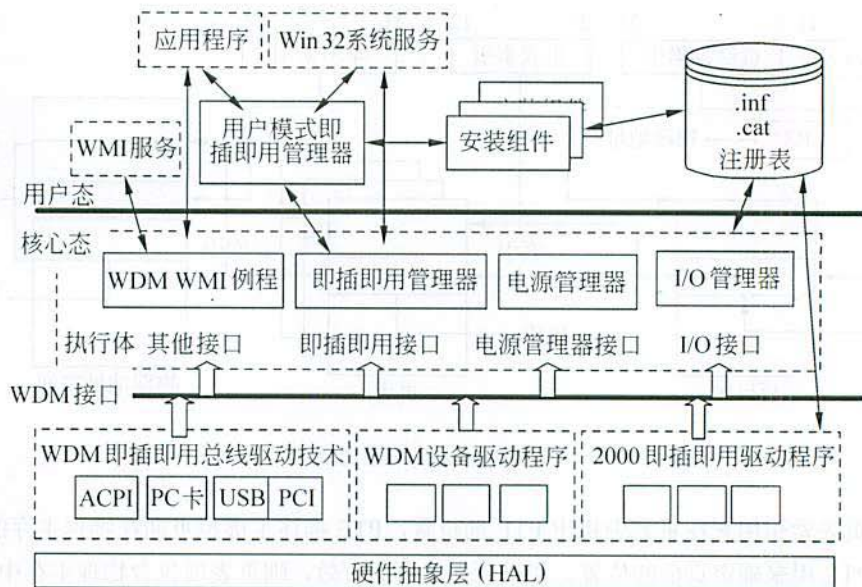
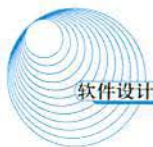


图 3-36 Windows 2000/XP 的 I/O 系统体系结构

在设备驱动层之上是 I/O 系统层, 它由一系列的管理器组成, 如 I/O 管理器、电源管理器、即插即用 (PnP) 管理器、WMI (Windows Management Instrument) 等。他们负责用户模式程序与核心模式的驱动程序的交互控制、即插即用资源分配等一系列工作。图中使用了双箭头表示 I/O 系统层与设备驱动层的关系, 但实际上 I/O 系统层也可直接与 HAL 通信, 例如 PnP 管理器就直接与总线驱动合作, 来检测并响应设备的加入和移除。

以上这些模块都运行在 Windows 2000/XP 的核心态, 它们是操作系统执行体的一部分, 操作系统对他们的错误操作没有保护机制, 也就是说如果这些模块出现问题整个系统就会崩溃。一般情况下, 在这 3 层中开发者只可编写驱动程序模块加入核心。

第4章 系统开发和运行知识

本章的主要内容包括软件工程的基础知识、软件项目管理基础知识、软件过程改进、系统分析与设计基础知识,以及系统运行维护和系统评价基础知识等。

在系统分析与系统设计知识部分,主要以结构化分析设计方法为主,面向对象分析与设计方法参见本书第10章。

4.1 软件工程基础知识

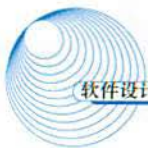
自从 1968 年首次提出“软件工程”这个名词以来，软件工程已经成为计算机软件的一个重要分支和研究方向。软件工程是指应用计算机科学、数学及管理科学等原理，以工程化的原则和方法来解决软件问题的工程，其目的是提高软件生产率、提高软件质量、降低软件成本。

软件工程涉及到软件开发、维护、管理等多方面的原理、方法、工具与环境,限于篇幅本章不能对软件工程做全面的介绍。根据软件设计师级考试大纲的要求,本章着重介绍软件开发过程中的原理,其他内容只作简单的介绍。

4.1.1 软件工程概述

早期的软件主要指程序，程序的开发采用个体工作方式，开发工作主要依赖于开发人员的个人技能和程序设计技巧。当时的软件通常缺少与程序有关的文档，软件开发的实际成本和进度往往与预计的相差甚远，软件的质量得不到保证，开发出来的软件常常不能使用户满意。随着计算机应用的需求不断增长，软件的规模也越来越大，然而软件开发的生产率远远跟不上计算机应用的迅速增长。此外，由于软件开发时缺少好的方法指导和工具辅助，同时又缺少相关文档，使得大量已有的软件难以维护。上述这些问题严重地阻碍了软件的发展，20 世纪 60 年代中期，人们把上述软件开发和维护过程中所遇到的各种问题称为“软件危机”。

1968 年在德国召开的 NATO (North Atlantic Treaty Organization, 北大西洋公约组织) 会议上首次提出了“软件工程”这个名词, 希望用工程化的原则和方法来克服软件危机。在此以后, 人们开展了软件开发模型、开发方法、工具与环境的研究, 提出了瀑布模型、演化模型、螺旋模型、喷泉模型等开发模型, 出现了面向数据流方法、面向数据结构的方法、面向对象方法等开发方法, 以及一批 CASE (Computer Aided Software Engineering, 计算机辅助的软件工程) 工具和环境。



1. 软件生存周期

同任何事物一样,一个软件产品或软件系统也要经历孕育、诞生、成长、成熟、衰亡等阶段,一般称为软件生存周期。把整个软件生存周期划分为若干阶段,使得每个阶段有明确的任务,使规模大、结构复杂和管理复杂的软件开发变得容易控制和管理。通常,软件生存周期包括可行性分析与项目开发计划、需求分析、设计(概要设计和详细设计)、编码、测试、维护等活动,可以将这些活动以适当的方式分配到不同的阶段去完成。

1) 可行性分析与项目开发计划

这个阶段必须要回答的问题是,要解决的问题是什么?该问题有可行的解决办法吗?若有解决的办法,则需要多少费用?需要多少资源?需要多少时间?要回答这些问题,就要进行问题定义、可行性分析,制订项目开发计划。

可行性分析与项目计划阶段的参加人员有用户、项目负责人、系统分析师。该阶段所产生的文档有可行性分析报告和项目开发计划。

2) 需求分析

需求分析阶段的任务不是具体地解决问题,而是准确地确定软件系统必须做什么,确定软件系统的功能、性能、数据、界面等要求,从而确定系统的逻辑模型。该阶段的参加人员有用户、项目负责人、系统分析师,产生的文档有软件需求说明书。

3) 概要设计

在概要设计阶段,开发人员要把确定的各项功能需求转换成需要的体系结构。在该体系结构中,每个成分都是意义明确的模块,即每个模块都和某些功能需求相对应。因此,概要设计就是设计软件的结构,明确软件由哪些模块组成,这些模块的层次结构是怎样的,这些模块的调用关系是怎样的,每个模块的功能是什么。同时还要设计该项目的应用系统的总体数据结构和数据库结构,即应用系统要存储什么数据,这些数据是什么样的结构,它们之间有什么关系。

概要设计阶段参加的人员有系统分析师和软件设计师,该阶段的主要文档有概要设计说明书。

4) 详细设计

详细设计阶段的主要任务就是对每个模块完成的功能进行具体描述,要把功能描述转变为精确的、结构化的过程描述。即该模块的控制结构是怎样的,先做什么,后做什么,有什么样的条件判定,有什么重复处理等等,并用相应的表示工具把这些控制结构表示出来。

详细设计阶段参加的人员有软件设计师和程序员,该阶段的主要文档有详细设计文档。

5) 编码

编码阶段就是把每个模块的控制结构转换成计算机可接受的程序代码,即写成某种特定程

序设计语言表示的源程序清单。

6) 测试

测试时保证软件质量的重要手段,其主要方式是在设计测试用例的基础上检查软件的各个组成部分。测试阶段的参加人员通常由另一部门(或单位)的软件设计师或系统分析师承担,该阶段产生的文档有软件测试计划和软件测试报告。

7) 维护

软件维护是软件生存周期中时间最长的阶段。已交付的软件投入正式使用后,便进入软件维护阶段,它可以持续几年甚至几十年。软件运行过程中可能由于各方面的原因,需要对其进行修改。其原因可能是运行中发现了软件隐含的错误而需要修改,也可能是为了适应变化了的软件工作环境而需要做适当变更,还可能是因为用户业务发生变化而需要扩充和增强软件的功能等。

2. 软件生存周期模型

软件生存周期模型是描述软件开发过程中各种活动如何执行的模型。软件生存周期模型确立了软件开发和演绎中各阶段的次序限制以及各阶段或机动的准则,确立开发过程所遵守的规定和限制,便于各种活动的协调,便于各种人员的有效通信,有利于活动重用,有利于活动管理。常见的软件生存周期模型有瀑布模型、演化模型、螺旋模型、喷泉模型等。

1) 瀑布模型(Waterfall Model)

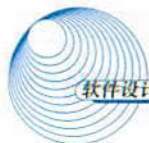
瀑布模型是将软件生存周期各个活动规定为依线性顺序连接的若干阶段的模型。它包括可行性分析、项目开发计划、需求分析、概要设计、详细设计、编码、测试和维护。它规定了由前至后、相互衔接的固定次序,如同瀑布流水,逐级下落。

瀑布模型为软件的开发和维护提供了一种有效的管理模式,根据这一模式制定开发计划,进行成本预算,组织开发力量,以项目的阶段评审和文档控制为手段有效地对整个开发过程进行指导,所以它是以文档作为驱动、适合于软件需求很明确的软件项目的模型。

但是瀑布模型在大量的软件开发实践中也逐渐暴露出它的严重缺点。它是一种理想的线性开发模式,缺乏灵活性,特别是无法解决软件需求不明确或不准确的问题。

2) 演化模型(Evolutionary Model)

大量的软件开发实践表明,许多开发项目在开始时对软件需求的认识是模糊的,因此很难一次开发成功。为了减少因对软件需求的了解不够确切而给开发工作带来的风险,我们可以在获取了一组基本的需求后,通过快速分析构造出该软件的一个初始可运行版本,这个初始的软件通常称之为原型(Prototype),然后根据用户在使用原型的过程中提出的意见和建议对原型进行改进,获得原型的新版本。重复这一过程,最终可得到令用户满意的软件产品。采用演化模型的开发过程,实际上就是从初始的原型逐步演化成最终软件产品的过程。演化模型特别适用于对软件需求缺乏准确认识的情况。



3) 螺旋模型 (Spiral Model)

对于复杂的大型软件, 开发一个原型往往达不到要求。螺旋模型将瀑布模型和演化模型结合起来, 加入了两种模型均忽略的风险分析, 弥补了这两种模型的不足。

螺旋模型将开发过程分为几个螺旋周期, 每个螺旋周期大致和瀑布模型相符合。每个螺旋周期分为 4 个工作步: 第一, 制定计划。确定软件的目标, 选定实施方案, 明确项目开发的限制条件; 第二, 风险分析。分析所选的方案, 识别风险, 消除风险; 第三, 实施工程。实施软件开发, 验证阶段性产品; 第四, 用户评估。评价开发工作, 提出修正建议, 建立下一个周期的开发计划。

4) 喷泉模型 (Water Fountain Model)

喷泉模型是一种以用户需求为动力, 以对象作为驱动力的模型, 适合于面向对象的开发方法。它克服了瀑布模型不支持软件重用和多项开发活动集成的局限性。喷泉模型使开发过程具有迭代性和无间隙性。迭代意味着模型中的开发活动常常需要重复多次, 在迭代过程中不断地完善软件系统。无间隙是指在开发活动(如分析、设计、编码)之间不存在明显的边界, 也就是说, 它不像瀑布模型那样, 需求分析活动结束后才开始设计活动, 设计活动结束后才开始编码活动, 而是允许各种开发活动交叉、迭代地进行。

3. 软件开发方法

软件开发方法是一种使用早已定义好的技术集及符号表示习惯来组织软件生产的过程。

1) 结构化方法

结构化方法由结构化分析、结构化设计、结构化程序设计构成, 它是一种面向数据流的开发方法。结构化分析是根据分解与抽象的原则, 按照系统中数据处理的流程, 用数据流图来建立系统的功能模型, 从而完成需求分析工作。结构化设计是根据模块独立性准则、软件结构优化准则将数据流图转换为软件的体系结构, 用软件结构图来建立系统的物理模型, 实现系统的概要设计。结构化程序设计是根据结构程序设计原理, 将每个模块的功能用相应的标准控制结构表示出来, 从而实现详细设计。

结构化方法总的指导思想是自顶向下、逐层分解, 它的基本原则是功能的分解与抽象。它是软件工程中最早出现的开发方法, 特别适合于数据处理领域的问题, 但是不适合解决大规模的、特别复杂的项目, 且难于适应需求的变化。

2) Jackson 方法

Jackson 方法是一种面向数据结构的开发方法。因为一个问题的数据结构与处理该数据结构的控制结构有着惊人的相似之处, 该方法就是根据这一思想而形成了最初的 JSP (Jackson Structure Programming) 方法。首先描述问题的输入、输出数据结构, 分析其对应性, 然后推出相应的程序结构, 从而给出问题的软件过程描述。

JSP 方法是以数据结构为驱动的, 适合于小规模的项目。当输入数据结构与输出数据结构之间没有对应关系时, 难以应用此方法。基于 JSP 方法的局限性, 又发展了 JSD (Jackson System Development) 方法, 它是 JSP 方法的扩充。

JSD 方法是一个完整的系统开发方法。首先建立现实世界的模型, 再确定系统的功能需求, 对需求的描述特别强调操作之间的时序性。它是以事件作为驱动的, 它是一种基于进程的开发方法, 所以应用于时序特点较强的系统, 包括数据处理系统和一些实时控制系统。

3) 维也纳开发方法 (VDM)

维也纳开发方法是一种基于模型的方法, 它以指称语义为基础。它的主要思想是将软件系统当作模型来描述, 把软件的输入/输出看作模型对象, 把这些对象在计算机内的状态看作该模型在对象上的操作。它的目的是从软件系统最高一级抽象直到最后生成目标的每一步都给予形式化说明, 以提高软件的可靠性。

4) 面向对象开发方法

面向对象开发方法的基本出发点是尽可能按照人类认识世界的方法和思维方法来分析和解决问题。客观世界是由许多具体的事物、事件、概念和规则组成, 这些均可被看成对象, 面向对象方法正是以对象作为最基本的元素, 它也是分析问题、解决问题的核心。

面向对象开发方法包括面向对象分析、面向对象设计、面向对象实现。面向对象开发方法有 Booch 方法、Coad 方法和 OMT 方法等。为了统一各种面向对象方法的术语、概念和模型, 1997 年推出了统一建模语言, 即 UML (Unified Modeling Language)。它是面向对象的标准建模语言, 通过统一的语义和符号表示, 使各种方法的建模过程和表示统一起来, 已成为面向对象建模的工业标准。

4.1.2 软件需求分析

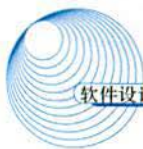
需求分析是软件生存周期中相当重要的一个阶段。由于开发人员熟悉计算机但不熟悉应用领域的业务, 用户熟悉应用领域的业务但不熟悉计算机, 因此对于同一个问题, 开发人员和用户之间可能存在认识上的差异。在需求分析阶段, 通过开发人员与用户之间的广泛交流, 不断澄清一些模糊的概念, 最终形成一个完整的、清晰的、一致的需求说明。可以说, 需求分析的好坏将直接影响到所开发的软件的成败。

1. 需求分析的任务

需求分析主要是确定待开发软件的功能、性能、数据、界面等要求。具体说来可有以下几点:

(1) 确定软件系统的综合要求

- ① 系统界面要求: 描述软件系统的外部特性, 即系统从外部输入哪些数据, 系统向外部



输出哪些数据。

- ② 系统的功能要求：列出软件系统必须完成的所有功能。
- ③ 系统的性能要求：如响应时间、吞吐量、处理时间、对主存和外存的限制等。
- ④ 安全性、保密性和可靠性方面的要求。
- ⑤ 系统的运行要求：如对硬件、支撑软件、数据通信接口等的要求。
- ⑥ 异常处理要求：在运行过程中出现异常情况（如临时性或永久性的资源故障，不合法或超出范围的输入数据、非法操作、数组越界等）时应采取的行动以及希望显示的信息。
- ⑦ 将来可能提出的要求：主要是为将来可能的扩充和修改做准备。

(2) 分析软件系统的数据要求

包括基本数据元素、数据元素之间的逻辑关系、数据量、峰值等。常用的数据描述手段是实体-关系模型。

(3) 导出系统的逻辑模型

在结构化分析方法中可用数据流程图来描述；在面向对象分析方法中可以用类模型来描述。

(4) 修正项目开发计划

在明确了用户的真正需求后，可以更准确地估算软件的成本和进度，从而修正项目开发计划。

(5) 可开发一个原型系统

对一些需求不够明确的软件，可以先开发一个原型系统，以验证用户的需求。

需要再次强调的是，需求分析阶段主要解决“做什么”的问题，而“怎么做”则由设计阶段来完成。

2. 需求的分类

软件需求就是系统必须完成的事，以及必须具备的品质。软件需求包括功能需求、非功能需求和设计约束 3 方面的内容。

- (1) 功能需求：所开发的软件必须具备什么样的功能。
- (2) 非功能需求：是指产品必须具备的属性或品质，如可靠性、性能、响应时间、容错性、扩展性等。
- (3) 设计约束：也称为限制条件、补充规约，这通常是对解决方案的一些约束说明。

3. 软件需求分析方法

需求分析方法由对软件的数据域和功能域的系统分析过程及其表示方法组成。它定义了表示系统逻辑视图和物理视图的方式。大多数的需求分析方法是由数据驱动的，也就是说，这些方法提供了一种表示数据域的机制，开发人员根据这种表示，确定软件功能及其他特性，最终建立一个待开发软件的抽象模型，即目标系统的逻辑模型。数据域具有 3 种属性：数据流、数

据内容和数据结构。通常，一种需求分析方法总要利用其中的一种或几种属性。

目前已经出现了许多需求分析方法，每一种分析方法都引入了不同的记号和分析策略，但是它们都具有以下的共性：支持数据域分析的机制、功能表示的方法、接口的定义、问题分解的机制以及对抽象的支持、逻辑视图和物理视图、系统抽象模型等。

4. 需求工程

需求工程就是包括创建和维护系统需求文档所必需的一切活动的过程，也就是指需求开发和需求管理两个工作。

(1) 需求开发：包括需求捕获、需求分析、编写规格说明书和需求验证 4 个阶段。在这个阶段需要确定产品所期望的用户类型、获取每种用户类型的需求、了解实际用户的任务和目标，以及这些任务所支持的业务需求、分析源于用户的信息、对需求进行优先级分类、将所搜集的需求编写成软件规格说明书和需求分析模型、对需求进行评审等工作。

(2) 需求管理：通常包括定义需求基线、处理需求变更、需求跟踪等方面的工作。

4.1.3 软件开发项目管理

为了使软件项目开发获得成功，必须对软件开发项目的工作范围、可能遇到的风险、需要的资源（人、硬件/软件）、要实现的任务、经历的里程碑、花费的工作量（成本），以及进度的安排等等做到心中有数。而软件项目管理可以提供这些信息。这种管理开始于技术工作开始之前，在软件从概念到实现的过程中持续进行，最后终止于软件工程过程结束。

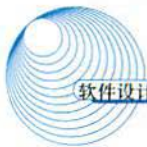
1. 成本估算

由于软件具有可见性差、量化难等特殊性，因此很难在项目完成前准确地估算出开发软件所需的工作量和费用。通常我们可以根据以往开发类似软件的经验（也可以是别人的经验）来进行成本估算。也可以将软件项目划分成若干个子系统或按照软件生存周期的各个阶段分别估算其成本，然后汇总出整个软件的成本。此外还可以使用经验公式和成本估算模型来进行估算。

1) 成本估算方法

(1) 自顶向下估算方法：估算人员参照以前完成的项目所耗费的总成本（或总工作量），来推算将要开发的软件的总成本（或总工作量），然后把它们按阶段、步骤和工作单元进行分配，这种方法称为自顶向下估算方法。自顶向下估算方法的主要优点是对系统级工作的重视，所以估算中不会遗漏诸如集成、配置管理之类的系统级事务的成本估算，且估算工作量大、速度快。它的缺点是往往不清楚低级别上的技术性困难问题，而这些困难将会使成本上升。

(2) 自底向上估算方法：自底向上估算方法是将待开发的软件细分，分别估算每一个子任务所需要的开发工作量，然后将它们加起来，得到软件的总开发量。这种方法的优点是对每一



部分的估算工作交给负责该部分工作的人来做,所以估算较为准确。其缺点是估算往往缺少与软件开发有关的系统级工作量,所以估算往往偏低。

(3) 差别估算方法:差别估算方法是将待开发项目与一个或多个已完成的类似项目进行比较,找出与某个相类似项目的若干不同之处,并估算每个不同之处对成本的影响,导出待开发项目的总成本。该方法的优点是可以提高估算的准确度,缺点是不容易明确“差别”的界限。

除了以上方法之外,还有专家估算法、类推估算法和算式估算法等。

(4) 专家估算法:依靠一个或多个专家对要求的项目做出估算,其精确性取决于专家对估算项目的定性参数的了解 and 他们的经验。

(5) 类推估算法:在自顶向下的方法中,它是将估算项目的总体参数与类似项目进行直接比较得到结果;在自底向上方法中,类推是在两个具有相似条件的工作单元之间进行。

(6) 算式估算法:专家估算法和类推估算法的缺点在于,他们依靠带有一定盲目性和主观性的猜测对项目进行估算。算式估算法则是企图避免主观因素的影响。用于估算的方法有两种基本类型:由理论导出和由经验导出。

2) 成本估算模型

常用的软件成本估算模型有 Putnam 模型和 COCOMO 模型。

(1) Putnam 模型:是一种动态多变量模型,它是假设在软件开发的整个生存期中工作量的分布。

(2) COCOMO 模型:是结构性成本模型,是最精确、最易于使用的成本估算模型之一。该模型可以分为:

① 基本 COCOMO 模型,是一个静态单变量模型,它是对整个软件系统进行估算。

② 中级 COCOMO 模型,是一个静态多变量模型。它将软件系统模型分为系统和部件两个层次,系统由部件构成,它把软件开发所需人力(成本)看作是程序大小和一系列“成本驱动属性”的函数。

③ 详细 COCOMO 模型,它将软件系统模型分为系统、子系统和模块 3 个层次,它除包括中级模型所考虑的因素外,还考虑了在需求分析、软件设计等每一步的成本驱动属性的影响。

2. 风险分析

当在软件工程环境中考虑风险时,主要是基于关心未来、关心变化、关心选择这 3 个概念提出的。在进行软件工程分析时,项目管理人员要进行 4 种风险评估活动,包括建立表示风险概念的尺度,描述风险引起的后果,估计风险影响的大小,确定风险估计的正确性。风险分析实际上是 4 个不同的活动:风险识别、风险预测、风险评估和风险控制。

1) 风险识别

风险识别是试图系统化地确定对项目计划(估算、进度、资源分配)的威胁。风险识别的

一个方法是建立风险条目检查表。该检查表可以用于识别风险,并使得人们集中来识别下列常见的已知的及可预测的风险。

- ① 产品规模:与要建造或要修改的软件的总体规模相关的风险。
- ② 商业影响:与管理或市场所加诸的约束相关的风险。
- ③ 客户特性:与客户的素质以及开发者和客户定期通信的能力相关的风险。
- ④ 过程定义:与软件过程被定义的程度以及它们被开发组织所遵守的程序相关的风险。
- ⑤ 开发环境:与用以构建产品的工具的可用性及质量相关的风险。
- ⑥ 构建的技术:与待开发软件的复杂性及系统所包含技术的“新奇性”相关的风险。
- ⑦ 人员数目及经验:与参与工作的软件工程师的总体技术水平及项目经验相关的风险。

2) 风险预测

风险预测又称风险估算,它从两个方面评估一个风险:风险发生的可能性或概率以及如果风险发生了所产生的后果。通常,项目计划人员与管理人员、技术人员一起进行4种风险预测活动:

- ① 建立一个尺度或标准,以反映风险发生的可能性。
- ② 描述风险的后果。
- ③ 估计风险对项目和产品的影响。
- ④ 标注风险预测的整体精确度,以免产生误解。

3) 风险评估

在进行风险评估时,建立了如下形式的三元组:

$$(r_i, l_i, x_i)$$

其中, r_i 表示风险; l_i 表示风险发生的概率; x_i 则表示风险产生的影响。

一个对风险评估很有用的技术就是定义风险参照水准。对于大多数软件项目来说,成本、进度和性能就是3种典型的风险参照水准。即对于成本超支、进度延期、性能降低(或它们的某种组合)有一个表明导致项目终止的水准。

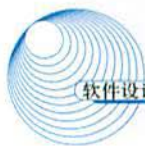
在风险评估过程中,需要执行以下4个步骤:

- ① 定义项目的风险参考水平值;
- ② 建立每一组 (r_i, l_i, x_i) 与每一个参考水平值之间的关系;
- ③ 预测一组临界点以定义项目终止区域,该区域由一条曲线或不确定区域所界定;
- ④ 预测什么样的风险组合会影响参考水平值。

4) 风险控制

这一步的所有风险分析活动只有一个目的——辅助项目组建立处理风险的策略。一个有效的策略必须考虑3个问题:

- ① 风险避免。



② 风险监控。

③ 风险管理及意外事件计划。

如果软件项目组对于风险采用主动的方法,则避免永远是最好的策略。这可以通过建立一个风险缓解计划来达到。

风险管理策略可以包含在软件项目计划中,或者风险管理步骤也可以组织成一个独立的风险缓解、监控和管理计划(RMMM计划)。RMMM计划将所有风险分析工作文档化,并由项目管理者作为整个项目计划中的一部分来使用。

3. 进度管理

进度的合理安排是如期完成软件项目的重要保证,也是合理分配资源的重要依据,因此进度安排是管理工作的一个重要组成部分。

1) 进度安排的方式

软件开发项目的进度安排有两种方式:

- (1) 系统最终交付日期已经确定,软件开发部门必须在规定期限内完成。
- (2) 系统最终交付日期只确定了大致的年限,最后交付日期由软件开发部门确定。

2) 进度安排的常用图形描述方法

进度安排的常用图形描述方法有 Gantt 图(甘特图)和计划评审技术 PERT(Program Evaluation & Review Technique)图。

(1) Gantt 图: Gantt 图中横坐标表示时间(如时、天、周、月、年等),纵坐标表示任务,图中的水平线段表示对一个任务的进度安排,线段的起点和终点对应在横坐标上的时间分别表示该任务的开始时间和结束时间,线段的长度表示完成该任务所需的时间。图 4-1 所示的 Gantt 图描述了 3 个任务的进度安排。该图表示:任务 1 首先开始,完成它需要 12 周时间;任务 2 在 2 周后开始,完成它需要 18 周;任务 3 在 12 周后开始,需要 10 周。

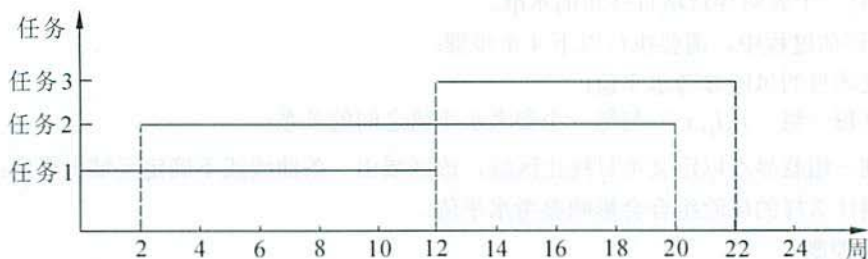


图 4-1 Gantt 图实例

Gantt 图能清晰地描述每个任务从何时开始,到何时结束以及各个任务之间的并行性;但

是它不能清晰地反映出各任务之间的依赖关系, 难以确定整个项目的关键所在, 也不能反映计划中有潜力的部分。

(2) PERT 图: PERT 图是一个有向图, 图中的箭头表示任务, 它可以标上完成该任务所需的时间, 图中的节点表示流入节点的任务的结束, 并开始流出节点的任务, 这里我们把节点成为事件。只有当流入该节点的所有任务都结束时, 节点所表示的事件才出现, 流出节点的任务才可以开始。事件本身不消耗时间和资源, 它仅表示某个时间点。一个事件有一个事件号和出现该事件的最早时刻和最迟时刻。最早时刻表示在此时刻之前从该事件出发的任务不可能开始; 最迟时刻表示从该事件出发的任务必须在此时刻之前开始, 否则整个工程就不能如期完成。每个任务还可以有一个松弛时间 (Slack Time), 表示在不影响整个工期的前提下, 完成该任务有多少机动余地。为了表示任务间的关系, 图中还可以加入一些空任务 (用虚线箭头表示), 完成空任务的时间为 0。

图 4-2 是 PERT 图的一个实例。不难看出, 图 4-2 中的松弛时间为 0 的这些任务是完成整个工程的关键路径, 其事件流为: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 6 \rightarrow 8 \rightarrow 10 \rightarrow 11$ 。

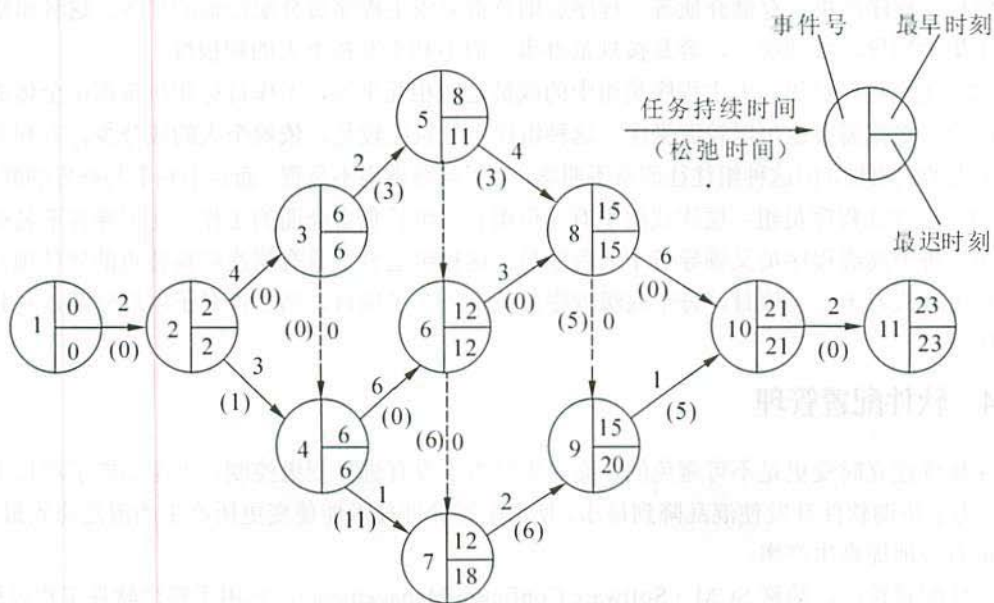
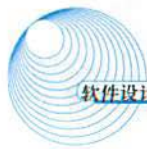


图 4-2 PERT 图实例

PERT 图不仅给出了每个任务的开始时间、结束时间和完成该任务所需的时间, 还给出了任务之间的关系, 即哪些任务完成后才能开始另外一些任务, 以及如期完成整个工程的关键路径。图中的松弛时间则反映了完成某些任务时可以推迟其开始时间或延长其所需完成的时间。



但是 PERT 图不能反映任务之间的并行关系。

4. 人员管理

合理地组织好参加软件项目的人员。有利于发挥每个人的作用,有利于软件项目的成功开发。在人员组织时,应考虑软件项目的特点、软件人员的素质等多方面的因素。

我们可以按软件项目对软件人员分组,如需求分析组、设计组、编码组、测试组、维护组等,为了控制软件的质量,还可以有质量保证组。

程序设计小组的组织形式也可以有多种,如主程序员组、无主程序员组、层次式程序员组等。

(1) 主程序员组:主程序员组由一名主程序员、一名后备程序员(back up programmer)、一名资料员和若干名程序员组成。主程序员由经验丰富、能力强的高级程序员担任,他是该组织的技术领导和项目负责人,全面负责软件项目的开发。后备程序员是主程序员的助手,协助主程序员工作,必要时能代替主程序员工作。资料员负责保存和管理所有的软件配置元素,如文档资料、程序清单、存储介质等。程序员则负责完成主程序员分配给他的任务。这种组织形式便于集中领导,步调统一,容易按规范办事,但不利于发挥个人的积极性。

(2) 无主程序员组:无主程序员组中的成员之间相互平等,工作目标和决策都由全体成员民主讨论,根据需要也可以轮流坐庄。这种组民主气氛比较足,依赖个人的成分少,有利于发挥每个人的积极性,但这种组往往职责不明确,出了问题谁也不负责,而且不利于与外界的联系。

(3) 层次式程序员组:层次式组中有一位组长,组长负责全面的工作,他领导若干名高级程序员,每个高级程序员又领导若干名程序员。这种组适合于具有层次结构特点的软件项目,该项目可分成若干个子项目,每个高级程序员负责一个子项目,然后再对子项目分解,并分配给程序员。

4.1.4 软件配置管理

在软件建立时变更是不可避免的,而变更时由于没有进行变更控制,可能加剧了项目中的混乱。为了协调软件开发使混乱降到最小,使用配置管理技术能使变更所产生的混乱降到最小,并能最有效地提高生产率。

软件配置管理,简称 SCM (Software Configure Management),它用于整个软件工程过程。其主要目标是:标识变更、控制变更、确保变更正确地实现、报告有关变更。SCM 是一组管理整个软件生存期各阶段中变更的活动。

1. 基线

基线是软件生存期中各开发阶段的一个特定点,它的作用是把开发各阶段工作的划分更加

明确化,使本来连续的工作在这些点上断开,以便于检查与肯定阶段成果。因此基线可以作为一个检查点,在开发过程中,当采用的基线发生错误时,我们可以知道所处的位置,返回到最近和最恰当的基线上。

2. 软件配置项

软件配置项 (Software Configure Item, SCI) 是软件工程中产生的信息项,它是配置管理的基本单位,对已经成为基线的 SCI,虽然可以修改,但必须按照一个特殊的、正式的过程进行评估,确认每一处修改。以下的 SCI 是 SCM 的对象,并可形成基线:

- 系统规格说明书;
- 软件项目实施计划;
- 软件需求规格说明书;
- 设计规格说明书 (数据设计、体系结构设计、模块设计、接口设计、对象描述 (使用面向对象技术));
- 源代码清单;
- 测试计划和过程、测试用例和测试结果记录;
- 操作和安装手册;
- 可执行程序 (可执行程序模块、连接模块);
- 数据库描述 (模式和文件结果、初始内容);
- 用户手册;
- 维护文档 (软件问题报告、维护请求、工程变更次序);
- 软件工程标准;
- 项目开发小结。

此外,许多软件工程组织把配置控制之下的软件工具,即编辑程序、编译程序、其他 CASE 工具的特定版本都作为软件配置的一部分列入其中。

3. 版本控制

软件配置实际上是一个动态的概念,它一方面随着软件生存期向前推进,SCI 的数量在不断增多,一些文档经过转换生成另一些文档,并产生一些信息。另一方面又随时会有新的变更出现,形成新的版本。

表达系统不同版本的一种表示如图 4-3 所示的演变图,在图中各个节点是一个完全的软件版本。软件的每一个版本都是 SCI (源代码、文档、数据) 的一个汇集,且各个版本都可能由不同的变种组成。

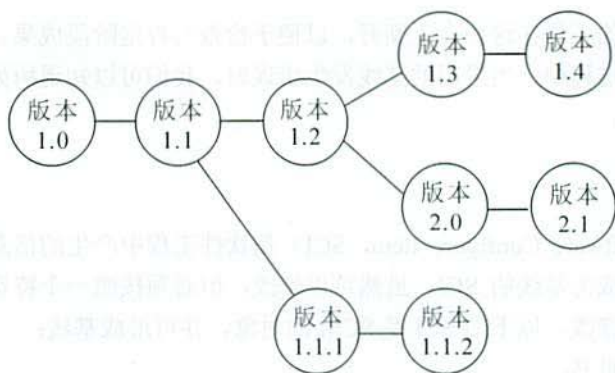
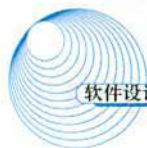


图 4-3 版本演变

4. 变更控制

软件工程过程中某一阶段的变更,均要引起软件配置的变更,这种变更必须严格加以控制和管理,保持修改信息,并把精确、清晰的信息传递到软件工程过程的下一步骤。

对于一个大型软件来说,不加控制的变更很快就會引起混乱。因此变更控制是一项最重要的软件配置任务。为有效地实现变更控制需借助于配置数据库和基线的概念。

配置数据库可以分为 3 类。

(1) 开发库:专供开发人员使用,其中的信息可能频繁地做修改,对其控制相当宽松。

(2) 受控库:在生存期某一阶段工作结束时发布的阶段产品,这些是与软件开发工作相关的计算机可读信息和人工可读信息。软件配置管理正是对受控库中的各个软件项进行管理,受控库也称为软件配置库。

(3) 产品库:在开发的软件产品完成系统测试后,作为最终产品存入产品库,等待交付用户或现场安装。

4.1.5 软件工具与软件开发环境

用来辅助软件开发、运行、维护、管理、支持等过程中的活动的软件称为软件工具。早期的软件工具主要用来辅助程序员编程,如编辑程序、编译程序、排错程序等。在提出了软件工程的概念以后,又出现了软件生存周期的概念,出现了许多开发模型和开发方法,同时软件管理也开始受到人们的重视。与此同时,出现了一批软件工具来辅助软件工程的实施,这些软件工具涉及到软件开发、维护、管理过程中的各项活动,并辅助这些活动高效、高质量地进行。因此,软件工具通常也称为 CASE (Computer Aided Software Engineering, 计算机辅助软件工程) 工具。

软件开发环境是指支持软件产品开发的软件系统,它由软件工具集和环境集成机制构成。工具集应包括支持软件开发相关过程、活动、任务的软件工具,以对软件开发提供全面的支持。环境集成机制为工具集成和软件开发、维护和管理提供统一的支持,它通常包括数据集成、控制集成和界面集成。

1. 软件工具

软件工具的种类繁多,很难有一种统一的分类方法,通常从不同的观点来进行分类。由于大多数软件工具仅支持软件生存周期过程中的某些特定的活动,所以通常可以按软件过程的活动来进行分类。

1) 软件开发工具

对应于软件开发过程的各种活动,软件开发工具通常有需求分析工具、设计工具、编码与排错工具、测试工具等。

(1) 需求分析工具:用以辅助软件需求分析活动的软件称为需求分析工具,它辅助系统分析员从需求定义出发,生成完整的、清晰的、一致的功能规范(Functional Specification)。功能规范是软件所要完成的功能的准确而完整的陈述,它描述该软件要做什么及只做什么。

按照需求定义的方法可将需求分析工具分为基于自然语言或图形描述的工具和基于形式化需求定义语言的工具。

(2) 设计工具:用以辅助软件设计活动的软件称为设计工具,它辅助设计人员从软件功能规范出发,得到相应的设计规范(Design Specification)。对应于概要设计活动和详细设计活动,设计工具通常可分为概要设计工具和详细设计工具。

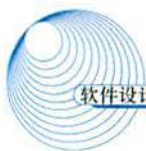
概要设计工具用以辅助设计人员设计目标软件的体系结构、控制结构和数据结构。详细设计工具用以辅助设计人员设计模块的算法和内部实现细节。除此之外还有基于形式化描述的设计工具和面向对象分析与设计工具。

(3) 编码与排错工具:辅助程序员进行编码活动的工具有编码工具和排错工具。编码工具辅助程序员用某种程序设计语言编制源程序,并对源程序进行翻译,最终转换成可执行的代码。因此,编码工具通常与编码所使用的程序语言密切相关。排错工具用来辅助程序员寻找源程序中错误的性质和原因,并确定其出错的位置。

2) 软件维护工具

辅助软件维护过程中活动的软件称为软件维护工具,它辅助维护人员对软件代码及其文档进行各种维护活动。软件维护工具主要有版本控制工具、文档分析工具、开发信息库工具、逆向工程工具和再工程工具。

(1) 版本控制工具:在软件开发和维护过程中一个软件往往有多个版本,版本控制工具用来存储、更新、恢复和管理一个软件的多个版本。



(2) 文档分析工具: 文档分析工具用来对软件开发过程中形成的文档进行分析, 给出软件维护活动所需的维护信息。例如, 基于数据流图的需求文档分析工具可给出对数据流图的某个成分(如加工)进行维护时的影响范围及被影响范围, 以便在该成分修改的同时考虑其影响范围内的其他成分是否也要修改。除此之外, 文档分析工具还可以得到被分析的文档的有关信息, 如文档各种成分的个数、定义及引用情况等。

(3) 开发信息库工具: 开发信息库工具用来维护软件项目的开发信息, 包括对象、模块等。它记录每个对象的修改信息(已确定的错误及重要改动)和其他变形(如抽象数据结构的多种实现), 还必须维护对象和与之有关信息之间的关系。

(4) 逆向工程工具: 逆向工程工具辅助软件人员将某种形式表示的软件(源程序)转换成更高抽象形式表示的软件。这种工具力图恢复源程序的设计信息, 使软件变得更容易理解。逆向工程工具分为静态的和动态的两种。

(5) 再工程工具: 再工程工具用来支持重构一个功能和性能更为完善的软件系统。目前的再工程工具主要集中在代码重构、程序结构重构和数据结构重构等方面。

3) 软件管理和软件支持工具

软件管理和软件支持工具用来辅助管理人员和软件支持人员的管理活动和支持活动, 以确保软件高质量地完成。辅助软件管理和软件支持的工具很多, 其中常用的工具有项目管理工具、配置管理工具和软件评价工具。

(1) 项目管理工具: 项目管理工具用来辅助软件的项目管理活动。通常项目管理活动包括项目的计划、调度、通信、成本估算、资源分配及质量控制等。一个项目管理工具通常把重点放在某一个或某几个特定的管理环节上, 而不提供对管理活动包罗万象的支持。

(2) 配置管理工具: 配置管理工具用来辅助完成软件配置项的标识、版本控制、变化控制、审计和状态统计等基本任务, 使得各配置项的存取、修改和系统生成易于实现, 从而简化了审计过程, 改进状态统计, 减少错误, 提高系统的质量。

(3) 软件评价工具: 软件评价工具用来辅助管理人员进行软件质量保证的有关活动。它通常可以按照某个软件质量模型(如 McCall 软件质量模型、ISO 软件质量度量模型等)对被评价的软件进行度量, 然后得到相关的软件评价报告。软件评价工具有助于软件的质量控制, 对确保软件的质量有重要的作用。

软件工具的种类非常多, 上面只列举了一些主要的也是常用的工具。软件工具可以从不同的角度进行分类, 上述分类只是其中较流行的一种, 而且这种分类并非是严格的, 有些工具可以属于这一类, 也可以属于另一类。

2. 软件开发环境

软件开发环境(Software Development Environment)是支持软件产品开发的软件系统。它由软

件工具集和环境集成机制构成,前者用来支持软件开发的相关过程、活动和任务;后者为工具集成和软件开发、维护和管理提供统一的支持,它通常包括数据集成、控制集成和界面集成。通过环境集成机制,各工具用统一的数据接口规范存储或访问环境信息库;各工具采用统一的界面形式,保证各工具界面的一致性,同时为各工具或开发活动之间的通信、切换、调度和协同工作提供支持。在软件开发环境中进行软件开发,可以使用环境中提供的各种工具,同时在环境信息库的支持下,一个工具所产生的结果信息可以为其他工具利用,使得软件开发的各项活动得到连续的支持,从而大大提高了软件的开发效率,提高了软件的质量。

软件开发环境的特征是:

- (1) 环境的服务是集成的。软件开发环境应支持多种集成机制,如平台集成、数据集成、界面集成、控制集成和过程集成等。
- (2) 环境应支持小组工作方式,并为其提供配置管理。
- (3) 环境的服务可用于支持各种软件开发活动,包括分析、设计、编程、测试、调试、文档等。

集成型开发环境是一种把支持多种软件开发方法和开发模型的软件工具集成在一起的软件开发环境。这种环境应该具有开放性和可剪裁性。开放性为环境外的工具集成到环境中来提供了方便,可剪裁性可根据不同的应用和不同的用户需求进行剪裁,以形成特定的开发环境。

4.1.6 软件过程管理

软件过程是软件生存周期中的一系列相关活动,即用于开发和维护软件及相关产品的一系列活动。

软件产品的质量取决于软件开发过程,具有良好软件过程的软件机构能够开发出高质量的软件产品。这一点虽然早已为人们公认,但切实地在软件过程方面开展工作也只是十多年前的事。1987年在美国国防部支持下,卡内基·梅隆大学率先推出了软件工程评估项目的研究成果——软件过程能力成熟度模型 CMM。很快就引起了软件界的广泛关注,并在此后引发了一系列反响,以至在其基础上形成了国际标准(ISO/IEC 15504)。

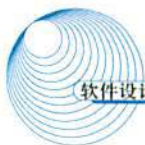
1. 软件过程评估的意义

软件过程评估是软件改进和软件能力评价的前提环节。

(1) 软件过程改进的需要

① 软件过程不断改进是软件工程的基本原理之一。1983年美国 TRW 公司的 B.W.Boehm 总结了该公司在 12 年内、总共花了 15 000 人年、先后开发五代指挥控制软件的经验,得出了以下 7 条原则:

- 按软件生存周期分阶段指定计划并认真实施;



- 逐阶段进行确认;
- 坚持严格的产品控制;
- 使用现代程序设计技术;
- 明确责任;
- 用人少而精;
- 不断改进开发过程。

这就是著名的软件工程七原理。由此可见,不断改进软件开发过程是软件工程的基本原理之一。

② 软件过程改进是软件生存周期的基本过程之一。软件工程界始终十分重视对软件过程的研究。20世纪70年代中期形成了软件生存周期的概念,1995年正式发布了一项国际标准,即ISO/IEC 12207 信息技术——软件生存周期过程,这是软件过程研究的一个重要成果。这项标准科学地定义了软件生存周期的过程,总共17个,其中一个就是改进过程。

(2) 降低软件风险的需要

① 软件采购者的需要。软件产品或软件服务的采购单位进行招标、选择承制者时,为了降低风险,需要对备选单位的软件过程能力进行评价,而这种评价的依据是对该单位的软件过程的评估结果。

② 软件承制者的需要。软件产品研制单位和软件服务单位在响应顾客的需要、进行投标时,为了降低风险,需要对自己的软件过程能力进行评价,避免承担力所不及的任务,而这种评价的依据仍然是根据实际需要,对相应软件过程的评估结果。

2. 软件能力成熟度模型 CMM 简介

软件能力成熟度模型是对软件组织进化阶段的描述,随着软件组织定义、实施、测量、控制和改进其软件过程,软件组织的能力经过这些阶段逐步前进。这个能力成熟度模型使软件组织能够较容易地确定其当前过程的成熟度并识别出其软件过程执行中的薄弱环节,确定对软件质量和过程改进最为关键的几个问题,从而形成对其过程的改进策略;软件组织只要关注并认真实施一组有限的关键实践活动,就能稳步地改善其全组织的软件过程,使全组织的软件过程能力持续增长。

软件过程能力成熟度模型 CMM 将软件过程能力成熟度划分为5级。

(1) 初始级(Initial): 软件过程是无序的,有时甚至是混乱的,对过程几乎没有定义,成功取决于个人努力。

(2) 可重复级(Repeatable): 建立了基本的项目管理过程来跟踪费用、进度和功能特性。制定了必要的过程纪律,能重复早先类似应用项目取得的成功。

(3) 已定义级(Defined): 已将软件管理和工程两方面的过程文档化、标准化,并综合成

该组织的标准软件过程。所有项目均使用经批准、剪裁的标准软件过程来开发和维护软件。

(4) 已管理级 (Managed): 收集对软件过程和产品质量的详细度量, 对软件过程和产品都有定量的理解与控制。

(5) 优化级 (Optimized): 过程的量化反馈和先进的新思想, 新技术促使过程不断改进。

每一个成熟度等级为过程改进达到下一个等级提供一个基础。当前一个等级没有达到时, 不能进入下一个等级。

3. 统一过程 (UP)

统一过程的特色是“用例和风险驱动, 以架构为中心, 迭代的增量开发过程”。迭代的意思是, 将整个软件开发项目划分为许多个小的“袖珍项目”, 每个“袖珍项目”都包含正常软件项目的元素: 计划、分析和设计、构造、集成和测试, 以及内部和外部发布。

在每个迭代中, 有 5 个核心 workflow: 捕获系统应该做什么的需求 workflow, 精华和结构化需求的分析 workflow, 用系统构架实现需求的设计 workflow, 构造软件的实现 workflow, 验证实现是否如期望那样工作的测试 workflow。

4. 极限编程 (XP)

XP 是一种轻量 (敏捷)、高效、低风险、柔性、可预测的、科学的软件开发方式。它由价值观、原则、实践和行为 4 个部分组成, 彼此相互依赖、关联, 并通过行为贯穿于整个生存周期。

(1) 四大价值观: 沟通、简单、反馈、勇气。

(2) 五个原则: 快速反馈、简单性假设、逐步修改、提倡更改、优质工作。

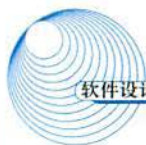
(3) 十二个最佳实践: 计划游戏 (快速制定计划、随着细节的不断变化而完善)、小型发布、隐喻 (找到合适的比喻传达信息)、简单设计、测试先行 (先写测试代码, 然后再编写程序)、重构、结对编程、集体代码所有制、持续集成、每周工作 40 个小时、现场客户、编码标准。

4.1.7 软件质量管理与质量保证

软件质量是指反映软件系统或软件产品满足规定或隐含需求的能力的特征和特性全体。软件质量保证是指为保证软件系统或软件产品充分满足用户要求的质量而进行的有计划、有组织的活动, 其目的是生产高质量的软件。

1. 软件质量特性

讨论软件质量首先要了解软件的质量特性。目前已经有多种软件质量模型来描述软件质量



特性。

1) ISO/IEC 9126 软件质量模型

ISO/IEC 9126 软件质量模型由 3 个层次组成：第一层是质量特性，第二层是质量子特性，第三层是度量指标。该模型的质量特性和质量子特性如图 4-4 所示。



图 4-4 ISO/IEC 软件质量模型

其中各质量特性和质量子特性的含义如下。

(1) 功能性 (Functionality): 与一组功能及其指定的性质的存在有关的一组属性。功能是指满足规定或隐含需求的那些功能。

① 适应性 (Suitability): 与对规定任务能否提供一组功能以及这组功能是否适合有关的软件属性。

② 准确性 (Accurateness): 与能够得到正确或相符的结果或效果有关的软件属性。

③ 互用性 (Interoperability): 与同其他指定系统进行交互操作的能力相关的软件属性。

④ 依从性 (Compliance): 使软件服从有关的标准、约定、法规及类似规定的软件属性。

⑤ 安全性 (Security): 与避免对程序及数据的非授权故意或意外访问的能力有关的软件属性。

(2) 可靠性 (Reliability): 与在规定的一段时间内和规定的条件下, 软件维持在其性能水平有关的能力。

① 成熟性 (Maturity): 与由软件故障引起实效的频度有关的软件属性。

② 容错性 (Fault Tolerance): 与在软件错误或违反指定接口的情况下, 维持指定的性能水平的能力有关的软件属性。

③ 易恢复性 (Recoverability): 与在故障发生后, 重新建立其性能水平并恢复直接受影响数据的能力, 以及为达到此目的所需的时间和努力有关的软件属性。

(3) 易使用性 (Usability): 与为使用所需的努力和由一组规定或隐含的用户对这样使用所作的个别评价有关的一组属性。

① 易理解性 (Understandability): 与用户为理解逻辑概念及其应用所付出的劳动有关的软件属性。

② 易学性 (Learnability): 与用户为学习其应用 (例如操作控制、输入、输出) 所付出的努力相关的软件属性。

③ 易操作性 (Operability): 与用户为进行操作和操作控制所付出的努力有关的软件属性。

(4) 效率 (Efficiency): 在规定条件下, 软件的性能水平与所用资源量之间的关系有关的软件属性。

① 时间特性 (Time Behavior): 与响应和处理时间以及软件执行其功能时的吞吐量有关的软件属性。

② 资源特性 (Resource Behavior): 与软件执行其功能时, 所使用的资源量以及使用资源的持续时间有关的软件属性。

(5) 可维护性 (Maintainability): 与进行规定的修改所需要的努力有关的一组属性。

① 易分析性 (Analyzability): 与为诊断缺陷或失效原因, 或为判定待修改的部分所需努力有关的软件属性。

② 易改变性 (Changeability): 与进行修改、排错或适应环境变换所需努力有关的软件属性。

③ 稳定性 (Stability): 与修改造成未预料效果的风险有关的软件属性。

④ 易测试性 (Testability): 为确认经修改软件所需努力有关的软件属性。

(6) 可移植性 (Portability): 与软件可从某一缓建转移到另一环境的能力有关的一组属性。

① 适应性 (Adaptability): 与一软件无需采用有别于为该软件准备的处理或手段就能适应不同的规定环境有关的软件属性。

② 易安装性 (Installability): 与在指定环境下安装软件所需努力有关的软件属性。

③ 一致性 (Conformance): 使软件服从与可移植性有关的标准或约定的软件属性。

④ 易替换性 (Replaceability): 与一软件在该软件环境中用来替代指定的其他软件的可能和努力有关的软件属性。

2) Mc Call 软件质量模型

Mc Call 软件质量模型从软件产品的运行、修正、转移等 3 个方面确定了 11 个质量特性 (见图 4-5)。Mc Call 也给出了一个 3 层模型框架, 第一层是质量特性, 第二层是评价准则, 第三层是度量指标。

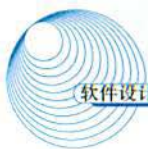


图 4-5 Mc Call 软件质量模型

2. 软件质量保证

软件质量保证是指为保证软件系统或软件产品充分满足用户要求的质量而进行的有计划、有组织的活动，其目的是生产高质量的软件。在软件质量方面强调 3 个要点：首先，软件必须满足用户规定的需求，与用户需求不一致的软件，就无质量可言；其次，软件应遵循规定标准所定义的一系列开发准则，不遵循这些准则的软件，其质量难以得到保证；第三，软件还应满足某些隐含的需求，例如希望有好的可理解性、可维护性等，而这些隐含的需求可能未被明确地写在用户规定的需求中，如果软件只满足它的显示需求而不满足其隐含需求，那么该软件的质量是令人置疑的。

软件质量保证包括了与以下 7 个主要活动相关的各种任务。

1) 应用技术方法

应该把软件质量设计到软件产品或软件系统中去，而不是在事后再施加质量保证。由于这个原因，软件质量保证首先从选择一组技术方法和工具开始，这些方法和工具帮助分析人员形成高质量的规格说明和高质量的设计。

2) 进行正式的技术评审

一旦形成了规格说明和设计，就要对它们进行质量评估。完成质量评估的中心活动是正式的技术评审。正式的技术评审是一种由技术人员实施的程式化会议，其唯一的目的是揭露质量问题。在多数情况下，评审能像测试一样有效地揭露软件中的缺陷。

3) 测试软件

软件测试组合了多种测试策略，这些测试策略带有一系列有助于有效地检测错误的测试用

例设计方法。许多软件开发人员把软件测试用作质量保证的“安全网”，也就是说，开发人员以为通过测试能揭露最多的错误，借此减轻对其他软件质量保证活动的需要。遗憾的是，即使是完成得很好的测试也不会像我们所期望的那样揭露所有的错误种类。

4) 标准的实施

对软件工程过程应用正式的开发标准和过程的程度在各公司中是不同的。多数情况，标准由客户或者某些章程确定。某些场合下标准是自己确定的。如果存在正式的标准，软件质量保证活动必须保证遵循这些标准。与标准是否一致的评估可以被软件开发人员作为正式技术评审的一部分来进行。

5) 控制变更

对软件质量的主要威胁来自“变更(change)”。对软件的每次变更都有可能引入错误或者引起传播错误的副作用。变更控制过程通过对变更的正式申请、评价变更的特性和控制变更的影响等直接提高软件的质量。变更控制应用软件开发期间和较后的软件维护阶段。

6) 度量(metrics)

度量是任何工程科学必备的活动。软件质量保证的重要目标是跟踪软件质量和评价方法及程序上的变更对软件质量的影响程度。要达到这个目标，应该搜集软件度量。软件度量包括某些技术上的度量和面向管理的度量。

7) 记录保存和报告

记录保存和报告为软件质量保证提供收集和传播软件质量保证信息的过程。评审、检查、变更控制、测试和其他软件质量保证活动的结果必须编程项目历史记录的一部分，并且应该把它传播给需要知道这些结果的开发人员。例如记录过程设计的每次正式技术评审结果，并把记录放置在文件夹中，该文件夹包含了有关模块的所有技术信息和软件质量保证信息。

3. 软件复杂性

软件度量的一个重要分支就是软件复杂性度量。对于软件复杂性度量的参数很多，主要有：

- 规模：即总共的指令数，或源程序行数。
- 难度：通常由程序中出现的操作数的数目所决定的量来表示。
- 结构：通常用与程序结构有关的度量来表示。
- 智能度：即算法的难易程度。

软件复杂性主要表现在程序的复杂性。程序的复杂性主要指模块内程序的复杂性。程序复杂性主要有以下几种度量方法：

1) 代码行度量法

度量程序的复杂性，最简单的方法就是统计程序的源代码行数。此方法的基本考虑是统计一个程序的源代码行数，并以源代码行数作为程序复杂性的度量。



2) McCabe 度量法

MCCabe 度量法是由 Thomas McCabe 提出的一种基于程序控制流的复杂性度量方法。McCabe 复杂性度量又称为环路度量,它认为程序的复杂性很大程度上取决于控制的复杂性。单一的顺序程序结构最为简单,循环和选择所构成的环路越多,程序就越复杂。这种方法以图论为工具,先画出程序图,然后用该图的环路数作为程序复杂性的度量值。程序图是退化的程序流程图,也就是说,把程序流程图中每个处理符号都退化成一个节点,原来连接不同处理符号的流线变成连接不同点的有向弧,这样得到的有向图就叫做程序图,如图 4-6 所示。程序图仅描述程序内部的控制流程,完全不表现对数据的具体操作以及分支和循环的具体条件。

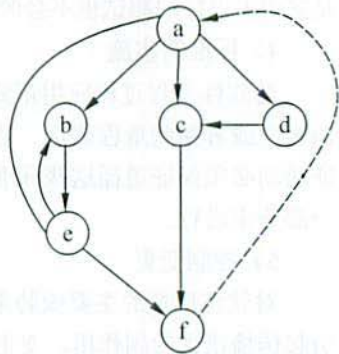


图 4-6 程序图的复杂性

根据图论,在一个强连通的有向图 G 中,环的个数 $V(G)$ 由以下公式给出:

$$V(G) = m - n + 2p$$

其中, $V(G)$ 是有向图 G 中的环路数, m 是图 G 中弧的个数, n 是图 G 中的节点数, p 是 G 中的强连通分量个数。在一个程序中,从程序图的入口点总能到达图中的任何一个节点,因此,程序总是连通的,但不是强连通的。为了使程序图成为强连通图,从图的入口点到出口点加一条用虚线表示的有向边,使图成为强连通图。这样就可以使用上式计算环路复杂性了。

以图 4-6 为例,其中节点数 $n=6$, 弧数 $m=9$, $p=1$, 则有:

$$V(G) = m - n + 2p = 9 - 6 + 2 = 5$$

即 McCabe 环路复杂的度量值为 5。

4. 软件评审

通常,把“质量”理解为“用户满意程度”。为了使得用户满意,有两个必要条件:设计的规格说明书符合用户的要求,这称为设计质量;程序按照设计规格说明所规定的情况正确执行,这称为程序质量。

软件的规格说明分为外部规格说明和内部规格说明。外部规格说明是从用户角度看的规格,包括硬件/软件系统设计、功能设计。内部规格说明是为了实现外部规格的更详细的规格,即软件模块结构与模块处理过程的设计。因此,内部规格说明是从开发者角度看的规格说明。设计质量是由外部规格说明决定的,程序是由内部规格说明决定的。

1) 设计质量的评审内容

设计质量评审的对象是在需求分析阶段产生的软件需求规格说明、数据需求规格说明,在软件概要设计阶段产生的软件概要设计说明书等。通常从以下几个方面进行评审:

(1) 评价软件的规格说明是否合乎用户的要求,即总体设计思想和设计方针是否明确;需求规格说明是否得到了用户或单位上级机关的批准;需求规格说明与软件的概要设计规格说明是否一致等。

(2) 评审可靠性。即是否能避免输入异常(错误或超载等)、硬件失效及软件失效所产生的失效,一旦发生应能及时采取代替手段或恢复手段。

(3) 评审保密措施实现情况。即是否对系统使用资格进行检查;是否对特定数据、特定功能的使用资格进行检查;在检查出有违反使用资格的情况后,能否向系统管理人员报告有关信息;是否提供对系统内重要数据加密的功能等。

(4) 评审操作特性实施情况。即操作命令和操作信息的恰当性、输入数据与输入控制语句的恰当性、输出数据的恰当性、应答时间的恰当性等。

(5) 评审性能实现情况。即是否达到所规定性能的目标值。

(6) 评审软件是否具有可修改性、可扩充性、可互换性和可移植性。

(7) 评审软件是否具有可测试性。

(8) 评审软件是否具有复用性。

2) 程序质量的评审内容

程序质量评审通常是从开发者的角度进行评审,与开发技术直接相关。它着眼于软件本身的结构、与运行环境的接口以及变更带来的影响而进行的评审活动。

(1) 软件的结构

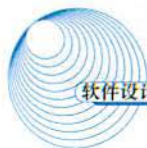
① 功能结构。在软件的各种结构中,功能结构是用户唯一能见到的结构。因此,功能结构是联系用户与开发者的规格说明,它在软件的设计中占有极其重要的地位。在评审软件的功能结构时,必须明确软件的数据结构。需要检查的项目有:

- 数据结构:包括数据名和定义、构成该数据的数据项、数据与数据间的关系。
- 功能结构:包括功能名和定义、构成该功能的子功能、功能与子功能之间的关系。
- 数据结构和功能结构之间的对应关系:包括数据元素与功能元素之间的对应关系、数据结构与功能结构的一致性。

② 功能的通用性。在软件的功能结构中,某些功能有时可以作为通用功能反复出现多次。从功能便于理解、增强软件的通用性及降低开发的工作量等观点出发,希望尽可能多地使功能通用化。需检查的项目包括抽象数据结构(抽象数据的名称和定义、抽象数据组成元素的定义);抽象功能结构。

③ 模块的层次。模块的层次是指程序模块结构。由于模块是功能的具体体现,所以模块层次应当根据功能层次来设计。

④ 模块结构。上述的模块层次结构是模块的静态结构。现在要检查模块的动态结构。模块分为处理模块和数据模块两类。模块间的动态结构也与这些模块分类有关。对这样的模块结



构进行检查的项目有:

- 控制流结构。规定了处理模块与处理模块之间的流程关系。检查处理模块之间的控制转移关系与控制转移形式(调用方式)。
- 数据流结构。规定了数据模块是如何被处理模块进行加工的流程关系。需检查处理模块与数据模块之间的对应关系、处理模块与数据模块之间的存取关系。
- 模块结构与功能结构之间的对应关系。包括功能结构与控制流结构的对应关系、功能结构与数据流结构之间的对应关系、每个模块的定义(功能、输入/输出数据)。

⑤ 处理过程的结构。处理过程是最基本的加工逻辑过程。对它的检查项目有:模块的功能结构与实现这些功能的处理过程的结构应明确对应;控制流应是结构化的;数据的结构与控制流之间的对应关系应是明确的,并且可根据这种对应关系来明确数据流程的关系;用于描述的术语标准化。

(2) 与运行环境的接口

运行环境包括硬件、其他软件 and 用户。主要的检查项目包括:

① 与硬件的接口。包括与硬件的接口约定,即对硬件的使用说明等所做出的规定、硬件故障时的处理和超载时的处理。

② 与用户的接口。包括与用户的接口约定,即输入数据的结构、输出数据的结构、异常输入时的处理、超载输入时的处理、用户存取资格的检查等。

5. 软件容错技术

提高软件质量和可靠性的技术大致可分为两类:一类是避开错误,即在开发的过程中不让差错潜入软件的技术;另一类是容错技术,即对某些无法避开的差错,使其影响减至最小的技术。

1) 容错软件定义

归纳容错软件的定义,有以下4种:

(1) 规定功能的软件,在一定程度上对自身错误的作用(软件错误)具有屏蔽能力,则称此软件为具有容错功能的软件,即容错软件。

(2) 规定功能的软件,在一定程度上能从错误状态自动恢复到正常状态,则称之为容错软件。

(3) 规定功能的软件,在因错误而发生错误时,仍然能在一定程度上完成预期的功能,则把该软件称为容错软件。

(4) 规定功能的软件,在一定程度上具有容错能力,则称之为容错软件。

2) 容错的一般方法

实现容错的主要手段是冗余。冗余是指对于实现系统规定功能是多余的那部分资源,包括硬件、软件、信息和时间。由于加入了这些资源,有可能使系统的可靠性得到较大的提高。通常冗余技术分为4类。

(1) 结构冗余。结构冗余是通常采用的冗余技术。按其工作方法,可以分为静态、动态和

混合冗余 3 种。

① 静态冗余。常用的有三模冗余 TMR (Triple Module Redundancy) 和多模冗余。静态冗余通过表决和比较来屏蔽系统中出现的错误。如三模冗余, 是对 3 个功能相同但由不同的人采用不同的方法开发出来的模块的运行结果通过表决, 以多数结果作为系统的最终结果。即如果模块中有一个出错, 这个错误能够被其他模块的正确结果“屏蔽”。由于无需对错误进行特别的测试, 也不必进行模块的切换就能实现容错, 故称为静态容错。

② 动态冗余。动态冗余的主要方式是多重模块待机储备。当系统测试到某工作模块出现错误时, 就用一个备用模块来顶替它并重新运行。这里包括检测、切换和恢复过程, 故称其为动态冗余。每当一个出错模块被其他备用模块顶替后, 冗余系统相当于进行了一次重构。各备用模块在其待机时可与主模块一同工作, 也可不工作。前者叫做热备份系统, 后者叫做冷备份系统。在热备份系统中备用模块在待机过程中其失效率为 0。

③ 混合冗余。它兼有静态冗余和动态冗余的长处。

(2) 信息冗余。为检测或纠正信息在运算或传输中的错误须外加一部分信息, 这种现象称为信息冗余。在通信和计算机系统中, 信息常以编码的形式出现。采用奇偶码、循环码等冗余码制式就可以发现甚至纠正这些错误。为了达到此目的, 这些码 (统称为误差校验码) 的码长远远大于不考虑误差校正时的码长, 增加了计算量和信道占用的时间。

(3) 时间冗余。时间冗余是指以重复执行指令或程序来消除瞬时错误带来的影响。多次重复执行不成功的情况, 通常的处理办法是发出中断, 转入错误处理程序, 或对程序进行复算, 或重新组合系统, 或放弃程序处理。在程序复算中比较常用的方法是程序滚回 (Program Rollback) 技术。

(4) 冗余附加技术。冗余附加技术是指为实现上述冗余技术所需的资源和技术。包括程序、指令、数据、存放和调动它们的空间和通道等。在屏蔽硬件错误的容错技术中, 冗余附加技术包括:

① 关键程序和数据的冗余存储和调用。

② 检测、表决、切换、重构、纠错和复算的实现。

在屏蔽软件错误的容错系统中, 冗余附加技术的构成包括:

① 冗余备份程序的存储及调用。

② 实现错误检测和错误恢复的程序。

③ 实现容错软件所需的固化程序。

4.2 系统分析基础知识

4.2.1 系统分析概述

1. 系统分析的目的和任务

系统分析的主要任务是对现行系统进一步详细调查, 将调查中所得到的文档资料集中, 对

组织内部整体管理状况和信息处理过程进行分析,为系统开发提供所需资料,并提交系统方案说明书。系统分析侧重于从业务全过程的角度进行分析,主要内容有:业务和数据的流程是否通畅,是否合理;数据、业务过程和组织管理之间的关系;原系统管理模式改革和新系统管理方法的实现是否具有可行性等。

确定的分析结果包括开发者对于现有组织管理状况的了解,用户对信息系统功能的需求,数据和业务流程,管理功能和管理数据指标体系以及新系统拟改动和新增的管理模型等。

最后提出信息系统的各种设想和方案,并对所有的设想和方案进行分析、研究、比较、判断和选择,获得一个最优的新系统的逻辑模型,并在用户理解计算机系统的工作流程和处理方式的情况下,将它明确地表达成书面资料——系统分析报告,即系统方案说明书。

2. 系统分析的主要步骤

企业信息系统是一个具有业务复杂性和技术复杂性的大系统,为了使目标系统既能实现当前系统的基本职能又能有所改进和提高,系统开发人员首先必须理解并描述出已经实际存在的当前系统,然后进行改进,从而创造出基于当前系统又高于当前系统的目标系统,即新系统。

系统分析过程一般按如图 4-7 所示的逻辑进行。

- (1) 认识、理解当前的现实环境,获得当前系统的“物理模型”。
- (2) 从当前系统的“物理模型”抽象出当前系统的“逻辑模型”。
- (3) 对当前系统的“逻辑模型”进行分析和优化,建立目标系统的“逻辑模型”。
- (4) 对目标系统的逻辑模型具体化(物理化),建立目标系统的物理模型。

系统开发的目的是把现有系统的物理模型转化为目标系统的物理模型,即图 4-7 中双虚线所描述的路径,而系统分析阶段的结果是得到目标系统的逻辑模型。逻辑模型反映了系统的功能和性质,而物理模型反映的是系统的某一种具体实现方案。

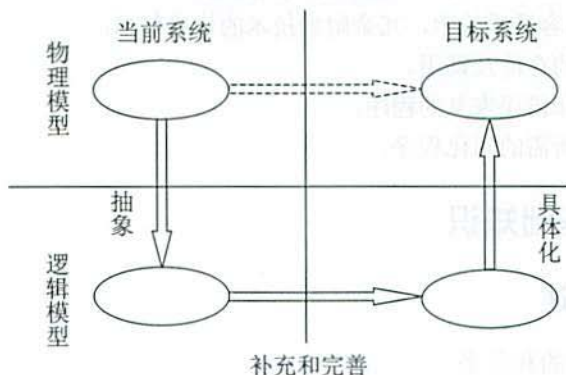


图 4-7 系统分析过程图

按照图 4-7 所示, 可将系统分析阶段的主要工作步骤分为:

- (1) 对当前系统进行详细调查, 收集数据。
- (2) 建立当前系统的逻辑模型。
- (3) 对现状进行分析, 提出改进意见和新系统应达到的目标。
- (4) 建立新系统的逻辑模型。
- (5) 编写系统方案说明书。

4.2.2 结构化分析方法

结构化分析 (Structured Analysis) 方法简称 SA 方法, 它是一种面向数据流的需求分析方法, 适用于分析大型数据处理系统。此种方法简单、实用, 现在已经得到广泛的使用。

结构化分析方法的基本思想是自顶向下逐层分解。分解和抽象是人们控制问题复杂性的两种基本手段。对于一个复杂的问题, 人们很难一下子考虑问题的所有方面和全部细节, 通常可以把一个大问题分解成若干个小问题, 每个小问题再分解成若干个更小的问题, 经过多次逐层分解, 每个最底层的问题都是足够简单、容易解决的, 于是复杂的问题也就迎刃而解了。这个过程就是分解的过程。

SA 方法的分析结果由以下几部分组成: 一套分层的数据流图、一本数据词典、一组小说明 (也称加工逻辑说明)、补充材料。

1. 数据流图

数据流图或称数据流程图 (Data Flow Diagram, DFD), 是一种便于用户理解、分析系统数据流的图形工具。它摆脱了系统的物理内容, 精确地在逻辑上描述系统的功能、输入、输出和数据存储等, 是系统逻辑模型的重要组成部分。

1) DFD 的基本成分

DFD 的基本成分及其图形表示方法如图 4-8 所示。

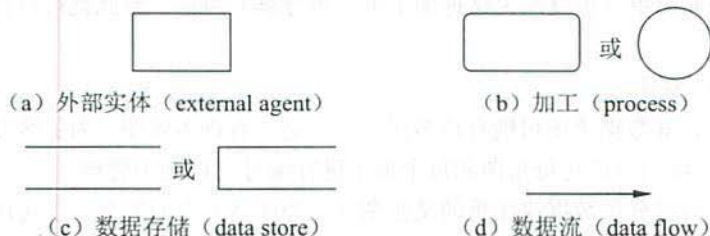
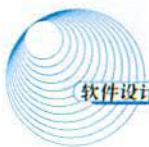


图 4-8 DFD 的基本成分



(1) 数据流。数据流由一组固定成分的数据组成,表示数据的流向。值得注意的是 DFD 中描述的是数据流,而不是控制流。除了流向数据存储或从数据存储流出的数据流不必命名外,每个数据流都必须有一个合适的名字,以反映该数据流的含义。

(2) 加工。加工描述了输入数据流到输出数据流之间的变换,也就是输入数据流经过什么处理后变成了输出数据流。每个加工有一个名字和编号。编号能反映出该加工位于分层 DFD 中的哪个层次和哪张图中,也能够看出它是哪个加工分解出来的子加工。

(3) 数据存储。数据存储用来表示暂时存储的数据,每个数据存储都有一个名字。

(4) 外部实体。外部实体是指存在于软件系统之外的人员或组织。它指出系统所需数据的发源地和系统所产生的数据的归宿地。

2) 分层数据流图的画法

(1) 画系统的输入和输出:把整个软件系统看作一个大的加工,然后根据系统从哪些外部实体接收数据流,以及系统发送数据流到哪些外部实体,就可以画出系统的输入和输出图,这张图称为顶层图。

(2) 画系统的内部:将顶层图的加工分解成若干个加工,并用数据流将这些加工连接起来,使得顶层图中的输入数据经过若干个加工处理后变换成顶层图的输出数据流。这张图称为 0 层图。从一个加工画出一张数据流图的过程实际上就是对这个加工的分解。

可以用下述的方法来确定加工:在数据流的组成或值发生变化的地方应画一个加工,这个加工的功能就是实现这一变化;也可根据系统的功能确定加工。

确定数据流的方法可以是:当用户把若干个数据看作一个单位来处理(这些数据一起到达,一起加工)时,可把这些数据看成一个数据流。

对于一些以后某个时间要使用的数据可以组织成一个数据存储来表示。

(3) 画加工的内部:把每个加工看作一个小系统,该加工的输入/输出数据流看成小系统的输入/输出数据流。于是可以用画 0 层图同样的方法画出每个加工的 DFD 子图。

(4) 对第(3)步分解出来的 DFD 子图中的每个加工,重复第(3)步的分解,直至图中的尚未分解的加工都足够简单(也就是说这种加工不必再分解)为止。至此就可以得到了一套分层数据流图。

3) 对图和加工进行编号

对于一个软件系统,其数据流图可能有许多层,每一层又有许多张图。为了区分不同的加工和不同的 DFD 子图,我们应该对每张图和每个加工进行编号,以利于管理。

(1) 父图与子图。假设分层数据流图里的某张图(记为图 A)中的某个加工可用另一张图(记为图 B)来分解,我们称图 A 是图 B 的父图,图 B 是图 A 的子图。在一张图中,有些加工需要进一步分解,有些加工则不必分解。因此,如果父图中有 n 个加工,那么它可以有 $0 \sim n$ 张子图(这些子图位于同一层),但每张子图都只对应于一张父图。

(2) 编号

- ① 顶层图只有一张，图中的加工也只有一个，所以不必编号。
- ② 0 层图只有一张，图中的加工号可以分别是“0.1, 0.2, …”或者是“1, 2, …”。
- ③ 子图号就是父图中被分解的加工号。
- ④ 子图中的加工号由图号、圆点和序号组成。

例如，某图中的某加工号为 2.4，这个加工分解出来的子图号就是 2.4，子图中的加工号分别为“2.4.1, 2.4.2, …”。

4) 实例

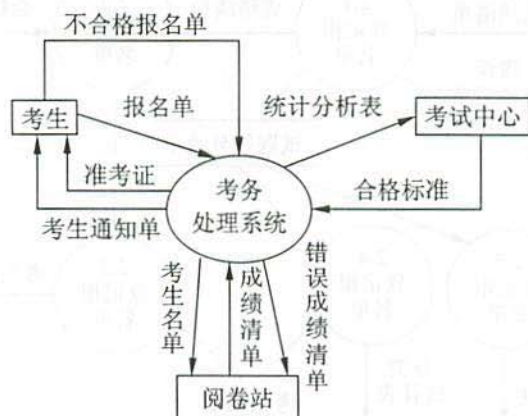
某考务处理系统有如下功能：

- (1) 对考生送来的报名单进行检查。
- (2) 对合格的报名单进行检查。
- (3) 对阅卷站送来的成绩清单进行检查，并根据考试中心指定的合格标准审定合格者。
- (4) 制作考生通知单（内含成绩合格/不合格标志）送给考生。
- (5) 按地区、年龄、文化程度、职业、考试级别等进行成绩分类统计和试题难度分析，产生统计分析表。

该考务处理系统的分层数据流图如图 4-9 所示。

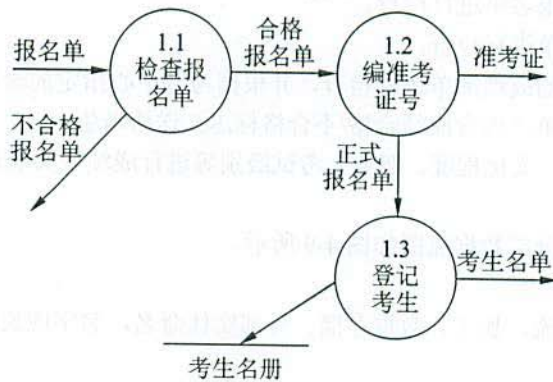
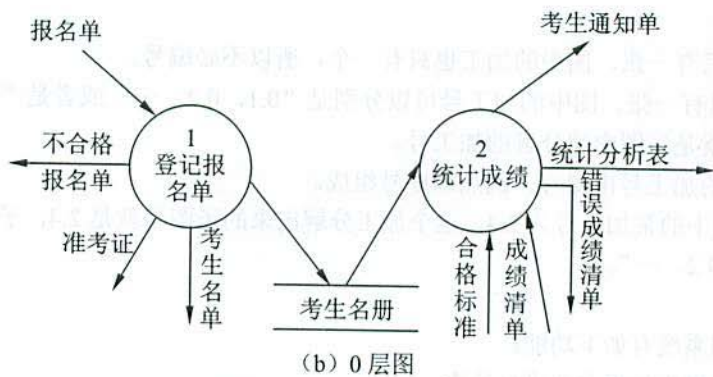
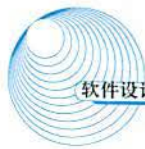
5) 应注意的问题

- (1) 适当地为数据流、加工、数据存储、外部实体命名，名字应反映该成分的实际含义，避免空洞的名字。
- (2) 画数据流而不要画控制流。
- (3) 一个加工的输出数据流不应与输入数据流同名，即使它们的组成成分相同。

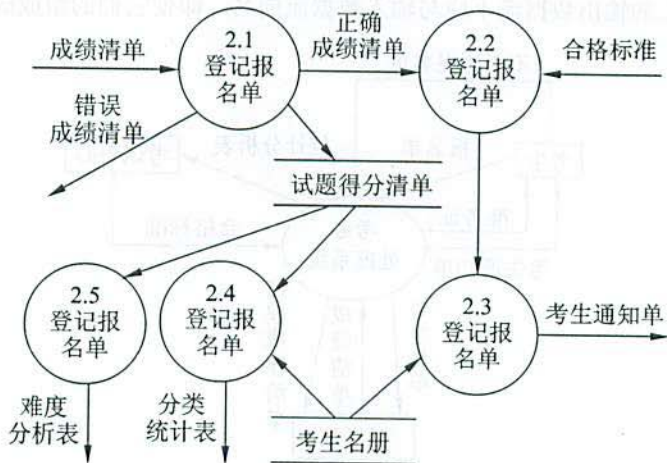


(a) 顶层图

图 4-9 考务处理系统分层数据流图



(c) 图 1



(d) 图 2

图 4-9 (续)

(4) 允许一个加工有多条数据流流向另一个加工, 也允许一个加工有两个相同的输出数据流流向两个不同的加工。

(5) 保持父图与子图平衡。也就是说, 父图中某加工的输入/输出数据流必须与它的子图的输入/输出数据流在数量和名字上相同。值得注意的是, 如果父图的一个输入(或输出)数据流对应于子图中几个输入(或输出)数据流, 而子图中组成这些数据流的数据项全体正好是父图中的这一个数据流, 那么它们仍然算是平衡的。

(6) 在自顶向下的分解过程中, 若一个数据存储首次出现时只与一个加工有关, 那么这个数据存储应作为这个加工的内部文件而不必画出。

(7) 保持数据守恒。也就是说, 一个加工所有输出数据流中的数据必须能从该加工的输入数据流中直接获得, 或者是通过该加工能产生的数据。

(8) 每个加工必须既有输入数据流, 又有输出数据流。

(9) 在整套数据流图中, 每个数据存储必须既有读的数据流, 又有写的数据流。但在某一子图中可能只有读没有写, 或者只有写没有读。

2. 数据词典(DD)

数据流图描述了系统的分解, 但没有对图中各成分进行说明。数据词典就是为数据流图中的每个数据流、文件、加工以及组成数据流或文件的数据项作出说明。其中对加工的描述称为“小说明”, 也可以称为“加工逻辑说明”。

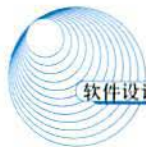
1) 数据词典的内容

数据词典有, 数据流、数据项、数据存储和基本加工 4 类条目。数据项是组成数据流和数据存储的最小元素。源点、终点不在系统之内, 故一般不在词典中说明。

(1) 数据流条目: 数据流条目给出了 DFD 中数据流的定义, 通常列出该数据流的各组成数据项。在定义数据流或数据存储组成时, 使用表 4-1 给出的符号。

表 4-1 在数据词典的定义式中出现的符号

符号	含义	举例及说明
=	被定义为	
+	与	$x = a + b$, 表示 x 由 a 和 b 组成
[... ...]	或	$x = [a b]$, 表示 x 由 a 或 b 组成
{...}	重复	$x = \{a\}$, 表示 x 由 0 个或多个 a 组成
$m\{...\}n$ 或 $\{...\}_m^n$	重复	$x = 2\{a\}5$ 或 $x = \{...\}_2^5$, 表示 x 中最少出现 2 次 a , 最多出现 5 次 a 。 5、2 为重复次数的上、下限
(...)	可选	$x = (a)$ 表示 a 可在 x 中出现, 也可不出现
"..."	基本数据元素	$x = "a"$, 表示 x 是取值为字符 a 的数据元素
-	连接符	$x = 1..9$, 表示 x 可取 1 到 9 中任意一个值



- (2) 数据存储条目: 数据存储条目是对数据存储的定义。
- (3) 数据项条目: 数据项条目是不可再分解的数据单位。
- (4) 加工条目: 加工条目是用来说明 DFD 中基本加工的处理逻辑的, 由于上层的加工是由下层的基本加工分解而来, 只要有了基本加工的说明, 就可理解其他加工。

2) 数据词典管理

词典管理主要是把词典条目按照某种格式组织后存储在词典中, 并提供排序、查找、统计等功能。如果数据流条目包含了来源和去向, 文件条目包含了读文件和写文件, 还可以检查数据词典与数据流图的一致性。

3. 加工逻辑的描述

加工逻辑也称为“小说明”。常用的加工逻辑描述方法有 3 种: 结构化语言、判定表和判定树。

1) 结构化语言

一种结构化语言(如结构化英语)是一种介于自然语言和形式化语言之间的半形式化语言, 是自然语言的一个受限子集。

结构化语言没有严格的语法, 它的结构通常可分为内层和外层。外层有严格的语法, 而内层的语法比较灵活, 可以接近于自然语言的描述。

(1) 外层: 用来描述控制结构, 采用顺序、选择和重复 3 种基本结构。

① 顺序结构: 是一组祈使语句、选择语句、重复语句的顺序排列。祈使语句指至少包含一个动词及一个名词, 指出要执行的动作及接受动作的对象。

② 选择结构: 一般用 IF-THEN-ELSE-ENDIF、CASE-OF-ENDCASE 等关键词。

③ 重复结构: 一般用 DO-WHILE-ENDDO、REPEAT-UNTIL 等关键词。

(2) 内层: 一般是采用祈使语句的自然语言短语, 使用数据词典中的名词和有限的自定义词, 其动词含义要具体, 尽量不用形容词和副词来修饰。还可使用一些简单的算法运算和逻辑运算符号。

2) 判定表

在有些情况下, 数据流图中的某个加工的一组动作依赖于多个逻辑条件的取值。这是用自然语言或结构化语言都不易于清楚地描述出来。而用判定表就能够清楚地表示复杂的条件组合与应做的动作之间的对应关系。

判定表由 4 部分组成, 用双线分割开 4 个区域, 如图 4-10 所示。

3) 判定树

判定树是判定表的变形, 一般情况下它比判定表更直观, 且易于理解和使用。

条件定义	条件取值的组合
动作定义	在各种取值的组合下应执行的动作

图 4-10 判定表结构

4.2.3 系统分析报告

完成整个系统分析阶段的工作后,作为工作成果,应提交一份完整的系统分析报告。系统分析报告一经确认,就成为具有约束力的指导性文件,成为下一阶段系统设计工作的依据和今后验收目标系统的检验标准。系统分析报告形成后必须组织各方面的人员(包括组织的领导、管理人员、专业技术人员、系统分析人员等)一起对已经形成的方案进行论证,尽可能发现其中的问题和不足。对于有争论的问题要重新核实当初的原始调查资料或进一步深入调查研究,对于重大的问题甚至可能需要调整或修改系统目标,重新进行系统分析。

在系统分析报告中,数据流图、数据字典和加工说明这3部分是主体,是系统分析报告中必不可少的组成部分。而其他各部分的内容,则应根据所开发的目标系统的规模、性质等具体情况酌情选用,不必生搬硬套。总之,系统分析报告必须简明扼要、抓住本质,反映出目标系统的全貌和开发人员的设想。

1. 系统分析报告的主要作用

- (1) 描述了目标系统的逻辑模型,作为开发人员进行系统设计和实施的基础。
- (2) 作为用户和开发人员之间的协议或合同,为双方的交流和监督提供基础。
- (3) 作为目标系统验收和评价的依据。

因此,系统分析报告是系统开发过程中的一份重要文档,必须完整一致、精确且简明易懂。

2. 系统分析报告的主要内容

一份完整的系统分析报告应该包括下述内容:

(1) 组织情况概述。

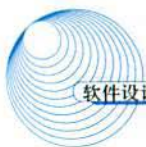
① 对分析对象的基本情况作概括性的描述,包括组织的结构、组织的目标、组织的工作过程和性质、业务功能。

② 系统于外部实体(其他系统或机构)间有哪些物质以及信息的交换关系和联系。

③ 参考资料和专门术语说明。

(2) 现行系统概述。

① 现行系统现状调查说明。通过现行系统的组织结构图、数据流图、概况表等,说明现行系统的目标、规模、主要功能、组织机构、业务流程、数据存储和数据流,以及存在的薄弱



环节。

② 系统需求说明。用户要求以及现行系统主要存在的问题等。

(3) 系统逻辑模型。

① 新系统拟定的业务流程及业务处理方式。提出明确的功能目标、并与现行系统进行比较和分析,重点要突出计算机处理的优越性。

② 新系统拟定的数据指标体系和分析优化后的数据流程,各个层次的数据流图、数据字典和加工说明,以及计算机系统将完成的工作部分。

③ 出错处理要求。

④ 其他特性要求。例如系统的输入输出格式、启动和退出等。

⑤ 遗留问题。根据目前条件,暂时不能满足的一些用户要求或设想,并提出今后解决的措施和途径。

(4) 新系统在各个业务处理缓解拟采用的管理方法、算法或模型。

(5) 与新的系统相配套的管理制度和运行体制的建立。

(6) 系统设计与实施的初步计划。

① 工作任务的分解。根据资源及其他条件确定各子系统开发的先后次序,在此基础上分解工作任务,落实到具体组织和个人。

② 根据系统开发资源与时间进度估计,指定进度安排计划。

③ 预算。对开发费用的进一步估计。

(7) 用户领导审批意见。

4.3 系统设计知识

4.3.1 系统设计的内容和步骤

在系统分析阶段,已经讲述了软件“做什么”的问题,并把这些需求怎样通过规格说明书描述出来,这也是目标系统的逻辑模型。进入了设计阶段,要把软件“做什么”的逻辑模型转换成“怎么做”的物理模型,即着手实现软件系统的需求。

系统设计的主要目的就是为系统制定蓝图,在各种技术和实施方法中权衡利弊,精心设计,合理使用各种资源,最终勾画出新系统的详细设计方案。

系统设计的主要内容包括新系统总体结构设计、代码设计、输出设计、输入设计、处理过程设计、数据存储设计、用户界面设计和安全控制设计等。

系统设计的基本任务大体上可以分为概要设计和详细设计两个步骤。

1. 概要设计的基本任务

1) 设计软件系统总体结构

其基本任务是：采用某种设计方法，将一个复杂的系统按功能划分成模块；确定每个模块的功能；确定模块之间的调用关系；确定模块之间的接口，即模块之间传递的信息；评价模块结构的质量。

软件系统总体结构的设计是概要设计关键的一步，直接影响到下一个阶段详细设计与编码的工作。软件系统的质量及一些整体特性都在软件系统总体结构的设计中决定。

2) 数据结构及数据库设计

(1) 数据结构的设计。逐步细化的方法也适用于数据结构的设计。在需求分析阶段，已经通过数据字典对数据的组成、操作约束、数据之间的关系等方面进行了描述，确定了数据的结构特性，在概要设计阶段要加以细化，详细设计阶段则规定具体的实现细节。在概要设计阶段，宜使用抽象的数据类型。

(2) 数据库的设计。数据库的设计指数据存储文件的设计，主要进行以下几方面设计：

① 概念设计。在数据分析的基础上，采用自底向上的方法从用户角度进行视图设计，一般用 ER 模型来表述数据模型。ER 模型既是设计数据库的基础，也是设计数据结构的基础。

② 逻辑设计。ER 模型是独立于数据库管理系统 (DBMS) 的，要结合具体的 DBMS 特征来建立数据库的逻辑结构。

③ 物理设计。对于不同的 DBMS，物理环境不同，提供的存储结构与存取方法各不相同。物理设计就是设计数据模式的一些物理细节，如数据项存储要求、存取方法、索引的建立等。

本节对数据库技术不做详细讨论，详细内容参见本书第七章数据库技术基础。

3) 编写概要设计文档

文档主要有概要设计说明书、数据库设计说明书、用户手册以及修订测试计划。

4) 评审

对设计部分是否完整地实现了需求中规定的功能、性能等要求，设计方法的可行性，关键的处理及内外部接口定义的正确性、有效性、各部分之间的一致性等全部进行评审。

2. 详细设计的基本任务

(1) 对每个模块进行详细的算法设计。用某种图形、表格、语言等工具将每个模块处理过程的详细算法描述出来。

(2) 对模块内的数据结构进行设计。

(3) 对数据库进行物理设计，即确定数据库的物理结构。

(4) 其他设计：根据软件系统的类型，还可能要进行以下设计：



① 代码设计。为了提高数据的输入、分类、存储、检索等操作,节约内存空间,对数据库中的某些数据项的值要进行代码设计。

② 输入/输出格式设计。

③ 人机交互设计。对于一个实时系统,用户与计算机频繁对话,因此要进行对话方式、内容、格式的具体设计。

(5) 编写详细设计说明书。

(6) 评审。对处理过程的算法和数据库的物理结构都要评审。

系统设计的结果是一系列的系统设计文件,这些文件是物理实现一个信息系统(包括硬件设备和编制软件程序)的重要基础。

4.3.2 系统设计的基本原理

1. 抽象

抽象是一种设计技术,重点说明一个实体的本质方面,而忽略或者掩盖不很重要或非本质的方面。抽象是一种重要的工具,用来将复杂的现象简化到可以分析、实验或者可以理解的程度。软件工程中从软件定义到软件开发要经历多个阶段。在这个过程中每前进一步都可看作是对软件解法的抽象层次的一次细化。抽象的最低层就是实现该软件的源程序代码。在进行模块化设计时也可以有多个抽象层次,最高抽象层次的模块用概括的方式叙述问题的解法,较低抽象层次的模块是对较高抽象层次模块对问题解法描述的细化。

2. 模块化

模块在程序中是数据说明、可执行语句等程序对象的集合,或者是单独命名和编址的元素,如高级语言中的过程、函数、子程序等。在软件的体系结构中,模块是可组合、分解和更换的单元。

模块化是指将一个待开发的软件分解成若干个小的简单部分——模块,每个模块可独立地开发、测试,最后组装成完整的程序。这是一种复杂问题的“分而治之”的原则。模块化的目的是使程序的结构清晰,容易阅读、理解、测试、修改。

3. 信息隐蔽

信息隐蔽是开发整体程序结构时使用的法则,即将每个程序的成分隐蔽或封装在一个单一的设计模块中,定义每一个模块时尽可能少地显露其内部的处理。在设计时首先列出一些可能发生变化的因素,在划分模块时将一个可能发生变化的因素隐蔽在某个模块的内部,使其他模块与这个因素无关。在这个因素发生变化时,我们只需修改含有这个因素的模块,而与其他模块无关。

信息隐蔽原则对提高软件的可修改性、可测试性和可移植性都有重要的作用。

4. 模块独立

模块独立是指每个模块完成一个相对独立的特定子功能，并且与其他模块之间的联系简单。衡量模块独立程度的标准有两个：耦合性和内聚性。

1) 耦合

耦合是指模块之间联系的紧密程度。耦合越高则模块的独立性越差。模块间耦合的高低取决于模块间接口的复杂性、调用的方式及传递的信息。模块的耦合有以下几种类型：

(1) 无直接耦合。指两个模块间没有直接的关系，他们分别从属于不同模块的控制与调用，它们之间不传递任何信息。因此，这种模块间耦合性最弱，模块独立性最高。

(2) 数据耦合。指两个模块之间有调用关系，传递的是简单的数据值，相当于高级语言中的值传递。这种耦合程度较低，模块的独立性较高。

(3) 标记耦合。指两个模块之间传递的数据结构，如高级语言中的数据组名、记录名、文件名等这些名字即为标记，其实传递的是这个数据结构的地址。

(4) 控制耦合。指一个模块调用另一个模块时，传递的是控制变量，被调模块通过该控制变量的值有选择地执行块内的某一功能。

(5) 公共耦合。指通过一个公共数据环境相互作用的那些模块之间的耦合。

(6) 内容耦合。这是程度最高的耦合。当一个模块直接使用另一个模块的内部数据，或通过非正常入口而转入另一个模块内部，这种模块之间的耦合为内容耦合，这种情况往往出现汇编程序设计中。

2) 内聚

内聚是指模块内部各元素之间联系的紧密程度。例如一个完成多个功能的模块的内聚度就比完成单一功能的模块的内聚度低。内聚度越低，模块的独立性越差。内聚性有以下几种类型：

(1) 偶然内聚。指一个模块内的各个处理元素之间没有任何联系。

(2) 逻辑内聚。指模块内执行几个逻辑上相似的功能，通过参数确定该模块完成哪一个功能。

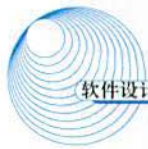
(3) 时间内聚。把需要同时执行的动作组合在一起形成的模块为时间内聚模块。

(4) 通信内聚。指模块内所有处理元素都在同一个数据结构上操作，或者指各处理使用相同的输入数据或者产生相同的输出数据。

(5) 顺序内聚。指一个模块中各个处理元素都密切相关于同一功能且必须顺序执行，前一功能元素的输出就是下一功能元素的输入。

(6) 功能内聚。这是最强的内聚。指模块内所有元素共同完成一个功能，缺一不可。

耦合性和内聚性是模块独立性的两个定性标准，将软件系统划分模块时，应尽量做到高内



聚、低耦合,提高模块的独立性。

4.3.3 系统总体结构设计

系统总体结构设计是要根据系统分析的要求和组织的实际情况来对新系统的总体结构形式和可利用的资源进行大致设计,这是一种宏观、总体上的设计和规划。下面介绍系统总体设计的主要内容。

1. 系统总体结构设计原则

为保证总体结构设计的顺利完成,主要应遵循以下几条原则:

(1) 分解——协调原则。整个系统是一个整体,具有整体目的和功能。但这些目的和功能的实现又是由相互联系的各个组成部分共同工作的结果。解决复杂问题的一个很重要的原则就是把它分解成多个小问题分别处理,在处理过程中根据系统总体要求协调各部门的关系。

(2) 自顶向下的原则。首先抓住系统总的功能目的,然后逐层分解,即先确定上层模块的功能,再确定下层模块的功能。

(3) 信息隐蔽、抽象的原则。上层模块只规定下层模块做什么和所属模块间的协调关系,但不规定怎么做,以保证各模块的相对独立性和内部结构的合理性,使得模块与模块之间层次分明,易于理解、实施和维护。

(4) 一致性原则。要保证整个软件设计过程中具有统一的规范、统一的标准和统一的文件模式等。

(5) 明确性原则。每个模块必须功能明确、接口明确,消除多重功能和无用接口。

(6) 模块之间的耦合尽可能小,模块的内聚度尽可能高。

(7) 模块的扇入系数和扇出系数要合理。一个模块直接调用其他模块的个数称为模块的扇出系数;反之,一个模块被其他模块调用时,直接调用的它的模块个数称为模块的扇入系数。模块的扇入、扇出系数必须适当。经验表明,一个设计成功的系统的平均扇入、扇出系数通常是3或4,一般不应超过7,否则会引起出错概率增大。但菜单调用型模块扇入与扇出系数可以大一些,公用模块扇入系数可以大一些。

(8) 模块的规模适当。过大的模块常常使系统分解得不充分,其内部可能包含了若干部分的功能,因此有必要进一步把原有的模块分解成若干功能尽可能单一的模块。但分解也必须适度,因为过小的模块有可能降低模块的独立性,造成系统接口的复杂性。

2. 子系统划分

1) 子系统划分的原则

为了便于今后系统开发和系统运行,子系统的划分应遵循如下几点原则:

(1) 子系统要具有相对独立性。子系统的划分,必须使得子系统的内部功能、信息等各方面的凝聚性较好。子系统独立可以减少子系统间的相互影响,有利于多人分工开发不同的模块,从而提高软件产品的生产率,保证软件产品的质量,同时也增强了系统的可维护性和适应性。

(2) 子系统之间数据的依赖性尽量小。子系统之间的联系要尽量减少,接口要简单明确。一个内部联系强的子系统对外部的联系必然很少,所以划分的时候应将联系较多者列入子系统内部,剩余的一些分散、跨度比较大的联系就成为这些子系统间的联系和接口。这样划分的子系统,将来调试、维护和运行都是非常方便的。

(3) 子系统划分的结果应使数据冗余较小。如果把相关的功能数据分布到各个不同的子系统中,则会有大量的原始数据需要调用,大量的中间结果需要保存和传递,大量计算工作将要重复进行。从而使得程序结构紊乱,数据冗余,不但给软件编制工作带来很大的困难,而且系统的工作效率也大大降低了。

(4) 子系统的设置应考虑今后管理发展的需要。子系统的设置仅依靠上述系统分析的结构是不够的,因为现存的系统由于各种原因,很可能没有考虑到一些高层次管理决策的要求。

(5) 子系统的划分应便于系统分阶段实现。信息系统的开发是一项较大的工程,它的实现一般都要分批进行,所以子系统的划分应能适应这种分期分批的实施。另外,子系统的划分还必须兼顾组织结构的要求。

(6) 子系统的划分应考虑到各类资源的充分利用。一个适当的子系统划分应该既考虑有利于各种设备资源在开发过程中的搭配使用,又要考虑到各类信息资源的合理分布和充分使用,以减少系统对网络资源的过分依赖,减少输入、输出、通信等设备的压力。

2) 子系统结构设计

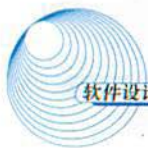
子系统结构设计任务是确定划分后的子系统模块结构,并画出模块结构图。这个过程中必须考虑以下几个问题。

- (1) 每个子系统如何划分成多个模块。
- (2) 如何确定子系统之间、模块之间传送的数据及其调用关系。
- (4) 如何评价并改进模块结构的质量。
- (4) 如何从数据流图导出模块结构图。

3. 系统模块结构设计

1) 模块的概念

模块是组成系统的基本单位,它的特点是可以组合、分解和更换。系统中任何一个处理功能都可以看成是一个模块。根据模块功能具体化程度的不同,可以分为逻辑模块和物理模块。在系统逻辑模型中定义的处理功能可视为逻辑模块。物理模块是逻辑模块的具体化,可以是一个计算机程序、子程序或若干条程序语句,也可以是人工过程的某项具体工作。



一个模块应具备以下4个要素:

- (1) 输入和输出: 模块的输入来源和输出去向都是同一个调用者, 即一个模块从调用者那里取得输入, 进行加工后再把输出返回给调用者。
- (2) 处理功能: 指模块把输入转换成输出所做的工作。
- (3) 内部数据: 指仅供该模块本身引用的数据。
- (4) 程序代码: 指用来实现模块功能的程序。

前两个要素是模块外部特性, 即反映了模块的外貌。后两个要素是模块的内部特性。在结构化设计中, 主要考虑的是模块的外部特性, 其内部特性只做必要了解, 具体的实现将在系统实施阶段完成。

2) 模块结构图

为了保证系统设计工作的顺利进行, 结构设计应遵循如下原则:

- (1) 所划分的模块其内部的凝聚性要强, 模块之间的联系要少, 即模块具有较强的独立性。
- (2) 模块之间的连接只能存在上下级之间的调用关系, 不能有同级之间的横向联系。
- (3) 整个系统呈树状结构, 不允许网状结构或交叉调用关系出现。
- (4) 所有模块(包括后继IPO图)都必须严格地分类编码并建立归档文件。

模块结构图主要关心的是模块的外部属性, 即上下级模块、同级模块之间的数据传递和调用关系, 并不关系模块的内部。

模块结构图是结构化设计中描述系统结构的图形工具。作为一种文档, 它必须严格地定义模块的名字、功能和接口, 同时还应当在模块结构图上反映出结构化设计思想。模块结构图由模块、调用、数据、控制和转接等5种基本符号组成, 如图4-11所示。



图4-11 模块结构图的基本符号

模块结构图基本符号的说明如下:

(1) 模块。这里所说的模块通常是指用一个名字就可以调用的一段程序语句。长方形中间标上能反映模块处理功能的模块名字。

(2) 调用。箭头总是由调用模块指向被调用模块, 但是应该理解被调用模块执行后又返回到调用模块。如果一个模块是否调用一个从属模块, 决定于调用模块内部的判断条件, 则该调用模块间的判断调用, 采用菱形符号表示。如果一个模块通过其内部的循环的功能来循环调用一个或多个从属模块, 则该调用称为循环调用, 用弧形箭头表示。判断调用和循环调用的表示

方法如图 4-12 所示。

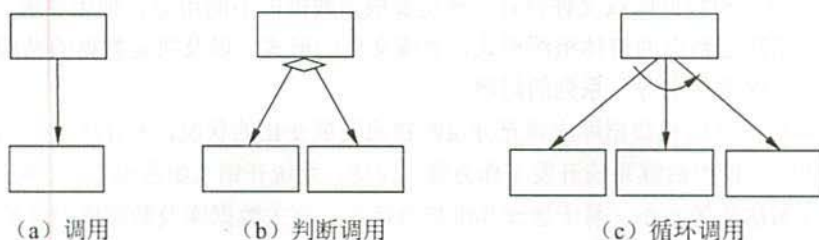


图 4-12 模块调用示例

(3) 数据。当一个模块调用另一个模块时，调用模块可以把数据传送到被调用模块供处理，而被调用模块又可以将处理的结构送回到被调用模块。在模块之间传送的数据，使用与调用箭头平行的带空心圆的箭头表示，并在旁边标上数据名。图 10-13 (a) 表示模块 A 调用模块 B 时，A 将数据 x、y 传送给 B，B 将处理结果数据 z 返回给 A。

(4) 控制信息。模块间有时还必须传送某些控制信息。例如，数据输入完成后给出的结束标志，文件读到末尾时所产生的文件结束标志等。控制信息与数据的主要区别是前者只反映数据的某种状态，不必进行处理。图 10-13 (b) 中“无此职工”就是用来表示送来的职工号有误的控制信息。

(5) 转接符号。当模块结构图在一张纸上画不下，需要转接到另一张纸上，或者为了避免图上线条交叉时，都可以使用转接符号，圆圈內加上标号，如图 4-14 所示。

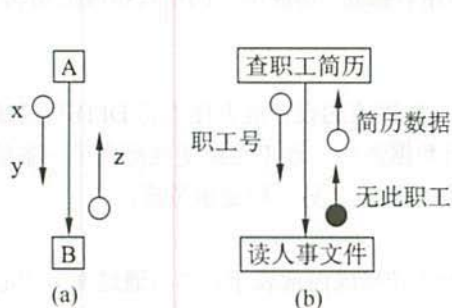


图 4-13 模块间的数据传递

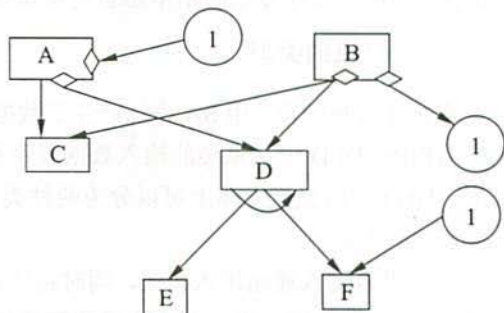
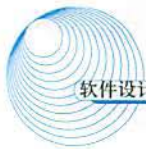


图 4-14 转接符号的使用

4. 数据存储设计

信息系统的主要任务是通过大量的数据获得管理所需要的信息，这就必须存储和管理大量的数据。因此建立一个良好的数据组织结构和数据库，使整个系统都可以迅速、方便、准确地



调用和管理所需的数据,是衡量信息系统开发工作好坏的主要指标之一。

数据结构组织和数据库或文件设计,就是要根据数据的不同用途、使用要求、统计渠道、安全保密性等来决定数据的整体组织形式、表或文件的形式,以及决定数据的结构、类别、载体、组织方式、保密级别等一系列的问题。

一个好的数据结构和数据库应该充分反映物流发展变化的状况,充分满足组织的各级管理要求。同时还应该使得后继系统开发工作方便、快捷、系统开销(如占用空间、网络传输频度、磁盘或光盘读写次数等)小、易于管理和维护等特点。有关数据库及数据库设计的相关内容可参见本书第七章。

在建立了数据的整体关系结构之后,剩下的就是要确定数据资源分布和安全保密属性了。其中数据资源的分布是针对分布数据库系统而言的,而安全保密属性的定义则是针对某些特殊信息,如财务数据等而言的。

(1) 数据资源分布:如果所规划和设计的系统是在网络环境之下,那么数据库设计必须考虑整个数据资源在网络各节点(包括网络服务器)上的分配问题。

(2) 数据的安全保密:一般数据库软件都提供定义数据安全保密性的基本功能。系统所提供的安全保密功能一般有 8 个等级(0~7 级),4 种不同方式(只读、只写、删除、修改),而且允许用户利用这 8 个等级的 4 种方式对每一个表自由地进行定义。

4.3.4 结构化设计方法

结构化设计(Structured Design, SD)方法是一种面向数据流的设计方法,它可以与 SA 方法衔接。结构化设计方法的基本思想是将系统设计成由相对独立、功能单一的模块组成的结构。

1. 信息流的类型

在需求分析阶段,用 SA 方法产生了数据流图。面向数据流的设计能方便地将 DFD 转换成程序结构图。DFD 中从系统的输入数据流到系统的输出数据流的一连串连续变换形成了一条信息流。DFD 的信息流大体上可以分为两种类型,一种是变换流,另一种是事务流。

1) 变换流

信息沿着输入通路进入系统,同时将信息的外部形式转换成内部表示,然后通过变换中心(也称主加工)处理,再沿着输出通路转换成外部形式离开系统。具有这种特性的信息流称为变换流。变换流型的 DFD 可以明显地分成输入、变换(主加工)、输出 3 大部分。

2) 事务流

信息沿着输入通路到达一个事务中心,事务中心根据输入信息(即事务)的类型在若干个动作序列(称为活动流)中选择一个来执行,这种信息流称为事务流。事务流有明显的事务中心,各活动流以事务中心为起点呈辐射状流出。

2. 变换分析

变换分析是从变换流型的 DFD 导出程序结构图。

1) 确定输入流和输出流, 分离出变换中心

我们把 DFD 中系统输入端的数据流称为物理输入, 系统输出端的数据流称为物理输出。物理输入通常要经过编辑、格式转换、合法性检查、预处理等辅助性的加工才能为主加工的真正输入(称它为逻辑输入)。从物理输入端开始, 一步步向系统的中间移动, 可找到离物理输入端最远, 但仍可被看作系统输入的那个数据流, 这个数据流就是逻辑输入。同样由主加工产生的输出(称为逻辑输出)通常也要经过编辑、格式转换、组成物理块、缓冲处理等辅助加工才能变成物理输出。从物理输出端开始, 一步步向系统的中间移动, 可找到离物理输出端最远, 但仍可被看作系统输出的那个数据流, 这个数据流就是逻辑输出。

DFD 中从物理输入到逻辑输入的部分构成系统的输入流, 从逻辑输出到物理输出的部分构成系统的输出流, 位于输入流和输出流之间的部分就是变换中心。

2) 第一级分解

第一级分解主要是设计模块结构的顶层和第一层。一个变换流型的 DFD 可映射成图 4-15 所示的程序结构图。图中顶层模块的功能就是整个系统的功能。输入控制模块用来接收所有的输入数据; 变换控制模块用来实现输入到输出的变换; 输出控制模块用来产生所有的输出数据。



图 4-15 变换分析的第一级分解

3) 第二级分解

第二级分解主要是设计中、下层模块。

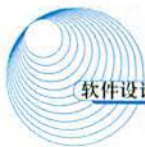
(1) 输入控制模块的分解: 从变换中心的边界开始, 沿着每条输入通路, 把输入通路上的每个加工映射成输入控制模块的一个低层模块。

(2) 输出控制模块的分解: 从变换中心的边界开始, 沿着每条输出通路, 把输出通路上的每个加工映射成输出控制模块的一个低层模块。

(3) 变换控制模块的分解: 变换控制模块通常没有通用的分解方法, 应根据 DFD 中变换部分的实际情况进行设计。

4) 事务分析

事务分析是从事务流型 DFD 导出程序结构图。



(1) 确定事务中心和每条活动流的流特性。图 4-16 给出了事务流型 DFD 的一般形式。其中事务中心(图中的 T)位于数条活动流的起点,这些活动流从该点成辐射状地流出。每条活动流也是一条信息流,它可以是变换流也可以是另一条事务流。一个事务流型的 DFD 由输入流、事务中心和若干条活动流组成。

(2) 将事务流型 DFD 映射成高层的程序结构。事务流型 DFD 的高层结构如图 4-17 所示。顶层模块的功能就是整个系统的功能。接收模块用来接收输入数据,它对应于输入流。发送模块是一个调度模块,控制下层的所有活动模块。每个活动流模块对应于一条活动流,它也是该活动流映射成的程序结构图中的顶层模块。

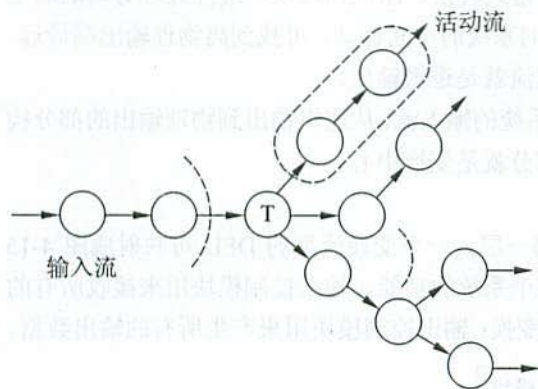


图 4-16 事务流



图 4-17 事务流型 DFD 的高层程序结构

(3) 进一步分解。接收模块的分解类同于变换分析中输入控制模块的分解。每个活动流模块根据其流特性(变换流或事务流)进一步采用变换分析或事务分析进行分解。

5) SD 方法的设计步骤

- (1) 复查并精化数据流图。
- (2) 确定 DFD 的信息流类型(变换流或事务流)。
- (3) 根据流类型分别实施变换分析或事务分析。
- (4) 根据系统设计的原则(参见 4.3.3 节)对程序结构图进行优化。

4.3.5 面向数据结构的设计方法

面向数据结构的设计方法以数据结构作为设计的基础,它根据输入/输出数据结构导出程序的结构,适用于规模不大的数据处理系统。Jackson 方法是一种典型的面向数据结构的设计方法。

1. Jackson 图

尽管程序中实际使用的数据结构有许多种,但这些数据结构中数据元素间的逻辑关系只有顺序、选择、重复3类。表示数据元素间逻辑关系的 Jackson 图如图 4-18 所示。

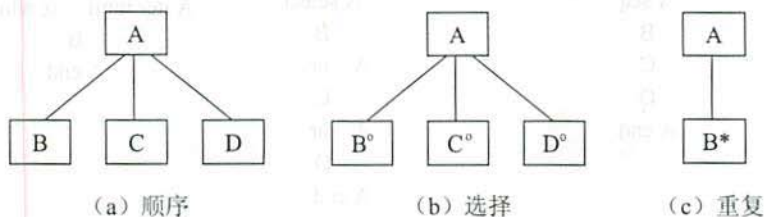


图 4-18 Jackson 图的基本符号

图 4-18 (a) 表示数据结构 A 由 B、C、D 3 个成分顺序组成,其次序是从左到右;图 4-18 (b) 表示数据结构 A 是 B 或 C 或 D 中的一个;图 4-18 (c) 表示数据结构 A 由 B 重复出现多次(可以是 0 次)组成。

Jackson 图同样也可用来表示程序结构,此时图中的方框表示模块,图 4-18 (a) 表示程序的顺序结构,图 4-18 (b) 表示程序的分支结构,图 4-18 (c) 表示程序的重复结构。

2. Jackson 方法的设计步骤

(1) 分析并确定输入和输出数据的逻辑结构,并用 Jackson 图表示。

(2) 找出输入数据结构与输出数据结构间有对应关系的数据单元。所谓有对应关系的数据单元是指有直接因果关系,在程序中可以同时处理的数据单元。对于重复结构的数据单元,必须在重复次数和次序都相同时才有对应关系。

(3) 用下述的 3 条规则从描述数据结构的 Jackson 图导出描述程序结构的 Jackson 图。

① 为每对有对应关系的数据单元按照它们在数据结构图中的层次在程序结构图的相应层次画一个处理框。如果这对有对应关系的数据单元在输入数据结构图中所处的层次与输出数据结构图中所处的层次不同,则取它们中较低的层次作为处理框在程序结构图中的层次。

② 为输入数据结构图中剩余的每个数据单元在程序结构图的相应层次上画一对应的处理框。

③ 为输出数据结构图中剩余的每个数据单元在程序结构图的相应层次上画一处理框。

(4) 列出所有的操作,并把它们分配到程序结构图的适当位置上。

(5) 用伪码表示程序。与描述程序结构的 Jackson 图对应的伪码如图 4-19 所示。

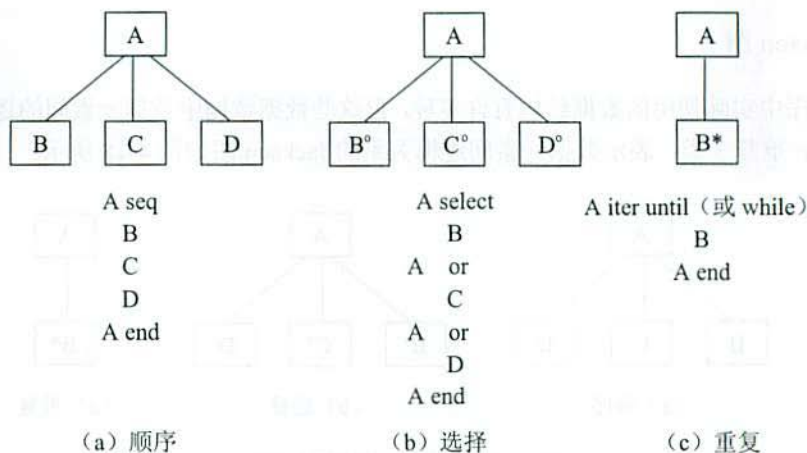
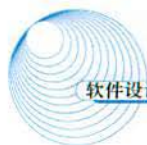


图 4-19 Jackson 图对应的伪码

4.3.6 系统详细设计

1. 代码设计

代码是用来表征客观事物的一组有序的符号,以便于计算机和人工识别与处理。代码的类指代码符号的表示形式一般有数字型、字母型、数字字母混合型等。3 种类型的代码各有所长,应根据使用者的要求、信息量的多少、信息交换的频率、使用者的习惯等方面综合考虑。

1) 代码设计的基本原则

(1) 唯一性。一个对象可能有多个名称,也可按不同的方式对它进行描述。但在一个编码体系中,一个对象只能赋予它唯一的代码。最简单的就是职工编号、学生的学号等。

(2) 合理性。代码结构与响应的分类体系相对应。

(3) 可扩充性。应留有充分的余地,以备将来不断扩充的需要。

(4) 简单性。结构尽可能简单,以减少各种差错。

(5) 适用性。代码尽可能反映对象的特点,以助记忆,便于填写。

(6) 规范性。国家有关编码标准是代码设计的重要依据,已有标准的必须遵循。在一个代码体系中,代码结构、类型、编写格式必须统一。

(7) 系统性。有一定的分组规则,从而在整个系统中具有通用性。例如,在会计领域中,一级会计科目由国家财政部进行标准分类,二级会计科目由各部委或行业协会统一进行标准分类,并且这个分类还必须参照一二级科目的规律进行。

2) 代码设计的步骤

(1) 确定代码对象。

(2) 考察是否已有标准代码。如果国家或者某个部门对某些事物已经规定了标准代码,那么应遵循这些标准代码。

(3) 根据代码的使用范围、使用时间等实际情况选择代码的种类与类型。

(4) 考虑检错功能。

(5) 编写代码表。代码编好后,要编制代码表,做详细说明,通知有关部门组织学习,以便正确使用。

目前常用的编码有顺序码、数字码、字符码和混合码,限于篇幅,这里不做详细论述。

2. 输出设计

从系统开发的角度看,输出决定输入,即输入信息只有根据输出要求才能确定。输出设计包括以下几方面的内容。

(1) 确定输出内容。输出内容的设计首先要确定用户在使用信息方面的要求。根据用户要求,设计输出信息的内容,包括信息形式(表格、图形、文字)、输出项目及数据结构、数据类型、位数及取值范围、数据的生成途径、完整性及一致性的考虑等。

(2) 选择输出设备与介质。常用的输出设备有显示终端、打印机、磁带机、绘图仪、缩微胶卷输出器、多媒体设备。输出介质有纸张、磁带、磁盘、缩微胶卷、光盘、多媒体介质等。这些设备和介质各有特点,应根据用户对输出信息的要求,结合现有设备和资金条件选择。

(3) 确定输出格式。输出格式要满足使用者的要求和习惯,做到格式清晰、美观、易于阅读和理解。

最终输出方式常用的只有两种:一种是报表输出,另一种是图形输出。采用哪种输出形式,应根据系统分析和业务管理的要求而定。一般来说,对于基层和具体事务的管理者,应用报表方式给出详细的记录数据为宜;而对于高层领导或宏观、综合管理部门,则应该使用图形方式用以显示综合数据或发展趋势等信息。

3. 输入设计

输入设计的目的是保证向系统输入正确的数据。在此前提下,做到输入方法简单、迅速、经济、方便。

1) 输入设计应遵循的原则

(1) 最小量原则。这就是保证满足处理要求的前提下使输入量最小。输入量越少,出错机会越小,花费时间越少,数据一致性越好。

(2) 简单性原则。输入的准备、输入过程应尽量容易,以减少错误的发生。

(3) 早检验原则。对输入数据的检验尽量接近源数据发生点,使错误能及时得到改正。

(4) 少转换原则。输入数据尽量使用其处理所需的形式记录,以免数据转换介质时发生



错误。

2) 输入设计的内容

(1) 确定输入数据内容。包括确定输入数据项名称、数据内容、精度、数值范围。

(2) 输入方式设计。主要是根据总体设计和数据库设计的要求来确定数据输入的具体形式。常用的输入方式有键盘输入, 模/数、数/模输入, 网络数据传送, 磁/光盘读入等。通常在设计新系统的输入方式时, 应尽量利用已有的设备和资源, 避免大批量的数据通过键盘输入。

(3) 输入格式设计。实际设计数据输入(特别是大批量的数据统计报表输入)时, 常常遇到统计报表(文件)结构与数据库文件结构不完全一致的情况, 如有可能, 应尽量改变统计报表或数据库关系表二者之一的结构, 使其一致, 以减少输入格式设计的难度。现在还可采用智能输入方式, 由计算机自动将输入送至不同表格。

(4) 校对方式设计。特别是针对数字、金额数等字段, 没有适当的校对措施作保证是很危险的。所以对一些重要的报表, 输入设计一定要考虑适当的校对措施, 以减少出错的可能性。但应指出的是绝对保证不出错的校对方式是没有的。常用的校对方式有人工校对、二次键入校对(同一批数据两次键入)和数据平衡校对。

4. 处理过程设计

总体结构设计将系统分解成许多模块, 并决定了每个模块的外部特征: 功能和界面。计算机处理过程的设计则要确定每个模块的内部特征, 即内部的执行过程, 包括局部的数据组织、控制流、每一步的具体加工要求及各种实施细节。通过这样的设计, 为编写程序制定一个周密的计划。

处理过程设计的关键是用一种合适的表达方法来描述每个模块的执行过程。这种表示方法应该简明、精确, 并由此能直接导出用编程语言表示的程序。常用的描述方式有图形、语言和表格 3 类, 如传统的框图、各种程序语言和判定表等。

1) 程序流程图

流程图(Flow Chart)即程序框图, 是历史最久、流行最广的一种图形表示方法。程序流程图包括 3 种基本成分: 即加工步骤(用方框表示)、逻辑条件(用菱形表示)、控制流(用箭头表示)。

图形表示的优点是直观、形象、容易理解。但从结构化程序设计的角度看, 流程图不是理想的表达工具。缺点之一是表示控制的箭头过于灵活。使用得当, 流程图简单易懂; 使用不当, 流程图可能非常难懂, 而且无法维护。流程图的另一个缺点是只描述执行过程而不能描述有关数据。

2) 盒图(NS图)

盒图是结构化程序设计出现之后, 为支持这种设计方法而产生的一种描述工具。在 NS 图

中,每个处理步骤用一个盒子表示。盒子可以嵌套。盒子只能从上头进入,从下头走出,除此之外别无其他出入口,所以盒图限制了随意的控制转移,保证了程序的良好结构。

3) 形式语言

形式语言是用来描述模块具体算法的非正式的且比较灵活的语言。其外层语法是确定的,而内层语法不确定。外层语法描述控制结构用类似一般编程语言的保留字,所以是确定的。内层语法故意不确定,可以按系统的具体情况和不同层次灵活选用,实际上可用自然语言来描述具体操作。

形式语言同结构性语言的想法是一致的。形式语言的优点是接近自然语言(英语),所以易于理解;其次,它可以作为注释嵌套在程序中成为内部文档,提高程序的自我描述性;第三,因为是语言形式,易于被计算机处理,可用行编辑程序或字处理系统对形式语言进行编辑修改。

4) 决策树

如果一个加工决策或判断的步骤较多,则使用形式语言时语句的嵌套层次太多,不利于基本加工的逻辑功能的清晰描述。决策树是一种图形工具,适合于描述加工中具有多个策略、每个策略和若干条件有关的逻辑功能。

5) 决策表

在基本加工中,如果判断的条件较多,各条件又相互组合、相应的决策方案较多的情形下用决策树来描述,树的结构比较复杂,图中各项注释比较繁琐。决策表也是一种图形工具,呈表形。决策表将比较复杂的决策问题简洁、明确、一目了然地描述出来。

5. 用户界面设计

用户界面是系统与用户之间的接口,也是控制和选择信息输入/输出的主要途径。用户界面设计应坚持友好、简便、实用、易于操作的原则。例如,在设计菜单时应尽量避免菜单嵌套层次过多和每选择一次还需确认一次的设计方式。另外,在设计大批数据输入屏幕界面时,应避免颜色过于鲜艳和多变。

界面设计包括菜单方式、会话管理方式、操作提示方式以及操作权限管理方式等。

1) 菜单方式

菜单是信息系统功能选择操作的最常用方式。按目前软件所提出的菜单设计工具,菜单的形式可以是下拉式、弹出式的,也可以是按钮选择方式的。

2) 会话管理方式

在所有的用户界面中,几乎毫无例外地会遇到人机会话问题。最为常见的有:当用户操作错误时,系统向用户发出提示和警告性信息;当系统执行用户操作命令遇到多种可能时,系统会要求用户进一步说明;系统定量分析的结果通过屏幕向用户发出控制型的信息等。这类会话的处理方式是让系统开发人员根据实际系统操作过程将会话语句写在程序中。



一般会话系统是面向企业领导的,会话系统设计必须满足会话的基本要求,如画面清晰、直观形象,明了简洁,具有容错和纠错能力,提供信息汉字化、图形化、表格化等。

因此,会话设计重点解决会话方式、容错能力和系统的模块结构。

在语音会话方式还没有广泛使用的今天,会话的基本工具是键盘、屏幕和打印机,常用的方式是应答式、菜单式、表格式和图形式。

纠错、容错的目的是保证会话的正确性,提高会话的效率,在系统中可采用如下方法:

(1) 提示法:分简单提示和重复提示法。

(2) 确认回答法:为用户误操作提供改错机会。

(3) 无效处理法:系统拒绝接收错误操作。

(4) 返回处理法:拒绝不熟悉系统的用户使用操作。

(5) 延时处理法:让用户有足够的时间理解系统的提问内容,防止错误回答。

(6) 帮助处理法:给用户提供帮助信息,并给予重新操作的机会。

3) 提示方式与权限管理方式

为了操作使用方便,在设计系统时,常常把操作提示和要点同时显示在屏幕的旁边,以使用户操作方便,这是当前比较流行的用户界面设计方式。另一种操作提示设计方式则是将整个系统操作说明书全送入到系统文件中,并设置系统运行状态指针。当系统运行操作时,指针随着系统运行状态来改变,当用户按“求助”键时,系统则立刻根据当前指针调出相应的操作说明。

与操作方式有关的另一个内容就是对数据操作权限的管理。权限管理一般都是通过入网口令和建网时定义该节点级别这两点相结合来实现的。

6. 安全控制设计

从数据环境和数据处理两方面看,影响系统安全的因素有:

(1) 环境性因素。是指管理机构的组织、硬件和系统软件、系统开发、自然环境等方面的因素。例如,组织方面:职责不分,没有监督机构等;硬件软件方面:硬件失灵,系统软件失灵,逻辑线路错误等;系统开发方面:没有按科学的方法开发系统和设计程序、系统未经测试和调试等;自然环境方面:火灾、水灾、风灾、地震等;安全管理方面:数据处理资源的接触是随意的,无必要的限制等。

(2) 数据处理因素。是指数据处理行为引起的各种情况。例如,输入环节录入错误信息、使用无效代码、击错功能键、丢失数据、重复输入、没有将数据存盘等;处理环节使用了错误程序、错误的数据文件,处理不及时、丢失数据文件和程序等;输出环节错误的发送报表或未及时发送,报告中的错误未加更正等。

要进行系统的安全控制,应针对影响系统安全的两方面因素入手,有的放矢,相应地进行

环境和数据处理两方面的有效控制,以保证系统安全有效地运行。

4.4 系统实施知识

4.4.1 系统实施概述

1. 系统实施的目的和任务

系统实施是新系统开发工作的最后一个阶段。所谓实施指的是将系统设计阶段的结果在计算机上实现,将原来纸面上的、类似于设计图式的新系统方案转换成可执行的应用软件系统。系统实施阶段的主要任务是:

(1) 按总体设计方案购置和安装计算机网络系统。硬件准备包括计算机主机、输入/输出设备、存储设备、辅助设备(稳压电源、空调设备等)、通信设备等。购置、安装和调试这些设备要花费大量的人力、物力,并且持续相当长的时间。

(2) 软件准备。软件准备包括系统软件、数据库管理系统以及一些应用程序。这些软件有些需要购买,有些需要组织人力编写。编写程序是系统实施阶段的重要任务之一。

(3) 培训。主要指用户的培训,包括主管人员和业务人员。这些人员多数来自现行系统,精通业务,但往往缺乏计算机知识。为了保证系统调试和运行顺利进行,应根据他们的基础提前进行培训,使他们适应、逐步熟悉新的操作方法。

(4) 数据准备。数据的收集、整理、录入是一项既繁重、劳动量又大的工作。而没有一定基础数据的准备,系统调试就不可能很好地进行。一般来说,确定数据库模型之后,就应进行数据的整理、录入。这样既分散了工作量,又可以为系统调试提供真实的数据。

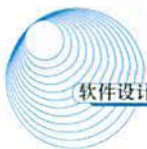
(5) 投入切换和试运行。

在系统实施过程中,还有若干非技术因素的影响。信息系统的最终受益人是企业的最高领导层,信息系统建设涉及到企业机构、权限的重组,只有具备进行变革权力的人才能真正地推进企业信息化。

企业在推行管理信息化时,总经理首先要了解企业一些公众的心理,如企业的各级员工的习惯心理,对信息系统使用持不信任态度的怀疑性排斥心理;此外,信息系统的使用将传统的金字塔管理变为扁平管理,使以前无法暴露的灰色行为,将被一览无遗;素质较低或年龄较大的员工对操作电脑系统具有畏惧心理。如果没有妥善的培训或疏导,这些都将成为系统应用的极大障碍。

2. 系统实施的步骤

系统开发工作沿着信息系统的生命周期逐渐推进,经过详细设计阶段后,便进入系统实施



阶段。

(1) 按总体设计方案购置和安装计算机网络系统。购置和安装硬件是比较简单的事情,只需要按总体设计的要求和可行性报告中财力资源的分析,选择好价格性能比高的设备,通知供货厂家按要求供货并安装即可。

(2) 建立数据库系统。如果前面数据与数据流程分析以及数据库设计工作进行得比较规范,而且开发者又对数据库技术比较熟悉的话,按照数据库设计的要求只需 1~2 个人一天即可建立起一个大型数据库结构。

(3) 程序设计。

(4) 收集有关数据并进行录入工作,然后进行系统测试。

(5) 人员培训、系统转换和试运行。

4.4.2 程序设计

程序设计的主要依据是系统设计阶段的 HIPO 图以及数据库结构和编码设计。

1. 程序设计方法

目前程序设计的方法大多按照结构化方法、原型方法、面向对象的方法进行。

程序设计的目的是为了实现开发者在系统分析和系统设计中提出管理方法和处理构想。所以在程序设计和实现中,建议应尽量借用已有的程序和各种开发工具,尽快尽好地实现系统,而不要在具体的程序设计和调试工作中花费过多的精力和时间。

1) 结构化程序设计方法

若遇到某些开发过程不规范,模块划分不细,或者是因特殊业务处理的需要模块程序量较大时,结构化程序设计方法是一种非常有效的方法。结构化的程序设计方法主要强调 3 点:

(1) 模块内部程序各部分要自顶向下的结构化划分。

(2) 各程序部分应按功能组合。

(3) 各程序部分的联系尽量使用调用子程序(CALL-RETURN)方式,不用或少用 GOTO 方式。

2) 快速原型式的程序开发方法

具体实施方法是首先将 HIPO 图中类似带有普遍性的功能模块集中,如菜单模块、报表模块、查询模块,统计分析和图像模块等,这些模块几乎是每个子系统必不可少的;然后再去寻找有无相应、可用的软件工具,如果没有则可以考虑开发一个能够适合各子系统情况的通用模块;最后用这些工具生成这些程序模块原型。如果 HIPO 图中有一些特定的处理功能和模块,而这些功能和模块又是现有工具不可能生成出来的,则再考虑编制一段程序加进去,利用现有的工具和原型方法可以很快地开发出所要的程序。

3) 面向对象程序设计方法

面向对象程序设计方法一般应与 OOD 所设计的内容相对应。它是一个简单直接的映射过程。即将 OOD 中所定义的范式直接用面向对象程序如 C++, Smalltalk, VisualC 等来取代即可。

2. 程序设计基本模块

一个信息系统的应用软件由很多程序模块组成, 这些程序模块可以归纳成为几种基本类型, 其结构如图 4-20 所示。

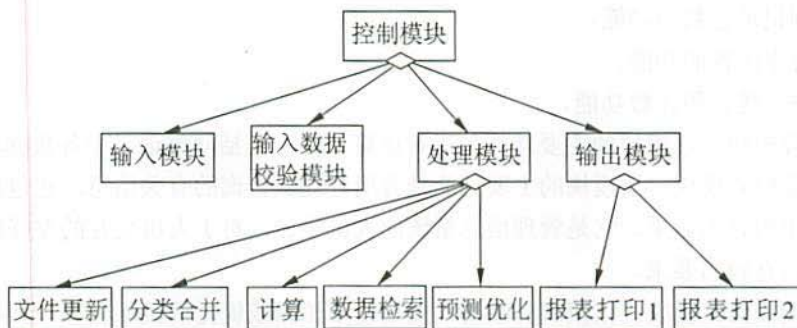


图 4-20 基本程序模块图

1) 控制模块

控制模块包括主控制模块和各级控制模块。控制模块的主要功能是根据用户要求信息, 由用户确定处理顺序, 然后控制转向各处理模块的入口。

2) 输入模块

主要用来输入数据, 输入方式有键盘输入和软盘输入两种。

3) 输入数据校验模块

该模块对已经输入计算机中的数据进行检查, 以保证原始数据的正确性。校验的方法通常有重复输入校验和程序校验两种。

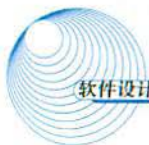
4) 输出模块

输出模块用来将计算机的运行结果通过屏幕、打印机或磁盘、磁带等设备输出给用户。在信息系统中, 一般都有大量的表格、图表需要输出, 因此输出模块的质量直接关系到整个系统的性能。

5) 处理模块

根据信息系统的不同应用要求有不同的处理功能, 通常有以下几种类型:

(1) 文件更新模块: 当系统应用的数据发生变化时, 需要修改数据文件。例如, 增加新的记录、修改数据项或记录、删除某些不需要的记录等。一般来说, 文件更新模块应该具有下述功能:



- ① 对记录中关键字的控制功能,通过关键字查找相应记录。
- ② 控制总记录数的功能,以便控制追加、插入记录的位置。
- ③ 具有按地址或字节访问的控制功能,以便确定修改数据的位置,控制插入或者追加的数据位置。

(2) 分类合并模块:分类合并模块的主要功能是对已经建立的文件,按某关键字进行分类合并。例如,在材料核算系统中耗用材料要按照材料类型合并处理。分类合并程序应该具有下述功能:

- ① 控制记录总数的功能。
 - ② 字符串比较的功能。
 - ③ 排序、统计和计数功能。
- (3) 计算模块:该模块的主要功能是进行计算处理,包括同类记录中各数据项的运算。
- (4) 数据检索模块:该模块的主要功能是为用户提供查询的有关信息,它包括输入查询要求和输出特定的查询结果。它是管理信息系统的人机接口,对于人机交互的友好程序以及查询响应时间等均有较高要求。

(5) 预测或优化模块:该模块的主要功能是使用预测或优化的数学模型,利用信息系统所提供的有关数据,进行计算和分析并输出结果,用来辅助企业或部门的管理人员进行决策。例如库存管理中的 ABC 分类、最佳订货量计算和财务管理中的资金分析等。

3. 程序设计语言的选择

每种程序设计语言都有自己的特点,为一个特定的开发项目选择编程语言时通常可以考虑下列一些因素:应用领域、算法和计算的复杂性、软件运行的环境(包括可使用的编译程序)、用户需求(特别是性能需求)、数据结构的复杂性、开发人员的水平等。

从应用领域来看,COBOL 语言适用于商业领域的应用,FORTRAN 语言适合于科学和工程计算的应用,PROLOG 和 LISP 适合于人工智能的应用,对于一些采用面向对象方法的应用系统通常可选用 C++或 Java。有些程序设计语言可应用于多种应用领域。例如 C 语言,它原先是为辅助开发 UNIX 操作系统而设计的,主要用于开发系统软件,现在已经广泛应用于其他领域。

开发和维护高级语言程序要比开发和维护低级语言程序容易得多,但是高级语言程序经过编译后所产生的目标程序要比完成相同功能的低级语言程序长得多,也就是说前者的功效要比后者的功效低。因此某些对运行时间或存储空间有过高要求的项目,或者不能提供高级语言编译程序的计算机上开发程序,往往要使用低级语言。在某些大型系统中,为了提高系统的运行效率,有时也对系统中在执行时间上起关键作用的模块局部使用低级语言。

当然,所选择的语言必须有可使用的编译程序,如果有多种语言都适合于某项目的开发时,也可以考虑选择开发人员比较熟悉的一种。

4.4.3 系统测试与调试

1. 系统测试的意义、目的及原则

系统测试是为了发现错误而执行程序的过程，成功的测试是发现尚未发现的错误的测试。

测试的目的就是希望能以最少的人力和时间发现潜在的各种错误和缺陷。应根据开发各阶段的需求、设计等文档或程序的内部结构精心设计测试实例，并利用这些实例来运行程序，以便发现错误的过程。信息系统测试应包括软件测试、硬件测试和网络测试。硬件测试、网络测试可以根据具体的性能指标来进行，此处所说的测试更多是指软件测试。

系统测试是保证系统质量和可靠性的关键步骤，是对系统开发过程中的系统分析、系统设计和实施的最后复查。根据测试的概念和目的，在进行信息系统测试时应遵循以下基本原则：

(1) 应尽早并不断地进行测试。测试不是在应用系统开发完之后才进行的。由于原始问题的复杂性、开发各阶段的多样性以及参加人员之间的协调等因素，使得在开发各个阶段都有可能出现的错误。因此，测试应贯穿在开发的各个阶段，尽早纠正错误，消除隐患。

(2) 测试工作应该避免由原开发软件的人或小组承担。因为开发人员往往不愿否认自己的工作，总认为自己开发的软件没有错误；另一方面，开发人员的错误很难由本人测试出来，很容易根据自己编程的思路来制定测试思路，具有局限性。测试工作应由专门人员进行，会更客观，更有效。

(3) 设计测试方案的时候，不仅要确定输入数据，而且要根据系统功能确定预期输出结果。将实际输出结果与预期结果相比较就能发现测试对象是否正确。

(4) 在设计测试实例时，不仅要设计有效合理的输入条件，也要包含不合理、失效的输入条件。测试的时候，人们往往习惯按照合理的、正常的情况进行测试，而忽略了对异常、不合理、意想不到的情况进行测试，而这些可能就是隐患。

(5) 在测试程序时，不仅要检验程序是否做了该做的事，还要检验程序是否做了不该做的事。多余的工作会带来副作用，影响程序的效率，有时会带来潜在的危害或错误。

(6) 严格按照测试计划来进行，避免测试的随意性。测试计划应包括测试内容、进度安排、人员安排、测试环境、测试工具和测试资料等。严格的按照测试计划可以保证进度，使各方面都得以协调进行。

(7) 妥善保存测试计划、测试例子，作为软件文档的组成部分，为维护提供方便。

(8) 测试例子都是精心设计出来的，可以为重新测试或追加测试提供方便。当纠正错误、系统功能扩充后，都需要重新开始测试，而这些工作重复性很高，可以利用以前的测试例子，或在其基础上修改，然后进行测试。



2. 测试过程

测试是开发过程中一个独立且非常重要的阶段,测试过程基本上与开发过程平行进行。

一个规范化的测试过程通常包括以下基本的测试活动。

(1) 制定测试计划。在制定测试计划时,要充分考虑整个项目的开发时间和开发进度以及一些人为因素和客观条件等,使得测试计划是可行的。测试计划的内容主要有:测试的内容、进度安排、测试所需的环境和条件、测试培训安排等。

(2) 编制测试大纲。测试大纲是测试的依据。它明确详尽地规定了在测试中针对系统的每一项功能或特性所必须完成的基本测试项目和测试完成的标准。

(3) 根据测试大纲设计和生成测试用例,产生测试设计说明文档,其内容主要有被测项目、输入数据、测试过程、预期输出结果等。

(4) 实施测试。测试的实施阶段是由一系列的测试周期组成的。在每个测试周期中,测试人员和开发人员将依据预先编制好的测试大纲和准备好的测试用例,对被测软件或设备进行完整的测试。

(5) 生成测试报告。测试完成后,要形成相应的测试报告,主要对测试进行概要说明,列出测试的结论,指出缺陷和错误。另外,给出一些建议,如可采用的修改方法,各项修改预计的工作量及修改的负责人员。

4.4.4 测试策略和测试方法

软件测试方法分为静态测试和动态测试。

1. 静态测试

静态测试指被测试程序不在机器上运行,而是采用人工检测和计算机辅助静态分析的手段对程序进行检测。

(1) 人工检测。人工检测是不依靠计算机而是靠人工审查程序或评审软件。

(2) 计算机辅助静态分析。利用静态分析工具对被测试程序进行特性分析,从程序中提取一些信息,以便检查程序逻辑的各种缺陷和可疑的程序构造。

2. 动态测试

动态测试是指通过运行程序发现错误。对软件产品进行动态测试时可以采用黑盒测试法和白盒测试法。

3. 测试用例的设计

测试用例由测试输入数据和与之对应的预期输出结构组成。在设计测试用例时,应当包括

合理的输入条件和不合理的输入条件。

1) 用黑盒法设计测试用例

黑盒测试也称为功能测试，在完全不考虑软件的内部结构和特性的情况下，测试软件的外部特性。进行黑盒测试主要是为了发现以下几类错误：

- (1) 是否有错误的功能或遗漏的功能。
- (2) 界面是否有误：输入是否正确接收；输出是否正确。
- (3) 是否有数据结构或外部数据库访问错误。
- (4) 性能是否能够接受。
- (5) 是否有初始化或终止性错误。

常用的黑盒测试技术有等价类划分、边值分析、错误猜测和因果图等。

2) 用白盒法设计测试用例

白盒测试也称为结构测试，根据程序的内部结构和逻辑来设计测试例子，对程序的路径和过程进行测试，检查是否满足设计的需要。

白盒测试常用的技术是逻辑覆盖和基本路径测试。逻辑覆盖考察用测试数据运行被测程序时对程序逻辑的覆盖程度。主要的覆盖标准有 6 种：语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖、路径覆盖。

白盒测试的原则是：

- (1) 程序模块中的所有独立路径至少执行一次。
- (2) 在所有的逻辑判断中，取“真”和取“假”的两种情况至少都能执行一次。
- (3) 每个循环都应在边界条件和一般条件下各执行一次。
- (4) 测试程序内部数据结构的有效性等。

4. 软件测试步骤

软件测试实际上分成 4 步进行。

1) 单元测试

单元测试也称为模块测试，在模块编写完成且无编译错误后就可以进行。如果选用机器测试，一般用白盒测试法，多个模块可以同时进行。单元测试主要检查模块的以下 5 个特征。

(1) 模块接口。模块的接口保证了测试模块的数据流可以正确地流入、流出。在测试中应检查以下要点：

- ① 测试模块的输入参数和形式参数在个数、属性、单位上是否一致。
- ② 调用其他模块时所给出的实际参数和被调用模块的形式参数在个数、属性、单位上是否一致。
- ③ 调用标准函数时所用的参数在属性、数目和顺序上是否正确。



- ④ 全局变量在各模块中的定义和用法是否一致。
- ⑤ 输入是否仅改变了形式参数。
- ⑥ 开/关的语句是否正确。
- ⑦ 规定的 I/O 格式是否与输入/输出语句一致。
- ⑧ 在使用文件之前是否已经打开文件或是用文件之后是否已经关闭文件。

(2) 局部数据结构。在单元测试中,局部数据结构出错是比较常见的错误,在测试时应重点考虑以下因素:

- ① 变量的说明是否合适。
- ② 是否使用了尚未赋值或尚未初始化的变量。
- ③ 变量的初始值或默认值是否正确。
- ④ 变量名是否有错(例如:拼写错)。

(3) 重要的执行路径。在单元测试中,对路径的测试是最基本的任务。由于不能进行穷举测试,需要精心设计测试例子来发现是否有计算、比较或控制流等方面的错误。

① 计算方面的错误:算术运算的优先次序不正确或理解错误、精度不够、运算对象的类型彼此不相容、算法错、表达式的符号表示不正确等。

② 比较和控制流的错误:本应相等的量由于精度造成不相等、不同类型进行比较、逻辑运算符不正确或优先次序错误、循环终止不正确(如多循环一次或少循环一次)或死循环、不恰当地修改循环变量、当遇到分支循环时出口错误等。

(4) 出错处理。成功的设计应该能预测到出错的条件并且有对出错处理的路径。虽然计算机可以显示出错信息的内容,但仍需要程序员对出错进行处理,保证其逻辑的正确性,以便于用户维护。

(5) 边界条件。边界条件的测试是单元测试的最后工作,也是非常重要的工作。软件容易在边界出现错误。

由于模块不是独立运行的程序,各模块之间存在调用与被调用的关系。在对每个模块进行测试时,需要开发两种模块:

① 驱动模块。相当于一个主程序,接收测试例子的数据,将这些数据送到测试模块,输出测试结果。

② 桩模块,也称为存根模块。桩模块用来代替测试模块中所调用的子模块,其内部可进行少量的数据处理,目的是为了检验入口、输出调用和返回的信息。

提高模块的内聚度可以简化单元测试。如果每个模块只完成一种功能,对于具体模块来讲,所需的测试方案数据就会显著减少,而且更容易发现和预测模块中的错误。

2) 集成测试

集成测试就是把模块按系统设计说明书的要求组合起来进行测试。即使所有模块都通过了

测试,但在组装之后,仍可能会出现問題。如穿过模块的数据被丢失、一个模块的功能对其他模块造成有害的影响、各个模块组装起来没有达到预期的功能、全局数据结构出现问题等。另外,单个模块的误差可以接受,但模块组合后,可能会出现误差累积,最后到不能接受的程度,所以需要组装测试。

通常组装测试有两种方法:一种是分别测试各个模块,再把这些模块组合起来进行整体测试,即非增量式集成。另一种是把下一个要测试的模块组合到已测试好的模块中,测试完后再将下一个需要测试的模块组合起来,进行测试,逐步把所有模块组合在一起,并完成测试,即增量式集成。非增量式集成可以对模块进行并行测试,能充分利用人力,并加快工程进度。但这种方法容易混乱,出现错误不容易查找和定位。增量式测试的范围一步步扩大,错误容易定位,而且已测试的模块可在新的条件下再测试,测试更彻底。

3) 确认测试

经过组装测试之后,软件就被集成起来,接口方面的问题已经解决,将进入软件测试的最后一个环节——确认测试。确认测试的任务就是进一步检查软件的功能和性能是否与用户要求的一样。系统方案说明书描述了用户对软件的要求,所以是软件有效性验证的标准,也是确认测试的基础。

确认测试,首先要进行有效性测试以及软件配置审查,然后进行验收测试和安装测试,经过管理部门的认可和专家的鉴定后,软件即可以交给用户使用。

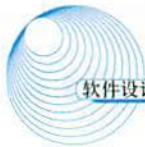
(1) 有效性测试:就是在模拟环境下,通过黑盒测试检验所开发的软件是否与需求规格说明书一致。在设计测试例子时,除了检测软件的功能和性能之外,还需要对软件的容错性、维护性等其他方面进行检测。测试人员可由开发者的内部人员组成,但最好是没有参加该项目的有经验的软件设计人员。在所有测试例子完成之后,若发现测试结果与预期的不符,这时要列出缺陷清单。在这个阶段才发现的严重错误,一般很难在预定的时间内纠正,需要与用户协商,寻找妥善解决问题的办法。

(2) 软件配置审查:主要是检查软件(源程序、目标程序)和文档(包括面向开发和用户的文档)是否齐全以及分类是否有序。确保文档、资料的正确和完善,以便维护阶段使用。

(3) 验收测试:是以用户为主的测试。软件开发人员和质量保证人员也应该参加。在验收测试之前,需要对用户进行培训,以便熟悉该系统。验收测试的测试例子由用户参与设计,主要验证软件的功能、性能、可移植性、兼容性、容错性等,测试时一般采用实际数据。

4) 系统测试

系统测试是将已经确认的软件、计算机硬件、外设和网络等其他因素结合在一起,进行信息系统的各种组装测试和确认测试,其目的是通过与系统的需求相比较,发现所开发的系统与用户需求不符或矛盾的地方。系统测试是根据系统方案说明书来设计测试例子的,常见的系统测试主要有以下内容:



(1) 恢复测试: 恢复测试监测系统的容错能力。检测方法是采用各种方法让系统出现故障, 检验系统是否按照要求能从故障中恢复过来, 并在约定的时间内开始事务处理, 而且不对系统造成任何伤害。如果系统的恢复是自动的(由系统自动完成), 需要验证重新初始化、检查点、数据恢复等是否正确。如果恢复需要人工干预, 就要对恢复的平均时间进行评估并判断它是否在允许的范围内。

(2) 安全性测试: 系统的安全性测试是检测系统的安全机制、保密措施是否完善, 主要是为了检验系统的防范能力。测试的方法是测试人员模拟非法入侵者, 采用各种方法冲破防线。系统安全性设计准则是使非法入侵者所花费的代价比进入系统后所得到的好处要大, 此时非法入侵已无利可图。

(3) 强度测试: 是对系统在异常情况下的承受能力的测试, 是检查系统在极限状态下运行时, 性能下降的幅度是否在允许的范围内。因此, 强度测试要求系统在非正常数量、频率或容量的情况下运行。强度测试主要是为了发现在有效的输入数据中可能引起不稳定或不正确的数据组合。例如, 运行使系统处理超过设计能力的最大允许值的测试例子、使系统传输超过设计最大能力的数据(包括内存的写入和读出)等。

(4) 性能测试: 检查系统是否满足系统设计方案说明书对性能的要求。性能测试覆盖了软件测试的各阶段, 而不是等到系统的各部分全部都组装之后才确定系统的真正性能。通常与强度测试结合起来进行, 并同时软件、硬件进行测试。软件方面主要从响应时间、处理速度、吞吐量、处理精度等方面来检测。

(5) 可靠性测试: 通常使用以下两个指标来衡量系统的可靠性: 平均失效间隔时间 MTBF (Mean Time Between Failures) 是否超过了规定的时限和因故障而停机时间 MTTR (Mean Time To Repairs) 在一年中不应超过多少时间。

(6) 安装测试: 在安装软件系统时, 会有多种选择。安装测试就是为了检测在安装过程中是否有误、是否容易操作等。主要监测系统的每一个部分是否齐全, 硬件的配置是否合理, 安装中需要产生的文件和数据库是否已产生, 其内容是否正确等。

4.4.5 调试

调试的任务就是根据测试时所发现的错误, 找出原因和具体的位置进行改正。调试工作主要由程序开发人员来进行, 谁开发的程序就由谁来进行调试。

目前常用的调试方法有如下几种:

(1) 试探法。调试人员分析错误的症状, 猜测问题的所在位置, 利用在程序中设置输出语句, 分析寄存器、存储器的内容等手段来获得错误的线索, 一步步地试探和分析出错误所在。这种方法效率很低, 适合于结构比较简单的程序。

(2) 回溯法。调试人员从发现错误症状的位置开始, 人工沿着程序的控制流程往回跟踪代

码,直到找出错误根源为止。这种方法适合于小型程序,对于大规模程序,由于其需要回溯的路径太多而变得不可操作。

(3) 对分查找法。这种方法主要用来缩小错误的范围,如果已经知道程序中的变量在若干位置的正确取值,可以在这些位置上给这些变量以正确值,观察程序运行输出结果,如果没有发现问题,则说明从赋予变量一个正确值开始到输出结果之间的程序没有出错,问题可能在除此之外的程序中,否则错误就在所考察的这部分程序中,对含有错误的程序段再使用这种方法,直到把故障范围缩小到比较容易诊断为止。

(4) 归纳法。归纳法就是从测试所暴露的问题出发,收集所有正确或不正确的数据,分析它们之间的关系,提出假象的错误原因,用这些数据来证明或反驳,从而查出错误所在。

(5) 演绎法。根据测试结果,列出所有可能的错误原因。分析已有的数据,排除不可能和彼此矛盾的原因。对余下的原因,选择可能性最大的,利用已有的数据完善该假设,使假设更具体。用假设来解释所有的原始测试结果,如果能解释这一切,则假设得以证实,也就找出错误;否则,要么是假设不完备或不成立,要么有多个错误同时存在,需要重新分析,提出新的假设,直到发现错误为止。

4.4.6 系统文档

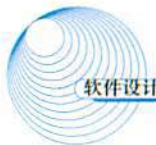
信息系统的文档是系统建设过程的“痕迹”,是系统维护人员的指南,是开发人员与用户交流的工具。规范的文档意味着系统是按照工程化开发的,意味着信息系统的质量有了形式上的保障。文档的欠缺、文档的随意性和文档的不规范,极有可能导致原来的开发人员流动以后,系统不可维护、不可升级,变成了一个没有扩展性、没有生命力的系统。

信息系统的文档,不但包括应用软件开发过程中产生的文档,还包括硬件采购和网络设计中形成的文档;不但包括上述有一定格式要求的规范文档,也包括系统建设过程中的各种来往文件、会议纪要、会计单据等资料形成的不规范文档,后者是建设各方谈判甚至索赔的重要依据;不但包括系统实施记录,也包括程序资料 and 培训教程等。

文档在系统开发人员、项目管理人员、系统维护人员、系统评价人员以及用户之间的多种作用总结如下:

(1) 用户与系统分析人员在系统规划和系统分析阶段通过文档进行沟通。这里的文档主要包括可行性研究报告、总体规划报告、系统开发合同、系统方案说明书等。有了文档,用户就能依次对系统分析员是否正确理解了系统的需求进行评价,如不正确,可以在已有文档的基础上进行修正。

(2) 系统开发人员与项目管理人员通过文档在项目期内进行沟通。这里的文档主要有系统开发计划(包括工作任务分解表、网络图、甘特图、预算分配表等)、系统开发月报以及系统开发总结报告等项目管理文件。有了这些文档,不同阶段之间的开发人员就可以进行工作的顺利衔接,同时还能降低因为人员流动带来的风险,因为接替人员可以根据文档理解前面人员的



设计思路或开发思路。

(3) 系统测试人员与系统开发人员通过文档进行沟通。系统测试人员可以根据系统方案说明书、系统开发合同、系统设计说明书、测试计划等文档对系统开发人员所开发的系统进行测试。系统测试人员再将评估结果撰写成系统测试报告。

(4) 系统开发人员与用户在系统运行期间进行沟通。用户通过系统开发人员撰写的文档运行系统。这里的文档主要是用户手册和操作指南。

(5) 系统开发人员与系统维护人员通过文档进行沟通。这里的文档主要有系统设计说明书和系统开发总结报告。有的开发总结报告写得很详细,分为研制报告、技术报告和技术手册3个文档,其中的技术手册记录了系统开发过程中的各种主要技术细节。这样,即使系统维护人员不是原来的开发人员,也可以在这些文档的基础上系统地进行维护与升级。

(6) 用户与维修人员在运行维护期间进行沟通。用户在使用信息系统过程中,将运行过程中的问题进行记载,形成系统运行报告和维修修改建议。系统维护人员根据维护修改建议以及系统开发人员留下的技术手册等文档,对系统进行维护和升级。

4.4.7 系统转换

在进行新旧系统转换以前,首先要进行新系统的试运行。在系统测试、调试中,我们使用的是系统测试数据,有些实际运行中可能出现的问题,很难通过这些数据被发现。所以,一个系统开发后让它实际运行一段时间是对系统最好的检验和测试方法。

1. 系统试运行阶段的主要工作

- (1) 对系统进行初始化、输入各原始数据记录。
- (2) 记录系统运行的数据和状况。
- (3) 核对新系统输出和旧系统(人工或计算机系统)输出的结果。
- (4) 对实际系统的输入方式进行考察(是否方便、效率如何、安全可靠、误操作保护等)。
- (5) 对系统实际运行、响应速度(包括运算速度、传递速度、查询速度、输出速度等)进行实际测试。

2. 系统转换的方式

新系统试运行成功之后,就可以在新系统和旧系统之间互相转换。新旧系统之间的转换方式有直接转换、并行转换和分段转换。

- (1) 直接转换。直接切换就是在确定新系统运行无误时,立刻启用新系统,终止旧系统运行。这种方式节省人员、设备费用。一般适用于一些处理过程不太复杂,数据不很重要的场合。
- (2) 并行转换。这种转换方式是新旧系统并行工作一段时间,经过一段时间的考验以后,

新系统正式替代旧系统。对于较复杂的大型系统,它提供了一个与旧系统运行结果进行比较的机会,可以对新旧两个系统的时间要求、出错次数和工作效率给以公正的评价。由于与旧系统并行工作,消除了尚未认识新系统之前的紧张和不安。在银行、财务和一些企业的核心系统中,这是一种经常使用的切换方式。它的主要特点是安全、可靠,但费用和工作量都很大,因为在相当长时间内系统要两套班子并行工作。

(3) 分段转换。分段转换又称逐步转换、向导转换、试点过渡法等。这种切换方式实际上是以上两种切换方式的结合。在新系统全部正式运行前,一部分一部分地代替旧系统。那些在转换过程中还没有正式运行的部分,可以在一个模拟环境中继续试运行。这种方式既保证了可靠性,又不至于费用太大。但是这种分段转换要求子系统之间有一定的独立性,对系统的设计和实现都有一定的要求,否则就无法实现这种分段转换的设想。

3. 系统转换实例

在实际工作中,切换方法较为灵活。一个信息系统从使用到成熟再到提高,是一个比较长的过程。只有遵循数据处理的阶段性,信息系统才能健康发展。现以一个连锁企业开始实施新系统为例。

(1) 初始阶段:企业首先为应用系统做基本资料的准备,进行总部、“配送”核心系统的实施。

(2) 推广阶段:总部、“配送”系统稳定后,先从 1~2 家门店试点开始,以门店核心模块为主,完成门店与总部的信息交换、物流过程。然后再逐步推广门店系统,直至完成所有门店的联网工作。

(3) 控制阶段:所有门店联网完成后,进行准确、及时的基本数据采集和调整工作。

(4) 集成阶段:再考虑自动补货、自动配货、财务接口等高级模块应用的工作。

(5) 管理阶段:进入数据的全面启用和介入管理决策。最后,系统步入成熟阶段。

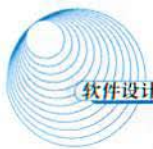
实际上每个企业在不同阶段的发展过程中,对具体问题的解决方法也是不同的,随时都会进行以上阶段的周期重复,随着每次重复时起点的不断升高,整个企业的数据应用水平也就随之逐步提高了。

4.5 系统运行和维护知识

4.5.1 系统维护概述

1. 系统可维护性概念

系统的可维护性可以定性的定义为:维护人员理解、改正、改动和改进这个软件的难易程



度。提高可维护性是开发管理信息系统所有步骤的关键目的,系统是否能被很好地维护,可用系统的可维护性这一指标来衡量。

1) 系统的可维护性的评价指标

(1) 可理解性。指别人能理解系统的结构、界面功能和内部过程的难易程度。模块化、详细设计文档、结构化设计和良好的高级程序设计语言等,都有助于提高可理解性。

(2) 可测试性。诊断和测试的容易程度取决于易理解的程度。好的文档资料有利于诊断和测试,同时,程序的结构、高性能的测试工具以及周密计划的测试工序也是至关重要的。为此,开发人员在系统设计和编程阶段就应尽力把程序设计成易诊断和测试的。此外,在系统维护时,应该充分利用在系统测试阶段保存下来的测试用例。

(3) 可修改性。诊断和测试的容易程度与系统设计所制定的设计原则有直接关系。模块的耦合、内聚、作用范围与控制范围的关系等,都对可修改性有影响。

2) 维护与软件文档

文档是软件可维护性的决定因素。由于长期使用的大型软件系统在使用过程中必然会经受多次修改,所以文档显得非常重要。

软件系统的文档可以分为用户文档和系统文档两类。用户文档主要描述系统功能和使用方法,并不关心这些功能是怎样实现的;系统文档描述系统设计、实现和测试等各方面的内容。

可维护性是所有软件都应具有的基本特点,必须在开发阶段保证软件具有可维护的特点。在软件工程的每一个阶段都应考虑并提高软件的可维护性,在每个阶段结束前的技术审查和管理复查中,应该着重对可维护性进行复审。

在系统分析阶段的复审过程中,应该对将来要改进的部分和可能会修改的部分加以注解并指明,并且指出软件的可移植性问题以及可能影响软件维护的系统界面;在系统设计阶段的复审期间,应该从容易修改、模块化和功能独立的目的出发,评价软件的结构和过程;在系统实施阶段的复审期间,代码复审应该强调编码风格和内部说明文档这两个影响可维护性的因素。在完成了每项维护工作之后,都应该对软件维护本身进行认真的复审。

3) 软件文档的修改

维护应该针对整个软件配置,不应该只修改源程序代码。如果对源程序代码的修改没有反映在设计文档或用户手册中,可能会产生严重的后果。每当对数据、软件结构、模块过程或任何其他有关的软件特点做了改动时,必须立即修改相应的技术文档。不能准确反映软件当前状态的设计文档可能比完全没有文档更坏。在以后的维护工作中很可能因文档不完全符合实际而不能正确理解软件,从而在维护中引入过多的错误。

2. 系统维护的内容及类型

系统维护主要包括硬件设备的维护、应用软件的维护和数据的维护。

1) 硬件维护

硬件的维护应由专职的硬件维护人员来负责,主要有两种类型的维护活动:一种是定期的设备保养性维护,保养周期可以是一周或一个月不等,维护的主要内容是进行例行的设备检查与保养,易耗品的更换与安装等;另一种是突发性的故障维护,即当设备出现突发性故障时,由专职的维修人员或请厂方的技术人员来排除故障,这种维修活动所花时间不能过长,以免影响系统的正常运行。

2) 软件维护

软件维护主要是指根据需求变化或硬件环境的变化对应用程序进行部分或全部的修改。修改时应充分利用源程序,修改后要填写程序修改登记表,并在程序变更通知书上写明新老程序的不同之处。

软件维护的内容一般有以下几个方面:

(1) 正确性维护。是指改正在系统开发阶段已发生而系统测试阶段尚未发现的错误。这方面的维护工作量要占整个维护工作量的 17%~21%。所发现的错误有的不太重要,不影响系统的正常运行,其维护工作可随时进行;而有的错误非常重要,甚至影响整个系统的正常运行,其维护工作必须制定计划,进行修改,并且要进行复查和控制。

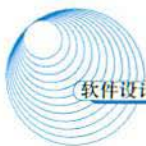
(2) 适应性维护。是指使应用软件适应信息技术变化和管理需求变化而进行的修改。这方面的维护工作量占整个维护工作量的 18%~25%。由于目前计算机硬件价格的不断下降,各类系统软件层出不穷,人们常常为改善系统硬件环境和运行环境而产生系统更新换代的需求;企业的外部市场环境和管理需求的不断变化也使得各级管理人员不断提出新的信息需求。这些因素都将导致适应性维护工作的产生。进行这方面的维护工作也要像系统开发一样,有计划、有步骤地进行。

(3) 完善性维护。这是为扩充功能和改善性能而进行的修改,主要是指对已有的软件系统增加一些在系统分析和设计阶段中没有规定的功能与性能特征。这些功能对完善系统功能是非常必要的。另外还包括对处理效率和编写程序的改进,这方面的维护占整个维护工作的 50%~60%,比重较大,也是关系到系统开发质量的重要方面。这方面的维护除了要有计划、有步骤地完成外,还要注意将相关的文档资料加入到前面相应的文档中去。

(4) 预防性维护。为了改进应用软件的可靠性和可维护性,为了适应未来的软硬件环境的变化,应主动增加预防性的新的功能,以使应用系统适应各类变化而不被淘汰。比如将专用报表功能改成通用报表生成功能,以适应将来报表格式的变化。这方面的维护工作量占整个维护工作量的 4%左右。

3) 数据维护

数据维护工作主要是由数据库管理员来负责,主要负责数据库的安全性和完整性以及进行并发性控制。数据库管理员还要负责维护数据库中的数据,当数据库中的数据类型、长度等发



生变化时,或者需要添加某个数据项、数据库时,要负责修改相关的数据库、数据字典,并通知有关人员。另外数据库管理员还要负责定期出版数据字典文件及一些其他数据管理文件,以保留系统运行和修改的轨迹。当系统出现硬件故障并得到排斥后要负责数据库的恢复工作。

数据维护中还有一项很重要的内容,那就是代码维护。不过代码维护发生的频率相对较小。代码的维护应由代码管理小组进行。变更代码应经过详细讨论,确定之后要用书面形式贯彻。代码维护的困难往往不在于代码本身的变更,而在于新代码的贯彻。为此,除了成立专门的代码管理小组外,各业务部门要指定专人进行代码管理,通过他们贯彻使用新代码。这样做的目的是要明确管理职责,有助于防止和更正错误。

3. 系统维护的管理和步骤

要强调的是,系统的修改往往会“牵一发而动全身”。程序、文件、代码的局部修改都可能影响系统的其他部分。因此,系统的维护工作应有计划有步骤地统筹安排,按照维护任务的工作范围、严重程度等诸多因素确定优先顺序,制定出合理的维护计划,然后通过一定的批准手续实施对系统的修改和维护。

通常对系统的维护应执行以下步骤:

(1) 提出维护或修改要求。操作人员或业务领导用书面形式向系统维护工作的主管人员提出对某项工作的修改要求。这种修改要求一般不能直接向程序员提出。

(2) 领导审查并做出答复,如同意修改则列入维护计划。系统主管人员进行一定的调查后,根据系统的情况和工组人员的情况,考虑这种修改是否必要、是否可行,做出是否修改、何时修改的答复。如果需要修改,则根据优先程度的不同列入系统维护计划。计划的内容应包括维护工作的范围、所需资源、确认的需求、维护费用、维护进度安排以及验收标准等。

(3) 领导分配任务,维护人员执行修改。系统主管人员按照计划向有关的维护人员下达任务,说明修改的内容、要求、期限。维护人员在仔细了解原系统的设计和开发思路的情况下对系统进行修改。

(4) 验收维护成果并登记修改信息。系统主管人员组织技术人员对修改部分进行测试和验收。验收通过后,将修改的部分嵌入系统,取代旧的部分。维护人员登记所做的修改,更新相关的文档,并将新系统作为新的版本通报用户和操作人员,指明新的功能和修改的地方。

在进行系统维护过程中,还要注意维护的副作用。维护的副作用包括两个方面,一是修改程序代码有时会发生灾难性的错误,造成原来运行比较正常的系统变得不能正常运行。为了避免这类错误,要在修改工作完成后进行测试,直至确认和复查无错为止;二是修改数据库中数据的副作用,当一些数据库中的数据发生变化时可能导致某些应用软件不再适应这些已经变化了的数据而产生错误。为了避免这类错误,一是要有严格的数据描述文件,即数据字典系统;二是要严格记录这些修改并进行修改后的测试工作。

总之,系统维护工作是信息系统运行阶段的重要工作内容,必须予以充分的重视。维护工作做得越好,信息系统的作用才能够得以充分发挥,信息系统的寿命也就越长。

4.5.2 系统评价

1. 系统评价概述

信息系统的评价分为广义和狭义两种。广义的信息系统评价是指从系统开发的一开始到结束的每一阶段都需要进行评价。狭义的信息系统评价则是指在系统建成并投入运行之后所进行的全面、综合的评价。

1) 系统评价的分类

按评价的时间与信息系统所处的阶段的关系,又可从总体上把广义的信息系统评价分成立项评价、中期评价和结项评价。

(1) 立项评价。指信息系统方案在系统开发前的预评价,即系统规划阶段中的可行性研究。评价的目的是决定是否立项进行开发,评价的内容是分析当前开发新系统的条件是否具备,明确新系统目标实现的重要性和可能性,主要包括技术上的可行性、经济上的可行性、管理上的可行性和开发环境的可行性等方面。由于事前评价所用的参数大都是不确定的,所以评价的结论具有一定的风险性。

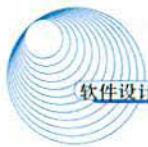
(2) 中期评价。项目中期评价包含两种含义:一是指项目方案在实施过程中,因外部环境出现重大变化,比如市场需求变化、竞争性技术或更完美的替代系统的出现,或者发现原先设计有重大失误等,需要对项目的方案进行重新评估以决定是继续执行还是终止该方案;另一种含义也可称为阶段评估,是指在信息系统开发正常情况下,对系统设计、系统分析、系统实施阶段的阶段性成果进行评估。由于一般都将阶段性成果的提交视为信息系统建设的里程碑,所以,阶段评估又可叫里程碑式评价。

(3) 结项评价。信息系统的建设是一个项目,是项目就需要有终结时间。结项评价是指项目准备结束时对系统的评价,一般是指在信息系统投入正式运行以后,为了了解系统是否达到预期的目的和要求而对系统运行的实际效果进行的综合评价。所以,结项评价又是狭义的信息系统评价。信息系统项目的鉴定是结项评价的一种正规的形式。结项评价的主要内容包括系统性能评价、系统的经济效益评价以及企业管理效率提高、管理水平改善、管理人员劳动强度减轻等间接效果。通过结项评价,用户可以了解系统的质量和效果,检查系统是否符合预期的目的和要求;开发人员可以总结开发工作的经验、教训,这对今后的工作十分有益。

2) 系统评价注意事项

在对信息系统进行评价考核的时候,应该注意以下几个问题。

(1) 信息系统通过基本资料录入、进货、订货、盘点、零售等各个环节采集进来,其中任



意一个环节的数据录入出现问题,都将导致最终报表的不准确,而报表不准确就意味着企业决策者无法根据报表决定企业的运作,更谈不上数据分析和决策支持了。这也是我们目前大部分使用了信息系统的企业普遍存在的问题。究竟是什么原因导致了数据采集的不准确呢?一些企业错误地将数据准确性作为考核信息部的一个指标,其实数据不准确更多源于管理上存在的问题,正因为数据源头非常多,所造成的数据不一致的问题不是信息部都能解决的,最终数据采集的成败将由最高管理层对数据采集各环节的管理力度所决定。而数据采集的成败又将最终决定整个信息系统应用的成败。

(2) 信息系统并不是万能系统。系统应用的过程中有些问题是信息系统擅长解决的,如大量的、重复的、规范性的事务处理;而有些问题是信息系统不擅长解决的,如特殊的、偶然的、不规范的经营管理内容。让信息系统做不擅长的工作,势必在应用的过程中投入的管理成本远远大于它所产生的效益。对这种灵活、多变的情况,不妨采用人工处理或通过制度的限制,尽量避免不规范的行为频繁发生,从而真正实现企业简单复制、快速扩张、规模效益的目的。

2. 系统评价的指标

我们从以下几方面综合考虑,建立起一套指标体系理论框架:

(1) 从信息系统的组成部分出发,信息系统是一个由人机共同组成的系统,所以可以按照运行效果和用户需求(人)、系统质量和技术条件(机)这两条线索构造指标。

(2) 从信息系统的评价对象出发,对于开发方来说,他们所关心的是系统质量和技术水平;对于用户方而言,关心的是用户需求和运行质量;系统外部环境则主要通过社会效益指标来反映。

(3) 从经济学角度出发,分别按系统成本、系统效益和财务指标等3条线索建立指标。

第 5 章 网络基础知识

计算机网络是由多台计算机组成的系统,与传统的单机系统、多机系统相比有很大的区别。计算机网络的结构、功能、组成以及实现技术更复杂,维护起来难度更大。本章将简要介绍计算机网络的体系结构、网络应用、WindowsNT 系统管理、网络构建与网络安全方面的基本内容,涉及到与网络有关的硬件和应用知识。

5.1 网络概述

由于对信息共享和信息传递应用的迫切需求,计算机应用日益渗透到各行各业。计算机已不仅仅是一种计算的工具,而是更多地应用于信息的收集、处理和传输。计算机参与社会信息化的进程,促进了信息产业的发展。而应用需求又推动了计算机技术的不断更新,越来越多的领域急需计算机在一定范围内“联合起来”,进行群体工作。在此环境下计算机网络应运而生。计算机网络是计算机技术与通信技术相结合的产物,它实现了远程通信、远程信息处理和资源共享。经过几十年的发展,计算机网络已由早期的“终端-计算机网”、“计算机-计算机网”成为现代具有统一网络体系结构的计算机网络。

5.1.1 计算机网络的概念

1. 计算机网络的发展

计算机网络的发展过程大致可以划分为 4 个阶段:

1) 具有通信功能的单机系统

该系统又称终端-计算机网络,是早期计算机网络的主要形式。它将一台计算机经通信线路与若干终端直接相连。美国于 20 世纪 50 年代建立的半自动地面防空系统 SAGE 就属于这一类网络。它把远距离的雷达和其他测量控制设备的信息通过通信线路送到一台旋风型计算机上进行处理和控制,首次实现了计算机技术与通信技术的结合。

2) 具有通信功能的多机系统

对终端-计算机网进行改进:即在主计算机的外围增加了一台计算机,专门用于处理终端的通信信息及控制通信线路,并能对用户的作业进行某些预处理操作,这台计算机称为“前端处理机”或“通信控制处理机”。在终端设备较集中的地方设置一台集中器,终端通过低速线路先汇集到集中器上,然后再用高速线路将集中器连到主机上。这就形成多机系统。



3) 以共享资源为目的的计算机网络

具有通信功能的多机系统是计算机-计算机网络,它是由若干台计算机互连的系统,即利用通信线路将多台计算机连接起来,在计算机之间进行通信。该网络有两种结构形式:一种形式是主计算机通过通信线路直接互连的结构,其中主计算机同时承担数据处理和通信工作;另一种形式是通过通信控制处理机间接地把各主计算机连接的结构,其中通信处理机和主计算机分工,前者负责网络上各主计算机间的通信处理和控制,后者是网络资源的拥有者,负责数据处理,它们共同组成资源共享的计算机网络。20世纪70年代,美国国防部高级研究计划局所研制的 ARPANET 是计算机-计算机网络的典型代表。最初该网仅由4台计算机连接而成,到1975年,已连接100多台不同型号的大型计算机。ARPANET成为第一个完善地实现分布式资源共享的网络,为计算机网络的发展奠定了基础。

在这期间,国际标准化组织(ISO)提出了开放系统互连参考模型 OSI/RM (Open System Interconnection Reference Model)。该模型定义了异种机联网所应遵循的框架结构。OSI/RM很快得到了国际上的认可,并为许多厂商所接受。由此使计算机网络的发展进入了新的阶段。

4) 以局域网及互联网为支撑环境的分布式计算机系统

局域网是继远程网之后发展起来的,它继承了远程网的分组交换技术和计算机的 I/O 总线结构技术。局域网的发展也促使计算机网络的模式发生了变革,即由早期的以大型机为中心的集中式模式转变为由微机构成的分布式计算机模式。

计算机网络的定义随网络技术的更新可从不同的角度给以描述。目前人们已公认的有关计算机网络的定义是:利用通信设备和线路将地理位置分散的、功能独立的自主计算机系统或由计算机控制的外部设备连接起来,在网络操作系统的控制下,按照约定的通信协议进行信息交换,实现资源共享的系统。

定义中涉及的“资源”应该包括硬件资源(CPU、大容量的磁盘、光盘以及打印机等)和软件资源(语言编译器、文本编辑器、各种软件工具、应用程序等)。

2. 计算机网络的功能

1) 数据通信

通信或数据传输是计算机网络主要功能之一,用以在计算机系统之间传送各种信息。利用该功能,地理位置分散的生产单位和业务部门可通过计算机网络连接在一起进行集中控制和管理。也可以通过计算机网络传送电子邮件,发布新闻消息和进行电子数据交换,极大地方便用户,提高工作效率。

2) 资源共享

资源共享是计算机网络最有吸引力的功能。通过资源共享可使网络中分散在异地的各种资源互通有无,分工协作,从而大大提高系统资源的利用率。资源共享包括软件资源共享和硬件

资源共享。

3) 负载均衡

在计算机网络中可进行数据的集中处理或分布式处理，一方面可以通过计算机网络将不同地点的主机或外设采集到的数据信息送往一台指定的计算机，在此计算机上对数据进行集中和综合处理，通过网络在各计算机之间传送原始数据和计算结果；另一方面当网络中某台计算机任务过重时，可将任务分派给其他空闲的多台计算机，使多台计算机相互协作，均衡负载，共同完成任务。

4) 高可靠性

指在计算机网络中的各台计算机可以通过网络彼此互为后备机，一旦某台计算机出现故障，故障机的任务就可由其他计算机代为处理，从而提高系统的可靠性。避免了单机无后备使用的情况下，计算机出现故障而导致系统瘫痪的现象，从而大大提高了系统的可靠性。

借助于计算机网络，在各种功能软件的支持下，人类可以进行高速的异地电子信息交换，并获得了多种服务，如新闻浏览和信息检索、传送电子邮件、多媒体电信服务、远程教育、网上营销、网上娱乐、远程医疗诊断等。

计算网络按照数据通信和数据处理的功能，可分为两层：内层通信子网和外层资源子网。如图 5-1 所示。通信子网（图中虚线内）的节点计算机和高速通信线路组成独立的数据系统，承担全网的数据传输、交换、加工和变换等通信处理工作，即将一个计算机的输出信息传送给另一个计算机。资源子网（图中点划线内虚线外）包括计算机、终端、通信子网接口设备、外部设备（如打印机、磁带机、绘图机等）及各种软件资源等，它负责全网的数据处理和向网络用户提供网络资源及网络服务。

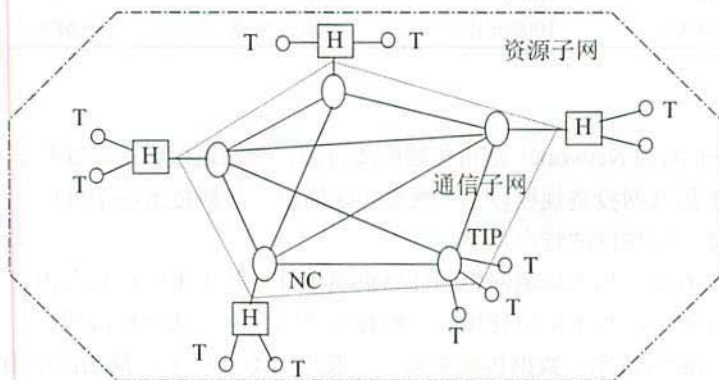
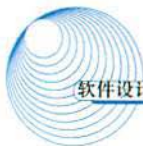


图 5-1 通信子网和资源子网关系图

H—主计算机； T—终端； TIP—集线器； NC—通信处理机



通信子网和资源子网的划分,完全符合国际标准化组织(ISO)所制定的开放式系统互连参考模型 OSI 的思想。其中通信子网对应于 OSI 中的低三层(物理层、数据链路层、网络层),而资源子网对应于 OSI 中的高三层(会话层、表示层、应用层)。这种划分将通信子网的任务从主机中抽取出来,由通信子网中的设备专门解决数据传输和通信控制问题。而资源子网中的计算机可集中精力处理数据,从而提高主机效率和网络的整体性能。

5.1.2 计算机网络的分类

计算机网络的分类方式很多,按照不同的分类原则,可以得到各种不同类型的计算机网络。例如,按通信距离可分为广域网(WAN)、局域网(LAN)和城域网(MAN);按信息交换方式可分为电路交换网、分组交换网和综合交换网;按网络拓扑结构可分为星型网、树型网、环型网和总线网;按通信介质可分为双绞线网、同轴电缆网、光纤网和卫星网等;按传输带宽可分为基带网和宽带网;按使用范围可分为公用网和专用网;按速率可分为高速网、中速网和低速网;按通信传播方式可分为广播式和点到点式。

这里主要介绍根据计算机网络的覆盖范围和通信终端之间相隔的距离不同将其分为局域网、城域网、广域网 3 类的情况,各类网络的特征参数如表 5-1 所示。

表 5-1 各类网络的特征参数

网络分类	缩写	分布距离	计算机分布范围	传输速率范围
局域网	LAN	10m 左右	房间	4Mb/s~1Gb/s
		100m 左右	楼宇	
		1000m 左右	校园	
城域网	MAN	10km	城市	50Kb/s~100Mb/s
广域网	WAN	100km 以上	国家或全球	9.6Kb/s~45Mb/s

1. 局域网

局域网(Local Area Network)是指传输距离有限,传输速度较高,以共享网络资源为目的的网络系统。由于局域网投资规模较小,网络实现简单,故新技术易于推广。局域网技术与广域网相比发展迅速。局域网的特点如下:

- (1) 分布范围有限。加入局域网中的计算机通常处在几千米的距离之内。通常它分布在一个学校、一个企业单位,为本单位使用。一般称为“园区网”或“校园网”。
- (2) 有较高的通信带宽,数据传输率高。一般为 1Mb/s 以上,最高已达 1000Mb/s。
- (3) 数据传输可靠,误码率低。误码率一般为 $10^{-4} \sim 10^{-6}$ 。
- (4) 通常采用同轴电缆或双绞线作为传输介质。跨楼宇时使用光纤。
- (5) 拓扑结构简单简洁,大多采用总线、星型、环型等,系统容易配置和管理。网上的计

计算机一般采用多路控制访问技术或令牌技术访问信道。

(6) 网络的控制一般趋向于分布式,从而减少了对某个节点的依赖性,避免并减小了一个节点故障对整个网络的影响。

(7) 通常网络归单一组织所拥有和使用。不受任何公共网络管理机构的规定约束,容易进行设备的更新和新技术的引用,以不断增强网络功能。

2. 城域网

城域网(Metropolitan Area Network)是规模介于局域网和广域网之间的一种较大范围的高速网络,一般覆盖临近的多个单位和城市,从而为接入网络的企业、机关、公司及社会单位提供文字、声音和图像的集成服务。城域网规范由 IEEE802.6 协议定义。

3. 广域网

广域网(Wide Area Network)又称远程网。它是指覆盖范围广、传输速率相对较低、以数据通信为主要目的数据通信网。广域网最根本的特点是:

(1) 分布范围广。加入广域网中的计算机通常处在数公里到数千公里的地方。因此网络所涉及的范围可为市、地区、省、国家乃至世界。

(2) 数据传输率低。一般为几十 Mb/s 以下。

(3) 数据传输可靠性随着传输介质的不同而不同,若用光纤,误码率一般在 $10^{-6} \sim 10^{-11}$ 之间。

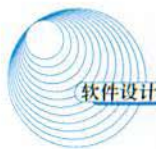
(4) 广域网常常借用传统的公共传输网来实现,因为单独建造一个广域网极其昂贵。

(5) 拓扑结构较为复杂,大多采用“分布式网络”,即所有计算机都与交换节点相连,从而实现网络中任何两台计算机都可以进行通信。

广域网的布局不规则,使得网络的通信控制比较复杂。尤其是使用公共传输网,要求连接到网上的任何用户都必须严格遵守各种标准和规程。设备的更新和新技术的引用难度较大。广域网可将一个集团公司、团体或一个行业的各个部门与子公司连接起来。这种网络一般要求兼容多种网络系统(异构网络)。

5.1.3 网络的拓扑结构

网络拓扑结构是指网络中通信线路和节点的几何排序,用以表示整个网络的结构外貌,反映各节点之间的结构关系。它影响着整个网络的设计、功能、可靠性和通信费用等重要方面,是计算机网络十分重要的要素。常用的网络拓扑结构有总线型、星型、环型、树型、分布式结构等。



1. 总线型结构

总线型拓扑结构如图 5-2 (a) 所示, 其特点为: 总线型拓扑结构中只有一条双向通路, 便于进行广播式传送信息; 总线型拓扑结构属于分布式控制, 无需中央处理器, 故结构比较简单; 节点的增、删和位置的变动较容易, 变动中不影响网络的正常运行, 系统扩充性能好; 节点的接口通常采用无源线路, 系统可靠性高; 设备少, 价格低, 安装使用方便; 由于电气信号通路多, 干扰较大, 因此对信号的质量要求高。负载重时, 线路的利用率较低。网上的信息延迟时间不确定, 故障隔离和检测困难。

2. 星型结构

星型结构中, 使用中央交换单元以放射状连接到网中的各个节点, 如图 5-2 (b) 所示。中央单元采用电路交换方式以建立所希望通信的两节点间专用的路径。通常用双绞线将节点与中央单元进行连接。其特点为: 维护管理容易, 重新配置灵活; 故障隔离和检测容易; 网络延迟时间短; 各节点与中央交换单元直接连通, 各节点之间通信必须经过中央单元转换; 网络共享能力差; 线路利用率低, 中央单元负荷重。

3. 环型结构

环型结构的信息传输线路构成一个封闭的环型, 各节点通过中继器连入网内, 各中继器间首尾相接, 如图 5-2 (c) 所示。信息单向沿环路逐点传送。其特点为: 环型网中信息的流动方向是固定的, 两个节点仅有一条通路, 路径控制简单; 有旁路设备, 节点一旦发生故障, 系统自动旁路, 可靠性高; 信息要串行穿过多个节点, 在网络中节点过多时传输效率低, 系统响应速度慢; 由于环路封闭, 扩充较难。

4. 树型结构

树型结构是总线型结构的扩充形式, 传输介质是不封闭的分支电缆。如图 5-2 (d) 所示。它主要用于多个网络组成的分级结构中。其特点同总线型网。

5. 分布式结构

分布式结构无严格的布点规定和形状, 各节点之间有多条线路相连, 如图 5-2 (e) 所示。其特点为: 分布式网有较高的可靠性, 当一条线路有故障时, 不会影响整个系统工作; 资源共享方便, 网络响应时间短; 由于节点与多个节点连接, 故节点的路由选择和流量控制难度大, 管理软件复杂; 硬件成本高。

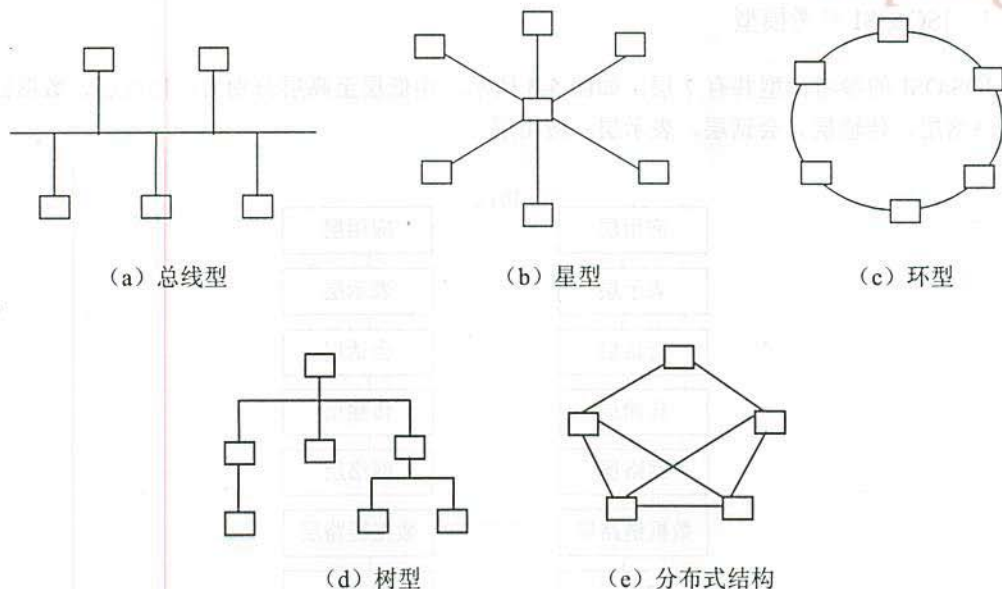


图 5-2 常用的网络拓扑结构

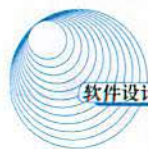
广域网与局域网所使用的网络拓扑结构有所不同,广域网多用分布式或树型结构,而局域网常使用总线型、环型、星型或树型结构。

5.2 ISO/OSI 网络体系结构

计算机网络是相当复杂的系统,相互通信的两个计算机系统必须高度协调才能正常工作。为了设计这样复杂的计算机网络,人们提出了将网络分层的方法。分层可将庞大而复杂的问题转化为若干较小的局部问题进行处理,从而使问题简单化。

国际标准化组织 ISO (International Standard Organization) 1977 年 3 月在 ISO/TC97 第九次会议上成立了新的技术委员会 ISO/TC97/SC16,专门研究网络通信的体系结构问题,并提出了开放系统互连参考模型 OSI/RM,它是一个定义异种计算机连接标准的框架结构。OSI 为连接分布式应用处理的“开放”系统提供了基础。所谓“开放”是指任何两个系统只要遵守参考模型和有关标准,都能够进行互连。OSI 采用了层次化结构的构造技术。

ISO 分委员会的任务是定义一组层次和每一层所完成的功能和服务。层次的划分应当从逻辑上将功能分组,层次应该足够地多,应使每一层小到易于管理的程度,但也不能太多,否则汇集各层的处理开销太大。



1. ISO/OSI 参考模型

ISO/OSI 的参考模型共有 7 层, 如图 5-3 所示。由低层至高层分别为: 物理层、数据链路层、网络层、传输层、会话层、表示层、应用层。

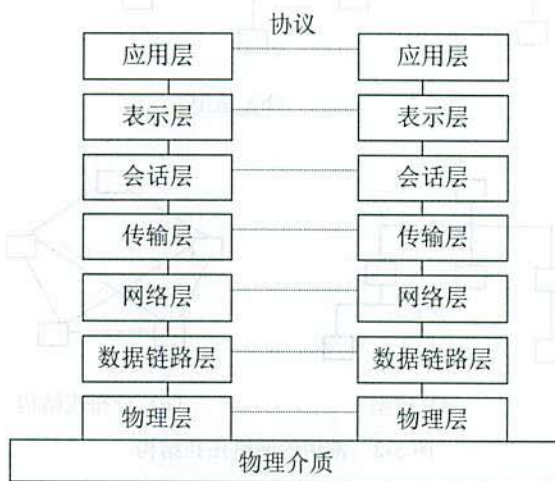


图 5-3 OSI 参考模型

1) OSI 参考模型的特性

- (1) 它是一种将异构系统互连的分层结构。
- (2) 提供了控制互连系统交互规则的标准框架。
- (3) 定义了一种抽象结构, 而并非具体实现的描述。
- (4) 不同系统上的相同层的实体称为同等层实体。
- (5) 同等层实体之间的通信由该层的协议管理。
- (6) 相邻层间的接口定义了原语操作和低层向高层提供的服务。
- (7) 所提供的公共服务是面向连接的或无连接的数据服务。
- (8) 直接的数据传送仅在最低层实现。
- (9) 每层完成所定义的功能, 修改本层的功能并不影响其他层。

2) OSI 参考模型各层的功能

OSI/RM 中的 1~3 层主要负责通信功能, 一般称为通信子网层。上三层(即 5~7 层)属于资源子网的功能范畴, 称为资源子网层。传输层起着衔接上下三层的作用。具体来说:

- (1) 物理层 (Physical Layer): 提供为建立、维护和拆除物理链路所需的机械、电气、功

能和规程的特性;提供有关在传输介质上传输非结构的位流及物理链路故障检测指示。

用户要传递信息就要利用一些物理媒体,如双绞线、同轴电缆等,但具体的物理媒体并不在OSI的7层之内,有人把物理媒体当作第0层,物理层的任务就是为它的上一层提供一个物理连接,以及它们的机械、电气、功能和过程特性。如规定使用电缆和接头的类型、传送信号的电压等。在这一层,数据还没有被组织,仅作为原始的位流或电气电压处理,单位是比特。

(2) 数据链路层(Data Link Layer):数据链路层负责在两个相邻节点间的线路上,无差错的传送以帧为单位的数据,并进行流量控制。每一帧包括一定数量的数据和一些必要的控制信息。和物理层相似,数据链路层要负责建立、维持和释放数据链路的连接。在传送数据时,如果接收点检测到所传数据中有差错,就要通知发方重发这一帧。

(3) 网络层(Network Layer):为传输层实体提供端到端的交换网络数据传送功能。使得传输层摆脱路由选择、交换方式、拥挤控制等网络传输细节;可以为传输层实体建立、维持和拆除一条或多条通信路径;对网络传输中发生的不可恢复的差错予以报告。

在计算机网络中进行通信的两个计算机之间可能会经过很多个数据链路,也可能还要经过很多通信子网。网络层的任务就是选择合适的网络间路由和交换节点,确保数据及时传送。网络层将数据链路层提供的帧组成数据包,包中封装有网络层包头,其中含有逻辑地址信息——源站点和目的站点地址的网络地址。

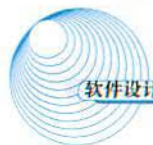
(4) 传输层(Transport Layer):为会话层实体提供透明、可靠的数据传输服务,保证端到端的数据完整性;选择网络层能提供最适宜的服务;提供建立、维护和拆除传输连接功能。传输层根据通信子网的特性最佳地利用网络资源,为两个端系统(也就是源站和目的站)的会话层之间提供建立、维护和取消传输连接的功能,并以可靠和经济的方式传输数据。在这一层,信息的传送单位是报文。

(5) 会话层(Session Layer):为彼此合作的表示层实体提供建立、维护和结束会话连接的功能;完成通信进程的逻辑名字与物理名字间的对应;提供会话管理服务。

这一层也可以称为会晤层或对话层,在会话层及以上的高层次中,数据传送的单位不再另外命名,统称为报文。会话层不参与具体的传输,它提供包括访问验证和会话管理在内的建立和维护应用之间通信的机制。如服务器验证用户登录便是由会话层完成的。

(6) 表示层(Presentation Layer):为应用层进程提供能解释所交换信息含义的一组服务,即将欲交换的数据从适合于某一用户的抽象语法,转换为适合于OSI系统内部使用的传送语法。提供格式化的表示和转换数据服务。数据的压缩、解压缩、加密和解密等工作都由表示层负责。

(7) 应用层(Application Layer):提供OSI用户服务,即确定进程之间通信的性质,以满足用户需要以及提供网络与用户应用软件之间的接口服务。例如事务处理程序、电子邮件和网络管理程序等。



2. 参考模型的信息流向

如图 5-4 所示, 设 A 系统的用户要向 B 系统的用户传送数据。A 系统用户的数据先送入应用层。该层给它附加控制信息 AH (头标) 后, 送入表示层。表示层对数据进行必要的变换并加头标 PH 后送入会话层。会话层亦加头标 SH 送入传输层。传输层将长报文分段后并加头标 TH 送至网络层。网络层将信息变成报文分组, 并加组号 NH 送至数据链路层。数据链路层将信息加上头标和尾标 (DH 及 DT) 变成帧, 经物理层按位发送到对方 (B 系统)。

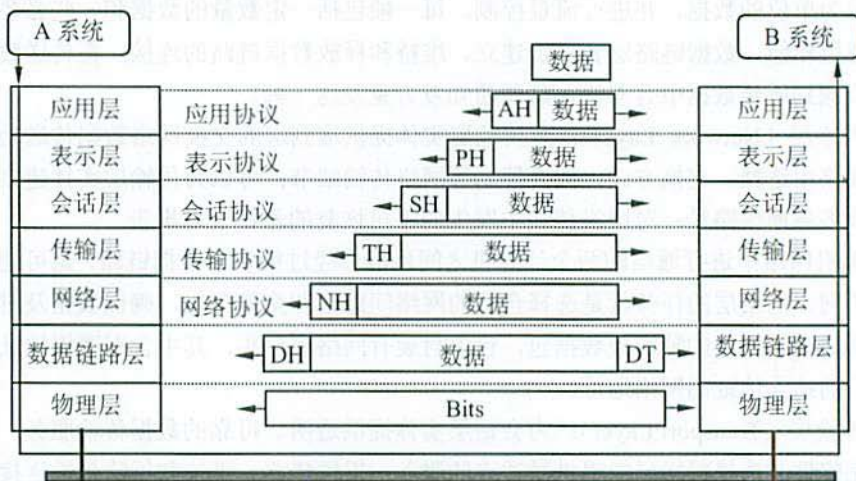


图 5-4 ISO/OSI RM 内信息流动

数据链路层将信息加上头标和尾标 (DH 及 DT) 变成帧, 经物理层按位发送到对方 (B 系统)。B 系统接收到信息后, 按照与 A 系统相反的动作, 层层剥去控制信息, 最后把原数据传送给 B 系统的用户。可见, 两系统中只有物理层是实通信, 而其余各层均为虚通信。因此, 图 5-4 中只有两物理层间有物理连接, 其余各层间均无连线。

5.3 网络互连硬件

构建一个实际的网络, 需要网络的传输介质、网络互连设备作为支持。本节主要介绍构建网络的传输介质和互连设备。

5.3.1 网络的设备

网络互连的目的是使一个网络的用户能访问其他网络的资源, 使不同网络上用户能够互相通信和交换信息, 实现更大范围的资源共享。在网络互连时, 一般不能简单地直接相连而是通

过一个中间设备来实现。按照 ISO/OSI 的分层原则, 这个中间设备要实现不同网络之间的协议转换功能。根据它们工作的协议层不同进行分类, 网络互连设备可以有中继器 (实现物理层协议转换, 在电缆间转发二进制信号)、网桥 (实现物理层和数据链路层协议转换)、路由器 (实现网络层和以下各层协议转换)、网关 (提供从最低层到传输层或以上各层的协议转换)、交换机等。

1. 网络传输介质互连设备

网络线路与用户节点具体衔接时, 需要网络传输介质的互连设备。如 T 型头 (细同轴电缆连接器)、收发器、RJ-45 (屏蔽或非屏蔽双绞线连接器)、RS232 接口 (目前微机与线路接口的常用方式)、DB-15 接口 (连接网络接口卡 的 AUI 接口)、VB35 同步接口 (连接远程的高速同步接口)、网络接口单元、调制解调器 (数字信号与模拟信号转换器) 等。

2. 物理层的互连设备

物理层的互连设备有中继器 (Repeater) 和集线器 (Hub)。

1) 中继器 (Repeater)

它是在物理层上实现局域网网段互连的, 用于扩展局域网网段的长度。由于中继器只在两个局域网网段间实现电气信号的恢复与整形, 因此它仅用于连接相同的局域段。

理论上说, 可以用中继器把网络延长到任意长的传输距离, 但是, 局域网中接入的中继器的数量将受时延和衰耗的影响, 因而必须加以限制。例如在以太网中最多使用 4 个中继器。以太网设计连线时指定两个最远用户之间的距离, 包括用于局域网的连接电缆不得超过 500m。即便使用了中继器, 典型的 Ethernet 局域网应用要求从头到尾整个路径不超过 1500m。中继器的主要优点是安装简便、使用方便、价格便宜。

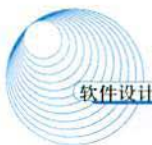
2) 集线器 (HUB)

可以看成是一种特殊的多路中继器, 亦具有信号放大功能。使用双绞线的以太网多用 HUB 扩大网络, 同时也便于网络的维护。以集线器为中心的网络优点是: 当网络系统中某条线路或某节点出现故障时, 不会影响网上其他节点的正常工作。集线器可分为无源 (Passive) 集线器、有源 (Active) 集线器和智能 (Intelligent) 集线器。

无源集线器只负责把多段介质连接在一起, 不对信号作任何处理, 每一种介质段只允许扩展到最大有效距离的一半; 有源集线器类似于无源集线器, 但它具有对传输信号进行再生和放大从而扩展介质长度的功能; 智能集线器除具有有源集线器的功能外, 还可将网络的部分功能集成到集线器中, 如网络管理、选择网络传输线路等。

3. 数据链路层的互连设备

数据链路层的互连设备有网桥、交换机。



1) 网桥 (Bridge)

用于连接两个局域网络段,工作于数据链路层。网桥要分析帧地址字段,以决定是否把收到的帧转发到另一个网络段上。确切地说,网桥工作于 MAC 子层,只要两个网络 MAC 子层以上的协议相同,都可以用网桥互连。

网桥检查帧的源地址和目的地址,如果目的地址和源地址不在同一个网络段上,就把帧转发到另一个网络段上;若两个地址在同一个网络段上,则不转发,所以网桥能起到过滤帧的作用。网桥的帧过滤特性很有用,当一个网络由于负载很重而性能下降时可以用网桥把它分成两个网络段并使得段间的通信量保持最小。例如把分布在两层楼上的网络分成每层一个网络段,段中间用网桥相连,这样的配置可以最大限度地缓解网络通信繁忙的程度,提高通信效率。同时由于网桥的隔离作用,一个网络段上的故障不会影响到另一个网络段,从而提高了网络的可靠性。

2) 交换机 (Switch)

交换机是一个具有简化、低价、高性能和高端口密集特点的交换产品,它是按每一个包中的 MAC 地址相对简单地决策信息转发。而这种转发决策一般不考虑包中隐藏的更深的其他信息。交换机转发数据的延迟很小,操作接近单个局域网性能,远远超过了普通桥接的转发性能。交换技术允许共享型和专用型的局域网段进行带宽调整,以减轻局域网之间信息流通出现的瓶颈问题。

交换机的工作过程为:当交换机从某一节点收到一个以太网帧后,将立即在其内存中的地址表(端口号-MAC 地址)进行查找,以确认该目的 MAC 的网卡连接在哪一个节点上,然后将该帧转发至该节点。如果在地址表中没有找到该 MAC 地址,也就是说,该目的 MAC 地址是首次出现,交换机就将数据包广播到所有节点。拥有该 MAC 地址的网卡在接收到该广播帧后,将立即做出应答,从而使交换机将其节点的“MAC 地址”添加到 MAC 地址表中。

交换机的 3 种交换技术:端口交换(用于将以太模块的端口在背板的多个网段之间进行分配、平衡);帧交换(处理方式:直通交换——提供线速处理能力,交换机只读出网络帧的前 14 个字节,便将网络帧传送到相应的端口上;存储转发——通过对网络帧的读取进行验错和控制;碎片丢弃——它检查数据包的长度是否够 64 个字节,如果小于 64 字节,说明是假包,则丢弃该包,否则发送该包);信元交换(采用长度固定的信元交换)。

4. 网络层互连设备

路由器(Router)是网络层互连设备。它是用于连接多个逻辑上分开的网络。逻辑网络是指一个单独的网络或一个子网,当数据从一个子网传输到另一个子网时,可通过路由器来完成。

路由器具有很强的异种网互连能力,互连网络最低两层协议可以互不相同,通过驱动软件接口到第三层上而得到统一。对于互连网络的第三层协议,如果相同,可使用单协议路由器进

行互连;如果不同,则应使用多协议路由器。多协议路由器同时支持多种不同的网络层协议,并可以设置为允许或禁止某些特定的协议。所谓支持多种协议是指支持多种协议的路由,而不是指不同类协议的相互转换。

通常把网络层地址信息叫做网络逻辑地址,把数据链路层地址信息叫做物理地址。路由器最主要的功能是选择路径。在路由器的存储器中维护着一个路径表,记录各个网络的逻辑地址,用于识别其他网络。在互连网络中,当路由器收到从一个网络向另一个网络发送的信息包时,将丢弃信息包的外层,解读信息包中的数据,获得目的网络的逻辑地址,使用复杂的程序来决定信息经由哪条路径发送最合适,然后重新打包并转发出去。路由器的功能还包括过滤、存储转发、流量管理、介质转换等。一些增强功能的路由器还可有加密、数据压缩、优先、容错管理等功能。由于路由器工作于网络层,它处理的信息量比网桥要多,因而处理速度比网桥慢。

5. 应用层互连设备

网关(Gateway)是应用层的互连设备。在一个计算机网络中,当连接不同类型而协议差别又较大的网络时,则要选用网关设备。网关的功能体现在 OSI 模型的最高层,它将协议进行转换,将数据重新分组,以便在两个不同类型的网络系统之间进行通信。由于协议转换是一件复杂的事,一般来说,网关只进行一对一转换,或是少数几种特定应用协议的转换,网关很难实现通用的协议转换。

5.4.2 网络的传输介质

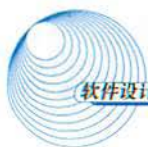
传输介质是信号传输的媒体,常用的介质分为有线介质和无线介质。有线介质有双绞线、同轴电缆和光纤等;无线介质有微波、红外线和微波等。

1. 有线介质

1) 双绞线(Twisted-Pair)

双绞线是现在最普通的传输介质,它分为屏蔽双绞线 STP 和非屏蔽双绞线 UTP。非屏蔽双绞线有电缆外皮作为屏蔽层,适用于网络流量不大的场合中。屏蔽式双绞线具有一个金属甲套,对电磁干扰具有较强的抵抗能力,适用于网络流量较大的高速网络协议应用。双绞线又可分为 3 类、4 类和 5 类、6 类和 7 类双绞线,现在常用的是 5 类 UTP,其频率带宽为 100MHz。6 类、7 类双绞线分别可工作于 200MHz 和 600MHz 的频率带宽上,且采用特殊设计的 RJ45 插头。

双绞线最多应用于 10BASE-T 和 100BASE-T 的以太网中,具体规定有:一段双绞线的最大长度为 100 米,只能连接一台计算机;双绞线的每端需要一个 RJ45 插件;各段双绞线通过集线器互连,利用双绞线最多可连接 64 个站点到中继器。



2) 同轴电缆 (Coaxial)

同轴电缆也像双绞线那样由一对导体组成。同轴电缆又分为基带同轴电缆(阻抗为 50Ω)和宽带同轴电缆(阻抗为 75Ω)。基带同轴电缆用来直接传输数字信号,它又分为粗同轴电缆和细同轴电缆,其中粗同轴电缆适用于较大局域网的网络干线,布线距离较长,可靠性较好,但是网络安装、维护等方面比较困难,造价较高,而细同轴电缆安装较容易,而且造价较低,但因受网络布线结构的限制,其日常维护不甚方便;宽带同轴电缆用于频分多路复用(FDM)的模拟信号发送,还用于不使用频分多路复用的高速数字信号发送和模拟信号发送。闭路电视所使用的CATV电缆就是宽带同轴电缆。

3) 光纤 (Fiber Optic)

光导纤维简称光纤,它重量轻,体积小。用光纤来传输电信号时,在发送端先要将其转换成光信号,而在接收端又要由光检波器还原成电信号,光纤是软而细的、利用内部全反射原理来传导光束的传输介质。按光源采用不同的发光管来分:发光二极管和注入型激光二极管。从而形成了多模光纤(Multimode Fiber)和单模光纤(Single Mode Fiber)。多模光纤使用的材料是发光二极管,价格较便宜,但定向性较差。单模光纤使用的材料是注入型二极管。其定向性好,损耗少,效率高,传播距离长,但价格昂贵。

2. 无线介质

无线传输介质都不需要架设或铺埋电缆或光纤,而是通过大气传输,目前有3种技术:微波、红外线和激光。

1) 微波

微波通信是在对流层视线距离范围内利用无线电波进行传输的一种通信方式,频率范围为 $2\text{GHz}\sim 40\text{GHz}$ 。微波通信是沿直线传播的,由于地球表面是曲面,微波在地面的传播距离有限,直接传播的距离与天线的高度有关,天线越高距离越远,但超过一定距离后就要用中继站来接力,两微波站的通信距离一般为 $30\text{km}\sim 50\text{km}$,长途通信时必须建立多个中继站。中继站的功能是变频和放大,进行功率补偿。微波通信分为模拟微波通信和数字微波通信两种。模拟微波通信主要采用调频制,数字微波通信大都采用相移键控(PSK)。微波通信的传输质量比较稳定,影响质量的主要因素是雨雪天气对微波产生的吸收损耗,不利地形或环境对微波所造成的衰减现象。

2) 红外线和激光

红外通信和激光通信也像微波通信一样,有很强的方向性,都是沿直线传播的。这3种技术都需要在发送方和接收方之间有一条视线(Line-of-Sight)通路,有时统称这三者为视线媒体。所不同的是红外通信和激光通信把要传输的信号分别转换为红外光信号和激光信号,直接在空间传播。由于这3种视线媒体都不需要铺设电缆,对于不论是在地下或用电线杆很难在建筑物

之间架设电缆,特别是要穿越的空间属于公共场所的局域网特别有用。但这3种技术对环境气候较为敏感,例如雨、雾和雷电。相对来说,微波一般对雨和雾的敏感度较低。

3) 卫星通信

卫星通信是以人造卫星为微波中继站,它是微波通信的特殊形式。卫星接收来自地面发送站发出的电磁波信号后,再以广播方式用不同的频率发回地面,由地面工作站接收。卫星通信可以克服地面微波通信距离的限制。一个同步卫星可以覆盖地球的三分之一以上表面,3个这样的卫星就可以覆盖地球上全部通信区域,这样地球上的各个地面站之间都可互相通信了。由于卫星信道频带宽,也可采用频分多路复用技术分为若干个子信道,有些用于由地面站向卫星发送(称为上行信道),有些用于由卫星向地面转发(称为下行信道)。卫星通信的优点是容量大、距离远,缺点是传播延迟时间长。

5.3.3 组建网络

在一个局域网中,其基本组成部件为:服务器、客户机、网络设备、通信介质、网络软件等。

(1) 服务器(Server):局域网的核心,根据它在网络中的作用,还可进一步分为文件服务器、打印服务器和通信服务器。

(2) 客户机(Client):客户机又称为用户工作站,是用户与网络应用接口设备。

(3) 网络设备:主要指一些硬件设备,如网卡、收发器、中继器、集中器、网桥、路由器等。网卡是一种必不可少的网络设备。常用的网卡有Ethernet(以太网)网卡、ARCNET网卡、ESIA总线网卡、Token-Ring网卡等。

(4) 通信介质:数据的传输媒体。不同的通信介质有着不同的传输特性。

(5) 网络软件:网络软件主要包括底层协议软件、网络操作系统(NOS)等。底层协议软件是由一组标准规则及软件构成,以使实体间或网络之间能够互相进行通信;网络操作系统主要对整个网络的资源和运行进行管理并为用户提供应用接口。

【例 5.1】 为了将两个相邻办公室的多台计算机连接成一个局域网,以方便传输文件、共享资源,最简单的连接方式如图 5-5 所示。

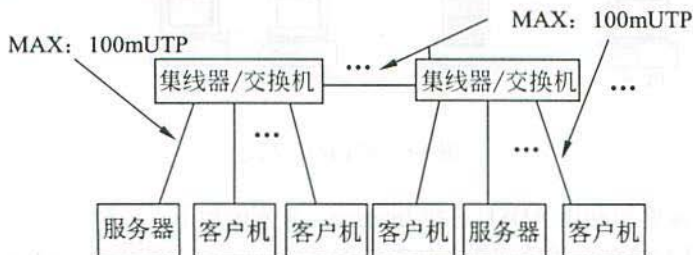


图 5-5 简单网络连接方式



采用集线器 HUB 将两个办公室的多台计算机连接成一个局域网, 如果感觉速度较慢, 可将 HUB 换成交换机 (Switch), 构成树型或总线型和星型结合的拓扑结构; 传输介质采用五类以上的非屏蔽双绞线 UTP, 使用 RJ45 连接 HUB 与计算机; 网卡采用 10/100M 自适应的以太网网卡; 协议使用 TCP/IP 或 NETBIEU 或其他协议。该网络安装和维护简单易行且费用低廉, 计算机和 HUB 之间的最大 UTP 电缆长度为 100m, 两个计算机之间的 (即端-端) 最多允许有 4 个 HUB 和 5 个电缆段, 即最大网络长度为 500m。

【例 5.2】 某校园/大型企业为本单位建设高速信息网络, 网络主干中心为千兆以太网的光纤局域网, 连接各学院、系、图书馆等信息网, 并接入 Internet。实现各级各类网络的互连互通, 为学校的各级单位、教师、学生提供方便、快捷的信息与教学服务, 从而有效地为科研、教学服务, 提高学校的整体水平。网络的构建如图 5-6 所示。

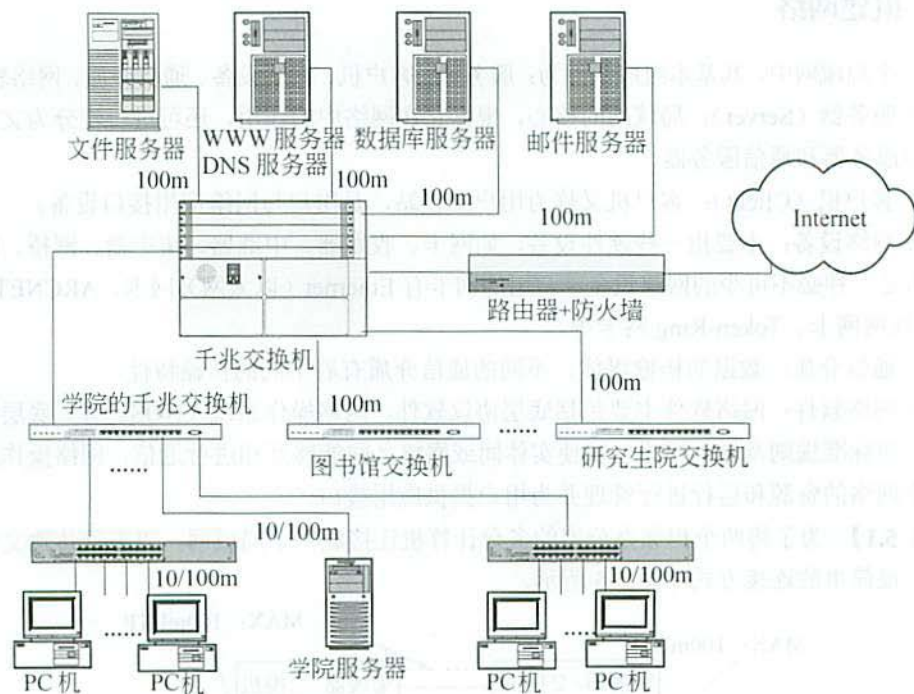


图 5-6 校园网的构建

【例 5.3】 某家庭想利用 ADSL 连接 Internet, 其原因如下:

(1) ADSL 具有很高的传输速率, 其下行 2M~8Mb/s, 上行 64K~640Kb/s 的传输速度, 为普通拨号 Modem 的百倍以上, 也是宽带上网中速度较高的一种。

(2) ADSL 上网和打电话互不干扰, ADSL 数据信号和电话音频信号以频分复用原理调制于各自频段互不干扰, 上网的同时可以使用电话, 避免了拨号上网的烦恼。

(3) ADSL 独享带宽安全可靠, 其他宽带方式虽然在速度方面有快过它的, 但有的是属于共享带宽方式, 如 Cable Modem 下行可达到 20Mb/s, 但它是一种粗糙的总线型广播网络, 成千上万用户争抢 20Mb/s 的带宽, 而非独享, 更为严重的是它属于总线型的网络, 先天的广播特性, 造成了信息传输的不安全性。ADSL 利用中国电信深入千家万户的电话网络, 先天形成星型结构的网络拓扑构造, 骨干网络采用中国电信遍布全国的光纤传输, 独享 2M~8Mb/s 带宽, 信息传递快速可靠安全:

(4) ADSL 费用低廉, 虽然电话线同时传递电话语音和数据, 但数据并不通过电话交换机, 因此不用拨号, 一直在线, 属于专线上网方式。这意味着使用 ADSL 上网不需要缴纳拨号上网的电话费用。另一方面不需要对原有电话线路进行改造, 用户端不需要购买价格昂贵的设备, 只需一个现在相当普及的 ADSL Modem 即可, 相对来说投资较少:

(5) ADSL 能提供真正的视频点播 VOD、网上游戏、交互电视、网上购物等宽带多媒体服务, 远程 LAN 接入、远地办公室、在家工作等高速数据应用, 远程医疗、远程教学、远地可视会议、体育比赛现场实时传送等。

因此决定申请一条 ADSL 线路, 通过拨号来连接 Internet。

ADSL 连接 Internet 的方式有两种: 专线接入和虚拟拨号接入。采用虚拟拨号方式的用户采用类似 Modem 和 ISDN 的拨号程序, 在使用习惯上与原来的方式没什么不同。采用专线接入的用户只要开机即可接入 Internet。

使用 ADSL 上网所需的硬设有一块网卡和 ADSL Modem。当然家用电脑是必须的。连接方式如图 5-7 所示。

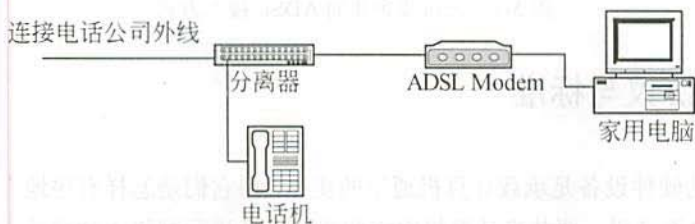
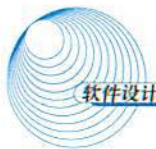


图 5-7 ADSL 单用户接入方式

如果不是家庭, 而是公司或企业, 即局域网通过 ADSL 接入 Internet, 其接入方法有两种:

(1) 直接通过 ADSL 连上网的那台主机设置成代理服务器, 然后本地局域网上的客户机通过该代理服务器访问外部信息资源, 这种方法的好处是需要申请一个账号或一个 IP 地址, 本地客户机可采用保留 IP 地址。



(2) 采用专线方式, 为局域网上的每台计算机向电话局申请 1 个 IP 地址, 这种方法的好处是无需设置一台专用的代理服务网关, 缺陷是目前对一条 ADSL 线路只能提供最多 8 个 IP 地址给局域网。

通过代理服务器将公司或企业接入 Internet 方式如图 5-8 所示。连接时需要注意的问题:

① 接口方式: 有以太网、USB 和 PCI 3 种。USB、PCI 适用于家庭用户, 性价比好, 小巧、方便、实用; 外置以太网口的产品只适用于企业和办公室的局域网, 它可以带多台机器进行上网。有的以太网接口的 ADSL Modem 同时具有桥接和路由功能。

② 分离器: 使上网和打电话互不干扰。

③ 支持的协议: ADSL Modem 上网拨号方式有 3 种: 专线方式(静态 IP)、PPPOA、PPPOE。普通用户多是采用 PPPOE (Point-to-Point Protocol Over Ethernet) 或 PPPOA (Point-to-Point Protocol Over ATM) 虚拟拨号的方式来上网。

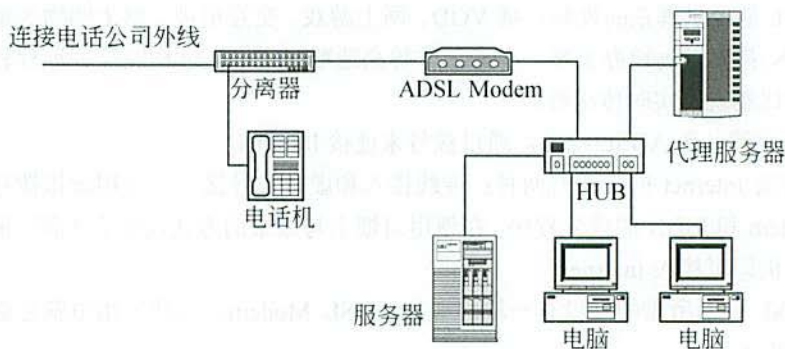


图 5-8 公司或企业的 ADSL 接入方式

5.4 网络的协议与标准

计算机网络的硬件设备是承载计算机通信的实体, 但它们是怎样有序地完成计算机之间的通信任务的呢? 也就是说, 要共享计算机网络的资源以及进行网络中的交换信息, 就需要实现不同系统中的实体的通信。两个实体要想成功地通信, 它们必须具有相同的语言, 在计算机网络中称为协议(或规程)。所谓的协议, 指的是网络中的计算机与计算机进行通信时, 为了能够实现数据的正常的发送与接收, 必须要遵循的一些事先约定好的规则(标准或约定), 在这些规程中明确规定了通信时的数据格式、数据传送时序以及相应的控制信息和应答信号等内容。下面主要介绍网络的标准、局域网的协议与广域网协议。

5.4.1 网络的标准

在网络的标准化方面,有许多标准化机构在工作,如上节介绍的国际标准化组织 ISO、国际电信联盟 ITU、电子工业协会 EIA、电气和电子工程师协会、因特网活动委员会等。

1. 电信标准

1865 年成立国际电信联盟 ITU (International Telecommunication Union), 1947 年 ITU 成为联合国的一个组织,它由 3 部分组成:

(1) ITU-R: 无线通信部门。ITU-R 的主要工作是确保无线电频率和卫星轨道为所有国家平等、有效和经济地利用,召开世界性和地区性大会来制定无线电法规和地区性协议,起草并通过有关技术、业务和系统的建议。

(2) ITU-T: 电信标准部门,下设许多研究组,研究组下设专题,从事网络管理、网络维护、业务运营、网络和终端的端对端传输特性、网络总体方面、多媒体业务和系统等等方面的研究。如 Q42/SG VII 专门研究 OSI 参考模型。

(3) ITU-D: 开发部门。主要宗旨是促进第三世界国家的电信发展。

1953—1993 年,ITU-T 被称为 CCITT (国际电报电话咨询委员会)。CCITT 建议自 1993 年起都打上了 ITU-T 标记。已经公布并使用的最重要的标准有:

(1) V 系列: ITU-T 提出的 V 系列标准主要是针对调制解调器的标准。如: V.90 是 56Kb/s 调制解调器的标准。

(2) X 系列: ITU-T 提出的 X 系列标准是应用于广域网的,该系列标准分为两组。

① X.1~X.39: 标准应用于终端形式、接口、服务设施和设备。最著名的标准是 X.25,它规定了数据包装和传送的协议。

② X.40~X.199: 标准管理网络结构、传输、发信号等。

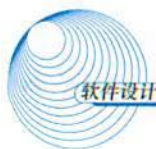
2. 国际标准

1946 年成立的国际标准化组织 ISO 负责制定各种国际标准,ISO 有 89 个成员国家,85 个其他成员。ISO 的任务是促进全球范围内的标准化及其有关活动,以利于国际间产品与服务的交流,以及在知识、科学、技术和经济活动中发展国际间的相互合作。例如: ISO 开发了开放式系统互连网络结构模型,模型定义了用于网络结构的 7 个数据处理层。

其他标准化组织:

(1) ANSI: 美国国家标准研究所,ISO 的美国代表。ANSI 设计了 ASCII 代码组,它是一种广泛使用的数据通信标准代码。

(2) NIST: 美国国家标准和技术研究所,美国商业部的标准化机构。



(3) IEEE: 电气和电子工程师协会 (Institute of Electrical and Electronics Engineers)。IEEE 设置了电子工业标准, IEEE 分成一些标准委员会 (或工作组), 每个工作组负责标准的一个领域, 工作组 802 设置了网络上的设备如何彼此通信的标准, 即 IEEE 802 标准委员会划分成的工作组有: 802.1 工作组 (协调低档与高档 OSI 模型)、802.2 工作组 (涉及逻辑数据链路标准)、802.3 工作组 (有关 CSMA/CD 标准在以太网的应用)、802.4 工作组 (令牌总线标准在 LAN 中的应用)、802.5 工作组 (设置有关令牌环网络的标准)。

(4) EIA: 电子工业协会 (Electronic Industries Association)。最为人熟悉的 EIA 标准之一是 RS-232C 接口, 这一通信接口允许数据在设备之间交换。

值得注意的是, ITU-T 和 ISO 之间有很好的合作和协调。

3. Internet 标准

Internet 的标准特点是自发而非政府干预的, 管理松散, 每个分网络均由各自分别管理, 目前已组成了一个民间性质的协会 ISOC (Internet Society) 进行必要的协调与管理, 有一个网络信息中心 (NIC) 来管理 IP 地址, 保证注册地址的唯一性, 并为用户提供一些文件, 介绍可用的服务。ISOC 设有 Internet 总体管理机构结构 IAB。

1969 年在 ARPANET 时代就开始发布请求评议 RFC (Request For Comments), 至今已超过 3000 个。

5.4.2 局域网协议

IEEE 局域网标准委员会对局域网的定义为: “局域网络中的通信被限制在中等规模的地理范围内, 如一所学校; 能够使用具有中等或较高数据速率的物理信道, 且具有较低的误码率; 局域网络是专用的, 由单一组织机构所使用。” 局域网技术由于具有规模小、组网灵活和结构规整的特点, 所以极易形成标准。事实上, 局域网技术也是在所有计算机网络技术中标准化程序最高的一部分。国际电子电气工程师协议 IEEE 早在 20 世纪 70 年代就制定了 3 个局域网标准: IEEE 802.3 (CSMA/CD, 以太网)、IEEE 802.4 (Token Bus, 令牌总线)、IEEE 802.5 (Token Ring, 令牌环)。由于它已被市场广泛接受, 所以 IEEE 802 系列标准已被 ISO 采纳为国际标准。而且, 随着网络技术的发展, 又出现了 IEEE 802.7 (FDDI)、IEEE 802.3u (快速以太网)、IEEE 802.12 (100VG-AnyLAN)、IEEE 802.3z (千兆以太网) 等新一代网络标准。

一个局域网的基本组成主要有: 网络服务器、网络工作站、网络适配器和传输介质。这些设备在特定网络软件支持下完成特定的网络功能。决定局域网特性的主要技术有 3 个方面: 用以传输数据的传输介质; 用以连接各种设备的拓扑结构; 用以共享资源的介质访问控制方法。它们在很大程度上决定了传输数据的类型、网络的响应时间、吞吐量和利用率, 以及网络应用等各种网络特性。不同的局域网协议最重要的区别是介质访问控制方法, 它对网络特性具有十

分重要的影响。

1. LAN 模型

ISO/OSI 的 7 层参考模型本身不是一个标准,在制定具体网络协议和标准时,要依 OSI/RM 参考模型作为“参照基准”,并说明与该“参照基准”的对应关系。在 IEEE802 局域网 (LAN) 标准中,只定义了物理层和数据链路层两层,并根据 LAN 的特点,把数据链路层分成逻辑链路控制 LLC (Logical Link Control) 子层和介质访问控制 MAC (Medium Access Control) 子层;还加强了数据链路层的功能,把网络层中的寻址、排序、流控和差错控制等功能放在 LLC 子层来实现。图 5-9 为 LAN 协议的层次以及与 OSI/RM 参考模型的对应关系。

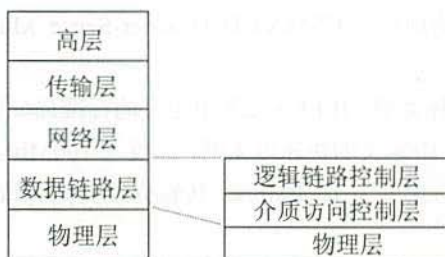


图 5-9 LAN 层次与 OSI/RM 的对应关系

1) 物理层

和 OSI 物理层的功能一样,主要处理在物理链路上发送、传递和接收非结构化的比特流,包括对带宽的频道分配和对基带的信号调制、建立、维持、撤销物理链路,处理机械的、电气的和过程的特性。其特点是可以采用一些特殊的通信媒体,在信息组成的格式上可以有多种。

2) 介质访问控制层 MAC

主要功能是控制对传输介质的访问,MAC 与网络的具体拓扑方式以及传输介质的类型有关,主要是介质的访问控制和对信道资源的分配。MAC 层还实现帧的寻址和识别,完成帧检测序列产生和检验等功能。

3) 逻辑链路控制层 LLC

可提供两种控制类型,即面向连接服务和非连接服务。其中,面向连接服务能够提供可靠的信道。逻辑链路控制层提供的主要功能是数据帧的封装和拆除,为高层提供网络服务的逻辑接口,能够实现差错控制和流量控制。

在计算机网络体系结构中,最具代表性和权威的是 ISO 的 OSI/RM 和 IEEE 的 802 协议。OSI 是设计和实现网络协议标准的最重要的参考模型和依据,而 IEEE802 则制定了一系列具体的局域网标准,并不断地增加新的标准,它们之间的关系如图 5-10 所示。

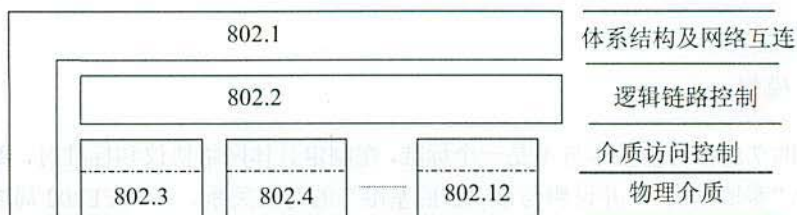
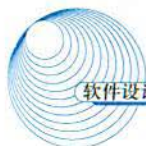


图 5-10 IEEE 802 标准系列间的关系

2. 以太网 (IEEE 802.3 标准)

以太网技术可以说是局域网技术中历史最悠久和最常用的一种。它采用的“存取方法”是带冲突检测的载波监听多路访问协议 CSMA/CD (Carrier-Sense Multiple Access with Collision Detection) 技术。

目前以太网主要包括 3 种类型: IEEE 802.3 中定义的标准局域网, 速度为 10Mb/s, 传输介质为细同轴电缆; IEEE 802.3u 中定义的快速以太网, 速度为 100Mb/s, 传输介质为双绞线; IEEE 802.3z 中定义的千兆以太网, 速度为 1000Mb/s, 传输介质为光纤或双绞线。

1) 介质访问技术

IEEE 802.3 所使用的介质访问协议 CSMA/CD 是让整个网络上的主机都以竞争的方式来抢夺传送数据的权力, 工作过程为: 首先侦听信道, 如果信道空闲则发送。如果信道忙, 则继续侦听, 直到信道空闲时立即发送。开始发送后再进行一段时间的检测, 方法是边发送边接收, 并将收、发信息相比较, 若结果不同, 表明发送的信息遇到碰撞, 于是立即停止发送, 并向总线上发出一串阻塞信号, 通知信道上各站冲突已发生。已发出信息的各站收到阻塞信号后, 等待一段随机时间, 等待时间最短的站将重新获得信道, 可重新发送。

在 CSMA/CD 中, 当检测到冲突并发出阻塞信号后, 为了降低再次冲突概率, 需要等待一个退避时间。退避算法有许多种, 常用的一种通用退避算法称为二进制指数退避算法。

2) IEEE 802.3——10Mb/s 以太网

IEEE 802.3——10Mb/s 以太网 定义过 10BASE5、10BASE2、10BASE-T、10BASE-F 等几种 (需要注明的是, 其中 10BASE-T 与 10BASE-F 的最后一项就是以线缆类型进行命名的, 其中 T 代表双绞线, F 代表光纤)。10BASE5 标准是最早的媒体规范, 它使用阻抗为 50Ω 的同轴粗缆。但由于同轴粗缆的缆线直径大, 所以比较笨重, 不易铺设。10BASE2 标准是为建立一个比 10BASE5 更廉价的局域网, 它使用阻抗为 50Ω 的同轴细缆, 唯一的差别就是它使得每两个节点间的距离限制从 500m 降为 185m。10BASE-T 标准是一个使用非屏蔽双绞线为传输介质的标准, 所要用到的非屏蔽双绞线只需 3 类线标准即可满足要求, 是一个成功的标准。10BASE-F 标准充分利用了新兴媒体光纤的距离长、传输性能好的优点, 大大改进了以太网技术。

3) IEEE 802.3u——100Mb/s 快速以太网

随着计算机技术的不断发展, 10Mb/s 的网络传输速度实在无法满足日益增大的需求。IEEE 802.3u 充分考虑到了向下兼容性: 它采用了非屏蔽双绞线(或屏蔽双绞线、光纤)作为传输媒介, 采用与 IEEE 802.3 一样的介质 CSMA/CD 访问控制层。IEEE 802.3u 常称为快速以太网。根据实现的介质不同, 快速以太网可以分为 100BaseTX、100BaseFX 和 100BaseT4 共 3 种。

100BaseTX 用 2 对 5 类非屏蔽双绞线(UTP), 或者 1 类、2 类屏蔽双绞线(STP)作为传输媒介, 来实现传输速度 100Mb/s 的网络, 最多支持两个中继器。100BaseFX 是 2 束多模光纤上的标准, 在没有中继设备的网络中最大距离为 400m。100BaseT4 利用 10Mb/s 的网络中使用的 3 类线有 2 对是空着没有利用的特点, 使用 4 对 3 类非屏蔽双绞线上提供传输速度为 100Mb/s 网络。

4) IEEE 802.3z——1000Mb/s 千兆以太网

IEEE 802.3z 是对介质访问控制(MAC)层规范进行了重新定义, 以维持适当的网络传输距离, 介质访问控制方法仍采用 CSMA/CD 协议, 并且重新制定了物理层标准, 使之能提供 1000Mb/s 的原始带宽。因此它仍是一种共享介质的局域网, 发送到网上的信号是广播式的, 接收站根据地址接收信号。网络接口硬件能监听线路上是否已存在信号, 以避免冲突, 或在没有冲突时重发数据。

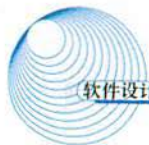
在物理层, 千兆以太网支持 3 种传输介质。光纤系统: 支持多模光纤和单模光纤系统, 多模光纤的工作距离为 500m, 单模光纤的工作距离为 2000m; 宽带同轴电缆系统: 其传输距离为 25m; 5 类 UTP 电缆: 其传输距离为 100m, 链路操作模式为半双工。千兆以太网采用以交换机为中心的星型拓扑结构, 主要用于交换机与交换机之间或者交换机与企业超级服务器之间的高速网络连接。通过将网络核心部件挂接到千兆以太网交换机上。

3. 令牌环网(IEEE 802.5)

令牌环是环型网中最普遍采用的介质访问控制, 它适用于环型网络结构的分布式介质访问控制, 其流行性仅次于以太网。令牌环网的传输介质虽然没有明确定义, 但主要基于屏蔽双绞线、非屏蔽双绞线两种, 拓扑结构可以有多种: 环型(最典型)、星型(采用得最多)、总线型(一种变形), 编码方法为差分曼彻斯特编码。

IEEE 802.5 的介质访问使用的是令牌环控制技术, 工作过程为: 首先, 令牌环网在网络中传递一个很小的帧, 称为“令牌”, 只有拥有令牌环的工作站才有权力发送信息; 令牌在网络上依次顺序传递; 当工作站要发送数据时, 等待捕获一个空令牌, 然后将要发送的信息附加到后边, 发往下一站, 如此直到目标站。然后将令牌释放; 如果工作站要发送数据时, 经过的令牌不是空的, 则等待令牌释放。

当信息帧绕环通过各站时, 各站都要将帧的目的地址与本站地址相比较, 如果地址符合,



说明是发送给本站的,则将帧复制到本站的接收缓冲器中,同时将帧送回到环上,使帧继续沿环传送;如果地址不符合,则简单地将信息帧重新送到环上即可。

4. FDDI (光纤分布式数据接口)

FDDI (Fiber Distributed Data Interface) 是类似令牌环网的协议,它用光纤作为传输介质,数据传输速度可达到 100Mb/s,环路长度可扩展到 200km,连接的站点数可以达到 1000 个。FDDI 采用一种新的编码技术,称为 4B/5B 编码。即每次对 4 位数据进行编码。每 4 位数据编码成 5 位符号,用光信号的存在或不存在来代表 5 位符号中的每一位是 1 还是 0。

光纤中传送的是光信号,有光脉冲表示 1,无光脉冲表示 0。这种简单编码的缺点是没有同步功能。在同轴电缆或双绞线作为传输介质的局域网中,通常采用曼彻斯特编码方式。它利用中间的跳变作为同步信号。这样对每一位数据单元产生两次瞬变,使带宽的利用率降低。5 位编码的 32 种组合中,实际只使用了 24 种,其中的 16 种用来做数据,其余 8 种用来做控制符号(如帧的起始和结束符号等)。4B/5B 编码中,其中 5 位码中“1”码至少为 2 位,按 NRZI 编码原理,信号中就至少有两次跳变,因此接收端可得到足够的同步信息。

FDDI 采用双环体系结构,两环上的信息反方向流动。双环中的一环称为主环,另一环称为次环。在正常情况下,主环传输数据,次环处于空闲状态。双环设计的目的是提供高可靠性和稳定性。FDDI 定义的传输介质有单模光纤和多模光纤两种。

5.4.3 广域网协议

广域网通常是指覆盖范围大,传输速率低,以数据通信为主要目的的数据通信网。随着信息技术的迅速发展,很多国家的数据通信业务的增长率已大大提高,特别是国际互联网的普及促进了数据通信网技术的发展。

在地域分布很远、很分散以致于无法用直接连接来接入局域网的场合,广域网(WAN)通过专用的或交换式的连接把计算机连接起来。这种广域连接可以通过公众网建立的,也可以是通过服务于某个专门部门的专用网建立起来。相对来说,广域网显得比较错综复杂,目前主要用于广域传输的协议比较多:PPP(点对点协议)、DDN、ISDN(综合业务数字网)、FR(帧中继)、ATM(异步传输模式)等。

1. 点对点协议 PPP

PPP 点对点协议主要用于“拨号上网”这种广域连接模式。它的优点在于简单、具备用户验证能力、可以解决 IP 分配等。它主要通过拨号或专线方式建立点对点连接发送数据,使其成为各种主机、网桥和路由器之间简单连接的一种共通的解决方案。

家庭拨号上网就是通过 PPP 在用户端和运营商的接入服务器之间建立通信链路。目前,

宽带接入正在成为取代拨号上网的趋势,在宽带接入技术日新月异的今天,PPP也衍生出新的应用。典型的应用是在ADSL(非对称数据用户线,Asymmetrical Digital Subscriber Line)接入方式当中,PPP与其他的协议共同派生出了符合宽带接入要求的新的协议,如PPPoE(PPP over Ethernet)、PPPoA(PPP over ATM)。

利用以太网(Ethernet)资源,在以太网上运行PPP来进行用户认证接入的方式称为PPPoE。PPPoE既保护了用户方的以太网资源,又完成了ADSL的接入要求,是目前ADSL接入方式中应用最广泛的技术标准。同样,在ATM网络上运行PPP协议来管理用户认证的方式称为PPPoA。它与PPPoE的原理相同,作用相同;不同的是它是在ATM网络上,而PPPoE是在以太网网络上运行,所以要分别适应ATM标准和以太网标准。

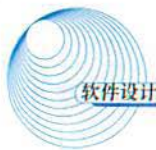
2. 数字用户线 xDSL

xDSL是各种数字用户线的统称,根据各种宽带通信业务需要,目前还有DSL技术和产品:ADSL(Asymmetric DSL)即不对称数字用户线、SDSL(Single pair DSL)单对线数字用户环路、IDSL(ISDN DSL)即ISDN用的数字用户线、RADSL(Rate adaptive DSL)速率自适应非对称型数字用户线、VDSL(Very high Bit rate DSL)甚高速数字用户线等。

ADSL是研制最早,发展较快的一种。它是在一对铜双绞线上,为用户提供上、下行非对称的传输速率(即带宽)。ADSL接入服务能做到较高的性能价格比,ADSL接入技术较其他接入技术具有其独特的技术优势:它的速率可达到上行1Mb/s,下行8Mb/s,速度非常快;另外,使用ADSL上网不需要占用电话线路,在电话和上网互不干扰的同时,大大节省了普通上网方式的话费支出;独享带宽安全可靠;安装快捷方便;价格实惠。它把线路按频段分成语音、上行和下行3个信道,故语音和数据可共用1对线。ADSL特别适合于像VOD业务和Internet和多媒体业务的应用。ADSL一般采用CAP和DMT两种线路编码调制技术。传输距离与线径、速率有关,一般在3km以上。因此ADSL是一种很有发展前途的数字接入技术。ADSL技术作为一种宽带接入方式,可以为用户提供中国电信宽带网的所有应用业务。采用各种拨号方式上网的用户将逐步过渡到ADSL宽带接入方式。ADSL在宽带接入中已经扮演着越来越重要的角色。

对于个人用户,在现有电话线上安装ADSL,只需在用户端安装一台ADSL Modem和一只分离器,用户线路不用任何改动,极其方便。数据线路为:PC→ADSL Modem→分离器→入户接线盒→电话线→DSL接入复用器→ATM/IP网络;语音线路为:话机→分离器→入户接线盒→电话线→DSL接入复用器→交换机。

对于企业用户,在现有电话线上安装ADSL和分离器,连接HUB或Switch。数据线路为:PC→以太网(HUB或Switch)→ADSL路由器→分离器→入户接线盒→电话线→DSL接入复用器→ATM/IP网络;语音线路为:话机→分离器→入户接线盒→电话线→DSL接入复用器→



交换机。

3. 数字专线 DDN

数字数据网 (Digital Data Network) 是采用数字传输信道传输数据信号的通信网, 可提供点对点、点对多点透明传输的数据专线出租电路, 为用户传输数据、图像、声音等信息。数字数据网是以光纤为中继干线网络, 组成 DDN 的基本单位是节点, 节点间通过光纤连接, 构成网状的拓扑结构。

DDN 专线就是市内或长途的数据电路, 电信部门将它们出租给用户做资料传输使用后, 它们就变成用户的专线, 直接进入电信的 DDN 网络, 因为这种电路是采用固定连接的方式, 不需经过交换机房, 所以称之为固定 DDN 专线。DDN 专线不仅需要铺设专用线路 (在 DDN 的客户端需要一个称为 DDN Modem 的 CSU/DSU 设备, 以及一个路由器) 从用户端进入主干网络, 而且需要付电信月租费、网络使用费和电路租用费等。其优势是网络传输速率高、时延小、质量好、网络透明度高、可支持任何规程、安全可靠。

4. 综合业务数字网 ISDN

综合业务数字网是建立在数字电话网基础上的网络。它能提供端到端的数字连接, 是将声音、数据、图像、传真等不同的业务综合在一个网络内进行传送和处理。客户只需一条普通的电话线, 通过网络接口 (NT) 及相应的终端设备, 拨通号码后就能达到既能传数据又能传图像还能打电话的目的。中国电信称其为“一线通”。

窄带综合业务数字网向用户提供的有基本速率 (2B+D, 144Kb/s) 和一次群速率 (30B+D, 2Mb/s) 两种接口。基本速率接口 (BRI), 是一条标准的 ISDN 用户电路, 它包含两个 B 信道和一个 D 信道, B 信道一般用来传输话音、数据和图像, D 信道用来传输信令或分组信息, 各个 B 信道均能够以 64Kb/s 的速率传输数据, D 信道能够以 16Kb/s 速率传输, 支持吞吐量达 144Kb/s 的分组交换数据。一次群速率接口 (PRI), 具有 30 个分立的或组合的 64Kb/s 的 B 信道和一个 64Kb/s 的 D 信道提供高达 2.048Mb/s 的传速率。

ISDN 可以提供电话、传真、可视图文及数据通信等多种业务。其优点是实现高可靠性及高质量的通信, 使用方便, 费用低廉。

5. 帧中继 FR

帧中继 (Frame Relay) 是在用户网络接口之间提供用户信息流的双向传送, 并保持顺序不变的一种承载业务。用户信息以帧为单位进行传输, 并对用户信息流进行统计复用。帧中继是综合业务数字网标准化过程中产生的一种重要技术, 它是在数字光纤传输线路逐渐代替原有的模拟线路, 用户终端智能化的情况下, 由 X25 分组交换技术发展起来的一种传输技术。

帧中继是一种基于可变帧长的数据传输网络,在传输过程中,网络内部可以采用“帧交换”,即以帧为单位进行传送;也可采用“信元交换”(信元 53 B)为单位进行传送。

帧中继提供一种简单的面向连接的虚电路分组服务,包括交换虚电路连接和永久虚电路连接。帧中继的优点有:降低网络互连费用、简化网络功能,提高网络性能、采用国际标准,各厂商产品相互兼容。

6. 异步传输模式 ATM

异步传输模式 ATM (Asynchronous Transfer Mode) 是 B-ISDN 的关键核心技术,它是一种面向分组的快速分组交换模式,使用了异步时分复用技术,将信息流分割成固定长度的信元。使用统一的信息单位能比较容易地实现各种信息流混合在一起的多媒体通信,并能根据业务类型、速率的需求动态地分配有效容量。ATM 能够根据需要改变传送速率,对高速信息传递频次高,而对低速信息传递频次低,按照统计复用的原理进行传输和交换,故 ATM 完全可以用单一的交换方式,灵活有效地支持频带分布范围极广的各种业务。

在 ATM 网络中,数据以定长的信元为单位进行传输,信元由信元头和信元体构成,每个信元 53 B,其中信元头 5 B,信元体 48 B。

ATM 的参考模型由 4 层构成,它们分别是:用户层、ATM 适配层、ATM 层和物理层。图 5-11 给出了简化后的 ATM 的参考模型。

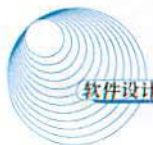


图 5-11 ATM 的参考模型

1) 用户层

由用户面、控制面和管理面组成。其各自的功能如下:

- (1) 用户面: 指用户要求的应用、协议及服务。如通常的数据协议、语音和视频应用等。
- (2) 控制面: 包含连接建立、维护和拆除等有关功能。
- (3) 管理面: 负责层次和平面的协调,通过层次管理支持用户平面和管理平面。



2) ATM 适配层

负责将用户层的信息转换成 ATM 网络可用的格式。等用户层把较长的数据分组交给 ATM 适配层后, ATM 适配层按规定的长度将数据分成若干信元体,再传给 ATM 层。

3) ATM 层

基本功能是负责生成信元,它不管信元体的内容,只为信元体生成信元头并附给信元体,以形成标准的信元格式。跨越 ATM 层到物理层的信息单元只能是 53 B 的信元。

4) 物理层

负责对信元进行编码,并将其交给物理介质。它汇集了各种被认可的物理线路协议,如 SDH (STM-1, STM-4, STM-16), 同步光纤网等。

ATM 连接有两种类型,即永久虚电路(PVC)和交换虚电路(SVC)。

(1) 永久虚电路 PVC: 其连接由管理员建立和拆除。

(2) 交换虚电路 SVC: 通过终端系统和 ATM 网络以及 ATM 网络内部的信令协议的操作来建立和拆除。

虽然在这里将 ATM 技术划在广域网部分来介绍,但 ATM 却是一种可以将局域网功能、广域网功能、语音、视频和数据集成进一个统一的协议设计。正是它的高度统一性和良好的可扩展性,给计算机网络技术掀开了新的一页。它具有的特点: 速度快、支持 622Mb/s 的传输速率、可扩展性好、较高的传输质量 QoS。ATM 提供了端到端解决方案的潜力,这意味着它的应用可以从桌面到局域网,一直延伸到广域网。

ATM 技术适用的范围为: 多媒体和视频应用、适合构架骨干网、无缝地集成广域网和局域网的能力。ATM 主要缺点是成本较高,不适合于小网络。

7. X.25 协议

X.25 在本地 DTE 和远程 DTE 之间提供一个全双工、同步的透明信道,并定义了 3 个相互独立的控制层: 物理层、链路层和分组层,它们分别对应于 ISO/OSI 的物理层、链路层和网络层,如图 5-12 所示。



图 5-12 X.25 层次模型

X.25 是在公用数据网上,以分组方式进行操作的 DTE(数据终端设备)和 DCE(数据通信设备)之间的接口。X.25 只是对公用分组交换网络的接口规范说明,并不涉及网络内部实现,它是面向连接的,支持交换式虚电路和永久虚电路。

物理层接口指 DTE 和网络之间的线路连接,X.25 指出可采用 V.24, V.35, G.703, X.21 等接口;链路层逻辑接口采用链路层协议,负责 DTE 和 DCE 之间的初始化、校验,并控制物理链路上的数据传输,用户数据以信息帧在 DTE 和 DCE 之间传送;分组层逻辑接口描述了呼叫的建立、保持和拆除的过程,以及数据和控制信息在分组中的格式。而默认的数据分组长度是 128 B。

5.4.4 Internet 协议

TCP/IP 作为 Internet 的核心协议,通过近 20 多年的发展已日渐成熟,并被广泛应用于局域网和广域网中,目前已成为事实上的国际标准。

1. TCP/IP 的特性

作为一个最早的、也是迄今为止发展最为成熟的互联网络协议系统,TCP/IP 包含许多重要的基本特性,这些特性主要表现在 5 个方面:逻辑编址、路由选择、域名解析、错误检测和流量控制以及对应用程序的支持等。

(1) 逻辑编址:每一块网卡在出厂时就由厂家分配了一个独一无二的永久性的物理地址。在 Internet 中,为每台连入因特网的计算机分配一个逻辑地址,这个逻辑地址被称为 IP 地址。一个 IP 地址可以包括:一个网络 ID 号,用来标识网络;一个子网络 ID 号,用来标识网络上的一个子网;另外,还有一个主机 ID 号,用来标识子网络上的一台计算机。这样,通过这个分配给某台计算机的 IP 地址,就可以很快地找到相应的计算机。

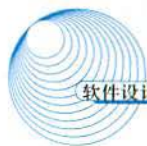
(2) 路由选择:在 TCP/IP 中包含了专门用于定义路由器如何选择网络路径的协议,即 IP 数据包的路由选择。

(3) 域名解析:虽然 TCP/IP 采用的是 32 位的 IP 地址,但考虑用户的记忆方便,专门设计了一种方便的字母式地址结构,称为域名或 DNS(域名服务)名字。将域名映射为 IP 地址的操作,称为域名解析。域名具有较稳定的特点,而 IP 地址则较易发生变化。

(4) 错误检测与流量控制:TCP/IP 具有分组交换确保数据信息在网络上可靠传递的特性,这些特性包括检测数据信息的传输错误(保证到达目的地的数据信息没有发生变化),确认已传递的数据信息已被成功地接收,监测网络系统中的信息流量,防止出现网络拥塞。

2. TCP/IP 分层模型

协议是对数据在计算机或设备之间传输时的表示方法进行定义和描述的标准。协议规定了



进行传输、检测错误以及传送确认信息等内容。TCP/IP 是个协议簇,它包含了多种协议。ISO/OSI 模型、TCP/IP 的分层模型及协议的对比如图 5-13 所示。

ISO/OSI 模型			TCP/IP 协议			TCP/IP 模型	
应用层	文件传输	远程登录	电子邮件	网络文件	网络管理	应用层	
表示层	协议 FTP	协议 Telnet	协议 SMTP	服务 协议 NFS	协议 SNMP		
会话层							
传输层	TCP			UDP		传输层	
网络层	IP	ICMP	ARP	RARP		网际层	
数据链路层	Ethernet IEEE 802.3	FDDI	Token-Ring/ IEEE 802.5	ARCnet	PPP/SLIP	网络接口层	
物理层						硬件层	

图 5-13 TCP/IP 模型与 OSI 模型的对比

从图 5-13 可知, TCP/IP 分层模型由 4 个层次构成, 即应用层、传输层、网际层和网络接口层, 其各层的功能简述如下。

(1) 应用层: 应用层处在分层模型的最高层, 用户调用应用程序来访问 TCP/IP 互联网络, 以享受网络上提供的各种服务。应用程序负责发送和接收数据。每个应用程序可以选择所需要的传输服务类型, 并把数据按照传输层的要求组织好, 再向下层传送, 包括独立的报文序列和连续字节流两种类型。

(2) 传输层: 传输层的基本任务是提供应用程序之间的通信服务。这种通信又叫端到端的通信。传输层既要系统地管理数据信息的流动, 还要提供可靠的传输服务, 以确保数据准确而有序的到达目的地。为了这个目的, 传输层协议软件需要进行协商, 让接收方回送确认信息及让发送方重发丢失的分组。在传输层与网际层之间传递的对象是传输层分组。

(3) 网际层: 网际层又称 IP 层, 主要处理机器之间的通信问题。它接收传输层请求, 传送某个具有目的地址信息的分组。该层主要完成:

① 把分组封装到 IP 数据报 (IP Datagram) 中, 填入数据报的首部 (也称为报头), 使用路由算法选择: 把数据报直接送到目标机或把数据报发送给路由器, 然后, 再把数据报交给下面的网络接口层中对应的网络接口模块。

② 处理接收到的数据报, 检验其正确性。使用路由算法来决定是在本地进行处理, 还是

继续向前发送。如果数据报的目标机处于本机所在的网络，该层软件就把数据报的报头剥去，再选择适当的传输层协议软件来处理这个分组。

③ 适时发出 ICMP 的差错和控制报文，并处理收到的 ICMP 报文。

(4) 网络接口层：网络接口层又称数据链路层，处于 TCP/IP 协议层之下，负责接收 IP 数据报，并把数据报通过选定的网络发送出去。该层包含设备驱动程序，也可能是一个复杂的使用自己的数据链路协议的子系统。

3. 网络接口层协议

TCP/IP 协议不包含具体的物理层和数据链路层，只定义了网络接口层作为物理层与网络层的接口规范。这个物理层可以是广域网，如 X.25 公用数据网，可以是局域网，如 Ethernet, Token-Ring, FDDI 等。任何物理网络只要按照这个接口规范开发网络接口驱动程序，都能够与 TCP/IP 协议集成起来。网络接口层处在 TCP/IP 协议的最底层，主要负责管理为物理网络准备数据所需的全部服务程序和功能。

4. 网际层协议——IP 协议

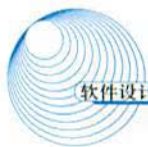
网际层是整个 TCP/IP 协议簇的重点。在网际层定义的协议除了 IP 外，还有 ICMP, ARP, RARP 等几个重要的协议。

IP 所提供的服务通常被认为是无连接的 (Connectionless) 和不可靠的 (Unreliable)。事实上，在网络性能良好的情况下，IP 传送的数据能够完好无损地到达目的地。所谓无连接的传输，是指没有确定目标系统在已做好接收数据准备之前就发送数据。与此相对应的就是面向连接的 (Connection Oriented) 传输 (如 TCP)，在该类传输中，源系统与目的系统的应用层数据传送之前需要进行三次握手。至于不可靠的服务是指目的系统不对成功接收的分组进行确认，IP 只是尽可能地使数据传输成功。但是只要需要，上层协议必须实现用于保证分组成功提供的附加服务。

由于 IP 只提供无连接、不可靠的服务，所以把差错检测和流量控制之类的服务授权给了其他的各层协议，这正是 TCP/IP 能够高效率工作的一个重要保证。这样，可以根据传送数据的属性来确定所需的传送服务以及客户应该使用的协议。例如，传送大型文件的 FTP 会话就需要面向连接的、可靠的服务 (因为如果稍有损坏，就可能导致整个文件无法使用)。

IP 协议的主要功能包括：将上层数据 (如 TCP, UDP 数据) 或同层的其他数据 (如 ICMP 数据) 封装到 IP 数据报中；将 IP 数据报传送到最终目的地；为了使数据能够在链路层上进行传输，对数据进行分段；确定数据报到达其他网络中的目的地的路径。

IP 协议软件的工作流程：当发送数据时，源计算机上的 IP 协议软件必须确定目的地是在同一个网络上，还是在另一个网络上。IP 通过执行这两项计算并对结果进行比较，才能确定数



据到达的目的地。如果两项计算的结果相同,则数据的目的地确定为本地,否则,目的地应为远程的其他网络。如果目的地在本地,那么 IP 协议软件就启动直达通信;如果目的地是远程计算机,那么 IP 必须通过网关(或路由器)进行通信,在大多数情况下,这个网关应当是默认网关。当源 IP 完成了数据报的准备工作时,它就将数据报传递给网络访问层,网络访问层再将数据报传送传输介质,最终完成数据帧发往目的计算机的过程。

当数据抵达目的计算机时,网络访问层首先接收该数据。网络访问层要检查数据帧有无错误,并将数据帧送往正确的物理地址。假如数据帧到达目的地时正确无误,网络访问层便从数据帧的其余部分中提取数据有效负载(Payload),然后将它一直传送到帧层次类型域指定的协议。在这种情况下,可以说数据有效负载已经传递给了 IP。

有关 IP 地址的具体情况参见 Internet 地址。

5. ARP 和 RARP 协议

地址解析协议 ARP(Address Resolution Protocol)及反地址解析协议 RARP 是驻留在网际层中的另一个重要协议。ARP 的作用是将 IP 地址转换为物理地址,RARP 的作用是将物理地址转换为 IP 地址。网络中的任何设备,主机、路由器、交换机等均有唯一的物理地址,该地址通过网卡给出,每个网卡出厂后都有不同的编号,这意味着用户所购买的网卡有着唯一的物理地址。另一方面,为了屏蔽底层协议及物理地址上的差异,IP 协议又使用了 IP 地址,因此,在数据传输过程中,必须对 IP 地址与物理地址进行相互转换。

用 ARP 进行 IP 地址到物理地址转换的过程为:当计算机需要与任何其他的计算机进行通信时,首先需要查询 ARP 高速缓存,如果 ARP 高速缓存中这个 IP 地址存在,便使用与它对应的物理地址,直接将数据报发送给所需的物理网卡;如果 ARP 高速缓存中没有该 IP 地址,那么 ARP 便在局域网上以广播方式发送一个 ARP 请求包;如果局域网上 IP 地址与某台计算机中的 IP 地址相一致,那么该计算机便生成一个 ARP 应答信息,信息中包含对应的物理地址;ARP 协议软件将 IP 地址与物理地址的组合添加到它的高速缓存中,这时即可开始数据通信。

RARP 负责物理地址到 IP 地址的转换。这主要用于无盘工作站上,网络上的无盘工作站在网卡上有自己的物理地址,但无 IP 地址,因此必须有一个转换过程。为了完成这个转换过程,网络中有一个 RARP 服务器,网络管理员事先必须把网卡上的 IP 地址和相应的物理地址存储到 IP RARP 服务器的数据库中。

6. 网际层协议——ICMP 协议

Internet 控制信息协议 ICMP(Internet Control Message Protocol)是网际层的另一个比较重要的协议。由于 IP 协议是一种尽力传送的通信协议,即传送的数据报可能丢失、重复、延迟或乱序传递,所以 IP 协议需要一种避免差错并在发生差错时报告的机制。ICMP 就是一个专门用

于发送差错报文的协议。ICMP 定义了 5 种差错报文（源抑制、超时、目的不可达、重定向、要求分段）和 4 种信息报文（回应请求、回应应答、地址屏蔽码请求、地址屏蔽码应答）。IP 在需要发送一个差错报文时要使用 ICMP，而 ICMP 却也是利用 IP 来传送报文的。ICMP 是让 IP 更加稳固、有效的一种协议，它使得 IP 传送机制变得更加可靠。而且利用 ICMP 还可以用于测试互联网，以得到一些有用的网络维护和排错的信息。例如著名的 ping 工具就是利用 ICMP 报文和进行目标是否可达测试。

7. 传输层协议——TCP 协议

TCP (Transmission Control Protocol) 传输控制协议是整个 TCP/IP 协议系列中最重要的协议之一。它在 IP 协议提供的不可靠数据服务的基础上，为应用程序提供了一个可靠的、面向连接的、全双工的数据传输服务。

TCP 协议是如何实现可靠性的呢？最主要和最重要的是 TCP 采用了一个叫重发 (Retransmission) 的技术。具体来说，在 TCP 传输过程中，发送方启动一个定时器，然后将数据包发出，当接收方收到了这个信息就给发送方一个确认 (Acknowledgement) 信息。而如果发送方在定时器到点之前没收到这个确认信息，就重新发送这个数据包。

利用 TCP 协议在源主机想与目的主机之间建立和关闭连接操作时，均需要通过三次握手来确认建立和关闭是否成功。三次握手方式如图 5-14 所示，它通过“序号/确认号”使得系统正常工作，从而使它们的序号达成同步。TCP 建立连接的三次握手过程为：

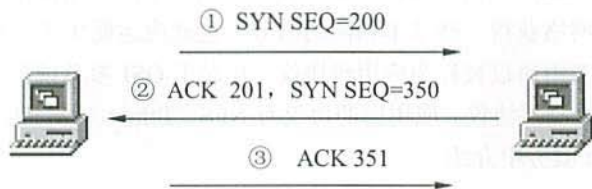


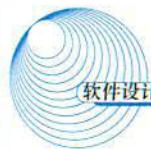
图 5-14 TCP 建立连接的“三次握手”过程

第一步：源主机发送一个 SYN（同步）标志位为 1 的 TCP 数据包，表示想与目标主机进行通信，并发送一个同步序列号（如：SEQ=200）进行同步。

第二步：目标主机愿意进行通信，则响应一个确认（ACK 位设置为 1）。并以下一个序列号为参考进行确认（如：201）。

第三步：源主机以确认来响应目标主机的 TCP 包。这个确认中包括它想要接收的下一个序列号（该帧可以含有发送的数据）。至此连接建立完成。

同样，关闭连接也进行三次握手。



8. 传输层协议——UDP 协议

用户数据报协议 UDP (User Datagram Protocol) 是一种不可靠的、无连接的协议,可以保证应用程序进程间的通信。与同样处在传输层的面向连接的 TCP 相比较,UDP 是一种无连接的协议,它的错误检测功能要弱得多。可以这样说,TCP 有助于提供可靠性;而 UDP 则有助于提高传输的高速率性。例如,必须支持交互式会话的应用程序(如 FTP 等)往往使用 TCP 协议;而自己进行错误检测或不需要错误检测的应用程序(如 DNS, SNMP 等)则往往使用 UDP。

UDP 协议软件的主要作用就是将 UDP 消息展示给应用层,它并不负责重新发送丢失的或出错的数据消息,不对接收到的无序 IP 数据报重新排序,不消除重复的 IP 数据报,不对已收到数据报进行确认,也不负责建立或终止连接。而这些问题是由使用 UDP 进行通信的应用程序负责处理的问题。

TCP 协议虽然提供了一个可靠的数据传输服务,但是它是牺牲通信量来实现的。也就是说,为完成同样一个任务,TCP 会需要更多的时间和通信量。这在网络不可靠的时候,通过耗费一些时间换来达到网络的可靠,但当网络十分可靠的情况下,它又通过浪费带宽来保证可靠性,这时 UDP 则以十分小的通信量浪费占据优势。

9. 应用层协议

计算机网络的广泛应用,人们也已经有了许多相同的基本的应用需求。为了让不同平台的计算机能够通过计算机网络获得一些基本相同的服务,也就应运而生了一系列应用级的标准,实现这些应用的标准的专用协议被称为应用级协议,相对于 OSI 参考模型来说,它们处于较高的层次结构,所以也称为高层协议。应用层的协议有 NFS、Telnet、SMTP、DNS、SNMP、FTP 等,详细情况在 Internet 服务中介绍。

5.5 Internet 及应用

Internet 是世界上规模最大,覆盖面最广且最具影响力的计算机互连网络,它是将分布在世界各地的计算机采用开放系统协议连接在一起,用来进行数据传输、信息交换和资源共享。在现阶段,Internet 作为未来信息高速公路的雏形,无论在科学研究、教育、金融,还是在商业、军事等部门,其影响都越来越大。

5.5.1 Internet 概述

Internet 从用户的角度来看,整个互联网在逻辑上是统一的、独立的,在物理上则由不同

的网络互连而成。从技术角度看, Internet 本身不是某一种具体的物理网络技术, 它是能够互相传递信息的众多网络的一个统称, 或者说它是一个网间网, 只要人们进入了这个互联网, 就是在使用 Internet。正是由于 Internet 的这种特性, 使得广大 Internet 用户不必关心网络的连接, 而只关心网络提供的丰富资源。

连入 Internet 的计算机网络种类繁多, 形式各异, 且分布在世界各地, 因此, 需要通过路由器(IP 网关)并借助各种通信线路或公共通信网络把它们连接起来。由于实现了与公用电话网的互连, 个人用户入网十分方便, 只要有电话和 Modem 即可, 这也是 Internet 迅速普及的原因之一。Internet 由美国的 ARPANET 网络发展而来, 因此, 它沿用了 ARPANET 使用的 TCP/IP 协议, 由于该协议非常有效且使用方便, 许多操作系统都支持它, 无论是服务器还是个人计算机都可安装使用。

对于全球性最大的互联网络, 总的说来是无确定的负责人, 它是由各自独立管理的网络互连构成的, 而这些网络都各自拥有自己的管理体系和政策法规。因此, 没有集中的负责掌管整个 Internet 的机构。尽管如此, 某些政府部门在制定 Internet 有关政策时实际上起着主导作用。Internet 目前的最高国际组织是 Internet 学会(Internet Society)。该学会是一个志愿者组织, 也是一个非盈利性的专业化组织, 其主要目标是促进 Internet 的改革与发展。该学会下分 Internet 体系结构研究会(IAB)和其他几个研究会, IAB 下又有工作组(IETF)、许可证管理局(ICRS)、技术研究组(IRTF)和编号管理局(IANA)等。IAB 的主要任务是为支持 Internet 的科研与开发提供服务。

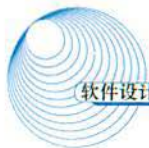
在 Internet 中, 分布着一些覆盖范围很广的大网络, 这种网络称为“Internet 主干网”, 它们一般属于国家级的广域网。例如我国的 CHINANET 和 CERNET 等就是中国的 Internet 的主干网。主干网一般只延伸到一些大城市或重要地方, 在那里设立主干网节点。每一个主干网节点可以通过路由器将广域网与局域网连接起来, 一个节点还可以通过另外的路由器与其他局域网互连, 由此形成一种网状结构。

5.5.2 Internet 地址

无论在网上是检索信息还是发送电子邮件, 都必须知道对方的 Internet 地址。它是能唯一确定 Internet 上每一台计算机、每个用户的位置。也就是说, Internet 上每一台计算机、每个用户都有唯一的地址来标识它是谁和在何处, 以方便于几千万个用户、几百万台计算机和成千上万的组织。Internet 地址格式主要有两种书写形式: 域名格式和 IP 地址格式。

1. 域名

域名(Domain Name)通常是用户所在的主机名字或地址。域名格式是由若干部分组成, 每个部分又称子域名, 它们之间用“.”分开, 每个部分最少由两个字母或数字组成。域名通



常按分层结构来构造,每个子域名都有其特定的含义。通常情况,一个完整、通用的层次型主机域名由4部分组成:

计算机主机名.本地名.组名.最高层域名

从右到左,子域名分别表示不同的国家或地区的名称(只有美国可以省略表示国家的顶级域名)、组织类型、组织名称、分组织名称和计算机名称等。域名地址的最后一部分子域名称为高层域名(或顶级域名),它大致上可以分成两类:一类是组织性顶级域名;另一类是地理性顶级域名。

例如:www.dzkjdx.edu.cn “cn”是地理性顶级域名,表示“中国”。

www.263.net “net”是组织性顶级域名,表示“网络技术组织机构”。

如果一个主机所在的网络级别较高,它可能拥有的域名仅3部分:

本地名.组名.最高层域名

现在,Internet地址管理机构IPRA(Internet PCA Registration Authority)和IANA(Internet Assigned Number Authority)负责Internet最高层域名的登记和管理。

2. IP地址

Internet地址是按名字来描述的,这种地址表示方式易于理解和记忆。实际上,Internet中的主机地址是用IP地址来唯一标识的。这是因为Internet中所使用的网络协议是TCP/IP协议,故每个主机必须用IP地址来标识。

每个IP地址都有4个小于256的数字组成,数字之间用“.”分开。Internet的IP地址共有32位,4个字节。它表示有两种格式:二进制格式和十进制格式。二进制格式是计算机所认识的格式,十进制格式是由二进制格式“翻译”过去的,主要是为了便于使用和掌握。例如,十进制IP地址129.102.4.11的表示方法与二进制的表示方法10000001 01100110 00000100 00001011相同,显然表示成带点的十进制格式则方便得多。

域名和IP地址是一一对应的,域名易于记忆,便于使用,因此得到比较普遍的使用。当用户和Internet上的某台计算机交换信息时,只需要使用域名,网络则会自动地将其转换成IP地址,找到该台计算机。

Internet中的地址可分为5类:A类、B类、C类、D类和E类。各类的地址分配方案如图5-15所示。在IP地址中,全0代表的网络,全1代表的是广播。

A类网络地址占有1个字节(8位),定义最高位为0来标识此类地址,余下7位为真正的网络地址,支持1~126个网络。后面的3个字节(24位)为主机地址,共提供 $2^{24}-2$ 个端点的寻址。A类网络地址第一个字节的十进制值为:000~127。

B类网络地址占有2个字节,使用最高两位为“10”来标识此类地址,其余14位为真正的网络地址,主机地址占后面的2个字节(16位),所以B类全部的地址有: $(2^{14}-2) \times (2^{16}-2) =$

16382×65534 个。B 类网络地址第一个字节的十进制值为: 128~191。



图 5-15 各类地址分配方案

C 类网络地址占有 3 个字节,它是最通用的 Internet 地址。使用最高三位为“110”来标识此类地址,其余 21 位为真正的网络地址,因此 C 类地址支持 $2^{21}-2$ 个网络。主机地址占最后 1 个字节,每个网络可多达 2^8-2 个主机。C 类网络地址第一个字节的十进制值为: 192~223。

D 类地址是相当新的。它的识别头是 1110,用于组播,例如用于路由器修改。D 类网络地址第一个字节的十进制值为: 224~239。

E 类地址为实验保留,其识别头是 1111。E 类网络地址第一个字节的十进制值为: 240~255。

网络软件和路由器使用子网掩码 (Subnet Mask) 来识别报文是仅存放在网络内部还是被路由转发到其他地方。在一个字段内“1”的出现表明一个字段包含所有或部分网络地址。“0”表明主机地址位置。例如,最常用的 C 类地址使用前三个 8 位来识别网络,最后一个 8 位识别主机。因此,子网掩码是 255.255.255.0。

子网地址掩码是相对特别的 IP 地址而言的,如果脱离了 IP 地址就毫无意义。它的出现一般是跟着一个特定的 IP 地址,用来为计算这个 IP 地址中的网络号部分和主机号部分提供依据。换句话说,就是在写一个 IP 地址后,再指明哪些是网络号部分,哪些是主机号部分。子网掩码的格式与 IP 地址相同,所有对应网络号的部分用 1 填上,所有对应的主机号部分用 0 填上。

A 类、B 类、C 类 IP 地址类默认的子网掩码如表 5-2 所示。

如果需要将网络进行子网划分,此时子网掩码可能不同于以上默认的子网掩码。例如,138.96.58.0 是一个 8 位子网化的 B 类网络 ID。基于 B 类的主机 ID 的 8 位被用来表示子网化的网络,对于网络 138.96.39.0,其子网掩码应为 255.255.255.0。



表 5-2 带点十进制符号表示的默认子网掩码

地址类	子网掩码位	子网掩码
A 类	11111111 00000000 00000000 00000000	255.0.0.0
B 类	11111111 11111111 00000000 00000000	255.255.0.0
C 类	11111111 11111111 11111111 00000000	255.255.255.0

例如：一个 B 类地址：172.16.3.4，为了直观地告诉大家前 16 位是网络号，后 16 位是主机号，就可以附上子网掩码：255.255.0.0（11111111 11111111 00000000 00000000）。

假定某单位申请的 B 类地址为 179.143.XXX.XXX。如果希望把它划分为 14（至少占二进制的 4 位）个虚拟的网络，则需要占 4 位作为主机位，子网使用掩码为 255.255.240.0～255.255.255.0 来建立子网。每个 LAN 有可有 $2^{12}-2$ 个主机，且各子网可具有相同的主机地址。

假设一个组织有几个相对大的子网，每个子网包括了 25 台左右的计算机；而又有一些相对较小的子网，每个子网大概只有几台计算机。这种情况下，可以将一个 C 类地址分成 6 个子网（每个子网可以包含 30 台计算机），这样解决了很大的问题。但是出现了一个新的情况，那就是大的子网基本上完全利用了 IP 地址范围，但是小的子网却造成了许多 IP 地址的浪费。为了解决这个新的难题，避免任何可以避免的 IP 浪费，就出现了允许应用不同大小的子网掩码来对 IP 地址空间进行子网划分的解决方案。这种新的方案就叫作可变长子网掩码 VLSM。

VLSM 用一个十分直观的方法来表示，那就是在 IP 地址后面加上“/网络号及子网络号编址比特数”来表示。例如：193.168.125.0/27，就表示前 27 位表示网络号。

例如，给定 135.41.0.0/16 的基于类的网络 ID，所需的配置是为将来使用保留一半的地址，其余的生成 15 个子网，达到 2000 台主机。

由于要为将来使用保留一半的地址，完成了 135.41.0.0 的基于类的网络 ID 的 1 位子网化，生成两个子网，135.41.0.0/17 和 135.41.128.0/17，子网 135.41.128.0/17 被选作为将来使用所保留的地址部分；135.41.0.0/17 被继续生成子网。

为达到划分 2000 台主机的 15 个子网的要求，需要将 135.41.128.0/17 的子网化的网络 ID 的 4 位子网化。这就产生了 16 个子网（135.41.128.0/21，135.41.136.0/21，…，135.41.240.0/21，135.41.248.0/21），允许每个子网有 2046 台主机。最初的 15 个子网化的网络 ID（135.41.128.0/21～135.41.240.0/21）被选定为网络 ID，从而实现了要求。

现在的 IP 协议的版本号为 4，所以也称之为 IPv4，为了方便网络管理员阅读和理解，使用了 4 个十进制数中间加小数点“.”来表示。但随着互联网的发展，IPv4 不论从地址空间上，还是协议的可用性上都无法满足互联网的新要求。这样出现一个新的 IP 协议 IPv6，它使用了 8 个十六进制数中间加小数点“:”来表示。IPv6 将原来的 32 比特地址扩展成为 128 位地址，彻

底解决了地址缺乏的问题。

5.5.3 Internet 服务

作为全世界最大的国际性计算机网络的 Internet, 为全球的科研界、教育界、娱乐界等方面提供了极其丰富的信息资源和最先进的信息交流手段。在 Internet 上, 时刻传送着大量的各种各样的信息, 从电影、实况转播到最尖端的科学研究等无所不包, 当然信息最多的还是科技信息, 如计算机软件、科技论文、图书馆/出版社目录、最新科技动态、电子杂志、产品推销、网络新闻等。而这些内容均可由 Internet 服务来为用户提供。

使用传输控制协议或用户数据报协议时, Internet IP 可支持 65535 种服务, 这些服务是通过各个端口到名字实现的逻辑连接。端口分两类: 一类是已知端口或称公共端口, 端口号从 0 到 1023, 这些端口由 Internet 赋值地址和端口号的组织 IANA (Internet Assigned Number Authority) 赋值; 另一类是需在 IANA 注册登记端口号, 从 1024 到 65535。

前面介绍 Internet 网络接口层、网际层协议和传输层协议, 本节主要介绍 Internet 的高层协议, 如 DNS 域名服务、E-mail 电子邮件服务、WWW 服务、FTP 文件传输服务等。

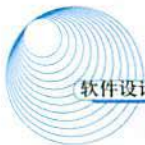
1. DNS 域名服务

Internet 中的域名地址与 IP 地址是等价的, 它们之间是通过域名服务 DNS 来完成映射变换的。实际上, DNS 是一种分布式地址信息数据库系统, 服务器中包含整个数据库的某部分信息, 并供客户查询。DNS 允许局部控制整个数据库的某些部分, 但数据库的每一部分都可通过全网查询得到。

域名系统采用的是客户机/服务器模式, 整个系统由解析器和域名服务器组成。解析器是客户方, 它负责查询域名服务器、解释从服务器返回来的应答、将信息返回给请求方等工作。域名服务器 (Domain Name Server) 是服务器方, 它通常保存着一部分域名空间的全部信息, 这部分域名空间称为区 (Zone)。一个域名服务器可以管理一个或多个区。域名服务器可以分为主服务器、Caching Only 服务器、转发服务器 (Forwarding Server)。

域名系统是一个分布式系统, 其管理和控制也是分布式的。一个用户 A 在查找另一用户 B 时域名系统的工作过程如下:

- (1) 解析器向本地域名服务器发出请求查阅用户 B 的域名。
- (2) 本地域名服务器向最高层域名服务器发出查询地址的请求。
- (3) 最高层域名服务器返回给本地域名服务器一个 IP 地址。
- (4) 本地域名服务器向组域名服务器发出查询地址的请求。
- (5) 组域名服务器返回给本地域名服务器一个 IP 地址。
- (6) 本地服务器向刚返回的域名服务器发出查询域名地址请求。



(7) IP 地址返回给本地域名服务器。

(8) 本地域名服务器将该地址返回给解析器。

因此,本地域名服务器为了得到一个 IP 地址常常需要查询多个域名服务器。于是,在查询地址的同时,本地域名服务器也就得到了许多其他域名服务器的信息,像它们的 IP 地址、所负责的区域等。本地域名服务器将这些信息连同最终查询到的主机 IP 地址全部存放在它的 Cache 中,以便将来参考。当下次解析器再查询与这些域名相关的信息时,就可以直接引用。这样,就大大减少了查询时间。

因此,访问主机的时候只需要知道域名,通过 DNS 服务器将域名变换为 IP 地址,DNS 所用的是 UDP 端口,端口号为 53。

2. Telnet 远程登录服务

远程登录服务是在 Telnet 协议的支持下,将用户计算机与远程主机连接起来,在远程计算机上运行程序,将相应的屏幕显示传送到本地机器,并将本地的输入送给远程计算机,由于这种服务基于 Telnet 协议且使用 Telnet 命令进行远程登录,故称为 Telnet 远程登录。

Telnet 是基于客户机/服务器模式的服务系统,它由客户软件、服务器软件以及 Telnet 通信协议等 3 部分组成。远程计算机又称为 Telnet 主机或服务器,本地计算机作为 Telnet 客户机来使用,它起到远程主机的一台虚拟终端(仿真终端)的作用,通过它用户可以与主机上的其他用户一样共同使用该主机提供的服务和资源。

当用户使用 Telnet 登录远程主机时,该用户必须在这个远程主机上拥有合法的账号和相应的密码,否则远程主机将会拒绝登录。在运行 Telnet 客户程序后,首先应该建立与远程主机的 TCP 连接,从技术上讲,就是在一个特定的 TCP 端口(端口号一般为 23)上打开一个套接字,如果远程主机上的服务器软件一直在这个周知的端口上侦听连接请求,则这个连接便会建立起来,此时用户的计算机就成为该远程主机的一个终端,便可以进行联机操作了,即以终端方式为用户提供人机界面。然后将用户输入的信息通过 Telnet 协议便可以传送给远程主机,主机在周知的 TCP 端口上侦听到用户的请求并处理后,将处理的结果通过 Telnet 协议返回给客户程序。最后客户机接收到远程主机发送来的信息,并经过适当的转换显示在用户计算机的屏幕上。

3. E-mail 电子邮件服务

电子邮件(E-mail)就是利用计算机进行信息交换的电子媒体信件。它是随着计算机网络而出现的,并依靠网络的通信手段实现普通邮件信息的传输。它是最广泛的一种服务。

电子邮件是一种通过计算机网络与其他用户进行联系的快速、简便、高效、价廉的现代化通信手段。如果要想使用 E-mail,必须首先拥有一个电子邮箱,它是由 E-mail 服务提供者为其用户建立在 E-mail 服务器磁盘上的专用于存放电子邮件的存储区域,并由 E-mail 服务器进行管

理。用户将使用 E-mail 客户软件在自己的电子邮箱里来收发电子邮件。电子邮件地址的一般格式：用户名@主机名，例如 fqzhang@china.com。

E-mail 系统基于客户机/服务器模式，整个系统由 E-mail 客户软件、E-mail 服务器和通信协议等 3 部分组成。E-mail 客户软件也称用户代理 (User Agent)，是用户用来收发和管理电子邮件的工具；E-mail 服务器主要充当“邮局”的角色，它除了为用户提供电子邮箱外，还承担着信件的投递业务，当用户发送一个电子邮件后，E-mail 服务器通过网络若干中间节点的“存储—转发”式的传递，最终把信件投递到目的地（收信人的电子邮箱）；E-mail 服务器主要采用 SMTP 协议，本协议描述了电子邮件的信息格式及其传递处理方法，保证被传送的电子邮件能够正确地寻址和可靠地传输，它是面向文本的网络协议，其缺点是不能用来传送非 ASCII 码文本和非文字性附件，在日益发展的多媒体环境中以及人们关注的邮件私密性方面，更显出它的局限性，后来的一些协议，包括多用途 Internet 邮件扩充协议 MIME 及增强私密邮件保护协议 PEM，弥补了 SMTP 协议的缺点。而 SMTP 协议是用在大型多用户、多任务的操作系统环境中，将它用在 PC 机上收信是十分困难的，所以在 TCP/IP 网络上的大多数邮件管理程序使用 SMTP 协议来发信，且采用 POP (Post Office Protocol) 协议（常用的是 POP3）来保管用户未能及时取走的邮件。

POP 协议有两个版本：POP2 和 POP3。目前使用的 POP3 既能与 STMP 共同使用，也可以单独使用，以传送和接受电子邮件。POP 协议是一种简单的纯文本协议，每次传输以整个 E-mail 为单位，不能提供部分传输。

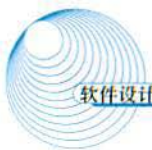
用户要传送 E-mail，首先需在联网的计算机上使用邮件软件编好邮件正文，填好邮件的收信人的 E-mail 地址、发信人电子邮件地址（或自动填上）、邮件的主题等内容，然后使用 E-mail 的发送命令发出。此时，E-mail 发送端与接收端的计算机在工作时并不直接进行通信，而是在发信端计算机送出邮件后，先到达自己所注册的邮件服务器主机，再在网络传输过程中经过多个计算机和路由器的中转，到达目的地的邮件服务器主机，送进收信人的电子邮箱，最后当邮件的接收者上网并启动电子邮件管理程序，它就会自动检查邮件服务器中的电子邮箱，若发现新邮件，便会下载到自己的计算机上，完成接收邮件的任务。

简单邮件传送协议 SMTP 和用于接收邮件的 POP3 协议均是利用 TCP 端口。SMTP 所用的端口号是 25，POP3 所用的端口号是 110。

4. WWW 服务

万维网 WWW (World Wide Web) 是一种交互式图形界面的 Internet 服务，具有强大的信息连接功能，是目前 Internet 中最受欢迎的、增长速度最快的一种多媒体信息服务系统。

万维网是基于客户机/服务器模式的信息发送技术和超文本技术的综合，WWW 服务器把信息组织为分布的超文本，这些信息节点可以是文本、子目录或信息指针。WWW 浏览程序为



用户提供基于超文本传输协议 HTTP (Hyper Text Transfer Protocol) 的用户界面, WWW 服务器的数据文件由超文本标记语言 HTML (Hyper Text Markup Language) 描述, HTML 利用统一资源定位器 URL 的指标是超媒体链接, 并在文本内指向其他网络资源。

超文本传输协议 (HTTP) 是一个 Internet 上的应用层协议, 是 Web 服务器和 Web 浏览器之间进行通信的语言。所有的 Web 服务器和 Web 浏览器必须遵循这一协议, 才能发送或接收超文本文件。HTTP 是客户机/服务器体系结构, 提供信息资源的 Web 节点 (即 Web 服务器) 可称作 HTTP 服务器, Web 浏览器则是 HTTP 服务器的客户。WWW 上的信息检索服务系统就是遵循 HTTP 协议运行的。在 HTTP 的帮助下, 用户可以只关心要检索的信息, 而无需考虑这些信息存储在什么地方。

在 Internet 上, 万维网整个系统由 Web 服务器、浏览器 (Browser) 和 HTTP 通信协议等 3 部分组成。Web 服务器提供信息资源; Web 浏览器将信息显示出来; HTTP 是为分布式超媒体信息系统而设计的一种网络协议, 主要用于域名服务器和分布式对象管理, 它能够传送任意类型数据对象, 以满足 Web 服务器与客户之间多媒体通信的需要, 从而成为 Internet 中发布多媒体信息的主要协议。

统一资源定位器 URL 是在 WWW 中标识某一特定信息资源所在位置的字符串, 是一个具有指针作用的地址标准。在 WWW 上查询信息, 必不可少的一项操作是在浏览器中输入查询目标的地址, 这个地址就是 URL, 也称 Web 地址, 俗称“网址”, 一个 URL 指定一个远程服务器域名和一个 Web 页。换言之, 每个 Web 页都有唯一的 URL。URL 也可指向 FTP、WAIS 和 gopher 服务器代表的信息。通常, 用户只需要了解和使用主页的 URL, 通过主页再访问其他页。当用户通过 URL 向 WWW 提出访问某种信息资源时, WWW 的客户服务器程序自动查找资源所在的服务器地址, 一旦找到, 立即将资源调出供用户浏览。

使用 WWW 的浏览程序 (例如, Internet Explore, Netscape, Mosaic 等), 网页的超文本链接将引导用户找到所需要的信息资源。

如果你已经是 Internet 的用户, 只要在自己的计算机上运行一个客户程序 (WWW 浏览器), 并给出需访问的 URL 地址, 就可以尽情浏览这些来自远方或近邻的各种信息。WWW 工作过程为: 首先通过局域网, 或通过电话拨号连入 Internet, 并在本地计算机上运行 WWW 浏览器程序, 然后根据想要获得的信息来源, 在浏览器的指定位置输入 WWW 地址, 并通过浏览器向 Internet 发出请求信息, 此时网络中的 IP 路由器和服务器将按照地址把信息传递到所要求的 WWW 服务器中, 而 WWW 服务器不断在一个周知的 TCP 端口 (端口号为 80) 上侦听用户的连接请求, 当服务器接收到请求后, 找到所要求的 WWW 页面, 最后服务器将找到的页面通过 Internet 传送回用户的计算机, 浏览器接收传来的超文本文件, 转换并显示在计算机屏幕上。

一个 URL (Web 地址) 包括以下几部分: 协议、主机域名、端口号 (任选)、目录路径 (任选) 和一个文件名 (任选)。其格式为:

scheme://host.Domain[: port]Upath/filename]

其中, scheme 指定服务连接的方式(协议), 通常有下列几种:

- file: 本地计算机上的文件。
- ftp: FTP 服务器上的文件。
- gopher: Gopher 服务器上的文件。
- http: WWW 服务器上的超文本文件。
- new: 一个 USenet 的新闻组。
- telnet: 一个 Telnet 站点。
- wais: 一个 WAIS 服务器。
- mailto: 发送邮件给某人。

在地址的冒号之后通常是两个反斜线, 表示后面是指定信息资源的位置, 其后是一个可选的端口号, 地址的最后部分是路径或文件名。如果端口号默认, 表示使用与某种服务方式对应的标准端口号。根据查询要求不同, 给出的 URL 中目录路径这一项可有可无。如果在查询中要求包括文件路径, 那么, 在 URL 中就要具体指出要访问的文件名称。

下面是一些 URL 的例子:

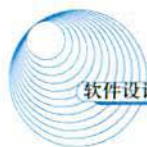
http://www.cctv.com/	中国中央电视台网址
http://www.xjtu.edu.cn/	西安交通大学网址
ftp://ftp.xjtu.edu.cn/	西安交通大学文件服务器
gopher://gopher.xjtu.edu.cn	西安交通大学 Gopher 服务器

5. FTP 文件传输服务

文件传输协议(FTP)用来在计算机之间传输文件。由于 Internet 是一座装满了各种计算机文件的宝库, 其中有免费和共享的软件、各种图片、声音、图像和动画文件, 还有书籍和参考资料等, 如果希望将它们下载到你的计算机上, 其中最主要的方法之一是通过 FTP(文件传输协议)来实现, 因此它是 Internet 中广为使用的一种服务。

通常, 一个用户需要在 FTP 服务器中进行注册, 即建立用户账号, 在拥有合法的登录用户名和密码后, 才有可能进行有效的 FTP 连接和登录。对于 Internet 中成千上万个 FTP 服务器来说, 这就给提供 FTP 服务的管理员带来很大的麻烦, 即需要为每一个使用 FTP 的用户提供一个账号, 这样做显然是不现实。实际上, Internet 的 FTP 服务是一种匿名(anonymous)FTP 服务, 它设置了一个特殊的用户名——anonymous, 供公众使用, 任何用户都可以使用这个用户名与提供这种匿名 FTP 服务的主机建立连接, 并共享这个主机对公众开放的资源。

匿名 FTP 的用户名是“anonymous”, 而密码通常是“guest”或者是使用者的 E-mail 地址。



当用户登录到匿名 FTP 服务器后,其工作方式与常规 FTP 相同。通常,出于安全的目的,大多数匿名 FTP 服务器只允许下载(Download)文件,而不允许上载(Upload)文件。也就是说,用户只能从匿名 FTP 服务器复制所需的文件,而不能将文件复制到匿名 FTP 服务器上。此外,匿名 FTP 服务器中的文件还加入一些保护性措施,确保这些文件不能被修改和删除,同时也可以防止计算机病毒的侵入。

FTP 是基于客户机/服务器模式的服务系统,它由客户软件、服务器软件和 FTP 通信协议 3 部分组成。FTP 客户软件运行在用户计算机上,在用户装入 FTP 客户软件后,便可以通过使用 FTP 内部命令与远程 FTP 服务器采用 FTP 通信协议建立连接或文件传送;FTP 服务器软件运行在远程主机上,并设置一个名叫 anonymous 的公共用户账号,向公众开放。

FTP 在客户与服务器的内部建立两条 TCP 连接:一条是控制连接,主要用于传输命令和参数(端口号为 21);另一条是数据连接,主要用于传送文件(端口号为 20)。FTP 服务器不断在 21 端口的上侦听用户的连接请求,当用户使用 anonymous 的用户名和 guest 或者用户 E-mail 地址的密码进行登录时,用户即发出连接请求,这样控制连接便建立起来,此时,用户名和密码将通过控制连接发送给服务器;服务器接收到这个请求后,便进行用户识别,然后向客户回送确认或拒绝的应答信息;用户看到登录成功的信息后,便可以发出文件传输的命令;服务器从控制连接上接收到文件名和传输命令(如 get)后,便在 20 端口发起数据连接,并在这个连接上将文件名所指明的文件传输给客户。只要用户不使用 close 或者其他命令关闭连接,便可以继续传输其他文件。

6. Gopher

Gopher 系统是采用客户机/服务器的模式,将 Internet 上的信息组织成某种索引,很方便地将用户从一处带向另一处。Gopher 内部集成了 Telnet、FTP 等工具,可以直接取出文件,而无需知道文件的所在处和文件获取工具等细节。目前它的应用正在变小。

5.6 WindowsNT 系统及管理

Windows NT 是 Microsoft 公司推出的 32 位高档网络操作系统。作为网络软件,它不再是操作系统的附加层,而是直接构造一个新的操作系统内核作为 NT 核心成分,即所谓的内置(Build-in)网络。

WindowsNT 既可充当服务器,又可充当客户机。也就是说,它可作为局域网络的服务器,对局域网上客户机提供多种服务;又可作为局域网上的客户机,访问网上任何服务器。此外,Windows NT 为开放的体系结构,与其他网络有很好的互操作性。常用的方式是将 Windows NT Server 作为服务器,将 Windows NT 作为客户机使用。

5.6.1 Windows NT 概述

Windows NT 将网络功能内置在操作系统之中, 因此, 无论是在分布式应用环境中, 还是在点对点网络环境中, Windows NT 都能提供强大的客户机/服务器的计算能力, 使基于 Windows NT 的计算机既可以是客户机, 也可以作为服务器。同时, Windows NT 还提供了在多种网络环境中的相互操作的能力。

Windows NT 网络采用模块化的体系结构。允许用新的模块来替换个别的网络组件, 而不必自底向上提供一组新的组件。

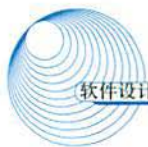
Windows NT 网络的体系结构如图 5-16 所示。Windows NT 的网络结构是按层次组织的, 并提供了扩充能力, 以便以后增加其他功能和服务。在 Windows NT 网络结构中, 通过边界定义了各个层次之间的统一接口, 它有两个主要的边界层: NDIS 和 TDI。

OSI/RM	NT 网络协议体系
应用层	文件 I/O 命名管道 邮件槽
表示层	环境子系统
会话层	重定向器 TDI
传输层	NetBIOS Sockets
网络层	NetBEUI TCP/IP NWLink DLC
数据链路层	NDIS 接口 NDIS 环境与驱动器
物理层	Ethernet/Token Ring 等

图 5.16 Windows NT 网络协议体系与模型

第一个边界层是 NDIS (网络设备接口规范)。它是 Windows NT 的设备驱动程序标准, 允许多个网络适配器 (简称网卡) 和多个网络协议共存。驱动程序是 32 位的, 且是可移植和多处理器相兼容的。在 Windows NT 中, NDIS 在 NDIS.SYS 模块中实现。

第二个边界层是 TDI (传输驱动程序接口)。它提供了驱动程序的通用接口。



1. Windows NT 的网络协议

在 Windows NT 网络中, 提供了如下 4 个网络协议。

(1) DLC 协议。它与 Windows NT 中的其他网络协议(如 NWLink, TCP/IP, NetBEUI)不同, 它是用于访问 IBM 大型机和直接与网络连接的 HP 打印机。

(2) TCP/IP 协议。它是一种广泛使用的网络协议, 且是可路由的, 即支持基于 IP 的网络互连。Windows NT TCP/IP 提供了 TCP/IP 网络环境和服务。

(3) NWLink 协议。它是一种与 IPX/SPX 协议相兼容的网络协议。IPX/SPX 协议是由 Novell 公司为其 NetWare 网络而开发的网络协议, 也是可路由的, 即支持基于 IPX 的网络互连。Windows NT 允许使用 NWLink 协议来建立网络。

(4) NetBEUI。它是 NetBIOS 扩展的用户接口, NetBIOS 是由 IBM 于 1995 年首先推出的。NetBEUI 主要用于小型局域网(LAN), 但不具备网络互连能力。

2. 分布式处理的 IPC 机制

在分布式计算环境中, 计算任务通常分为两部分: 客户和服务端。客户可以通过高性能的服务器来扩展自身的处理能力。在该应用模式中, 客户和服务端之间必须提供网络连接并支持双向数据传送。Windows NT 提供了多种进程间通信(IPC)机制。

(1) 命名管道和邮件槽。命名管道支持面向连接的消息通信机制, 允许应用程序通过网络来共享内存。Windows NT 提供了一种特殊的应用编程接口(API), 以提高使用命名管道时的安全性。邮件槽提供了基于无连接的消息通信机制, 它不保证消息传送的正确性, 主要用于广播消息。Windows NT 下的 Computer Browser 浏览器服务就利用了邮件槽。

(2) NetBIOS。它是在 PC 环境下客户机/服务器应用程序的标准编程接口, 已被用于 IPC 机制。NetBIOS 客户机/服务器应用程序可以通过不同的协议进行网络通信: NetBEUI (NBF), NWLink 上的 NetBIOS (NWBLink) 和 TCP/IP 上的 NetBIOS (NetBT)。从编程的角度来看, 更高层次的接口(例如命名管道和 RPC)具有更好的灵活性和可移植性。

(3) Windows Sockets。它是 Windows 环境下的套接字(Socket)接口, 它为 Windows 环境下的网络编程提供了标准化的 API。

(4) 远程过程调用(RPC)。它提供一种灵活的、可移植的 IPC 机制, 使应用程序能在客户机/服务器环境下的两台计算机之间建立通信。如果客户机和服务器是在同一台计算机上, 则可使用本地过程调用(LPC)机制在进程和子系统之间传送信息。

(5) 网络动态数据交换(NetDDE)。它通过在应用程序之间打开的两个单向管道来提供信息共享能力。NetDDE 是动态数据交换(DDE)的扩充, DDE 在跨越网络的计算机之间使用。在默认情况下, NetDDE 服务不是自动启动的, 可以使用 Control Panel Services, Command

Prompt 或 Windows NT Server Manager 等实用程序来启动 NetDDE 服务。

3. Windows NT 网络模型

Windows NT 可以组成两种模型的网络：域模型和工作组模型。

1) 域模型

域模型 (Domain) 是管理和安全性控制都集中的网络方案，如图 5-17 所示。

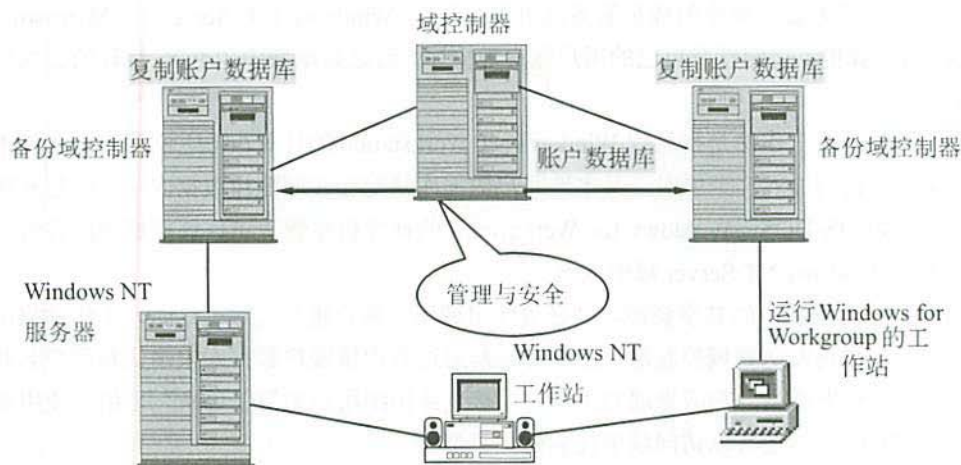
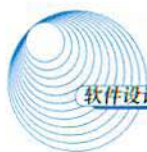


图 5-17 域模型

它采用基于客户机/服务器的结构来组织和管理网络，它使用一组共享公用账户数据库及其安全性策略的计算机的逻辑连接，通过一个公共的账户数据库将这些计算机进行安全性控制和集成化管理。每个域都有唯一的名称，并由一个域控制器进行管理。Windows NT 域中的账户数据库是由域控制器统一管理的，并且所有成员都共享一个账户数据库，因此，Windows NT Server 具有统一管理的域、账户权限、安全性和网络资源的能力。域模型共有 4 种：单域模型、主域模型、多主域模型和完全信任模型。他们应用于不同的场合，例如在一个较大的网络中，由于用户账号较多，它们所属组织和职能不同或地理范围不同，要共享的资源也不同，相互业务联系有松有紧，这时可将网络设置成多主域模型，将同属于一个组织的用户设置为一个域，当一个域内的用户需要访问另一个域中的资源时可在两域之间建立信任关系，从而实现跨域访问。

一个 Windows NT Server 的域主要由 3 种组件来构成：

(1) 域控制器。必须由运行 Windows NT Server 的服务器来充当域控制器，主要负责维护全域的账户数据库。域控制器可分成主域控制器 (PDC) 和备份域控制器 (BDC)。



① 主域控制器 PDC: 它是在 Windows NT Server 安装过程中首先建立有关域信息的计算机, 其中包含域的账户和安全规划的原始表。一个域中只能有一个 PDC, 所有账户数据库的修改都要在 PDC 上进行。当 PDC 不在网络中工作时, 则账户数据库不能做任何修改。

② 备份域控制器 BDC: 它要安装在由 PDC 构成的域中。PDC 将周期性地复制域账户数据库信息给 BDC。BDC 可以协助 PDC 进行域用户登录的确认, 以减轻 PDC 的负担。在一个域中可以有多个 BDC。如果 PDC 失效, 管理员可以将 BDC 升级到 PDC。

(2) 成员服务器。域中的成员服务器可以是运行 Windows NT Server 或 Microsoft LAN Manager 的计算机。它们没有自己的用户账户数据库, 而是共享域中由 PDC 管理的公共的账户数据库。

(3) 工作站。工作站是指运行 Windows NT Workstation 的计算机, 它们都有自己的本地账户数据库。当它们加入一个域时, 其本地账户数据库将附加在域账户数据库中。运行其他客户操作系统, 如 MS-DOS, Windows for Workgroups 的计算机尽管也可以连接域中的资源, 但它们不能成为 Windows NT Server 域的成员。

当用户要访问域中的共享资源, 则必须使用域用户账户进行登录。这时, 工作站将用户输入的用户名和密码发送到域控制器, 由域控制器通过查询域账户数据库来确认其用户名和密码的合法性, 并给出适当的响应来通知工作站, 接受或拒绝用户的登录请求。当用户使用域用户账户登录到网络后, 便可以访问域中任意的共享资源。

2) 工作组模型

工作组模型是将资源、管理和安全性都分散在网络各个计算机上的网络方案, 如图 5-18 所示。Windows NT Workstation 主要用于建立工作组网络环境。

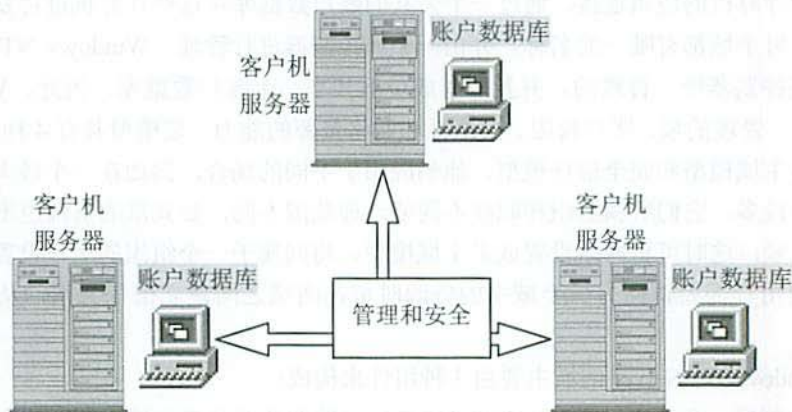


图 5-18 工作组模型

工作组是一组计算机的逻辑连接,它们为共同的目的而组合在一起。例如,同一部门的计算机组成一个工作组,通过工作组的网络连接,使工作组的成员可以访问工作组中其他计算机共享的资源,这种网络也称对等式网络或点对点式网络。工作组的特点是:工作组中的每个计算机具有自己管理的账户数据库,每个工作组有唯一的标识名称,无登录检验,用户数量有限以及管理分散等。

工作组模型中的每一台计算机都可以既作为服务器,又可以作为工作站,并有自己的账号、管理和安全策略。

工作组模型的优点:对少量较集中的工作站很方便、容易共享分布式的资源。管理员维护工作少、实现简单。但也存在一些缺点:对工作站数量较多的网络不合适,无集中式的账号管理、无集中式的资源管理、无集中式的安全性。

5.6.2 WindowsNT 系统管理

Windows NT 的系统管理包括网络用户管理、用户环境管理和网络资源管理。

1. 网络用户管理

1) 用户账号管理

在 Windows NT 网络中有两种用户账号:域用户账号和非域用户账号。域用户账号是建立在 Windows NT Server 域控制器上,由域管理员管理的;非域用户账号是建立在 Windows NT Workstation 计算机上或者建立在 Windows NT Server 成员服务器计算机上,由它们的计算机管理员管理。在安装 Windows NT Server/Workstation 时,系统将自动建立两个内置管理员(Administrator)账号和来宾(Guest)账号,并授予不同的权限。前者权限最大,用于管理其他用户账号和网络资源;后者权限最小,用于临时使用网络。

Windows NT Server 网络的每个用户要登录到网络中并访问网络资源,都需要在服务器上拥有一个账号,对网络中所有资源的访问权限,都是通过用户账号赋予的。

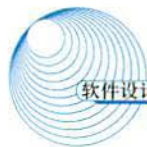
一个 Windows NT 网络中的用户账号包括用户登录时所必须的用户名和用户密码,用户所属的组以及用户使用此系统的权限。另外,用户账号还包括用户全称、用户账号描述、用户环境登录文件、登录工作站列表以及登录时间等信息。

2) 组账号管理

组账号是用户账号的集合。组账号的用途在于简化对用户账号的管理,因为将权限授予组后,组中的所有成员都会可以自动地继承这些权限。组账号也分成全局和本地两种。

全局组是由一个域中的多个用户账号组成,且只能包含该域的用户账号。全局意味着该组可以接受多个域的授权,通过建立委托关系去访问资源域中的资源。

本地组是由一个或多个域中的全局组或用户账号组成,但它不能包含其他域的本地组。本



地组意味着该组只能接受本域的授权,并且只有处于账号域中的全局组或用户账号才能加入到本地组中。

2. 用户环境管理

用户环境是指用户桌面的工作环境,如屏幕颜色、鼠标设置、窗口尺寸和位置以及网络连接和打印机连接等。可以使用下列工具配置和管理用户环境:

(1) 用户配置文件。用户配置文件用于设置用户桌面工作环境,如显示器设置和网络连接等。用户的特定设置将自动保存到用户配置文件中。

(2) 系统策略编辑器。通过建立系统规则来控制用户工作环境及系统配置,如更改桌面设置和限制用户在桌面上的操作。

(3) 登录脚本。登录脚本由批处理文件或可执行文件组成。用户每次登录时,系统都要运行该文件。登录脚本中可以包含建立网络连接、启动应用程序等操作系统命令。

(4) 环境变量。环境变量指定计算机的搜索路径、临时文件目录和其他类似的信息。

3. 网络资源管理

1) 网络资源的共享

Windows NT Server 为网络用户提供了共享目录和共享打印机的资源共享服务。共享目录服务允许有权限的用户访问共享目录及其文件;共享打印机服务允许用户使用网络打印机进行打印。共享目录是用户通过网络访问文件的唯一方法。在设置共享目录时,必须指定一个共享名来代表该目录,共享名可以和实际目录名相同,但不是必须的。理论上,用户可以访问共享目录下的所有子目录与文件。然而,对于在 NTFS 上的共享目录,可以使用 NTFS 目录权限来控制对共享目录下某些目录的访问。

在 Windows NT 上,用户可以通过在“网上邻居”中双击网络上的计算机名查看共享名。

2) 网络资源访问控制

资源共享后,可以通过设置访问权限来控制某些用户对共享资源的访问,建立适当的网络资源访问安全级。在 Windows NT Server 上,如果采用的是 NTFS 文件系统,则可使用“资源管理器”设置目录和文件权限,这些权限适用于在该计算机上工作的用户;在目录共享的情况下,也适用于通过网络访问这些目录和文件的用户。如果采用的是 FAT 文件系统,则无法控制在该计算机上工作的用户,因为 FAT 中的文件总是可读取并可更改的。但可以通过为共享目录设置权限来控制通过网络访问共享目录的用户,这种权限称为共享权限。可见,NTFS 安全性要优于 FAT。

5.7 网络安全

随着网络的发展,网络的安全性显得非常重要。这是由于怀有恶意的攻击者窃取、修改网络上传输的信息,通过网络冒充合法用户非法进入远程主机,获取存储在主机中的机密信息,或者占用网络资源,阻止其他用户使用等。这些问题的产生对网络运营部门和用户的信息安全构成了威胁,影响了计算机网络的进一步推广应用。因此如何对计算机网络上的各种非法行为进行主动控制和有效防御,是计算机网络安全亟待解决的问题。

5.7.1 网络安全概述

计算机网络安全是指计算机、网络系统的硬件、软件以及系统中的数据受到保护,不因偶然的或恶意的原因而遭到破坏、更改、泄露,确保系统能连续和可靠地运行,使网络服务不中断。网络安全从本质上来讲就是网络上的信息安全。它涉及的领域相当广泛。这是因为在目前的各种通信网络中存在着各种各样的安全漏洞和威胁。从广义来说,凡是涉及到网络上信息的保密性、完整性、可用性、真实性和可控性的相关技术和理论,都是网络安全所要研究的领域。

1. 产生威胁网络安全的主要原因

网络的安全之所以能受到威胁,主要原因有以下几个方面:

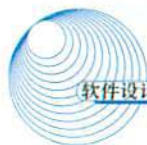
(1) 计算机存储和处理是有关国家安全的政治、经济、军事和国防的情况以及一些部门、机构、组织的秘密信息或是个人的敏感信息、隐私,因此便成为某类人攻击的目标。

(2) 随着 Internet 的发展,存取控制、逻辑连接的数目不断地增加,软件规模不断地膨胀,很容易使系统存在缺陷。

(3) 在网络中,信息是从一台计算机的存储系统流向另一台计算机的存储系统,在大多数的情况下,信息离开信息源后必须经过多个中间节点才能到达目的地。在整个传输过程中,信息的发送者和接收者只能对发送和接收过程加以控制,而对中间传输过程则无权进行有效地控制。如果信息传输路由中存在不可信或具有攻击者的中间节点,信息的安全性就会受到严重的威胁。信息可能会被修改、破坏或泄露,因此,计算机网络的运行机制存在着严重的安全隐患。

(4) 计算机网络的运行机制是协议控制机制,不同的节点之间的信息交换是按照事先定义好的固定机制,通过交换协议数据单元完成的。对于每个节点来说,通信即意味着对一系列从网络上到达的协议数据单元进行响应。根据以上的分析,这些从网上到达的协议数据单元的真实性是无法保证的。同时,由于协议本身固有的安全漏洞或协议实现中产生的安全漏洞也会造成许多安全问题。

在现有的网络环境中,由于存在不同的操作系统、不同厂家的硬件平台,故增加了网络安



全的复杂性,其中有技术上和管理上的诸多原因。一个好的安全的网络应该是由主机系统、应用和服务、路由、网络、网络管理及管理制度等诸多因素决定的。系统、各种服务以及应用软件的漏洞随着时间的推延会越来越多地被发现,然后被修补,随着系统的升级会有许多新的漏洞又出现,再进行修补。因此,这是个长期的循环往复的过程。

2. 网络安全涉及的主要内容

(1) 运行系统安全。即保证信息处理和传输系统的安全。包括计算机系统机房环境和系统设备的保护、计算机结构设计上的安全性考虑、硬件系统的可靠安全与运行、计算机操作系统和应用软件的安全、数据库系统的安全、电磁信息泄漏的防护等。

(2) 信息系统的安全。包括用户口令鉴别、用户存取权限限制、方式控制、安全审计、安全问题跟踪、计算机病毒防治、数据加密。

(3) 信息传播的安全。信息传播后果的安全,包括信息过滤、不良信息的过滤等。它侧重于防止和控制非法、有害的信息传播的后果。

(4) 信息内容的安全。即狭义的信息安全。它侧重于信息的保密性、真实性、完整性。避免攻击者利用系统的安全漏洞进行窃听、冒充、诈骗等有益于合法用户的行为。本质上是保护用户的利益和隐私。

3. 信息系统对安全的基本需求

(1) 保密性。保护资源免遭非授权用户“读出”。包括:传输信息的加密,存储信息加密,防电磁泄露。

(2) 完整性。保护资源免遭非授权用户“写入”。包括数据完整性、软件完整性、操作系统完整性、内存及磁盘完整性、信息交换的真实性和有效性。

(3) 可用性。保护资源免遭破坏或干扰。防止病毒入侵和系统瘫痪,防止信道拥塞及拒绝服务,防止系统资源被非法抢占。

(4) 可控性。对非法入侵提供检测与跟踪并能干预其入侵行为。

(5) 可核查性。可追查安全事故的责任人。对违反安全策略的事件提供审计手段,能记录和追踪他们的活动。

4. 网络的安全威胁主要类型

(1) 物理威胁。物理威胁是指用计算机硬件和存储介质不受到偷窃、废物搜寻及歼敌活动。这里的废物搜寻者就像捡破烂者,不过他在废纸篓中搜寻的是机密信息。

(2) 网络攻击。计算机网络的使用对数据造成了新的安全威胁。攻击者可通过网络上存在着电子窃听、入侵拨号入网、冒名顶替等方式进行入侵攻击、偷窃与篡改。

(3) 身份鉴别。由于身份鉴别通常是用设置口令的手段实现的, 入侵者可通过口令圈套、密码破译等方法扰乱身份鉴别。

(4) 编程威胁。所谓编程威胁是指通过病毒进行攻击的一种方法。由于病毒是一种能进行自我复制的代码, 在网络间不断传播更具有危害性。

(5) 系统漏洞。系统漏洞也称为系统陷阱或代码漏洞, 这通常源于操作系统设计者有意设置的, 目的是为了使用户在失去对系统的访问权时, 仍有机会进入系统。入侵者可使用扫描器发现系统陷阱, 从而进行攻击。

5.7.2 网络的信息安全

网络安全涉及到许多内容, 如系统的软硬件的安全, 对网内的服务器与 PC 机使用开机热启动密码、带复杂口令的屏幕保护且启动屏保时间不要太长、尽量不要使用文件共享功能(如需要, 应设置访问控制权限和密码)、对重要文件设标识与验证置读写口令等; 严格控制管理员级的账户的使用范围、随时更新系统各种密码、规定可以访问的用户数。网络的信息安全归纳起来主要就是信息的存储安全和传输安全。

1. 信息的存储安全

信息的存储安全包括使用信息使用的安全(如用户的标识与验证、用户存取权限限制、安全问题跟踪等)、计算机病毒防治、系统安全监控、数据的加密、防止非法的攻击等。

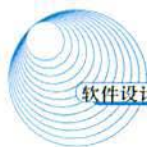
1) 用户的标识与验证

用户的标识与验证主要是限制访问系统的人员。它是访问控制的基础, 是对用户身份的合法性验证。方法有两种: 一是基于人的物理特征的识别, 包括签名识别法、指纹识别法、语音识别法; 二是基于用户所拥有特殊安全物品的识别, 包括智能 IC 卡识别法、磁条卡识别法。

2) 用户存取权限限制

用户存取权限限制主要是限制进入系统的用户所能做的操作。存取控制是对所有的直接存取活动通过授权进行控制以保证计算机系统安全保密机制, 是对处理状态下的信息进行保护。一般有两种方法, 一是隔离控制法, 二是限制权限法。

(1) 隔离控制法: 是在电子数据处理成分的周围建立屏障, 以便在该环境中实施存取规则。隔离控制技术的主要实现方式包括物理隔离方式、时间隔离方式、逻辑隔离方式、密码技术隔离方式等。其中物理隔离方式各过程使用不同的物理目标, 是一种有效的方式。传统的多网环境一般通过运行两台计算机实现物理隔离。现在我国已经生产出了拥有自主知识产权的涉密计算机, 它采用双硬盘物理隔离技术, 通过运行一台计算机, 即可在物理隔离的状态下切换信息网和公共信息网, 实现一机双网或一机多网的功能。还有另外一种方式就是加装隔离卡, 一块隔离卡带一块硬盘、一块网卡, 连同本机自带的硬盘网卡, 使用不同的网络环境。当然, 物理



隔离方式对于系统的要求比较高,必须采用两整套互不相关的设备,其人力、物力、财力的投入都是比较大的。但也是很有效的方式,因为两者就如两条平行线,永不交叉,自然也就安全了。

(2) 限制权限法:是有效地限制进入系统的用户所进行的操作。即对用户进行分类管理,安全密级授权不同的用户分在不同类别;对目录、文件的访问控制进行严格的权限控制、防止越权操作;放置在临时目录或通信缓冲区的文件要加密,用完尽快移走或删除。

3) 系统安全监控

系统必须建立一套安全监控系统,全面监控系统的活动,并随时检查系统的使用情况,一旦有非法入侵者进入系统,能及时发现并采取相应措施,确定和堵塞安全及保密的漏洞。应当建立完善的审计系统和日志管理系统,利用日志和审计功能对系统进行安全监控。管理员还应当经常做到:监控当前正在进行的进程,正在登录的用户情况;检查文件的所有者、授权、修改日期情况和文件的特定访问控制属性;检查系统命令安全配置文件、口令文件、核心启动运行文件、任何可执行文件的修改情况;检查用户登录的历史记录和超级用户登录的记录。如发现异常,及时处理。

4) 计算机病毒防治

计算机网络服务器必须加装网络病毒自动检测系统,以保护网络系统的安全,防范计算机病毒的侵袭,并且必须定期更新网络病毒检测系统。

由于计算机病毒具有隐蔽性、传染性、潜伏性、触发性和破坏性等特点,所以需要建立计算机病毒防治管理制度:

- (1) 经常从软件供应商网站下载、安装安全补丁程序和升级杀毒软件。
- (2) 定期检查敏感文件。对系统的一些敏感文件定期进行检查,保证及时发现已感染的病毒和黑客程序。
- (3) 使用高强度的口令。尽量选择难于猜测的口令,对不同的账号选用不同的口令。
- (4) 经常备份重要数据。要做到每天坚持备份。
- (5) 选择、安装经过公安部认证的防病毒软件,定期对整个硬盘进行病毒检测、清除工作。
- (6) 可以在计算机和互联网之间安装使用防火墙,提高系统的安全性。
- (7) 当计算机不使用时,不要接入互联网,一定要断掉连接。
- (8) 重要的计算机系统和网络一定要严格与互联网物理隔离。
- (9) 不要打开陌生人发来的电子邮件,无论它们有多么诱人的标题或者附件。同时也要小心处理来自于熟人的邮件附件。
- (10) 正确配置系统和使用病毒防治产品。正确配置系统,充分利用系统提供的安全机制,提高系统防范病毒的能力,减少病毒侵害事件。了解所选用防病毒产品的技术特点,正确配置以保护自身系统的安全。

5) 数据的加密

数据的加密主要是防止非法窃取或调用。包括文件信息的加密、数据库数据的安全与加密、磁介质加密等。

(1) 文件信息的加密：一是文件加密，即对文件的代码或数据本身进行的数据源加密；二是文件名加密，就是利用文件名屏幕显示形式的变换，使得实际注册的文件名与显示的不相符或者根本不显示，造成无法访问文件。

(2) 数据库数据的安全与加密：通过对数据库系统的管理和对数据库数据的加密来实现。包括用户身份的识别和确认、访问操作的鉴别和控制、审计和跟踪、数据库外加密、数据库内加密等。

(3) 磁介质加密：加密的目的在于防复制。磁介质加密的主要方法包括固化部分程序、激光穿孔加密、掩膜加密和芯片加密、修改磁介质参数表等。

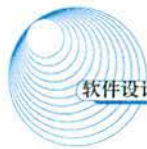
6) 计算机网络安全

计算机网络安全主要是指计算机网络抵御来自外界侵袭等应采取的安全措施。它是网络信息安全的最外一层防线。目前主要通过采用安全防火墙系统、安全代理服务器、安全加密网关等来实现。主要包括网络边界的安全和网络内部的安全控制和防范。

(1) 网络边界的安全。网络边界主要是指本单位(或部门)的网络与外界网络或 Internet 网互连的出口边界。其安全主要是针对经边界进出访问和传输数据包时要采取的控制和防范措施。政府各部门的计算机网络应当采用统一的国际互联网出口，以便加强管理。计算机网络与 Internet 网或外界其他网络接入口处必须设置的安全防火墙系统，该防火墙要具有加密功能或安全加密网关。要定期扫描网络的安全漏洞，及时消除网络安全的隐患。Internet 网或外界其他网络上的授权用户要通过安全防火墙或安全加密网关远程进入政府网络时，必须配备电子印章认证系统，只有认证系统通过的授权用户才可进入。

计算机网络系统一般不要设置拨号访问服务器和提供远程 Modem 接入，如确需设置，必须采用如下措施：设置访问控制服务器，对拨号上网的用户身份、电话号码等进行验证；要求拨号用户采用比较安全的口令，并确保不把用户名和口令外传给任何其他人；在拨号访问服务器和网络之间设置安全防火墙，对远程访问进行控制和监测；对拨号上网的电话号码严格保密。

(2) 网络内部的安全控制和防范。网络内部安全是指应采取防范措施以控制外界远程用户(或网络)对网络内部数据存取。常用的技术手段包括网络安全监测报警系统、数据加/解密卡、电子印章系统等。具体应采取以下措施：在网络中应采取对信息进行相应级别的数据源加密；对信息所需采用的各种加解密设备采用统一的加密算法和密钥管理，并具有权限分级，以适合不同级别的用户存取。同时密钥必须定期更换；加装网络安全监测报警系统，定期扫描网络的安全漏洞，一旦有非法用户进入网络读取信息或篡改网络系统配置，则自动报警；必须对计算机网络用户读取信息进行身份认证，并由此获得相应身份的读取权限，以适应不同级别的用户



读取不同级别的信息；未经或通不过系统认证的用户，将无权读取信息，网络边界安全设备将禁止信息传出网络；信息从网络边界传输出去以前，必须经过相应的数据源加密后才能传输，否则将无法通过网络边界安全设备。

2. 信息的传输安全

1) 加密的主要方式

信息的传输加密是面向线路的加密措施，有链路加密、节点加密和端-端加密3种。

(1) 链路加密：链路加密只对两个节点之间（不含信息源和目的地两个端点本身）的通信信道线路上传输的信息进行加密保护。它是一种链式连接的加密方式，属于公共加密。其缺点是：每个节点要配置加密单元（信道加密机），相邻节点必须使用相同的密钥，节点的数据是明文。

(2) 节点加密：节点加密的加/解密都在节点中进行，即每个节点里装有加、解密的保护装置，用于完成一个密钥向另一个密钥的转换。这样，节点中的数据不会出现明文。但由于每个节点要加装安全单元或保护装置，因此需要公共网络提供配合。

(3) 端-端加密：端-端加密为系统网络提供从信息源到目的地传送的数据的加密保护。可以是主机到主机，终端到终端，终端到主机或到处理进程，或从数据的处理进程到处理进程，而不管数据在传送中经过了多少中间节点，数据不被解密。用户或主机都可独自采用这种加密技术而不会影响别的用户或主机，这比较适于在分组交换网中采用。

加密措施是保护信息的最后防线，被公认为是保护信息传输唯一实用的方法。在物理安全不足的地方，也是保护存储信息的十分有效而经济的方法。对信息进行加密保护是在密钥的控制下，通过密码算法将敏感的机密明文数据变换成不可懂的密文数据，称加密（Encryption）。对一些重要的口令、数据、电子邮件用加密软件进行加密后，再发送或保存。信息即使被截获，攻击者也要耗费时间、精力去解密，从而为采取补救措施赢得了时间。

2) 基本加密算法

信息的传输加密技术是指发送方用加密密钥，通过加密设备或算法，将信息加密后发送出去。接收方在收到密文后，用解密密钥将密文解密，恢复为明文。如果传输中有人窃取，他只能得到无法理解的密文，从而对信息起到保密作用。基本的加密算法有两种：对称密钥加密和非对称密钥加密，用于保证电子商务中数据的保密性、完整性、真实性和非抵赖服务。

(1) 对称密钥加密：也叫私有密钥加密（Private Key Encryption），即发送和接收数据的双方必须使用相同的/对称的密钥对明文进行加密和解密运算。最著名的对称密钥加密标准是数据加密标准（Data Encryption Standard，简称 DES）。DES 是一种使用 56 个数据位的密钥来操作 64 位数据块的块加密算法，由 IBM 公司推出，可同时对大量数据进行快速加密。DES 算法曾

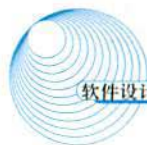
经过广泛的分析和测试，被认为是一种非常安全的系统。采用这种方法的主要问题是密钥的生成、注入、存储、管理、分发等很复杂，特别是随着用户的增加，密钥的需求量成倍增加。在网络通信中，大量密钥的分配是一个难以解决的问题。对称密钥加密的特点是简单，但用户必须也只能让接收人知道自己使用的密钥，双方必须共同保守密钥，任何一方失误均会导致密钥的泄露。目前已有一些比 DES 算法更安全的对称密钥加密算法，如：IDEA 算法，RC2、RC4 算法，Skipjack 算法等。

(2) 非对称密钥加密：也叫公开密钥加密 (Public Key Encryption)，由美国斯坦福大学赫尔曼教授于 1977 年提出。它主要指每个人都有唯一一对对应的密钥：公开密钥和私有密钥，公钥对外公开，私钥由个人秘密保存；用其中一把密钥来加密，就只能用另一把密钥来解密。加密密钥对外是公开，使任何用户都可将传送给此用户的信息用公开密钥加密发送，而用户唯一保存的私人密钥是保密的，也只有它能将密文复原、解密。虽然解密密钥理论上可由加密密钥推算出来，但这种算法设计在实际上是不可能的，或者虽然能够推算出，但要花费很长的时间而成为不可行的。所以将加密密钥公开也不会危害密钥的安全。公开密钥加密技术解决了密钥的发布和管理问题。使用公开密钥技术，进行数据通信的双方可以安全地确认对方身份和公开密钥，提供通信的可鉴别性。非对称加密算法主要有 RSA、DSA、Diffie-Hellman、PKCS、PGP 等。

5.7.3 防火墙技术

所谓防火墙 (Firewall) 是建立在内外网络边界上的过滤封锁机制，它认为内部网络是安全和可信的，而外部网络被认为是不安全和不可信的。防火墙的作用是防止未经授权的访问进入被保护的内部网络，通过边界控制强化内部网络的安全策略。它的实现有多种形式，但原理很简单，可以把它想象为一对开关，其中一个用来阻止传输，另一个用来允许传输。防火墙作为网络安全体系的基础和核心控制设备，它贯穿于受控网络通信主干线，对通过受控干线的任何通信行为进行安全处理，如控制、审计、报警、反应等，同时也承担着繁重的通信任务。由于其自身处于网络系统中的敏感位置，自身还要面对各种安全威胁，因此，选用一个安全、稳定和可靠的防火墙产品，其重要性不言而喻。

在网络层，防火墙被用来处理信息在内外网络边界的流动，它可以确定来自哪些地址的信息可以通过或者禁止哪些目的地址的主机。在传输层，这个连接可以被端到端的加密，也就是进程到进程的加密。在应用层，它可以进行用户级的身份认证、日志记录和账号管理等。因此防火墙技术简单说就是一套身份认证、加密、数字签名和内容检查集成一体的安全防范措施。所有来自 Internet 的传输信息和内部网络发出的传输信息都要穿过防火墙，由防火墙进行分析，以确保它们符合站点设定的安全策略，以提供一种内部节点或网络与 Internet 的安全屏障。



1. 防火墙的分类

防火墙技术经历了包过滤、应用代理网关和状态检测 3 个发展阶段。包过滤型的防火墙通常直接转发报文,它对用户完全透明,速度较快;应用代理网关防火墙是通过服务器建立连接的,可以有更强的身份验证和注册功能;状态检测防火墙是在其核心部分建立状态连接表,并将进出网络的数据当成一个个的会话,利用状态表跟踪每一个会话状态。状态监测对每一个包的检查不仅根据规则表,更考虑了数据包是否符合会话所处的状态,因此提供了完整的对传输层的控制能力。

1) 包过滤型防火墙

包过滤防火墙一般有一个包检查块(通常称为包过滤器),数据包过滤可以根据数据包头中的各项信息来控制站点与站点、站点与网络、网络与网络之间的相互访问,但无法控制传输数据的内容,因为内容是应用层数据,而包过滤器处在网络层和数据链路层(即 TCP 和 IP 层)之间。通过检查模块,防火墙能够拦截和检查所有出站和进站的数据,它首先打开包,取出包头,根据包头的信息确定该包是否符合包过滤规则,并进行记录。对于不符合规则的包,应进行报警并丢弃该包。

包过滤防火墙工作在网络层,对数据包的源 IP 地址及目的 IP 地址具有识别和控制作用,对于传输层,也只能识别数据包是 TCP 还是 UDP 及所用的端口信息。由于只对数据包的 IP 地址、TCP/UDP 协议和端口进行分析,如果一条规则阻止包传输或接收,则此包便不被允许通过,否则该包可以被继续处理。包过滤防火墙的处理速度较快,并且易于配置。

(1) 包过滤防火墙的优点:防火墙对每条传入和传出网络的包实行低水平控制;每个 IP 包的字段都被检查,例如源地址、目的地址、协议、端口等;防火墙可以识别和丢弃带欺骗性源 IP 地址的包;包过滤防火墙是两个网络之间访问的唯一来源;包过滤通常被包含在路由器数据包中,不需要额外的系统来处理。

(2) 包过滤防火墙缺点:不能防范黑客攻击,因为网管不可能区分出可信网络与不可信网络的界限;不支持应用层协议,因为它不认识数据包中的应用层协议,访问控制粒度太粗;不能处理新的安全威胁。

2) 应用代理网关防火墙

应用代理网关防火墙彻底隔断内网与外网的直接通信,内网用户对外网的访问变成防火墙对外网的访问,然后再由防火墙转发给内网用户。所有通信都必须经应用层代理软件转发,访问者任何时候都不能与服务器建立直接的 TCP 连接,应用层的协议会话过程必须符合代理的安全策略要求。

(1) 应用代理网关的优点:可以检查应用层、传输层和网络层的协议特征,对数据包的检测能力比较强。

(2) 应用代理网关的缺点:

① 难于配置。由于每个应用都要求单独的代理进程,这就要求网管能理解每项应用协议的弱点,并能合理地配置安全策略,由于配置繁琐,难于理解,容易出现配置失误,最终影响内网的安全防范能力。

② 处理速度非常慢。断掉所有的连接,由防火墙重新建立连接,理论上可以使应用代理防火墙具有极高的安全性。但是实际应用中并不可行,因为对于内网的每个 Web 访问请求,应用代理都需要开一个单独的代理进程,它要保护内网的 Web 服务器、数据库服务器、文件服务器、邮件服务器,及业务程序等,就需要建立一个个的服务代理,以处理客户端的访问请求。这样,应用代理的处理延迟会很大,内网用户的正常 Web 访问不能及时得到响应。

总之,应用代理防火墙不能支持大规模的并发连接,对速度要求高的行业不能使用这类防火墙。另外,防火墙核心要求预先内置一些已知应用程序的代理,使得一些新出现的应用在代理防火墙内被无情地阻断,不能很好地支持新应用。

3) 状态检测技术防火墙

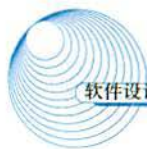
状态检测技术防火墙结合了代理防火墙的安全性和包过滤防火墙的高速度等优点,在不损失安全性的基础上将代理防火墙的性能提高了 10 倍。

Internet 上使用的是 TCP/IP 协议, TCP 协议的每个可靠连接均需要经过“客户端同步请求”、“服务器应答”、“客户端再应答”三次握手。例如最常用到的 Web 浏览、文件下载、收发邮件等都要经过这三次握手。这反映出数据包并不是独立的,而是前后之间有着密切的状态联系,基于这种状态变化,引出了状态检测技术。

状态检测防火墙摒弃了包过滤防火墙仅考查数据包的 IP 地址等几个参数,而不关心数据包连接状态变化的缺点,在防火墙的核心部分建立状态连接表,并将进出网络的数据当成一个个的会话,利用状态表跟踪每一个会话状态。状态监测对每一个包的检查不仅根据规则表,更考虑了数据包是否符合会话所处的状态,因此提供了完整的对传输层的控制能力。状态检测防火墙在提高安全防范能力的同时也改进了流量处理速度。因为它采用了一系列优化技术,使防火墙性能大幅度提升,能应用在各类网络环境中,尤其是在一些规则复杂的大型网络上。

2. 典型防火墙的体系结构

一个防火墙系统通常是由过滤路由器和代理服务器组成。过滤路由器是一个多端口的 IP 路由器,它能够拦截和检查所有出站和进站的数据。代理服务防火墙使用了与包过滤器不同的方法。代理服务器使用一个客户程序与特定的中间节点(防火墙)连接,然后中间节点与期望的服务器进行实际连接。与包过滤器不同的是,使用这种类型的防火墙,内部与外部网络之间不存在直接连接。因此,即使防火墙发生了问题,外部网络也无法获得与被保护的网络的连接。代理提供了详细的注册及审计功能,这大大提高了网络的安全性,也为改进现有软件的安全性



能提供了可能。它是基于特定协议的,如 FTP、HTTP 等。为了通过代理支持一个新的协议,必须改进代理服务器以适应新协议。典型防火墙的体系结构包括过滤路由器、双宿主主机、被屏蔽主机、被屏蔽子网等类型。

1) 包过滤路由器

包过滤路由器又称屏蔽路由器,是最简单也是最常用的防火墙。它一般作用在网络层,对进出内部网络的所有信息进行分析,并按照一定的安全策略(过滤规则)对进出内部网络的信息进行限制。包过滤的核心就是安全策略即包过滤算法的设计。包过滤型防火墙往往可用一台过滤路由器来实现,对所接收的每个数据包做出允许或拒绝的决定。如图 5-19 所示。

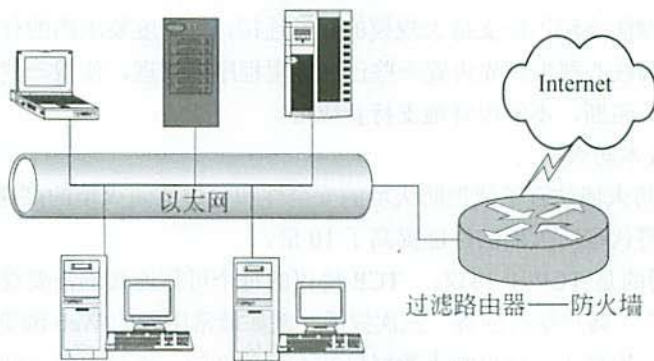


图 5-19 包过滤路由器防火墙

采用包过滤路由器的防火墙优点在于速度快、实现方便。缺点是安全性能差;不同操作系统环境下 TCP 和 UDP 端口号所代表的应用服务协议类型有所不同,故兼容性差;没有或较少的日志记录能力。

2) 双宿主主机

双宿主主机结构是围绕着至少具有两个网络接口的双宿主主机(又称堡垒主机)而构成的,每一个接口都连接在物理和逻辑上分离的不同的网段,代理服务器软件在双宿主主机上运行,如图 5-20 所示。双宿主主机内外的网络均可与双宿主主机实施通信,但内外网络之间不可直接通信,内外网络之间的 IP 数据流被双宿主主机完全切断。主机结构采用主机取代路由器执行安全控制功能,受保护网除了看到堡垒主机外,不能看到其他任何系统。同时堡垒主机不转发 TCP/IP 通信报文,网络中的所有服务都必须由此主机的相应代理程序来支持。

双宿主主机防火墙的优势是:堡垒主机运行的系统软件可用于维护系统日志、硬件复制日志、远程日志等,有利于网络管理员的日后检查。其缺点是:由于唯一隔开内部网和外部因特网之间的屏障,若入侵者得到了双宿主主机的访问权,内部网络就会被入侵,所以为了保证内

部网的安全，双宿主主机首先要禁止网络层的路由功能，还应具有强大的身份认证系统，尽量减少防火墙上用户的账户数目。

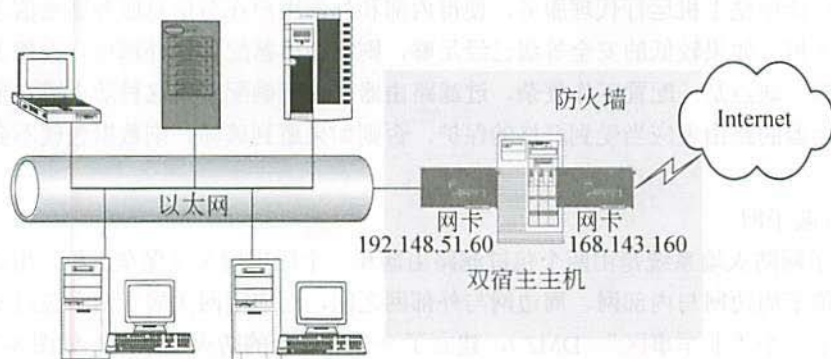


图 5-20 双宿主主机防火墙

3) 屏蔽主机网关

屏蔽主机网关防火墙是由过滤路由器和应用网关组成。过滤路由器的作用是进行包过滤；应用网关的作用是代理服务，即在内部网络与外部网络之间建立两道安全屏障。屏蔽主机网关防火墙的结构如图 5-21 所示。

对于这种防火墙系统，堡垒主机配置在内部网络上，而包过滤路由器则放置在内部网络和 Internet 之间。在路由器上进行规则配置，使得外部系统只能访问堡垒主机，去往内部系统上其他主机的信息全部被阻塞。由于内部主机与堡垒主机处于同一个网络，内部系统是否允许直接访问 Internet，或者是要求使用堡垒主机上的代理服务来访问 Internet 由机构的安全策略来决定。对路由器的过滤规则进行配置，使得其只接受来自堡垒主机的内部数据包，就可以强制内部用户使用代理服务。

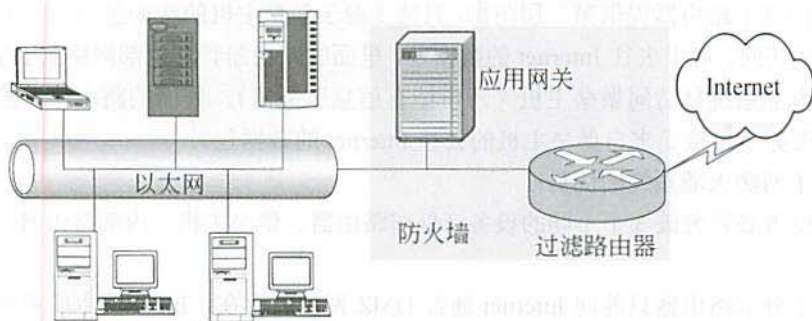
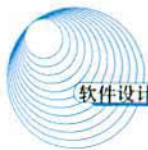


图 5-21 屏蔽主机网关防火墙



屏蔽主机网关防火墙的优点是：安全等级较高。可以提供公开的信息服务的服务器，如 Web、FTP 等，可以放置在由包过滤路由器和堡垒主机共用的网段上。如果要求有特别高的安全特性，可以让堡垒主机运行代理服务，使得内部和外部用户在与信息服务器通信之前，必须先访问堡垒主机。如果较低的安全等级已经足够，则将路由器配置让外部用户直接去访问公共的信息服务器。缺点是：配置工作复杂。过滤路由器是否正确配置是这种防火墙安全与否的关键，过滤路由器的路由表应当受到严格的保护，否则如果遭到破坏，则数据包就不会被路由到堡垒主机上。

4) 被屏蔽子网

被屏蔽子网防火墙系统是由两个包过滤路由器和一个应用网关（堡垒主机）组成。包过滤路由器分别位于周边网与内部网、周边网与外部网之间，而应用网关居于两个包过滤路由器的中间，形成了一个“非军事区”（DMZ），建立了一个最安全的防火墙系统。如图 5-22 所示。

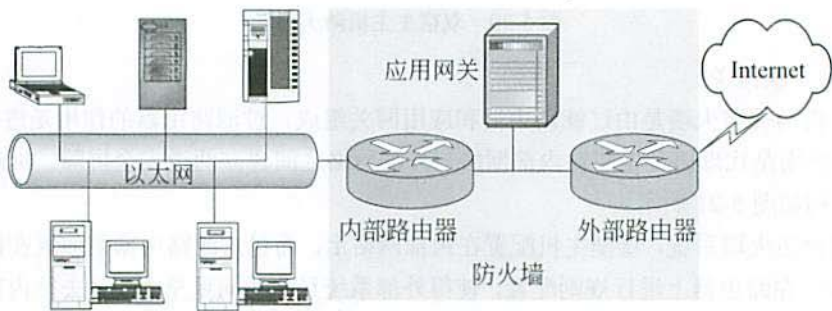


图 5-22 被屏蔽子网防火墙

对于进来的信息，外面的这个路由器用于防范通常的外部攻击（如源地址欺骗和源路由攻击），并管理 Internet 到 DMZ 网络的访问。它只允许外部系统访问堡垒主机（还可能有信息服务器）。里面的这个路由器提供第二层防御，只接受源于堡垒主机的数据包，负责的是管理 DMZ 到内部网络的访问。对于去往 Internet 的数据包，里面的路由器管理内部网络到 DMZ 网络的访问。它允许内部系统只访问堡垒主机（还可能有信息服务器）。外面的路由器上的过滤规则要求使用代理服务（只接受来自堡垒主机的去往 Internet 的数据包）。

被屏蔽子网防火墙系统的优势：

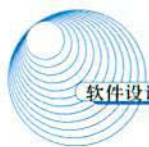
- (1) 入侵者必须突破 3 个不同的设备（外部路由器、堡垒主机、内部路由器）才能侵袭内部网络。
- (2) 由于外部路由器只能向 Internet 通告 DMZ 网络的存在，Internet 上的系统不需要有路由器与内部网络相对。这样网络管理员就可以保证内部网络是“不可见”的，并且只有在 DMZ 网络上选定的系统才对 Internet 开放。

(3) 由于内部路由器只向内部网络通告 DMZ 网络的存在, 内部网络上的系统不能直接通往 Internet, 这样就保证了内部网络上的用户必须通过驻留在堡垒主机上的代理服务才能访问 Internet。

(4) 包过滤路由器直接将数据引向 DMZ 网络上所指定的系统, 消除了双宿堡垒主机的必要。

(5) 内部路由器在作为内部网络和 Internet 之间最后的防火墙系统时, 能够支持比双宿堡垒主机更大的数据包吞吐量。

(6) 由于 DMZ 网络是一个与内部网络不同的网络, NAT (网络地址变换) 可以安装在堡垒主机上, 从而避免在内部网络上重新编址或重新划分子网。



第6章 多媒体基础知识

6.1 多媒体的基本概念

6.1.1 媒体的分类

媒体的概念范围相当广泛,按照国际电话电报咨询委员会(Consultative Committee on International Telephone and Telegraph, CCITT)的定义,媒体可以归类为:

(1) 感觉媒体(Perception Medium):指直接作用于人的感觉器官,使人产生直接感觉的媒体。如引起听觉反应的声音,引起视觉反应的图像等。

(2) 表示媒体(Representation Medium):指传输感觉媒体的中介媒体,即用于数据交换的编码。如图像编码(JPEG、MPEG)、文本编码(ASCII、GB2312)和声音编码等。

(3) 表现媒体(Presentation Medium):指进行信息输入和输出的媒体。如键盘、鼠标、扫描仪、话筒、摄像机等为输入媒体;显示器、打印机、喇叭等为输出媒体。

(4) 存储媒体(Storage Medium):指用于存储表示媒体的物理介质。如硬盘、软盘、磁盘、光盘、ROM及RAM等。

(5) 传输媒体(Transmission Medium):指传输表示媒体的物理介质。如电缆、光缆、电磁波等。

我们通常所说的“媒体(Media)”包括其中的两点含义。一是指信息的物理载体(即存储和传递信息的实体),如手册、磁盘、光盘、磁带以及相关的播放设备等;二是指承载信息的载体即信息的表现形式(或者说传播形式),如文字、声音、图像、动画、视频等,即CCITT定义的存储媒体和表示媒体。表示媒体又可以分为3种类型:视觉类媒体(位图图像、矢量图形、图表、符号、视频、动画)、听觉类媒体(音响、语音、音乐)、触觉类媒体(点、位置跟踪;力反馈与运动反馈)。视觉和听觉类媒体是信息传播的内容,触觉类媒体是实现人机交互的手段。

多媒体就是指多种信息载体的表现形式和传递方式。通常多媒体是对多种媒体的融合,即将音频、视频、图像和计算机技术、通信技术集成到同一数字环境中,以协同表示更丰富和复杂的信息。为了将不同的媒体能够有机地连接起来,往往按照实际需要建立一种链接机制或结构,我们把这种链接机制或结构称为超媒体(HyperMedia)。在计算机中,超媒体是一个信息

存储和检索系统,它把文字、图形、图像、动画、声音、视频等媒体集成为一个相关的基本信息系统。通常,如果信息主要是以文字的形式表示,那么称为超文本;如果还包含有图形、影视、动画、音乐或其他媒体,一般称为超媒体。互联网的 WWW 应用是超媒体技术的最好例子。

6.1.2 多媒体的特征

在计算机领域中,“多媒体”常常被当作为“多媒体技术”的同义词,多媒体技术就是指利用计算机技术把文本、图形、图像、声音、动画和电视等多种媒体综合起来,使多种信息建立逻辑连接,并能对它们进行获取、压缩、加工处理、存储,集成为一个具有交互性的系统。多媒体技术的内涵和范围极其广泛,所涉及的技术也极为广泛,其主要特性为:

(1) 多样性:主要表现在信息媒体的多样化。多样性使得计算机处理的信息空间范围扩大,不再局限于数值、文本或特殊对待的图形和图像,可以借助于视觉、听觉和触觉等多感觉形式实现信息的接收、产生和交流。

(2) 集成性:主要表现在多媒体信息(文字、图形、图像、语音、视频等信息)的集成和操作这些媒体信息的软件和设备的集成。多媒体信息的集成是将各种信息媒体按照一定的数据模型和组织结构集成为一个有机的整体。操作媒体信息的软件和设备的集成是将与多媒体相关的各种硬件和软件集成为一个理想的环境,使得充分共享和操作使用多媒体信息。

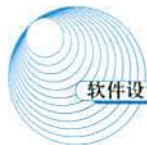
(3) 交互性:这是多媒体应用有别于传统信息交流媒体的主要特点之一。传统信息交流媒体只能单向地、被动地传播信息,而多媒体技术引入交互性后则可实现人对信息的主动选择、使用、加工和控制。通过交互与反馈,达到更加有效的控制和使用信息,为人们提供发挥创造力的环境,增强了人们的参与感,同时也为多媒体技术应用开辟了更加广阔的领域。

(4) 非线性:多媒体技术的非线性特点将改变人们传统循序性的读写模式。以往人们读写方式大都采用章、节、页的框架,循序渐进地获取知识,而多媒体技术将借助超文本链接的方法,把内容以一种更灵活、更具变化的方式呈现给读者。

(5) 实时性:是指在人的感官系统允许的情况下进行多媒体处理和交互。当人们给出操作命令时,相应的多媒体信息都能够得到实时控制。

(6) 信息使用的方便性:用户可以按照自己的需要、兴趣、任务要求、偏爱和认知特点来使用信息,获取图、文、声等信息表现形式。

(7) 信息结构的动态性:用户可以按照自己的目的和认知特征重新组织信息,即增加、删除或修改节点,重新建立链接等。



6.2 音频

6.2.1 数字声音基础

1. 声音信号

声音是通过空气传播的一种连续的波,称为声波。声波在时间和幅度上都是连续的模拟信号,通常称为模拟声音(音频)信号。人们对声音的感觉主要有音量、音调和音色3个指标。

(1) 音量(也称响度):声音的强弱程度,取决于声音波形的幅度,即取决于振幅的大小和强弱。

(2) 音调:人对声音频率的感觉表现为音调的高低,取决于声波的基频。基频越低,给人的感觉越低沉,频率高则声音尖锐。

(3) 音色:由混入基音(基波)的泛音(谐波)所决定,每种声音又都有其固定的频率和不同音强的泛音,从而使得它们具有特殊的音色效果。人们能够分辨具有相同音高的钢琴和小号声音,就是因为它们具有不同的音色。一个声波上的谐波越丰富,音色越好。

对声音信号的分析表明,声音信号由许多频率不同的信号组成,通常称为复合信号,而把单一频率的信号称为分量信号。声音信号的一个重要参数就是带宽(Bandwidth),它用来描述组成声音的信号的频率范围。PC机处理的音频信号主要是人耳能听得到的音频信号(Audio),它的频率范围是20Hz~20kHz。可听声包括:

- ① 话音(也称语音):人的说话声,频率范围通常为300Hz~3400Hz。
- ② 音乐:由乐器演奏形成(规范的符号化声音),其带宽可达到20Hz~20kHz。
- ③ 其他声音:如风声、雨声、鸟叫声、汽车鸣笛声等,它们起着效果声或噪声的作用,其带宽范围也是20Hz~20kHz。

声音信号的两个基本参数是幅度和频率。幅度是指声波的振幅,通常用动态范围表示,一般用分贝(dB)为单位来计量。频率是指声波每秒钟变化的次数,用Hz表示。人们把频率小于20Hz声波信号称为亚音信号(也称次音信号);频率范围为20Hz~20kHz的声波信号称为音频信号;高于20kHz的信号称为超音频信号(也称超声波)。

2. 声音信号的数字化

声音信号是一种模拟信号,计算机要对它进行处理,必须将它转换成为数字声音信号,即用二进制数字的编码形式来表示声音。最基本的声音信号数字化方法是取样-量化法,它分成

如下 3 个步骤:

(1) 采样: 采样是把时间连续的模拟信号转换成时间离散、幅度连续的信号。在某些特定的时刻获取声音信号幅值叫做采样, 由这些特定时刻采样得到的信号称为离散时间信号。一般都是每隔相等的一小段时间采样一次, 其时间间隔称为取样周期, 它的倒数称为采样频率。采样定理是选择采样频率的理论依据, 为了不产生失真, 采样频率不应低于声音信号最高频率的两倍。因此, 语音信号的采样频率一般为 8kHz, 音乐信号的采样频率则应在 40kHz 以上。采样频率越高, 可恢复的声音信号分量越丰富, 其声音的保真度越好。

(2) 量化: 量化处理是把幅度上连续取值(模拟量)的每一个样本转换为离散值(数字量)表示, 因此量化过程有时也称为 A/D 转换(模数转换)。量化后的样本是用二进制数来表示的, 二进制数的位数的多少反映了度量声音波形幅度的精度, 称为量化精度, 也称为量化分辨率。例如, 每个声音样本若用 16bit (2B) 表示, 则声音样本的取值范围是 0~65536, 精度是 1/65536; 若只用 8bit (1B) 表示, 则样本的取值范围是 0~255, 精度是 1/256。量化精度越高, 声音的质量越好, 需要的存储空间也越多; 量化精度越低, 声音的质量越差, 而需要的存储空间少。

(3) 编码: 经过采样和量化处理后的声音信号已经是数字形式了, 但为了便于计算机的存储、处理和传输, 还必须按照一定的要求进行数据压缩和编码, 即选择某一种或者几种方法对它进行数据压缩, 以减少数据量, 再按照某种规定的格式将数据组织成为文件。

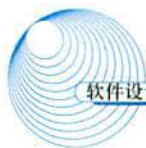
经过数字化处理之后的数字声音的主要参数参见表 6-1。

表 6-1 数字化处理之后的数字声音的主要参数

参数	说明
采样频率	表示每秒钟内采样的次数。采样的 3 个标准频率分别为: 44.1kHz、22.05 kHz 和 11.05 kHz
量化位数	反映度量声音波形幅度的精度。声音信号的量化精度一般为 8 bit、12 bit 或 16 bit
声道数目	单声道一次产生一组声音波形数据, 双声道则一次同时产生两组声音波形数据
数据率	表示每秒钟的数据量, 以 Kb/s 为单位
压缩比	同一段时间间隔内的音频数据压缩前的数据量与压缩后的数据量之比。压缩比通常小于 1

3. 个人计算机中的声音表示方法

个人计算机中的数字声音有两种不同的表示方法。一种称为波形声音(也称为自然声音), 通过对实际声音的波形信号进行数字化(取样和量化)而获得, 它能高保真地表示现实世界中任何客观存在的真实声音。例如, 44.1kHz×16bit 的 CD 质量的声音, 8kHz×8bit 的数字语音等。波形声音的数据量比较大。另一种是“合成声音”, 它使用符号(参数)对声音进行描述, 然后通过合成(Synthesize)的方法生成声音。例如, MIDI 音乐(用符号描述的乐器演奏的音乐



声音)、合成语音(用声母、韵母或清音、基音频率等参数描述的语音)等。符号化的声音表示方法虽然所产生的声音没有自然声那么真实、逼真,但数据量要比波形声音小得多(2~3个数量级),而且能产生自然界中不存在的声音,其编辑处理也比波形声音更加方便一些。

声音信息的计算机处理比数值信息和文本信息的处理要复杂一些,以波形声音的编辑处理为例,常见的基本编辑操作有:声音的分段、拼接、音量调整、降低采样频率、均匀化、时间扩展、声音的各种效果处理(如颤抖、延时、混响、音调变换)等。

个人计算机和多媒体系统中对数字声音的处理是与应用密切相关的,涉及到多方面的声音信息处理技术,大体可以分为:

- (1) 声音的获取、重建与播放。
- (2) 数字声音的编辑处理。
- (3) 数字声音的存储与检索。
- (4) 数字声音的传输。
- (5) 数字语音与文本的相互转换等。

6.2.2 波形声音

波形声音信息是一个用来表示声音振幅的数据序列,它是通过对模拟声音按一定间隔采样获得的幅度值,再经过量化和编码后得到的便于计算机存储和处理的数据格式。声音信号数字化后,其数据传输率(b/s)与信号在计算机中的实时传输有直接关系,而其总数据量又与计算机的存储空间有直接关系。未经压缩的数字音频数据传输率可按下式计算:

$$\text{数据传输率 (b/s)} = \text{采样频率 (Hz)} \times \text{量化位数 (b)} \times \text{声道数}$$

其中数据传输率以每秒比特(b/s)为单位;采样频率以Hz为单位;量化以b为单位。

波形声音经过数字化后所需占用的存储空间可用如下公式计算:

$$\text{声音信号数据量} = \text{数据传输率} \times \text{持续时间} / 8 \text{ (B)}$$

【例 6.1】若语音信号的带宽通常为 300~3400Hz,量化精度为 8b,单声道输出,计算每秒钟及每小时的数据量。

解:据题意,数字化时的采样频率为 8kHz,根据上述公式每秒钟的数据量为:

$$\text{采样频率} \times \text{量化位数} \times \text{声道数} / 8 = 8\text{kHz} \times 8\text{b} \times 1 / 8 = 64\text{Kb/s} = 8\text{KB/s}$$

1 小时数字语音的数据量大约是 28MB。

【例 6.2】若光盘(Compact Disc, CD)盘片上所存储的立体声高保真数字音乐的带宽为 20~20000Hz,采样频率为 44.1kHz,量化精度为 16b,双声道,计算 1 小时的数据量是。

解:每秒钟的数据量为 1411.2Kb/s (176.4KB/s),1 小时的数据量大约是 635MB。

可见,数字波形声音数据量非常大,因此在编码的时候常常要采用压缩的方式来压缩数字

数据以减少存储空间和提高传输效率(降低传输带宽)。根据统计分析结果,语音信号中包含大量的冗余信息,再加上还可以利用人的听觉感知特性,因此,产生了许多能满足实际应用需求的压缩算法。一个好的数据压缩算法通常应能满足下列需求:

- 压缩倍数高,压缩后的数据率低。
- 解码后的信号失真小,质量高。
- 算法简单,执行速度快,延迟时间短。
- 编码器/解码器的成本低。

数字语音的数据压缩方法很多,从原理上可分为3类:

(1) 波形编码:这是一种直接对取样、量化后的波形进行压缩处理的方法。例如脉冲编码调制(PCM)、自适应差分脉冲编码(ADPCM)、子带编码(SBC)等。波形编码的特点是通用性强,不仅适应于数字语音的压缩,而且对所有使用波形表示的数字声音都有效;可获得高质量的语音,但很难达到很高的压缩比。

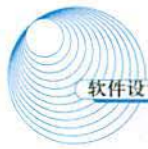
(2) 参数编码(也称为模型编码):这是一种基于声音生成模型的压缩方法(Production Model-Based Compression),从语音波形信号中提取生成的语音参数,使用这些参数通过语音生成模型重构出语音。例如线性预测编码(LPC)、声码器(vocoder)等。它的优点是能达到很高的压缩比,缺点是信号源必须已知,受声音生成模型的限制,质量还不理想。

(3) 混合编码:波形编码虽然可提供高质量的语音,但数据率比较高,很难低于16Kb/s;参数编码的数据率虽然可降低到3Kb/s甚至更低,但它的音质根本不能与波形编码相比。混合编码是上述两种方法的结合,它既能达到高的压缩比,又能保证一定的质量,但算法相对复杂一些,例如码激励线性预测(CELP)、混合激励线性预测(MELP)等。

基于波形的压缩编码方法虽然数据率比较高,但能保证高质量地重建语音,且算法简单、易实现,能较好地保持原有声音的特点,便于编辑处理,在多媒体计算机和多媒体文档中有广泛的应用。

数字语音压缩编码有多种国际标准,如G.711、G.721、G.726、G.727、G.722、G.728、G.729A、G.723.1、IS96(CDMA)等。

数字语音由于频带比较窄,又可以通过语音生成模型进行比较好的模拟,因此经过压缩编码后码率比较低,存储与传输问题不大。而对于有高保真度要求的全频带声音,由于其带宽达到20kHz以上,又有双声道甚至多声道的要求,数据量相当客观,对压缩编码的要求更高。全频带声音的压缩编码方法与数字语音的处理方法不同,它不但依据波形本身的相关性,而且还利用人的听觉系统特性来达到压缩声音的目的,这种压缩编码称为感知声音编码(Perceptual Audio Coding)。在国际标准MPEG中,先后为视频图像伴音的数字宽带声音制定了MPEG-1 Audio、MPEG-2 Audio、MPEG-2AAC、MPEG-4 Audio等多种数据压缩编码的标准。MPEG



处理的是 10~20000Hz 频率范围的声音信号,数据压缩的主要依据是人耳的听觉特性,特别是人耳存在着随声音频率变化的听觉域,以及人耳的听觉掩蔽特性。

6.2.3 声音合成

个人计算机和多媒体系统中的声音,除了数字波形声音之外,还有一类是使用符号表示的,由计算机合成的声音,包括语音合成和音乐合成。

1. 语音合成

语音合成目前主要指从文本到语音的合成,也称为文语转换。采用文语转换的方法输出语音,应预先建立语音参数数据库、发音规则库等。需要输出语音时,系统按需求先合成语音单元,再按语音学规则或语言学规则,连接成自然的语流。文语转换的参数数据库不随发音时间增长而加大,但规则库却随语音质量的要求而增大。语音合成有多方面的应用,例如,海量查询与声讯服务、航班动态查询、电话报税、有声 E-mail 等,这些业务都需要以准确、清晰的语音通过电话进行操作提示并提供查询结果。一般来说,对合成的语音要求能够做到:可听懂、自然、延迟时间短、速度可控制等。

文语转换原理上一般分成两步,第一步先将文字序列转换成音韵序列,第二步再由语音合成器生成语音波形(见图 6-1)。其中第一步涉及语言学处理,例如分词、字音转换等,以及一整套有效的韵律控制规则;第二步需要使用语音合成技术,能按要求实时合成出高质量的语音流。



图 6-1 文语转换过程示意

从合成采用的技术来说,语音合成可分为发音参数合成、声道模型参数合成和波形编辑合成。

(1) 发音参数合成:这种方法对人的发音过程进行直接模拟,它定义了唇、舌、声带的相关参数,如开口度、舌高度、舌位置、声带张力等。由这些发音参数估计声道截面积函数,进而计算声波。但由于人发音生理过程复杂,理论计算与物理模拟之间的差异,合成语音的质量目前还不理想。

(2) 声道模型参数合成:这种方法基于声道截面积函数或声道谐振特性合成语音,如共振

峰合成器、LPC 合成器。这类合成器的比特率低,音质适中。为改善音质,发展了混合编码技术,主要手段是改善激励,如码本激励、多脉冲激励、长时预测规则码激励等。这样比特率有所增大,同时音质得到提高。

(3) 波形编辑合成:波形编辑合成技术是直接把语音波形数据库中的波形相互拼接在一起,输出连续语流。这种语音合成技术用原始语音波形代替参数,而且这些语音波形取自自然语言的词或句子,它隐含了声调、重音、发音速度的影响,合成的语言清晰自然。其质量普遍高于参数合成。

2. 音乐合成

音乐是用乐谱进行描述由乐器演奏而成的。乐谱的基本组成单元是音符(notes),最基本的音符有 7 个,所有不同音调的音符少于 128 个。

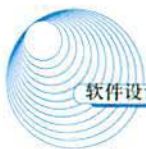
音符代表的是音乐,音乐与噪声的区别主要在于它们是否有周期性。乐音的波形随着时间作周期性变化,噪声则不然。音乐的要素有音调、音色、响度和持续时间。音调指声波的基频,基频低,声音低沉;基频高,声音高昂。不同乐器有不同的音色,音色是由声音的频谱决定的。响度即声音的强度。一首乐曲中每一个乐音的持续时间是变化的,从而形成旋律。音乐可以使用电子学原理合成出来(生成相应的波形),各种乐器的音色也可以进行模拟。电子乐器由两部分组成:

(1) 演奏控制器:演奏控制器是一种输入和记录实时乐曲演奏信息的设备。它的作用是像传统乐器那样用于演奏,驱动音源发声,同时它也是电脑音乐系统的输入设备。其类型有:键盘、气息(呼吸)控制器、弦乐演奏器等。

(2) 音源(也称为音乐合成器):音源是具体产生声音波形的部分,即电子乐器的发声部分。它通过电子线路把演奏控制器送来的乐音合成出来。最常用的音源有两类:

① 数字调频合成器(FM):FM 是使高频振荡波的频率按调制信号规律变化的一种调制方式。在音乐合成器中,数字载波波形和调制波形有很多种,采用的调制波频率和调制指数,就可以方便地合成具有不同频谱分布的波形,再现某些乐器的音色。我们可以这种方法得到具有独特效果的电子模拟声,创造出丰富多彩的并且是真实乐器所不具备的音色。

② PCM 波形合成器(波表合成法):使用 FM 合成法来产生逼真的乐音是不甚理想的,有些音乐几乎不能产生,因此就自然地转向采用 PCM 波形合成法,即波表合成法。这种方法把真实乐器发出的声音以数字的形式记录下来,将它们放在一个波形表中,合成音乐时以查表匹配方式获取真实乐器波形。这种以真实声音波形为基础的音源具有音色真实,丰满的质量,合成的音乐基本上能达到以假乱真的效果。但是,由于采样波形不能代表所有的真实的演奏状态,而且音乐的音色在某种程度上还取决于演奏者的演奏技术,因而还是达不到演奏者演奏音乐的临场感觉效果。



6.2.4 MIDI

MIDI (Musical Instrument Digital Interface) 是乐器数字接口的缩写,泛指数字音乐的国际标准。MIDI 标准规定了电子乐器与计算机之间连接的电缆硬件以及电子乐器之间、乐器与计算机之间传送数据的通信协议的规范;规定了音乐的数字表示,包含音符、定时、乐器指派等规范。由于 MIDI 标准定义了计算机音乐程序、合成器及其他电子设备交换信息和电子信号的方式,所以可以解决不同电子乐器之间不兼容的问题。符合 MIDI 规范的设备称为 MIDI 设备,通过 MIDI 接口,不同 MIDI 设备之间可进行信息交换。带有 MIDI 接口以及专用的 MIDI 电缆计算机、合成器和其他 MIDI 设备连接在一起,即可构成计算机音乐系统。数据由 MIDI 设备的键盘产生,可通过声音合成器还原为声音。通过计算机可以控制乐器的输出,并能接收、存储和处理经过编码的音乐数据。

MIDI 数据不是单个采样点的编码(波形编码),而是乐谱的数字描述,称为 MIDI 消息。乐谱由音符序列、定时、音色、音量等组成,每个消息对应一个音乐事件(如键压下、键释放等),一组 MIDI 消息送到 MIDI 音源时,音源即合成出相应的音乐。

MIDI 文件是计算机中用于存储和交换 MIDI 消息的一种数据文件,它由一系列的 MIDI 消息组成。它不仅包含着与音乐演奏有关的消息,如音符、速度、表情等信息,而且还包含了大量与音响有关的控制信息,如声像、效果、均衡等。MIDI 文件中包含了多达 16 个通道的演奏定义,包括每个通道的演奏音符信息,如键、音长、音量和力度(击键时,键达到最低位置的速度)等。标准 MIDI 文件格式采用的文件扩展名为.mid,它是音序软件的文件交换标准,也是商业音乐作品发行的标准。

音序器(Sequencer)又称声音序列发生器,有硬件产品,也有软件产品(也称音乐软件或作曲软件)。它具有丰富的编辑和存储功能。首先,音序器可将演奏者实时演奏的音符(Note)、节奏信息以及各种表情控制信息,如速度、触键力度、颤音以及音色变化等以数字方式,在电脑时钟的基础上,按时间或节拍顺序记录下来,然后对记录下来的信息进行修改编辑,经过编辑修改的演奏信息在任意时刻都可以发送给音源,音源即可自动演奏播放。关键是要实现“分轨录音”,而这些属于不同声部的演奏信息可被音序器记录在不同的 MIDI 通道中,通过音源,音序器可将所有 MIDI 通道中的演奏信息同时自动播放演奏。这样,一个人就可完成相当于一个乐队的多声部演奏和录音任务。使用以音序器软件为核心的计算机个人音乐系统,彻底改变了传统的音乐制作方式和概念,原来需要由多人才能完成的工作现在只需一个人即可,记录音乐的方式也由原来的乐谱变成了 MIDI 文件,音乐作品由修改困难变为可进行任意的编辑修改。

MIDI 音乐与高保真的波形声音相比,虽然在音质方面还有一些差距,也无法合成出所有各种不同的声音(如语音),且生成的音乐质量与使用的音源硬件有关但它的数量级极少(比 CD-DA 少 3 个数量级,比 MP3 少 2 个数量级),又易于编辑修改,还可以与波形声音同时播放。

6.2.5 声音文件格式

数字声音在计算机中存储和处理时,其数据必须以文件的形式进行组织,所选用的文件格式必须得到操作系统和应用软件的支持。如同文本文件一样,在互联网上和各种不同计算机以及应用软件中使用的声音文件格式也互不相同。下面通过声音文件的特征后缀名来逐一介绍。

(1) Wave 文件 (.WAV): Microsoft 公司的音频文件格式,它来源于对声音模拟波形的采样。用不同的采样频率对声音的模拟波形进行采样可以得到一系列离散的采样点,以不同的量化位数(8位或16位)把这些采样点的值转换成二进制数,然后存入磁盘,这就产生了声音的 WAV 文件,即波形文件。利用该格式记录的声音文件能够 and 原声基本一致,质量非常高,但文件数据量大。

(2) Module 文件 (.MOD): 该格式的文件里存放乐谱和乐曲使用的各种音色样本,具有回放效果明确、音色种类无限等优点。

(3) MPEG 文件 (.MP3): 现在最流行的声音文件格式,因其压缩率大,在网络可视电话通信方面应用广泛,但和 CD 唱片相比,音质不能令人非常满意。

(4) RealAudio 文件 (.RA): 这种格式具有强大的压缩量和极小的失真,它也是为了解决网络传输带宽资源而设计的,因此主要目标是压缩比和容错性,其次才是音质。

(5) MIDI 文件 (.MID/.RMI): 它是目前较成熟的音乐格式,实际上已经成为一种产业标准,其科学性、兼容性、复杂程度等各方面当然远远超过本文前面介绍的所有标准(除交响乐 CD、Unplug CD 外,其他 CD 往往都是利用 MIDI 制作出来的),General MIDI 就是最常见的通行标准。作为音乐工业的数据通信标准,MIDI 能指挥各音乐设备的运转,而且具有统一的标准格式,能够模仿原始乐器的各种演奏技巧甚至无法演奏的效果,而且文件的长度非常小。.RMI 可以包括图片标记和文本。

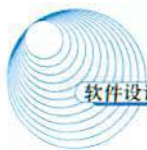
(6) Voice 文件 (.VOC): Creative 公司波形音频文件格式,也是声霸卡(sound blaster)使用的音频文件格式。每个 VOC 文件由文件头块(header block)和音频数据块(data block)组成。文件头包含一个标识版本号和一个指向数据块起始的指针。数据块分成各种类型的子块。如声音数据静音标识 ASCII 码文件重复的结果重复以及终止标志、扩展块等。

(7) Sound 文件 (.SND): Sound 文件是 NeXT Computer 公司推出数字声音文件格式,支持压缩。

(8) Audio 文件 (.AU): Audio 文件是 Sun Microsystems 公司推出的一种经过压缩的数字声音文件格式,是互联网上常用的声音文件格式。

(9) AIFF 文件 (.AIF) 文件: Apple 计算机的音频文件格式。Windows 的 Convert 工具可以把 AIF 格式的文件换成 Microsoft 的 WAV 格式的文件。

(10) CMF 文件 (.CMF): Creative 公司的专用音乐格式,与 MIDI 差不多,音色、效果上



有些特色,专用于 FM 声卡,但其兼容性也很差。

6.3 图形和图像

6.3.1 彩色与图像基础

1. 彩色的基本概念

彩色是创建图像的基础,在计算机上使用彩色没有什么特殊之处,只不过是它有一套特定的记录和处理彩色的技术。因此,要理解图像处理软件中所出现的各种有关彩色的术语,首先要具备基本的彩色理论知识。

彩色是通过光被人们感知的,物体由于内部物质的不同,受光线照射后,产生光的分解现象,一部分光线被吸收,其余的被反射或投射出来,成为人们所见的物体的彩色。所以,彩色和光有密切关系,同时还与被光照射的物体有关,并与观察者有关。

彩色光作用于人眼,使之产生彩色视觉。为了能确切地表示某一彩色光的度量,可以用亮度、色调和色饱和度三个物理量来描述,并称之为色彩三要素。

(1) 亮度:亮度是描述光作用于人眼时引起的明暗程度感觉,是指彩色明暗深浅程度。一般说来,对于发光物体,彩色光辐射的功率越大,亮度越高;反之,亮度越低;对于不发光的物体,其亮度取决于吸收或者反射光功率的大小。

(2) 色调:色调是指颜色的类别,如红色、绿色、蓝色等不同颜色就是指色调。由光谱分析可知,不同波长的光呈现不同的颜色,人眼看到一种或多种波长的光时所产生的彩色感觉,反映出颜色的类别。某一物体的色调取决它本身辐射的光谱成分或在光的照射下所反射的光谱成分对人眼刺激的视觉反应。

(3) 色饱和度:色饱和度是指某一颜色的深浅程度(或浓度)。对于同一种色调的颜色,其饱和度越高,则颜色越深,如深红、深绿、深蓝等;其饱和度越低,则颜色越淡,如淡红、淡绿、淡黄等;高饱和度的深色光可掺入白色光被冲淡,降为低饱和度的淡色光。因此,色饱和度可认为是某色调的纯色掺入白色光的比例。例如,一束高饱和度的蓝色光投射到屏幕上会被看成深蓝色光,若再将一束白色光也投射到屏幕上并与深蓝色重叠,则深蓝色变成淡蓝色,而且投射的白色光越强,颜色越淡,即饱和度越低。相反,由于在彩色电视的屏幕上的亮度过高,则色饱和度降低,颜色被冲淡,这时可以降低亮度(白光)而使色饱和度增大,颜色加深。

2. 三基色原理

从理论上讲,任何一种颜色都可以用 3 种基本颜色按不同比例混合得到。自然界常见的各

种颜色光,都可由红(Red)、绿(Green)、蓝(Blue)3种颜色光按不同比例相配而成。同样,绝大多数颜色光也可以分解成红、绿、蓝3种颜色光,这就是色度学中最基本的三基色原理。当然,三基色的选择不是唯一的,可以选择其他3种颜色为三基色。但是,3种颜色必须是相互独立的,即任何一种颜色都不能由其他两种颜色合成。由于人眼对红、绿、蓝3种颜色光最敏感,因此由这3种颜色相配所得的彩色范围也最广,所以一般都选这3种颜色作为基色。把3种基色光按不同比例相加称之为相加混色,由红、绿、蓝三基色进行相加混色的情况如下:

红色+绿色=黄色

红色+蓝色=品红

绿色+蓝色=青色

红色+绿色+蓝色=白色

红色+青色=绿色+品红=蓝色+黄色=白色

凡是两种色光混合而成白光,则这两种色光互为补色。

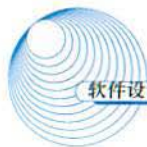
3. 彩色空间

彩色空间指彩色图像所使用的颜色描述方法,也称为彩色模型。在PC机和多媒体系统中,表示图形和图像的颜色常常涉及不同的彩色空间,如RGB彩色空间、CMY彩色空间、YUV彩色空间等。不同的彩色空间对应不同的应用场合,各有其特点。因此,数字图像的生成、存储、处理及显示时对应不同的彩色空间,从理论上讲,任何一种颜色都可以在上述彩色空间中精确地进行描述。

(1) RGB彩色空间:计算机中的彩色图像一般都用R、G、B分量表示,彩色显示器通过发射出3种不同强度的电子束,使屏幕内侧覆盖的红、绿、蓝荧光材料发光而产生色彩。这种彩色的表示方法称为RGB彩色空间表示法。因为彩色显示器的输入需要R、G、B彩色分量,通过3个分量的不同比例,在显示屏幕可合成任意所需要的颜色。所以无论多媒体系统中间过程采用什么形式的彩色空间表示,最后的输出一定要转换成RGB彩色空间表示。

(2) CMY彩色空间:RGB彩色空间中,不同的颜色的光是通过相加混合实现的,而彩色打印的纸张是不能发射光线的,因而彩色打印机就不能采用RGB颜色来打印,它只能使用能够吸收特定的光波而反射其他光波的油墨或颜料来实现。用油墨或颜料进行混合得到的彩色称为相减混色。之所以称为相减混色,是因为减少(吸收)了人眼识别颜色所需要的反射光。根据三基色原理,油墨或颜料的三基色是青(Cyan)、品红(Magenta)和黄(Yellow)。可以用这3种颜色的油墨或颜料按不同比例混合成任何一种由油墨或颜料表现的颜色,这种彩色表示方法称为CMY彩色空间。

(3) YUV彩色空间:在现代彩色电视系统中,通常采用三管彩色摄像机或彩色CCD摄像机,它把摄得的彩色图像信号,经过分色、放大和校正得到R、G、B三基色,再经过矩阵变



换得到的亮度信号 Y 、色差信号 U ($R-Y$) 和 V ($B-Y$)，最后发送端将这 3 个信号分别进行编码，用同一信道发送出去。这就是通常常用的 YUV 彩色空间。电视图像一般都是采用 Y 、 U 、 V 分量表示，即一个亮度分量 (Y) 和两个色差 (U 和 V) 分量表示。由于亮度和色度是分离的，解决了彩色和黑白显示系统的兼容问题，如果只有 Y 分量而没有 U 、 V 分量，那么所表示的图像是黑白灰度图像。

6.3.2 计算机中的图形数据表示

在计算机中的图形数据有两种常用的表示形式，一种称为几何图形或矢量图形，简称图形；另一种称为点阵图像或位图图像。

1. 矢量图形

矢量图形是用一系列计算机指令来描述和记录的一幅图的内容，即通过指令描述构成一幅图的所有直线、曲线、圆、圆弧、矩形等图元的位置、维数和形状，也可以用更为复杂的形式表示图像中的曲面、光照、材质等效果。矢量图法实质上是用数学的方式（算法和特征）来描述一幅图形图像，在处理图形图像时根据图元对应的数学表达式进行编辑和处理。在屏幕上显示一幅图形图像时，首先要解释这些指令，然后将描述图形图像的指令转换成屏幕上显示的形状和颜色。编辑矢量图的软件通常称为绘图软件，如适于绘制机械图、电路图的 AutoCAD 软件等。这种软件可以产生和操作矢量图的各个成分，并对矢量图形进行移动、缩放、移动、叠加、旋转和扭曲等变换。编辑图形时将指令转变成屏幕上所显示的形状和颜色，显示时也往往能看到绘图的过程。由于所有的矢量图形部分都可以用数学的方法加以描述，从而使得计算机可以对其进行任意的放大、缩小、旋转、变形、扭曲、移动、叠加等变换，而不会破坏图像的画面。但是，用矢量图形格式表示复杂图像（如人物、风景照片）并要求很高时，将需要花费大量的时间进行变换、着色、处理光照效果等。因此，矢量图形主要用于表示线框型的图画、工程制图、美术字等。多数 CAD 和 3D 造型软件使用矢量图形作为基本的图形存储格式。

2. 位图图像

位图图像是指用像素点来描述的图。图像一般是用摄像机或扫描仪等输入设备捕捉实际场景画面，离散化为空间、亮度、颜色（灰度）的序列值，即把一幅彩色图或灰度图分成许许多多的像素（点），每个像素用若干二进制位来指定该像素的颜色、亮度和属性。位图图像在计算机内存中由一组二进制位组成，这些位定义图像中每个像素点的颜色和亮度。屏幕上一个点也称为一个像素，显示一幅图像时，屏幕上的一个像素也就对应于图像中的某一个点。根据组成图像的像素密度和表示颜色、亮度级别的数目，又可将图像分为二值图（黑白图）和彩色图

两大类,彩色图还可以分为真彩色图、伪彩色图等。图像适合于表现比较细腻,层次较多,色彩较丰富,包含大量细节的图像,并可直接、快速地在屏幕上显示出来。但占用存储空间较大,一般需要进行数据压缩。

6.3.3 图像的获取

将现实世界的景物或物理介质上的图文输入计算机的过程称为图像的获取(Capturing)。在多媒体应用中的基本图像可通过不同的方式获得,一般来说,可以直接利用数字图像库的图像;可以利用绘图软件创建图像;可以利用数字转换设备采集图像。

1. 利用数字图像库

在 CD-ROM 光盘上和互联网上的图像库中,图像的内容较丰富,图像尺寸和图像深度可选的范围也较广。而且图像的质量完全可以满足一般用户的要求,但图像的内容也许不具备用户的创意设计。用户可根据需要选择已有的图像,或再做进一步的编辑和处理。

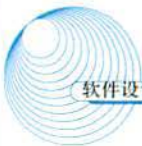
2. 利用绘图软件创建图像

目前 Windows 环境的大部分图像编辑软件都具有一定的绘图功能。这些软件大多具有较强的功能和很好的图形用户接口,还可以利用鼠标、画笔及画板来绘制各种图形,并进行彩色、纹理、图案等的填充和加工处理。对于一些小型的图形、图标、按钮等,直接制作很方便,但这不足以描述自然景物和人像。也有一些较专业的绘画软件,通过数字化画板和画笔在屏幕上绘画。这种软件要求绘画者具有一定的美术知识及创意基础。

3. 利用数字转换设备采集图像

数字转换设备可以把采集到的图像转换成计算机能够记录和处理数字图像数据。例如,对印刷品、照片或照相底片等进行扫描,用数码相机或数码摄像机对选定的景物进行拍摄等。从现实世界中获取数字图像所使用的设备通称为图像获取设备。一幅彩色图像可以看作二维连续函数 $f(x, y)$, 其彩色 f 是坐标 (x, y) 的函数,从二维连续函数到离散的矩阵表示,同样包含采样、量化和编码的数字化过程。数字转换设备获取图像的过程实质上是信号扫描和数字化的过程,它的处理步骤大体分为 3 步:

(1) 采样: 在 x, y 坐标上对图像进行采样(也称为扫描),类似于声音信号在时间轴上的采样要确定采样频率一样,在图像信号坐标轴上的采样也要确定一个采样间隔,这个间隔即为图像分辨率。有了采样间隔,就可以逐行对原始图像进行扫描。首先设 y 坐标不变,对 x 轴按采样间隔得到一行离散的像素点 x_n 及相应的像素值。使 y 坐标也按采样间隔由小到大的变化,就可以得到一个离散的像素矩阵 $[x_n, y_n]$, 每个像素点有一个对应的色彩值。简单地说,将一幅



画面划分为 $M \times N$ 个网格, 每个网格称为一个取样点, 用其亮度值来表示。这样, 一幅连续的图像就转换为以取样点值组成的一个阵列(矩阵)。

(2) 量化: 将扫描得到的离散的像素点对应的连续色彩值进行 A/D 转换(量化), 量化的等级参数即为图像深度。这样, 像素矩阵中的每个点 (x_n, y_n) 都有对应的离散像素值 f_n 。

(3) 编码: 把离散的像素矩阵按一定方式编成二进制码组。最后, 把得到的图像数据按某种图像格式记录在图像文件中。

6.3.4 图像的属性

描述一幅图像需要使用图像的属性。图像的属性包含分辨率、像素深度、真/伪彩色、图像的表现法和种类等。

1. 分辨率

我们经常遇到的分辨率有两种, 即显示分辨率和图像分辨率。

(1) 显示分辨率: 是指显示屏上能够显示出的像素数目。例如, 显示分辨率为 1024×768 表示显示屏分成 768 行(垂直分辨率), 每行(水平分辨率)显示 1024 个像素, 整个显示屏就含有 796432 个显像点。屏幕能够显示的像素越多, 说明显示设备的分辨率越高, 显示的图像质量越高。

(2) 图像分辨率: 是指组成一幅图像的像素密度, 也是用水平和垂直的像素表示, 即用每英寸多少点(dpi)表示数字化图像的大小。例如, 用 200dpi 来扫描一幅 2×2.5 英寸的彩色照片, 那么得到一幅 400×500 个像素点的图像。它实质上是图像一数字化的采样间隔, 由它确定组成一幅图像的像素数目。对同样大小的一幅图, 如果组成该图的图像像素数目越多, 则说明图像的分辨率越高, 图像看起来就越逼真。相反, 图像显得越粗糙。因此, 不同的分辨率会造成不同的图像清晰度。

图像分辨率与显示分辨率是两个不同的概念。图像分辨率确定的是组成一幅图像像素数目, 而显示分辨率确定的是显示图像的区域大小。它们之间的关系是:

① 图像分辨率大于显示分辨率时, 在屏幕上只能显示部分图像。例如, 当图像分辨率为 800×600 , 屏幕分辨率为 640×480 时, 屏幕上只能显示一幅图像的 64% 左右。

② 图像分辨率小于屏幕分辨率时, 图像只占屏幕的一部分。例如, 当图像分辨率为 320×240 , 屏幕分辨率为 640×480 时, 图像只占屏幕的 1/4。

2. 图像深度

图像深度是指存储每个像素所用的位数, 它也是用来度量图像的色彩分辨率的。像素深度确定彩色图像的每个像素可能有的颜色数, 或者确定灰度图像的每个像素可能有的灰度级数。

它决定了彩色图像中可出现的最多颜色数，或灰度图像中的最大灰度等级。如一幅图像的图像深度为 b 位，则该图像的最多颜色数或灰度级为 2^b 种。显然，表示一个像素颜色的位数越多，它能表达的颜色数或灰度级就越多。例如，只有 1 个分量的单色图像，若每个像素有 8 位，则最大灰度数目为 $2^8=256$ ；一幅彩色图像的每个像素用 R、G、B 3 个分量表示，若 3 个分量的像素位数分别为 4、4、2，则最大颜色数目为 $2^{4+4+2}=2^{10}=1024$ ，就是说像素的深度为 10 位，每个像素可以是 2^{10} 种颜色中的一种。表示一个像素的位数越多，它能表达的颜色数目就越多，它的深度就越深。

3. 真彩色和伪彩色

真彩色 (True Color) 是指组成一幅彩色图像的每个像素值中，有 R、G、B 3 个基色分量，每个基色分量直接决定显示设备的基色强度，这样产生的彩色称为真彩色。例如，用 RGB8:8:8 方式表示一幅彩色图像，也就是 R、G、B 分量都用 8 位来表示，可生成的颜色数就是 2^{24} 种，每个像素的颜色就是由其中的数值直接决定的。这样得到色彩可以反映原图像的真实色彩，一般认为是真彩色。通常，在一些场合把 RGB8:8:8 方式表示的彩色图像称为真彩色图像或全彩色图像。

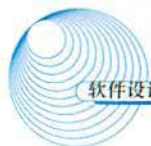
为了减少彩色图形的存储空间，在生成图像时，对图像中不同色彩进行采样，产生包含各种颜色的颜色表，即彩色查找表。图像中每个像素的颜色不是由 3 个基色分量的数值直接表达，而是把像素值作为地址索引在彩色查找表中查找这个像素实际的 R、G、B 分量，我们将图像的这种颜色表达方式称为伪彩色。需要说明的是，对于这种伪彩色图像的数据除了保存代表像素颜色的索引数据外，还要保存一个色彩查找表（调色板）。彩色查找表可以是一个预先定义的表，也可以是对图像进行优化后产生的色彩表。常用的 256 色的彩色图像使用了 8 位的索引，即每个像素占用一个字节。

6.3.5 图形图像转换

图形和图像之间在一定的条件下可以转换，如采用光栅化（点阵化）技术可以将图形转换成图像；采用图形跟踪技术可以将图像转换成图形。一般可以通过硬件（输入/输出设备）或软件实现图形和图像之间转换。

1. 图形和图像的硬件转换

一张工程图纸，一般认为它是图形，其实在它被输入计算机以前我们还不能称它为图形或图像，当我们将它用扫描仪输入到 Photoshop，它就变成图像信息（点位图）；当我们用数字化仪来将它输入到 AutoCAD 后，它就变成图形信息（矢量图）。也就是说同一个对象既可作为图形处理也可以作为图像处理。那么到底哪种过程更有效，要看被处理的对象性质和要达到的



处理结果。当然,我们可以先将它用扫描仪扫进计算机,变成图像信息,再用一定的软件(如 Corel-Trace, Photoshop 的轮廓跟踪)人工或自动地勾勒出它的轮廓,这个过程称为向量化,也就是图像转换为图形的过程,这个过程必然会丢失许多细节,所以通常是适用于工程绘图领域。

如果我们用 AutoCAD 软件作好了一张图,较合理的方法是用绘图仪将它输出,但是也可以用打印机将它输出,这时计算机必须先将图形转换为打印机的扫描线,这个过程称为光栅化,也就是图形转换为图像的过程。如果用 Photoshop 软件作好了一张图,较合理的方法是用打印机将它输出,这样可以得到较多的层次和细节,如果一定要用绘图仪输出,就必然会丢失许多图像的细节。

2. 图形和图像的软件转换

图形和图像都是以文件的形式存放在计算机存储器中,可以通过应用软件的实现文件格式之间的转换,达到图形和图像之间的转换。随着图形和图像处理技术的发展,出现了很多较好格式转换软件,如 CorelDraw 软件,它几乎提供所有文件格式之间的转化。同时也出现了一些较好文件格式,如 Adobe 公司 EPS 格式文件,它是一种兼并图形图像各自优点的文件格式。转换并不表示可以任意互换,实际上许多转换是不可逆的,转换的次数越多,丢失的信息就越多,特别是图形和图像之间的转化。例如,当我们将一个 BMP 格式文件转化为 GIF、TIFF 等格式时问题还不大,但如果将它转化为 DIF 等格式时就丢失了许多细节,甚至像一个矩形中间的填充色块,都用几条线表示。又如,将一个 AutoCAD 的 DWG 文件转化为 DIF 时问题还不大,但如果将它转化为 BMP、GIF、TIFF 时,就要必须考虑分辨率和彩色数。这两个参数决定了最终图像文件的大小和它的使用价值。总之,从本质上讲各种不同的文件格式,是在对不同性质的处理对象或同一对象的不同处理侧面采用一种最为科学、合理和方便的描述方法。应该根据处理对象的特点选择或转化为相应的文件格式,以及选择相应的输入/输出设备。

6.3.6 图像的压缩编码

扫描生成一幅图像时,实际上就是按一定的图像分辨率和一定的图像深度对模拟图片或照片进行采样,从而生成一幅数字化的图像。图像分辨率越高,图像深度越深,则数字化后的图像效果越逼真,图像数据量越大。如果按照像素点及其深度映射的图像数据大小采样,可用下面的公式估算数据量:

$$\text{图像数据量} = \text{图像的总像素} \times \text{图像深度} / 8 \text{ (B)}$$

其中图像的总像素为图像的水平方向像素乘以垂直方向像素数。

例如,一幅 640×480 的 256 色图像,其文件大小约为:

$$640 \times 480 \times 8 / 8 \approx 300 \text{KB}$$

可见,数字图像的数据量也很大,需要很大的存储空间存储数据图像。更重要的是,在现

代通信中，特别是互联网上开展的各种应用中，图像传输速度是一项很重要的指标。以使用拨号接入互联网的家庭用户为例，假设数据传输速度为 56Kb/s，则理想情况下，传输一幅分辨率为 640×480 的 6.5 万色的未经压缩的图像大约需要 1~2min。因此，采用压缩编码技术，减少图像的数据量，是提高网络传输速度的重要手段。

由于数字图像中的数据相关性很强，或者说，数据的冗余度很大，因此对数字图像进行大幅度的数据压缩是完全可能的。而且，人眼的视觉有一定的局限性，即使压缩前后的图像有一定失真，只要限制在人眼允许的误差范围之内，也是允许的。

数据压缩可分成两类，一类是无损压缩，另一类是有损压缩。

1. 无损压缩编码

无损压缩利用数据的统计冗余进行压缩，可以保证在数据压缩和还原过程中，图像信息没有损耗或失真，图像还原（解压缩）时可完全恢复（即重建后的图像与原始图像完全相同）。例如，在多媒体应用中常用行程长度编码（RLE）、增量调制编码（DM）、霍夫曼（Huffman）编码等。

1) 行程长度编码（Run-Length Encoding, RLE）

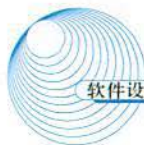
某些图像往往有许多颜色相同的图块。在这些图块中，许多连续的扫描行都具有同一种颜色，或者同一扫描行上有许多连续的像素都具有相同的颜色值。在这些情况下就可以不需要存储每一个像素的颜色值，而仅仅存储一个像素值以及具有相同颜色的像素数目。这种编码称为行程编码。具有同一颜色的连续像素的数目称为行程长度。其压缩率的大小取决于图像本身。如果图像中具有相同颜色的横向色块越大，这样的图像块数目越多，压缩比就越大；反之就越小。

2) 增量调制编码（Delta Modulation Encoding, DM）

自然图像往往有在比较大的范围内，图像的颜色虽不完全一致，但变化不大的特点。因此，在这些区域中，相邻像素的像素值相差很小，具有很大的相关性。在一幅图像中，除了轮廓特别明显的地方以外，大部分区域都具有这种特点。增量调制编码就利用图像相邻像素值的相关性来压缩每个像素值的位数，以达到减少存储容量的目的。增量调制编码压缩图像时，不存储扫描行上每个像素的实际值，仅存储每一行上第一个像素的实际值。其后，依次存储每一个像素的像素值与前一个像素值之差，即增量值。

3) 霍夫曼（Huffman）编码

大多数图像常常包含单色的大面积图块，而且某些颜色比其他颜色出现得更频繁，因此可以采用霍夫曼编码方式。霍夫曼编码的基本方法是先对图像数据扫描一遍，计算出各种像素出现的概率，按概率的大小指定不同长度的唯一码字，由此得到一张该图像的霍夫曼码表。编码后的图像数据记录的是每个像素的码字，而码字与实际像素值的对应关系记录在码表中。码表



是附在图像文件中的。实际应用中,霍夫曼编码常与其他编码方法结合使用,以获得更大的压缩比。

2. 有损压缩编码

有损压缩方法利用人眼视觉对图像中的某些频率成分不敏感的特性,采用一些高效的有限失真数据压缩算法,允许压缩过程中损失一定的信息。采用有损压缩后的数据进行图像重建时,重建后的图像与原始图像虽有一定的误差,并不影响人们对图像含义的正确理解。但换来了较大的压缩比,大幅度减少了图像信息中的冗余信息。一般为了得到较高的数据压缩比,数字图像的压缩一般都采用有损压缩。经常使用的有损压缩方法有预测编码、变换编码、矢量编码和基于模型的编码。实际使用时常常是多种压缩方法的结合。

总之,图像压缩的方法很多,不同方法有不同的适用场合和范围。

6.3.7 多媒体数据压缩编码的国际标准

计算机中使用的图像压缩编码方法有多种国际标准和工业标准。目前使用得当广泛的编码及压缩标准有 JPEG、MPEG 和 H.261。

1. JPEG (Joint Photographic Experts Group)

是一个由 ISO 和 IEC 两个组织机构联合组成的一个专家组,负责制定静态和数字图像数据压缩编码标准,这个专家组地区性的算法称为 JPEG 算法,并且成为国际上通用的标准,因此又称为 JPEG 标准。JPEG 是一个适用范围很广的静态图像数据压缩标准,既可用于灰度图像又可用于彩色图像。JPEG 专家组开发了两种基本的压缩算法,一种是以离散余弦变换(Discrete Cosine Transform, DCT)为基础的有损压缩算法,另一种是采用以预测技术为基础的无损压缩算法。使用有损压缩算法时,在压缩比为 25:1 的情况下,压缩后还原得到的图像与原始图像相比较,人们难于察觉它们之间的区别,因此得到了广泛的应用。例如,在 V-CD 和 DVD-Video 电视图像压缩技术中,就使用 JPEG 的有损压缩算法来取消同方向上的冗余数据。为了在保证图像质量的前提下进一步提高压缩比, JPEG 专家组制定了 ISO/IEC 15444 JPEG2000 (简称 JP2000) 标准,这个标准中采用了小波变换(Wavelet)算法。

2. MPEG (Moving Pictures Experts Group)

是一个动态图像压缩标准,由 ISO 和 IEC 两个组织机构联合组成的一个活动图像专家组,于 1990 年形成了一个标准草案,将 MPEG 标准分成两个阶段:第一阶段(MPEG-1)是针对传输率为 1Mb/s 到 1.5Mb/s 的普通电视质量的视频信号的压缩;第二阶段(MPEG-2)目标则是对每秒 30 帧的 720×572 分辨率的视频信号进行压缩;在扩展模式下, MPEG-2 可以对分辨率达

1440×1152 高清晰度电视 (HDTV) 的信号进行压缩。MPEG 标准分成 MPEG 视频、MPEG 音频和视频音频同步 3 个部分。1999 年发布了 MPEG-4 多媒体应用标准、目前推出了 MPEG-7 多媒体内容描述接口标准等。每个新标准的产生都极大地推动了数字视频的发展和更广泛的应用。

3. H.261

H.261 视频通信编码标准由国际电话电报咨询委员会 (Consultative Committee on International Telephone and Telegraph, CCITT) 于 1998 年提出的电话/会议电视的建议标准, 该标准又称为 P×64K 标准, 其中 P 是取值为 1~30 的可变参数; P=1 或 P=2 时支持 1/4 通用中间格式 (Quarter Common Intermediate Format, QCIF) 的帧率较低的视频电话传输; P≤6 时支持通向中间格式 (Common Intermediate Format, CIF) 的帧率较高的电视会议数据传输。P×64K 视频压缩算法也是一种混合编码方案, 即基于 DCT 的变换编码和带有运动预测差分脉冲编码调制 (DPCM) 的预测编码方法的混合。在低传输率时 (P=1 或 P=2, 即 64Kb/s 或 128Kb/s) 除 QCIF 外还可以使用亚帧技术, 即每间隔一帧 (或数帧) 处理一帧, 压缩比例可达 50:1 左右。CCITT 推出的 H.263 标准用于低位速率通信的电视图像编码。

6.3.8 图形、图像文件格式

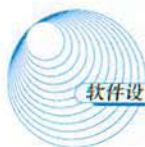
数字图像在计算机中存储时, 其文件格式繁多, 下面简单介绍几种常用的文件格式。

1. BMP 文件 (.BMP)

BMP (Bitmap-File) 图像文件是 Windows 操作系统采用的图像文件格式, 在 Windows 环境下运行的所有图像处理软件几乎都支持 BMP 图像文件格式。它是一种与设备无关的位图格式, 目的是为了让 Windows 能够在任何类型的显示设备上输出所存储的图像。BMP 采用位映射存储格式, 除了图像深度可选以外, 一般不采用其他任何压缩, 所以占用的存储空间较大。BMP 文件的图像深度可选 1 位、4 位、8 位及 24 位, 既有黑白、16 色、256 色和真彩色之分。

2. GIF 文件 (.GIF)

GIF 是 CompuServe 公司开发的图像文件格式, 它以数据块为单位来存储图像的相关信息。GIF 文件格式采用了 LZW (Lempel-Ziv-Walch) 无损压缩算法按扫描行压缩图像数据。它可以在一个文件中存放多幅彩色图像, 每一幅图像都由一个图像描述符、可选的局部彩色表和图像数据组成。如果把存储于一个文件中的多幅图像逐幅读出来显示到屏幕上, 可以像播放幻灯片那样显示或者构成简单的动画效果。GIF 的图像深度从 1 位到 8 位, 即最多支持 256 种色彩的



图像。

GIF 文件格式定义了两种数据存储方式,一种是按行连续存储,存储顺序与显示器的显示顺序相同;另一种是按交叉方式存储。由于显示图像需要较长的时间,使用这种方法存放图像数据,用户可以在图像数据全部收到之前看到这幅图像的全貌,而不觉得等待时间太长。目前,GIF 文件格式在 HTML 文档中得到广泛使用。

3. TIFF 文件 (.TIF)

TIFF 文件是由 Aldus 和 Microsoft 公司为扫描仪和桌面出版系统研制开发的一种较为通用的图像文件格式。TIFF 是电子出版 CD-ROM 中的一个重要的图像文件格式。TIFF 格式非常灵活易变,它又定义了 4 类不同的格式:TIFF-B 适用于二值图像;TIFF-G 适用于黑白灰度图像;TIFF-P 适用于带调色板的彩色图像;TIFF-R 适用于 RGB 真彩图像。无论在视觉上还是其他方面,都能把任何图像编码成二进制形式而不丢失任何属性。

4. PCX 文件 (.PCX)

PCX 文件是 PC Paintbrush (PC 画笔)的图像文件格式。PCX 的图像深度可选为 1 位、4 位、8 位,对应单色、16 色及 256 色,不支持真彩色。PCX 文件采用 RLE 行程编码,文件体中存放的是压缩后的图像数据。因此,将采集到的图像数据写成 PCX 格式文件时,要对其进行 RLE 编码;而读取一个 PCX 文件时首先要对其进行解码,才能进一步显示和处理。

5. PNG 文件格式

PNG 文件是作为 GIF 的替代品而开发的,它能够避免使用 GIF 文件所遇到的常见问题。它从 GIF 那里继承了许多特征,增加了一些 GIF 文件所没有的特性。用来存储灰度图像时,灰度图像的深度可达 16 位,存储彩色图像时,彩色图像的深度可达 48 位。在压缩数据时,它采用了一种在 LZ77 算法基础上改进的无损压缩算法。

6. JPEG 文件 (.JPG)

JPEG 文件采用一种有损压缩算法,其压缩比约为 5:1 至 50:1,甚至更高。对一幅图像按 JPEG 格式进行压缩时,可以根据压缩比与压缩效果要求选择压缩质量因子。JPG 格式文件的压缩比例很高,非常适用于要处理大量图像的场合,它是一种有损压缩的静态图像文件存储格式,压缩比例可以选择,支持灰度图像、RGB 真彩色图像和 CMYK 真彩色图像。

7. Targe 文件 (.TGA)

Targe 文件格式用于存储彩色图像,可支持任意大小的图像,最高彩色数可达 32 位。专业

图形用户经常使用 TGA 点阵格式保存具有真实感的三维有光源图像。

8. WMF 文件 (.WMF)

WMF 文件只使用在 Windows 中,它保存的不是点阵信息,而是函数调用信息。它将图像保存为一系列 GDI(图形设备接口)的函数调用,在恢复时,应用程序执行源文件(即执行一个个函数调用)在输出设备上画出图像。WMF 文件具有设备无关性,文件结构好,但是解码复杂,其效率比较低。

9. EPS 文件 (.EPS)

EPS 文件是用 PostScript 语言描述的 ASCII 图形文件,在 PostScript 图形打印机上能打印出高品质的图形,能够表示 32 位图形(图像)。EPS 文件格式分为 Photoshop EPS 格式和标准 EPS 格式,其中标准 EPS 格式又可分为图形格式和图像格式。

10. DIF 文件 (.DIF)

DIF 文件是 AutoCAD 中的图形,它以 ASCII 方式存储图像,表现图形在尺寸大小方面十分精确,可以被 CorelDraw, 3DS 等软件调用编辑。

11. CDR 文件 (.CDR)

CDR 文件是 CorelDraw 的文件格式,所有 CorelDraw 应用程序均能使用的图形(图像)文件。

6.4 动画和视频

6.4.1 动画

动画是将静态的图像、图形及图画等按一定时间顺序显示而形成连续的动态画面。从传统意义上说,动画是通过在连续多格的胶片上拍摄一系列画面,并将胶片以一定的速度放映,从而产生动态视觉的技术和艺术。电影放映的标准是每秒放映 24 帧(画面),每秒遮挡 24 次,刷新率是每秒 48 次。一般说来,动画是一种动态生成一系列相关画面的处理方法,其中的每一幅与前一幅略有不同。计算机动画是在传统动画的基础上,使用计算机图形图像技术而迅速发展起来的一门高新技术。计算机动画是采用连续播放静止图像的方法产生景物运动的效果,即使用计算机产生图形、纹像运动的技术。计算机动画的原理与传统动画基本相同,只是在传统动画的基础上把计算机技术用于动画的处理和应用,并可以达到传统动画所达不到的效果。



动画的内容不仅实体在运动,而且色调、纹理、光影效果也可以不断改变。计算机生成的动画不仅可记录在胶片上,而且还可以记录在磁带、磁盘和光盘上,放映时不仅使用计算机显示器显示,而且可以使用电视机屏幕显示以及使用投影仪投影到银幕的方法显示。在多媒体应用中,计算机动画可以十分简单,也可以十分复杂,从某个对象、物体的运动到电视广告、动画片等。

动画的本质是运动。根据运动的控制方式可将计算机动画分为实时动画和逐帧动画两种。实时动画是用算法来实现物体的运动;逐帧动画是在传统动画基础上引申而来的,也即通过一帧一帧显示动画的图像序列而实现运动的效果。根据视觉空间的不同,计算机动画又有二维动画和三维动画之分。

1. 实时动画

实时动画采用各种算法来实现运动物体的运动控制。采用的算法有运动学算法、动力学算法、反向运动学算法、反向动力学算法、随机运动算法等。在实时动画中,计算机对输入的数据进行快速处理,并在人眼察觉不到的时间内将结果随时显示出来。实时动画的响应时间与许多因素有关,如动画图像大小、动画图像复杂程度、运算速度快慢(计算机)以及图形的计算是采用软件还是硬件等。

2. 矢量动画

矢量动画是由矢量图衍生出的动画形式。矢量图是利用数学函数来记录和表示图形线条、颜色、尺寸、坐标等属性,矢量动画通过各种算法实现各种动画效果,如位移、变形、变色等。也就是说,矢量动画是通过计算机的处理,使矢量图产生运动效果形成的动画。使用矢量动画,可以使一个物体在屏幕上运动,并改变其形状、大小、颜色、透明度、旋转角度以及其他一些属性参数。矢量动画采用实时绘制的方式显示一幅矢量图,当图形放大或缩小时,都保持光滑的线条,不会影响质量,也不会改变文件的容量。

3. 二维动画

二维动画是对传统动画的一个改进,它不仅具有模拟传统动画的制作功能,而且可以发挥计算机所特有的功能,如生成的图像可以复制、粘贴、翻转、放大/缩小、任意移位以及自动计算等。图形、图像技术都是计算机动画处理的基础。图像是指用像素点组成的画面,而图形是指几何形体组成的画面。在二维动画处理中,图像技术有利于绘制实际景物,可用于绘制关键帧、画面叠加、数据生成;图形技术有利于处理线条组成的画面,可用于自动或半自动的中间画面生成。计算机在二维动画中的作用包括输入和编辑关键帧、计算和生成中间帧、定义和显示运动路径、产生特技效果、实现画面与声音的同步、控制运动系列的记录等。二维动画的处理主要包括两个基本步骤:第一个步骤是屏幕绘画;第二个步骤是动画生成。屏幕绘画主要是

使用图像处理软件完成,有时为了自动或半自动生成动画也采用图形方法来描述画面。动画生成以屏幕绘画的结果(作为关键帧)为基础进行生成处理,最终完成动画创作。

4. 三维动画

三维画面中的景物有正面,也有侧面和反面,调整三维空间的视点,能看到不同的内容。二维画面则不然,无论怎么看,画面的内容是不变的。三维与二维动画的区别主要在于采用不同的方法获得动画中的景物运动效果。三维动画的制作过程不同于传统动画制作。根据剧情的要求,首先要建立角色、实物和景物的三维数据模型,再对模型进行光照明着色(真实感设计),然后使模型动起来,即模型可以在计算机控制下在三维空间中运动,或近或远,或旋转或移动,或变形或变色等,最后对运动的模型重新生成图像再刷新屏幕,形成运动图像。

建立三维动画物体模型称为造型,也就是在计算机内生成一个具有一定形体的几何模型。在计算机中大致有3种形式来记录一个物体的模型。

(1) 线框模型:用线条来描述一个形体,一般包括顶点和棱边。例如用8条线来描述一个立方体。

(2) 表面模型:用面的组合来描述形体,如用6个面来描述一个立方体。

(3) 实体模型:任何一个物体都可以分解成若干个基本形体的组合,如一个立方体可以分解为各种形体的组合。这种用基本形体组合物体的模型就是实体模型。

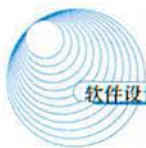
三维动画的处理需要综合使用上述3种模型。一般情况下,先用线框模型进行概念设计,再将线框模型处理成表面模型以方便显示,使用实体模型进行动画处理。同一形体的3种模型可以相互转换。

物体模型只有通过光和色的渲染,才能产生自然界中常见的真实物体效果,这在动画中称为着色(真实感设计)。对物体着色是产生真实感图形图像的重要过程,它涉及到物体的材质、纹理以及照射的光源等方面。

(1) 材质:描述任何物体除了造型以外,还必须有一定的附加特征(属性),来指明它的外在特性。物体的外在特性在很大程度上取决于构成它的材料。一般把材料的性质简称为材质。不同的材质表现出的质感是不同的。材质主要用来说明物体对入射光线作出的反应。一般来说,光线或被反射、吸收,或者被透射、折射。反射的色光正是该物体呈现的颜色,而透射、折射产生的色光与材质有很大关系。

(2) 纹理:纹理是物体表面细节,大多数物体的表面具有纹理。有了纹理可以改变物体的外观,甚至改变其形状。物体的纹理一般分为两种:一种是颜色纹理,如墙面贴纸、陶器上的图案等。颜色纹理取决于物体表面的光学性质。另一种是几何纹理,如人的皮肤、橘子的褶皱等。几何纹理与物体表面的微观几何形状有关。

(3) 光源:给一个场景着色时必须知道有关光源的特性:光源的位置、颜色、亮度、方向



等, 这些信息要由用户通过照明模型设定。决定光照射到物体表面并形成颜色的方法称为浓淡处理。浓淡处理要用到物体表面的几何材质信息, 并对入射光进行考察, 从而找出表面反射的色光。在进行浓淡处理时, 一般对每个物体表面的每一个点分别进行处理, 而且仅考虑来自光源的光照效果, 而不考虑来自其他物体发出的光的影响。

三维动画处理的基本目的是控制形体模型的运动, 获得运动显示效果。其处理过程中涉及建立线框模型、表面模型和实体模型。此外, 一个好的三维动画应用系统能够将形体置于指定的灯光环境中, 使形体的色彩在灯光下生成光线反映和阴影效果。运动物体不仅表现为几何位置改变, 还带有光、色、受力、碰撞以及物体本身的变形等。动画控制也称为运动模拟。首先, 计算机要确定每个物体的位置和相互关系, 建立其运动轨迹和速度, 选择运动形式(平移、旋转、扭曲等)。然后, 需确定物体形体的变态方式和变异速度。如果光源确定好了以后, 调整拍摄的位置、方向、运动轨迹及速度, 就可以显示观看画面效果。

三维动画最终要生成一幅幅二维画面, 并按一定格式记录下来, 这个过程称为动画生成。动画生成后, 可以在屏幕上播放, 也可以录制在光盘或录像带上。

6.4.2 模拟视频

1. 模拟视频原理

电视是当代最有影响的多媒体信息传播工具, 在综合文、图、声、像等作为信息传播媒体这一点上完全与多媒体系统相同, 不同的是电视系统不具备交互性, 传播的信号是模拟信号。电视信号记录的是连续的图像或视像以及伴音(声音)信号。电视信号通过光栅扫描的方法显示在荧光屏(屏幕)上, 扫描从荧光屏的顶部开始, 一行一行地向下扫描, 直至荧光屏的最底部, 然后返回到顶部, 从新开始扫描。这个过程产生的一个有序的图像信号的集合, 组成了电视图像中的一幅图像, 称为一帧, 连续不断的图像序列就形成了动态视频图像。水平扫描线所能分辨出的点数称为水平分辨率, 一帧中垂直扫描的行数称为垂直分辨率。一般来说, 点越小, 线越细, 分辨率越高。每秒钟所扫描的帧数就是帧频, 一般在每秒 25 帧时人眼就不会感觉到闪烁。彩色电视系统采用相加混色, 使用 RGB 作为三基色进行配色, 产生 R、G、B 3 个输出信号。RGB 信号可以分别传输, 也可以组合起来传输。根据亮度色度原理, 任何彩色信号都可以分解为亮度和色度。

2. 彩色电视的制式

电视信号的标准也称为电视的制式, 目前世界各地使用的标准不完全相同, 制式的区分主要在于其帧频的不同、分辨率的不同、信号带宽及载频的不同、彩色空间的转换关系不同等。世界上现行的彩色电视制式主要有 NTSC 制、PAL 制和 SECAM 制 3 种, 见表 6-2 所示。

表 6-2 彩色数字电视制式

TV 制式	帧频(Hz)	行/帧	亮度带宽 (MHz)	彩色副载波 (MHz)	色度带宽 (MHz)	声音载波 (MHz)
NTSCM	30	525	4.2	3.58	1.3 (I)、0.6 (Q)	4.5
PAL	25	625	6.0	4.43	1.3 (U)、1.3 (V)	6.5
SECAM	25	625	6.0	4.25	>1.0 (U)、>1.0 (V)	6.5

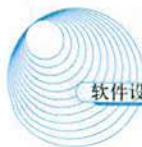
美国、加拿大、日本、韩国、中国台湾、菲律宾等国家和地区采用 NTSCM 制式；德国、英国、中国大陆、中国香港、新西兰等国家和地区采用 PAL 制式；法国、东欧、中东一带采用 SECAM 制式。

我国电视制式 (PAL) 采用 625 行隔行扫描光栅，分两场扫描。行扫描频率为 15625Hz，周期为 64 μ s；场扫描频率为 50Hz，周期为 20ms；帧频是 25Hz，周期为 40ms。在发送电视信号时，每一行中传送图像的时间是 52.2 μ s，对应行扫描的正程时间，其余的 11.8 μ s 不传送图像，对应行扫描的逆程时间加入行消隐信号和行同步信号，这样不影响行扫描发送或显示图像信息。每一场扫描的行数为 625/2 行，其中 25 行作场回扫，不传送图像。

采用隔行扫描比采用逐行扫描所占用的信号传输带宽要减少一半，这样，有利于信道的利用；有利于信号传输和处理。采用每秒 25 帧的帧频 (25 Hz) 能以最少的信号容量有效地满足人眼的视觉残留特性。采用 50Hz 的场频是因为我国的电网频率为 50Hz，采用 50Hz 的场刷新频率可有效地去掉电网信号的干扰。

6.4.3 数字视频

视频信息是指活动的、连续的图像序列。一幅图像称为一帧，帧是构成视频信息的基本单元。在多媒体应用系统中，视频以其直观和生动等特点得到广泛的应用。视频与动画一样，是由一幅幅帧序列组成，这些帧以一定的速率播放，使观看者得到连续运动的感觉。计算机的数字视频是基于数字技术的图像显示标准，它可将模拟视频信号输入到计算机进行数字化视频编辑制成数字视频。全屏幕视频是指显示的视频图像充满整个屏幕，能以 30 帧/秒的速度刷新画面，使画面不会产生闪烁和不连贯的现象。电视机、激光视盘、摄像机等都可提供丰富多彩的模拟视频信号，常常需要把这些信号与计算机图形图像结合在一个共同的空间，通过处理达到最佳的效果，然后输出到计算机的显示器或其他电视设备上。模拟视频信号进入计算机，首先需要解决模拟视频信息的数字化问题。与音频数字化一样，视频数字化的目的是将模拟信号经模数转换和彩色空间变换等过程，转换成计算机可以显示和处理的数字信号。由于电视和计算机的显示机制不同，因此要在计算机上显示视频图像需要作许多处理。例如，电视是隔行扫描，计算机的显示器通常是逐行扫描；电视是亮度 (Y) 和色度 (C) 的复合编码，而 PC 机的显示



器工作在 RGB 空间;电视图像的分辨率和显示屏的分辨率也各不相同。这些问题在电视图像数字化过程中都需考虑。一般,对模拟视频信息进行数字化采取如下方式:

(1) 先从复合彩色电视图像中分离出彩色分量,然后数字化。目前市场上的大多数电视信号都是复合的全电视信号,如录像带、激光视盘等存储设备上的电视信号。对这类信号的数字化,通常是将其分离成 YUV、YIQ 或 RGB 彩色空间的分量信号,然后用 3 个 A/D 转换器分别进行数字化。这种方式称为复合数字化。

(2) 先对全彩色电视信号数字化,然后在数字域中进行分离,以获得 YUV、YIQ 或 RGB 分量信号。用这种方法对电视图像数字化时,只需一个高速 A/D 转换器。这种方式称为分量数字化。

视频信息数字化的过程比声音复杂一些,它是以一幅幅彩色画面为单位进行的。分量数字化方式是较多使用的一种方式。电视信号使用的彩色空间是 YUV 空间,即每幅彩色画面有亮度(Y)和色度(U、V)3个分量,对这3个分量需分别进行取样和量化,得到一幅数字图像。由于人眼对色度信号的敏感程度远不如对亮度信号那么灵敏,所以色度信号的取样频率可以比亮度信号的取样频率低一些,以减少数字视频的数据量。目前使用的色度信号取样格式如表 6-3 所示。

表 6-3 色度信号取样格式

格式	说明
4:4:4 格式	指在每条扫描线上每 4 个连续的取样点取 4 个亮点样本 Y 、4 个色度样本 C_r 和 4 个色度样本 C_b , 这就相当于每个像素用 3 个样本表示
4:2:2 格式	指在每条扫描线上每 4 个连续的取样点取 4 个亮点样本 Y 、2 个色度样本 C_r 和 2 个色度样本 C_b , 平均每个像素用 2 个样本表示
4:2:0 格式	指在水平和垂直方向上每 2 个连续的取样点上取 2 个亮点样本 Y 、1 个色度样本 C_r 和 1 个色度样本 C_b , 平均每个像素用 1.5 个样本表示。H.261、H.263、和 MPEG-1 视频标准均使用这种取样格式, 每个像素平均使用 12 比特来表示
4:1:1 格式	指在每条扫描线上每 4 个连续的取样点取 4 个亮点样本 Y 、1 个色度样本 C_r 和 1 个色度样本 C_b , 平均每个像素用 1.5 个样本表示。数字录像机 DVC 使用这种格式

CCIR601 标准推荐使用 4:2:2 格式,并对采样频率、采样结构、彩色空间转换等都做了严格的规定。使用 4:2:2 格式时,亮点样本 Y 用 13.5MHz 采样频率,色度样本 C_r 、 C_b 用 6.75MHz 的采样频率。

6.4.4 数字视频标准

国际无线电咨询委员会(International Radio Consultative Committee, CCIR)制定的广播级

质量数字电视编码标准,即 CCIR601 标准,为 PAL、NTSC 和 SECAM 电视制式之间确定了共同的数字化参数。该标准规定了彩色电视图像转换成数字图像所使用的采样频率、采样结构、彩色空间转换等。这个标准对多媒体的开发和应用十分重要。

1. 采样频率

CCIR 为 PAL、NTSC 和 SECAM 电视制式制定的共同的电视图像采样频率标准为:

采样频率=13.5MHz

2. 分辨率

PAL 和 SECAM 制式的亮度信号,每一扫描行采集 864 个样本点,而对于 NTSC 制式的亮度信号,每一扫描行采集 858 个样本点。CCIR 601 规定对所有制式,每一扫描行的有效样本点数均为 720 个。

3. 数据量

CCIR 601 规定,每个样本点都按 8 位数字化,即有 256 个等级。但实际上亮度信号占 220 级,色度信号占 225 级,其他位作同步、编码等控制使用。

6.4.5 视频压缩编码

数字图像数据的数据量大,而数字视频信息的数据量就更加突出。例如,每帧 352×240 像素点,图像深度 16 位的图像,其数据量约为 1.3Mb,每秒 30 帧,其数据量就高达 40Mb/s,这样大的数据量无论是传输、存储还是处理,都是极大的负担。为了解决这个问题必须对数字视频信息进行压缩编码处理。

视频压缩的目标是在尽可能保证视觉效果的前提下减少视频数据率。视频是连续的静态图像,其压缩编码算法与静态图像的压缩编码算法有某些共同之处。但是视频还有其自身的特性,在压缩时必须考虑其运动特性。由于视频信息中各画面内部有很强信息相关性,相邻画面又有高度的相容性(连贯性),再加上人眼的视觉特性,所以数字视频的数据量可压缩几十倍甚至几百倍。视频信息压缩编码的方法很多,一般在选择或设计视频压缩编码算法时需要掌握一些视频压缩的基本概念。

1. 无损压缩与有损压缩

视频压缩中无损和有损压缩的概念与静态图像基本类似。无损压缩指压缩前和解压缩后的数据完全一致。多数的无损压缩都采用 RLE 行程编码算法。这种算法特别适合于由计算机生成的图像,它们一般具有连续的色调。但是无损算法一般对数字视频和自然图像的压缩效果不理



想, 因为其色调细腻, 不具备大块的连续色调。

有损压缩意味着解压缩后的数据与压缩前的数据不一致。在压缩的过程中要丢失一些人眼和人耳所不敏感的图像或声音信息, 而且丢失的信息不可恢复。几乎所有高压压缩的算法都采用有损压缩, 这样才能达到低数据率的目标。丢失的数据率与压缩比有关, 压缩比越小, 丢失的数据越多, 解压缩后的效果一般越差。此外, 某些有损压缩算法采用多次重复压缩的方式, 这样还会引起额外的数据丢失。

2. 帧内和帧间压缩

帧内压缩也称为空间压缩。同一景物表面上各采样点的颜色之间往往存在着连贯性, 但是基于离散像素采样来表示景物颜色的方式通常没有利用景物表面颜色的空间连贯性, 从而产生了空间冗余。当压缩一帧视频时, 仅考虑本帧的数据而不考虑相邻帧之间的冗余信息, 这实际上与静态图像压缩类似。由于帧内压缩时各个帧之间没有相互关系, 所以压缩后的视频数据仍可以以帧为单位进行编码。帧内压缩一般达不到很高的压缩(很小的压缩比)。

视频具有运动的特性, 故还可以采用帧间压缩的方法。采用帧间压缩基于许多视频或动画的连续前后两帧具有很大的相关性, 或者说前后两帧信息变化很小的特点。例如, 演示一个球在静态背景前滚动的视频片段中, 连续两帧中的大部分的图像是基本不变的(背景不变), 即连续的视频其相邻帧之间具有冗余信息。根据这一特性, 压缩相邻之间的冗余就可以进一步提高压缩量, 减小压缩比。帧间压缩也称为时间压缩, 它通过比较时间轴上不同帧之间的数据进行压缩。帧差值算法是一种典型的时间压缩法, 它通过比较本帧与相邻帧之间的差异, 仅记录本帧与其相邻帧的差值, 这样可以大大减少数据量。例如, 如果一段视频不包含大量超常的剧烈运动景象, 而是由一帧一帧的正常运动构成, 采用这种算法就可以达到很好的效果。

3. 对称和不对称编码

对称性是压缩编码的一个关键特征。对称意味着压缩和解压缩占用相同的计算处理能力和时间。对称算法适合实时压缩和传送视频, 如视频会议应用就是以采用对称的压缩编码算法为好。而在电子出版和其他多媒体应用中, 一般把视频预先压缩处理好, 尔后再播放, 因此可以采用不对称编码。不对称或非对称意味着压缩时需要花费大量的处理能力和时间, 而解压缩时则能较好地实时回放, 即以不同的速度进行压缩和解压缩。一般地说, 压缩一段视频的时间比回放(解压缩)该视频的时间要多得多。例如, 压缩 3 分钟的视频片段可能需要 10 多分钟的时间, 而该片段实时回放只需 3 分钟。

目前, 国际标准化组织制定的有关视频(及其伴音)压缩编码的几种标准及其应用范围可参见表 6-4。

表 6-4 压缩编码的标准

名称	源图像格式	压缩后的码率	主要应用
MPEG—1	CIF 格式	1.5Mb/s	适用于 VCD
CCITT H.261	CIF 格式	P×64 Kb/s	应用于视频通信，如可视电话、会议电视等
	QCIF 格式	P=1、2 时支持 QCIF	
		P≥6 时支持 QIF	
MPEG—2	720×576×25	5~15 Mb/s	DVD、150 路卫星电视直播等
	352×288×25	<5 Mb/s	交互式多媒体应用等
	1440×1152×50	80 Mb/s	HDTV 领域
MPEG—4	多种不同的视频格式	最低可达 64 Kb/s	虚拟现实、远程教育、交互式视频等

6.4.6 视频文件格式

1. GIF 文件 (.GIF)

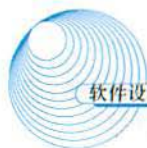
GIF (Graphics Interchange Format) 是 CompuServe 公司推出的一种高压缩比的彩色图像文件。GIF 格式采用无损压缩方法中效率较高的 LZW 算法，主要用于图像文件的网络传输。考虑到网络传输的实际情况，GIF 图像格式除了一般的逐行显示方式之外，还增加了渐显方式，也就是说，在图像传输过程中，用户可以先看到图像的大致轮廓，然后随着传输过程的继续而逐渐看清图像的细节部分，从而适应了用户的观赏心理。目前因特网上大量采用的彩色动画文件多为这种 GIF 格式。

2. Flic 文件 (.FLI/.FLC)

Flic 文件是 Autodesk 公司在其出品的 Autodesk Animator/ Animator Pro/3D Studio 等 2D/3D 动画制作软件中采用的彩色动画文件格式。其中.FLI 是最初的基于 320×200 分辨率的动画文件格式，.FLC 是.FLI 的进一步扩展，采用了更高效的数据压缩技术，其分辨率也不再局限于 320×200。Flic 文件采用行程编码 (RLE) 算法和 Delta 算法进行无损的数据压缩，具有较高的数据压缩率。

3. AVI 文件 (.AVI)

AVI (Audio Video Interleaved) 是 Microsoft 公司开发的一种符合 RIFF 文件规范的数字音频与视频文件格式，Windows 95/98、OS/2 等多数操作系统直接支持。AVI 格式允许视频和音频交错在一起同步播放，支持 256 色和 RLE 压缩，但 AVI 文件并未限定压缩标准。因此，AVI 文件格式只是作为控制界面上的标准，不具有兼容性，用不同压缩算法生成的 AVI 文件，必须使用相同的解压缩算法才能播放出来。AVI 文件目前主要应用在多媒体光盘上，用来保存电影、



电视等各种影像信息,有时也出现在因特网上,供用户下载、欣赏新影片的片段。

4. Quick Time 文件 (.MOV/.QT)

Quick Time 是 Apple 公司开发的一种音频、视频文件格式,用于保存音频和视频信息,具有先进的视频和音频功能,被 Apple Mac OS、Windows 95/98/NT 等主流平台支持。Quick Time 文件支持 25 位彩色,支持 RLE、JPEG 等领先的集成压缩技术,提供 150 多种视频效果,并配有提供了 200 多种 MIDI 兼容音响和设备的声音装置。新版本的 Quick Time 进一步扩展了原有功能,包含了基于 Internet 应用的关键特性,能够通过 Internet 提供实时的数字化信息流、工作流与文件回放功能。此外,Quick Time 还采用了 Quick Time VR (QTVR) 技术的虚拟现实技术,用户通过鼠标或键盘的交互式控制,可以观察某一地点周围 360 度的景象,或者从空间任何角度观察某一物体。Quick Time 以其领先的多媒体技术和跨平台特性;较小的存储空间要求;技术细节的独立性以及系统的高度开放性,得到广泛的认可和应用。

5. MPEG 文件 (.MPEG/.MPG/.DAT)

MPEG 文件格式是运动图像压缩算法的国际标准,它包括 MPEG 视频、MPEG 音频和 MPEG 系统(视频、音频同步)3 个部分。MPEG 压缩标准是针对运动图像设计的,其基本方法是:单位时间内采集并保存第一帧信息,然后只存储其余帧对第一帧发生变化的部分,从而达到压缩的目的。MPEG 的平均压缩比为 50:1,最高可达 200:1,压缩效率非常高,同时图像和音响的质量也非常好,并且在 PC 机上有统一的标准格式,兼容性相当好。

6. RealVideo 文件 (.RM)

RealVideo 文件是 Real Networks 公司开发的一种新型流式视频文件格式,他包含在 Real Networks 公司所制定的音频视频压缩规范 RealMideo 中,主要用来在低速率的广域网上实时传输活动视频影像,可以根据网络数据传输速率的不同而采用不同的压缩比率,从而实现影像数据的实时传输和实时播放。RealVideo 除了可以以普通的视频文件形式播放之外,还可以与 RealVideo 服务器相配合,在数据传输过程中边下载边播放视频影像,而不必像大多数视频文件那样,必须先下载然后才能播放。

6.5 多媒体网络

计算机网络是将多个计算机连接起来以实现计算机通信以及广泛实现多媒体信息共享,在网络上的分布式与协作操作就不可避免。多媒体空间的合理分布和有效的协作操作将极大地缩小个体与群体、局部与全球的工作差距。超越时空限制、充分利用信息、协同合作、相互交流、

节约大量的时间和经费等是多媒体的基本目标。

网络具备传播信息的强大功能，并且在实际生活中扮演了媒体的角色。其次，国内各种机构都已开辟网上传播的新领域，网络报纸杂志、网络广播娱乐、网上教育、电子商务等业务应运而生，互联网逐渐达到了作为大众传播媒体的标准。

6.5.1 超文本与超媒体

1. 超文本的概念

超文本是一种文本，与一般的文本文件的差别主要是组织方式不同，它是将文本中遇到的一些相关内容通过链接组织在一起，用户可以很方便地阅览这些相关内容。超文本是一种文本管理技术，它以节点为单位组织信息，在节点与节点之间通过表示它们之间关系的链加以连接，构成特定内容的信息网络。节点、链和网络是超文本所包含的3个基本要素。

(1) 节点：超文本中存储信息的单元，由若干个文本信息块（可以是若干屏、窗口、文件或小块信息）组成。节点大小按需要而定。

(2) 链：建立不同节点（信息块）之间的联系。每个节点都有若干个指向其他节点或从其他节点指向该节点的指针，该指针称为链。链通常是有向的，即从链源（源节点）指向链宿（目的节点）。链源可以是热字、热区、图元、热点或节点等。一般链宿都是节点。

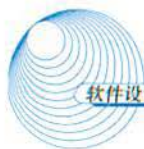
(3) 网络：由节点和链组成的一个非单一、非顺序的非线性网状结构。

文本中的词、短语、图像、声音剪辑或影视剪辑之间的链接，或者其他的文件、超文本文件的链接，称为超链接（热链接）。词、短语、图像、声音剪辑或影视剪辑和其他的文件通常被称为对象或者称为文档元素，因此超链接是对象之间或者文档元素之间的链接。建立相互链接的这些对象不受空间位置的限制，它们可以在同一个文件内也可以在不同的文件之间，还可以通过互联网与世界上的任何一台联网计算机上的文件建立链接关系。

2. 超媒体的概念

用超文本方式组织和处理多媒体信息就是超媒体。超媒体不仅包含文字而且还可以包含图形、图像、动画、声音和影视图像片断，这些媒体之间也是用超链接组织的，而且它们之间的链接也是错综复杂的。超媒体与超文本之间的不同是，超文本主要是以文字的形式表示信息，建立的链接关系主要是文句之间的链接关系。超媒体除使用文本外，还使用图形、图像、动画、声音或影视片断等多种媒体来表示信息，建立的链接关系是图形、图像、动画、声音或影视片断等多种媒体之间的链接关系。

网页是Web站点上的文档。在Web网页上，为了区分有链接关系和没有链接关系的文档元素，对有链接关系的文档元素通常用不同颜色或下划线来表示。



6.5.2 流媒体的基本概念

流媒体是指在网络中使用流式传输技术的连续时基媒体,而流媒体技术是指把连续的影像和声音信息经过压缩处理之后放到专用的流服务器上,让浏览者一边下载一边观看、收听,而不需要等到整个多媒体文件下载完成就可以即时观看和收听的技术。流媒体融合了多种网络以及音视频技术,在网络中要实现流媒体技术,必须完成流媒体的制作、发布、传播、播放等环节。

(1) 流媒体系统通过某种流媒体技术,完成流媒体文件的压缩生成,经过服务器发布,然后在客户端完成流媒体文件的解压播放的整个过程。因此,一个流媒体系统一般由3部分组成,流媒体开发工具,用来生成流式格式的媒体文件;流媒体服务器组件,用来通过网络服务器发布流媒体文件;流媒体播放器,用于客户端对流媒体文件的解压和播放。目前应用比较广泛的流媒体系统主要有 Windows Media 系统、Real System 系统和 Quick Time 系统等。

(2) 流媒体的传输一般采用建立在用户数据报协议 UDP (User Datagram Protocol) 之上的实时传输协议和实时流协议 RTP/RTSP 来传输实时的影音数据。RTP 是针对多媒体数据流的一种传输协议,它被定义为在一对一或一对多的传输情况下工作,提供时间信息和实现流同步。RTSP 协议定义了一对多的应用程序如何有效地通过 IP 网络传送多媒体数据。

(3) 流式文件格式与多媒体压缩文件有所不同,编码的目的是为了适合在网络环境中边下载边播放。将压缩文件编码成流式文件,还需增加许多附加信息,为了使客户端接收到的数据包可以重新有序地播放。在实际的网络应用环境,并不包含流媒体数据文件,而是流媒体发布文件,例如.RAM、.ASX 等,它们本身不提供压缩格式,也不描述影视数据,其作用是以特定的方式安排影视数据的播放。不同流媒体系统具有不同的流式文件格式,例如,Real System 系统具有的文件格式有 RM、.RA、.RP、.RT; Windows Media 系统具有的文件格式是.ASF、.ASX。

(4) 浏览器是通过互联网邮件 MIME 来识别各种不同的简单文件格式。Web 浏览器都是基于 HTTP 协议的,HTTP 协议中都内建有 MIME。Web 浏览器能够通过 HTTP 中内建的 MIME 来标记 Web 上的多媒体文件格式,包括各种流式文件。

(5) 媒体播放器是一个应用软件,主要功能是用于播放多种格式的音频、视频序列。可以作为单独的应用程序运行,或作为一个复合文档中的一个嵌入对象。

6.5.3 互联网上获取声音和影视的方法

多媒体网络技术 (Multimedia Networking) 在互联网上有很多应用,大致可分成两类:一类是以文本为主的数据通信,包括文件传输、电子邮件、网络新闻和 Web 等;另一类是以声音和视频图像为主的通信,通常把任何一种声音通信或图像通信的网络应用称为多媒体网络应用。

通常,声音或视频文件放在 Web 服务器上,由 Web 服务器通过 HTTP 协议把文件传送给用户。也可将声音或视频文件放在声音/视频流式播放服务器(Streaming Server)上,由流式播放服务器通过流放协议把文件传送给用户。流式播放服务器简称流放服务器。

目前声音和视频点播应用还没有完全直接集成到现在的 Web 浏览器中,所以一般采用媒体播放器来播放声音、音乐、动画和影视。典型的媒体播放器具有解压缩、消除抖动、错误纠正、用户播放控制等功能。现在可以将多媒体应用插件(Plug in)嵌入浏览器内部,与浏览器软件协同工作。这种技术把媒体播放器的用户接口软件放在 Web 客户机的用户界面上,浏览器在当前 Web 页面上保留屏幕空间,并且由媒体播放器来管理。客户机可使用多种方法来读取声音和影视文件,其中常见的方法有如下 3 种。

1. 通过 Web 浏览器把声音/影视文件从 Web 服务器传送给媒体播放器

客户机读取声音/影视文件的最简方法是将声音/影视文件放到 Web 服务器上,然后通过浏览器把文件传送给媒体播放器,其过程如下:

(1) Web 浏览器与 Web 服务器建立 TCP 连接,然后提交 HTTP 请求,请求传送声音/影视文件。

(2) Web 服务器给 Web 浏览器发送响应请求消息以及请求的声音/影视文件。

(3) Web 浏览器检查 HTTP 响应消息中的内容的类型,调用相应的媒体播放器,然后把声音/影视文件或者是指向文件的指针传递给媒体播放器。

(4) 媒体播放器播放声音/影视文件。

这种方法,媒体播放器必须通过 Web 浏览器才能从 Web 服务器上得到声音/影视文件,需要把整个文件下载到浏览器之后才把它传送给媒体播放器。这样就产生了播放延迟,影响了播放。因此,存在较大的延迟问题。

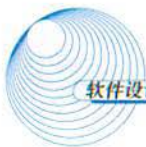
2. 直接把声音/影视文件从 Web 服务器传送给媒体播放器

采用媒体播放器与 Web 服务器直接建立链接的方法,可以改进通过 Web 浏览器产生的延迟。在 Web 服务器和媒体播放器之间建立直接的 TCP 连接,可以实现把声音/影视文件直接传送给媒体播放器,其实现方法如下:

(1) 用户通过超级链接以请求传送声音/影视文件。这个超级链接不是直接指向声音/影视文件,而是指向一个播放说明文件,这个文件包含有实际的声音/影视文件的地址(URL)。播放说明文件被封装在 HTTP 消息中。

(2) Web 浏览器接收 HTTP 响应消息中的内容的类型,调用相应的媒体播放器,然后把响应消息中的播放说明文件传送给媒体播放器。

(3) 媒体播放器直接与 Web 服务器建立 TCP 连接,然后把传送声音/影视文件的 HTTP 请



求消息发送到 TCP 连接上。

(4) 在 HTTP 响应消息中把声音/影视文件传送给该媒体播放器并开始播放。

这种方法依然使用 HTTP 传送文件,不容易获得满意的交互性能(用户与 Web 服务器),如暂停、从头开始播放。

3. 通过多媒体流放服务器将声音/影视文件传送给媒体播放器

采用 Web 服务器和流放服务器,将声音/影视文件直接传送给媒体播放器的方法,Web 服务器用于 Web 页面服务,流放服务器用于声音/影视文件服务。其结构见图 6-2 所示。采用这种结构,媒体播放器向流放服务器请求传送文件,而不是向 Web 服务器请求传送文件,媒体播放器和流放服务器之间使用流放协议进行通信,声音/影视文件可以使用 UDP(用户数据包协议)直接从流放服务器传送给媒体播放器。

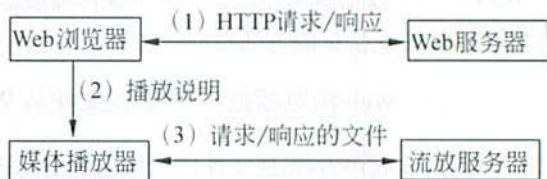


图 6-2 声音/影视文件传送给媒体播放器

6.6 多媒体计算机系统

一般的多媒体系统是由多媒体硬件系统和多媒体软件系统组成。也即将多媒体信息和计算机交互式控制相结合,由对媒体信号的获取、生成、存储、处理和传输数字化所组成的一个完整的系统。

通常将具有对多种媒体进行处理能力的计算机称为多媒体计算机(Multimedia Personal Computer, MPC)。传统的 PC 机处理的信息往往仅限于文字和数字,人机之间的交互只能通过键盘和显示器,缺乏多样性的交流信息途径。为了改变人机交互的接口,使计算机能够集声、文、图、像处理于一体,适应多媒体系统功能目标和应用需求,人们一方面改进 PC 机体系结构,使 PC 机性能升级,适应更丰富、更复杂的数据类型。芯片设计技术的发展,将多媒体和通信功能集成到了 CPU 芯片中,形成了专用的多媒体微处理器,使得处理音频和视频就如处理数字和文字一样快捷。为了加快多媒体信息处理的速度,Micro Unity、Philips 等公司将媒体处理器与通用的 CPU 结合,扩展了 CPU 的多媒体处理和通信功能。Intel 公司推出了带有 MMX 技术的处理器。MMX 技术提供了面向多媒体和通信功能的特性,并保持了微处理器的体系结构。另一方面运用多媒体专用芯片和板卡,集成以 PC 机为中心的组平台。随着微电子集成电路技术和计算机技术的发展,现代高性能 PC 机为适应各种应用领域对处理速度和容量的要求,其体系结构发生了很大的变化,性能得到很大的改善,可以构成多媒体计算机。目前 PC

机的多媒体功能大都通过附加插件和设备实现的,如音频卡、视频卡、3D 图形卡、网络卡以及 CD-ROM 驱动器、扫描仪、数码相机等。因此一个完整的多媒体计算机系统由多媒体计算机硬件和多媒体计算机软件组成。

6.6.1 多媒体计算机硬件系统

多媒体硬件系统可以看成是在 PC 机的基础上进行了硬件扩充,以适应多媒体信息处理功能的需求。计算机硬件及声像等媒体输入/输出设备构成的多媒体硬件平台。

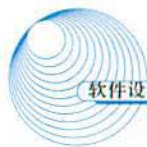
多媒体计算机的主要硬件除了常规的硬件如主机、软盘驱动器、硬盘驱动器、显示器、打印机之外,还要有音频信息处理硬件、视频信息处理硬件及光盘驱动器等。

(1) 音频卡 (Sound Card): 又称声卡,用于处理音频信息,它可以把话筒、录音机、电子乐器等输入的声音信息进行模数转换 (A/D)、压缩等处理,也可以把经过计算机处理的数字化的声音信号通过还原 (解压缩)、数模转换 (D/A) 后用音箱播放出来,或者用录音设备记录下来。

目前,市场上声卡的种类很多,较流行的有 Creative Labs 的声霸卡 (Sound Blaster) 系列。声卡的分类主要根据其数据采样量位数来确定,通常分为 8 位、16 位和 32 位等。位数越多,其量化精度越高,音质就越好。音频卡的关键技术包括数字音频、音乐合成 (FM 合成和波形表合面)、MIDI (数字音乐的国际标准,是指乐器数字接口) 和音效。数字音频部分具有 44.1kHz 的采样率,8 位以上的分辨率;具有录音和播放声音信号的功能;同时具有压缩采样信号的能力,最常用的压缩方法是自适应脉冲编码调制。数字音频的实现有不同的方法和芯片,大多数采用的是 CODEC 芯片,它具有硬件压缩功能;部分采用的是 DSP+ADC (数字信号处理芯片+A/D 转换器) 方法,它利用软件的方法压缩数字音频信号。波形表合成使用 DSP 技术,它要求大容量的 ROM,以获得高质量的演奏效果。

(2) 视频卡 (Video Card): 也称为显示卡,是基于 PC 机的一种多媒体视频信号处理平台,用来支持视频信号的输入与输出。它可以采集视频源、声频源和激光视盘机、录像机、摄像机等设备的信息,经过编辑或特技等处理而产生非常精美的画面。还可以对这些画面进行捕捉、数字化、冻结、存储、压缩、输出等操作。对画面的修整、像素显示调整、缩放功能等都是视频卡支持的标准功能。视频卡可细分为视频捕捉卡、视频处理卡、视频播放卡以及 TV 编码器等专用卡,其功能是连接摄像机、VCR 影碟机、TV 等设备,以便获取、处理和表现各种动画和数字化视频媒体。

(3) 光盘驱动器: 分为只读光盘驱动器 (CD-ROM、DVD-ROM) 和可读写光盘驱动器 (CD-R, CD-RW), 可读写光盘驱动器又称刻录机。CD-ROM 及 DVD-ROM 光盘驱动器目前是多媒体应用的主要设备,主要用于存储大量的影像、声音、动画、程序数据和高分辨率图像信息。CD-ROM



可为多媒体计算机提供 650M 的存储容量,DVD-ROM 的存储量更大,双面可达 17GB。CD-ROM 驱动器价格便宜,对广大用户来说已经是必须配置的设备。CD-ROM 的速度和兼容性也是重要的技术指标。单倍速(150KB/s)是播放 CD-Audio 的最低要求,而 10 倍速的数据传输率可达 1500 KB/s。

(4) 扫描仪用于把摄影作品、绘画作品或其他印刷材料上的文字和图像,甚至实物,扫描输入到计算机中,进而对这些图像信息进行加工处理、管理、使用、存储和输出。扫描仪是获取图像的一种较简单的方法,现在已成为较流行的图像输入设备。扫描仪的种类很多,常用的有手持式扫描仪、滚筒式扫描仪、平板式扫描仪等。

① 手持式扫描仪通过手工移动扫描仪扫描图稿,一般来说,扫描宽度为 10.5,分辨率为 300~2400dpi,色彩深度为 18~24bit。其特点是体积小、重量轻、价格低,但扫描的图像容易失真。

② 滚筒式扫描仪通过软件控制扫描仪自动完成扫描过程。其扫描宽度大,可以用作大幅面图稿的输入,分辨率为 600~4800dpi,色彩深度为 24~36bit。其特点是速度快、精度高、操作简单以及连续扫描(自动进纸)。

③ 平板式扫描仪通过软件控制扫描仪自动完成扫描过程,扫描宽度为 A4 幅面,分辨率为 600~9600dpi,色彩深度为 24~36bit。其特点是速度快、精度高、操作简单但其价格比手持扫描仪高。

(5) 光学字符阅读器是一种文字自动输入的设备。它通过扫描或摄像等光学输入方法从纸介质上获取文字的图像信息,利用各种模式识别算法分析文字的形态特征,判断文字的标准编码,并按通用格式存储在文本文件中。

(6) 触摸屏是一种随多媒体技术发展而使用的输入设备。当手指点在屏幕上的菜单、图符等图形/图像按钮时,产生触摸信号,该信号经过变换后成为计算机可以处理操作命令,从而实现人机交互。

(7) 数字化仪是一种图形输入设备,它由平板加上连接的手动定位装置组成,主要用于输入地图、气象图等线型图。可以通过手动定位笔方便地获得每个线段的起始坐标,从而实现线型图输入。

(8) 操纵杆是一种提供位置信息的输入设备,可支持其他定点输入设备,将它们(数字化仪、光笔等)输入的位置和按键信息提供给应用程序。在多媒体计算机上,操纵杆常用作游戏控制器,用来操纵电子游戏。它可以以模拟信号方式工作,也可以以数字信号方式工作,并可兼容各种游戏模式。

(9) 绘图仪是一种图形输出设备;投影仪是一种将计算机输出的视频信号投影到幕布上的设备。激光视盘播放器(影碟机)是一种视频播放设备。

6.6.2 多媒体软件系统

多媒体计算机软件系统主要包括多媒体操作系统、多媒体应用软件的开发工具和多媒体应用软件。

1. 多媒体操作系统

多媒体操作系统必须具备以下的功能：对多媒体环境下的各个任务进行管理和调度。支持多媒体应用软件运行；对多媒体声像及其他多媒体信息的控制和实时处理；支持多媒体的输入/输出及相应的软件接口；对多媒体数据和多媒体设备的管理和控制以及图形用户界面管理等。也就是说它能够像一般操作系统处理文字、图形、文件那样去处理音频、图像、视频等多媒体信息；并能够对 CD-ROM 驱动器、录像机、MIDI 设备、数码相机、扫描仪等各种多媒体设备进行控制和管理。传统计算机所使用操作系统或多或少的支持多媒体，例如，Windows 3.1 在支持多媒体方面要比 DOS 强，但它们还不是真正意义上的多媒体操作系统。随着多媒体技术的发展，使得传统的操作系统增加了许多适应多媒体的内容，并研制出具备多媒体功能的操作系统。Apple 公司的 Quick Time 以及当前流行的 Windows 系列产品 Windows 98、Windows Me、Windows 2000、Windows XP 等都具备多媒体功能。

2. 多媒体创作工具软件

多媒体创作工具是在多媒体操作系统之上的系统软件，它提供了建立多媒体节目的构件和框架的功能，可以实现媒体的组接和交互跳转功能。通常，多媒体创作工具软件是用于开发多媒体应用程序的应用工具软件，帮助应用开发人员提高开发工作效率。它们大体上都是一些应用程序生成器，它将各种媒体素材按照超文本节点和链结构的形式进行组织，形成多媒体应用系统。

多媒体创作工具软件按照组织方式与数据管理大致上可分为以下几类：

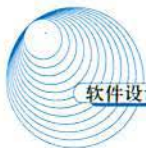
1) 页面模式的创作工具

这类创作工具按照类似于书的页面来组织和管理，具有出色的超文本和超媒体功能。这类工具主要有：

(1) Power Point：它是一种最简单实用的基于页面创作工具软件。每一个画面可以看成是一个页面，可以分别进行生成、编辑和排列。

(2) Tool Book：脚本模式的应用程序可以被想象成一本有许多页的书。每页是展示在它自己窗口中的一个画面，它包含许多多媒体对象（图形、按钮等）和大量的交互信息。页是比 Power Point 更丰富的一种结构，可以在一页之内进行交互。

2) 时序模式的创作工具



这类创作工具按照时间顺序来组织数据或事件,这种顺序的排列一般是以帧为单位。这如同电影编导过程,可以精确控制什么时间播放什么镜头。这种工具适合于处理动画、视频图像等。这类创作工具主要有:

(1) **Director**: 它是以总谱为基础,以角色和帧为对象的多媒体创作工具。总谱可以看作电影中导演脚本的形象表现。角色是指所有要单独控制的素材,包括声音、文本、图形、图像、调色板、视频、动画和按钮作为角色统一管理。每个角色都有自己的属性,可以通过多种方法进行控制。

(2) **Flash**: 它是广泛应用于网页交互多媒体动画设计工具软件,具有提供各种创建原始动画素材的方式,可将图形图像生成逐帧动画,支持多种文件格式(能导入/导出位图、视频、音频等媒体文件)和通用的浏览器,以及强大的交互功能。

3) 图标模式的创作工具

这类创作工具以对象或事件的顺序来组织数据,以流程线为主干,将各种媒体逐个组接在流线中。通常,一个应用程序结构的建立是通过从图标板中拖拽图标放入应用程序工作区间,并把它们联系起来产生一个应用程序的逻辑结构。这是一种特别适合于一般用户使用的创作方式。这类工具的典型代表是 **Authorware**。

Authorware 是用来创作与发行互动式与学习的软件开发工具众多的开发者用它来进行教育培训、教学、多媒体应用软件的开发。一个应用程序结构的建立是通过从图标板中拖拽图标放入应用程序工作区间,并把它们联系起来产生一个应用程序的逻辑结构。这是一种特别适合于一般用户使用的创作方式。支持 **ActiveX**、**Oracle Video Server**、**Flash**、媒体元素浏览器以及许多图形/图像格式(**BMP**、**DIB**、**GIF**、**JPEG**、**PhotoShop 3**、**PNG**、**TARGA** 等),并能够以这些图形/图像的原始格式来处理的压缩。

4) 窗口模式的创作工具

一个窗口是屏幕上的一个与用户交互的对象。在窗口中的所有控件和对象都是通过窗口接受控制。**Visual Basic**、**Visual C++**、**Delphi** 等编程语言都是基于程序语言的集成包,它提供了对窗口及其对象的图形创作方式。

3. 多媒体素材编辑软件

在多媒体应用中,很重要的一个环节是制作所需要的各种媒体素材。多媒体素材编辑软件用于采集、整理和编辑各种媒体数据。多媒体编辑软件主要包括:

1) 文本工具

其功能主要是文字处理(编辑、排版、识别等),常用的字处理工具有 **WPS**、**Notebook**(记事本)、**Writer**(写字板)、**Word**、**OCR**(光学字符识别)等。

2) 图形/图像工具

主要功能包括显示图形/图像、图形/图像编辑、图像压缩、图像捕捉、图形/图像素材库，常用的图形/图像处理工具有：

(1) PhotoShop: 用于图像设计、编辑与处理，其功能强大，是使用最多的一种图形/图像工具软件。

(2) Illustrator: 主要用于产品包装、网页图形、演示、标志设计、字形处理、工程绘图等。

(3) PhotoDeluxe (中文版): 主要用于数码相片处理。

(4) PageMaker (中文版): 它是一个专业排版与图形制作的工具软件。

(5) CorelDraw: 它是一种矢量图形/图像软件，广泛用于企业形象设计、广告设计和印刷设计等。

(6) AutoCAD (中文版): 它是一种矢量图形/图像软件，广泛用于机械设计、建筑设计等。

(7) Freehand: 它是一种矢量图形制作软件。

(8) 3DS Max: 它是一种功能强大，广泛使用三维图形图像编辑软件。

(9) Screen Thief: 它是支持静态抓图的一种工具，并支持 BMP、GIF、PCX、RLE 等多种文件格式。

3) 动画工具

主要包括动画显示、动画编辑、动画素材库等功能，常用的动画工具软件如：

(1) GIF Construction Set: 它是一种能够处理和创建 GIF 格式文件的功能强大的工具，能快速、专业地为网页创建透明、交错和活动 GIF 文件。

(2) Xara3D: 它是一种 3D 图形工具软件，可用于制作高质量的三维动画，全面支持中文。

4) 视频工具

主要包括显示视频、视频编辑、视频压缩、视频捕捉、视频素材库等功能。常用的动画工具软件有：

(1) Media Studio Pro (中文版): 它是一个功能强大的专业桌面数码视频编辑软件，提供一套视频捕捉、编辑、转换及特效制作等艺术解决方案。

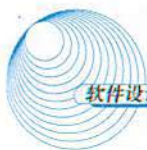
(2) Premiere: 它是一种视频编辑功能强大的视频编辑软件，提供了编辑、特技处理、剪辑等视频编辑功能，以及编辑静态图像和声音的工具。

5) 音频工具

主要包括 4 个功能：音频播放、音频编辑、音频录制、声音素材库等。常用的音频工具软件有：

(1) CoolEditPro: 它是一种功能很强的数字音频处理软件，提供了多轨编辑、数字信号处理 (DSP) 等功能，支持 WAV、MP3、AU、MPEG、MOV、AVI 等众多的音频格式。

(2) GoldWave: 它是一种小巧好用的数码录音及编辑软件，除具有许多效果处理外，还有文件格式转换功能，支持多种声音格式，如 WAV、MP3、AU、MPEG、MOV、AVI 等。



(3) **Cake Walk Pro Audio**: 它是目前流行的专业音乐制作工具软件,可以用来作曲、配器、演奏、录音、合成等,功能十分强大。

6) 播放工具

主要用于显示、浏览或播放图像、音频、视频等多媒体数据。常用的播放工具软件有:

(1) **Media Player (媒体播放器)**: Windows 操作系统内置的媒体播放器,主要用于控制多媒体设备并播放多媒体文件,如声音、音乐、动画、视频等。

(2) **ACDSee**: 它是一种图像浏览工具,支持 BMP、GIF、JPG、TGA 等各种常见的图形/图像文件格式,图片打开速度极快。

4. 多媒体应用软件

多媒体应用软件是实现用户的应用程序及演示软件,它是直接面向用户或信息发送和接收的软件。这类直接与用户接口,用户只要根据应用软件所给出的操作命令,通过最简单的操作便可使用这些软件。(根据多媒体系统终端用户要求而开发的应用软件)例如,特定的专业信息管理系统;语音/Fax/数据传输调制管理应用系统;多媒体监控系统;CD-ROM 光盘的播放软件;各种多媒体 CAI 软件;各种游戏软件等。在人们探讨应用多媒体技术解决自己面临的应用实际问题时,设计建造出各式各样的应用软件系统,使最终用户运用多媒体系统能够方便、易学、好用。除面向终端用户而制定的应用软件外,另一类是面向某一个领域的用户应用软件系统,这是面向大规模用户的系统产品,如多媒体会议系统、点播电视服务(VOD)医用、家用、军用、工业应用等。

6.7 虚拟现实的概念

虚拟现实 VR (Virtual Reality) 技术将是继多媒体、计算机网络之后,最具有应用前景的一种技术。它可应用于建模与仿真、科学计算可视化、设计与规划、教育与训练、医学、艺术与娱乐等方面。虚拟现实技术是一项综合的技术,涉及计算机科学、电子学、心理学、计算机图形学、人机接口技术、传感技术及人工智能技术等。这种技术的特点在于,运用计算机对现实世界进行全面仿真,创建与现实社会类似的环境,通过多种传感设备使用户“投入”到该环境中,实现用户与该环境直接进行自然交互。

1. 虚拟现实技术的主要特征

(1) **多感知**: 就是说除了一般计算机所具有的视觉感知外,还有听觉感知、力觉感知、触觉感知、运动感知、甚至包括味觉感知、嗅觉感知等。理想的虚拟现实就是应该具有人所具有的感知功能。

(2) 沉浸(又称临场感):是指用户感到作为主角存在于模拟环境中的真实程度。理想的模拟环境应该达到使用户难以分辨真假的程度。

(3) 交互:是指用户对模拟环境内物体的可操作程度和从环境得到反馈的自然程度(包括实时性)。例如,用户可以用手去直接抓取环境中的物体,这时手有握着东西的感觉,并可以感觉物体的重量,视场中的物体也随着手的移动而移动。

2. 关键技术

将现实世界的多维信息映射到计算机的数字空间生成相应的虚拟世界主要包括基本模型构建、空间跟踪、声音定位、视觉跟踪和视点感应等关键技术。

(1) 基本模型构建:基本模型的构建是应用计算机技术生成虚拟世界的基础,它将真实世界的对象物体在相应的三维虚拟世界中重构,并根据系统需求保存部分物理属性。模型构建首先要建立对象物体的几何模型,确定其空间位置和几何元素的属性。例如,通过 CAD/CAM 或二维图纸构建产品或建筑的三维几何模型;通过卫星、遥感或航拍照片构造大型虚拟战场。为了增强虚拟环境的真实感,物理特性和行为规则建模需表现出对象物体的质量、动量、材料等物理特性,并在虚拟环境中遵循一定的运动和动力学规律。

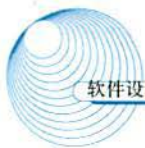
(2) 空间跟踪:虚拟环境的空间跟踪主要是通过头盔显示器、数据手套、数据衣等交互设备上的空间传感器,确定用户的头、手、躯体或其他操作物在三维虚拟环境中的位置和方向。跟踪系统一般由发射器、接收器和电子部件组成。目前的跟踪系统有电磁、机械、光学、超声等几类。例如,数据手套是 VR 系统常用的一种人机交互设备,通过手指上的弯曲、扭曲传感器和手掌上的弯度、弧度传感器,确定手及关节的位置和方向,从而实现环境中的虚拟手及其对虚拟物体的操纵。

(3) 声音跟踪:利用不同声源的声音到达某一特定地点的时间差、相位差、声压差等进行虚拟环境的声音跟踪。声音跟踪一般包括若干个发射器、接受器和控制单元。它可以与头盔显示器相连,也可以与数据衣、数据手套等其他设备相连。

(4) 视觉跟踪与视点感应:视觉跟踪技术使用从视频摄像机到 X-Y 平面阵列,周围光或者跟踪光在图像投影平面不同时刻和不同位置上的投影,计算被跟踪对象的位置和方向。视觉跟踪的实现必须考虑精度和操作范围间的折中选择,采用多发射器和多传感器的设计能增强视觉跟踪的准确性,但使系统变得复杂并且成本高。视点感应需要与显示技术相结合,采用多种定位方法(眼罩定位、头盔显示、遥视技术和基于眼肌的感应技术)可确定用户在某一时刻的视线。

3. 感知方式

在虚拟环境中获取视觉、听觉、力觉和触觉等感官认知等关键技术,是保证虚拟世界中的



事物所产生的各种刺激以尽可能自然的方式反馈给用户。

(1) 视觉感知: 虚拟环境中大部分具有一定形状的物体或现象, 可以通过多种途径使用户产生真实感很强的视觉感知。CRT 显示器、大屏幕投影、多方位电子墙、立体眼镜、头盔显示器(HMD) 等是 VR 系统中常见的显示设备。不同的头盔显示器具有不同的显示技术, 根据光学图像被提供的方式, 头盔显示设备可分为投影式和直视式。

(2) 听觉感知: 听觉是仅次于视觉的感知途径, 虚拟环境的声音效果, 可以弥补视觉效果不足, 增强环境逼真度。用户所感受的三维立体声音, 有助于用户在操作中对声音定位。

(3) 力觉和触觉感知: 能否让参与者产生“沉浸”感的关键因素之一是参与者能否在操纵虚拟物体的同时, 感受到虚拟物体的反作用力, 从而产生触觉和力觉感知。例如, 当用手扳动虚拟驾驶系统的汽车挡位杆时, 手能感觉到挡位杆的震动和松紧。力觉感知主要由计算机通过力反馈手套、力反馈操纵杆对手指产生运动阻尼从而使用户感受到作用力的方向和大小。由于人的力觉感知非常敏感, 对力反馈装置精度的要求很高。如果没有触觉反馈, 当用户接触到虚拟世界的某一物体时容易使手穿过物体。解决这种问题的有效方法是在用户的交互设备中增加触觉反馈。触觉反馈主要是基于视觉、气压感、振动触感、电子触感和神经肌肉模拟等方法来实现的。

4. 虚拟现实技术的分类

普通意义上的虚拟现实, 需要大型计算机、头盔式显示器、立体眼镜、数据手套、洞穴式投影、密封舱等一系列传感辅助设施来实现的一种三维现实, 人们通过这些设施以自然的方式(如头的转动、手的运动等) 向计算机送入各种动作信息, 并且通过视觉、听觉以及触觉设施使人们得到三维的视觉、听觉及触觉等感觉世界, 随着人们不同的动作, 这些感觉也随之改变。事实上, 随着科学技术的飞速发展, 虚拟现实技术出现了多样化的发展趋势, 虚拟实现技术不仅仅是指那些戴着头盔和手套等技术, 而且还应该包括一切与之有关的具有自然模拟、逼真体验的技术与方法, 它的根本目标就是达到真实体验和基于自然技能的人机交互, 能够达到或者部分达到这样目标的系统就称为虚拟现实系统。根据用户参与 VR 的不同形式以及沉浸的程度不同, 我们可以把各种类型的虚拟现实技术大致划分 4 类。

(1) 桌面虚拟现实: 桌面虚拟现实利用个人计算机和低级工作站进行仿真, 将计算机的屏幕作为用户观察虚拟境界的一个窗口。通过各种输入设备实现与虚拟现实世界的充分交互, 这些外部设备包括鼠标、追踪球、力矩球等。它要求参与者使用输入设备, 通过计算机屏幕观察 360 度范围内的虚拟境界, 并操纵其中的物体, 但这时参与者缺少完全的沉浸, 因为它仍然会受到周围现实环境的干扰。桌面虚拟现实最大特点是缺乏真实的现实体验, 但是成本也相对较低, 因而, 应用比较广泛。常见桌面虚拟现实技术有基于静态图像的虚拟现实 QuickTime VR、虚拟现实造型语言 VRML、桌面三维虚拟现实、MUD 等。

(2) 完全沉浸的虚拟现实:高级虚拟现实系统提供完全沉浸的体验,使用户有一种置身于虚拟境界之中的感觉。它利用头盔式显示器或其他传感设备,把参与者的视觉、听觉和其他感觉封闭起来,并提供一个新的、虚拟的感觉空间,并利用位置跟踪器、数据手套、其他手控输入设备、声音等使得参与者产生一种身临其境、全心投入和沉浸其中的感觉。常见的沉浸式系统有基于头盔式显示器的系统、投影式虚拟现实系统、远程存在系统等。

(3) 增强现实性的虚拟现实:增强现实性的虚拟现实不仅是利用虚拟现实技术来模拟现实世界、仿真现实世界,而且要利用它来增强参与者对真实环境的感受,也就是增强现实中无法感知或不方便的感受。

(4) 分布式虚拟现实:分布式虚拟现实系统是基于网络的虚拟环境,在这个环境中,位于不同物理环境位置的多个用户或多个虚拟环境通过网络相联结,或者多个用户同时参加一个虚拟现实环境,通过计算机与其他用户进行交互,并共享信息。在分布式虚拟现实系统中,多个用户可通过网络对同一虚拟世界进行观察和操作,以达到协同工作的目的。



第7章 数据库技术基础

数据库系统本质上是一个用计算机存储记录的系统。数据库管理系统是位于用户与操作系统之间的一层数据管理软件,其基本目标是提供一个可以方便地、有效地存取数据库信息的环境。数据库就是信息的集合,它是收集计算机数据的仓库或容器,系统用户可以对这些数据执行一系列操作。设计数据库系统的目的是为了管理大量信息,给用户提数据的抽象视图,即系统隐藏有关数据存储和维护的某些细节。对数据的管理涉及到信息存储结构的定义,信息操作机制的提供,安全性保证,以及多用户对数据的共享问题。

本章主要介绍一些背景知识和基本概念,使读者了解数据库的基本内容,形成数据库系统的总体框架,了解数据库系统在计算机系统中的地位,以及数据库系统的功能。

7.1 基本概念

7.1.1 数据库与数据库管理系统

数据是描述事物的符号记录,它具有多种表现形式,可以是文字、图形、图像、声音、语言等。信息是现实世界事物的存在方式或状态的反映。信息具有可感知、可存储、可加工、可传递和可再生等自然属性,信息已是社会各行各业不可缺少的资源,这也是信息的社会属性。数据是经过组织的比特的集合,而信息是具有特定释义和意义的数据。

数据库系统(DataBase System, DBS)广义上讲是由数据库、硬件、软件和人员组成,其中管理的对象是数据。数据是经过组织的比特的集合,而信息是具有特定释义和意义的数据。

(1) 数据库(DataBase, DB):是指长期储存在计算机内的、有组织的、可共享的数据集合。数据库中的数据按一定的数学模型组织、描述和储存,具有较小的冗余度、较高的数据独立性和易扩展性,并可为各种用户共享。

系统所使用的所有数据存储在一个或几个数据库中。一般尽量集成在一个数据库中,为各用户程序所使用。

(2) 硬件:构成计算机系统的各种物理设备,包括存储数据所需的外部设备。硬件的配置应满足整个数据库系统的需要。

(3) 软件:其中包括操作系统、数据库管理系统及应用程序。数据库管理系统(DataBase Management System, DBMS)是数据库系统的核心软件,要在操作系统的支持下工作,解决如何科学地组织和储存数据,如何高效地获取和维护数据的系统软件。其主要功能包括:数据定

义功能、数据操纵功能、数据库的运行管理和数据库的建立与维护。

(4) 人员：人员主要有4类。第一类为系统分析员和数据库设计人员：系统分析员负责应用系统的需求分析和规范说明，他们和用户及数据库管理员一起确定系统的硬件配置，并参与数据库系统的概要设计。数据库设计人员负责数据库中数据的确定、数据库各级模式的设计。第二类为应用程序员，负责编写使用数据库的应用程序，这些应用程序可对数据进行检索、建立、删除或修改；第三类为最终用户，他们应用系统的接口或利用查询语言访问数据库；第四类用户是数据库管理员（Data Base Administrator, DBA），负责数据库的总体信息控制。DBA的具体职责包括：决定数据库中的信息内容和结构；决定数据库的存储结构和存取策略；定义数据库的安全性要求和完整性约束条件；监控数据库的使用和运行；数据库的性能改进、数据库的重组和重构，以提高系统的性能。

7.1.2 DBMS 的功能

DBMS 主要是实现对共享数据有效的组织、管理和存取。因此，DBMS 应具有如下几个方面的功能：

1. 数据定义

DBMS 提供数据定义语言（Data Definition Language, DDL），用户可以对数据库的结构描述，包括外模式、模式和内模式的定义；数据库的完整性定义；安全保密定义，如口令、级别、存取权限等。这些定义存储在数据字典中，是 DBMS 运行的基本依据。

2. 数据库操作

DBMS 向用户提供数据操纵语言（Data Manipulation Language, DML），实现对数据库中数据的基本操作，如检索、插入、修改和删除。DML 分为两类：宿主型和自含型。所谓宿主型是指将 DML 语句嵌入某种主语言（如 C、COBOL 等）中使用；自含型是指可以单独使用 DML 语句，供用户交互使用。

3. 数据库运行管理

数据库在运行期间多用户环境下的并发控制、安全性检查和存取控制、完整性检查和执行、运行日志的组织管理、事务管理和自动恢复等是 DBMS 的重要组成部分。这些功能可以保证的数据库系统的正常运行。

4. 数据组织、存储和管理

DBMS 分类组织、存储和管理各种数据，包括数据字典、用户数据、存取路径等；要确定



以何种文件结构和存取方式在存储级上组织这些数据,以提高存取效率。实现数据间的联系、数据组织和存储的基本目标是提高存储空间的利用率。

5. 数据库的建立和维护

数据库的建立和维护包括数据库的初始建立、数据的转换、数据库的转储和恢复、数据库的重组和重构、性能监测和分析等。

6. 其他功能

如 DBMS 与网络中其他软件系统的通信功能,一个 DBMS 与另一个 DBMAS 或文件系统的数据转换功能等。

7.1.3 DBMS 的特征及分类

1. DBMS 的特征

通过 DBMS 管理数据的特点如下:

(1) 数据结构化且统一管理。数据库中的数据由 DBMS 统一管理。由于数据库系统采用复杂的数据模型表示数据结构。数据模型不仅描述数据本身的特点,还描述数据之间的联系。数据不再面向某个应用,而是面向整个应用系统。数据易维护、易扩展,数据冗余明显减少,真正实现了数据的共享。

(2) 有较高的数据独立性。数据的独立性是指数据与程序独立,将数据的定义从程序中分离出去,由 DBMS 负责数据的存储,应用程序关心的只是数据的逻辑结构,无需了解数据在磁盘上的数据库中的存储形式,从而简化应用程序,大大减少应用程序编制的工作量。数据的独立性包括:数据的物理独立性、数据的逻辑独立性。

(3) 数据控制功能。DBMS 提供了数据控制功能,以适应共享数据的环境。数据控制功能包括对数据库中数据的安全性、完整性、并发和恢复的控制。

① 数据库的安全性保护:数据库的安全性(Security)是指保护数据库以防止不合法的使用所造成的数据泄漏、更改或破坏。这样,用户只能按规定对数据进行处理,例如,划分了不同的权限,有的用户只能有读数据的权限,有的用户有修改数据的权限,用户只能在规定的权限范围内操纵数据库。

② 数据的完整性:数据库的完整性是指数据库正确性和相容性,是防止合法用户使用数据库时向数据库加入不符合语义的数据。保证数据库中数据是正确的,避免非法的更新。

③ 并发控制:在多用户共享的系统中,许多用户可能同时对同一数据进行操作。并发操作带来的问题是数据的不一致性,主要有3类:丢失更新、不可重复读和读脏数据。其主要原因是:事务的并发操作破坏了事务的隔离性。DBMS 的并发控制子系统负责协调并发事务的执

行, 保证数据库的完整性不受破坏, 避免用户得到不正确的数据。

④ 故障恢复: 数据库中的 4 类故障是事务内部故障、系统故障、介质故障及计算机病毒。故障恢复主要是指恢复数据库本身, 即在故障引起数据库当前状态不一致后, 将数据库恢复到某个正确状态或一致状态。恢复的原理非常简单, 就是要建立冗余 (Redundancy) 数据。换句话说, 确定数据库是否可恢复的方法就是其包含的每一条信息是否都可以利用冗余地存储在别处的信息重构。冗余是物理级的, 通常认为逻辑级是没有冗余的。

2. DBMS 的分类

DBMS 通常分为以下 3 类。

(1) 关系数据库系统 RDBS (Relation DataBase Systems): 是支持关系模型的数据库系统, 在关系模型中, 实体以及实体间的联系都是用关系来表示。在一个给定的现实世界领域中, 相应于所有实体及实体之间的联系的关系的集合构成一个关系数据库也有型和值之分。关系数据库的型也称为关系数据库模式, 是对关系数据库的描述, 是关系模式的集合。关系数据库的值也称为关系数据库, 是关系的集合。关系数据库模式与关系数据库通常统称为关系数据库。

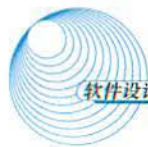
微机方式下的简单的 DBMS 系统, 包括常见的 FOXBASE、FOXPRO、DBASE、ACCESS 等, 这类 DBMS 包括网络版和单用户版, 严格地讲不能算是真正的关系型数据库, 对许多关系类型的概念并不支持, 但它却因为简单实用、价格低廉而拥有很大的用户市场。

(2) 面向对象的数据库系统 OODBS (Object-Oriented DataBase system): 是支持以对象形式对数据建模的数据库管理系统。这包括对以下功能的支持: 支持对象的类, 支持类属性的继承, 支持子类。面向对象的数据库管理系统比关系数据更优越的观念已经在一些产品中体现。一个面向对象数据库系统必须符合以下两个条件: 它必须是一个 DBMS, 它必须是面向对象的, 也就是说, 它要与现在的面向对象语言中的定义一致, 有复杂对象、对象标识、封装、对象或类、继承性、与推迟绑定结合的重载、扩展性、计算的完整性等特点。

(3) 对象关系数据库系统 ORDBS (Object-Oriented Relation DataBase system): 在传统的关系数据模型基础上, 提供元组、数组、集合一类更为丰富的数据类型以及处理新的数据类型操作的能力, 这样形成的数据模型称为“对象关系数据模型”, 基于对象关系数据模型的 DBS 称为对象关系数据库系统。

7.1.4 数据库的三级模式结构

数据库系统是数据密集型应用的核心, 其体系结构受数据库运行所在的计算机系统的影响很大, 尤其是受计算机体系结构中的联网、并行和分布的影响。站在不同的角度或不同层次上看数据库系统体系结构也不同。站在最终用户的角度看, 数据库系统体系结构分为集中式、分布式、C/S (客户/服务器) 和并行结构。站在数据库管理系统的角度看, 数据库系统体系结构



一般采用三级模式结构。

实际上数据库的产品很多,它们支持不同的数据模型,使用不同的数据库语言,建立在不同的操作系统上。数据的存储结构也各不相同,但体系结构基本上都具有相同的特征,采用“三级模式和两级映射”。如图 7-1 所示。

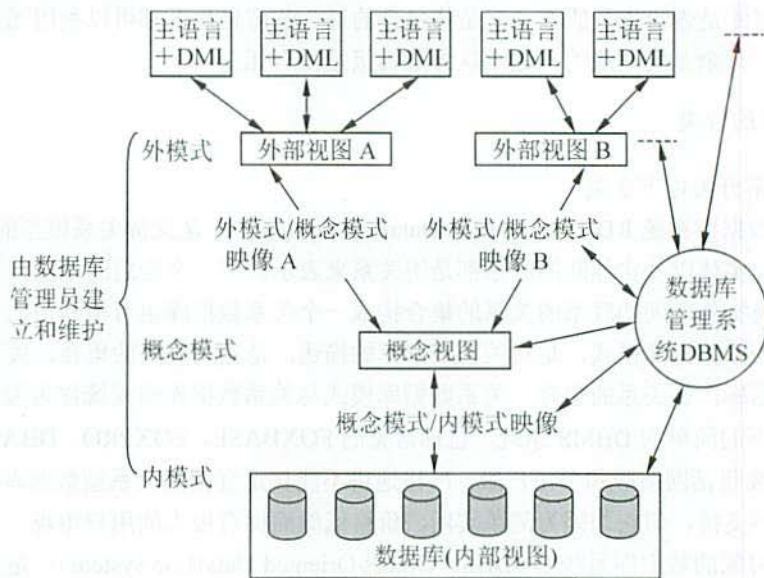


图 7-1 数据库系统体系结构

数据库系统采用三级模式结构,这是数据库管理系统内部的系统结构。数据库有“行”和“值”的概念。“行”是指对某一数据的结构和属性的说明,“值”行的一个具体赋值。

数据库系统设计员可在视图层、逻辑层和物理层对数据抽象,通过外模式、概念模式和内模式来描述不同层次上的数据特性。

1. 概念模式

概念模式也称模式,是数据库中全部数据的逻辑结构和特征的描述,它由若干个概念记录类型组成,只涉及行的描述,不涉及具体的值。概念模式的一个具体值称为模式的一个实例,同一个模式可以有很多实例。

概念模式反映的是数据库的结构及其联系,所以是相对稳定的;而实例反映的是数据库某一时刻的状态,所以是相对变动的。

需要说明的是,概念模式不仅要描述概念记录类型,还要描述记录间的联系、操作、数据的完整性、安全性等要求。但是,概念模式不涉及存储结构、访问技术等细节。只有这样,概

念模式才算做到了“物理数据独立性”。

描述概念模式的数据定义语言称为“模式 DDL” (Schema Data Definition Language)。

2. 外模式

外模式也称用户模式或子模式,是用户与数据库系统的接口,是用户用到的那部分数据的描述。它由若干个外部记录类型组成。用户使用数据操纵语言 DML (Data Manipulation Language) 对数据库进行操作,实际上是对外模式的外部记录进行操作。

描述外模式的数据定义语言称为“外模式 DDL”。有了外模式后,程序员不必关心概念模式,只与外模式发生联系,按外模式的结构存储和操纵数据。

3. 内模式

内模式也称存储模式是数据物理结构和存储方式的描述,是数据在数据库内部的表示方式。定义所有的内部记录类型、索引和文件的组织方式,以及数据控制方面的细节。

例如:记录的存储方式是顺序存储,按照 B 树结构存储,还是 Hash 方法存储;索引按照什么方式组织;数据是否压缩存储,是否加密;数据的存储记录结构有何规定。

需要说明的是,内部记录并不涉及到物理记录,也不涉及到设备的约束。比内模式更接近于物理存储和访问的那些软件机制是操作系统的一部分(即文件系统)。例如,从磁盘上读、写数据。

描述内模式的数据定义语言称为“内模式 DDL”。

总之,数据按外模式的描述提供给用户,按内模式的描述存储在磁盘上,而概念模式提供了连接这两级模式的相对稳定的中间观点,并使得两级的任意一级的改变都不受另一级的牵制。

4. 两级映像

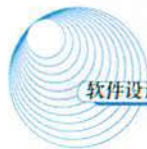
数据库系统在三级模式之间提供了两级映像:模式/内模式映像、外模式/模式映像。正因为这两级映射保证了数据库中的数据具有较高的逻辑独立性和物理独立性。

(1) 模式/内模式的映像:该映像存在于概念级和内部级之间,实现了概念模式到内模式之间的相互转换。

(2) 外模式/模式的映像:该映像存在于外部级和概念级之间,实现了外模式到概念模式之间的相互转换。

数据的独立性是指数据与程序独立,将数据的定义从程序中分离出去,由 DBMS 负责数据的存储,从而简化应用程序,大大减少应用程序编制的工作量。数据的独立性是由 DBMS 的二级映像功能来保证的。数据的独立性包括数据的物理独立性和数据的逻辑独立性。

(1) 数据的物理独立性是指当数据库的内模式发生改变时,数据的逻辑结构不变。由于应



用程序处理的只是数据的逻辑结构,这样物理独立性可以保证,当数据的物理结构改变了,应用程序不用改变。但是,为了保证应用程序能够正确执行,我们需要修改概念模式/内模式之间的映像。

(2) 数据的逻辑独立性是指用户的应用程序与数据库的逻辑结构是相互独立的。数据的逻辑结构发生变化后,用户程序也可以不修改。但是,为了保证应用程序能够正确执行,我们需要修改外模式/概念模式之间的映像。

7.2 数据模型

7.2.1 数据模型的基本概念

模型就是对现实世界特征的模拟和抽象,数据模型是对现实世界数据特征的抽象。对于具体的模型人们并不陌生,如航模飞机、地图、建筑设计沙盘都是具体的模型。从事物的客观特性到计算机里的具体表示经历了3个数据领域:现实世界、信息世界和机器世界。

(1) 现实世界。现实世界的的数据就是客观存在的各种报表、图表和查询格式等原始数据。计算机只能处理数据,所以首先要解决的问题是按用户的观点对数据和信息建模,即抽取数据库技术所研究的数据,分门别类,综合出系统所需要的数据。

(2) 信息世界。是现实世界在人们头脑中的反映,人们用符号、文字记录下来。在信息世界中,数据库常用的术语是实体、实体集、属性和码。

(3) 机器世界。是按计算机系统的观点对数据建模。换句话说,对于现实世界的问题如何表达为信息世界的问题,而信息世界的问题如何在具体的机器世界表达。机器世界中数据描述的术语有字段、记录、文件和记录码。

信息世界与机器世界相关术语的对应关系如下。

① 属性与字段:属性是描述实体某方面的特性,字段标记实体属性的命名单位。例如,用“书号、书名、作者名、出版社、日期”5个属性描述书的特性,对应有5个字段。

② 实体与记录:实体表示客观存在,并能区别的事物(如一个学生、一本书);记录是字段的有序集合,一般一条记录描述一个实体。例如“10001,DATABASE SYSTEM CONCEPTS,China Machine Press,2000-2”,描述的是一个实体,对应一条记录。

③ 码与记录码:码是能唯一区分实体的属性或属性集,记录码是唯一标识文件中的每条记录的字段或字段集。

④ 实体集与文件:实体集是具有共同特性的实体的集合,文件是同一类记录的汇集。例如所有学生构成了学生实体集,而所有学生记录组成了学生文件。

⑤ 实体型与记录型:实体型是属性的集合,如表示学生学习情况的属性的集合为实体型

(Sno, Sname, Sage, Grade, SD, Cno...)。记录型是记录的结构定义。

7.2.2 数据模型的三要素与常用的数据模型

1. 数据模型的三要素

数据库结构的基础是数据模型，是用来描述数据的一组概念和定义。数据模型的三要素是数据结构、数据操作、数据的约束条件。

(1) 数据结构：是所研究的对象类型的集合，是对系统静态特性的描述。

(2) 数据操作：对数据库中各种对象（型）的实例（值）允许执行的操作的集合，包括操作及操作规则。如操作有检索、插入、删除、修改，操作规则有优先级别等。数据操作是对系统动态特性的描述。

(3) 数据的约束条件：是一组完整性规则的集合。也就是说，对于具体的应用数据必须遵循特定的语义约束条件，以保证数据的正确、有效和相容。例如，某单位人事管理中，要求在职的“男”职工的年龄必须大于 18 岁小于 60 岁，工程师的基本工资不能低于 1500 元，每个职工可担任一个工种，这些要求是可以通过建立数据的约束条件来实现。

2. 常用的数据模型

最常用的数据模型分为概念数据模型和基本数据模型。

(1) 概念数据模型：也称信息模型，是按用户的观点对数据和信息建模，是现实世界到信息世界的第一层抽象，强调其语义表达功能，易于用户理解，是用户和数据库设计人员交流的语言，主要用于数据库设计。这类模型中最著名的是实体联系模型，简称 E-R 模型。

(2) 基本数据模型：它是按计算机系统的观点对数据建模，是现实世界数据特征的抽象，用于 DBMS 的实现。基本的数据模型有层次模型、网状模型、关系模型。

值得注意的是，目前出现了许多数据库应用的新领域，采用基本的数据模型有一定的局限性，所以面向对象模型（Object Oriented Model）越来越受到关注。

7.2.3 E-R 模型

E-R 模型是实体-联系模型的简称，所采用的 3 个主要概念是实体、联系和属性。E-R 模型是软件工程设计中的一个重要方法，因为它接近于人的思维方式，容易理解并且与计算机无关，所以用户容易接受。但是，ER 模型只能说明实体间的语义联系，还不能进一步地说明详细的数据结构。一般遇到实际问题，应先设计一个 E-R 模型，然后再把它转换成计算机能接受的数据模型。

1. 实体

实体现实世界中可以区别于其他对象的“事件”或“物体”。例如，企业中的每个人都是一



个实体。每个实体有一组特性(属性)来表示,其中的某一部分属性可以唯一标识实体,如职工号。实体集是具有相同属性的实体集合,例如,学校所有教师具有相同的属性,因此教师的集合可以定义为一个实体集;学生具有相同的属性,因此学生的集合可以定义为另一个实体集。

2. 联系

实体的联系分为实体内部的联系和实体与实体之间的联系。实体内部的联系反映数据在同一记录内部各字段间的联系。我们讨论实体集之间联系。

1) 两个不同实体之间的联系

两个不同实体集之间存在一对一、一对多和多对多的联系类型。

(1) 一对一: 指实体集 E_1 中的一个实体最多只与实体集 E_2 中的一个实体相联系。记为 $1:1$ 。

(2) 一对多: 表示实体集 E_1 中的一个实体可与实体集 E_2 中的多个实体相联系。记为 $1:n$ 。

(3) 多对多: 表示实体集 E_1 中的多个实体可与实体集 E_2 中的多个实体相联系。记为 $m:n$ 。

图 7-2 表示两个不同实体集之间的联系, 其中:

(1) 电影院里一个座位只能坐一个观众, 因此观众与座位之间是一个 $1:1$ 的联系, 联系名为“V_S”, 用 E-R 图表示如图 7-2 (a) 所示。

(2) 部门 DEPT 和职工 EMP 实体集, 若一个职工只能属于一个部门, 那么, 这两个实体集之间应是一个 $1:n$ 的联系, 联系名为“D_E”, 用 E-R 图表示如图 7-2 (b) 所示。

(3) 工程项目 PROJ 和职工 EMP 实体集, 若一个职工可以参加多个项目, 一个项目可以由多个职工参加, 那么, 这两个实体集之间应是一个 $m:n$ 的联系, 若联系名为“PR_E”, 用 E-R 图表示如图 7-2 (c) 所示。

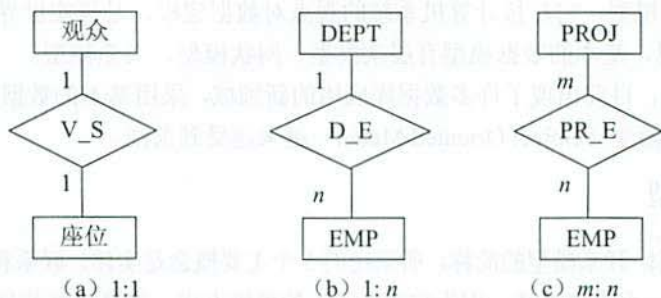


图 7-2 两个不同实体集之间的 $1:1$ 、 $1:n$ 和 $m:n$ 联系

2) 两个以上不同实体集之间的联系

两个以上不同实体集之间存在 $1:1:1$ 、 $1:1:n$ 、 $1:m:n$ 和 $r:m:n$ 的联系。例如, 图 7-5 表示了 3 个不同实体集之间的联系。其中:

(1) 图 7-3 (a) 表示供应商 Supp、项目 Proj 和零件 Part 之间的多对多 ($r:n:m$) 的联系, 联系名为“SP_P”。表示供应商为多个项目供应多种零件, 每个项目可用多个供应商供应的零件, 每种零件可由不同的供应商供应的语义。

(2) 图 7-3 (b) 表示病房、病人和医生之间的一对多 ($1:n:m$) 的联系, 联系名为“P_D”。表示一个特护病房有多个病人和多个医生, 一个医生只负责一个病房, 一个病人只属于一个病房的语义。

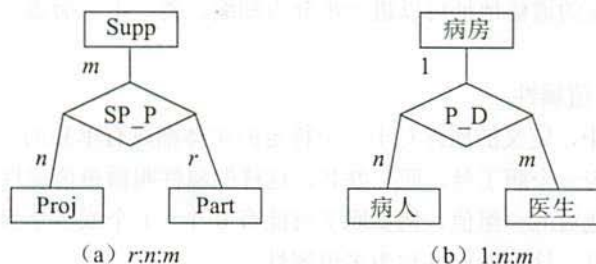


图 7-3 3 个不同实体集之间的 $r:n:m$ 和 $1:n:m$ 联系

注意：3 个实体集之间的多对多的联系和 3 个实体集两两之间的多对多的联系的语义是不同的。例如，供应商和项目实体集之间的“合同”联系，表示供应商为哪几个工程签了合同。供应商与零件两个实体集之间的“库存”联系，表示供应商库存零件的数量。项目与零件两个实体集之间的“组成”联系，表示一个项目有哪几种零件组成。

3) 同一实体集内的二元联系

同一实体集内的各实体之间也存在 $1:1$ 、 $1:n$ 和 $m:n$ 的联系，如图 7-4 所示。

从图中可见，职工实体集中的领导与被领导联系是 $1:n$ 的。但是，职工实体集中的婚姻联系是 $1:1$ 的。

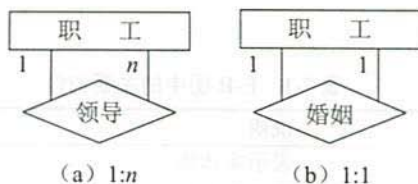


图 7-4 同一实体集之间的 $1:n$ 和 $1:1$ 联系

3. 属性

属性是实体某方面的特性。例如，职工实体集具有职工号、姓名、年龄、参加工作时间、



通信地址等属性。每个属性都有其取值范围,如职工号为 0001~9999 的 4 位整型数,姓名为 10 位的字符串,年龄的取值范围为 18~60 等。在同一实体集中,每个实体的属性及其域是相同的,但可能取不同的值。E-R 模型中的属性有如下分类。

1) 简单属性和复合属性

简单属性是原子的、不可再分的。复合属性可以细分为更小的部分(即划分为别的属性)。有时用户希望访问整个属性,有时希望访问属性的某个成分,那么在模式设计时可采用复合属性。例如,职工实体集的通信地址可以进一步分为邮编、省、市、街道。若不特别声明,我们通常指的是简单属性。

2) 单值属性和多值属性

我们所举的例子中,定义的属性对于一个特定的实体都只有单独的一个值。例如,对于一个特定的职工,只对应一个职工号、职工姓名,这样的属性叫做单值属性。但是,在某些特定情况下,一个属性可能对应一组值。例如职工可能有 0 个、1 个或多个亲属,那么职工的亲属的姓名可能有多个数目。这样的属性称为多值属性。

3) NULL 属性

当实体在某个属性上没有值或属性值未知时,使用 NULL 值。表示无意义或不知道。








4) 派生属性

派生属性可以从其他属性得来。例如,职工实体集中有“参加工作时间”和“工作年限”属性,那么“工作年限”的值可以由当前时间和参加工作时间得到。这里,“工作年限”就是一个派生属性。

4. E-R 方法

概念模型中最常用的方法为实体-联系方法,简称 E-R 方法。该方法直接从现实世界中抽象出实体和实体间的联系,然后用非常直观的 E-R 图来表示数据模型。在 E-R 图中有如表 7-1 所示的几个主要构件。

表 7-1 E-R 图中的主要构件

构件		说明
矩形		表示实体集
菱形		表示联系集
椭圆		表示属性
线段		将属性与相关的实体集连接,或将实体集与联系集相连
双椭圆		表示多值属性
虚椭圆		表示派生属性
双线		表示一个实体全部参与到联系集中

(1) 在 E-R 图中, 实体集中作为主码的一部分属性以下划线标明。另外, 在实体集与联系的线段上标上联系的类型。

(2) 在本书中, 若不引起误解, 实体集有时简称实体, 联系集有时简称联系。

【例 7.1】 学校由若干个系, 每个系有若干名教师和学生; 每个教师可以担任若干门课程, 并参加多项项目; 每个学生可以同时选修多门课程。请设计某学校的教学管理的 E-R 模型, 要求给出每个实体、联系的属性。

解: 某学校的教学管理的 E-R 模型应该有 5 个实体: 系、教师、学生、项目、课程。

(1) 设计各实体属性如下:

系 (系号, 系名, 主任名)

教师 (教师号, 教师名, 职称)

学生 (学号, 姓名, 年龄, 性别)

项目 (项目号, 名称, 负责人)

课程 (课程号, 课程名, 学分)

(2) 各实体之间的联系有: 教师担任课程的 1:n “任课” 联系; 教师参加项目的 n:m “参加” 联系; 学生选修课程的 n:m “选修” 联系; 教师、学生与系之间的所属关系的 1:n:m “领导” 联系。其中“参加”联系有一个排名属性, “选修”联系有一个成绩属性。

通过上述分析, 某学校的教学管理的 E-R 模型如图 7-5 所示。

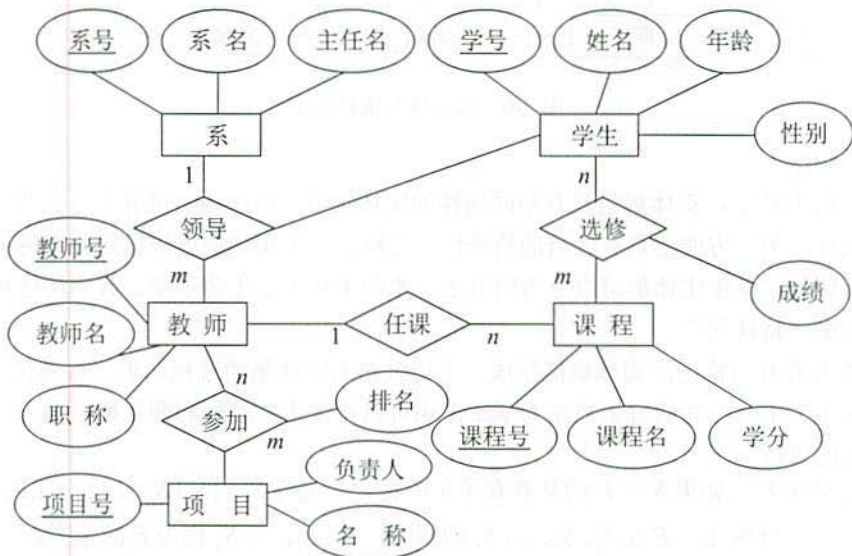


图 7-5 学校教学管理 E-R 模型

特别需要指出的是, E-R 模型强调的是语义, 与现实世界的问题密切相关。这句话的意思是, 尽管都是学校教学管理, 但由于不同的学校教学管理的方法可能会不同的语义, 因此会得到不同的 E-R 模型。在数据库设计中, 常用 E-R 模型来描述现实世界到信息世界的问题。因为 E-R 模型易于用户理解, 是用户和数据库设计人员交流的语言。

5. 扩充的 E-R 模型

尽管基本的 E-R 模型足以对大多数数据库特征建模, 但数据库某些情况下的特殊语义, 仅用基本 E-R 模型无法表达清楚。在这一节中, 将讨论扩充的 E-R 模型, 包括弱实体、特殊化、概括、聚集等概念。

1) 弱实体

在现实世界中有一种特殊的联系, 这种联系代表实体间的所有 (Ownership) 关系, 例如职工与家属的联系, 家属总是属于某职工的。这种实体对于另一些实体具有很强的依赖关系, 即一个实体的存在必须以另一个实体为前提, 我们将这类实体称为弱实体。

在扩展的 E-R 图中弱实体用双线矩形框表示。图 7-6 为职工与家属的 E-R 图。



图 7-6 弱实体与依赖联系

2) 特殊化

前面已经介绍了, 实体集是具有相同属性的实体集合。但在现实世界中, 某些实体一方面具有一些共性, 另一方面还具有各自的特殊性。这样, 一个实体集可以按照某些特征区分为几个子实体。例如, 学生实体集可以分为研究生、本科生和大专生等子集。我们将这种普遍到特殊的过程叫做“特殊化”。

将几个具有共同特性的实体集概括成一个更普遍的实体集的过程叫做“普遍化”。例如, 可以将大专生、本科生和研究生概括为学生; 还可以将学生、教师和职工概括为人。这就是从特殊到一般的过程。

设有实体集 E , 如果 S 是 E 的某些真子集的集合, 记为 $S = \{S_i | S_i \subset E, i = 1, 2, \dots, n\}$, 则称 S 是 E 的一个特殊化, E 是 S_1, S_2, \dots, S_n 的超类, S_1, S_2, \dots, S_n 称为 E 的子类。

如果 $\bigcup_{i=1}^n S_i = E$, 则称 S 是 E 的全特殊化, 否则是 E 的部分特殊化。

如果 $S_i \cap S_j = \emptyset, i \neq j$, 则 S 是不相交特殊化, 否则是重叠特殊化。

教职工实体集中的某个职工既是在职生又是教师或工人, 那么在职生、教师和工人应该是重叠特殊化; 而在在职生、教师和工人的集合等于教职工, 所以是全部特殊化。

在扩充的 E-R 模型中, 子类继承超类的所有的属性和联系, 但是, 子类还有自己特殊的属性和联系。例如, 研究生除了学习, 还要参加科研项目。那么, 研究生不仅要继承学生的所有属性, 还要增加学位类型、导师的属性, 并且需要增加与项目的联系。

在扩充的 E-R 图中超类-子类关系模型使用特殊化圆圈和连线的一般方式来表示。超类到圆圈有一条连线, 连线为双线, 则表示全特殊化, 连线为单线表示部分特殊化; 双竖边矩形框表示子类; 有符号“U”的线表示特殊化; 圆圈中的“d”表示不相交特殊化; 圆圈中的“o”表示重叠特殊化; 超类与圆圈用单线相连, 则表示部分特殊化。图 7-7 给出了一个特殊化应用实例。

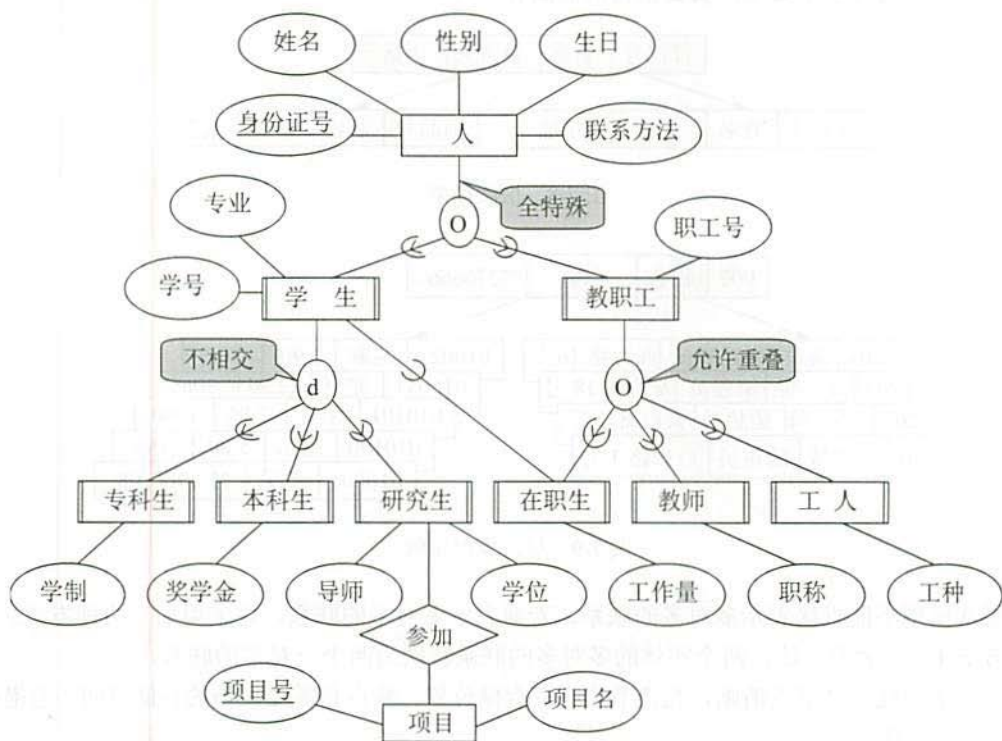
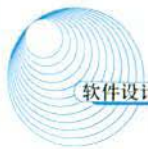


图 7-7 特殊化应用实例



7.2.4 层次模型

层次模型(Hierarchical Model)采用树型结构表示数据与数据间的联系。在层次模型中,每一个节点表示一个记录类型(实体),记录之间的联系用节点之间的连线表示,并且根节点以外的其他节点有且仅有一个双亲节点。

【例 7.2】某商场的部门、员工和商品 3 个实体的 PEP 模型如图 7-8 所示。在该模型中,每个部门有若干个员工,每个部门负责销售的商品有若干种,即该模型还表示部门到员工之间的一对多(1:n)的联系,部门到商品之间的一对多(1:n)的联系。

图 7-8 给出的只是 PEP 模型的“型”,而不是“值”。在数据库中,所谓“型”就是数据库模式,而“值”就是数据库实例。模式是数据库的逻辑设计,而数据库实例是给定时刻数据库中数据的一个快照。图 7-9 表示销售部的一个实例。该实例表示在某一时刻销售部是由李军负责,销售部下属有 4 个员工,负责销售的商品有 5 种。

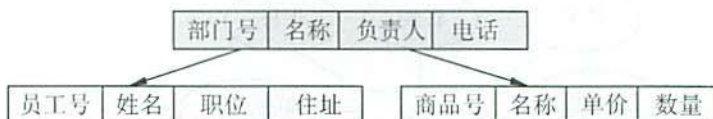


图 7-8 层次模型

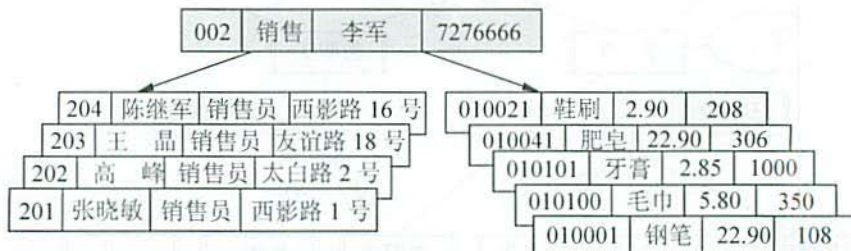


图 7-9 层次模型实例

层次模型不能直接表示多对多的联系。若要表示多对多的联系,可采用如下两种方法。

方法 1: 冗余节点法。两个实体的多对多的联系转换为两个一对多的联系。

该方法的优点是节点清晰,允许节点改变存储位置。缺点是需要额外的存储空间,有潜在的数据不一致性。

方法 2: 采用虚拟节点分解法。将冗余节点转换为虚拟节点。虚拟节点是一个指引元,指向所代替的节点。该方法的优点是减少对存储空间的浪费,避免数据不一致性。缺点是改变存储位置可能引起虚拟节点中指针的修改。

【例 7.3】员工与商品，员工可以销售多种商品，一种商品可以由多个员工销售，所以员工与商品之间是一个多对多的联系，如图 7-10 所示。采用冗余节点法将其转换为两个一对多的联系如图 7-11 所示。采用虚拟节点分解法，将冗余节点转换为虚拟节点如图 7-12 所示。



图 7-10 员工与商品的多对多联系



图 7-11 多对多联系转换为冗余节点表示



图 7-12 多对多联系转换为虚拟节点表示

层次模型的特点是：记录之间的联系通过指针实现，比较简单，查询效率高。

层次模型的缺点是：只能表示 $1:n$ 的联系，尽管有许多辅助手段实现 $m:n$ 的联系，但较复杂不易掌握；由于层次顺序严格和复杂，插入删除操作的限制比较多，导致应用程序编制比较复杂。1968 年美国 IBM 公司推出的 IMS 系统（信息管理系统）是典型的层次模型系统，20 世纪 70 年代在商业上得到了广泛的应用。

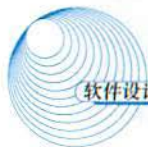
7.2.5 网状模型

采用网络结构表示数据与数据间的联系的数据模型称为网状模型（Network Model）。在网状模型中，允许一个以上的节点无双亲，一个节点可以有多个的双亲。

网状模型（也称 DBTG 模型）是一个比层次模型更普遍性的数据结构，是层次模型的一个特例。网状模型可以直接地描述现实世界。因为，第一去掉了层次模型的两个限制，第二，允许两个节点之间有多种联系（称之为复合联系）。

网状模型中的每个节点表示一个记录类型（实体），每个记录类型可以包含若干个字段（实体的属性），节点间的连线表示记录类型之间一对多的联系。层次模型和网状模型的主要区别如下：

- （1）网状模型中子女节点与双亲节点的联系不唯一，因此需要为每个联系命名。
- （2）网状模型允许复合链，即两个节点之间有两种以上的联系，如图 7-13（a）所示。
- （3）网状模型不能表示记录之间的多对多的联系，需要引入联结记录来表示多对多的联系。



系。如图 7-13 (b) 所示。

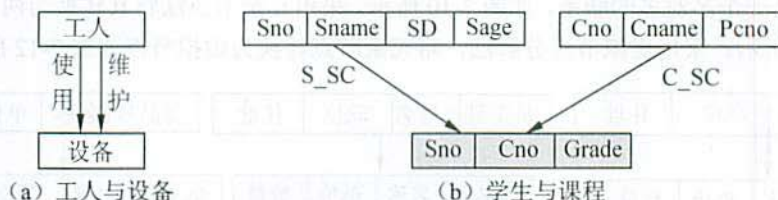


图 7-13 网状模型举例

【例 7.4】 学生、课程以及他们之间的多对多的联系不能直接用网状模型表示。因为一个学生可以选若干门课，而一门课可以被多个学生选。为此，引入选课联结记录，如图 7-13 (b) 所示。这样，学生与选课之间的 S-SC 是一对多的联系，课程与选课之间的 C-SC 也是一对多的联系。

图 7-13 中，Sno、Sname、SD、Sage、Cno、Cname、Pcno 和 Grade 分别表示学号、姓名、系、年龄、课程号、课程名、选修课程号和成绩。

通常，网状数据模型没有层次模型那样严格的完整性约束条件，但 DBTG 在模式 DDL 中提供了定义 DBTG 数据库完整性的若干概念和语句，主要有：

- (1) 支持记录码的概念。记录码能唯一标识记录的数据项的集合。
- (2) 保证一个联系中双亲记录和子女记录之间是一对多的联系。
- (3) 以支持双亲记录和子女记录之间的某些约束条件。例如图 7-13 (b) 中，当插入一条选课记录“010014, 100, 98”时，只有学生实体中存在学号为“010014”学生记录，课程实体存在课程号，系统才认为是合法的操作。

网状模型的主要优点是：能更为直接地描述现实世界，具有良好的性能，存取效率高。

网状模型的主要缺点是：结构复杂。例如，当应用环境不断扩大时，数据库结构就变得很复杂，不利于最终用户掌握。编制应用程序难度比较大。DBTG 模型的 DDL、DML 语言复杂，记录之间的联系是通过存取路径来实现的，因此程序员必须了解系统结构的细节，增加了编写应用程序的负担。

7.2.6 关系模型

关系模型 (Relation Model) 是目前最常用的数据模型之一。关系数据库系统采用关系模型作为数据的组织方式，在关系模型中用表格结构表达实体集，以及实体集之间的联系，其最大特色是描述的一致性。关系模型是由若干个关系模式组成的集合。一个关系模式相当于一个记录型，对应于程序设计语言中类型定义的概念。关系是一个实例，也是一张表，对应于程序设计语言中的变量的概念。给定变量的值随时间可能发生变化；类似地，当关系被更新时，关系

实例的内容也随时间发生了变化。

【例 7.5】 教学数据库的 4 个关系模式如下：

S (Sno, Sname, SD, Sage, Sex) : 学生 S 关系模式, 属性为学号、姓名、系、年龄和性别。

T (Tno, Tname, Age, Sex) : 教师 T 关系模式, 属性为教师号、姓名、年龄和性别。

C (Cno, Cname, Pcnno) : 课程 C 关系模式, 属性为课程号、课程名和选修课程号。

SC (Sno, Cno, Grade) : 学生选课 SC 关系模式, 属性为学号、课程号和成绩。

关系模式中有下划线的属性是主码属性。图 7-14 是教学模型的一个具体的实例。

S 学生关系					T 教师关系			
Sno	Sname	SD	Age	Sex	Tno	Tname	Age	Sex
01001	贾皓昕	IS	20	男	001	方铭	34	女
01002	姚勇	IS	20	男	002	章雨敬	58	男
03001	李晓红	CS	19	女	003	王平	48	女

SC 选课			C 课程关系		
Sno	Cno	Grade	Cno	Cname	Pcno
01001	C001	90	C001	MS	
01001	C002	91	C002	IC	
01002	C001	95	C003	C++	C002
01002	C003	89	C004	OS	C002
03001	C001	91	C005	DBMS	C004

图 7-14 关系模型的实例

关系模型与网状模型、层次模型最大的差别是用主码而不是用指针导航数据, 表格简单、直观易懂, 用户只需要简单地查询语句就可以对数据库进行操作, 无需涉及存储结构和访问技术等细节。关系模型的优点是概念单一, 存储路径对用户是透明的, 所以具有更好的数据独立性和安全保密性, 简化了程序的开发和数据库的建立工作。正因为这一点, 关系模型已经成为许多商用处理应用中的主要数据模型。

7.3 关系代数

7.3.1 关系数据库的基本概念

1. 属性和域

在现实世界中, 要描述一个事物常常取若干特征来表示。这些特征称为属性 (attribute)。



例如学生学号、姓名、性别、系别、年龄、籍贯等属性来描述。每个属性的取值范围所对应一个值的集合,称为该属性的域(Domain)。例如,学号的域是6位整型数;姓名的域是10位字符;性别的域为{男,女};……。

一般在关系数据模型中,对域还加了一个限制,所有的域都应是原子数据(atomic data)。例如,整数、字符串是原子数据,而集合、记录、数组是非原子数据。关系数据模型的这种限制称为第一范式(First Normal Form,简称1NF)条件。但也有些关系数据模型突破了1NF的限制,称为非1NF的。

2. 笛卡儿积与关系

定义 7.1 设 $D_1, D_2, D_3, \dots, D_n$ 为任意集合,定义 $D_1, D_2, D_3, \dots, D_n$ 的笛卡儿积为:

$$D_1 \times D_2 \times D_3 \times \dots \times D_n = \{(d_1, d_2, d_3, \dots, d_n) \mid d_i \in D_i, i = 1, 2, 3, \dots, n\}$$

其中每一个元素 $(d_1, d_2, d_3, \dots, d_n)$ 叫做一个 n 元组 (n -tuple 属性的个数),元组的每一个值 d_i 叫做元组一个分量,若 $D_i (i = 1, 2, 3, \dots, n)$ 为有限集,其基数(Cardinal number 元组的个数)

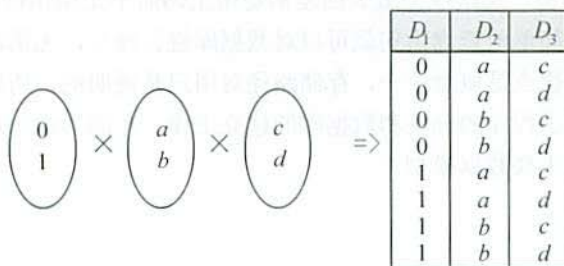
为 $m_i (i = 1, 2, 3, \dots, n)$,则 $D_1 \times D_2 \times D_3 \times \dots \times D_n$ 的基数 M 为: $M = \prod_{i=1}^n m_i$ 笛卡儿积可以用二维表来表示。

【例 7.6】 若 $D_1 = \{0, 1\}$, $D_2 = \{a, b\}$, $D_3 = \{c, d\}$, 求 $D_1 \times D_2 \times D_3$ 。

解: 根据定义,笛卡儿积中的每一个元素应该是一个三元组,每个分量来自不同的域,因此结果为:

$$D_1 \times D_2 \times D_3 = \{(0, a, c), (0, a, d), (0, b, c), (0, b, d), (1, a, c), (1, a, d), (1, b, c), (1, b, d)\}$$

用二维表表示如图 7-15 所示。



D_1	D_2	D_3
0	a	c
0	a	d
0	b	c
0	b	d
1	a	c
1	a	d
1	b	c
1	b	d

图 7-15 $D_1 \times D_2 \times D_3$ 笛卡儿积的二维表表示

定义 7.2 $D_1 \times D_2 \times D_3 \times \dots \times D_n$ 的子集叫做在域 $D_1, D_2, D_3, \dots, D_n$ 上的关系,记为 $R (D_1, D_2, D_3, \dots, D_n)$,称关系 R 为 n 元关系。

定义 7.2 可以得出一个关系也可以用二维表来表示。关系中属性的个数称为“元数”,元组

的个数称为“基数”。关系模型中的术语与一般术语的对应情况可以通过图 7-16 中的学生关系说明。图中学生关系模式可表示为：学生 (S_no, Sname, SD, Sex)。该学生关系的主码为 S_no，属性分别为 S_no、Sname、SD 和 Sex，对属性 Sex 的域为男、女，等等。而该学生关系的元数为 4，基数为 6。

属性 1	属性 2	属性 3	属性 4	关系模型术语	一般术语
S_no	Sname	SD	Sex	属性	字段、数据项
100101	张军生	通信	男	元组 1	记录 1
100102	黎晓华	通信	男	元组 2	记录 2
100103	赵敏	通信	女	元组 3	记录 3
200101	李斌斌	电子工程	男	元组 4	记录 4
300102	王莉娜	计算机	女	元组 5	记录 5
300103	吴晓明	计算机	男	元组 6	记录 6

图 7-16 学生关系与术语的对应情况

3. 关系的相关名词

(1) 目或度 (Degree)：这里的 R 表示关系的名字， n 是关系的目或度。

(2) 候选码 (Candidate Key)：若关系中的某一属性或属性组的值能唯一的标识一个元组，则称该属性或属性组为候选码。

(3) 主码 (Primary Key)：若一个关系有多个候选码，则选定其中一个为主码。

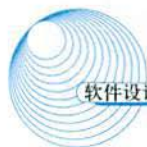
(4) 主属性 (Non-Key attribute)：包含在任何候选码中的诸属性称为主属性。不包含在任何候选码中的属性称为非码属性。

(5) 外码 (Foreign key)：如果关系模式 R 中的属性或属性组非该关系的码，但它是其他关系的码，那么该属性集对关系模式 R 而言是外码。

例如，客户与贷款之间的借贷联系 $c-l$ ($c-id, loan-no$)，属性 $c-id$ 是客户关系中的码，所以 $c-id$ 是外码；属性 $loan-no$ 是贷款关系中的码，所以 $loan-no$ 也是外码。

(6) 全码 (All-key)：关系模型的所有属性组是这个关系模式的候选码，称为全码。

例如，关系模式 $R(T, C, S)$ ，属性 T 表示教师，属性 C 表示课程，属性 S 表示学生。假设一个教师可以讲授多门课程，某门课程可以由多个教师讲授，学生可以听不同教师讲授的不同课程，那么，要想区分关系中的每一个元组，这个关系模式 R 的码应为全属性 T 、 C 和 S ，即 All-key。



4. 关系的3种类型

(1) 基本关系(通常又称为基本表或基表):是实际存在的表,它是实际存储数据的逻辑表示。

(2) 查询表:查询结果对应的表。

(3) 视图表:是由基本表或其他视图表导出的表。由于它本身不独立存储在数据库中,数据库中只存放它的定义,所以常称为虚表。

5. 关系数据库模式

在数据库中要区分型和值。关系数据库中的型也称为关系数据库模式,是关系数据库结构的描述。它包括若干域的定义以及在这些域上定义的若干关系模式。实际上,关系的概念对应于程序设计语言中变量的概念,而关系模式对应于程序设计语言中类型定义的概念。关系数据库的值是这些关系模式在某一时刻对应的关系的集合,通常称之为关系数据库。

定义 7.3 关系的描述称为关系模式(Relation Schema)。可以形式化地表示为:

$$R(U, D, dom, F)$$

其中, R 表示关系名; U 是组成该关系的属性名集合; D 是属性的域; dom 是属性向域的映像集合; F 为属性间数据的依赖关系集合。

通常将关系模式简记为:

$$R(U) \text{ 或 } R(A_1, A_2, A_3, \dots, A_n)$$

其中 R 为关系名, $A_1, A_2, A_3, \dots, A_n$ 为属性名或域名,属性的向域的映像常常直接说明属性的类型、长度。通常在关系模式主属性上加下划线表示该属性为主码属性。

例如:学生关系 S 有学号 Sno 、学生姓名 $Sname$ 、系名 SD 、年龄 SA 属性;课程关系 C 有课程号 Cno 、课程名 $Cname$ 、先修课程号 $PCno$ 属性;学生选课关系 SC 有学号 Sno 、课程号 Cno 、成绩 $Grade$ 属性。定义关系模式及主码如下(本题未考虑 F 属性间数据的依赖,该问题在后续内容讨论)。

(1) 学生关系模式 $S(\underline{Sno}, Sname, SD, SA)$ 。

(2) 课程关系模式 $C(\underline{Cno}, Cname, PCno)$ $Dom(PCno) = Cno$ 。这里, $PCno$ 是选修课程号,来自 Cno 域,但由于 $PCno$ 属性名不等于 Cno 值域名,所以要用 Dom 来定义。但是,不能将 $PCno$ 直接改为 Cno ,因为在关系模型中,各列属性必须取相异的名字。

(3) 学生选课关系模式 $SC(\underline{Sno}, \underline{Cno}, Grade)$ 。 SC 关系中的 Sno 、 Cno 又分别为外码。因为它们分别是 S 、 C 关系中的主码。

6. 完整性约束

完整性规则提供了一种手段来保证当授权用户对数据库作修改时不会破坏数据的一致性。

因此,完整性规则防止的是对数据的意外破坏。关系模型的完整性规则是对关系的某种约束条件。关系的完整性共分为 3 类:实体完整性、参照完整性(也称引用完整性)、用户定义完整性。

(1) 实体的完整性(Entity Integrity)规定基本关系 R 的主属性 A 不能取空值。

(2) 参照的完整性(Referential Integrity)现实世界中的实体之间往往存在某种联系,在关系模型中实体及实体间的联系是用关系来描述的,这样自然就存在着关系与关系间的引用。

例如,员工和部门关系模式表示如下,其中在关系模式主属性上加下划线表示该属性为主码属性。

员工(员工号, 姓名, 性别, 参加工作时间, 部门号)

部门(部门号, 名称, 电话, 负责人)

这两个关系存在着属性的引用,即员工关系中的“部门号”值必须是确实存在的部门的部门号,即部门关系中有该部门的记录。也就是说,员工关系中的“部门号”属性取值要参照部门关系的“部门号”属性取值。

参照完整性规定,若 F 是基本关系 R 的外码,它与基本关系 S 的主码 K_s 相对应(基本关系 R 和 S 不一定是不同的关系)则对于 R 中每个元组在 F 上的值必须为:或者取空值(F 的每个属性值均为空值);或者等于 S 中某个元组的主码值。

(3) 用户定义的完整性(User defined Integrity)就是针对某一具体的关系数据库的约束条件,反映某一具体应用所涉及的数据必须满足的语义要求,由应用的环境决定。例如,银行的用户账户规定必须大于等于 100000,小于 999999。

7. 关系运算

关系操作的特点是操作对象和操作结果都是集合。而非关系数据模型的数据操作方式则为一次一个记录的方式。关系数据语言分为 3 类:关系代数语言、关系演算语言和具有关系代数和关系演算双重特点的语言(如 SQL)。关系演算语言包含元组关系演算语言(如 Alpha, Quel)和域关系演算语言(如 QBE)。

关系代数语言、元组关系演算和域关系演算是抽象查询语言,它与具体的 DBMS 中实现的实际语言并不一样,但是可以用它评估实际系统中的查询语言能力的标准。

关系代数运算符有 4 类:集合运算符,专门的关系运算符,算术比较符和逻辑运算符。根据运算符的不同,关系代数运算可分为传统的集合运算和专门的关系运算。传统的集合运算是从关系的水平方向进行的,包括并、交、差及广义笛卡儿积。专门的关系运算既可以从关系的水平方向进行运算,又可以向关系的垂直方向运算,包括选择、投影、连接以及除法。如表 7-2 所示。表 7-2 中并、差、笛卡儿积、投影、选择是 5 种基本的运算,因为其他运算可以通过基本的运算导出。

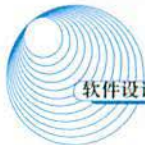


表 7-2 关系代数运算符

运算符			含义		
集合	\cup	并	比较	$>$	大于
运算	$-$	差	运算	\geq	大于等于
符	\cap	交	符	$<$	小于
	\times	笛卡儿积		\leq	小于等于
				$=$	等于
				\neq	不等于
专门的	σ	选择	逻辑	\neg	非
关符运	π	投影	运算	\wedge	与
算符	\bowtie	连接	符	\vee	或
	\div	除			

7.3.2 5 种基本的关系代数运算

5 种基本的关系代数运算包括并、差、笛卡尔积、投影、选择,其他运算可以通过基本的关系运算导出。

1. 并 (Union)

关系 R 与 S 具有相同的模式,即 R 与 S 的元数相同(结构相同)。关系 R 与 S 并由属于 R 或属于 S 的元组构成的集合组成,记作 $R \cup S$,其形式定义如下:

$$R \cup S = \{t \mid t \in R \vee t \in S\}$$

式中 t 为元组变量。

2. 差 (Difference)

关系 R 与 S 具有相同的模式,关系 R 与 S 的差由属于 R 但不属于 S 的元组构成的集合,记作 $R - S$,其形式定义如下:

$$R - S = \{t \mid t \in R \wedge t \notin S\}$$

3. 广义笛卡儿积 (Extended Cartesian Product)

两个元数分别为 n 目和 m 目的关系 R 和 S 的广义笛卡儿积是一个 $(n+m)$ 列的元组的集合。元组的前 n 列是关系 R 的一个元组,后 m 列是关系 S 的一个元组。记作 $R \times S$,其形式定义如下:

$$R \times S = \{t \mid t = \langle t^n, t^m \rangle \wedge t^n \in R \wedge t^m \in S\}$$

如果 R 和 S 中有相同的属性名, 可在属性名前加关系名作为限定, 以示区别。若 R 有 K_1 个元组, S 有 K_2 个元组。则 R 和 S 的广义笛卡儿积有 $K_1 \times K_2$ 个元组。

注意: 在本教材中的序偶 $\langle t'', t'' \rangle$ 解释为元组 t'' 和 t'' 拼接成的一个元组。

4. 投影 (Projection)

投影运算是从关系的垂直方向进行运算, 在关系 R 中选择出若干属性列 A 组成新的关系, 记作 $\pi_A(R)$, 其形式定义如下:

$$\pi_A(R) = \{t[A] \mid t \in R\}$$

5. 选择 (Selection)

选择运算是从关系的水平方向进行运算, 是从关系 R 中选择满足给定条件的诸元组, 记作 $\sigma_F(R)$, 其形式定义如下:

$$\sigma_F(R) = \{t \mid t \in R \wedge F(t) = \text{True}\}$$

其中, F 中的运算对象是属性名 (或列的序号) 或常数, 运算符算术比较符 ($<$, \leq , $>$, \geq , $=$, \neq) 和逻辑运算符 (\wedge , \vee , \neg)。例如, $\sigma_{1>6}(R)$ 表示选取 R 关系中第一个属性值大于等于第六个属性值的元组; $\sigma_{1>6'}(R)$ 表示选取 R 关系中第一个属性值大于等于 '6' 的元组。

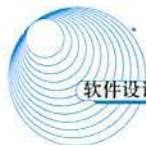
【例 7.7】设有关系 R 、 S 如图 7-17 所示, 请求出: $R \cup S$, $R - S$, $R \times S$, $\pi_{A,C}(R)$, $\sigma_{A>B}(R)$, $\sigma_{3<4}(R \times S)$ 。

解: $R \cup S$, $R - S$, $R \times S$, $\pi_{A,C}(R)$, $\sigma_{A>B}(R)$, $\sigma_{3<4}(R \times S)$ 结果如图 7-18 所示。

其中, $R \times S$ 后生成的关系属性名有重复, 按照关系“属性不能重名”的性质, 通常采用“关系名.属性名”的格式。对于 $\sigma_{3<4}(R \times S)$ 的含义是 $R \times S$ 后“选取第三个属性值小于第四个属性值”的元组。由于 $R \times S$ 的第三个属性为 $R.C$, 第四个属性是 $S.A$, 因此 $\sigma_{3<4}(R \times S)$ 的含义也是 $R \times S$ 后“选取 $R.C$ 值小于 $S.A$ 值”的元组。

R 关系			S 关系		
A	B	C	A	B	C
a	b	c	b	a	d
b	a	d	d	f	g
c	d	e	f	h	k
d	f	g			

图 7-17 关系 R 和 S



$R \cup S$		
A	B	C
a	b	c
b	a	d
c	d	e
d	f	g
f	h	k

$R - S$		
A	B	C
a	b	c
c	d	e

$\pi_{A,C}(R)$	
A	C
a	c
b	d
c	e
d	g

$\sigma_{A>B}(R)$		
A	B	C
B	a	d

$R \times S$					
R.A	R.B	R.C	S.A	S.B	S.C
a	b	c	b	a	d
a	b	c	d	f	g
a	b	c	f	h	k
b	a	d	b	a	d
b	a	d	d	f	g
b	a	d	f	h	k
c	d	e	b	a	d
c	d	e	d	f	g
c	d	e	f	h	k
d	f	g	b	a	d
d	f	g	d	f	g
d	f	g	f	h	k

$\sigma_{3<4}(R \times S)$					
R.A	R.B	R.C	S.A	S.B	S.C
a	b	c	d	f	g
a	b	c	f	h	k
b	a	d	f	h	k
c	d	e	f	h	k

图 7-18 运算结果

7.3.3 扩展的关系代数运算

扩展的关系代数运算可以从基本的关系运算中导出。主要包括选择、投影、连接、除法、广义笛卡儿积、外连接。

1. 交 (Intersection)

关系 R 与 S 具有相同的模式，关系 R 与 S 的交由属于 R 同时又属于 S 的元组构成的集合，关系 R 与 S 的交记作 $R \cap S$ ，其形式定义如下：

$$R \cap S = \{t | t \in R \wedge t \in S\}$$

显然， $R \cap S = R - (R - S)$ ，或者 $R \cap S = S - (S - R)$ 。

2. 连接 (Join)

连接分为 θ 连接、等值连接及自然连接 3 种。连接运算是从两个关系 R 和 S 的笛卡儿积中选取满足条件的元组。因此，可以认为笛卡儿积是无条件连接，其他的连接操作认为是有条件连接。下面分述如下。

(1) θ 连接：从 R 与 S 的笛卡儿积中选取属性间满足一定条件的元组。记作：

$$R \bowtie_{X\theta Y} S = \{t | t = \langle t^n, t^m \rangle \wedge t^n \in R \wedge t^m \in S \wedge t^n[X] \theta t^m[Y]\}$$

其中, ' $X\theta Y$ '为连接的条件, θ 是比较运算符, X 和 Y 分别为 R 和 S 上度数相等, 且可比的属性组。 $t^n[X]$ 表示 R 中 t^n 元组的相应于属性 X 的一个分量。 $t^m[Y]$ 表示 S 中 t^m 元组的相应于属性 Y 的一个分量。需要说明的是, θ 连接也可以表示为:

$$R \bowtie_{i\theta j} S = \{t | t = \langle t^n, t^m \rangle \wedge t^n \in R \wedge t^m \in S \wedge t^n[i] \theta t^m[j]\}$$

其中: $i=1, 2, 3, \dots, n$, $j=1, 2, 3, \dots, m$, ' $i\theta j$ '的含义为从两个关系 R 和 S 中选取 R 的第 i 列和 S 的第 j 列之间满足 θ 运算的元组进行连接。

θ 连接可以由基本的关系运算笛卡儿积和选取运算导出。因此 θ 连接可表示为:

$$R \bowtie_{X\theta Y} S = \sigma_{X\theta Y}(R \times S) \text{ 或 } R \bowtie_{i\theta j} S = \sigma_{i\theta(j)}(R \times S)$$

【例 7.8】 设有关系 R 、 S 如图 7-17 所示, 求 $R \bowtie_{R.A < S.B} S$ 。

$R.A < S.B$

解: 本题连接的条件为 $R.A < S.B$, 意为将 R 关系中属性 A 的值小于 S 关系中属性 B 的值的元组取出来作为结果集的元组。结果集为 $R \times S$ 后选出满足条件的元组, 并且结果集的属性为: $R.A, R.B, R.C, S.A, S.B, S.C$ 。结果如图 7-19 所示。

$R.A$	$R.B$	$R.C$	$S.A$	$S.B$	$S.C$
a	b	c	d	f	g
a	b	c	f	h	k
b	a	d	d	f	g
b	a	d	f	h	k
c	d	e	d	f	g
c	d	e	f	h	k
d	f	g	d	f	g
d	f	g	f	h	k

图 7-19 $R \bowtie_{R.A < S.B} S$

$R.A < S.B$

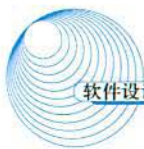
(2) 等值连接: 当 θ 为“=”时, 称之为等值连接, 记为 $R \bowtie_{X=Y} S$, 其形式定义如下:

$X=Y$

$$R \bowtie_{X=Y} S = \{t | t = \langle t^n, t^m \rangle \wedge t^n \in R \wedge t^m \in S \wedge t^n[X] = t^m[Y]\}$$

$X=Y$

(3) 自然连接: 是一种特殊的等值连接, 它要求两个关系中进行比较的分量必须是相同的属性组, 并且在结果集中将重复属性列去掉。若 t^n 表示 R 关系的元组变量, t^m 表示 S 关系的元组变量; R 和 S 具有相同的属性组 B , 且 $B = (B_1, B_2, \dots, B_K)$; 并假定 R 关系的属性为 $A_1, A_2, \dots, A_{n-k}, B_1, B_2, \dots, B_K$, S 关系的属性为 $B_1, B_2, \dots, B_K, B_{K+1}, B_{K+2}, \dots, B_m$; 为 S 的元组变量



t^m 去掉重复属性 B 所组成的新的元组变量为 t^m 。自然连接可以记为 $R \bowtie S$ ，其形式定义如下：

$$R \bowtie S = \left\{ t \mid t = \langle t^n, t^m \rangle \wedge t^n \in R \wedge t^m \in S \wedge R.B_1 = S.B_1 \wedge R.B_2 = S.B_2 \wedge \dots \wedge R.B_n = S.B_n \right\}$$

自然连接可以由基本的关系运算笛卡儿积和选取运算导出，因此自然连接可表示为：

$$R \bowtie S = \prod_{A_1, A_2, \dots, A_{n-k}, R.B_1, R.B_2, \dots, R.B_K, B_{K+1}, B_{K+2}, \dots, B_m} (\sigma_{R.B_1=S.B_1 \wedge R.B_2=S.B_2 \wedge \dots \wedge R.B_k=S.B_k} (R \times S))$$

特别需要说明的是，一般连接是从关系的水平方向运算，而自然连接不仅要关系的水平方向，而且要从关系的垂直方向运算。因为自然连接要去掉重复属性，如果没有重复属性，那么自然连接就转化为笛卡儿积。

【例 7.9】 设有关系 R 、 S 如图 7-20 所示，求： $R \bowtie S$ 。

解：本题要求 R 与 S 关系的自然连接，自然连接是一种特殊的等值连接，它要求两个关系中进行比较的分量必须是相同的属性组，并且在结果中将重复属性列去掉。本题 R 与 S 关系中相同的属性组为 AC ，因此，结果集中的属性列应为： $ABCD$ 。其结果如图 7-21 所示。

A	B	C
a	b	c
b	a	d
c	d	e
d	f	g

(a) 关系 R

A	C	D
a	c	d
d	f	g
b	d	g

(b) 关系 S

图 7-20 关系 R 、 S

A	B	C	D
a	b	c	d
b	a	d	g

图 7-21 $R \bowtie S$

3. 除 (Division)

除运算是同时从关系的水平方向和垂直方向进行运算。给定关系 $R(X, Y)$ 和 $S(Y, Z)$ ， X 、 Y 、 Z 为属性组。 $R \div S$ 应当满足元组在 X 上的分量值 x 的象集 Y_x 包含关系 S 在属性组 Y 上投影的集合。其形式定义如下：

$$R \div S = \{ t^n[X] \mid t^n \in R \wedge \pi_Y(S) \subseteq Y_x \}$$

其中： Y_x 为 x 在 R 中的象集， $x = t^n[X]$ 。且 $R \div S$ 的结果集的属性组为 X 。

【例 7.10】 设有关系 R 、 S 如图 7-22 所示，求： $R \div S$ 。

解：根据除法定义，此题的 X 为属性 AB ， Y 为属性 CD 。 $R \div S$ 应当满足元组在属性 AB 上的分量值 x 的象集 Y_x 包含关系 S 在 CD 上投影的集合。

关系 S 在 Y 上的投影为 $\pi_{CD}(S) = \{(c, d), (e, f)\}$ 。对于关系 R ，属性组 X (即 AB) 可以取 3 个值 $\{(a, b), (b, d), (c, k)\}$ ，它们的象集分别为：

A	B	C	D
a	b	c	d
a	b	e	f
a	b	h	k
b	d	e	f
b	d	d	l
c	k	c	d
c	k	e	f

(a) R

C	D
c	d
e	f

(b) S

A	B
a	b
c	k

(c) $R \div S$

图 7-22 $R \div S$

象集 $CD_{(a,b)} = \{(c,d), (e,f), (h,k)\}$

象集 $CD_{(b,d)} = \{(e,f), (d,l)\}$

象集 $CD_{(c,k)} = \{(c,d), (e,f)\}$

由于上述象集包含 $\pi_{CD}(S)$ 有 (a,b) 和 (c,k) , 所以, $R \div S = \{(a,b), (c,k)\}$, 结果如图 7-22

(c) 所示。

【例 7.11】 设学生课程数据库中有: 学生 S、课程 C、学生选课 SC 3 个关系, 如图 7-23 所示。请用关系代数表达式表达如下检索问题。

Sno	Sname	Sex	SD	Age
3001	王 平	女	计算机	18
3002	张 勇	男	计算机	19
4003	黎 明	女	机 械	18
4004	刘明远	男	机 械	19
1041	赵国庆	男	通 信	20
1042	樊建玺	男	通 信	20

S

Cno	Cname	Pcno	Credit
1	数 据 库	3	3
2	数 学		4
3	操作系统	4	4
4	数据结构	7	3
5	数字通信	6	3
6	信息系统	1	4
7	程序设计	2	2

C

Sno	Cno	Grade
3001	1	93
3001	2	84
3001	3	84
3002	2	83
3002	3	93
1042	1	84
1042	2	82

SC

图 7-23 S、C、SC 关系

- ① 检索选修课程名为“数学”的学生号和学生姓名。
- ② 检索至少选修了课程号为“1”和“3”的学生号。
- ③ 检索选修了“操作系统”或“数据库”课程的学号和成绩。
- ④ 检索年龄在18到20之间(含18和20)的女生的学号、姓名及年龄。
- ⑤ 检索选修了“刘平”老师所讲课程的学生的学号、姓名及成绩。
- ⑥ 检索选修全部课程的学生姓名。
- ⑦ 检索选修课程包括“1042”学生所学的课程的学生学号。
- ⑧ 检索不选修“2”课程的学生姓名和所在系。

解: ① 检索选修课程名为“数学”的学生号和学生姓名的关系代数表达式如下:

$$\pi_{Sno, Sname}(\sigma_{Cname='数学'}(S \bowtie SC \bowtie C)) \text{ 或 } \pi_{1,2}(\sigma_{8='数学'}(S \bowtie SC \bowtie C))$$

对于上述表达式 $S \bowtie SC \bowtie C$ 自然连接后重复的属性列为学号 Sno 和课程号 Cno, 去掉重复属性列的结果如图 7-24 所示。从图中可见, 满足课程名为“数学”的只有三个元组, 对 Sno 和 Cno 投影的结果如图 7-25 所示。由于 Sno、Cno 和 Cname 分别对应第 1、2 和 8 列属性, 所以上述表达式还可以写为: $\pi_{1,2}(\sigma_{8='数学'}(S \bowtie SC \bowtie C))$ 。

Sno	Sname	Sex	SD	Age	Cno	Grade	Cname	Pcno	Credit
3001	王 平	女	计算机	18	1	93	数 据 库	3	3
3001	王 平	女	计算机	18	2	84	数 学		4
3001	王 平	女	计算机	18	3	84	操作系统	4	4
3002	张 勇	男	计算机	19	2	83	数 学		4
3002	张 勇	男	计算机	19	3	93	操作系统	4	4
1042	樊建玺	男	通 信	20	1	84	数 据 库	3	3
1042	樊建玺	男	通 信	20	2	82	数 学		4

图 7-24 $S \bowtie SC \bowtie C$

② 检索至少选修了课程号为“1”和“3”的学生号可有如下两种解题思路。

第一种: 关系代数表达式为 $\pi_1(\sigma_{1=4 \wedge 2=1' \wedge 5=3'}(SC \times SC))$ 。若设 $SC \times SC$ 中的第一个 SC 关系为 S1, 与第二个 SC 关系为 S2, 那么, 该关系表达式的含义为先从 $SC \times SC$ 中选取满足条件 $S1.Sno = S2.Sno$ 、 $S1.Cno = 1'$ 、 $S2.Sno = 3'$ 的元组, 最后投影第一个属性列 Sno 即为所求结果集。

第二种: 关系代数表达式为 $\pi_{Sno, Cno}(SC) \div \pi_{Cno}(\sigma_{Cno=1' \vee Cno=3'}(C))$, 分析如下:

表达式 $\pi_{Cno}(\sigma_{Cno=1' \vee Cno=3'}(C))$ 就是构造一个临时关系 K, 其属性为 Cno, 结果如下:

$$K = \pi_{Cno}(\sigma_{Cno=1' \vee Cno=3'}(C)) = \{1, 3\}$$

Sno	Sname
3001	王 平
3002	张 勇
1042	樊建玺

图 7-25 对 Sno 和 Cno 的投影结果

查询表达式 $\pi_{Sno, Cno}(SC) \div K$ 的结果集的学生号所选的课程号应包括 K 。

求解的过程就是对 $\pi_{Sno, Cno}(SC)$ 的每一个元组逐一求某一学生的象集。因为所求的 Sno ，所以 X 为 Sno ， Y 为 Cno ，所以象集为 Cno_{Sno} 。将 Sno 的值逐一代入求象集：

$$Cno_{3001} = \{1, 2, 3\}$$

$$Cno_{3002} = \{2, 3\}$$

$$Cno_{1042} = \{1, 2\}$$

上可以看出，只有 3001 包含了 K 在 Cno 的投影，所以， $\pi_{Sno, Cno}(SC) \div K = \{3001\}$ 。

③ 检索选修了“操作系统”或“数据库”课程的学号和姓名的关系代数表达式为：

$$\pi_{Sno, Sname}(S \bowtie (\sigma_{Cname='操作系统' \vee Cname='数据库'}(SC \bowtie C)))$$

④ 检索年龄在 18~20 之间（含 18 和 20）的女生的学号、姓名及年龄的关系代数表达式为：

$$\pi_{Sno, Sname, Age}(\sigma_{Age \leq '18' \wedge Age \geq '20'}(S))$$

⑤ 检索选修了“数据库”课程的学生的学号、姓名及成绩的关系代数表达式为：

$$\pi_{Sno, Sname, Grade}(\sigma_{Cname='数据库'}(S \bowtie SC \bowtie C))$$

⑥ 检索选修全部课程的学生姓名及所在系的关系代数表达式为：

$$\pi_{Sname, SD}(S \bowtie (\pi_{Sno, Cno}(SC) \div \pi_{Cno}(C)))$$

对于本题给出的具体关系，求解过程分析如下：

- 表示全部课程的临时关系 $K = \pi_{Cno}(C) = \{1, 2, 3, 4, 5, 6, 7\}$ ；
 - 查询选修了所有课程的学生号为 $\pi_{Sno, Cno}(SC) \div K = \{\phi\}$ ，因为学生所选课程分别为：
 $Cno_{3001} = \{1, 2, 3\}$ ， $Cno_{3002} = \{2, 3\}$ 和 $Cno_{1042} = \{1, 2\}$ ，所以 3001、3002 和 1042 都没有包含 K ，故结果集为空。
 - 与 S 关系进行自然连接在对学生姓名 $Sname$ 和学号 SD 投影的结果也为空。
- ⑦ 检索选修课程包含“1042”学生所学的课程的学生学号的关系表达式如下：

$$\pi_{Sno, Cno}(SC) \div \pi_{Cno}(\sigma_{Sno='1042'}(SC))$$

⑧ 检索不选修“2”课程的学生姓名和所在系的关系代数表达式为：

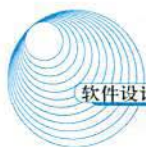
$$\pi_{Sname, SD}(SC) - \pi_{Sname, SD}(\sigma_{Cno='2'}(S \times SC))$$

对于上式也可以用属性列号替换属性名，将上式写成如下等价的关系代数表达式：

$$\pi_{2,4}(SC) - \pi_{2,4}(\sigma_{6='2'}(S \times SC))$$

4. 广义投影 (Generalized Projection)

广义投影运算允许在投影列表中使用算术运算，实现了对投影运算的扩充。



若有关系 R , 条件 F_1, F_2, \dots, F_n 中的每一个都是涉及 R 中常量和属性的算术表达式, 那么广义投影运算的形式定义如下:

$$\pi_{F_1, F_2, \dots, F_n}(R)$$

【例 7.12】 信贷额度关系模式 credit-in ($C_name, limit, Credit_balance$), 属性分别表示用户姓名、信贷额度和到目前为止的花费。图 7-26 (a) 表示了关系 credit-in 的一个具体。若要查询给出每个用户还能花费多少可以用关系代数表达式: $\pi_{C_name, limit - credit_balance}(credit - in)$ 来表示, 查询结果如图 7-26 (b) 所示。

C_name	limit	Credit_balance
王伟峰	2500	1800
吴 楨	3100	2000
黎建明	2380	2100
刘 柯	5600	3600
徐国平	8100	5800
景莉红	6000	4500

(a) credit-in

C_name	Limit - Credit_balance
王伟峰	700
吴 楨	1100
黎建明	280
刘 柯	2000
徐国平	2300
景莉红	1500

(b) $\pi_{C_name, limit - credit_balance}(credit - in)$

图 7-26 信贷额度关系

5. 外连接 (Outer Join)

外连接运算是连接运算的扩展, 可以处理缺失的信息。对于图 7-23 的 S 和 SC 关系, 当我们对其进行自然连接 $S \bowtie SC$ 时, 其结果如图 7-27 所示。

Sno	Sname	Sex	SD	Age	Cno	Grade
3001	王 平	女	计算机	18	1	93
3001	王 平	女	计算机	18	2	84
3001	王 平	女	计算机	18	3	84
3002	张 勇	男	计算机	19	2	83
3002	张 勇	男	计算机	19	3	93
1042	樊建玺	男	通 信	20	1	84
1042	樊建玺	男	通 信	20	2	82

图 7-27 $S \bowtie SC$

从图 7-27 可以看出 S 与 SC 的自然连接 $S \bowtie SC$ 的结果丢失了黎明、刘明远、赵国庆相关信息。可是, 使用外连接我们可以避免这样的信息丢失。外连接运算有 3 种: 左外连接、右外连接和全外连接。

1) 左外连接 (Left Outer Join) $\bowtie\leftarrow$

左外连接取出左侧关系中所有与右侧关系中任一元组都不匹配的元组,用空值 Null 充填所有来自右侧关系的属性,构成新的元组,将其加入自然连接的结果中。对于图 7-23 的 S 和 SC 关系,当我们对其进行左外连接 $S \bowtie_{\text{左}} SC$ 时,其结果如图 7-28 所示。

Sno	Sname	Sex	SD	Age	Cno	Grade
3001	王 平	女	计算机	18	1	93
3001	王 平	女	计算机	18	2	84
3001	王 平	女	计算机	18	3	84
3002	张 勇	男	计算机	19	2	83
3002	张 勇	男	计算机	19	3	93
4003	黎 明	女	机 械	18	Null	Null
4004	刘明远	男	机 械	19	Null	Null
1041	赵国庆	男	通 信	20	Null	Null
1042	樊建玺	男	通 信	20	1	84
1042	樊建玺	男	通 信	20	2	82

图 7-28 $S \bowtie_{\text{左}} SC$

2) 右外连接 (Right Outer Join) $\bowtie_{\text{右}}$

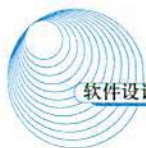
取出右侧关系中所有与左侧关系中任一元组都不匹配的元组,用空值 Null 填充所有来自左侧关系的属性,构成新的元组,将其加入自然连接的结果中。对于图 7-23 的 S 和 SC 关系,当我们对其进行右外连接 $SC \bowtie_{\text{右}} S$ 时,其结果如图 7-29 所示。

Sno	Cno	Grade	Cname	Pcno	Credit
3001	1	93	数 据 库	3	3
3001	2	84	数 学		4
3001	3	84	操作系统	4	4
3002	2	83	数 学		4
3002	3	93	操作系统	4	4
1042	1	84	数 据 库	3	3
1042	2	82	数 学		4
Null	4	Null	数据结构	7	3
Null	5	Null	数字通信	6	3
Null	6	Null	信息系统	1	4
Null	7	Null	程序设计	2	2

图 7-29 $SC \bowtie_{\text{右}} S$

3) 全外连接 (Full Outer Join) $\bowtie_{\text{全}}$

完成左外连接和右外连接的操作。即填充左侧关系中所有与右侧关系中任一元组都不匹配的元组,又填充右侧关系中所有与左侧关系中任一元组都不匹配的元组,将产生的新元组加入自然连接的结果中。



【例 7.13】设有关系 R 、 S 如图 7-30 所示, 求: $R \bowtie S, R \ltimes S, R \ltimes S$ 。

对于图 7-30 的 R 、 S 关系, 当我们对其进行左外连接 $R \bowtie S$, 右外连接 $R \ltimes S$, 全外连接 $R \ltimes S$ 时, 其结果分别如图 7-31 (a)、(b) 和 (c) 所示。

A	B	C
a	b	c
b	a	d
c	d	e
d	f	g

(a) 关系 R

B	C	D
b	c	d
d	e	g
f	d	g
d	e	c

(b) 关系 S

图 7-30 关系 R 、 S

A	B	C	D
a	b	c	d
c	d	e	g
c	d	e	c
b	a	d	Null
d	f	g	Null

(a) 左外连接 $R \bowtie S$

A	B	C	D
a	b	c	d
c	d	e	g
c	d	e	c
Null	f	d	g

(b) 右外连接 $R \ltimes S$

A	B	C	D
a	b	c	d
c	d	e	g
c	d	e	c
b	a	d	Null
d	f	g	Null
Null	f	d	g

(c) 全外连接 $R \ltimes S$

图 7-31 $R \bowtie S, R \ltimes S, R \ltimes S$

7.4 关系数据库 SQL 语言简介

关系数据库具有坚实的理论基础, 这一理论有助于关系数据库的设计和用户对数据库信息需求的有效处理。它涉及的内容有: 关系模型的基本知识、关系数据库的标准语言 SQL、查询优化以及关系数据理论。对象-关系数据模型扩展关系数据模型的方式是通过提供一个包括复杂数据类型和面向对象的更丰富的类型系统。关系查询语言(特别在 SQL 中)需要做相应扩展以处理这些更丰富的类型系统。这种扩展试图在扩展建模能力的同时保留关系的基础——特别是对数据的说明性存取。对象-关系数据库系统(即以对象-关系模型为基础的数据库系统)为想要使用面向对象特征的关系数据库用户提供了一个方便的迁移途径。

7.4.1 SQL 数据库体系结构

SQL (Structured Query Language) 是在关系数据库中最普遍使用的语言它不仅包含数据查

询功能,还包括插入、删除、更新和数据定义功能。和自然语言的方言一样,存在许多不同类型的 SQL。目前,主要有 3 个标准:ANSI(美国国家标准机构)SQL;对 ANSI SQL 进行修改后在 1992 年采用的标准称之为 SQL-92 或 SQL2;最近的 SQL-99 标准也称 SQL3 标准。SQL-99 从 SQL-92 扩充而来并增加了对对象关系特征和许多其他新的功能。其次,各个厂家提供不同版本的 SQL。这些版本不仅都包含原始的 ANSI 标准,而且在很大程度上支持 SQL-92 标准,并在 SQL-92 的基础上做了修改和扩展,包括部分 SQL-99 的标准。

尽管人们习惯性地称 SQL 是一个“查询语言”,但实际上它的功能远非查询信息这么简单。主要包括数据查询(Query)、数据操纵(Manipulation)、数据定义(Definition)和数据控制(Control)功能,是一种通用的、功能强大的关系数据库语言。

1. SQL 的特点

(1) 综合统一:非关系模型的数据语言分为模式定义语言和数据操纵语言,其缺点是,当要修改模式时,必须停止现有数据库的运行、转储数据、修改模式编译后再重装数据库。SQL 是集数据定义、数据操纵和数据控制功能于一体,语言风格统一,可独立完成数据库生命周期的所有活动。

(2) 高度非过程化:非关系数据模型的数据操纵语言是面向过程的,若要完成某项请求时,必须指定存储路径;而 SQL 语言是高度非过程化语言,当进行数据操作时,只要指出“做什么”,无需指出“怎么做”,存储路径对用户来说是透明的,提高了数据的独立性。

(3) 面向集合的操作方式:非关系数据模型采用的是面向记录的操作方式,操作对象是一条记录。而 SQL 语言采用面向集合的操作方式,其操作对象、查找结果可以是元组的集合。

(4) 两种使用方式:第一种方式,用户可以在终端键盘上键入 SQL 命令,对数据库进行操作,故称之为自含式语言。第二种方式,将 SQL 语言嵌入到高级语言程序中,所以又是嵌入式语言。

(5) 语言简洁、易学易用:SQL 语言功能极强,完成核心功能只用了 9 个动词,包括如下 4 类:

- 数据查询:SELECT。
- 数据定义:CREATE、DROP、ALTER。
- 数据操纵:INSERT、UPDATE、DELETE。
- 数据控制:GRANT、REVOKE。

2. SQL 支持三级模式结构

SQL 语言支持关系数据库的三级模式结构,其中,视图对应外模式、基本表对应模式、存储文件对应内模式。具体结构如图 7-32 所示。

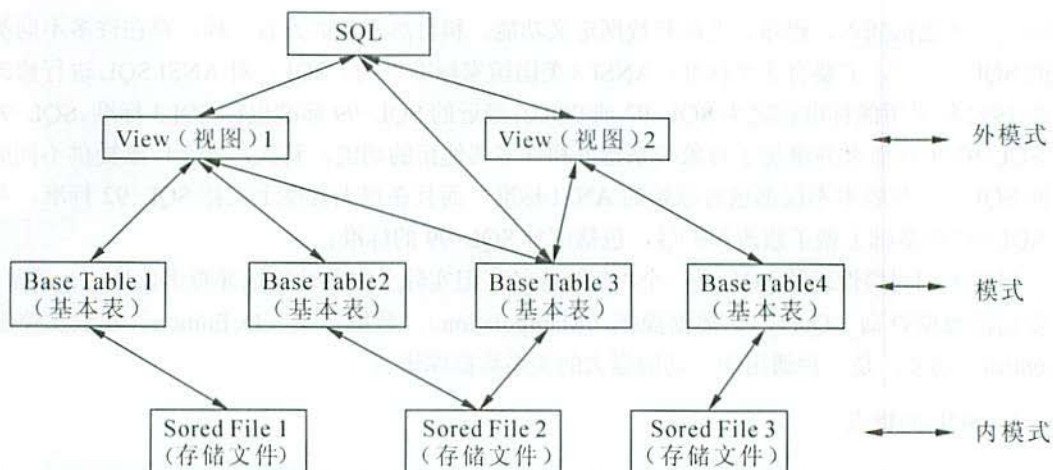
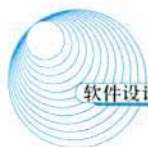


图 7-32 关系数据库的三级模式结构

7.4.2 SQL 的基本组成

SQL 由以下几个部分组成:

(1) 数据定义语言 (DDL): SQL DDL 提供定义关系模式和视图、删除关系和视图、修改关系模式的命令。

(2) 交互式数据操纵语言 (DML): SQL DML 提供查询、插入、删除和修改的命令。

(3) 事务控制 (Transaction Control): SQL 提供定义事务开始和结束的命令。

(4) 嵌入式 SQL 和动态 SQL (Embedded SQL and Dynamic SQL) 用于嵌入到某种通用的高级语言 (C、C++、Java、PL/I、Cobol、VB 等等) 中混合编程。其中 SQL 负责操纵数据库, 高级语言负责控制程序流程。

(5) 完整性 (Integrity): SQL DDL 包括定义数据库中的数据必须满足的完整性约束条件的命令, 对于破坏完整性约束条件的更新将被禁止。

(6) 权限管理 (Authorization): SQL DDL 中包括说明对关系和视图的访问权限。

本章主要介绍基本的 DDL 和 DML、嵌入式 SQL 和动态 SQL。后续内容将介绍数据库互连技术, 如 ODBC 等等。

7.4.3 SQL 数据定义

1. 创建表 (CREATE TABLE)

语句格式:

```
CREATE TABLE <表名> (<列名><数据类型>[列级完整性约束条件]
    [, <列名><数据类型>[列级完整性约束条件]]...
    [, <表级完整性约束条件>]);
```

列级完整性约束条件有：NULL（空）、UNIQUE（取值唯一），如 NOT NULL UNIQUE 表示取值唯一，不能取空值。

【例 7.14】 建立一个供应商、零件数据库。其中“供应商”表 S（Sno, Sname, Status, City）分别表示为供应商代码、供应商名、供应商状态、供应商所在城市。“零件”表 P（Pno, Pname, Color, Weight, City）表示零件号、零件名、颜色、重量及产地。其中，数据库要满足如下要求：

- （1）供应商代码不能为空，且值是唯一的，供应商的名也是唯一的。
- （2）零件号不能为空，且值是唯一的；零件名不能为空。
- （3）一个供应商可以供应多个零件，而一个零件可以由多个供应商供应。

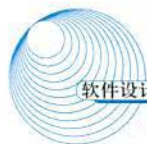
分析：根据题意供应商和零件分别要建立一个关系模式。供应商和零件之间是一个多对多的联系，在关系数据库中，多对多联系必须生成一个关系模式，而该模式的码是该联系两端实体的码加上联系的属性构成的，若该联系名为 SP，那么关系模式为 SP（Sno, Pno, Qty），其中，Qty 表示零件的数量。

根据上述分析，用 SQL 建立一个供应商、零件数据库如下：

```
CREATE TABLE S( Sno   CHAR(5) NOT NULL UNIQUE,
    Sname  CHAR(30) UNIQUE,
    Status CHAR(8),
    City   CHAR(20)
    PRIMARY KEY(Sno));

CREATE TABLE P( Pno   CHAR(6),
    Pname  CHAR(30) NOT NULL,
    Color  CHAR(8),
    Weight NUMERIC(6, 2),
    City   CHAR(20)
    PRIMARY KEY(Pno));

CREATE TABLE SP( Sno   CHAR(5),
    Pno   CHAR(6),
    Status CHAR(8),
    Qty   NUMERIC(9),
    PRIMARY KEY(Sno, Pno),
    FOREIGN KEY(Sno) REFERENCES S(Sno),
    FOREIGN KEY(Pno) REFERENCES P(Pno));
```

从上述定义可以看出,“Sno CHAR(5) NOT NULL UNIQUE”语句定义了 Sno 的列级完整性约束条件,取值唯一,不能取空值。

需要说明的是:PRIMARY KEY(Sno)已经定义了 Sno 为主码,所以,“Sno CHAR(5) NOT NULL UNIQUE”语句中的“NOT NULL UNIQUE”可以省略。另外,FOREIGN KEY(Sno) REFERENCES S(Sno)定义了 在 SP 关系中 Sno 为外码其取值必须来自 S 关系的 Sno 域。同理,在 SP 关系中 Pno 也定义为外码。

2. 修改表和删除表

(1) 修改表 (ALTER TABLE)。语句格式:

```
ALTER TABLE <表名>[ADD<新列名><数据类型>[完整性约束条件]]  
[DROP<完整性约束名>]  
[MODIFY <列名><数据类型>];
```

例如,向“供应商”表 S 增加 Zap“邮政编码”可用如下语句:

```
ALTER TABLE S ADD Zap CHAR(6);
```

注意,不论基本表中原来是否已有数据,新增加的列一律为空。

又如,将 Status 字段改为整型可用如下信息:

```
ALTER TABLE S MODIFY Status INT;
```

(2) 删除表 (DROP TABLE)。语句格式:

```
DROP TABLE <表名>
```

例如,执行 DROP TABLE Student; 此后关系 Student 不再是数据库模式的一部分,关系中的元组也无法访问。

3. 定义和删除索引

数据库中的索引与书籍中的索引类似,在一本书中,利用索引可以快速查找所需信息,无须阅读整本书。在数据库中,索引使数据库程序无须对整个表进行扫描,就可以在其中找到所需数据。书中的索引是一个词语列表,其中注明了包含各个词的页码。而数据库中的索引是某个表中一列或者若干列值的集合和相应的指向表中物理标识这些值的数据页的逻辑指针清单。

索引分为:聚集索引和非聚集索引。聚集索引是指索引表中索引项的顺序与表中记录的物理顺序一致的索引。

(1) 建立索引。语句格式:

CREATE [UNIQUE][CLUSTER]INDEX <索引名>
ON <表名> (<列名>[<次序>][, <列名>[<次序>]]...);

参数说明:

- 次序: 可选 ASC (升序) 或 DSC (降序), 默认值为 ASC。
- UNIQUE: 表明此索引的每一个索引值只对应唯一的数据记录。
- CLUSTER: 表明要建立的索引是聚簇索引, 意为索引项的顺序是与表中记录的物理顺序一致的索引组织。

【例 7.15】 假设供应销售数据库中有供应商 S、零件 P、工程项目 J、供销情况 SPJ 关系, 希望建立 4 个索引。其中: 供应商 S 中 Sno 按升序建立索引; 零件 P 中 Pno 按升序建立索引; 工程项目 J 中 Jno 按升序建立索引; 对供销情况 SPJ 中 Sno 按升序, Pno 按降序, Jno 按升序建立索引。

解: 根据题意建立的索引如下:

```
CREATE UNIQUE INDEX S-SNO ON S(Sno);
CREATE UNIQUE INDEX P-PNO ON P(Pno);
CREATE UNIQUE INDEX J-JNO ON J(Jno);
CREATE UNIQUE INDEX SPJ-NO ON SPJ(Sno ASC,Pno DESC,JNO ASC);
```

(2) 删除索引。语句格式:

DROP INDEX <索引名>

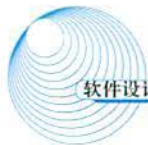
例如, 执行 DROP INDEX StudentIndex; 此后索引 StudentIndex 不再是数据库模式的一部分。

4. 定义、删除、更新视图

视图是从一个或者多个表或视图中导出的表, 其结构和数据是建立在对表的查询基础上的。和真实的表一样, 视图也包括几个被定义的数据列和多个数据行, 但从本质上讲, 这些数据列和数据行来源于其所引用的表。因此, 视图不是真实存在的基础表而是一个虚拟表, 视图所对应的数据并不实际地以视图结构存储在数据库中, 而是存储在视图所引用的表中。

(1) 使用视图的优点和作用。

- ① 可以使视图集中数据、简化和定制不同用户对数据库的不同数据要求。
- ② 使用视图可以屏蔽数据的复杂性, 用户不必了解数据库的结构, 就可以方便地使用和管理数据, 简化数据权限管理和重新组织数据以便输出到其他应用程序中。
- ③ 视图可以使用户只关心他感兴趣的某些特定数据和他们所负责的特定任务, 而那些不需要的或者无用的数据则不在视图中显示。



- ④ 视图大大地简化了用户对数据的操作。
- ⑤ 视图可以让不同的用户以不同的方式看到不同或者相同的数据集。
- ⑥ 在某些情况下,由于表中数据量太大,因此在表的设计时常将表进行水平或者垂直分割,但表的结构的变化会对应用程序产生不良的影响。
- ⑦ 视图提供了一个简单而有效的安全机制。

(2) 视图的创建。语句格式:

CREATE VIEW 视图名 (列表名)

AS SELECT 查询子句

[WITH CHECK OPTION];

视图的创建中,必须遵循如下规定:

- ① 子查询可以是任意复杂的 SELECT 语句,但通常不允许含有 order by 子句和 DISTINCT 短语。
- ② WITH CHECK OPTION 表示对 UPDATE,INSERT,DELETE 操作时保证更新、插入、或删除的行满足视图定义中的谓词条件(即子查询中的条件表达式)。
- ③ 组成视图的属性列名或者全部省略或者全部指定。如果省略属性列名,则隐含该视图由 SELECT 子查询目标列的主属性组成。

【例 7.16】 建立“计算机系”(CS 表示计算机系)学生的视图,并要求进行修改、插入操作时保证该视图只有计算机系的学生。

CREATE VIEW CS-STUDENT

AS SELECT Sno,Sname,Sage,Sex

FROM Student

WHERE SD='CS'

WITH CHECK OPTION;

由于 CS-STUDENT 视图使用了“WITH CHECK OPTION”子句,因此,对该视图进行修改、插入操作时 DBMS 会自动加上 SD='CS'的条件,保证该视图只有计算机系的学生。

(3) 视图的撤销。语句格式:

DROP VIEW 视图名

例如, DROP VIEW CS-STUDENT 将删除视图 CS-STUDENT。

7.4.4 SQL 数据查询

SQL 的数据操纵功能包括 SELECT (查询)、INSERT (插入)、DELETE (删除)和 UPDATE

(修改) 4 条语句。SQL 语言对数据库的操作十分灵活方便, 原因在于 SELECT 语句中的成分丰富多样的元组, 有许多可选形式, 尤其是目标列和条件表达式。

1. Select 基本结构

数据库查询是数据库的核心操作, SQL 语言提供了 SELECT 语句进行数据库的查询。

语句格式:

```
SELECT [ALL|DISTINCT]<目标列表达式>[,<目标列表达式>]...
FROM <表名或视图名>[,<表名或视图名>]
[WHERE <条件表达式>]
[GROUP BY <列名 1>[HAVING<条件表达式>]]
[ORDER BY <列名 2>[ASC|DESC]...]
```

SQL 查询中的子句顺序: SELECT、FROM、WHERE、GROUP BY、HAVING 和 ORDER BY。但是 SELECT、FROM 是必须的, 而且, HAVING 子句只能与 GROUP BY 搭配起来使用。

(1) SELECT 子句对应的是关系代数中的投影运算, 用来列出查询结果中的属性。其输出可以是列名、表达式、集函数 (AVG、COUNT、MAX、MIN、SUM), DISTINCT 选项可以保证查询的结果集中不存在重复元组。

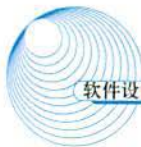
(2) FROM 子句对应的是关系代数中的笛卡儿积, 它列出的是表达式求值过程中需扫描的关系即在 FROM 子句中出现多个基本表或视图时, 系统首先执行笛卡儿积操作。

(3) WHERE 子句对应的是关系代数中的选择谓词。WHERE 子句的条件表达式中可以使用的运算符如表 7-3 所示。

表 7-3 WHERE 子句的条件表达式中可以使用的运算符

运算符		含义	运算符		含义
集合成员运算符	IN	在集合中	算术运算符	>	大于
	NOT IN	不在集合中		≥	大于等于
字符串匹配运算符	LIKE	与_和%进行单个 多个字符匹配		<	小于
				≤	小于等于
空值比较运算符	IS NULL	为空		=	等于
	IS NOT NULL	不能为空		≠	不等于
			逻辑运算符	AND	与
				OR	或
				NOT	非

一个典型的 SQL 查询具有如下形式:



```
select  A1, A2, ..., An
      from  r1, r2, ..., rm
      where  p
```

对应关系代数表达式为: $\Pi_{A_1, A_2, \dots, A_n}(\sigma_p(r_1 \times r_2 \times \dots \times r_m))$ 。

2. 简单查询

SQL 最简单的查询是找出关系中满足特定条件的元组, 这些查询与关系代数中的选择操作类似。简单查询只需要使用 3 个保留字 SELECT、FROM 和 WHERE。

【例 7.17】 查询学生-课程数据库中计算机系学生的学号、姓名及年龄。

```
SELECT  Sno, Sname, Age
FROM    S
WHERE   SD='CS';
```

注意, 通常为了便于理解查询语句的结构, 在写 SQL 语句时要将保留字如 FORM 或 WHERE 作为每一行的开头。但是, 如果一个查询或子查询非常短的时候, 可以直接将它写在一行上, 这种风格使得查询语句很紧凑, 也具有很好的可读性。如上例也可写成如下形式:

```
SELECT  Sno, Sname, Age FROM  S WHERE  SD='CS';
```

【例 7.18】 查询数学系全体学生的详细信息。

```
SELECT  * FROM  S WHERE  SD='MS';
```

【例 7.19】 查询学生的出生年份。

```
SELECT  Sno, 2004-Age FROM  S;
```

3. 连接查询

若查询涉及两个以上的表, 则称为连接查询。

【例 7.20】 检索选修了课程号为“C1”的学生号和 student 姓名可用连接查询和嵌套查询实现, 实现方法如下:

```
SELECT  Sno, Sname
FROM    S, SC
WHERE   S. Sno=SC. Sno AND SC.Cno='C1'
```

【例 7.21】 检索选修课程名为“MS”的学生号和 student 姓名可用连接查询和嵌套查询实现, 实现方法如下:

```

SELECT Sno, Sname
FROM S, SC, C
WHERE S.Sno=SC.Sno AND SC.Cno=C.Cno AND C.Cname='MS'
  
```

【例 7.22】 检索至少选修了课程号为“C1”和“C3”的学生号，实现方法如下：

```

SELECT Sno
FROM SC SCX, SC SCY
WHERE SCX.Sno=SCY.Sno AND SCX.Cno='C1' AND SCY.Cno='C3'
  
```

4. 子查询与聚集函数

1) 子查询

子查询也称嵌套查询。嵌套查询是指一个 SELECT-FROM-WHERE 查询块可以嵌入另一个查询块之中。在 SQL 中允许多重嵌套。

【例 7.23】 可以采用嵌套查询来实现例 7-21。

```

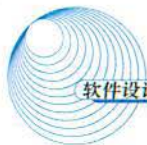
SELECT Sno, Sname
FROM S
WHERE Sno IN
  (SELECT Sno
   FROM SC
   WHERE Cno IN
    (SELECT Cno
     FROM C
     WHERE Cname='MS'))
  
```

2) 聚集函数

聚集函数是一个值的集合为输入，返回单个值的函数。SQL 提供了 5 个预定义集函数：平均值 AVG、最小值 MIN、最大值 MAX、求和 SUM 以及计数 COUNT。如表 7-4 所示。

表 7-4 集函数的功能

集函数名	功能
COUNT([DISTINCT ALL]*)	统计元组个数
COUNT([DISTINCT ALL]<列名>)	统计一列中值的个数
SUM([DISTINCT ALL]<列名>)	计算一列（该列应为数值型）中值的总和
AVG([DISTINCT ALL]<列名>)	计算一列（该列应为数值型）值的平均值
MAX([DISTINCT ALL]<列名>)	求一列值的最大值
MIN([DISTINCT ALL]<列名>)	求一列值的最小值



使用 ANY 和 ALL 谓词必须同时使用比较运算符,其含义及等价的转换关系见表 7-5。用集函数实现子查询通常要比直接用 ALL 或 ANY 查询效率要高。

表 7-5 ANY、ALL 谓词含义及等价的转换关系

谓词	语义	等价转换关系
>ANY	大于子查询结果中的某个值	>MIN
>ALL	大于子查询结果中的所有值	>MAX
<ANY	小于子查询结果中的某个值	<MAX
<ALL	小于子查询结果中的所有值	<MIN
>=ANY	大于等于子查询结果中的某个值	>=MIN
>=ALL	大于等于子查询结果中的所有值	>=MAX
<=ANY	小于等于子查询结果中的某个值	<=MAX
<=ALL	小于等于子查询结果中的所有值	<=MIN
◇ANY	不等于子查询结果中的某个值	--
◇ALL	不等于子查询结果中的任何一个值	NOT IN
=ANY	等于子查询结果中的某个值	IN
=ALL	等于子查询结果中的所有值	--

【例 7.24】 查询课程 C1 的最高分和最低分以及高低分之间的差距。

```
SELECT MAX(G),MIN(G),MAX(G)-MIN(G)
FROM Sc
WHERE Cno='C1'
```

【例 7.25】 查询其他系比计算机系 CS 所有学生年龄都要小的学生姓名及年龄。

方法 1: (用 ALL 谓词)

```
SELECT Sname, Age
FROM S
WHERE Age< ALL
      (SELECT Age
       FROM S
        WHERE SD='CS')
AND SD<◇'CS'
```

方法 2: (用 MIN 集函数) 从等价的转换关系表 7-5 中可见,“<ALL”可用“<MIN”代换。

```
SELECT Sname, Age
FROM S
```

```

WHERE Age<
  (SELECT MIN(Age)
   FROM S
   WHERE SD='CS' )
AND SD<>'CS'
  
```

方法 2 实际上是找出计算机系年龄最小的学生的年龄,只要其他系的学生年龄比这个年龄小,那么就应在结果集。

【例 7.26】 查询其他系比计算机系某一学生年龄小的学生姓名及年龄。

方法 1: (用 ANY 谓词)

```

SELECT Sname, Age
FROM S
WHERE Age< ANY
  (SELECT Age
   FROM S
   WHERE SD='CS')
AND SD<>'CS'
  
```

方法 2: (用 MAX 集函数) 从等价的转换关系表 7-5 中可见,“<ANY”可用“<MAX”代换。

```

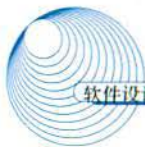
SELECT Sname, Age
FROM S
WHERE Age<
  (SELECT MAX(Age)
   FROM S
   WHERE SD='CS' )
AND SD<>'CS'
  
```

说明: 方法 2 实际上是找出计算机系年龄最大的学生的年龄,只要其他系的学生年龄比这个年龄大,那么就应在结果集。

5. 分组查询

1) GROUP BY 子句

在 WHERE 子句后面加上 GROUP BY 子句可以对元组进行分组,保留字 GROUP BY 后面跟着一个分组属性列表。最简单的情况是, FROM 子句后面只有一个关系,根据分组属性对它



的元组进行分组。SELECT 子句中使用的聚集操作符仅用在每个分组上。

【例 7.27】 学生数据库中的 SC 关系, 查询每个学生的平均成绩。

```
SELECT Sno, AVG(Grade)
FROM SC
GROUP BY Sno
```

该语句是将 SC 关系的元组重新组织, 并进行分组使得学号为 3001 的元组被组织在一起, 3002 的元组被组织在一起, 依次类推; 然后分别求出各个学生的平均值输出。

2) HAVING 子句

假如元组在分组前按照某种方式加上限制, 使得不需要的分组为空, 可以在 GROUP BY 子句后面跟一个 HAVING 子句即可。

注意: 当元组含有空值时, 应该记住以下两点:

① 空值在任何聚集操作中被忽视。它对求和、求平均值和计数都没有影响。它也不能是某列的最大值或最小值。例如, COUNT(*) 是某个关系中所有元组数目之和, 但 COUNT(A) 却是 A 属性非空的元组个数之和。

② NULL 值又可以在分组属性中看作是一个一般的值。例如, SELECT A, AVG(B) FROM R 中, 当 A 的属性值为空时, 就会统计 A=NULL 的所有元组中 B 的均值。

【例 7.28】 供应商数据库中的 S、P、J、SPJ 关系, 查询某工程至少用了 3 家供应商 (包含 3 家) 供应的零件的平均数量, 并按工程号的降序排列。

```
SELECT JNO, AVG(QTY)
FROM SPJ
GROUP BY JNO
HAVING COUNT(DISTINCT(SNO))>2
ORDER BY JNO DESC;
```

根据题意“某工程至少用了 3 家供应商 (包含 3 家) 供应的零件”, 应该按照工程号分组, 而且应该加上条件供应商的数目。但是需要注意的是, 一个工程项目可能用了同一个供应商的多种零件, 因此, 在统计供应商数的时候需要加上 DISTINCT, 以避免重复统计导致错误的结果。假如按工程号 JNO='J1' 分组, 结果如表 7-6 所示。

从表 7-6 我们可以看出, 如果不加 DISTINCT, 统计的数为 7, 而加了 DISTINCT, 统计的数是 5。

表 7-6 按工程号 JNO='J1' 分组

Sno	Pno	Jno	Qty
S1	P1	J1	200
S2	P3	J1	400
S2	P3	J1	200
S2	P5	J1	100
S3	P1	J1	200
S4	P6	J1	300
S5	P3	J1	200

6. 更名运算

SQL 提供可为关系和属性重新命名的机制, 这是通过使用具有如下形式的 `as` 子句来实现的:

Old-name as new-name

`as` 子句即可出现在 `select` 子句中, 也可出现在 `from` 子句中。

【例 7.29】 查询计算机学生的 `Sname` 和 `Age`, 但 `Sname` 用姓名表示, `Age` 用年龄表示。其语句如下:

```
SELECT  Sname as 姓名, Age as 年龄
FROM    S
WHERE   Age <
        (SELECT  MAX(Age)
         FROM    S
         WHERE   SD='CS')
AND SD <> 'CS'
```

SQL 中的元组变量必须和特定的关系相联系。元组变量是通过 `from` 子句中使用 `as` 子句来定义的。我们通过例 7-30 来说明。

【例 7.30】 查询计算机选修了 `C1` 课程的学生姓名 `Sname` 和成绩 `Grade`。其语句如下:

```
SELECT  Sname, Grade
FROM    Students as x, sc as y
WHERE   x.sno=y.sno and y.cno='C1'
```

元组变量在比较同一关系的两个元组是非常有用的。

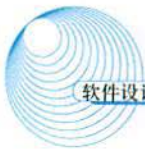
7. 字符串操作

对于字符串进行的最通常的操作是使用操作符 `like` 的模式匹配。使用两个特殊的字符来描述模式: “`%`” 匹配任意字符串; “`_`” 匹配任意一个字符。模式是大小写敏感的。例如:

“`Marry%`” 匹配任何以 “`Marry`” 开头的字符串; “`%idge%`” 匹配任何包含 “`idge`” 的字符串, 例如 “`Marryidge`”、“`Rock Ridge`”、“`Mianus Bridge`” 和 “`Ridgeway`”。

“`__`” 匹配只含两个字符的字符串; “`__%`” 匹配至少包含两个字符的字符串。

【例 7.31】 学生关系模式为 (`Sno`, `Sname`, `Sex`, `SD`, `Age`, `Add`), 其中: `Sno` 为学号, `Sname` 为姓名, `Sex` 为性别, `SD` 为所在系, `Age` 为年龄, `Add` 为家庭住址。请查询:



- ① 查询家庭住址包含“科技路”的学生姓名。
- ② 检索名字为“晓军”的学生姓名、年龄和所在系。

解 ① 家庭住址包含“科技路”的学生姓名的 SQL 语句如下:

```
SELECT Sname
FROM S
WHERE Add like '%科技路%'
```

- ② 名字为“晓军”的学生姓名、年龄和所在系的 SQL 语句如下:

```
SELECT Sname, Age, SD
FROM S
WHERE Sname LIKE '___晓军'
```

为了使模式中包含特殊模式字符(即%和_),在 SQL 中允许使用 escape 关键词来定义转义符。转义字符紧靠着特殊字符,并放在它的前面,表示该特殊字符被当成普通字符。例如在 like 比较中使用 escape 关键词来定义转义符,例如使用反斜杠“\”作为转义符。

Like 'ab\%cd%' escape '\', 匹配所有以 ab%cd 开头的字符串。

Like 'ab\\cd%' escape '\', 匹配所有以 ab\cd 开头的字符串。

8. 视图的查询

【例 7.32】 建立“计算机系”(CS 表示计算机系)学生的视图如下,并要求进行修改、插入操作时保证该视图只有计算机系的学生。

```
CREATE VIEW CS-STUDENT
AS SELECT Sno, Sname, Sage, Sex
FROM Student
WHERE SD='CS'
WITH CHECK OPTION;
```

此时要查询计算机年龄小于 20 岁的学号及年龄的 SQL 语句如下:

```
SELECT Sno, Age FROM CS-STUDENT WHERE SD='CS' AND Age<20;
```

系统执行该语句时,通常先将其转换成等价的对基本表的查询,然后执行查询语句。即当查询视图表时,系统先从数据字典中取出该视图的定义,然后将定义中的查询语句和对该视图的查询语句结合起来,形成一个修正的查询语句。对上例修正之后的查询语句为:

```
SELECT Sno, Age FROM S WHERE SD='CS' AND Age<20;
```

7.4.5 SQL 数据更新

1. 插入语句

要在关系数据库中插入数据，我们可以指定被插入的元组，或者用查询语句选出一批待插入的元组。插入语句的基本格式如下：

```
INSERT INTO 基本表名 (字段名[,字段名]...)
VALUES (常量[,常量]...); 查询语句
INSERT INTO 基本表名 (列表名)
SELECT 查询语句
```

【例 7.33】 将学号为“3002”、课程号为“C4”、成绩为 98 的元组插入 SC 关系中。其语句如下：

```
INSERT INTO SC
VALUES('3002', 'C4', 98)
```

【例 7.34】 首先创建了一个新的视图 v_employees，该视图基于表 employees 创建。

```
CREATE VIEW v_employees(number, name, age, sex, salary)
AS
SELECT number, name, age, sex, salary
FROM employees
WHERE name='张三'
```

然后通过执行以下语句使用该视图向表 employees 中添加一条新的数据记录。

```
INSERT INTO v_employees
VALUES(001, '李力', 22, 'm', 2000)
```

2. 删除语句

```
DELETE FROM 基本表名
[WHERE 条件表达式]
```

【例 7.35】 利用表 employees 中姓名为张然的记录。

```
DELETE FROM employees
WHERE name='张然'
```




3. 修改语句

UPDATE 基本表名

SET 列名=值表达式 (,列名=值表达式...)

[WHERE 条件表达式]

【例 7.36】 将教师的工资增加 5%。

```
UPDATE teachers
```

```
SET Salary = Salary*1.05
```

【例 7.37】 将教师的工资小于 1000 的增加 5%工资。

```
UPDATE teachers
```

```
SET Salary = Salary*1.05
```

```
WHERE Salary <= 1000
```

使用视图可以更新数据记录,但应该注意的是,更新的只是数据库中的基表。

【例 7.38】 创建了一个基于表 employees 的视图 v_employees,然后通过该视图修改表 employees 中的记录。其语句如下:

```
CREATE VIEW v_employees
```

```
AS
```

```
SELECT * FROM employees
```

```
UPDATE v_employees
```

```
SET name='张然'
```

```
WHERE name='张三'
```

7.4.6 SQL 的访问控制

数据控制的是控制用户对数据的存储权力,是由 DBA 来决定的。但是,某个用户对某类数据具有何种权利,是个政策问题而不是技术问题。DBMS 的功能就是保证这些决定的执行。因此,DBMS 数据控制应具有如下功能:

(1) 通过 GRANT 和 REVOK 将授权通知系统,并存入数据字典。

(2) 当用户提出请求时,根据授权情况检查是否执行操作请求。

SQL 标准包括 Delete、Insert、Select 和 Update 权限。Select 权限对应于 Read 权限,SQL 还包括了 References 权限,用来限制用户在创建关系时定义外码的能力。如果即将创建的关系中包含参照其他关系的属性的外码,那么用户必须在这些属性上具有 References 权限。References 权限之所以会成为一个有用的特征,其原因比较微妙。

1. 授权的语句格式

```

GRANT <权限>[,<权限>]...
  [ON<对象类型><对象名>]
  TO <用户>[,<用户>]...
  [WITH GRANT OPTION];
  
```

注意：不同类型的操作对象有不同的操作权限，常见的操作权限如表 7-7 所示。

表 7-7 常见的操作权限

对象	对象类型	操作权限
属性列	TABLE	SELECT, INSERT, UPDATE, DELETE, ALL PRIVILEGES (4 种权限的总和)
视图	TABLE	SELECT, INSERT, UPDATE, DELETE, ALL PRIVILEGES (4 种权限的总和)
基本表	TABLE	SELECT, INSERT, UPDATE, DELETE, ALTER, INDEX ALL PRIVILEGES (6 种权限的总和)
数据库	DATABASE	CREATETAB 建立表的权限，可由 DBA 授予普通用户

说明：

- PUBLIC：接受权限的用户可以是单个或多个具体的用户，PUBLIC 参数可将权限赋给全体用户。
- WITH GRANT OPTION：若指定了此子句，那么，获得了权限的用户还可以将权限赋给其他用户。

【例 7.39】 将对供应商 S、零件 P、项目 J 的所有操作权限赋给用户 User1 及 User2。

```
GRANT ALL PRIVILEGES ON TABLE S, P, J TO User1, User2;
```

【例 7.40】 将对供应商 S 的插入权限赋给用户 User1，并允许将此权限赋给其他用户。

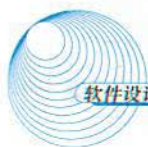
```
GRANT INSERT ON TABLE S TO User1 WITH GRANT OPTION;
```

【例 7.41】 DBA 把数据库 SPJ 中建立表的权限赋给用户 User1。

```
GRANT CREATETAB ON DATABASE SPJ TO User1;
```

2. 收回权限语句格式

```
REVOKE <权限>[,<权限>]...
```

[ON<对象类型><对象名>]
FROM <用户>[,<用户>]…;

【例 7.42】 将用户 User1 及 User2 对供应商 S、零件 P、项目 J 的所有操作权限收回。

REVOKE ALL PRIVILEGES ON TABLE S, P, J FROM User1, User2;

【例 7.43】 将所有用户对供应商 S 的所有查询权限收回。

REVOKE SELECT ON TABLE S FROM PUBLIC;

【例 7.44】 将 User1 用户对供应商 S 的供应商编号 Sno 的修改权限收回。

REVOKE UPDATE(Sno) ON TABLE S FROM User1;

7.4.7 嵌入式 SQL

SQL 提供了将 SQL 语句嵌入到某种高级语言中的使用方式,但是如何识别嵌入在高级语言中的 SQL 语句?通常采用预编译的方法。该方法的关键问题是必须区分主语言中嵌入的 SQL 语句,以及主语言和 SQL 间的通信问题。采用的方法由 DBMS 的预处理程序对源程序进行扫描,识别出 SQL 语句,把它们转换为主语言调用语句,以使主语言编译程序能识别它,最后由主语言的编译程序将整个源程序编译成目标码。为了区分主语言与 SQL 语言,需要在所有的 SQL 语句前加前缀 EXEC SQL,而 SQL 的结束标志随主语言的不同而不同。

例如,PL/I 和 C 语言的引用格式为:

EXEC SQL <SQL 语句>;

又如,COBOL 语言的引用格式为:

EXEC SQL <SQL 语句> END-EXEC

嵌入式 SQL 与主语言之间的通信采用 3 种方式:

(1) SQL 通信区:SQL 通信区(SQL Communication Area, SQLCA)向主语言传递 SQL 语句执行的状态信息,使主语言能够根据此信息控制程序流程。

(2) 主变量:也称共享变量。主语言向 SQL 语句提供参数主要通过主变量,主变量由主语言的程序定义,并用 SQL 的 DECLARE 语句说明。引用时,为了与 SQL 属性名相区别,需在主变量前加“:”。

【例 7.45】 根据共享变量 givensno 的值,查询学生关系 students 中学生的姓名、年龄和性别。

```
EXEC SQL SELECT sname,age,sex
      INTO :Msno,:Mcno,:givensno
      FROM students
      WHERE sno=:Msno;
```

(3) 游标 SQL 语言是面向集合的,一条 SQL 语句可产生或处理多条记录。而主语言是面向记录的,一组主变量一次只能放一条记录,所以,引入游标,通过移动游标指针来决定获取哪一条记录。

7.5 关系数据库规范化

在关系模型中,一个数据库模式是关系模式的集合。关系数据理论是指导数据库设计的基础,关系数据库设计是数据库语义学的问题。要保证构造的关系既能准确地反映现实世界,又有利于应用和具体的操作。关系数据库设计理论的核心是数据间的函数依赖,衡量的标准是关系规范化的程度及分解的无损连接和保持函数依赖性。关系数据库设计的目标是生成一组合适的、性能良好的关系模式,以减少系统中信息存储的冗余度,但又可方便地获取信息。

7.5.1 函数依赖

数据依赖是通过一个关系中属性间值的相等与否体现出来的数据间的相互关系,是现实世界属性间联系和约束的抽象,是数据内在的性质,是语义的体现。函数依赖则是一种最重要、最基本的数据依赖。

(1) 函数依赖: 设 $R(U)$ 是属性集 U 上的关系模式, X 、 Y 是 U 的子集。若对 $R(U)$ 的任何一个可能的关系 r , r 中不可能存在两个元组在 X 上的属性值相等,而在 Y 上的属性值不等,则称 X 函数决定 Y 或 Y 函数依赖于 X , 记作: $X \rightarrow Y$ 。

(2) 非平凡的函数依赖: 如果 $X \rightarrow Y$, 但 $Y \not\subseteq X$, 则称 $X \rightarrow Y$ 是非平凡的函数依赖。一般情况下总是讨论非平凡的函数依赖。

(3) 平凡的函数依赖: 如果 $X \rightarrow Y$, 但 $Y \subseteq X$, 则称 $X \rightarrow Y$ 是平凡的函数依赖。

(4) 完全函数依赖: 在 $R(U)$ 中, 如果 $X \rightarrow Y$, 并且对于 X 的任何一个真子集 X' , 都有 X' 不能决定 Y , 则称 Y 对 X 完全函数依赖, 记作: $X \xrightarrow{f} Y$ 。

例如: 给定一个学生选课关系 $SC(Sno, Cno, G)$, 我们可以得到 $F=\{(Sno, Cno) \rightarrow G\}$, 对 (Sno, Cno) 中的任何一个真子集 Sno 或 Cno 都不能决定 G , 所以, G 完全依赖于 Sno, Cno 。

(5) 部分函数依赖: 如果 $X \rightarrow Y$, 但 Y 不完全函数依赖于 X , 则称 Y 对 X 部分函数依赖, 记作: $X \xrightarrow{p} Y$ 。部分函数依赖也称局部函数依赖。

(6) 传递依赖: 在 $R(U, F)$ 中, 如果 $X \rightarrow Y$, $Y \not\subseteq X$, $Y \nrightarrow X$, $Y \rightarrow Z$, 则称 Z 对 X 传递依赖。



(7) 码: 设 K 为 $R(U, F)$ 中的属性的组合, 若 $K \rightarrow U$, 且对于 K 的任何一个真子集 K' , 都有 K' 不能决定 U , 则 K 为 R 的候选码, 若有多个候选码, 则选一个作为主码。候选码通常也称候选关键字。

(8) 主属性和非主属性: 包含在任何一个候选码中的属性叫做主属性, 否则叫做非主属性。

(9) 外码: 若 $R(U)$ 中的属性或属性组 X 非 R 的码, 但 X 是另一个关系的码, 则称 X 为外码。

(10) 值依赖定义: 若关系模式 $R(U)$ 中, X, Y, Z 是 U 的子集, 并且 $Z = U - X - Y$ 。当且仅当对 $R(U)$ 的任何一个关系 r , 给定一对 (x, z) 值, 有一组 Y 的值, 这组值仅仅决定于 x 值而与 z 值无关, 则称“ Y 多值依赖于 X ”或“ X 多值决定 Y ”成立。记为: $X \twoheadrightarrow Y$ 。

多值依赖具有如下 6 条性质:

- 多值依赖具有对称性。即若 $X \twoheadrightarrow Y$, 则 $X \twoheadrightarrow Z$, 其中 $Z = U - X - Y$ 。
- 多值依赖的传递性。即若 $X \twoheadrightarrow Y$, $Y \twoheadrightarrow Z$, 则 $X \twoheadrightarrow Z - Y$ 。
- 函数依赖可以看成是多值依赖的特殊情况。
- 若 $X \twoheadrightarrow Y$, $X \twoheadrightarrow Z$, 则 $X \twoheadrightarrow YZ$ 。
- 若 $X \twoheadrightarrow Y$, $X \twoheadrightarrow Z$, 则 $X \twoheadrightarrow Y \cap Z$ 。
- 若 $X \twoheadrightarrow Y$, $X \twoheadrightarrow Z$, 则 $X \twoheadrightarrow Z - Y$ 。

(11) 函数依赖的公理系统 (Armstrong 公理系统): 设关系模式 $R(U, F)$, 其中 U 为属性集, F 是 U 上的一组函数依赖, 那么有如下推理规则。

- A1 自反律: 若 $Y \subseteq X \subseteq U$, 则 $X \rightarrow Y$ 为 F 所蕴含。
- A2 增广律: 若 $X \rightarrow Y$ 为 F 所蕴含, 且 $Z \subseteq U$, 则 $XZ \rightarrow YZ$ 为 F 所蕴含。
- A3 传递律: 若 $X \rightarrow Y$, $Y \rightarrow Z$ 为 F 所蕴含, 则 $X \rightarrow Z$ 为 F 所蕴含。

根据上述 3 条推理规则又可推出下述 3 条推理规则。

- 合并规则: 若 $X \rightarrow Y$, $X \rightarrow Z$, 则 $X \rightarrow YZ$ 为 F 所蕴含。
- 伪传递率: 若 $X \rightarrow Y$, $WY \rightarrow Z$, 则 $XW \rightarrow Z$ 为 F 所蕴含。
- 分解规则: 若 $X \rightarrow Y$, $Z \subseteq Y$, 则 $X \rightarrow Z$ 为 F 所蕴含。

引理: $X \rightarrow A_1 A_2, \dots, A_k$ 成立的充分必要的条件是 $X \rightarrow A_i$ 成立 ($i=1, 2, 3, \dots, k$)。证明略。

7.5.2 规范化

关系数据库设计的方法之一就是设计满足适当范式的模式, 通常可以通过判断分解后的模式达到几范式来评价模式规范化的程度。范式有: 1NF、2NF、3NF、BCNF、4NF 和 5NF, 其中 1NF 级别最低。这几种范式之间 $5NF \subset 4NF \subset BCNF \subset 3NF \subset 2NF \subset 1NF$ 成立。通过分解, 可以将一个低一级范式的关系模式转换成若干个高一级范式的关系模式, 这种过程叫做规范化。下

面将给出各个范式的定义。

1. 1NF (第一范式)

定义: 若关系模式 R 的每一个分量是不可再分的数据项, 则关系模式 R 属于第一范式 (1NF)。

例如, 供应者和它所提供的零件信息, 关系模式 FIRST 和函数依赖集 F 如下:

FIRST (Sno, Sname, Status, City, Pno, Qty)

$F = \{ \text{Sno} \rightarrow \text{Sname}, \text{Sno} \rightarrow \text{Status}, \text{Status} \rightarrow \text{City}, (\text{Sno}, \text{Pno}) \rightarrow \text{Qty} \}$

若具体的关系如表 7-8 所示。从表 7-8 可以看出, 每一个分量都是不可再分的数据项, 所以是 1NF 的。

表 7-8 FIRST

Sno	Sname	Status	City	Pno	Qty
S1	精益	20	天津	P1	200
S1	精益	20	天津	P2	300
S1	精益	20	天津	P3	480
S2	盛锡	10	北京	P2	168
S2	盛锡	10	北京	P3	500
S3	东方红	30	北京	P1	300
S3	东方红	30	北京	P2	280
S4	泰达	40	上海	P2	460

但是, 1NF 存在 4 个问题:

(1) 冗余度大: 例如每个供应者的 Sno, Sname, Status, City 要与其供应的零件的种类一样多。

(2) 引起修改操作的不一致性: 例如供应者 S1 从“天津”搬到“上海”, 若稍不注意, 就会使一些数据被修改, 另一些数据没有被修改, 导致数据修改的不一致性。

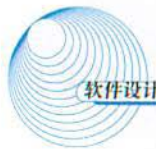
(3) 插入异常: 关系模式 FIRST 的主码为 Sno、Pno, 按照关系模式实体完整性规定主码不能取空值或部分取空值。这样, 当某个供应者的某些信息未提供时 (如 Pno), 则不能进行插入操作, 这就是所谓的插入异常。

(4) 删除异常: 若供应商 S4 的 P2 零件销售完了, 并且以后不再销售 P2 零件, 那么应删除该元组。这样, 在基本关系 FIRST 找不到 S4, 可 S4 又是客观存在的。

正因为上述 4 个原因, 所以要对模式进行分解, 并引入了 2NF。

2. 2NF (第二范式)

定义: 若关系模式 $R \in 1NF$, 且每一个非主属性完全依赖于码, 则关系模式 $R \in 2NF$ 。



换句话说,当 1NF 消除了非主属性对码的部分函数依赖,则称为 2NF。

例如: FIRST 关系中的码是 Sno、Pno, 而 $Sno \rightarrow Status$, 因此非主属性 Status 部分函数依赖于码, 故非 2NF 的。

若此时, 将 FIRST 关系分解为: FIRST1 (Sno, Sname, Status, City) 和 FIRST2 (Sno, Pno, Qty)。其中, $FIRST1 \in 2NF$, $FIRST2 \in 2NF$ 。因为分解后的关系模式 FIRST1 的码为 Sno, 非主属性 Sname、Status、City 完全依赖于码 Sno, 所以属于 2NF; 关系模式 FIRST2 的码为 Sno、Pno, 非主属性 Qty 完全依赖于码, 所以也属于 2NF。

3. 3NF (第三范式)

定义: 若关系模式 $R(U, F)$ 中若不存在这样的码 X , 属性组 Y 及非主属性 $Z(Z \notin Y)$ 使得 $X \rightarrow Y$, ($Y \not\rightarrow X$) $Y \rightarrow Z$ 成立, 则关系模式 $R \in 3NF$ 。

即当 2NF 消除了非主属性对码的传递函数依赖, 则称为 3NF。

例如: $FIRST1 \notin 3NF$, 因为在分解后的关系模式 FIRST1 中有 $Sno \rightarrow Status, Status \rightarrow City$, 存在着非主属性 City 传递依赖于码 Sno。若此时将 FIRST1 继续分解为:

$FIRST11(Sno, Sname, Status) \in 3NF$

$FIRST12(Status, City) \in 3NF$

通过上述分解, 数据库模式 FIRST 转换为 FIRST11(Sno, Sname, Status), FIRST12(Status, City), FIRST2 (Sno, Pno, Qty) 3 个子模式。由于这 3 个子模式都达到了 3NF, 因此称分解后的数据库模式达到了 3NF。

可以证明, 3NF 的模式必是 2NF 的模式。产生冗余和异常的两个重要原因是部分依赖和传递依赖。因为 3NF 模式中不存在非主属性对码的部分函数依赖和传递函数依赖, 所以具有较好的性能。对于非 3NF 的 1NF、2NF 其性能弱, 一般不宜作为数据库模式, 通常要将它们变换成为 3NF 或更高级别的范式, 这种变换过程称为“关系模式的规范化处理”。

4. BCNF (巴克斯范式)

定义: 若关系模式 $R \in 1NF$, 若 $X \rightarrow Y$ 且 $Y \not\subseteq X$ 时, X 必含有码, 则关系模式 $R \in BCNF$ 。

即当 3NF 消除了主属性对码的部分和传递函数依赖, 则称为 BCNF。

结论: 一个满足 BCNF 的关系模式, 应有如下性质:

- (1) 所有非主属性对每一个码都是完全函数依赖;
- (2) 所有非主属性对每一个不包含它的码, 也是完全函数依赖;
- (3) 没有任何属性完全函数依赖于非码的任何一组属性。

例如, 设 $R(Pno, Pname, Mname)$ 的属性分别表示零件号、零件名和厂商名, 如果约定,

每种零件号只有一个零件名,但不同的零件号可以有相同的零件名;每种零件可以有多个厂商生产,但每家厂商生产的零件应有不同的零件名。这样我们可以得到如下一组函数依赖:

$Pno \rightarrow Pname, (Pname, Mname) \rightarrow Pno$

由于该关系模式 R 中的候选码为 $(Pname, Mname)$ 或 $(Pno, Mname)$,因而关系模式 R 的属性都是主属性,不存在非主属性对码的传递依赖,所以 R 是 3NF 的。但是,主属性 $Pname$ 传递依赖于码 $(Pname, Mname)$,因此 R 不是 BCNF 的。当一种零件由多个生产厂家生产时,零件名与零件号间的联系将多次重复,带来冗余和操作异常现象。若将 R 分解成:

$R_1(Pno, Pname)$ 和 $R_2(Pno, Mname)$

就可以解决上述问题,并且分解后的 R_1, R_2 都属于 BCNF。

5. 4NF (第四范式)

定义:(4NF) 关系模式 $R \in 1NF$,若对于 R 的每个非平凡多值依赖 $X \twoheadrightarrow Y$ 且 $Y \not\subseteq X$ 时, X 必含有码,则关系模式 $R(U, F) \in 4NF$ 。

4NF 是限制关系模式的属性间不允许有非平凡且非函数依赖的多值依赖。

注意:如果只考虑函数依赖,关系模式最高的规范化程度是 BCNF;如果考虑多值依赖,关系模式最高的规范化程度是 4NF。

7.5.3 模式分解及分解应具有的特性

1. 分解

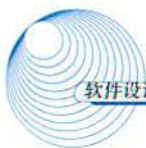
定义:(分解) 关系模式 $R(U, F)$ 的一个分解是指 $\rho = \{R_1(U_1, F_1), R_2(U_2, F_2), \dots, R_n(U_n, F_n)\}$, 其中: $U = \bigcup_{i=1}^n U_i$, 并且没有 $U_i \subseteq U_j, 1 \leq i, j \leq n, F_i$ 是 F 在 U_i 上的投影, $F_i = \{X \rightarrow Y \mid X \rightarrow Y \in F^+ \wedge XY \subseteq U_i\}$

对一个给定的模式进行分解,使得分解后的模式是否与原来的模式等价有 3 种情况:

- (1) 分解具有无损连接性;
- (2) 分解要保持函数依赖;
- (3) 分解既要无损连接性;又要保持函数依赖。

2. 无损连接

定义:(无损连接) $\rho = \{R_1(U_1, F_1), R_2(U_2, F_2), \dots, R_k(U_k, F_k)\}$ 是关系模式 $R(U, F)$ 的一个分解,若对 $R(U, F)$ 的任何一个关系 r 均有 $r = m_{\rho}(r)$ 成立,则成分解 ρ 具有无损连接性(简称无损分



解)。其中, $m_p(r) = \bigwedge_{i=1}^k \pi_{R_i}(r)$ 。

定理: 关系模式 $R(U, F)$ 的一个分解, $\rho = \{R_1(U_1, F_1), R_2(U_2, F_2)\}$ 具有无损连接的充分必要的条件是:

$$U_1 \cap U_2 \rightarrow U_1 - U_2 \in F^+ \quad \text{或} \quad U_1 \cap U_2 \rightarrow U_2 - U_1 \in F^+$$

证明略。

3. 保持函数依赖

定义: 设关系模式 $R(U, F)$ 的一个分解, $\rho = \{R_1(U_1, F_1), R_2(U_2, F_2), \dots, R_k(U_k, F_k)\}$, 如果 $F^+ = (\bigcup_{i=1}^k \pi_{R_i}(F^+))$, 则称分解 ρ 保持函数依赖。

4. 判别一个分解的无损连接性和保持函数依赖的算法

算法 1: 判别一个分解的无损连接性。

$\rho = \{R_1(U_1, F_1), R_2(U_2, F_2), \dots, R_k(U_k, F_k)\}$ 是关系模式 $R(U, F)$ 的一个分解, $U = \{A_1, A_2, \dots, A_n\}$, $F = \{FD_1, FD_2, \dots, FD_p\}$, 并设 F 是一个最小依赖集, 记 FD_i 为 $X_i \rightarrow A_{ij}$, 其步骤如下:

(1) 建立一张 n 列 k 行的表, 每一列对应一个属性, 每一行对应分解中的一个关系模式。若属性 $A_j \in U_i$, 则在 j 列 i 行上填上 a_{ij} , 否则填上 b_{ij} ;

(2) 对于每一个 FD_i 做如下操作: 找到 X_i 所对应的列中具有相同符号的那些行。考察这些行中 A_{ij} 列的元素, 若其中有 a_{ij} , 则全部改为 a_{ij} , 否则全部改为 b_{mli} , m 是这些行的行号最小值。

如果在某次更改后, 有一行成为: a_1, a_2, \dots, a_n , 则算法终止。且分解 ρ 具有无损连接性, 否则不具有无损连接性。

对 F 中 p 个 FD 逐一进行一次这样的处理, 称为对 F 的一次扫描。

(3) 比较扫描前后, 表有无变化, 如有变化, 则返回第 (2) 步, 否则算法终止。

如果发生循环, 那么前次扫描至少应使该表减少一个符号, 表中符号有限, 因此, 循环必然终止。

【例 7.46】 关系模式 $R(U, F)$, 其中 $U = \{A, B, C, D, E\}$, $F = \{AC \rightarrow E, E \rightarrow D, A \rightarrow B, B \rightarrow D\}$, 请判断如下两个分解是否无损连接的。

$$(1) \rho_1 = \{R_1(AC), R_2(ED), R_3(AB)\}$$

$$(2) \rho_2 = \{R_1(ABC), R_2(ED), R_3(ACE)\}$$

解: (1) 判断 ρ_1 是否无损的。根据“算法 1”构造一个二维矩阵表。

模式 \ 属性	A	B	C	D	E
$R_1(AC)$	a_1	b_{12}	a_3	b_{14}	b_{15}
$R_2(ED)$	b_{21}	b_{22}	b_{23}	a_4	a_5
$R_3(AB)$	a_1	a_2	b_{33}	b_{34}	b_{35}

根据 F 中的 $AC \rightarrow E$, 上表中 AC 属性列上没有两行相同的, 故不能修改上表。又由于 $E \rightarrow D$ 在 E 属性列上没有两行相同的, 故不能修改上表。根据 $A \rightarrow B$ 对上表进行处理, 由于属性列 A 上第一行、第三行相同为 a_1 , 所以将属性列 B 上 b_{12} 改为同一符号 a_2 。修改后的表如下:

模式 \ 属性	A	B	C	D	E
$R_1(AC)$	a_1	a_2	a_3	b_{14}	b_{15}
$R_2(ED)$	b_{21}	b_{22}	b_{23}	a_4	a_5
$R_3(AB)$	a_1	a_2	b_{33}	b_{34}	b_{35}

根据 F 中的 $B \rightarrow D$ 对上表进行处理, 由于属性列 B 上第一行、第三行相同为 a_2 , 所以将属性列 D 上 b_{14} 、 b_{34} 改为同一符号 b_{14} , 取行号最小值 (因为在属性列 D 上的第一行、第三行没有 a_4)。修改后的表如下:

模式 \ 属性	A	B	C	D	E
$R_1(AC)$	a_1	a_2	a_3	b_{14}	b_{15}
$R_2(ED)$	b_{21}	b_{22}	b_{23}	a_4	a_5
$R_3(AB)$	a_1	a_2	b_{33}	b_{14}	b_{35}

反复检查函数依赖集 F , 无法修改上表, 故分解 ρ_1 是有损的。

(3) 判断 ρ_2 是否无损的。根据算法 1 构造一个二维矩阵如下表所示。 $F = \{AC \rightarrow E, E \rightarrow D, A \rightarrow B, B \rightarrow D\}$

模式 \ 属性	A	B	C	D	E
$R_1(ABC)$	a_1	a_2	a_3	b_{14}	b_{15}
$R_2(ED)$	b_{21}	b_{22}	b_{23}	a_4	a_5
$R_3(ACE)$	a_1	b_{32}	a_3	b_{34}	a_5

根据 F 中的 $AC \rightarrow E$ 在 AC 属性列上第一行、第三行相同为 a_1a_3 , 所以将属性列 E 上 b_{15} 改为同一符号 a_5 。修改后的表如下:



模式 \ 属性	A	B	C	D	E
$R_1(ABC)$	a_1	a_2	a_3	b_{14}	a_5
$R_2(ED)$	b_{21}	b_{22}	b_{23}	a_4	a_5
$R_3(ACE)$	a_1	b_{23}	a_3	b_{34}	a_5

$E \rightarrow D$ 在 E 属性列上第一行、第二行、第三行相同为 a_5 , 所以将属性列 D 上 b_{14} 、 b_{34} 改为同一符号 a_4 。修改后的表如下:

模式 \ 属性	A	B	C	D	E
$R_1(ABC)$	a_1	a_2	a_3	a_4	a_5
$R_2(ED)$	b_{21}	b_{22}	b_{23}	a_4	a_5
$R_3(ACE)$	a_1	b_{23}	a_3	a_4	a_5

从修改后的表可以看出第一行全为 a , 故分解 ρ_2 是无损连接的。

算法 2: 转换成 3NF 的保持函数依赖的分解。

$\rho = \{R_1(U_1, F_1), R_2(U_2, F_2), \dots, R_k(U_k, F_k)\}$ 是关系模式 $R(U, F)$ 的一个分解, $U = \{A_1, A_2, \dots, A_n\}$, $F = \{FD_1, FD_2, \dots, FD_p\}$, 并设 F 是一个最小依赖集, 记 FD_i 为 $X_i \rightarrow A_{ij}$, 其步骤如下:

(1) 对 $R(U, F)$ 的函数依赖集 F 进行极小化处理 (处理后的结果仍记为 F);

(2) 找出不在 F 中出现的属性, 将这样的属性构成一个关系模式。把这些属性从 U 中去掉, 剩余的属性仍记为 U 。

(3) 若有 $X \rightarrow A \in F$, 且 $XA = U$, 则 $\rho = \{R\}$, 算法终止。

(4) 否则, 对 F 按具有相同左部的原则分组 (假定分为 k 组), 每一组函数依赖 F_i 所涉及的全部属性形成一个属性集 U_i 。若 $U_i \subseteq U_j (i \neq j)$ 就去掉 U_i 。由于经过了步骤 (2), 故

$U = \bigcup_{i=1}^k U_i$, 于是 $\rho = \{R_1(U_1, F_1), R_2(U_2, F_2), \dots, R_k(U_k, F_k)\}$ 构成 $R(U, F)$ 的一个保持函数依赖的分解。并且, 每个 $R_i(U_i, F_i)$ 均属于 3NF 且保持函数依赖。

【例 7.47】 关系模式 $R(U, F)$, 其中 $U = \{C, T, H, I, S, G\}$, $F = \{CS \rightarrow G, C \rightarrow T, TH \rightarrow I, HI \rightarrow C, HS \rightarrow I\}$, 将其分解成 3NF 并保持函数依赖。

解: 根据“算法 2”求解如下:

(1) F 已为最小函数依赖集, 所以转 (2);

(2) R 中的所有属性均在 F 中都出现, 所以转 (3);

(3) 对 F 按具有相同左部的原则分组为 $R_1 = CSG$, $R_2 = CT$, $R_3 = THI$, $R_4 = HIC$, $R_5 = HSR$ 。

所以关系模式 R 分解成 3NF 并保持函数依赖的分解为 $\rho = \{R_1(CSG), R_2(CT), R_3(THI), R_4(HIC), R_5(HSR)\}$

算法 3: 将一个关系模式转换成 3NF, 使它既具有无损连接又保持函数依赖的分解。

输入: 关系模式 R 和 R 的最小函数依赖集 F 。

输出: $R(U, F)$ 的一个分解 $\rho = \{R_1(U_1, F_1), R_2(U_2, F_2), \dots, R_k(U_k, F_k)\}$, R_i 为 3NF, 且 ρ 具有无损连接又保持函数依赖的分解。

操作步骤如下:

- (1) 根据“算法 2”求出保持依赖的分解 $\rho = \{R_1, R_2, \dots, R_n\}$;
- (2) 判断分解 ρ 是否具有无损连接性, 若有, 转 (4);
- (3) 令 $\rho = \rho \cup \{X\}$, 其中 X 是 R 的码;
- (4) 输出 ρ 。

【例 7.48】 对“例 7.47”的关系模式 $R(U, F)$ 将其分解成 3NF, 使 ρ 具有无损连接又保持函数依赖的分解。

解: 根据“算法 3”求解如下:

- (1) 据“例 7-48”得 3NF 保持函数依赖的分解如下:

$$\rho = \{R_1(CSG), R_2(CT), R_3(THI), R_4(HIC), R_5(HSR)\}$$

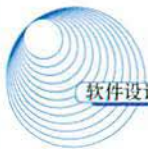
- (2) 根据“算法 1”构造一个二维矩阵如下表所示。

模式 \ 属性	C	T	H	I	S	G
$R_1(CSG)$	a_1	b_{12}	b_{13}	b_{14}	a_5	a_6
$R_2(CT)$	a_1	a_2	b_{23}	b_{24}	b_{25}	b_{26}
$R_3(THI)$	b_{31}	a_2	a_3	a_4	b_{35}	b_{36}
$R_4(HIC)$	a_1	b_{42}	a_3	a_4	b_{45}	b_{46}
$R_5(HSI)$	b_{51}	b_{52}	a_3	a_4	a_5	b_{56}

根据 F 中的 $C \rightarrow T$, 对上表进行处理, 由于属性列 C 上第一行、第二行及第四行相同为 a_1 , 所以将属性列 T 上 b_{12} 、 b_{42} 改为同一符号 a_2 。又根据 $HI \rightarrow C$ 将属性列 C 上 b_{31} 、 b_{51} 改为同一符号 a_1 , 修改后的表如下:

模式 \ 属性	C	T	H	I	S	G
$R_1(CSG)$	a_1	a_2	b_{13}	b_{14}	a_5	a_6
$R_2(CT)$	a_1	a_2	b_{23}	b_{24}	b_{25}	b_{26}
$R_3(THI)$	a_1	a_2	a_3	a_4	b_{35}	b_{36}
$R_4(HIC)$	a_1	a_2	a_3	a_4	b_{45}	b_{46}
$R_5(HIS)$	a_1	b_{52}	a_3	a_4	a_5	b_{56}

根据 F 中的 $CS \rightarrow G$, 对上表进行处理, 由于属性列 CS 上第一行、第五行相同为 a_1 、 a_5 ,



所以将属性列 G 上 b_{56} 改为同一符号 a_6 。又根据 $C \rightarrow T$ 将属性列 T 上 b_{52} 改为同一符号 a_2 , 修改后的表如下:

模式 \ 属性	C	T	H	I	S	G
$R_1(CSG)$	a_1	a_2	b_{13}	b_{14}	a_5	a_6
$R_2(CT)$	a_1	a_2	b_{23}	b_{24}	b_{25}	b_{26}
$R_3(THI)$	a_1	a_2	a_3	a_4	b_{35}	b_{36}
$R_4(HIC)$	a_1	a_2	a_3	a_4	b_{45}	b_{46}
$R_5(HIS)$	a_1	a_2	a_3	a_4	a_5	a_6

从上可见, 找到一行(第5行)为全 a , 所以分解 ρ 是无损连接并保持函数依赖的。

7.6 数据库的控制功能

7.6.1 事务管理

事务是一个操作序列, 这些操作“要么都做, 要么都不做”, 是数据库环境中不可分割的逻辑工作单位。事务和程序是两个不同的概念, 一般一个程序可包含多个事务。

事务的4个特性: 原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)和持久性(Durability)。这4个特性也称事务的ACID性质。

(1) 原子性: 事务的所有操作在数据库中要么全做要么全都不做。如银行转账中的两个操作必须作为一个单位来处理, 不能只执行部分操作。

(2) 一致性: 一个事务独立执行的结果, 将保持数据的一致性, 即数据不会因为事务的执行而遭受破坏。数据的一致性是对现实世界的真实状态的描述, 如银行转账业务执行后也应该是账目平衡的。数据库在运行过程中会出现瞬间的不一致状态, 如从A账户减去 x 元到给B账户加上 x 元之前这段时间数据是不一致, 但这种不一致只能出现在事务执行过程中, 并且不一致的数据不能被其他事务所访问。一致性可以由DBMS的完整性约束机制来自动完成, 而复杂的事务则由应用程序来完成。

(3) 隔离性: 一个事务的执行不能被其他事务干扰。并发事务在执行过程中可能会对同一数据进行操作, 这些事务的操作应该不会相互干扰, 是相互隔离的。如事务执行中数据不一致性状态出现时不能让其他事务读取到不一致的数据。

(4) 持久性: 一个事务一旦提交, 它对数据库的改变必须是永久的, 即便系统出现故障时也是如此。如转账事务执行成功后, A、B两个账户上的余额就是一个新的值, 在没有出现下一个事务对其修改之前一直保持不变, 即使系统出现故障, 也应该恢复到这个值。

7.6.2 数据库的备份与恢复

在数据库的运行过程中,难免会出现计算机系统的软、硬件故障,这些故障会影响数据库中数据的正确性,甚至破坏数据库,使数据库中全部或部分数据丢失。因此,数据库的关键技术在于建立冗余数据,即备份数据。如何在系统出现故障后能够及时使数据库恢复到故障前的正确状态,就是数据库恢复技术。

1. 数据库备份的必要性

事实上,在当今信息社会,最珍贵的财产并不是计算机软件,更不是计算机硬件,而是企业在长期发展过程中所积累下来的业务数据。建立网络最根本的用途是要更加方便地传递与使用数据,但人为错误、硬盘损坏、电脑病毒、断电或是天灾人祸等都有可能造成数据的丢失。所以应该强调指出:“数据是资产,备份最重要。”应当理解备份意识实际上就是数据的保护意识,在危机四伏的网络环境中,数据随时有被毁灭的可能。系统灾难的发生,不是是否会有问题,而是迟早的问题。造成系统数据破坏、丢失的原因很多,有些还往往被人们忽视。正确分析威胁数据安全的因素,及时地备份数据,能使系统的安全防护更有针对性。在数据库中的4类故障是:事务内部故障、系统故障、介质故障及计算机病毒。

(1) 事务内部故障:事务内部的故障有的是可以通过事务程序本身发现。例如,银行转账事务,将账户A的金额转 x 到账户B,此时应该将账户A的余额 $-x$,将账户B的余额 $+x$ 。如果账户A的余额不足,那么,两个事务都不做,否则都做。但有些是非预期的,不能由事务程序处理的,如运算溢出、并发事务发生死锁等。

(2) 系统故障:通常称为软故障,是指造成系统停止运行的任何事件,使得系统要重新启动,如CPU故障、操作系统故障、突然停电等。

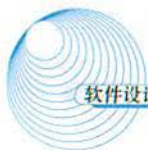
(3) 介质故障:通常称为硬故障。如磁盘损坏、磁头碰撞、瞬时强磁干扰。此类故障发生几率小,但破坏性最大。

(4) 计算机病毒:是一种人为的故障和破坏,是一些恶作剧研制的一种计算机程序,可以繁殖和传播。

2. 故障的恢复方法

恢复的基本原理是“建立数据冗余”(重复存储)。建立冗余数据的方法进行数据转储和登记日志文件。数据的转储分为:静态转储和动态转储、海量转储和增量转储。

(1) 静态转储和动态转储:静态转储是指在转储期间不允许对数据库进行任何存取、修改操作;动态转储是在转储期间允许对数据库进行存取、修改操作,因此,转储和用户事务可并发执行。



(2) 海量转储和增量转储: 海量转储是指每次转储全部数据; 增量转储是指每次只转储上次转储后更新过的数据。

(3) 日志文件: 在事务处理的过程中, DBMS 把事务开始、事务结束以及对数据库的插入、删除和修改的每一次操作写入日志文件。一旦发生故障, DBMS 的恢复子系统利用日志文件撤销事务对数据库的改变, 回退到事务的初始状态。因此, DBMS 利用日志文件来进行事务故障恢复和系统故障恢复, 并可协助后备副本进行介质故障恢复。

事务恢复有 3 个步骤:

- ① 反向扫描文件日志 (即从最后向前扫描日志文件), 查找该事务的更新操作。
- ② 对事务的更新操作执行逆操作。
- ③ 继续反向扫描日志文件, 查找该事务的其他更新操作, 并做同样的处理。直到事务的开始标志。

3. 数据库镜像

为了避免磁盘介质出现故障影响数据库的可用性, 许多 DBMS 提供数据库镜像功能用于数据库恢复。需要说明的是, 数据库镜像是通过复制数据实现的, 但频繁地复制数据会降低系统的运行效果, 因此实际应用中往往只对关键的数据和日志文件镜像。

在 SQL 语言中事务定义的语句有 3 条:

- ① BEGIN TRANSACTION 事务开始。
- ② COMMIT 事务提交。该操作表示事务成功地结束, 它将通知事务管理器该事务的所有更新操作现在可以被提交或永久地保留。
- ③ ROLLBACK 事务回滚。该操作表示事务非成功地结束, 它将通知事务管理器出故障了, 数据库可能处于不一致状态, 该事务的所有更新操作必须回滚或撤销。

7.6.3 并发控制

所谓并发操作是指在多用户共享的系统中, 许多用户可能同时对同一数据进行操作。并发操作带来的问题是数据的不一致性, 主要有 3 类: 丢失更新、不可重复读和读脏数据。其主要原因是: 事务的并发操作破坏了事务的隔离性。DBMS 的并发控制子系统负责协调并发事务的执行, 保证数据库的完整性不受破坏, 避免用户得到不正确的数据。

并发控制的主要技术是封锁。基本封锁的类型有: 排他锁 (简称 X 锁或写锁) 和共享锁 (简称 S 锁或读锁)。

1. 排他型封锁 (X 封锁)

- (1) 排他锁: 若事务 T 对数据对象 A 加上 X 锁, 则只允许 T 读取和修改 A, 其他事务都

不能再对 A 加任何类型的锁，直到 T 释放 A 上的锁。

(2) 共享锁：若事务 T 对数据对象 A 加上 S 锁，则只允许 T 读取 A，但不能修改 A，其他事务只能再对 A 加 S 锁，直到 T 释放 A 上的 S 锁。这就保证了其他事务可以读 A，但在 T 释放 A 上的 S 锁之前不能对 A 进行任何修改。

2. 三级封锁协议

(1) 一级封锁协议：事务在修改数据 R 之前必须先对其加 X 锁，直到事务结束才释放。事务结束包括：正常结束 (COMMIT) 和非正常结束 (ROLLBACK)。一级封锁协议可以解决丢失更新问题。

(2) 二级封锁协议：在一级封锁协议的基础上，加上事务 T 在读数据 R 之前必须先对其加 S 锁，读完后即可释放 S 锁。二级封锁协议可以解决读脏数据的问题。但是由于二级封锁协议读完了数据后即可释放 S 锁，所以不能保证可重复读。

(3) 三级封锁协议：在一级封锁协议的基础上，加上事务 T 在读数据 R 之前必须先对其加 S 锁，直到事务结束时释放 S 锁。三级封锁协议除了防止了丢失修改和不读“脏”数据外，还进一步防止了不可重复读。

3. 活锁与死锁

所谓活锁是指当事务 T_1 封锁了数据 R，事务 T_2 请求封锁数据 R 于是 T_2 等待，当 T_1 释放了 R 上的封锁后，系统首先批准了 T_3 请求，于是 T_2 仍等待，当 T_3 释放了 R 上的封锁后，又批准了 T_4 请求……依次类推，使得 T_2 可能永远等待。这种现象就是活锁。

所谓死锁是指两个以上的事务分别请求封锁对方已经封锁的数据，导致长期等待而无法继续运行下去的现象叫做死锁。

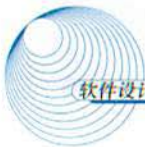
4. 并发调度的可串行性

多个事务的并发执行是正确的，当且仅当其结果与某一次序串行地执行它们时的结果相同，我们称这种调度策略是可串行化的调度。

可串行性是并发事务正确性的准则，按这个准则规定，一个给定的并发调度，当且仅当它是可串行化的才认为是正确调度。

5. 两段封锁协议

所谓两段封锁协议是指所有事务必须分两个阶段对数据项加锁和解锁。即事务分两个阶段，第一阶段是获得封锁，事务可以获得任何数据项上的任何类型的锁，但不能释放；第二阶段是释放封锁，事务可以释放任何数据项上的任何类型的锁，但不能申请。



6. 封锁的粒度

封锁对象的大小称为封锁的粒度。封锁的对象可以是逻辑单元(如属性、元组、关系、索引、整个索引直至整个数据库),也可以是物理单元(如数据页或索引页)。

7. 事务的嵌套问题

事务是不能嵌套的,因为这违背了事务的原子性。如果事务 T_2 嵌套在事务 T_1 内,并且发生了如下事件:

```
BEGIN TRANSACTION(transaction  $T_1$ );
```

```
...
```

```
BEGIN TRANSACTION(transaction  $T_2$ );
```

```
Transaction B updates tuple t;
```

```
COMMIT (transaction  $T_2$ );
```

```
ROLLBACK (transaction  $T_1$ );
```

如果此时将元组 t 恢复为事务 T_1 发生前的值,则事务 T_2 的 COMMIT 实际上就不是真正的 COMMIT。相反,如果保证了 T_2 的 COMMIT 的真实性,元组 t 就不能恢复为事务 T_1 发生前的值,从而 T_1 的 ROLLBACK 就不是真正的 ROLLBACK。

事务不能嵌套是指当且仅当当前没有事务在运行时,程序才能执行 BEGIN TRANSACTION 操作。实际上,通过放弃内层事务的持久性,可允许事务嵌套。也就是说,内层事务的 COMMIT 将提交该事务的更新操作,只限于外层事务的范围。如果外层事务以回滚操作终止,内层事务也必须回滚。从纯语义的角度看,像 SQL 这样缺少显式 BEGIN TRANSACTION 的语言中,很难实现事务嵌套。因为必须有某种显式的方式指出内层事务的初始点,用以标识内层事务失败时应回滚的终点。

7.6.4 安全性和授权

除了完整性约束提供保护意外引入的不一致性之外,数据库中存储的数据还要防止未经授权的访问和恶意的破坏或修改。这一节中,我们要看一看数据误使用或故意使数据不一致的一些方式,然后我们给出防止这些情况的机制。

1. 安全性违例

(1) 恶意访问的形式

- ① 未经授权读取数据(窃取信息)。
- ② 未经授权修改。

③ 未经授权破坏数据。

(2) 数据库安全性措施。数据库安全性(Data Base Security)指保护数据库不受恶意访问。绝对杜绝数据库的恶意滥用是不可能的,但可以使那些企图在没有适当授权情况下访问数据库的代价足够高,以阻止绝大多数这样的访问企图。为了保护数据库,我们必须在几个层次上采取安全性措施:

① 数据库系统层次。数据库系统的某些用户获得的授权可能只允许他访问数据库中有限的部分,而另外一些用户获得的授权可能允许他提出查询,但不允许他修改数据。保证这样的授权限制不被违反是数据库系统的责任。

② 操作系统层次。不管数据库系统多安全,操作系统安全性方面的弱点总是可能成为对数据库进行未经授权访问的一种手段。

③ 网络层次。由于几乎所有的数据库系统都允许通过终端或网络进行远程访问,网络软件的安全性和物理安全性一样重要,不管在因特网上还是在私有的网络内。

④ 物理层次。计算机系统所位于的节点(一个或多个)必须在物理上受到保护,以防止入侵者强行闯入或暗中潜入。

⑤ 人员层次。对用户的授权必须格外小心,以减少授权用户接受贿赂或其他好处而给入侵者提供访问机会的可能性。

为了保证数据库安全,我们必须在上述所有层次上进行安全性维护。如果较低层次上(物理层次或人员层次)安全性存在缺陷,高层安全性措施即使很严格也可能被绕过。在此我们将在数据库系统层次上讨论安全性。

2. 授权

我们可以赋予用户在数据库各个部分上的几种形式的授权,其中包括:

- (1) READ 授权允许读取数据,但不允许修改数据。
- (2) INSERT 授权允许插入新数据,但不允许修改已经存在的数据。
- (3) UPDATE 授权允许修改数据,但不允许删除数据。
- (4) DELETE 授权允许删除数据。

我们可以赋予用户可以获得上面的所有授权类型或其中一部分的组合,也可以根本不获得任何授权。除了以上几种对数据访问的授权外,用户还可以获得修改数据库模式的授权:

- (1) INDEX 授权允许创建和删除索引。
- (2) RESOURCE 授权允许创建新关系。
- (3) ALTERATION 授权允许添加或删除关系中的属性。
- (4) DROP 授权允许删除关系。

DROP 授权和 DELETE 授权的区别在于 DELETE 授权只允许对元组进行删除。如果用户



删除了关系中的所有元组, 关系仍然存在, 只不过是空的。如果关系被删除, 那么关系就不再存在了。

我们通过 RESOURCE 授权来控制创建新关系的能力。具有 RESOURCE 授权的用户在创建新关系后自动获得该关系上的所有权限。

INDEX 授权看起来似乎是不必要的, 因为索引的创建和删除不会改变关系中的数据。事实上, 索引是提高性能的一种结构。但是, 索引也会消耗空间, 并且所有数据库的修改都需要更新索引。如果 INDEX 授权被授予所有用户, 那么执行更新操作的用户倾向于删除索引, 而提出查询的用户倾向于创建大量索引。为了使数据库管理员能够管理系统资源的使用, 我们有必要将索引的创建作为一种权限来看待。

最大的授权形式是给数据库管理员的。数据库管理员可以给新用户授权, 可以重构数据库等。这一授权形式类似于操作系统中提供给超级用户或操作员的权限。

3. 授权与视图

我们介绍了视图的概念, 视图是给用户个性化数据库模型的一种手段。视图可以隐藏用户不需要看见的数据。视图隐藏数据的能力既可以用于简化系统的使用, 又可以用于实现安全性。由于视图只允许用户关注那些感兴趣的数据, 它简化了系统的使用。尽管用户可能不被允许直接访问某个关系, 但用户可能被允许通过一个视图访问该关系的一部分。因此, 关系级的安全性和视图级的安全性可以结合起来, 用于限制用户只能访问所需数据。

在我们的银行例子中, 考虑一个需要知道在各支行有贷款的所有客户姓名的职员。该职员不能看到与客户具体贷款相关的信息。因此, 该职员对 loan 关系的直接访问必须被禁止, 但是, 如果该职员要访问所需信息, 就必须得到对视图 Cust-Loan 的访问, 这一视图由所有客户姓名及其贷款支行构成。此视图可以用 SQL 定义如下:

```
create view cust-loan as
(select branch-name, customer-name
 from borrower, loan
 where borrower.loan-number=loan.loan-number)
```

假设该职员提出如下 SQL 查询:

```
select*
from cust-loan
```

显然, 该职员被允许看到此查询的结果。但是, 当查询处理器将此查询转换为数据库中的事实关系上的查询时, 它产生的是 borrower 和 loan 上的查询。因此, 系统对职员查询授权的检

查必须在查询处理开始之前进行。

创建视图并不需要 RESOURCE 授权。创建视图的用户不一定能获得该视图上的所有权限，得到的权限不会为他提供超过原有授权的其他授权。例如，在用来定义视图的关系上没有 UPDATE 授权的用户不能得到相应视图上的 UPDATE 授权。如果用户创建一个视图，而此用户在该视图上不能获得任何授权，这样的视图创建请求将被系统拒绝。在 cust-loan 的例子中，视图的创建者必须在关系 borrower 和 loan 上都具有 READ 授权。

4. 权限的授予

获得了某种形式授权的用户可能被允许将此授权传递给其他用户。但是，必须对授权可能怎样在用户间传递格外小心，以保证这样的授权在未来的某个时候可以被收回。

例如，银行数据库中 loan 关系上 UPDATE 权限的授予。假设最初数据库管理员将 loan 上的 UPDATE 权限授给用户 U_1 、 U_2 和 U_3 ，接下来他们又可以再将定授权传递给其他用户。授权从一个用户到另一个用户的传递可以表示为授权图 (Authorization Graph)。该图的节点是用户。如果用户 U_i 将 loan 上的 UPDATE 权限授给用户 U_j ，则图中包含边 $U_i \rightarrow U_j$ 。图的根是数据库管理员。图 7-33 给出了一个示例的图，请注意用户 U_1 和 U_2 都给 U_5 授予了权限；而 U_4 只从 U_1 处获得了授权。

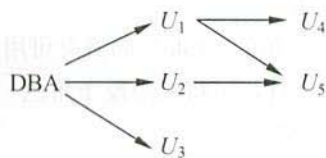


图 7-33 权限授予图

用户具有授权当且仅当存在从授权图的根（即代表数据库管理员的节点）到代表该用户的节点的路径。

设数据库管理员决定收回用户 U_1 的授权。由于用户 U_4 从 U_1 处获得授权，因此其权限也应该被收回。可是，用户 U_5 既从 U_1 处又从 U_2 处获得了授权。由于数据库管理员没有从 U_2 处收回 loan 上的 UPDATE 授权， U_5 继续拥有 loan 上的 UPDATE 授权。如果 U_2 最后从 U_5 处收回授权，则 U_5 失去授权。

一对狡猾的用户可能企图通过相互授权来破坏权限回收规则，如图 7-34 (a) 所示。如果数据库管理员从 U_2 收回权限， U_2 保留了通过 U_3 获得的授权，如图 7-34 (b) 所示。如果权限接着从 U_3 处收回， U_3 似乎保留了通过 U_2 获得的授权，如图 7-34 (c) 所示。然而，当数据库管理员从 U_3 处收回权限时，从 U_3 到 U_2 的边以及从 U_2 到 U_3 的边就不再是从数据库管理员开始的路径的一部分了。我们要求授权图中的所有边都必须是某条从数据库管理员开始的路径的一部分。 U_2 和 U_3 之间的边将被删除，结果授权图如图 7-35 所示。

5. 角色

考虑一个有很多出纳的银行。每一个出纳必须对同一组关系具有同种类型的权限。无论何时指定一个新的出纳，他都必须被单独授予所有这些授权。

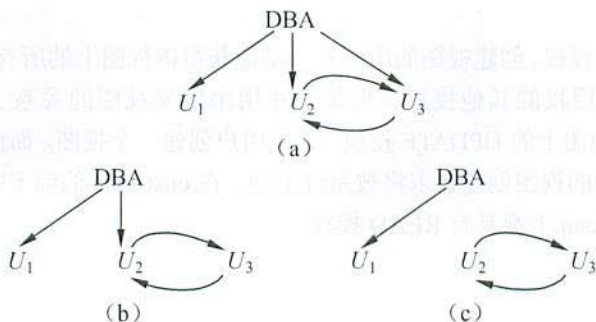
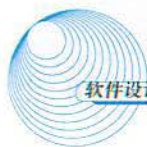


图 7-34 破坏权限回收的企图

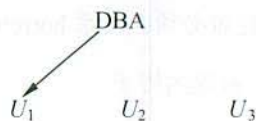


图 7-35 授权图

一个更好的机制是指明所有出纳应该有的授权，并单独标示出哪些数据库用户是出纳。系统可以用这两条信息来确定每一个有出纳身份的人的权限。当一个人被新雇用为出纳时，必须给他分配一个用户标识符，并且必须将他标示为一个出纳，而不需要重新单独给予出纳权限。

角色 (role) 的概念可用于该机制。在数据库中建立一个角色集，和授予给每一个单个用户一样，可将权限授予角色。分配给每个数据库用户一些他 (或她) 有权扮演的角色 (也可能是空的)。

例如银行数据库中，角色的例子可以包括 teller、branch-manager、auditor 和 system-administrator。一个不是很合适的方法是建立一个 teller 用户号，允许每一个出纳用这个出纳用户号来连接数据库。该机制的问题是它无法鉴别出到底哪个出纳执行了事务，从而导致安全隐患。应用角色的好处是需要每个用户用自己的用户号连接数据库。

任何可以授予一个用户的权限都可以授予一个角色。给用户分配角色就跟给用户授权一样。和其他授权一样，一个用户也可以被授予给他人分配角色的权限。这样，可以授予支行经理分配出纳角色的权限。

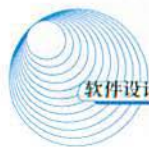
6. 审计追踪

很多安全的数据库应用软件需要维护审计追踪 (audit trail)。审计追踪是一个对数据库所有更改 (插入/删除/更新) 的日志，还包括一些其他信息，如哪个用户执行了更改和什么时候执行的更改等。

审计追踪在几个方面加强了安全性。例如，如果发现一个账户的余额不正确，银行也许希望跟踪所有在这个账户上的更新来找到错误 (或欺骗性) 的更新，同时也会找到执行这个更新的人。然后银行就可以利用审计追踪跟踪这些人所做的所有的更新来找到其他错误或欺骗性

的更新。

可以在关系更新操作上定义适当的触发器来建立一个审计追踪(利用标示用户名和时间的系统变量)。然而,很多的数据库系统提供了内置机制来建立审计追踪,用起来会更加方便。具体怎么建立审计追踪的细节随着不同的数据库系统而不同,可参考数据库系统用户手册来了解具体细节。



第8章 数据结构

数据结构是计算机程序设计的重要理论和技术基础,它所讨论的内容和提倡的技术方法,无论是对学习计算机领域的其他课程,还是对从事软件项目的开发都有着重要的作用。学习数据结构要达到的目标是:学会从问题入手,分析和研究计算机加工的数据结构的特性,以便为应用所涉及的数据选择适当的逻辑结构、存储结构及其相应的操作方法,并初步掌握算法的时间复杂度和空间复杂度概念。

数据结构是指数据元素的集合(或数据对象)及元素间的相互关系和构造方法,一个数据结构 B 可用一个二元组表示为: $B=(A, R)$, 其中 A 是数据元素的非空有限集合, R 是定义在 A 上的关系的非空有限集合。结构就是元素之间的关系。在数据结构中,元素之间的相互关系是数据的逻辑结构,数据元素及元素之间关系的存储形式称为存储结构(或物理结构)。

数据结构按照逻辑关系的不同分为线性结构和非线性结构两大类,其中非线性结构又可分为树结构和图结构。

算法与数据结构密切相关,数据结构是算法设计的基础,算法总是建立在一定的数据结构基础之上,合理的数据结构可使算法简单而高效。

8.1 线性结构

线性结构的特点是数据元素之间是一种线性关系,数据元素“一个接一个地排列”,易于进行查找、插入和删除等操作,主要用于对客观世界中具有单一的前驱和后继的数据关系进行描述。

8.1.1 线性表

线性表是最简单、最基本、也是最常用的一种线性结构。它有两种存储方法:顺序存储和链式存储,主要的基本操作是插入、删除和检索等。

1. 线性表的定义

一个线性表是 n 个元素的有限序列,其中 $n \geq 0$,通常表示为 (a_1, a_2, \dots, a_n) 。其特点是,在非空的数据元素集合中存在唯一的一个称作“第一个”的元素;存在唯一的一个称作“最后一个”的元素;除第一个元素外,序列中的每个元素均只有一个直接前驱;除最后一个元素外,

序列中的每个元素均只有一个直接后继。

2. 线性表的存储结构

1) 线性表的顺序存储

线性表的顺序存储是指用一组地址连续的存储单元依次存储线性表中的数据元素,从而使得逻辑上相邻的两个元素在物理位置上也相邻,如图 8-1 所示。在这种存储方式下,存储元素间的逻辑关系无需占用额外的存储空间。

一般地,以 $LOC(a_1)$ 表示线性表中第一个元素的存储位置,在顺序存储结构中,第 i 个元素 a_i 的存储位置为:

$$LOC(a_i) = LOC(a_1) + (i-1) \times L$$

其中 L 是表中每个元素所占空间的大小。根据该计算关系,可随机存取表中的任意一个元素。

线性表采用顺序存储结构的优点是可以随机存取表中的元素,缺点是插入和删除操作需要移动元素。插入元素前要移动元素以挪出空的存储单元,然后再插入元素;删除元素时同样需要移动元素,以填充被删除的元素空出来的存储单元。

在表长为 n 的线性表中插入新元素时,共有 $n+1$ 个插入位置,在位置 1 (元素 a_1 所在位置) 插入元素时需要移动 n 个元素,在位置 $n+1$ (元素 a_n 所在位置之后) 插入元素时不需要移动元素,因此,等概率下插入元素时平均移动元素的次数 E_{insert} 为:

$$E_{insert} = \sum_{i=1}^{n+1} P_i \times (n-i+1) = \frac{1}{n+1} \sum_{i=1}^{n+1} (n-i+1) = \frac{n}{2}$$

其中, P_i 表示在表中的位置 i 插入元素的概率。

在表长为 n 的线性表中删除元素时,共有 n 个可删除的元素,删除元素 a_1 时需要移动 $n-1$ 个元素,删除元素 a_n 时不需要移动元素,因此,等概率下删除元素时平均移动元素的次数 E_{delete} 为:

$$E_{delete} = \sum_{i=1}^n q_i \times (n-i) = \frac{1}{n} \sum_{i=1}^n (n-i) = \frac{n-1}{2}$$

其中, q_i 表示删除元素 a_i 的概率。

2) 线性表的链式存储

线性表的链式存储是用节点来存储数据元素,节点的地址可以连续,也可以不连续,因此存储数据元素的同时必须存储元素之间的逻辑关系。另外,节点空间只有在需要的时候才申请,无需事先分配。基本的节点结构如下所示:

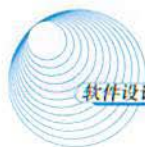
数据域	指针域
-----	-----

。

其中,数据域用于存储数据元素的值,指针域则存储当前元素的直接前驱或直接后继信息,指针域中的信息称为指针(或链)。 n 个节点通过指针连成一个链表,若节点中只有一个指针域,

a_1
a_2
\vdots
a_{n-1}
a_n

图 8-1 线性表元素的顺序存储



则称为线性链表(或单链表),如图8-2所示。



图8-2 线性表元素的单链表存储

在链式存储结构中,只需要一个指针(称为头指针)指向第一个节点,就可以顺序访问到表中的任意一个元素。为了简化对链表状态的判定和处理,特别引入一个不存储数据元素的节点,称为头节点,将其作为链表的第一个节点并令头指针指向该节点。

在链式存储结构下进行插入和删除,其实质都是对相关指针的修改。在单向链表中的 $*p$ 节点(图8-3(a)中元素 a 所在节点)后插入新元素节点 $*s$ (图8-3(a)中元素 c 所在节点)时,首先将 $*p$ 节点的后继节点指针赋给 $*s$ 的指针域,如图8-3(a)中①所示,然后将 $*p$ 节点的指针域修改为指向 $*s$ 节点,如图8-3(a)中②所示。

在单向链表中删除节点 $*p$ 的后继节点(图8-3(b)中元素 b 所在节点)时,令 $*p$ 节点的指针域指向 $*p$ 节点的后继的后继节点(图8-3(b)中元素 c 所在节点),从而将元素 b 所在的节点从链表中摘除,指针的变化情况如图8-3(b)所示。

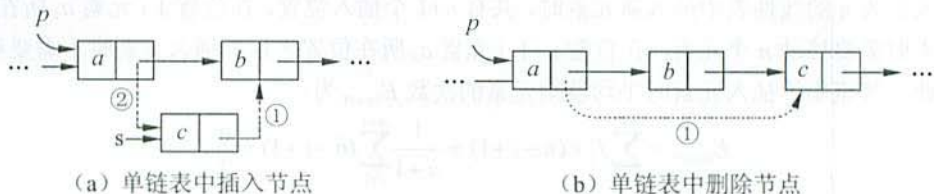


图8-3 在单向链表中插入、删除节点时的指针变化示意图

设单链表节点类型的定义为:

```
typedef struct node{
    int data;
    struct node *link;
}NODE,*LinkList;
```

下面给出单链表上的查找、插入和删除运算的实现过程。

【函数】 单链表的查找运算。

```
LinkList Find_List(LinkList L,int k) /*L为带头节点单链表的头指针*/
/*在表中查找第k个元素,若找到,返回该元素节点的指针,否则,返回空指针 NULL*/
{ LinkList p; int i;
```

```

i=1; p = L->link;          /*初始时, 令 p 指向第一个元素节点, i 为计数器*/
while (p && i < k) {        /*顺指针向后查找, 直到 p 指向第 k 个元素节点或 p 指向空*/
    p = p->link; i++;
}
if (p && i==k) return p;    /*存在第 k 个元素且指针 p 指向该元素节点*/
return NULL;               /*第 k 个元素不存在*/
} /*Find_List*/
  
```

【函数】 单链表的插入运算。

```

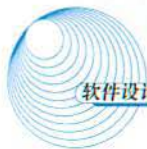
int Insert_List (LinkedList L, int k, int elem) /*L 为带头节点单链表的头指针*/
/*将元素 elem 插入表中的第 k 个元素之前, 若成功则返回 0, 否则返回-1*/
/*将元素 elem 插入第 k 个元素之前等同于将元素插入在第 k-1 个元素之后*/
{
    LinkedList p,q;
    if (k == 1) p = L;          /*元素 elem 插入在第 1 个元素之前*/
    else p = Find_List(L, k-1); /*查找表中的第 k-1 个元素并令 p 指向该元素节点*/
    if (!p) return -1;          /*表中不存在第 k-1 个元素*/
    q = (NODE *)malloc(sizeof(NODE));
    if (!q) return -1;
    q->data = elem;
    q->link = p->link; p->link = q; /*元素 elem 插入第 k-1 个元素之后*/
    return 0;
} /* Insert_List */
  
```

【函数】 单链表的删除运算。

```

int Delete_List (LinkedList L, int k) /*L 为带头节点单链表的头指针*/
/*删除表中的第 k 个元素节点, 若成功返回 0, 否则, 返回-1*/
/*删除第 k 个元素, 相当于令第 k-1 个元素节点中的指针指向第 k+1 个元素的节点*/
{
    LinkedList p,q;
    if (k == 1) p = L;          /*删除第一个元素节点*/
    else p = Find_List(L, k-1); /*查找表中的第 k-1 个元素并令 p 指向该元素节点*/
    if (!p) return -1;          /*表中不存在第 k-1 个元素*/
    q = p->link;                /*令 q 指向第 k 个元素节点*/
    p->link = q->link; free(q); /*删除节点*/
    return 0;
} /* Delete_List */
  
```

线性表采用链表作为存储结构时, 其缺点是不能进行数据元素的随机访问, 优点是插入和删除操作不需要移动元素。根据节点中指针信息的实现方式, 还有其他几种链表结构:



(1) 双向链表: 每个节点包含两个指针, 分别指出当前节点元素的直接前驱和直接后继。从表中任意的元素节点出发, 可从两个方向上遍历链表。

(2) 循环链表: 在单向链表(或双向链表的基础上), 令表尾节点的指针指向表中的第一个节点, 构成循环链表, 其特点是从表中任意节点开始遍历整个链表。

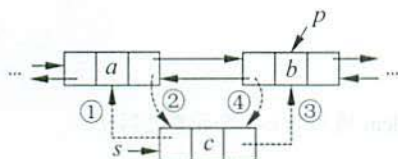
(3) 静态链表: 借助数组来描述线性表的链式存储结构。

若双向链表中节点的 front 和 next 指针域分别指示当前节点的直接前驱和直接后继, 则在双向链表中插入节点 s 时指针的变化情况如图 8-4 (a) 所示, 其操作过程可表示为:

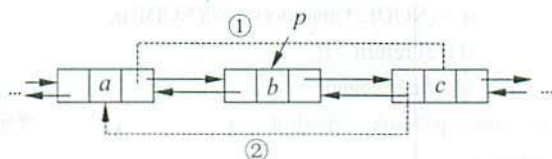
- ① $s \rightarrow \text{front} = p \rightarrow \text{front};$
- ② $p \rightarrow \text{front} \rightarrow \text{next} = s; /* 或者表示为 $s \rightarrow \text{front} \rightarrow \text{next} = s; * /$$
- ③ $s \rightarrow \text{next} = p;$
- ④ $p \rightarrow \text{front} = s;$

在双向链表中删除节点时指针的变化情况如图 8-4 (b) 所示, 其操作过程可表示为:

- ① $p \rightarrow \text{front} \rightarrow \text{next} = p \rightarrow \text{next};$
- ② $p \rightarrow \text{next} \rightarrow \text{front} = p \rightarrow \text{front};$



(a) 双向链表中插入节点



(b) 双向链表中删除节点

图 8-4 双向链表插入和删除节点时的指针变化示意图

8.1.2 栈和队列

栈和队列是软件设计中常用的两种数据结构, 它们的逻辑结构和线性表相同。其特点在于运算受到了限制: 栈按“后进先出”的规则进行操作, 队列按“先进先出”的规则进行操作, 故称运算受限的线性表。

1. 栈

1) 栈的定义及基本运算

(1) 栈的定义: 栈是只能通过访问它的一端来实现数据存储和检索的一种线性数据结构。换句话说, 栈的修改是按先进后出的原则进行的。因此, 栈又称为先进后出 (FILO, 或后进先出) 的线性表。在栈中进行插入和删除操作的一端称为栈顶 (top), 相应地, 另一端称为栈底 (bottom)。不含数据元素的栈称为空栈。

(2) 栈的基本运算

① 初始化栈 $\text{InitStack}(S)$: 创建一个空栈 S 。

② 判栈空 $\text{StackEmpty}(S)$: 当栈 S 为空栈时返回“真”值, 否则返回“假”值。

③ 入栈 $\text{Push}(S, x)$: 将元素 x 加入栈顶, 并更新栈顶指针。

④ 出栈 $\text{Pop}(S)$: 将栈顶元素从栈中删除, 并更新栈顶指针。若需要得到栈顶元素的值, 可将 $\text{Pop}(S)$ 定义为一个返回栈顶元素值的函数。

⑤ 读栈顶元素 $\text{Top}(S)$: 返回栈顶元素的值, 但不修改栈顶指针。

利用上述 5 种基本运算, 可以实现基于栈结构的应用问题的求解。

2) 栈的存储结构

(1) 顺序存储: 栈的顺序存储指用一组地址连续的存储单元依次存储自栈顶到栈底的数据元素, 同时附设指针 top 指示栈顶元素的位置。采用顺序存储结构的栈也称为顺序栈。在顺序存储方式下, 需要预先定义(或申请)栈的存储空间, 也就是说, 栈空间的容量是有限的。因此, 在顺序栈中, 当一个元素入栈时, 需要判断是否栈满(栈空间中没有空闲单元), 若栈满, 则元素入栈会发生上溢现象。

(2) 栈的链式存储: 为了克服顺序存储的栈可能存在上溢的不足, 可以用链表存储栈中的元素。用链表作为存储结构的栈也称为链栈。由于栈中元素的插入和删除仅在栈顶一端进行, 因此不必设置头节点, 链表的头指针就是栈顶指针。链栈的表示如图 8-5 所示。

(3) 栈的应用: 栈的典型应用包括表达式求值、括号匹配等。在计算机语言的实现以及将递归过程转变为非递归过程的处理中, 栈有重要的作用。

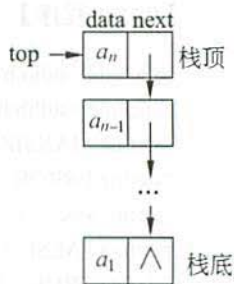


图 8-5 链栈示意图

【例 8.1】 迷宫问题。

求解迷宫中从入口到出口的路径是一个经典的程序设计问题。迷宫可以用一个二维数组表示, 如图 8-6 (a) 所示。

迷宫问题的求解方法是: 从入口出发, 顺某个方向向前探索, 若能走通, 再继续前进; 否则, 沿原路返回, 换一个方向继续探索, 直至所有可能的通路都探索到为止。为了保证在任何位置上都能沿原路退回, 显然需要一个先进后出的栈结构来保存从入口到当前位置的路径。

设迷宫的数据已经存放在文本文件中, 如图 8-6 (c) 所示, 第一行的两个整数分别表示迷宫的行数和列数, 第二行的前两个整数表示入口位置, 后两个整数表示出口位置。从第三行开始的数据表示迷宫, 其中 1 表示墙、0 表示通路。程序中用一个二维字符数组表示迷宫, 以字符“0”和“1”分别表示通路及墙。迷宫的出口和入口可以设定在迷宫中的任何位置。为了避免搜索过程中出现下标越界问题, 在迷宫的四周分别加上一堵墙, 如图 8-6 (b) 所示。

在寻找出口的过程中, 用一个栈 S 记录从入口到当前位置所走的路径。为了避免因为尝试

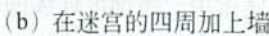


图 8-6 迷宫

【例 8.1 程序】 求解迷宫问题，找出从入口到出口的一条路径。

```
/*迷宫中位置的坐标*/
```

```

S->elem = (SElemType *)malloc(MAXSIZE*MAXSIZE*sizeof(SElemType));
if (!S->elem) return ERROR;
S->top = 0; return OK;
/*InitStack*/
int Push(Stack *S, SElemType e){          /*元素 e 入栈*/
    if (S->top >= MAXSIZE*MAXSIZE) return ERROR;
    S->elem[S->top++] = e; return OK;
}/*Push*/
int Pop(Stack *S, SElemType *e){          /*栈顶元素出栈, 由 e 带回栈顶元素*/
    if (S->top <= 0) return ERROR;
    *e = S->elem[--S->top]; return OK;
}/*Pop*/
int Empty(Stack S){                      /*若栈为空, 函数返回 TRUE;否则返回 FALSE */
    if (S.top == 0) return TRUE;
    return FALSE;
}/*Empty*/
int createMaze(char *filename, Position *startpos, Position *endpos){
/*从文件 filename 中读入数据创建迷宫, 由参数带回入口位置和出口位置*/
    FILE *fp;
    int i,j,rows,cols,temp;
    Position start,end;
    fp = fopen(filename,"r");
    if (!fp)
    { printf("open file %s error!\n",filename);
    return ERROR;
    }
    if (!feof(fp))
    { fscanf(fp,"%d %d",&rows,&cols);          /*读入迷宫的行数和列数*/
      fscanf(fp,"%d %d",&start.x,&start.y);    /*读入迷宫的入口位置*/
      fscanf(fp,"%d %d",&end.x,&end.y);        /*读入迷宫的出口位置*/
    }
    for(i = 1;i <= rows;i++) /*读入迷宫数据*/
        for(j = 1;j <= cols;j++)
            { fscanf(fp,"%d",&temp);maze[i][j]=48+temp;}
    fclose(fp);
/*在迷宫四周加墙壁*/
    for(i = 0;j = 0;i <= rows+1;i++) maze[i][j] = '1';
    for(i = 0;j = cols+1;i <= rows+1;i++) maze[i][j] = '1';
    for(i = 0;j = 0;j <= cols+1;j++) maze[i][j] = '1';
    for(i = rows+1,j = 0;j <= cols+1;j++) maze[i][j] = '1';

```




```
*startpos = start; *endpos = end;
return OK;
}/*createMaze*/
int canPass(Position curpos)
{ if (maze[curpos.x][curpos.y] == '0') return TRUE;
  return FALSE;
}/*canPass*/
void markPos(Position curpos, MarkTag tag){ /*为已经探索过的位置加标记*/
    switch(tag){
    case YES: maze[curpos.x][curpos.y] = '.'; break; /*路径标记*/
    case NO:  maze[curpos.x][curpos.y] = '#'; break; /*死胡同标记*/
    }/*switch*/
}/*markPos*/
Position nextPos(Position curpos, Direction dir)
/*根据当前的位置坐标和下一步要探索的方向 dir 求下一步要走的位置坐标*/
{ Position nextpos;
  switch(dir){
    case RIGHT: nextpos.x = curpos.x; nextpos.y = curpos.y+1; break;
    case DOWN: nextpos.x = curpos.x+1; nextpos.y = curpos.y; break;
    case LEFT: nextpos.x = curpos.x; nextpos.y = curpos.y-1; break;
    case UP: nextpos.x = curpos.x-1; nextpos.y = curpos.y; break;
  }/*switch*/
  return nextpos;
}/*nextPos*/
Direction nextDir(Direction dir){
  switch(dir){ /*按照 RIGHT、DOWN、LEFT、UP 的次序进行试探路径*/
    case RIGHT: return DOWN;
    case DOWN: return LEFT;
    case LEFT: return UP;
  }/*switch*/
}/*nextDir*/
int Solve(Stack *S, Position start, Position end){
/*若迷宫中存在从入口 start 到出口 end 的通道, 则求得一条存放在栈 S 中, 并返回 TRUE*/
/*若迷宫中不存在从入口 start 到出口 end 的通道, 则返回 FALSE*/
    Position curpos;
    SElemType e;
    int curstep=1;
    if (InitStack(S)==ERROR) return FALSE;
    curpos = start;
    do{
```

```

if (canPass(curpos)){
    markPos(curpos,YES);
    e.order = curstep; e.seat = curpos; e.di = RIGHT;
    Push(S,e);
    if(curpos.x == end.x && curpos.y == end.y)
        return TRUE; /*找到从入口到出口的通道*/
    curpos = nextPos(curpos,RIGHT);
    curstep++;
}
else{
    if(!Empty(*S)){
        if (Pop(S,&e)==ERROR) return FALSE;
        while(e.di == UP && !Empty(*S)){
            /*4个方向都试探过，没有找到通路也不能继续探索，则回溯*/
            curpos = e.seat; markPos(curpos,NO);
            if (Pop(S,&e)==ERROR) return FALSE;
        }/*while*/

        if(e.di!=UP){ /*4个方向还没有试探完*/
            e.di = nextDir(e.di);
            Push(S,e); /*换下一个方向探索*/
            curpos = nextPos(e.seat,e.di);
        }/*if*/
    }
}
} while(!Empty(*S));
return FALSE;
}/*Solve*/

void main(void)
{
    Position startPos,endPos;
    Stack path;
    SElemType e;
    char *fname="in.txt";
    if (createMaze(fname,&startPos,&endPos)==ERROR) return;
    Solve(&path,startPos,endPos);
    while (!Empty(path)) { /*输出口到入口的路径*/
        Pop(&path,&e);
        printf("(%d,%d)\n",e.seat.x,e.seat.y);
    }
}

```


2. 队列

1) 队列的定义及基本运算

(1) 队列的定义。队列是一种先进先出(FIFO)的线性表,它只允许在表的一端插入元素,而在表的另一端删除元素。在队列中,允许插入元素的一端称为队尾(rear),允许删除元素的一端称为队头(front)。

(2) 队列的基本运算

- 初始化队 InitQueue(Q): 创建一个空的队列 Q。
- 判队空 Empty(Q): 当队列为空时返回“真”值,否则返回“假”值。
- 入队 EnQueue(Q,x): 将元素 x 加入到队列 Q 的队尾,并更新队尾指针。
- 出队 DeQueue(Q): 将队头元素从队列 Q 中删除,并更新队头指针。
- 读队头元素 Frontque(Q): 返回队头元素的值,但不更新队头指针。

2) 队列的存储结构

(1) 队列的顺序存储。队列的顺序存储结构又称为顺序队列,它也是利用一组地址连续的存储单元存放队列中的元素。由于队列中元素的插入和删除限定在表的两端进行,因此设置队头指针和队尾指针,分别指示出当前的队首元素和队尾元素。

设顺序队列 Q 的容量为 6,其队头指针为 front,队尾指针为 rear,头、尾指针和队列中元素之间的关系如图 8-7 所示。

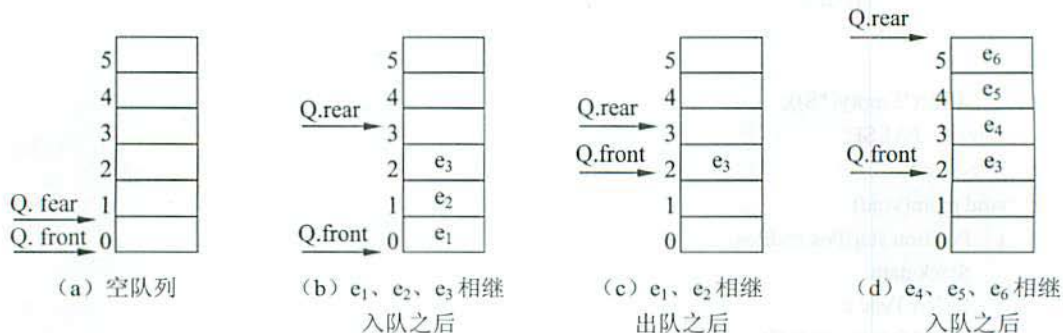


图 8-7 队列的头、尾指针与队列中元素之间的关系

在顺序队列中,为了降低运算的复杂度,元素入队时,只需修改队尾指针,元素出队时,只需修改队头指针。由于顺序队列的存储空间是提前设定的,所以队尾指针会有一个上限值,当队尾指针达到该上限时,就不能只通过修改队尾指针来实现新元素的入队操作了。此时,可通过整除取余运算将顺序队列假想成一个环状结构,如图 8-8 所示,称之为循环队列。

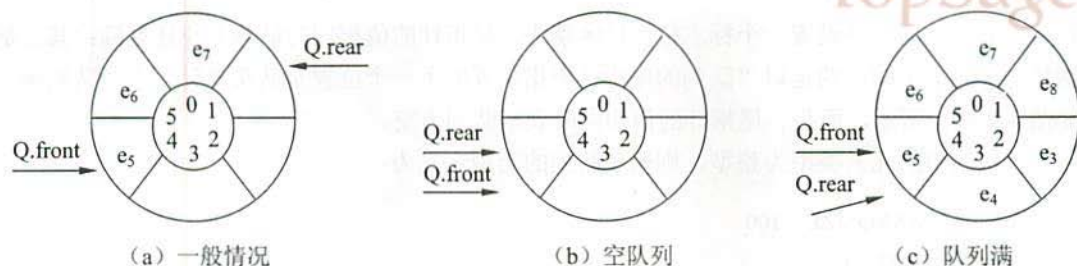


图 8-8 循环队列的头、尾指针示意图

设循环队列 Q 的容量为 MAXSIZE , 初始时队列为空, 且 $Q.\text{rear}$ 和 $Q.\text{front}$ 都等于 0, 如图 8-9 (a) 所示。

元素入队时, 修改队尾指针: $Q.\text{rear} = (Q.\text{rear} + 1) \% \text{MAXSIZE}$; 如图 8-9 (b) 所示。

元素出队时, 修改队头指针: $Q.\text{front} = (Q.\text{front} + 1) \% \text{MAXSIZE}$; 如图 8-9 (c) 所示。

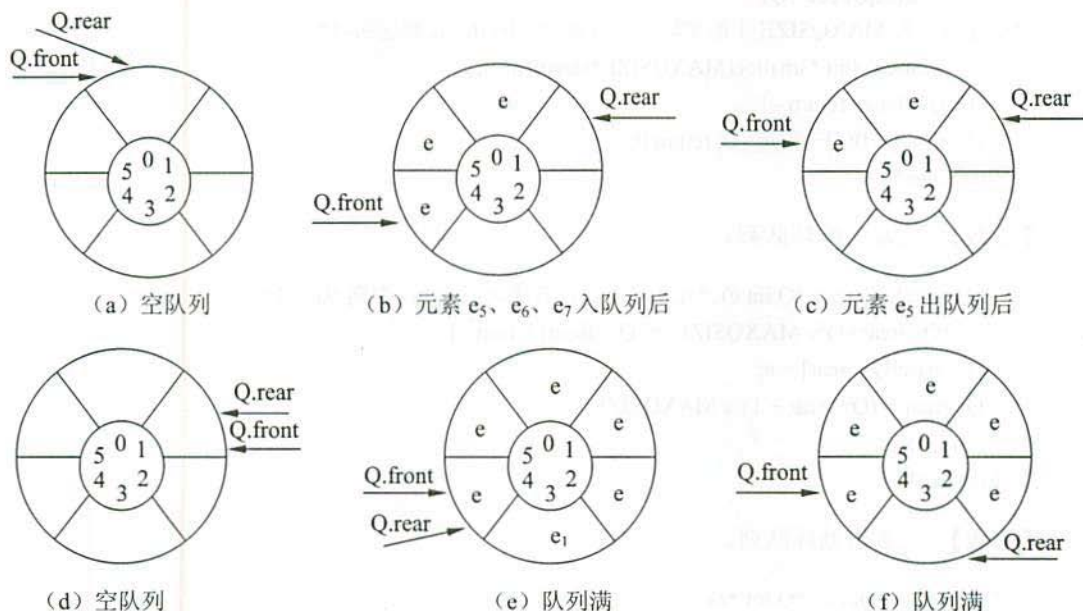


图 8-9 循环队列的头、尾指针示意图

根据出队列操作的定义, 当出队操作导致队列变为空时, 则有 $Q.\text{rear} = Q.\text{front}$, 如图 8-9(d) 所示; 若队列满, 则 $Q.\text{rear} = Q.\text{front}$, 如图 8-9(e)所示。

在队列空和队列满的情况下, 循环队列的队头、队尾指针指向的位置是相同的, 此时仅仅根据 $Q.\text{rear}$ 和 $Q.\text{front}$ 之间的关系无法断定队列的状态。为了区别队空和队满的情况, 可采用两



种处理方式: 其一是设置一个标志位, 以区别头、尾指针的值相同时队列是空还是满; 其二是牺牲一个元素空间, 约定以“队列的尾指针所指位置的下一个位置是队头指针”表示队列满, 如图 8-9 (f) 所示, 而头、尾指针的值相同时表示队列为空。

设队列中的元素类型为整型, 则循环队列的类型定义为:

```
#define MAXQSIZE 100
typedef struct {
    int *base;          /*循环队列的存储空间*/
    int front;          /*队头指针*/
    int rear;           /*队尾指针*/
} SqQueue;
```

【函数】 创建一个空的循环队列。

```
int InitQueue(SqQueue *Q)
/*创建容量为 MAXQSIZE-1 的空队列, 若成功返回 0; 否则返回-1*/
{
    Q->base = (int *)malloc(MAXQSIZE*sizeof(int));
    if (!Q->base) return -1;
    Q->front = 0; Q->rear = 0; return 0;
} /*InitQueue*/
```

【函数】 元素入循环队列。

```
int EnQueue(SqQueue *Q, int e) /*元素 e 入队, 若成功返回 0; 否则返回-1*/
{
    if ((Q->rear+1)%MAXQSIZE == Q->front) return -1;
    Q->base[Q->rear] = e;
    Q->rear = (Q->rear + 1)%MAXQSIZE;
    return 0;
} /*EnQueue*/
```

【函数】 元素出循环队列。

```
int DeQueue(SqQueue *Q, int *e)
/*若队列不空, 则删除队头元素, 由参数 e 带回其值并返回 0; 否则返回-1*/
{
    if (Q->rear == Q->front) return -1;
    *e = Q->base[Q->front];
    Q->front = (Q->front + 1)%MAXQSIZE;
    return 0;
} /*DeQueue*/
```

(2) 队列的链式存储：队列的链式存储也称为链队列。这里为了便于操作，给链队列添加一个头节点，并令头指针指向头节点。因此，队列为空的判定条件是：头指针和尾指针的值相同，且均指向头节点。队列的链式存储结构如图 8-10 所示。

3) 队列的应用

队列结构常用于处理需要排队的场合，如操作系统中处理打印任务的打印队列、离散事件的计算机模拟等。

【例 8.2】离散事件模拟。

假设某银行有 3 个窗口对外接待客户，从早晨银行开门起就不断有客户进入银行。由于每个窗口在每个时刻只能接待一位客户，因此在客户比较多时需要排队。对于刚进入银行的客户，如果某个窗口的业务员正空闲，则可上前办理业务；反之，若 3 个窗口前均有客户，他便会排在人数最少的队伍后面。下面的程序模拟客户在银行的这种活动并计算一天中客户在银行的平均逗留时间。

为了计算所有客户在银行的平均逗留时间，需要掌握每个客户到达银行和离开银行的时间，后者减去前者即为每个客户在银行的逗留时间。用当天的客户数除以所有客户的逗留时间之和便是所求的平均逗留时间。称客户到达银行和离开银行这两个时刻发生的事情为“事件”，则整个模拟程序将按照事件发生的先后顺序进行处理。

程序中要处理的事件有两类：一类是客户到达事件，另一类是客户离开事件。前一类事件发生的时刻随客户到来自然形成；后一类事件发生的时刻则由客户办理业务所需时间和等待时间而定。由于程序按事件发生时刻的先后顺序进行处理，则事件表应该是有序表，其主要操作是插入和删除事件。

模拟程序中需要的另一种数据结构是表示客户排队的队列，由于前面假设银行有 3 个窗口，因此程序中需要 3 个队列，队列中存储有关客户的主要信息：客户到达的时刻和客户办理业务所需的时间。每个队列的队头客户即为正在办理业务的客户，他办完业务离开队列时就是即将发生的客户离开事件的时刻，也就是说，每个队头客户都存在一个将要驱动的客户离开事件。因此，在任何时刻即将发生的事件只有下列 4 种可能：一是新的客户到达；二是 1 号窗口客户离开；三是 2 号窗口客户离开；四是 3 号窗口客户离开。所以在该模拟程序中只需要两种数据类型：有序链表和队列。

客户到达事件的处理：在实际的银行中，客户到达的时刻及其办理业务所需时间都是随机的，因此在模拟程序中用随机数来代替。不失一般性，假设第一个客户进门的时刻为 0，即为模拟程序处理的第一个事件，之后每个客户到达的时刻在前一客户到达时设定。因此，在客户到达事件发生时需产生两个随机数：其一为该时刻达到的客户办理业务所需要的时间 *durtime*；其二为下一客户将要到达的时间间隔 *interval*，假设当前事件的发生事件为 *occurTime*，则下一个



图 8-10 链队列示意图



客户到达事件发生的时刻为 $\text{occurTime} + \text{interval}$ 。由此应产生一个新的客户到达事件插入事件表：刚到达的客户则应插入到当前所包含元素最少的队列中；若该队列在插入前为空，则还应产生一个客户离开事件插入事件表。

客户离开事件的处理：首先计算要离开的客户在银行逗留的时间，然后从队列中删除该客户，同时查看队列是否为空，若不空则设定一个新的客户离开事件。

离散事件模拟过程的流程图如图 8-11 所示。

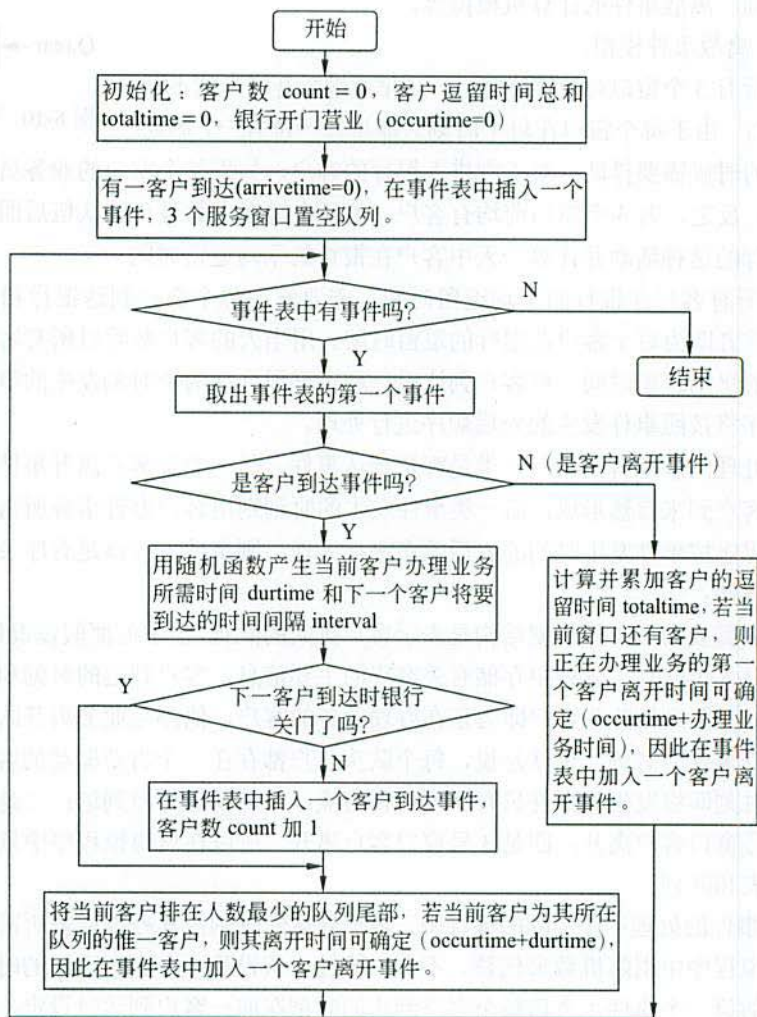


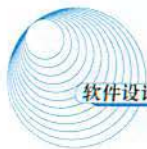
图 8-11 离散事件模拟流程图

【例 8.2 程序】 离散事件模拟程序。

```

#include <stdio.h>
#include <stdlib.h>
#define CLOSETIME      1000      /*银行终止营业时间*/
#define M              3        /*服务窗口数*/
typedef struct QueueNode {      /*队列节点*/
    int arrivetime;            /*客户到达时间*/
    int duration;              /*客户办理业务所需的时间*/
    struct QueueNode *next;
} QueueNode;
typedef struct QueueHeader {
    QueueNode *front,*rear;    /*队列的头、尾指针*/
    int length;                /*队列长度*/
} QueueHeader;
typedef struct EventNode {      /*事件链表节点类型*/
    int occurTime,eventType;    /*事件发生的时刻和事件类型*/
    struct EventNode *next;
} EventNode;
typedef struct EventList {
    struct EventNode *front,*rear; /*事件链表头、尾指针*/
    int eventNum;                  /*事件数目*/
} EventList;
QueueHeader q[M+1];              /*M 为服务窗口数量*/
EventList *eventlst;
int seed = 300;
int generate(EventNode **p)
{
    *p=(EventNode *)malloc(sizeof(EventNode));
    if (!(*p)) { printf("allocation error!"); return -1;}
    return 0;
} /* generate */

void InsertEvent(EventList *eventlst,int occurTime,int etype)
/*下一个客户的到达事件加入事件链表*/
{
    EventNode *p,*q,*qre;
    p = (EventNode *)malloc(sizeof(EventNode));
    p->occurTime = occurTime; p->eventType=etype;
    eventlst->eventNum++;
  
```

```
q = eventlst->front;   qre = NULL;
while (q && occurTime>q->occurTime)
    { qre = q;  q = q->next; }
if (!qre) {p->next=eventlst->front; eventlst->front=p;}
else {p->next = q; qre->next  =p;}
if (eventlst->eventNum == 1) eventlst->rear = p;
}/* InsertEvent */

void DeleteEvent(EventList *eventlst,EventNode **event)
/*删除事件链表中的第一个事件节点并返回该事件对应的数据*/
{  EventNode *p;
  p = eventlst->front;
  eventlst->front = p->next;
  if (--eventlst->eventNum < 1) eventlst->rear = eventlst->front;
  *event = p;
}/* DeleteEvent */

void EnQueue(QueueHeader *swindow,int t1,int t2) /*当前客户加入服务窗口*/
{  QueueNode *p;
  p=(QueueNode *)malloc(sizeof(QueueNode));
  p->arrivetime = t1; p->duration = t2;
  swindow->length++;
  if (swindow->length == 1)
      { p->next = NULL;  swindow->front = p; swindow->rear = p; }
  else  p->next = swindow->rear->next;
      {  swindow->rear->next = p;  swindow->rear = p; }
}/* EnQueue */

void DeQueue(QueueHeader *swindow,QueueNode *f) /*删除指定窗口的排头客户*/
{ QueueNode *p;
  if (swindow->length>0)
      { f->arrivetime = swindow->front->arrivetime;
        f->duration = swindow->front->duration;
        p = swindow->front;  swindow->front = swindow->front->next;
        swindow->length--;
        if (swindow->length == 0) swindow->rear = NULL;
        free(p);
      }
}
```

```

/*DeQueue*/
void random(int *durtime,int *intervaltime) /*产生两个随机数*/
{
    *durtime=rand()+1;    *intervaltime=rand()+1;
}/* random */

int rand() /*随机数发生函数*/
{
    int k = (int)(((double)seed/32767)*10000)%29;
    seed = (13107*seed+6553) % 32767;
    return k;
}/* rand */

int Minlength( ) /*取当前人数最少的服务窗口号码*/
{
    int minx,k,i;
    minx = q[1].length;k=1;
    for(i = 2;i< M+1;++i)
        if (q[i].length < minx) { minx = q[i].length; k = i;}
    return k;
}/* Minlength */

void main()
{
    int durtime,interval,totaltime = 0;
    int i,c,count = 0; /*count 用于计算客户数*/
    QueueNode f;
    EventNode *pe,*event;
    eventlst=(EventList *)malloc(sizeof(EventList));
    if (eventlst == NULL) return;
    if (generate(&pe) == -1) return; /*产生一个事件节点*/
    pe->occurTime = 0; pe->eventType = 0; /*设定第一个客户到达事件*/
    pe->next = NULL;
    eventlst->front = pe; eventlst->rear = pe; /*第一个客户到达事件插入事件表*/
    eventlst->eventNum = 1;
    for(i=1;i<M+1;i++) /*置当前各服务窗口队列为空*/
    {
        q[i].front = q[i].rear=NULL;
        q[i].length = 0;
    }
    while(eventlst->eventNum > 0) /*事件表不空*/
    {
        DeleteEvent(eventlst,&event); /*取事件表中的第一个事件节点*/
        if (event->eventType == 0) /*处理客户到达事件*/
            { printf("A client entered! ");

```



```

count++;                                /*累计客户数*/
/*产生当前客户所需的服务时间和下一客户到达的时间间隔*/
random(&durtime,&interval);
if ((event->occurTime+interval) < CLOSETIME) /*下一个客户到达事件加入事件表*/
    InsertEvent(eventlst,event->occurTime+interval,0);
c = Minlength();                        /*取当前人数最少的窗口号码*/
EnQueue(&q[c],event->occurTime,durtime); /*当前客户加入 c 号窗口*/
printf("Insert to window %d",c);
printf(" (arrive time:%d duration:%d)\n",event->occurTime,durtime);
if (q[c].length == 1)
    InsertEvent(eventlst,event->occurTime+durtime,c);
}
else                                    /*处理客户离开事件*/
{ i = event->eventType;                  /*i 号窗口的客户要离开*/
  DeQueue(&q[i],&f);                    /*删除 i 号窗口的排头客户*/
  printf("service window %d: A client left!\n",i);
  totaltime += event->occurTime - f.arrivetime; /*累加客户的逗留时间*/
  if (q[i].length != 0)                  /*设定 i 号窗口的一个离开事件并插入事件链表*/
      InsertEvent(eventlst,event->occurTime+q[i].front->duration,i);
}
}
printf("\nTotal Customers:%d\n",count);
printf("Average duration: %d\n",totaltime/count);
}

```

8.1.3 串

串(字符串)是一种特殊的线性表,它的数据元素仅由一个字符组成。计算机中非数值问题处理的对象经常是字符串数据,如在汇编和高级语言的编译程序中,源程序和目标程序都是字符串数据;在事务处理程序中,姓名、地址等一般也是作为字符串处理的。另外串还具有自身的特性,常常把一个串作为一个整体来处理。这里介绍串的定义、基本运算、存储结构及串的模式匹配算法。

1. 串的定义及基本运算

1) 串的定义

串是仅由字符构成的有限序列,是取值范围受限的线性表。一般记为 $S = 'a_1a_2...a_n'$, 其中 S 是串名,单引号括起来的字符序列是串值。

2) 串的几个基本概念

- 空串: 长度为零的串, 空串不包含任何字符。
- 空格串: 由一个或多个空格组成的串。虽然空格是一个空白符, 但它也是一个字符, 计算串长度时要将其计算在内。
- 子串: 由串中任意长度的连续字符构成的序列称为子串。含有子串的串称为主串。子串在主串中的位置指子串首次出现时, 该子串的第一个字符在主串的位置。空串是任意串的子串。
- 串相等: 指两个串长度相等且对应位置上的字符也相同。
- 串比较: 两个串比较大小时以字符的 ASCII 码值作为依据。比较操作从两个串的第一个字符开始进行, 字符的 ASCII 码值大者所在的串为大; 若其中一个串先结束, 则以串长较大者为大。

3) 串的基本操作

- 赋值操作 $\text{StrAssign}(s, t)$: 将串 t 的值赋给串 s 。
- 连接操作 $\text{Concat}(s, t)$: 将串 t 接续在串 s 的尾部, 形成一个新串。
- 求串长 $\text{StrLength}(s)$: 返回串 s 的长度。
- 串比较 $\text{StrCompare}(s, t)$: 比较两个串的大小。返回值 -1、0 和 1 分别表示 $s < t$ 、 $s = t$ 和 $s > t$ 3 种情况。
- 求子串 $\text{SubString}(s, \text{start}, \text{len})$: 返回串 s 中从 start 开始的、长度为 len 的字符序列。

以上 5 种最基本的串操作构成了串的最小操作子集, 利用它们就可以实现串的其他运算。

2. 串的存储结构

1) 串的静态存储: 定长存储结构

串的静态存储结构就是串的顺序存储结构, 用一组地址连续的存储单元来存储串值的字符序列。由于串中的元素为字符, 所以可通过程序语言提供的字符数组定义串的存储空间, 也可以根据串长的需要动态申请字符串的空间。

2) 串的链式存储: 块链

字符串也可以采用链表作为存储结构, 当用链表存储串中的字符时, 每个节点中可以存储一个字符, 也可以存储多个字符, 此时要考虑存储密度问题。

在链式存储结构中, 节点大小的选择和顺序存储方法中数组空间大小的选择一样重要, 它直接影响对串处理的效率。

3. 串的模式匹配

子串的定位操作通常称为串的模式匹配, 它是各种串处理系统中最重要、最基本的运算之一。子串

也称为模式串。

1) 朴素的模式匹配算法

该算法也称为布鲁特-福斯算法,其基本思想是从主串的第一个字符起与模式串的第一个字符比较,若相等,则继续逐个字符进行后续的比较,否则从主串的第二个字符起与模式串的第一个字符重新比较,直至模式串中每个字符依次和主串中的一个连续的字符序列相等时为止,此时称为匹配成功,否则称为匹配失败。

【函数】 以字符数组存储字符串,实现朴素的模式匹配算法。

```
int Index(char S[],char T[],int pos)
/*查找并返回模式串 T 在主串 S 中从 pos 开始的位置(下标),若 T 不是 S 的子串,则返回-1*/
{
    int i,j,slen,tlen;
    i = pos; j = 0; /*i,j 分别用于指示出主串字符和模式串字符的位置*/
    slen = strlen(S); tlen = strlen(T); /*计算主串和模式串的长度*/
    while (i < slen && j < tlen){
        if (S[i] == T[j]) {i++;j++;}
        else {i = i - j + 1; /*主串字符的位置指针回退*/
              j = 0;
            }
    } /*while*/
    if (j >= tlen)
        return i - tlen;
    return -1;
} /* Index */
```

假设主串和模式串的长度分别为 n 和 m ,位置序号从 1 开始计算,下面分析朴素模式匹配算法的时间复杂度,位置序号从 1 开始计算:设从主串的第 i 个位置开始与模式串匹配成功,在前 $i-1$ 趟匹配中,每趟不成功的匹配都是模式串的第一个字符与主串中相应的字符不相同,则在前 $i-1$ 趟匹配中,字符的比较共进行了 $i-1$ 次,因第 i 趟成功匹配的字符比较次数为 m ,所以总的字符比较次数为 $(i-1+m)$ 且 $1 \leq i \leq n-m+1$ 。若在这 $n-m+1$ 个起始位置上匹配成功的概率相同,则在最好情况下,匹配成功时字符间的平均比较次数为:

$$\sum_{i=1}^{n-m+1} p_i (i-1+m) = \frac{1}{n-m+1} \sum_{i=1}^{n-m+1} (i-1+m) = \frac{1}{2}(n+m)$$

因此,在最好情况下匹配算法的时间复杂度为 $O(n+m)$ 。

而在最坏的情况下,每一趟不成功的匹配都是模式串的最后一个字符与主串中相应的字符不相等,则主串中新一趟的起始位置为 $i-m+2$,若设第 i 趟匹配时成功,则前 $i-1$ 趟不成功的匹配中,每趟都比较了 m 次,总共比较了 $i \times m$ 次。因此,最坏情况下的平均比较次数为

$$\sum_{i=1}^{n-m+1} p_i(i \times m) = \frac{m}{n-m+1} \sum_{i=1}^{n-m+1} i = \frac{1}{2} m(n+m)$$

由于 $n \geq m$, 所以该算法在最坏情况下的时间复杂度为 $O(n \times m)$ 。

2) 改进的模式匹配算法

改进的模式匹配算法又称为 KMP 算法, 其改进之处在于: 每当匹配过程中出现相比较的字符不相等时, 不需要回溯主串的指针, 而是利用已经得到的“部分匹配”的结果, 将模式串向后“滑动”尽可能远的距离, 再继续进行比较。此算法可在 $O(n+m)$ 的时间内完成。

设模式串为 $'p_1 \cdots p_m'$, KMP 匹配算法的思想是: 当模式串中的字符 p_j 与主串中相应的字符 S_i 不相等时, 因其前 $j-1$ 个字符 $'p_1 \cdots p_{j-1}'$ 已经获得了匹配, 所以若模式串中的 $'p_1 \cdots p_{k-1}'$ 和 $'p_{j-k+1} \cdots p_{j-1}'$ 相同, 这时可令 p_k 与 S_i 进行比较, 从而使 i 无须向前回退。

在 KMP 算法中, 依据模式串的 next 函数值实现子串的滑动。若令 $\text{next}[j]=k$, 则 $\text{next}[j]$ 表示当模式串中的 p_j 与主串中相应字符不相等时, 令模式串的 p_k 与主串的相应字符进行比较。

next 函数的定义如下:

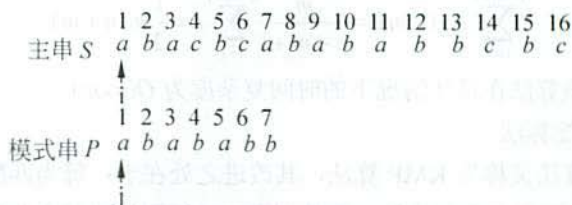
$$\text{next}[j] = \begin{cases} 0 & \text{当 } j=1 \text{ 时} \\ \max\{k \mid 1 < k < j \text{ 且 } 'p_1 \cdots p_{k-1}' = 'p_{j-k+1} \cdots p_{j-1}'\} & \\ 1 & \text{其他情况} \end{cases}$$

【函数】求模式串的 next 函数。

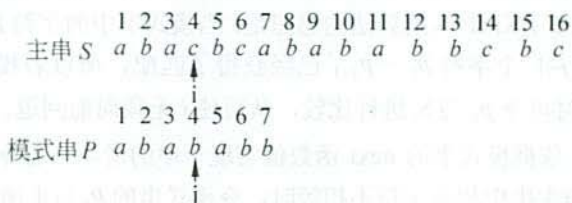
```
void Get_next(char *p, int next[]) /*求模式串 p 的 next 函数值并存入数组 next*/
{
    int i, j, slen;
    slen = strlen(p); i = 0;
    next[0] = -1; j = -1; /*若字符数组的下标从 0 开始, 则相应的 next 函数值减 1*/
    while(i < slen) {
        if(j == -1 || p[i] == p[j]) {++i; ++j; next[i] = j;}
        else j = next[j];
    } /*while*/
} /*Get_next*/
```

例如, 设主串为“abcbabcbcabababbcb”, 模式串为“abababb”, 则 KMP 算法的匹配过程如图 8-12 所示。先求得模式串的 next 函数值如下所示。

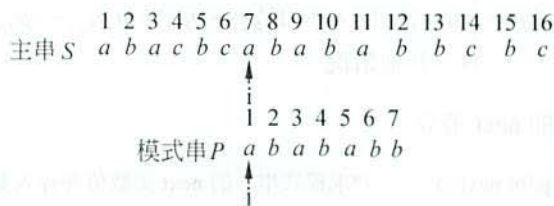
j	1	2	3	4	5	6	7
模式串	a	b	a	b	a	b	b
$\text{next}[j]$	0	1	1	2	3	2	3



(a) i 是主串指针的下标, j 是模式串指针的下标



(b) 若 $S[i]=P[j]$, 则 $i++$, $j++$; 否则, $j=\text{next}[j]$, 即 $S[i]$ 与 $P[\text{next}[j]]$ 比较, 重复该步, 若 j 等于 0, 则令 $i++$, $j=1$



(c) 重复 (b) 直到完全匹配或到达主串结尾匹配失败

图 8-12 KMP 模式匹配过程示意图

【函数】 KMP 模式匹配算法, 模式串第一个字符的下标为 0。

```
int Index_KMP(char *s, char *p, int pos, int next[])
/*利用模式串 p 的 next 函数, 求 p 在主串 s 中从第 pos 个字符开始的位置*/
/*若匹配成功则返回模式串在主串中的位置(下标), 否则返回-1*/
{ int i, j, slen, plen;
  i = pos - 1;
  j = -1;
  slen = strlen(s); plen = strlen(p);
  while (i < slen && j < plen) {
    if (j == -1 || s[i] == p[j]) { ++i; ++j; }
    else j = next[j];
  }
}
```

```

    }/*while*/
    if (j >= plen) return i - plen;
    else return -1;
  }/*Index_KMP*/

```

8.2 数组、矩阵和广义表

数组与广义表可看作是线性表的推广,其特点是数据元素仍然是一个表。这里讨论多维数组的逻辑结构和存储结构、特殊矩阵和矩阵的压缩存储、广义表的逻辑结构、存储结构和基本运算。

8.2.1 数组

1. 数组的定义及基本运算

1) 数组的定义

数组是定长线性表在维数上的扩张,即线性表中的元素又是一个线性表。 n 维数组是一种“同构”的数据结构,其每个数据元素类型相同,结构一致。

设有 n 维数组 $a[b_1, b_2, \dots, b_n]$, 其每一维的下界都为 1, b_i 是第 i 维的上界。从数据结构的逻辑关系角度来看, a 中的每个元素 $a[j_1, j_2, \dots, j_n]$ ($1 \leq j_i \leq b_i$) 都被 n 个关系所约束。在每个关系中,除第一个和最后一个元素外,其余元素都只有一个直接后继和一个直接前驱。因此就单个关系而言,这 n 个关系仍是线性的。

以二维数组 $A[m, n]$ 为例,我们可以把它看成是一个定长的线性表,它的每个元素也是一个定长线性表。

$$A_{m \times n} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

A 可看成一行向量形式的线性表:

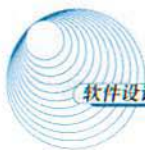
$$A_{mn} = [[a_{11}a_{12} \cdots a_{1n}], [a_{21}a_{22} \cdots a_{2n}], \cdots, [a_{m1}a_{m2} \cdots a_{mn}]];$$

或列向量形式的线性表:

$$A_{mn} = [[a_{11}a_{21} \cdots a_{m1}], [a_{12}a_{22} \cdots a_{m2}], \cdots, [a_{1n}a_{2n} \cdots a_{mn}]]$$

2) 数组结构的特点

(1) 数据元素数目固定。一旦定义了一个数组结构,就不再有元素的增减变化。



(2) 数据元素具有相同的类型。

(3) 数据元素的下标关系具有上下界的约束且下标有序。

3) 数组的两个基本运算

(1) 给定一组下标, 存取相应的数据元素。

(2) 给定一组下标, 修改相应的数据元素中某个数据项的值。

几乎所有的程序设计语言都提供了数组类型。实际上, 在语言中把数组看成是具有共同名字的同类型多个变量的集合。

2. 数组的顺序存储

由于数组一般不作插入和删除运算, 也就是说, 一旦定义了数组, 则结构中的数据元素个数和元素之间的关系就不再发生变动, 因此数组适合于采用顺序存储结构。

由于计算机的内存结构是一维线性的, 因此存储多维数组时必须按某种方式进行降维处理, 即将数组元素排成一个线性序列, 这就产生了次序约定问题。因为多维数组是由较低一维的数组定义的, 依次类推, 通过这种递推关系将多维数组的数据元素排成一个线性序列。

对于数组, 一旦确定了它的维数和各维的长度, 便可为它分配存储空间。反之, 只要给出一组下标便可求得相应数组元素的存储位置, 也就是说, 在数据的顺序存储结构中, 数据元素的位置是其下标的线性函数。

二维数组的存储结构可分为以行为主序和以列为主序的两种方法, 如图 8-13 所示。

设每个数据元素占用 L 个单元, m 、 n 为数组的行数和列数, $\text{Loc}(a_{11})$ 表示元素 a_{11} 的地址, 那么以行为主序优先存储的地址计算公式为:

$$\text{Loc}(a_{ij}) = \text{Loc}(a_{11}) + ((i-1) \times n + (j-1)) \times L$$

同样, 以列为主序优先存储的地址计算公式为:

$$\text{Loc}(a_{ij}) = \text{Loc}(a_{11}) + ((j-1) \times m + (i-1)) \times L$$

推广至多维数组的存储, 按下标顺序存储表示先排最右的下标, 从右向左直到排到最左下标, 而逆下标顺序则正好相反。

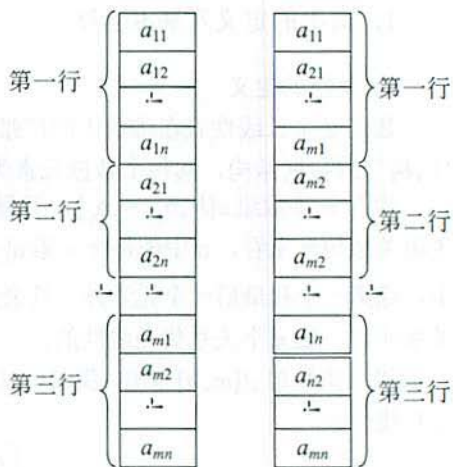


图 8-13 二维数组的两种存储方式

8.2.2 矩阵

矩阵是很多科学与工程计算问题中研究的数学对象。在数据结构中,我们感兴趣的是如何存储矩阵中的元素,从而使矩阵的各种运算能有效地进行。

在一些矩阵中,存在很多值相同的元素或者是零元素。为了节省存储空间,可以对这类矩阵进行压缩存储。压缩存储的含义是为多个值相同的元素只分配一个存储单元,对零元素不分配存储单元。假如值相同的元素或零元素在矩阵中的分布有一定的规律,则称此类矩阵为特殊矩阵,否则称为稀疏矩阵。

1. 特殊矩阵

若矩阵中元素(或非零元素)的分布有一定的规律,则称之为特殊矩阵。常见的特殊矩阵有对称矩阵、三角矩阵、对角矩阵等。对于特殊矩阵,由于其非零元素的分布有一定的规律,所以可将其压缩存储在一维数组中,并建立起每个非零元素在矩阵中的位置与其在一维数组中的位置之间的对应关系。

若矩阵 $A_{n \times n}$ 中的元素特点为 $a_{ij} = a_{ji}$ ($1 \leq i, j \leq n$), 则称之为 n 阶对称矩阵。

若为对称矩阵中的每一对元素分配一个存储单元,那么就可将 n^2 个元素压缩存储到能存放 $n(n+1)/2$ 个元素的存储空间中。不失一般性,我们以行为主序存储下三角(包括对角线)中的元素。假设以一维数组 $B[n(n+1)/2]$ 作为 n 阶对称矩阵 A 的存储结构,则 $B[k]$ ($1 \leq k \leq n(n+1)/2$) 与矩阵元素 a_{ij} 之间存在着——对应的关系:

$$k = \begin{cases} \frac{i(i-1)}{2} + j & \text{当 } i \geq j \\ \frac{j(j-1)}{2} + i & \text{当 } i < j \end{cases}$$

对角矩阵是指矩阵中的非零元素都集中在以主对角线为中心的带状区域中,即除了主对角线上和直接在对角线上、下方若干条对角线上的元素外,其余的矩阵元素都为零。一个 n 阶的三对角矩阵如图 8-14 所示。

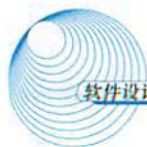
若以行为主序将 n 阶三对角矩阵 $A_{n \times n}$ 的非零元素存储在一维数组 $B[k]$ ($1 \leq k \leq 3*n-2$) 中,则元素位置之间的对应关系为:

$$k = 3*(i-1) - 1 + j - i + 1 + 1 = 2i + j - 2 \quad (1 \leq i, j \leq n)$$

其他特殊矩阵可作类似的计算,这里不再一一说明。

2. 稀疏矩阵

在一个矩阵中,若非零元素的个数远远少于零元素的个数,且非零元素的分布没有规律,



则称之为稀疏矩阵。对于稀疏矩阵,存储非零元素时必须同时存储其位置(即行号和列号),所以三元组 (i, j, a_{ij}) 可唯一确定矩阵A中的一个元素。由此,一个稀疏矩阵可由表示非零元素的三元组及其行、列数唯一确定。图8-15所示的是一个6行7列的稀疏矩阵,其三元组表为: $((1,2,12),(1,3,9),(3,1,-3),(3,6,14),(4,3,24),(5,2,18),(6,1,15),(6,4,-7))$ 。

$$A_{n \times n} = \begin{bmatrix} a_{1,1} & a_{1,2} & & & & & \\ a_{2,1} & a_{2,2} & a_{2,3} & & & & \\ & a_{3,2} & a_{3,3} & a_{3,4} & & & \\ & & \cdots & \cdots & \cdots & & \\ & & & a_{i,i-1} & a_{i,i} & a_{i,i+1} & \\ & & & & \cdots & \cdots & \cdots \\ & & & & & a_{n,n-1} & a_{n,n} \end{bmatrix}$$

图8-14 三对角矩阵示意图

$$\begin{bmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{bmatrix}$$

图8-15 稀疏矩阵示意图

稀疏矩阵的三元组表的顺序存储结构称为三元组顺序表,常用的三元组表的链式存储结构是十字链表。

8.2.3 广义表

1. 广义表的定义

广义表是线性表的推广,是由零个或多个单元素或子表所组成的有限序列。

广义表与线性表的区别在于:线性表的元素都是结构上不可分的单元素,而广义表的元素既可以是单元素,也可以是有结构的表。

广义表一般记为

$$LS = (\alpha_1, \alpha_2, \dots, \alpha_n)$$

其中 $\alpha_i (1 \leq i \leq n)$ 既可以是单个元素,又可以是广义表,分别称为原子和子表。

广义表的长度是指广义表中元素的个数。

广义表的深度是指广义表展开后所含的括号的最大层数。

2. 广义表的基本操作

与线性表类似,广义表也有查找、插入和删除等操作。由于广义表的结构较复杂,其各种运算的实现也不如线性表简单,这里只讨论两个重要的运算:

(1) 取表头 head (LS): 非空广义表 LS 的第一个元素称为表头,它可以是一个单元素,也可以是一个子表。

(2) 取表尾 tail (LS): 在非空广义表中, 除表头元素之外, 由其余元素所构成的表称为表尾。非空广义表的表尾必定是一个表。

3. 广义表的特点

(1) 广义表可以是多层次的结构, 因为广义表的元素可以是子表, 而子表的元素还可以是子表;

(2) 广义表中的元素可以是已经定义的广义表的名字, 所以一个广义表可为其他广义表所共享;

(3) 广义表可以是一个递归的表, 即广义表中的元素也可以是本广义表的名字。

4. 广义表的存储结构

由于广义表中的元素本身又可以具有结构, 它是一种带有层次的非线性结构, 因此难以用顺序存储结构表示, 通常采用链式存储结构。由上面讨论可知, 若广义表不空, 则可分解为表头和表尾两部分。反之, 一对确定的表头和表尾可唯一确定一个广义表。针对原子和子表可分别设计不同的节点结构, 如图 8-16 所示。对于广义表 $LS=(a,(b,c,d))$, 其链式存储结构如图 8-17 所示。



图 8-16 广义表的链表节点结构

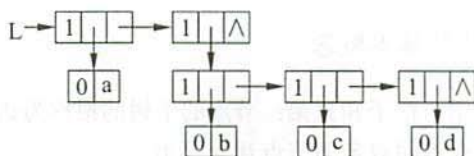


图 8-17 广义表的存储结构示意图

8.3 树

树型结构是一种非常重要的非线性结构, 该结构中一个数据元素可以有两个或两个以上的直接后继元素, 树可以用来描述客观世界中广泛存在的层次结构关系。

8.3.1 树的定义及基本运算

1. 树的定义

树是 n ($n \geq 0$) 个节点的有限集合。当 $n=0$ 时称为空树。在任一非空树 ($n>0$) 中:

- ① 有且仅有一个称为根的节点。
- ② 其余节点可分为 m ($m \geq 0$) 个互不相交的有限集 T_1, T_2, \dots, T_m , 其中每个 T_i 又都是

一棵树, 并且称为根节点的子树。

树的定义是递归的, 它表明了树本身的固有特性, 也就是一棵树由若干棵子树构成, 而子树又由更小的子树构成。

该定义只给出了树的组成特点, 若从数据结构的逻辑关系角度来看, 树中元素之间有明显的层次关系。对树中的某个节点, 它最多只和上一层的一个节点(即其双亲节点)有直接的关系, 而与其下一层的多个节点(即其子树节点)有直接关系, 如图 8-18 所示。通常, 凡是分等级的分类方案都可以用具有严格层次关系的树结构来描述。

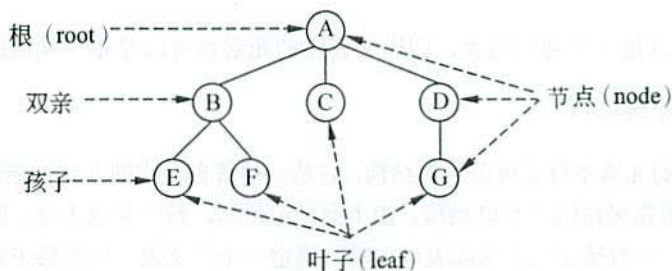


图 8-18 树结构示意图

2. 树中的基本概念

(1) 双亲、孩子和兄弟：节点的子树的根称为该节点的孩子；相应地该节点称为其子节点的双亲。具有相同双亲的节点互为兄弟。

(2) 节点的度：一个节点的子树的个数记为该节点的度。

(3) 叶子节点：也称为终端节点，指度为零的节点。

(4) 内部节点：度不为零的节点称为分支节点或非终端节点。除根节点之外，分支节点也称为内部节点。

(5) 节点的层次：根为第一层，根的孩子为第二层，依次类推，若某节点在第 i 层，则其孩子节点就在第 $i+1$ 层。

(6) 树的高度：一棵树的最大层数记为树的高度（或深度）。

(7) 有序（无序）树：若将树中节点各子树看成是从左到右具有次序的，即不能交换，则称该树为有序树，否则称为无序树。

3. 树的遍历运算

在应用树结构时，常要求按某种次序获得树中全部节点的信息，这可通过树的遍历操作来实现，树的遍历操作也是树中其他运算的重要基础。

8.3.2 二叉树的定义及基本运算

1. 二叉树的定义

二叉树是 $n(n \geq 0)$ 个节点的有限集合, 它或者是空树 ($n=0$), 或者是由一个根节点及两棵不相交的、分别称为左、右子树的二叉树所组成。可见二叉树同样具有递归性质。

特别需要注意的是, 尽管树和二叉树的概念之间有许多联系, 但它们是两个不同的概念。树和二叉树之间最主要的区别是: 二叉树的节点的子树要区分左子树和右子树, 即使在节点只有一棵子树的情况下, 也要明确指出该子树是左子树还是右子树。另外, 二叉树的节点的最大度为 2, 而树中不限制节点的度数, 如图 8-19 所示

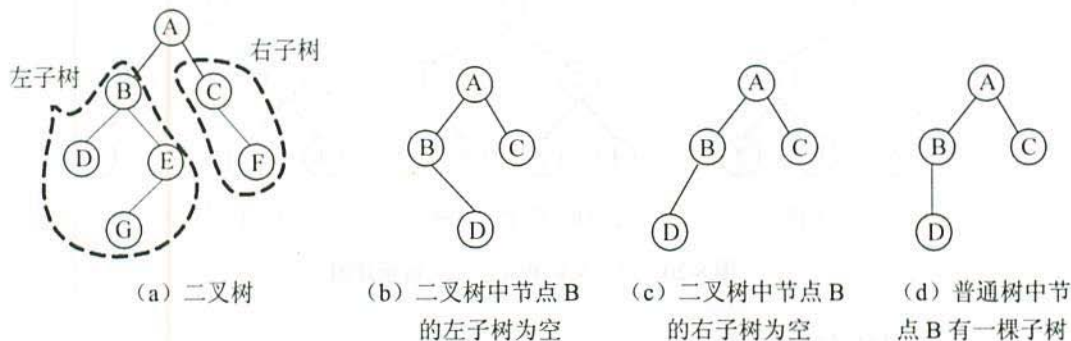


图 8-19 二叉树与普通树

2. 二叉树的运算

二叉树的基本运算是遍历, 其他运算可建立在遍历运算的基础上。

8.3.3 二叉树的性质

(1) 二叉树第 i 层 ($i \geq 1$) 上至多有 2^{i-1} 个节点。

此性质只要对层数 i 进行数学归纳证明即可。

(2) 深度为 k 的二叉树至多有 $2^k - 1$ 个节点 ($k \geq 1$)。

由性质 1, 每一层的节点数都取最大值即可: $\sum_{i=1}^k 2^{i-1} = 2^k - 1$ 。

(3) 对任何一棵二叉树, 若其终端节点数为 n_0 , 度为 2 的节点数为 n_2 , 则 $n_0 = n_2 + 1$ 。

(4) 具有 n 个节点的完全二叉树的深度为 $\lfloor \log_2 n \rfloor + 1$ 。

(5) 对一棵有 n 个节点的完全二叉树的节点按层次自左至右进行编号, 则对任一节点



$i(1 \leq i \leq n)$ 有:

若 $i=1$, 则节点 i 是二叉树的根, 无双亲; 若 $i>1$, 则其双亲为 $\left\lfloor \frac{i}{2} \right\rfloor$ 。

若 $2i>n$, 则节点 i 没有左孩子, 否则其左孩子为 $2i$ 。

若 $2i+1>n$, 则节点 i 没有右孩子, 否则其右孩子为 $2i+1$ 。

若深度为 k 的二叉树有 2^k-1 个节点, 则称其为满二叉树。可以对满二叉树中的节点进行连续编号: 约定编号从根节点起, 自上而下、自左至右依次进行。深度为 k 、有 n 个节点的二叉树, 当且仅当其每一个节点都与深度为 k 的满二叉树中编号从 1 至 n 的节点一一对应时, 称之为完全二叉树。满二叉树和完全二叉树的示意图如图 8-20 所示。

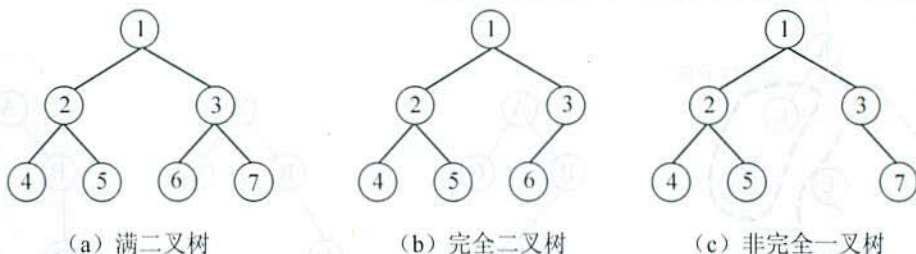


图 8-20 满二叉树和完全二叉树示意图

8.3.4 二叉树的存储结构

1. 二叉树的顺序存储结构

用一组地址连续的存储单元存储二叉树中的数据元素, 必须把节点排成一个适当的线性序列, 并且节点在这个序列中的相互位置能反映出节点之间的逻辑关系。对于深度为 k 的完全二叉树, 除第 k 层外, 其余各层中含有最大的节点数, 即每一层的节点数恰为其上一层节点数的两倍, 由此从一个节点的编号可推知其双亲、左孩子和右孩子的编号。

假设有编号为 i 的节点, 则有:

若 $i=1$ 时, 该节点为根节点, 无双亲;

若 $i>1$ 时, 该节点的双亲节点为 $\left\lfloor \frac{i}{2} \right\rfloor$;

若 $2i \leq n$, 则该节点的左孩子编号为 $2i$, 否则无左孩子;

若 $2i+1 \leq n$, 则该节点的右孩子编号为 $2i+1$, 否则无右孩子;

若 i 为奇数且不为 1, 则该节点左兄弟的编号为 $i-1$, 否则无左兄弟;

若 i 为偶数且小于 n , 则该节点右兄弟的编号为 $i+1$, 否则无右兄弟。

完全二叉树的顺序存储结构如图 8-21 (a) 所示。

显然, 完全二叉树采用顺序存储结构既简单又节省空间, 对于一般的二叉树, 则不宜采用顺序存储结构。因为一般的二叉树来也必须按照完全二叉树的形式存储, 也就是要添上一些实际并不存在的“虚节点”, 这将造成空间的浪费, 如图 8-21 (b) 所示。

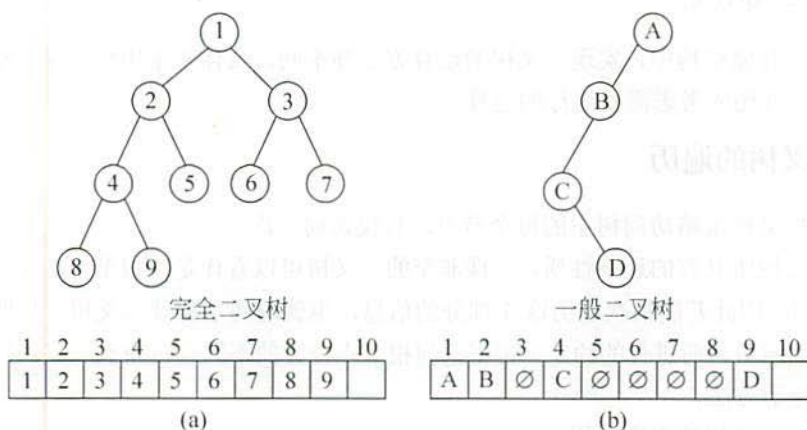


图 8-21 二叉树的顺序存储结构

在最坏情况下, 一个深度为 k 且只有 k 个节点的二叉树(单支树)却需要 $2^k - 1$ 个存储单元。

2. 二叉树的链式存储结构

由于二叉树中节点包含有数据元素、左子树的根、右子树的根及双亲等信息, 因此可以用三叉链表或二叉链表(即一个节点含有 3 个指针或含有 2 个指针)来存储二叉树, 链表的头指针指向二叉树的根节点, 如图 8-22 所示。

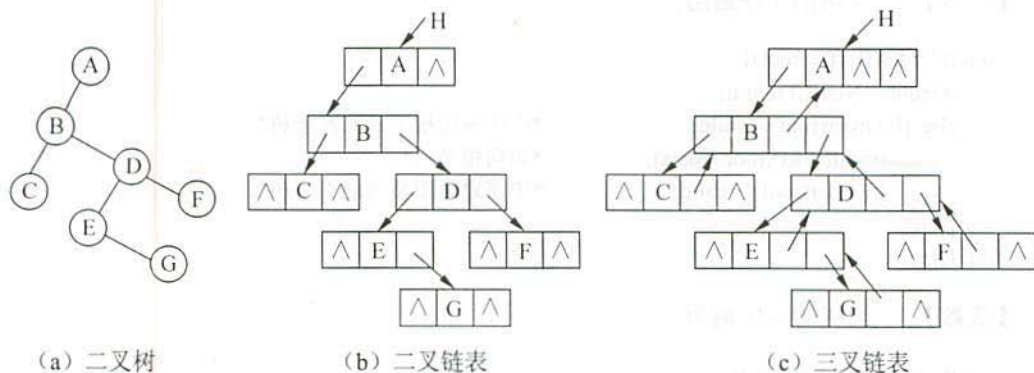


图 8-22 二叉树的链表存储结构

设节点中的数据元素为整型, 则二叉链表的节点类型定义如下:

```
typedef struct BiTNode{
    int    data;
    struct BiTNode *lchild,*rchild;
}BiTNode,*BiTree;
```

在不同的存储结构中, 实现二叉树的运算方法亦不同, 具体应采用什么存储结构, 除考虑二叉树的形态外还应考虑需要进行的运算。

8.3.5 二叉树的遍历

遍历是按某种策略访问树中的每个节点, 且仅访问一次。

由于二叉树所具有的递归性质, 一棵非空的二叉树可以看作是由根节点、左子树和右子树 3 部分构成的, 因此若能依次遍历这 3 部分的信息, 也就遍历了整棵二叉树。按照左子树的遍历在右子树的遍历之前进行的约定, 根据访问根节点位置的不同, 可得到二叉树的前序、中序和后序 3 种遍历方法。

【函数】 二叉树的先序遍历。

```
void PreOrder(BiTree root){
    if (root==NULL) return;
    else {printf("%d",root->data);          /*访问根节点*/
        PreOrder(root->lchild);             /*先序遍历根节点的左子树*/
        PreOrder(root->rchild);             /*先序遍历根节点的右子树*/
    }/*if*/
}/*PreOrder*/
```

【函数】 二叉树的中序遍历。

```
void InOrder(BiTree root){
    if (root==NULL) return;
    else {InOrder(root->lchild);             /*中序遍历根节点的左子树*/
        printf("%d",root->data);            /*访问根节点*/
        InOrder(root->rchild);              /*中序遍历根节点的右子树*/
    }/*if*/
}/*InOrder*/
```

【函数】 二叉树的后序遍历。

```
void PostOrder(BiTree root){
    if (root==NULL) return;
```

```

else {PostOrder (root->lchild);      /*后序遍历根节点的左子树*/
      PostOrder(root->rchild);        /*后序遍历根节点的右子树*/
      printf("%d",root->data);        /*访问根节点*/
    }/*if*/
  }/*PostOrder*/

```

从树根节点出发, 3 种方法的遍历路线如图 8-23 所示。该路线从根节点出发, 逆时针沿着二叉树的外缘移动, 对每个节点均途经 3 次。若第一次经过节点时进行访问, 则是先序遍历; 若第二次(或第三次)经过节点时访问节点, 则是中序遍历(或后序遍历)。因此, 只要将搜索路线上所有在第一次、第二次和第三次经过的节点信息分别输出, 即可分别得到该二叉树的先序、中序和后序遍历序列。粗略地讲, 若去掉 3 种遍历算法中的打印输出语句, 则 3 种遍历方法基本相同。这说明 3 种遍历的搜索途径相同。

遍历二叉树的基本操作就是访问节点, 不论按照哪种次序遍历, 对含有 n 个节点的二叉树, 遍历算法的时间复杂度都为 $O(n)$ 。因为在遍历的过程中, 每进行一次递归调用, 都是将函数的“活动记录”压入栈中, 因此, 栈的容量恰为树的深度。所以, 在最坏情况下, 二叉树是有 n 个节点且深度为 n 的单枝树, 遍历算法的空间复杂度也为 $O(n)$ 。

借助于一个栈结构, 可将二叉树的递归遍历算法转换为非递归算法。下面以中序遍历为例给出中序遍历的非递归算法。

【函数】 二叉树的中序非递归遍历算法。

```

typedef struct SNode{                /*链栈的节点类型*/
    BiTree elem;                     /*栈中的元素是指向二叉链表节点的指针*/
    struct SNode *next;
}SNode;
int InOrderTraverse(BiTree root)    /*二叉树的非递归遍历算法*/
{
    BiTree p;
    SNode *q,*Stacktop = NULL;      /*用不带头节点的单链表作为栈的存储结构*/
    p = root;                        /*p 指向树根节点 */
    while (p != NULL || Stacktop != NULL) {
        if (p != NULL) /*不是空树*/
        {
            q = (SNode *)malloc(sizeof(SNode));

```

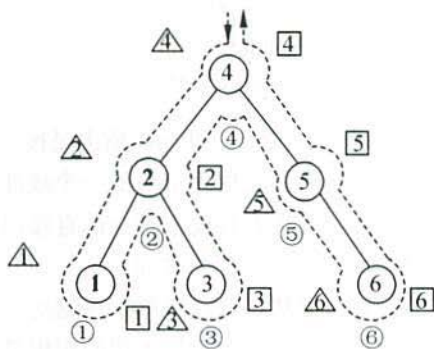


图 8-23 3 种遍历过程执行示意图



```
if (q == NULL) return -1;
q->next = Stacktop; q->elem = p;
Stacktop = q; /*根节点指针入栈*/
p = p->lchild; /*进入根的左子树*/
}
else
{ q = Stacktop; Stacktop = Stacktop->next; /*栈顶元素出栈*/
  printf("%d", q->elem->data); /*访问根节点*/
  p = q->elem->rchild; /*进入根的右子树*/
  free(q); /*释放原栈顶元素的节点空间*/
} /*if*/
} /*while*/
} /*InOrderTraverse*/
```

遍历二叉树的过程实质上是按一定规则,将树中的节点排成一个线性序列的过程,因此遍历操作得到的是树中节点的一个线性序列。在每一种序列中,有且仅有一个起始点和一个终止点,其余节点有且仅有唯一的直接前驱和直接后继。显然,关于节点的前驱和后继的讨论是针对某一个遍历序列而言的。

对二叉树还可以进行层序遍历。设二叉树的根节点所在层数为 1,层序遍历就是从树的根节点出发,首先访问第 1 层的树根节点,然后从左到右依次访问第二层上的节点,其次是第三层上的节点,依次类推,自上而下、自左至右逐层访问树中各层上节点的过程就是层序遍历。

【算法】 二叉树的层序遍历算法。

```
void LevelOrder(BiTree root) /*二叉树的层序遍历算法*/
{ Bitree p;
  InitQueue(Q); /*创建一个空队列*/
  EnQueue(Q, root) /*将根指针加入队列*/
  while (!Empty(Q)) { /*若队列不空*/
    DeQueue(Q, p); /*队头元素出队,并使 p 取队头元素的值*/
    printf("%d", p->data); /*访问节点*/
    if (p->lchild) EnQueue(p->lchild);
    if (p->rchild) EnQueue(p->rchild);
  } /*while*/
} /*LevelOrder*/
```

8.3.6 线索二叉树

1. 线索二叉树的定义

二叉树的遍历实质上是对一个非线性结构进行线性化的过程,它使得每个节点(除第一个

和最后一个外)在这些线性序列中有且仅有一个直接前驱和直接后继。但在二叉链表存储结构中,只能找到一个节点的左、右孩子信息,而不能直接得到节点在任一遍历序列中的前驱和后继信息,这些信息只有在遍历的动态过程中才能得到。因此,引入线索二叉树来保存这些动态过程得到的信息。

2. 建立线索二叉树

为了保存节点在任一序列中的前驱和后继信息,可以考虑在每个节点中增加两个指针域来存放遍历时得到的前驱和后继信息。这样就可以为以后的访问带来方便,但增加指针信息会降低存储空间的利用率,因此可考虑采用其他的方法。

若 n 个节点的二叉树采用二叉链表做存储结构,则链表中必然有 $n+1$ 个空指针域,可以利用这些空指针域来存放节点的前驱和后继信息。线索链表的节点结构如图 8-24 所示。

ltag	lchild	data	rchild	rtag
------	--------	------	--------	------

图 8-24 线索链表的节点结构

其中:

$$\begin{aligned} \text{ltag} &= \begin{cases} 0 & \text{rchild 域指示节点的左孩子} \\ 1 & \text{lchild 域指示节点的直接前驱} \end{cases} \\ \text{rtag} &= \begin{cases} 0 & \text{rchild 域指示节点的右孩子} \\ 1 & \text{lchild 域指示节点的直接后继} \end{cases} \end{aligned}$$

若二叉树的二叉链表采用图 8-24 所示的节点结构,则相应的链表称为线索链表,其中指向节点前驱、后继的指针称为线索。加上线索的二叉树称为线索二叉树。对二叉树以某种次序遍历使其成为线索二叉树的过程称为线索化。中序线索二叉树及其存储结构如图 8-25 所示。

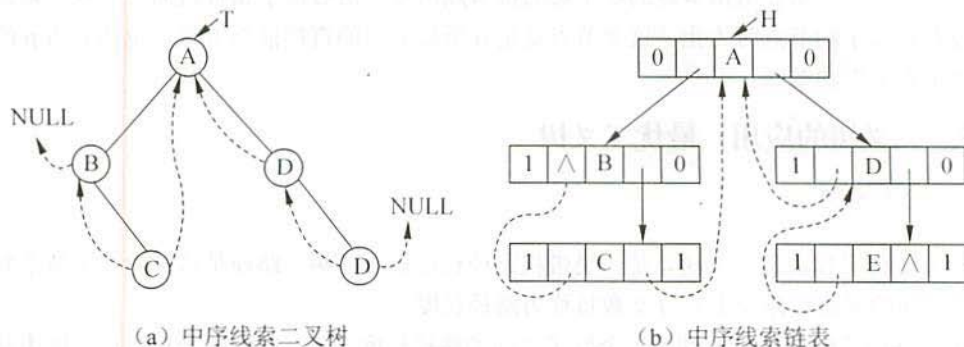
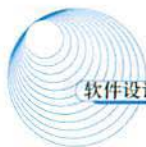


图 8-25 线索二叉树及其存储结构示意图



那么如何进行线索化呢?因为按某种次序将二叉树线索化,实质上是在遍历过程中用线索取代空指针。因此,若设指针 p 指向正在访问的节点,则遍历时设立一个指针 pre ,使其始终指向刚刚访问过的节点,这样就记下了遍历过程中节点被访问的先后关系。

在遍历的过程中,设指针 p 指向正在访问的节点:

若 p 所指向的节点有空指针域,则将相应的标志域置为 1:

若 $pre \neq \text{NULL}$ 且 pre 所指的节点的 $rtag$ 等于 1,则 $pre \rightarrow rchild = p$;

若 p 所指向节点的 $ltag$ 等于 1,则令 $p \rightarrow lchild = pre$;

使 pre 指向刚刚访问过的节点,即令 pre 等于 p 。

需要说明的是,用这种方法得到的线索二叉树,其线索是不完整的,也就是说,部分节点的前驱或后继信息还不能从其存储结构中直接得到。

3. 访问线索二叉树

如何在线索二叉树中查找节点的前驱和后继呢?

以中序线索二叉树为例,令 p 指向树中的某个节点,查找 p 所指节点的后继节点的方法是:

若 $p \rightarrow rtag = 1$,则 $p \rightarrow rchild$ 即指向其后继节点;

若 $p \rightarrow rtag = 0$,则 p 所指节点的中序后继必然是其右子树中进行中序遍历得到的第一个节点。也就是说,从 p 所指节点的右子树的根节点出发,沿左孩子指针链向下查找,直到找到一个没有左孩子的节点时为止,这个节点就是 p 所指节点的直接后继节点,也称其为 p 的右子树中“最左下”的节点。

令 p 指向中序线索树中的某个节点,则查找 p 所指节点的直接前驱的方法是:

若 $p \rightarrow ltag = 1$,则 $p \rightarrow lchild$ 即指向其前驱节点;

若 $p \rightarrow ltag = 0$,则 p 所指节点的中序前驱必然是其左子树中进行中序遍历得到的最后一个节点。也就是说,从 p 所指节点的左子树的根节点出发,沿右孩子指针链向下查找,直到找到一个没有右孩子的节点时为止,这个节点就是 p 所指节点的直接前驱节点,也称其为 p 的左子树中“最右下”的节点。

8.3.7 二叉树的应用:最优二叉树

1. 哈夫曼树

哈夫曼树又称最优二叉树,是一类带权路径长度最短的树。路径是从树中一个节点到另一个节点之间的通路,路径上的分支数目称为路径长度。

树的路径长度是从树根到每一个叶子之间的路径长度之和。节点的带权路径长度为从该节点到树根之间的路径长度与该节点权的乘积。

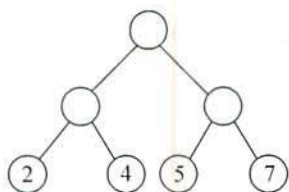
树的带权路径长度为树中所有叶子节点的带权路径长度之和, 记为

$$WPL = \sum_{k=1}^n w_k l_k$$

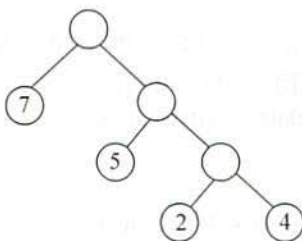
其中 n 为带权叶子节点数目, w_k 为叶子节点的权值, l_k 为叶子节点到根的路径长度。

哈夫曼树是指权值为 w_1, w_2, \dots, w_n 的 n 个叶子节点的二叉树中带权路径长度最小的二叉树。

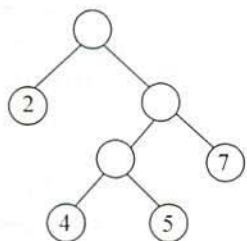
例如, 图 8-26 中所示的具有 4 个叶子节点的二叉树, 其中以 8-27 (b) 所示的二叉树的带权路径长度最小。



(a) $WPL = 2 \times (2 + 4 + 5 + 7) = 36$



(b) $WPL = 3 \times 6 + 10 + 7 = 35$



(c) $WPL = 9 \times 3 + 14 + 2 = 43$

图 8-26 不同带权路径长度的二叉树

那么如何构造最优二叉树呢? 构造最优二叉树的哈夫曼算法如下:

① 根据给定的 n 个权值 $\{w_1, w_2, \dots, w_n\}$, 构成 n 棵二叉树的集合 $F = \{T_1, T_2, \dots, T_n\}$, 其中每棵树 T_i 中只有一个带权为 w_i 的根节点, 其左右子树均空。

② 在 F 中选取两棵根节点的权值最小的树作为左、右子树构造一棵新的二叉树, 置新构造二叉树的根节点的权值为其左、右子树根节点的权值之和。

③ 从 F 中删除这两棵树, 同时将新得到的二叉树加入到 F 中。

重复 ②、③, 直到 F 中只含一棵树时为止。这棵树便是最优二叉树 (哈夫曼树)。

由此算法可知, 以选中的两棵子树构成新的二叉树, 谁作为左子树, 谁作为右子树, 并没有明确。所以具有 n 个叶子节点的权值为 w_1, w_2, \dots, w_n 的最优二叉树 (哈夫曼树) 不唯一, 但其 WPL 值是唯一确定的。

当给定了 n 个权值后, 构造出的最优二叉树中的节点数目 m 就确定了, 即 $m = 2 \times n - 1$, 所以可用一维结构数组来存储最优二叉树。

```
#define MAXLEAFNUM 50 /*最优二叉树中最大叶子数目*/
struct node{
    char ch; /*当前节点表示的字符, 对于非叶子节点, 此域不用*/
    int weight; /*当前节点的权值*/
}
```




```
int parent;           /*当前节点的父节点的下标, 为0时表示无父节点*/
int lchild,rchild;    /*当前节点的左、右孩子节点的下标, 为0时表示无孩子节点*/
} HuffmanTree[2*MAXLEAFNUM];
typedef char* HuffmanCode[MAXLEAFNUM+1];
```

【函数】 创建最优二叉树。

```
void createHTree(HuffmanTree HT, char *c,int *w, int n)
/*数组 c[0..n-1]和 w[0..n-1]存放了 n 个字符及其概率, 构造哈夫曼树 HT*/
{   int i,s1,s2;
    if (n<=1) return;
    for(i=1; i<=n; i++) /*根据 n 个权值构造 n 棵只有根节点的二叉树*/
    {   HT[i].ch = c[i-1]; HT[i].weight = w[i-1];
        HT[i].parent=HT[i].lchild=HT[i].rchild=0;
    }
    for(;i<2*n;i++)
    { HT[i].parent=0; HT[i].lchild=0; HT[i].rchild=0; }
    for(i=n+1; i<2*n; i++) /*构造哈夫曼树*/
    /*从 HT[1..i-1]中选择 parent 为 0 且 weight 最小的两棵树, 其序号为 s1 和 s2*/
    {   select(HT,i-1,s1,s2);
        HT[s1].parent = i;      HT[s2].parent = i;
        HT[i].lchild = s1;      HT[i].rchild = s2;
        HT[i].weight = HT[s1].weight + HT[s2].weight;
    } /*for*/
} /*creatHTree*/
```

2. 哈夫曼编码

若对每个字符编制相同长度的二进制码, 则称为等长编码。例如英文字符集中的 26 个字符可采用 5 位二进制串表示, 按等长编码格式构造一个字符——编码表。发送方按照编码表对信息原文进行编码后送出电文, 接收方对接收到的二进制代码按每 5 位一组进行分割, 通过查字符的编码表即可得到字符, 实现译码。

等长编码方案的实现方法比较简单, 但对通信中的原文进行编码后, 所得电文的码串过长, 不利于提高通信效率, 因此希望缩短码串的总长度。如果对每个字符设计长度不等的编码, 且让电文中出现次数较多的字符采用尽可能短的编码, 那么传送的电文总长度则可减少。

要设计长度不等的编码, 必须满足下面的条件: 任一字符的编码都不是另一个字符的编码的前缀, 这种编码也称为前缀码。对给定的字符集 $D=\{d_1, d_2, \dots, d_n\}$ 及字符的使用频率 $W=\{w_1, w_2, \dots, w_n\}$, 构造其最优前缀码的方法为: 以 d_1, d_2, \dots, d_n 作为叶子节点, w_1, w_2, \dots, w_n

作为叶子节点的权值,构造出一棵最优二叉树,然后将树中每个节点的左分支标上“0”(或“1”),右分支标上“1”(或“0”),则每个叶子节点代表的字符的编码就是从根到叶子的路径上的“0”、“1”组成的串。最优前缀编码方法如图 8-27 所示。

【函数】 从每个叶子节点出发追溯到树根,逆向找出最优二叉树中叶子节点的编码。

```
void HuffmanCoding(HuffmanTree HT,HuffmanCode HC,int n)
/*n 个叶子节点在哈夫曼树 HT 中的下标为 1~n, 第 i(1≤i≤n)个叶子
的编码存放 HC[i]中*/
```

```
{   char *cd; int i,start,c,f;
    if (n <= 1) return;
    cd = (char *)malloc(n);    cd[n-1] = '\0';
    for(i = 1; i <= n; i++) {
        start = n-1;
        for(c = i, f = HT[i].parent; f != 0; c = f, f = HT[f].parent)
            if (HT[f].lchild == c) cd[--start] = '0';
            else cd[--start] = '1';
        HC[i] = (char *)malloc(n-start);
        strcpy(HC[i],&cd[start]);
    }/*for*/
    delete cd;
}/*HuffmanCoding*/
```

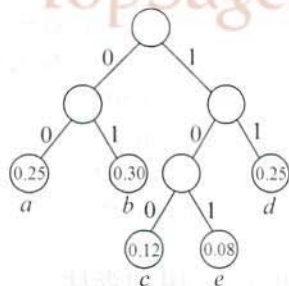


图 8-27 前缀编码示例

利用哈夫曼树译码的过程为：从根节点出发，按二进制位串中的 0 和 1 确定是进入左分支还是右分支，当到达叶子节点时译出与该叶子对应的字符。若电文未结束，则回到根节点继续进行上述过程。

【函数】 用最优二叉树进行译码。

```
void Decoding(HuffmanTree HT,int n,char *buff)
/*利用具有 n 个叶子节点的最优二叉树(存储在数组 HT 中)进行译码, 叶子的下标为 1~n*/
/*buff 指向原文的编码序列*/
{   int p=2*n-1;
    while (*buff) {
        if ((*buff)=='0') p=HT[p].lchild;/*进入左分支*/
        else p = HT[p].rchild;    /*进入右分支*/
        if (HT[p].lchild==0 && HT[p].rchild==0){ /*到达一个叶子节点*/
```




```
printf("%c",HT[p].ch);  
p=2*n-1; /*回到树根*/  
} /*if*/  
buff++;  
} /*while*/  
} /*Decoding*/
```

8.3.8 树和森林

1. 树的存储结构

(1) 树的双亲表示法: 用一组地址连续的单元存储树的节点, 并在每个节点中附设一个指示器, 指示其双亲节点在该存储结构中的位置(节点所在数组元素的下标)。显然这种表示对于求指定节点的双亲或祖先都十分方便, 但对于求指定节点的孩子及后代则需要遍历整个数组, 如图 8-28 所示。

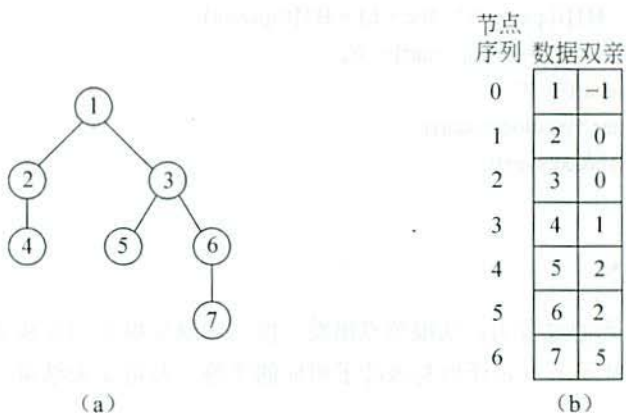


图 8-28 树的双亲表示法示意图

(2) 树的孩子表示法: 在存储结构中用指针指示出节点的每个孩子, 由于树中每个节点的子树数目不尽相同, 因此在采用链式存储结构时可以考虑多重链表。因为每个节点的指针数目不好确定, 对于定长的节点可依据树的度来设置节点中的指针, 显然会造成极大的浪费; 若设置节点中的指针数目可以不相等, 则运算时又不方便, 为此可以考虑为树中每个节点的孩子建立一个链表, 即把每个节点的孩子节点看成一个线性表, 则 n 个节点的树具有 n 个单链表。将这 n 个单链表的头指针又排成一个线性表, 如图 8-29 (a) 所示。

也可以将双亲表示法和孩子表示法结合起来, 形成树的双亲孩子表示结构, 如图 8-29 (b) 所示。

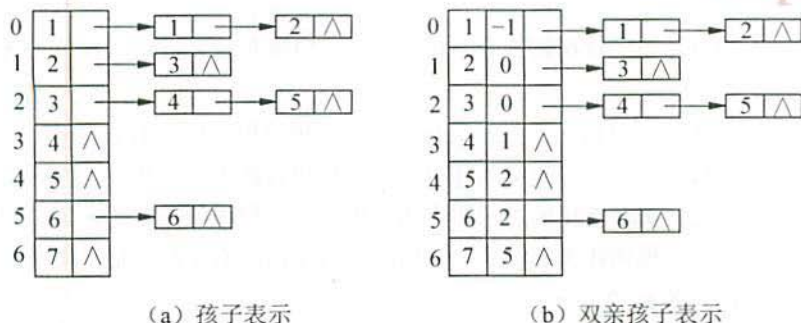


图 8-29 图 8-28 (a) 树的孩子表示法

(3) 孩子兄弟表示法：又称二叉链表表示法。在链表的节点中设置两个指针域分别指向该节点的第一个孩子和下一个兄弟，如图 8-30 所示。

树的孩子兄弟表示法为实现树、森林与二叉树之间的转换提供了基础，充分利用二叉树的有关算法来实现树及森林的操作，对难于把握规律的树和森林有着重要的现实意义。

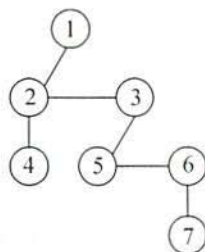


图 8-30 图 8-28 (a) 树的孩子兄弟表示法

2. 树和森林的遍历

(1) 树的遍历。由于树中每个节点可以有两棵以上的子树，因此遍历树的方法有两种：先根遍历和后根遍历。

① 树的先根遍历是先访问树的根节点，然后依次先根遍历根的各棵子树。对树的先序遍历等同于对转换所得的二叉树进行先序遍历。

② 树的后根遍历是先依次后根遍历树根的各棵子树，然后访问树根节点。树的后根遍历等同于对转换所得的二叉树进行中序遍历。

(2) 森林的遍历。按照森林和树的相互递归的定义，可以得出森林的两种遍历的方法。

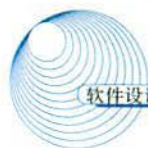
① 先序遍历森林：若森林非空，访问森林中第一棵树的根节点，先序遍历第一棵子树根节点的子树森林，再先序遍历除第一棵树之外剩余的树所构成的森林。

② 中序遍历森林：若森林非空，中序遍历森林中第一棵树的子树森林，访问第一棵树的根节点，中序遍历除第一棵树之外剩余的树所构成的森林。

3. 树、森林和二叉树之间的相互转换

树、森林和二叉树之间有一种自然的对应关系，它们之间可以互相进行转换，即任何一个森林或一棵树可以对应一棵二叉树，而任一棵二叉树也能对应到一个森林或一棵树上。

(1) 树、森林转换为二叉树。利用树的孩子兄弟表示法可导出树与二叉树的对应关系，在



树的孩子兄弟表示法中,从物理结构上看与二叉树的二叉链表表示法相同,因此就可以用这种同一存储结构的不同解释将一棵树转换为一棵二叉树,如图 8-31 所示。一棵树可转换成唯一的一棵二叉树。

由于树根没有兄弟,所以树转换为二叉树后,二叉树的根一定没有右子树。这样,将一个森林转换为一棵二叉树的方法是:先将森林中的每一棵树转换为二叉树,再将第一棵树的根作为转换后的二叉树的根,第一棵树的左子树作为转换后二叉树根的左子树,第二棵树作为转换后二叉树的右子树,第三棵树作为转换后二叉树根的右子树的右子树,依次类推,森林就可以转换为一棵二叉树,如图 8-32 所示。

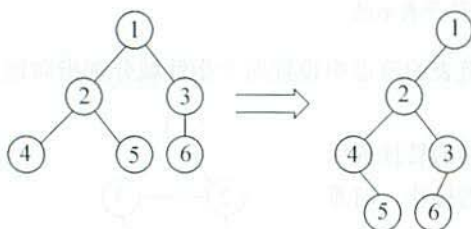


图 8-31 树转换为二叉树

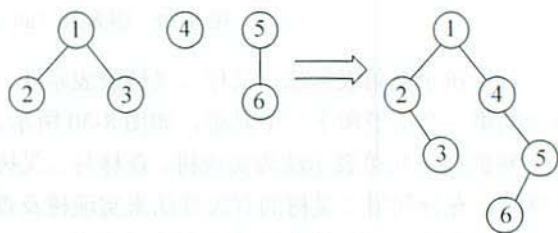


图 8-32 森林转换为二叉树

(2) 二叉树转换为树和森林。二叉树可转换为唯一的树或森林,如图 8-33 所示。

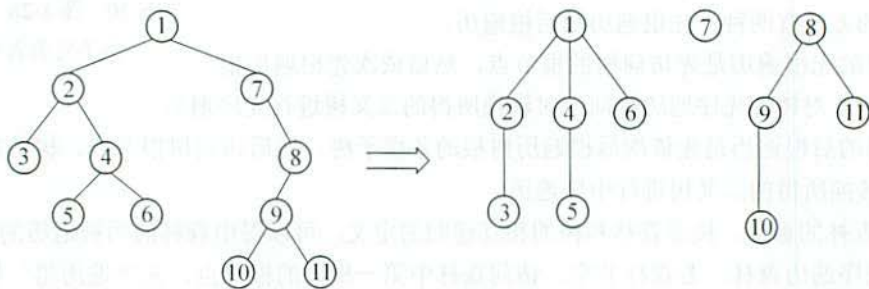


图 8-33 二叉树转换为树和森林

8.4 图

图是比树结构更复杂的一种数据结构。在线性结构中,除首节点没有前驱、末节点没有后继之外,一个节点只有唯一的一个直接前驱和唯一的一个直接后继。在树结构中,可认为除根节点没有前驱节点,其余的每个节点只有唯一的一个前驱(双亲)节点和多个后继(子树)节点。而在图结构中,任意两个节点之间都可能直接有直接的关系,所以图中一个节点的前驱节点和

后继节点的数目是没有限制的。

8.4.1 图的定义

图 G 是由两个集合 V 和 E 构成的二元组, 记作 $G = (V, E)$, 其中 V 是图中顶点的非空有限集合, E 是图中边的有限集合。从数据结构的逻辑关系角度来看, 图中任一顶点都有可能与其他顶点有关系, 而图中所有顶点都有可能与某一点有关系。在图中, 数据结构中的数据元素用顶点表示, 数据元素之间的关系用边表示。

(1) 有向图: 若图中每条边都是有方向的, 那么顶点之间的关系用 $\langle v_i, v_j \rangle$ 表示, 它说明从 v_i 到 v_j 有一条有向边 (也称为弧)。 v_i 是有向边的起点, 称为弧尾; v_j 是有向边的终点, 称为弧头。所有边都有方向的图称为有向图。

(2) 无向图: 若图中的每条边都是无方向的, 顶点 v_i 和 v_j 之间的边用 (v_i, v_j) 表示。因此, 在有向图中 $\langle v_i, v_j \rangle$ 与 $\langle v_j, v_i \rangle$ 分别表示两条边, 而在无向图中 (v_i, v_j) 与 (v_j, v_i) 表示的是同一条边。

(3) 无向完全图与有向完全图: 若一个无向图具有 n 个顶点, 而每一个顶点与其他 $n-1$ 个顶点之间都有边, 则称之为无向完全图。显然, 含有 n 个顶点的无向完全图共有 $\frac{n(n-1)}{2}$ 条边。类似地, 有 n 个顶点的有向完全图中弧的数目为 $n(n-1)$, 即任意两个不同顶点之间都有方向相反的两条弧存在。

(4) 度、出度和入度: 顶点 v 的度是指关联于该顶点的边的数目, 记作 $D(v)$ 。若 G 为有向图, 顶点的度表示该顶点的入度和出度之和。顶点的入度是以该顶点为终点的有向边的数目, 而顶点的出度指以该顶点为起点的有向边的数目, 分别记为 $ID(v)$ 和 $OD(v)$ 。无论是有向图还是无向图, 顶点数 n 、边数 e 与各顶点的度之间有以下关系:

$$e = \frac{1}{2} \sum_{i=1}^n D(v_i)$$

(5) 路径: 在无向图 G 中, 从顶点 v_p 到顶点 v_q 的路径是指存在一个顶点序列 $v_p, v_{i1}, v_{i2}, \dots, v_{in}, v_q$, 使得 $(v_p, v_{i1}), (v_{i1}, v_{i2}), \dots, (v_{in}, v_q)$ 均属于 $E(G)$ 。若 G 是有向图, 其路径也是有方向的, 它由 $E(G)$ 中的有向边 $\langle v_p, v_{i1} \rangle, \langle v_{i1}, v_{i2} \rangle, \dots, \langle v_{in}, v_q \rangle$ 组成。路径长度是路径上边或弧的数目。第一个顶点和最后一个顶点相同的路径称为回路或环。若一条路径上除了 v_p 和 v_q 可以相同外, 其余顶点均不相同, 则称其为简单路径。

(6) 子图: 若有两个图 $G=(V, E)$ 和 $G'=(V', E')$, 如果 $V' \subseteq V$ 且 $E' \subseteq E$, 则称 G' 为 G 的子图。

(7) 连通图与连通分量: 在无向图 G 中, 若从顶点 v_i 到顶点 v_j 有路径, 则称顶点 v_i 和顶点 v_j 是连通的。如果无向图 G 中任意两个顶点都是连通的, 则称其为连通图。无向图 G 的极

大连通子图称为 G 的连通分量。

(8) 强连通图与强连通分量：在有向图 G 中，如果对于每一对顶点 $v_i, v_j \in V$ 且 $v_i \neq v_j$ ，从顶点 v_i 到顶点 v_j 和从顶点 v_j 到顶点 v_i 都存在路径，则称图 G 为强连通图。有向图中的极大连通子图称为有向图的强连通分量。

(9) 网：边（或弧）带权值的图称为网。

(10) 有向树：如果一个有向图恰有一个顶点的入度为 0，其余顶点的入度均为 1，则是一棵有向树。

从图的逻辑结构的定义来看，图中的顶点之间不存在全序关系（即无法将图中的顶点排列成一个线性序列），任何一个顶点都可被看成第一个顶点；另一方面，任一顶点的邻接点之间也不存在次序关系。为了便于运算，我们给图中每个顶点赋予一个序号值。

8.4.2 图的存储结构

1. 邻接矩阵表示法

图的邻接矩阵表示是利用一个矩阵来表示图中顶点之间的关系。对于具有 n 个顶点的图 $G=(V, E)$ 来说，其邻接矩阵是一个 n 阶方阵，且满足

$$A[i][j] = \begin{cases} 1 & \text{若 } (v_i, v_j) \text{ 或 } \langle v_i, v_j \rangle \text{ 是 } E \text{ 中的边} \\ 0 & \text{若 } (v_i, v_j) \text{ 或 } \langle v_i, v_j \rangle \text{ 不是 } E \text{ 中的边} \end{cases}$$

由邻接矩阵的定义可知，无向图的邻接矩阵是对称的，有向图的邻接矩阵就不一定对称了。借助于邻接矩阵容易判定任意两个顶点之间是否有边（或弧）相连，并且容易求得各个顶点的度。对于无向图，顶点 v_i 的度是邻接矩阵中第 i 行（或列）的值不为 0 的元素个数；对于有向图，第 i 行（或列）值不为 0 的元素个数是顶点 v_i 的出度 $OD(v_i)$ ，第 j 列的非 0 元素个数是顶点 v_j 的入度 $ID(v_j)$ 。图 8-34 所示的有向图和无向图的邻接矩阵分别为 A 和 B 。

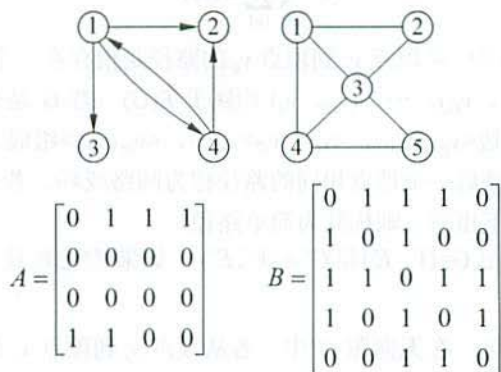


图 8-34 有向图和无向图

网（赋权图）的邻接矩阵可定义为：

$$A[i][j] = \begin{cases} W_{ij} & \text{若 } (v_i, v_j) \text{ 或 } \langle v_i, v_j \rangle \text{ 是 } E \text{ 中的边} \\ \infty & \text{若 } (v_i, v_j) \text{ 或 } \langle v_i, v_j \rangle \text{ 不是 } E \text{ 中的边} \end{cases}$$

其中， W_{ij} 是边上的权值。

图 8-35 所示的是网及其邻接矩阵 C 。

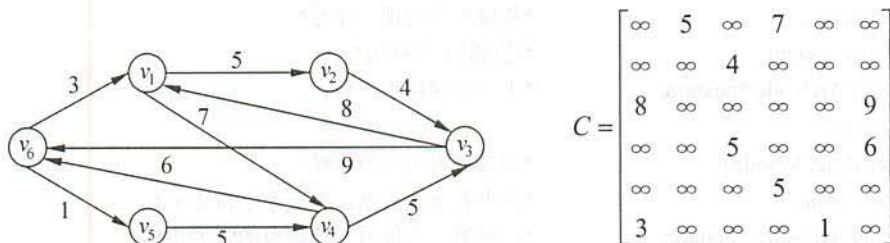


图 8-35 一个网及其邻接矩阵表示

若用邻接矩阵表示图，则对应的数据类型可定义为：

```
#define MaxN 30 /*图中顶点数目的最大值*/
typedef int AdjMatrix[MaxN][MaxN];
或 typedef double AdjMatrix[MaxN][MaxN]; /*赋权图*/
typedef struct {
    int Vnum; /*图中的顶点数目*/
    AdjMatrix Arcs;
} Graph;
```

2. 邻接链表表示法

邻接链表指的是：为图的每个顶点建立一个单链表，第 i 个单链表中的节点表示依附于顶点 v_i 的边（对于有向图是以 v_i 为尾的弧）。邻接链表中的节点有表节点和表头节点两种类型，如下所示。



其含义为：

- (1) adjvex: 指示与顶点 v_i 邻接的顶点的序号；
- (2) nextarc: 指示下一条边或弧的节点；
- (3) info: 存储和边或弧有关的信息，如权值等；
- (4) data: 存储顶点 v_i 的名或其他有关信息；



(5) firstarc: 指示链表中的第一个节点。

这些表头节点(可以链相连)通常以顺序结构的形式存储,以便随机访问任一顶点的链表。若图用邻接链表来表示,则对应的数据类型可定义如下:

```
#define MaxN 30 /*图中顶点数目的最大值*/
typedef struct ArcNode{ /*邻接链表的表节点*/
    int adjvex; /*邻接顶点的顶点序号*/
    double weight; /*边(弧)上的权值*/
    struct ArcNode *nextarc; /*下一个邻接顶点*/
}EdgeNode;
typedef struct VNode{ /*邻接链表的头节点*/
    char data; /*顶点表示的数据,以一个字符表示*/
    struct ArcNode *firstarc; /*指向第一条依附于该顶点的弧的指针*/
}AdjList[MaxN];
typedef struct {
    int Vnum; /*图中顶点的数目*/
    AdjList Vertices;
}Graph;
```

显然,对于有 n 个顶点、 e 条边的无向图来说,其邻接链表需用 n 个头节点和 $2e$ 个表节点。

对于无向图的邻接链表,顶点 v_i 的度恰为第 i 个邻接链表中表节点的数目,而在有向图中,为求顶点的入度,必须扫描逐个邻接表,这是因为第 i 个邻接链表中表节点的数目只是顶点 v_i 的出度。为此,可以建立一个有向图的逆邻接链表。有向图的邻接表和逆邻接表如图8-36所示。

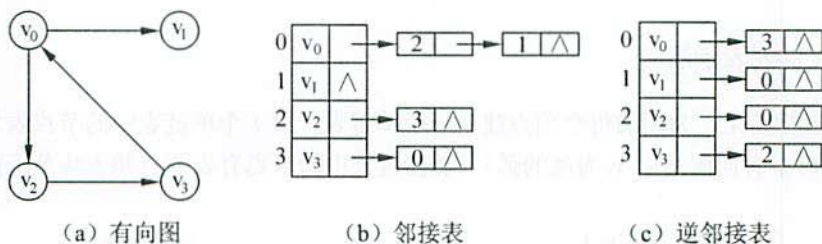


图 8-36 一个有向图及其邻接表和逆邻接表表示

8.4.3 图的遍历

与树的遍历相似,图的遍历也是从某个顶点出发,沿着某条搜索路径对图中所有顶点进行访问且只访问一次。图的遍历算法是求解图的连通性问题、拓扑排序及求关键路径等算法的基础。

图的遍历要比树的遍历复杂得多。因为图的任一个节点都可能与其余顶点相邻接,所以在访问了某个顶点之后,可能沿着某路径又回到该节点上,为了避免顶点的重复访问,在图的遍历过程中,必须记下每个已访问过的顶点。

1. 深度优先搜索 (DFS)

此种方法类似于树的先根遍历,在第一次经过一个顶点时就进行访问操作。从图 G 中任一节点 v 出发按深度优先搜索法进行遍历的步骤如下:

- (1) 设立搜索指针 p , 使 p 指向顶点 v ;
- (2) 访问 p 节点, 并使 p 指向与 p 顶点相邻接的且尚未被访问过的节点;
- (3) 若 p 不空, 则重复步骤 (2), 否则执行步骤 (4);
- (4) 沿着刚才访问的次序、方向回溯到一个尚有邻接顶点且未被访问过的顶点, 并使 p 指向这个未被访问的顶点, 然后重复步骤 (2), 直至所有的顶点均被访问为止。

该算法的特点是尽可能先对纵深方向搜索, 因此可以很容易得到其递归的遍历算法。

【函数】 以邻接链表表示图的深度优先搜索算法。

```
int visited[MaxN] = {0};           /*调用遍历算法前所有的顶点都没有被访问过*/
void Dfs(Graph G,int i) {
    EdgeNode *t; int j;
    printf("%d",i);                /*访问序号为 i 的顶点*/
    visited[i] = 1;                 /*序号为 i 的顶点已访问过*/
    t = G.Vertices[i].firstarc;     /*取顶点 i 的第一个邻接顶点*/
    while(t != NULL) {              /*检查所有与顶点 i 相邻接的顶点*/
        j = t->adjvex;               /*顶点 j 为顶点 i 的一个邻接顶点*/
        if (visited[j] == 0)         /*若顶点 j 未被访问过*/
            Dfs(g,j);                /*从顶点 j 出发进行深度优先搜索*/
        t = t->nextarc;              /*取顶点 i 的下一个邻接顶点*/
    } /*while*/
} /*Dfs*/
```

从函数 $Dfs()$ 之外调用 Dfs 可以访问到所有与一个指定顶点有路径相通的其他顶点。若图是不连通的, 则下一次应从另一个未被访问过的顶点出发, 再次调用 $Dfs()$ 进行遍历, 直到将图中所有的顶点都访问到为止。深度优先的搜索过程如图 8-37 所示。

深度优先遍历图的过程实质上是对某个顶点查找其邻接点的过程, 其耗费的时间取决于所采用的存储结构。当图用邻接矩阵表示时, 查找所有顶点的邻接点所需时间为 $O(n^2)$ 。若以邻接表作为图的存储结构, 则需要 $O(e)$ 的时间复杂度查找所有顶点的邻接点。因此, 当以邻接表作为存储结构时, 深度优先搜索遍历图的时间复杂度为 $O(n+e)$ 。

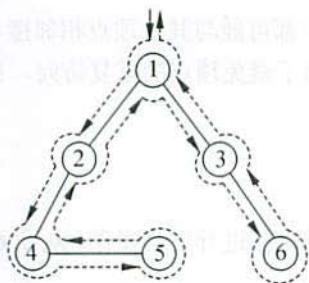
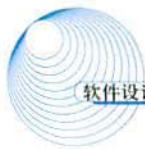


图 8-37 深度优先搜索遍历过程

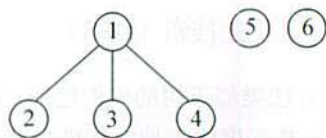


图 8-38 一个不连通的无向图

2. 广度优先搜索 (BFS)

图的广度优先搜索方法为: 假设从图中某个顶点 v 出发, 在访问了 v 之后依次访问 v 的各个未被访问过的邻接点, 然后分别从这些邻接点出发依次访问它们的邻接点, 并使“先被访问的顶点的邻接点”先于“后被访问的顶点的邻接点”被访问, 直至图中所有已被访问的顶点的邻接点都被访问到。若此时还有未被访问的顶点, 则另选图中的一个未被访问的顶点作为起点, 重复上述过程, 直至图中所有的顶点都被访问到为止。对图 8-38 所示的图进行广度优先搜索, 得到的序列为: 1, 2, 3, 4, 5, 6。

广度优先遍历图的特点是尽可能先对横向进行搜索, 即最先访问的顶点的邻接点亦先被访问。为此, 引入队列结构来保存已访问过的顶点序列, 即每当一个顶点被访问后, 就将其放入队中, 当队头顶点出队时, 就访问其未被访问的邻接点并令这些邻接点入队。

【算法】 以邻接链表表示图的广度优先搜索算法。

```
void Bfs(Graph G) {                                     /*广度优先遍历图 G*/
    EdgeNode *t; int i,j,k;
    int visited[G.Vnum] = {0};                          /*调用遍历算法前所有的顶点都没有被访问过*/
    InitQueue(Q);                                         /*创建一个空队列*/
    for(i=0;i<G.Vnum;i++) {
        if(!visited[i]) {                                /*顶点 i 未被访问过*/
            EnQueue(Q,i); printf("%d",i); visited[i]=1; /*访问顶点 i*/
            while(!Empty(Q)){                             /*若队列不空*/
                Dequeue(Q,k);
                for(t = G.Vertices[k].firstarc;t; t = t->nextarc){
                    /*检查所有与顶点 k 相邻接的顶点*/
                    j = t->adjvex;                         /*顶点 j 是顶点 k 的一个邻接点*/
                    if(visited[j] == 0) {                 /*若顶点 j 未被访问过, 将 j 加入队列*/
                        EnQueue(Q,j);
                        printf("%d",j);                   /*访问序号为 j 的顶点*/
                    }
                }
            }
        }
    }
}
```

```

        visited[j] = 1;           /*置顶点j已被访问过标志*/
    }/*if*/
}/*for*/
}/*while*/
}/*if*/
}/*for i*/
}/*Bfs*/

```

在广度优先遍历算法中, 每个顶点至多进一次队列。

遍历图的过程实质上是通过边或弧找邻接点的过程, 因此广度优先搜索遍历图和深度优先搜索遍历图的时间复杂度相同, 其不同之处仅仅在于对顶点访问的次序不同。

8.4.4 生成树及最小生成树

1. 生成树的概念

一个连通图的生成树是一个极小连通子图, 它包含图中的全部顶点, 但只有构成一棵树的 $n-1$ 条边。图 8-39 所示的是图及其生成树和非生成树。

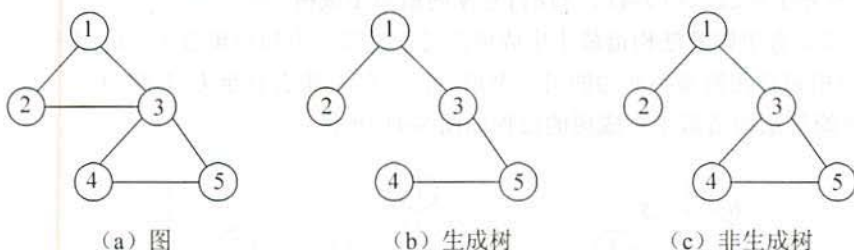


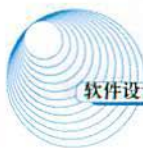
图 8-39 一个无向图的生成树和非生成树

按深度和广度优先搜索分别进行遍历将得到不同的生成树, 分别称为深度优先生成树和广度优先生成树。例如, 图 8-40 所示的是图的一棵深度优先生成树和一棵广度优先生成树。



图 8-40 图的搜索生成树

对于有 n 个顶点的连通图, 至少有 $n-1$ 条边, 而生成树中恰好有 $n-1$ 条边, 所以连通图的



生成树是该图的极小连通子图。若在图的生成树中任意加一条边,则必然形成回路。

图的生成树不是唯一的。从不同的顶点出发,选择不同的存储方式,可以得到不同的生成树。对于非连通图而言,每个连通分量中的顶点集和遍历时走过的边集一起构成若干棵生成树,把它们称为非连通图的生成树森林。

2. 最小生成树

由于生成树不唯一,从不同的顶点出发可得到不同的生成树。对于连通网来说,边是带权值的,生成树的各边也带权值,于是就把生成树各边的权值总和称为生成树的权,把权值最小的生成树称为最小生成树。求解最小生成树有许多实际的应用。常用的最小生成树算法有普里姆(Prim)算法和克鲁斯卡尔(Kruskal)算法。

1) 普里姆(Prim)算法

假设 $N=(V, E)$ 是连通网, TE 是 N 上最小生成树中边的集合。算法从顶点集合 $U=\{u_0\}(u_0 \in V)$ 、边的集合 $TE=\{\}$ 开始,重复执行下述操作:在所有 $u \in U, v \in V-U$ 的边 $(u, v) \in E$ 中找一条代价最小的边 (u_0, v_0) ,把这条边并入集合 TE ,同时将 v_0 并入集合 U ,直至 $U=V$ 时为止。此时 TE 中必有 $n-1$ 条边,则 $T=(V, \{TE\})$ 为 N 的最小生成树。

由此可知,普里姆算法构造最小生成树的过程是以一个顶点集合 $U=\{u_0\}$ 作初态,不断寻找与 U 中顶点相邻且代价最小的边的另一个顶点,扩充 U 集合直至 $U=V$ 时为止。

用普里姆算法构造最小生成树的过程如图 8-41 所示。

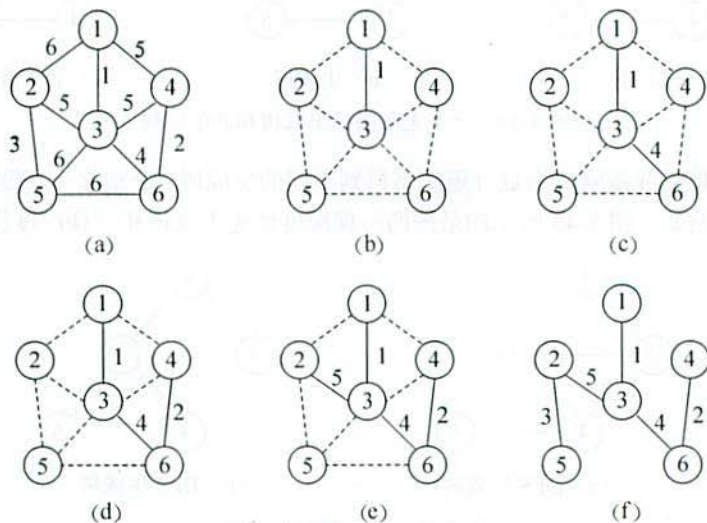


图 8-41 普里姆算法构造最小生成树的过程

普里姆算法的时间复杂度为 $O(n^2)$, 与图中的边数无关, 因此该算法适合于求边稠密的网的最小生成树。

2) 克鲁斯卡尔 (Kruskal) 算法

克鲁斯卡尔求最小生成树的算法思想为: 假设连通网 $N=(V, E)$, 令最小生成树的初始状态为只有 n 个顶点而无边的非连通图 $T=(V, \{\})$, 图中每个顶点自成一个连通分量。在 E 中选择代价最小的边, 若该边依附的顶点落在 T 中不同的连通分量上, 则将此边加入到 T 中, 否则舍去此边而选择下一条代价最小的边。依次类推, 直至 T 中所有顶点都在同一连通分量上为止。

用克鲁斯卡尔算法构造最小生成树的过程如图 8-42 所示。

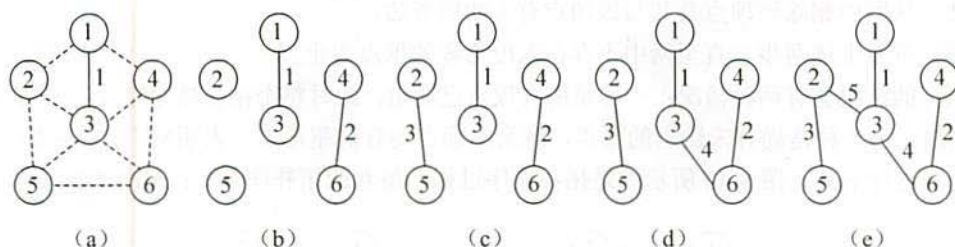


图 8-42 克鲁斯卡尔算法构造最小生成树的过程

克鲁斯卡尔算法的时间复杂度为 $O(e \log e)$, 与图中的顶点数无关, 因此该算法适合于求边稀疏的网的最小生成树。

8.4.5 拓扑排序和关键路径

1. AOV 网

在工程领域, 一个大的工程项目通常被划分为许多较小的子工程 (称为活动)。显然当这些子工程都完成时, 整个工程也就完成了。在有向图中, 若以顶点表示活动, 用有向边表示活动之间的优先关系, 则称这样的有向图为以顶点表示活动的网 (Activity On Vertex network), 简称 AOV 网。在网中, 若从顶点 v_i 到顶点 v_j 有一条有向路径, 则顶点 v_i 是 v_j 的前驱, 顶点 v_j 是 v_i 的后继。若 $\langle v_i, v_j \rangle$ 是网中的一条弧, 则顶点 v_i 是 v_j 的直接前驱, 顶点 v_j 是 v_i 的直接后继。AOV 网中的弧表示了活动之间的优先关系, 也可以说是一种活动进行时的制约关系。

在 AOV 网中不应出现有向环, 若存在的话, 则意味着某项活动必须以自身任务的完成为先决条件, 显然这是荒谬的。因此, 若要检测一个工程是否可行, 首先就应检查对应的 AOV 网是否存在回路。不存在回路的 AOV 网称为有向无环图, 或 DAG (Directed Acycline Graph) 图。检测的方法是对有向图构造其顶点的拓扑有序序列。若图中所有顶点都在它的拓扑有序序列中, 则该 AOV 网中必定不存在环。

2. 拓扑排序及其算法

拓扑排序是将 AOV 网中所有顶点排成一个线性序列的过程, 并且该序列满足: 若在 AOV 网中从顶点 v_i 到 v_j 有一条路径, 则在该线性序列中, 顶点 v_i 必然在顶点 v_j 之前。

拓扑排序即指对 AOV 网构造拓扑序列的操作。一般情况下, 假设 AOV 图代表一个工程计划, 则 AOV 网的一个拓扑排序就是一个工程顺利完成的可行方案。

对 AOV 网进行拓扑排序的方法如下:

- (1) 在 AOV 网中选择一个入度为零 (没有前驱) 的顶点且输出它;
- (2) 从网中删除该顶点及其与该顶点有关的所有边;
- (3) 重复上述两步, 直至网中不存在入度为零的顶点为止。

执行的结果会有两种情况: 一种是所有顶点已输出, 此时整个拓扑排序完成, 说明网中不存在回路; 另一种是尚有未输出的顶点, 剩余的顶点均有前驱顶点, 表明网中存在回路, 拓扑排序无法进行下去。图 8-43 所示的是拓扑排序过程, 得到的拓扑序列为: 6, 1, 4, 3, 2, 5。

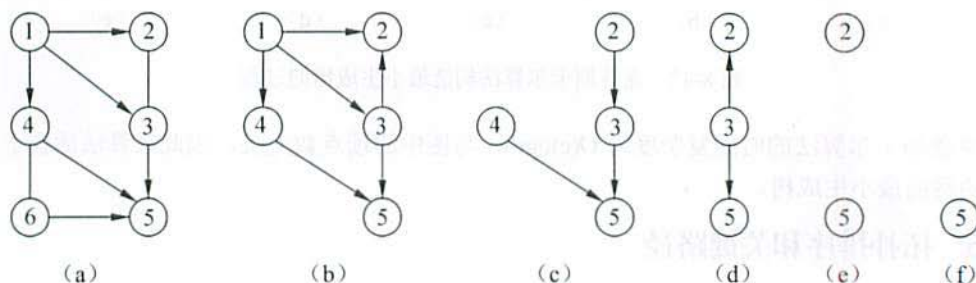


图 8-43 拓扑排序过程

当有向图中无环时, 也可以利用深度优先遍历进行逆拓扑排序。由于图中无环, 从图中某点出发进行深度优先搜索遍历时, 最先退出 DFS 函数的顶点即是出度为零的顶点, 它是拓扑有序序列中最后的一个顶点。由此, 按退出 DFS 函数的先后记录下来的顶点序列即为逆向的拓扑有序序列。拓扑排序算法的时间复杂度为 $O(n+e)$ 。

3. AOE 网

若在带权有向图 G 中以顶点表示事件, 以有向边表示活动, 边上的权值表示该活动持续的时间, 则这种带权有向图称为用边表示活动的网 (Activity On Edge network), 简称 AOE 网。通常在 AOE 网中列出了完成预定工程计划所需进行的活动、每项活动的计划完成时间、要发生哪些事件以及这些事件和活动间的关系, 从而可以分析该项工程是否实际可行并估计工程完成的最短时间, 以及影响工程进度的关键活动; 进一步可以进行人力、物力的调度和分配, 以

达到缩短工期的目的。

用 AOE 网表示一项工程计划时, 顶点所表示的事件实际上就是某些活动已经完成、某些子活动可以动工的标志。具体地说, 顶点所表示的事件是指该顶点所有进入边所表示的活动均已完成、从它的出发的边所表示的活动均可以开始的一种状态。

一般情况下, 每项工程都有一个开始状态和一个结束状态, 所以在 AOE 网中至少有一个入度为零的开始顶点, 称为源点, 另外, 应有一个出度为零的结束顶点, 称为汇点。AOE 网中不应存在有向回路, 否则整个工程无法完成。

与 AOV 网不同, AOE 网所关心的是:

- (1) 完成该工程至少需要多少时间?
- (2) 哪些活动是影响整个工程进度的关键?

由于 AOE 网中的某些活动能够并行地进行, 所以完成整个工程所需的时间是从开始顶点到结束顶点的最长路径的长度。这里的路径长度是指该路径上的权值之和。

4. 关键路径和关键活动

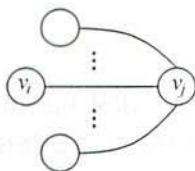
从源点到汇点的路径中, 长度最长的路径称为关键路径。关键路径上的所有活动均是关键活动。如果任何一项关键活动没有按期完成, 则会影响整个工程的进度, 而提高关键活动的速度通常可以缩短整个工程的工期。假设在 n 个顶点的 AOE 网中, 顶点 v_0 表示源点、顶点 v_{n-1} 表示汇点, 则引入顶点事件的最早、最晚发生时间, 活动的最早、最晚开始时间等术语。

(1) 顶点事件的最早发生时间 $ve(j)$ 。 $ve(j)$ 是指从源点 v_0 到 v_j 的最长路径长度(时间)。这个时间决定了所有从 v_j 发出的弧所表示的活动能够开工的最早时间。

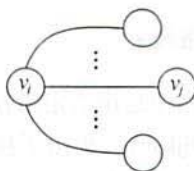
$ve(j)$ 计算方法为

$$\begin{cases} ve(0) = 0 \\ ve(j) = \max \{ve(i) + dut(<i, j>)\} & <i, j> \in T, 1 \leq j \leq n-1 \end{cases}$$

其中, T 是所有到达顶点 j 的弧的集合; $dut(<i, j>)$ 是弧 $<i, j>$ 上的权值; n 是网中的顶点数(即汇点的序号), 如图 8-44 (a) 所示。



(a) 顺推事件的最早发生时间



(b) 逆推事件的最晚发生时间

图 8-44 推导事件的最早、最晚发生时间示意图

显然,上式是一个从源点开始的递推公式。必须在 v_j 的所有前驱顶点事件的最早发生时间全部得出后才能计算 $ve(j)$ 。这样必须对 AOE 网进行拓扑排序,然后按拓扑有序序列逐个求出各顶点事件的最早发生时间。

(2) 顶点事件的最晚发生时间 $vl(i)$ 。 $vl(i)$ 是指在不推迟整个工期的前提下,事件 v_i 的最晚发生时间。对一个工程来说,计划用几天时间完成是可以从 AOE 网求得的,其数值就是汇点 v_{n-1} 的最早发生时间 $ve(n-1)$,而这个时间也就是 $vl(n-1)$ 。其他顶点事件的 vl 应从汇点开始,逐步向源点方向递推才能求得,所以 $vl(i)$ 的计算公式为

$$\begin{cases} vl(n-1) = ve(n-1) \\ vl(i) = \min\{vl(j) - dut(<i,j>)\} & <i,j> \in S, 0 \leq i \leq n-2 \end{cases}$$

其中, S 是所有从顶点 i 发出的弧的集合,如图 8-44 (b) 所示。

显然,必须在顶点 v_i 的所有后继顶点事件的最晚发生时间全部得出后才能计算 $vl(i)$ 。这样必须对 AOE 网逆拓扑排序,由逆拓扑序列递推计算各顶点的 vl 值。

(3) 活动 a_k 的最早开始时间 $e(k)$ 。 $e(k)$ 是指弧 $<i,j>$ 所表示的活动 a_k 最早可开工时间。

$$e(k) = ve(i)$$

这说明活动 a_k 的最早开始时间等于事件 v_i 的最早发生时间。这与 AOE 网中关于事件的含义的解释是完全一致的。

(4) 活动 a_k 的最晚开始时间 $l(k)$ 。 $l(k)$ 是指在不推迟整个工期的前提下,该活动得最晚开始时间。若活动 a_k 由弧 $<i,j>$ 表示,则

$$l(k) = vl(j) - dut(<i,j>)$$

对于活动 a_k 来说,若 $e(k)=l(k)$,则表示活动 a_k 是关键活动,它说明该活动最早可开工时间与整个工程计划允许该活动最晚的开工时间一致,施工期一点也不能拖延。若活动 a_k 不能按期完成,则工程将延期;若活动 a_k 提前完成,则可能使整个工程提前完工。

由关键活动组成的路径是关键路径。依照上述的计算关键活动的方法,就可形成求 AOE 网的关键路径。

8.4.6 最短路径

1. 单源点最短路径

所谓单源点最短路径是指给定带权有向图 G 和源点 v_0 ,求从 v_0 到 G 中其余各顶点的最短路径。迪杰斯特拉(Dijkstra)提出了按路径长度递增的次序产生最短路径的算法其思想是:把网中所有的顶点分成两个集合 S 和 T , S 集合的初态只包含顶点 v_0 , T 集合的初态为网中除 v_0 之外的所有顶点。凡以 v_0 为源点,已经确定了最短路径的终点并入 S 集合中;顶点集合 T 则是尚未确定最短路径的顶点的集合。按各顶点与 v_0 间最短路径长度递增的次序,逐个把 T 集合中

的顶点加入到 S 集合中去,使得从 v_0 到 S 集合中各顶点的路径长度始终不大于从 v_0 到 T 集合中各顶点的路径长度。

为了能方便地求出从 v_0 到 T 集合中各顶点最短路径的递增次序,算法引入了一个辅助向量 $dist$ 。它的某一个分量 $dist[i]$ 表示当前求出的从 v_0 到终点 v_i 的最短路径长度。这个路径长度并不一定是真正最短路径长度。它的初始状态为:若从 v_0 到 u 有弧,则 $dist[u]$ 为弧上的权值;否则,置 $dist[i]$ 为 ∞ 。显然,长度为 $dist[u] = \min\{dist[i] | v_i \in V(G)\}$ 的路径就是从 v_0 出发的长度最短的一条最短路径。此路径为 (v_0, u) , 这时顶点 u 应从集合 T 中删除,将其并入集合 S 。

设网用邻接矩阵 $arcs$ 存储,那么每次选出一个顶点 u 并使之并入集合 S 后,就修改 T 集合中各顶点的最短路径长度 $dist$ 。对于 T 集合中的某一个顶点 i 来说,其最短路径或者是 (v_0, v_i) , 或者是 (v_0, v_u, v_i) , 绝不可能有其他的选择。也就是说,若

$$dist[u] + arcs[u][i] < dist[i]$$

则修改 $dist[i]$, 使

$$dist[i] = dist[u] + arcs[u][i]$$

对 T 集合中各顶点的 $dist$ 进行修改后,再从中挑选出一个路径长度最小的顶点,从 T 集合中删除之并将其并入 S 集合。依次类推,就能求出源点到其余各顶点的最短路径长度。

对于图 8-45 所示的有向网,用迪杰斯特拉算法求解顶点 v_0 到达其余顶点的最短路径的过程如表 8-1 所示。

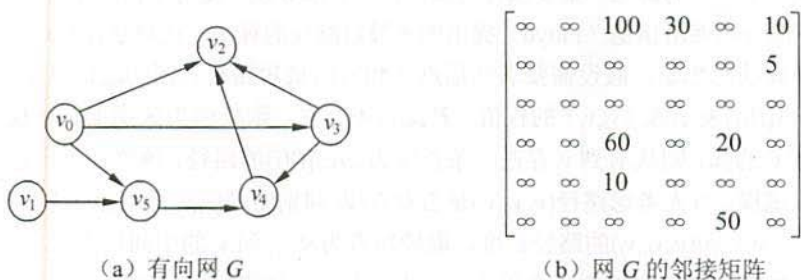
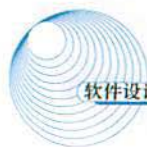


图 8-45 有向网 G 及其邻接矩阵

表 8-1 迪杰斯特拉算法求解图 8-45 中顶点 v_0 到 v_1 、 v_2 、 v_3 、 v_4 、 v_5 最短路径的过程

终 点	1	2	3	4	5
v_1	∞	∞	∞	∞	∞
v_2	100 (v_0, v_2)	100 (v_0, v_2)	90 (v_0, v_3, v_2)	90 (v_0, v_3, v_4, v_2)	无



续表

终 点	1	2	3	4	5
v_3	30 (v_0, v_3)	30 (v_0, v_3)			
v_4	∞	60 (v_0, v_5, v_4)	50 (v_0, v_3, v_4)		
v_5	10 (v_0, v_5)				
说明	从 v_0 到 v_1, v_2, v_3, v_4, v_5 的路径中, (v_0, v_5)最短, 则将顶点 v_5 加入 S 集合, 并且更新 v_0 到 v_4 的路径	从 v_0 到 v_1, v_2, v_3, v_4 的路径中, (v_0, v_3)最短, 则将顶点 v_3 加入 S 集合, 并且更新 v_0 到 v_2, v_0 到 v_4 的路径	从 v_0 到 v_1, v_2, v_4 的路径中, (v_0, v_3, v_4)最短, 则将顶点 v_4 加入 S 集合, 并且更新 v_0 到 v_2 的路径	从 v_0 到 v_1, v_2 的路径中, (v_0, v_3, v_4, v_2)最短, 则将顶点 v_4 加入 S 集合	
集合 S	$\{v_5\}$	$\{v_5, v_3\}$	$\{v_5, v_3, v_4\}$	$\{v_5, v_3, v_4, v_2\}$	

2. 每对顶点间的最短路径

若每次以一个顶点为源点, 重复执行迪杰斯特拉算法 n 次, 便可求得网中每一对顶点之间的最短路径。下面介绍弗洛伊德(Floyd)提出的求最短路径的算法, 该算法在形式上要简单一些。

弗洛伊德算法思想是: 假设需要求从顶点 v_i 到 v_j 的最短路径。图的边信息采用邻接矩阵的方式存储, $arcs[i][j]$ 表示弧 (v_i, v_j) 的权值, 若此弧不存在, 则权值为区别于有效权值的一个数。如果存在 v_i 到 v_j 的弧, 则从 v_i 到 v_j 存在一条长度为 $arcs[i][j]$ 的路径, 该路径不一定是最短路径, 尚需进行 n 次试探。首先考虑路径 (v_i, v_0, v_j) 是否存在(即判别路径 (v_i, v_0) 和 (v_0, v_j) 是否存在)。如果存在, 则比较 (v_i, v_j) 和 (v_i, v_0, v_j) 的路径长度, 取较短者为从 v_i 到 v_j 的中间顶点的序号不大于 0 的最短路径。假如在路径上再增加一个顶点 v_1 , 也就是说, 如果 (v_i, \dots, v_1) 和 (v_1, \dots, v_j) 分别是当前找到的中间顶点的序号不大于 0 的最短路径, 那么 $(v_i, \dots, v_1, \dots, v_j)$ 就有可能是从 v_i 到 v_j 的中间顶点的序号不大于 1 的最短路径。将它和已经得到的从 v_i 到 v_j 的中间顶点的序号不大于 0 的最短路径相比较, 从中选出中间顶点的序号不大于 1 的最短路径之后, 再增加一个顶点 v_2 继续进行试探。依次类推。在一般情况下, 若 (v_i, \dots, v_k) 和 (v_k, \dots, v_j) 分别是 v_i 到 v_k 和从 v_k 到 v_j 的中间顶点的序号不大于 $k-1$ 的最短路径, 则将 $(v_i, \dots, v_k, \dots, v_j)$ 与已经得到的从 v_i 到 v_j 的中间顶点的序号不大于 $k-1$ 的最短路径相比较, 其长度较短者便是从 v_i 到 v_j 的中间顶点的序号不大于 k 的最短路径。这样, 在经过 n 次比较后, 最后求得的必是从 v_i 到 v_j 的最短路径。按此方法, 可以同时求得各对顶点间的最短路径。

8.4.7 图的应用

【例 8.3】 A 城市是一个有 n 个景点的旅游胜地, 公交公司为方便游客, 在每个景点都设置了公共汽车站, 并开通了 m 条单向的公交线路。每条公交线路从设置在某个景点的公交车站出发, 途经若干个景点, 最后到达终点。设一名游客位于编号为 `start` 的景点, 他要去编号为 `end` 的景点, 他希望求解出从 `start` 到 `end` 的最少换车次数。旅游景点和对应的公交线路可用一个有向图描述, 用于创建图的数据及含义如下表所示。若两个景点之间有直达公交车, 则对应的两个顶点之间引一条权值为 1 的弧, 那么从景点 x 到景点 y 的最少上车次数就是顶点 x 到顶点 y 的最短路径长度, 最少上车次数减去 1 就是最少的换车次数。根据表中数据构造的有向图如图 8-46 所示。

输入数据	含义
7 3	共 7 个旅游景点(从 0~6 依次编号), 开设了 3 条单向公交线路
0 6	起始景点和终止景点
5 6 -1	第一条线路: 从 5 号景点出发, 到达 6 号景点, -1 表示本条线路结束
3 6 2 5 -1	第二条线路: 从 3 号景点出发, 依次经过 6 号景点、2 号景点, 到达 5 号景点
1 0 2 4 -1	第三条线路: 从 1 号景点出发, 依次经过 0 号景点、2 号景点, 到达 4 号景点

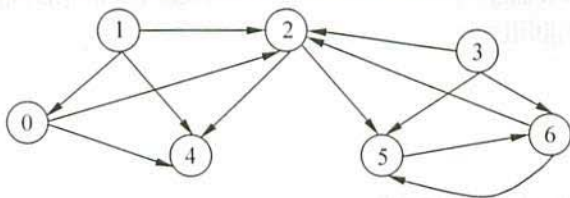
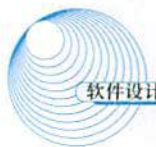


图 8-46 旅游景点及公交线路示意图

【例 8.3 程序】 求最少上车次数。

```
#include <stdio.h>
#include <stdlib.h>
#define INFINITY 9999
#define MAXVNUM 30
typedef struct{
    int Vnum;
    int arcs[MAXVNUM][MAXVNUM]; /*图的存储结构为邻接矩阵*/
}Graph;

int createGraph(Graph *g,int *start,int *end) /*读入数据, 建立有向图*/
```

```
int n,m,i,j,k,s,count;
int t[MAXVNUM];
printf("输入旅游景点数目和公交线路条数: ");
scanf("%d%d",&n,&m);
if (n <= 1 || m < 1) return -1;
printf("输入起始景点编号和终止景点编号: ");
scanf("%d%d",start,end);
if (*start < 0 || *start > n-1 || *end < 0 || *end > n-1) return -1;
g->Vnum = n;
for(i = 0; i < n; i++)
    for(j = 0; j < n; j++)
        g->arcs[i][j] = INFINITY; /*邻接矩阵初始化*/
for(s = 0; s < m; s++){
    printf("请输入第%d 条公交线路途径各景点的编号: \n",s+1);
    scanf("%d",&k); count = 0;
    while (k != -1) { /*t[]暂时存储当前公交线路途径的各个景点*/
        t[count++] = k; scanf("%d",&k);
    } /*while*/
    for(i = 0; i < count-1; i++)
        for(j = i+1; j < count; j++) /*当前线路中, 从 t[i]到 t[j]有直达公交车*/
            g->arcs[t[i]][t[j]] = 1;
} /*for*/
return 0;
} /* createGraph */

int findMinnum(Graph g,int start,int end) {
/*找出并返回图中从顶点 start 到 end 的最短路径长度(最少上车次数)*/
    int s[MAXVNUM]; /*s[i]==1 表示已求出到达景点 i 的最少上车次数*/
    int i,j,u,*dist;
    int min;
    if (start == end) return 0;
    dist = (int *)malloc(sizeof(int)); /*存储 start 到各景点的最少上车次数*/
    if (dist == NULL) return -1;
    for(i = 0; i < g.Vnum; i++) {
        dist[i] = g.arcs[start][i]; /*从 start 可直达的景点的上车次数置 1*/
        s[i] = 0; /*所有景点 i 的最少上车次数还未找到*/
    } /*for*/
    s[start] = 1; /*已找到景点 start 的最少上车次数*/
    dist[start] = 0; /*从景点 start 到 start 的最少上车次数等于 0*/
    for(i = 0; i < g.Vnum; i++) {
```

```

min = INFINITY;    u = start;
for(j = 0;j<g.Vnum;++j)
    if (s[j] == 0 && dist[j] < min) {
        min = dist[j];
        u = j;                /*u 是从 start 出发能够到达的所有景点中上车次数最少者*/
    }
s[u] = 1;            /*已经找到从景点 start 到 u 的最少上车次数, 将 u 加入集合 s*/
for(j = 0;j < g.Vnum;++j) /*更新当前情况下其他景点的最少上车次数*/
    if (s[j]==0 && min+g.arcs[u][j]<dist[j]) dist[j]=min+g.arcs[u][j];
}/*for*/
return dist[end];/*返回从景点 start 到 end 的最少上车次数*/
}/* findMinnum */

int main(void)
{ int start,end,m;
  Graph G;
  if (createGraph (&G,&start,&end)==-1){
    printf("创建有向图失败!\n");    return -1;
  }/*if*/
  m = findMinnum(G,start,end);        /*求从 start 到 end 的最少上车次数*/
  if (m < INFINITY)
    printf("\n 从景点%d 到%d 的最少换车次数为: %d",start,end,m-1);
  else printf("\n 无解!");
  return 0;
}

```

8.5 查找

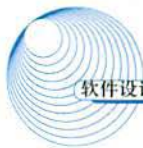
8.5.1 查找的基本概念

1. 基本概念

查找是一种常用的基本运算。查找表指由同一类型的数据元素(或记录)构成的集合。由于“集合”这种数据结构中的数据元素之间存在着完全松散的关系。因此,查找表是一种非常灵活的数据结构。

(1) 静态查找表通常将只进行以下两种操作的查找表称为静态查找表。

- ① 查询某个“特定”的数据元素是否在查找表中。
- ② 检索某个“特定”的数据元素的各种属性。



(2) 动态查找表若在查找过程中同时插入查找表中不存在的数据元素, 或者从查找表中删除已存在的某个数据元素, 则称此类查找为动态查找表。对查找表经常要进行的另外两种操作是:

- ① 在查找表中插入一个数据元素。
- ② 从查找表中删除一个数据元素。

关键字是数据元素(或记录)的某个数据项的值, 用它来识别(标识)这个数据元素(或记录)。主关键字指能唯一标识一个数据元素的关键字。次关键字指能标识多个数据元素的关键字。

根据给定的某个值, 在查找表中确定是否存在一个其关键字等于给定值的记录或数据元素。若表中存在这样的记录, 则称查找成功, 此时或者给出整个记录的信息, 或者指出记录在查找表中的位置; 若表中不存在关键字等于给定值的记录, 则称查找不成功, 此时的查找结果用一个“空”记录或“空”指针表示。

2. 平均查找长度

对于查找算法来说, 其基本操作是“将记录的关键字与给定值进行比较”。因此, 通常以“其关键字和给定值进行过比较的记录个数的平均值”作为衡量查找算法好坏的依据。

为确定记录在查找表中的位置, 需与给定关键字值进行比较的次数的期望值称为查找算法在查找成功时的平均查找长度。

对于含有 n 个记录的表, 查找成功时的平均查找长度定义为: $ASL = \sum_{i=1}^n P_i C_i$, 其中, P_i

为对表中第 i 个记录进行查找的概率, 且 $\sum_{i=1}^n P_i = 1$, 一般情况下, 均认为查找每个记录的概率是相等的, 即 $P_i = 1/n$; C_i 为找到表中其关键字与给定值相等的记录时(为第 i 个记录), 和给定值已进行过比较的关键字数。显然, C_i 随查找方法的不同而不同。

8.5.2 静态查找表

1. 顺序查找

顺序查找的基本思想是: 从表中的一端开始, 逐个进行记录的关键字和给定值的比较, 若找到一个记录的关键字与给定值相等, 则查找成功; 若整个表中的记录均比较过, 仍未找到关键字等于给定值的记录, 则查找失败。

顺序查找的方法对于顺序存储方式和链式存储方式的查找表都适用。

从顺序查找的过程可见, C_i 取决于所查记录在表中的位置。若需查找的记录正好是表中的第一个记录时, 仅需比较一次; 若查找成功时找到的是表中的最后一个记录, 则需比较 n 次。一般情况下, $C_i = n - i + 1$, 因此在等概率情况下, 顺序查找成功的平均查找长度为

$$ASL_{ss} = \sum_{i=1}^n P_i C_i = \frac{1}{n} \sum_{i=1}^n (n-i+1) = \frac{n+1}{2}$$

也就是说,成功查找的平均比较次数约为表长的一半。若所查记录不在表中,则必须进行 $n+1$ 次比较才能确定失败。

与其他查找方法相比,顺序查找方法在 n 值较大时,其平均查找长度较大,查找效率较低,但这种方法也有优点,那就是算法简单且适应面广,对查找表的结构没有要求,无论记录是否按关键字有序排列均可应用。

2. 折半查找

1) 折半查找的方法

设查找表的元素存储在一维数组 $r[1..n]$ 中,那么在表中的元素已经按关键字递增(或递减)的方式排序的情况下,进行折半查找的方法是:首先将待查元素的关键字(key)值与表 r 中间位置上(下标为 mid)的记录的关键字进行比较,若相等,则查找成功;若 $key > r[mid].key$,则说明待查记录只可能在后半个子表 $r[mid+1..n]$ 中,下一步应在后半个子表中再进行折半查找,若 $key < r[mid].key$,说明待查记录只可能在前半个子表 $r[1..mid-1]$ 中,下一步应在 r 的前半个子表中进行折半查找,这样通过逐步缩小范围,直到查找成功或子表为空时失败为止。

【函数】 设有一个整型数组中的元素是按非递减的方式排列的,在其中进行折半查找的算法为:

```
int Bsearch(int r[],int low,int high,int key)
/*元素存储在数组 r[low..high],用折半查找的方法在数组 r 中找值为 key 的元素*/
/*若找到则返回该元素的下标,否则返回-1*/
{   int mid;
    while(low <= high) {
        mid = (low+high)/2;
        if (key == r[mid]) return mid;
        else if (key < r[mid]) high = mid-1;
        else low = mid+1;
    }/*while*/
    return -1;
}/*Bsearch*/
```

【函数】 设有一个整型数组中的元素是按非递减的方式排列的,在其中进行折半查找的递归算法如下:

```
int Bsearch2(int r[],int low,int high,int key)
/*元素存储在数组 r[low..high],用折半查找的方法在数组 r 中找值为 key 的元素*/
```



```

/*若找到则返回该元素的下标, 否则返回-1*/
{   int mid;
    if (low <= high) {
        mid = (low+high)/2 ;
        if (key == r[mid]) return mid;
        else if (key<r[mid]) return Bsearch2(r,low,mid-1,key);
        else return Bsearch2(r,mid+1,high,key);
    }/*if*/
    return -1;
}/*Bsearch2*/

```

2) 折半查找的性能分析

折半查找的过程可以用一棵二叉树描述, 方法是: 以当前查找区间的中间位置上的记录作为根, 左半个子表和右半个子表中的记录分别作为根的左子树和右子树上的节点, 这样构造的二叉树称为折半查找判定树。例如具有 11 个节点的折半查找判定树如图 8-47 所示。

从折半查找判定树可以看出, 查找成功时, 折半查找的过程恰好走了一条从根节点到被查找节点的路径, 与关键字进行比较的次数即为被查找节点在树中的层数。因此, 折半查找在查找成功时进行比较的关键字数最多不超过树的深度, 而具有 n 个节点的判定树的深度为 $\lfloor \log_2 n \rfloor + 1$, 所以折半查找在查找成功时和给定值进行比较的关键字数最多为 $\lfloor \log_2 n \rfloor + 1$ 。

给判定树中所有节点的空指针域加一个指向方形节点的指针, 称这些方形节点为判定树的外部节点 (与之相对, 称那些圆形节点为内部节点), 如图 8-48 所示。那么折半查找不成功的过程就是走了一条从根节点到外部节点的路径。与给定值进行比较的关键字数等于该路径上内部节点个数。因此折半查找在查找不成功时和给定值进行比较的关键字数最多也不会超过 $\lfloor \log_2 n \rfloor + 1$ 。

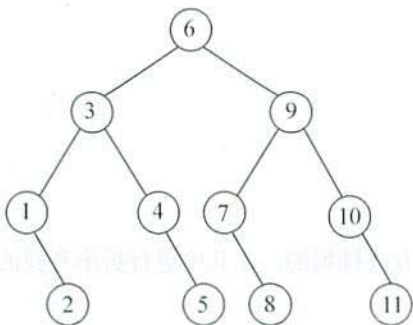


图 8-47 具有 11 个节点的折半查找判定树

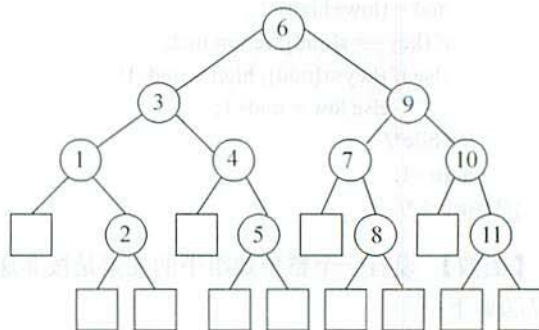


图 8-48 加上外部节点的判定树

那么折半查找的平均查找长度是多少呢? 为了方便起见, 不妨设节点总数为 $n=2^h-1$, 则判定树是深度为 $h=\log_2(n+1)$ 的满二叉树。在等概率情况下, 折半查找的平均查找长度为

$$ASL_{bs} = \sum_{j=1}^n P_j C_j = \frac{1}{n} \sum_{j=1}^n j \times 2^{j-1} = \frac{n+1}{n} \log_2(n+1) - 1$$

当 n 值较大时, $ASL_{bs} \approx \log_2(n+1) - 1$ 。

折半查找比顺序查找的效率要高, 但它要求查找表进行顺序存储并且按关键字有序排列。因此, 当需要对表进行元素的插入或删除操作时, 需要移动大量的元素。所以折半查找适用于表不易变动且又经常进行查找的情况。

3. 分块查找

分块查找又称索引顺序查找, 是对顺序查找方法的一种改进, 其效率介于顺序查找与折半查找之间。

在分块查找过程中, 首先将表分成若干块, 每一块中的关键字不一定有序, 但块之间是有序的, 即后一块中所有记录的关键字均大于前一个块中最大的关键字; 此外, 还建立了一个“索引表”, 索引表按关键字有序。图 8-49 所示的是一个表及其索引表。

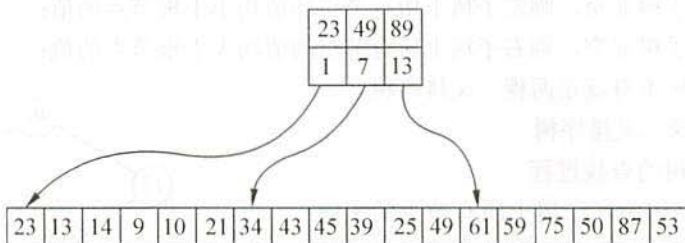


图 8-49 表及其索引表

因此, 查找过程分为两步: 第一步在索引表中确定待查记录所在的块; 第二步在块内顺序查找。

由于分块查找实际上是两次查找的过程, 因此整个分块查找的平均查找长度应该是两次查找的平均查找长度(块内查找与索引查找)之和, 所以分块查找的平均查找长度为

$$ASL_{bs} = L_b + L_w$$

其中 L_b 为查找索引表的平均查找长度, L_w 为块内查找时的平均查找长度。

分块查找时可将长度为 n 的表均匀地分成 b 块, 每块含有 s 个记录, 即有 $b = \left\lceil \frac{n}{s} \right\rceil$ 。在等概率的情况下, 块内查找的概率为 $\frac{1}{s}$, 每块的查找概率为 $\frac{1}{b}$, 若用顺序查找确定元素所在的块,



则分块查找的平均查找长度为

$$ASL_{bs} = L_b + L_w = \frac{1}{b} \sum_{j=1}^b j + \frac{1}{s} \sum_{i=1}^s i = \frac{b+1}{2} + \frac{s+1}{2} = \frac{1}{2} \left(\frac{n}{s} + s \right) + 1$$

可见其平均查找长度在这种条件下不仅与表长 n 有关, 而且和每一块中的记录数 s 有关。可以证明, 当 s 取 \sqrt{n} 时, ASL_{bs} 取最小值 $\sqrt{n} + 1$, 这时的查找效率较顺序查找要好得多, 但远不及折半查找。

考虑到索引表是一个有序表, 因此可以用折半查找确定元素所在的块。

8.5.3 动态查找表

动态查找表的特点表结构本身在查找过程中动态生成的, 即对于给定值 key , 若表中存在关键字等于 key 的记录, 则查找成功返回, 否则插入关键字为 key 的记录。

1. 二叉排序树

1) 二叉排序树的定义

二叉排序树又称二叉查找树, 它或者是一棵空树, 或者是具有如下性质的二叉树:

- ① 若它的左子树非空, 则左子树上所有节点的值均小于根节点的值;
- ② 若它的右子树非空, 则右子树上所有节点的值均大于根节点的值;
- ③ 左、右子树本身就是两棵二叉排序树。

图 8-50 为一棵二叉排序树。

2) 二叉排序树的查找过程

因为二叉排序树中左子树上所有节点的关键字均小于根节点的关键字; 右子树上所有节点的关键字均大于根节点的关键字, 所以在二叉排序树上进行查找, 与折半查找过程类似。

在二叉排序树上进行查找的过程为: 若二叉排序树非空, 将给定值与根节点的关键字值相比较, 若相等, 则查找成功; 若不等, 则当根节点的关键字值大于给定值时, 到根的左子树中进行查找; 否则到根的右子树中进行查找。若找到, 则查找过程是走了一条从树根到所找到节点的路径; 否则, 查找过程终止于一棵空树。

设二叉排序树以二叉链表为存储结构, 节点的类型定义如下:

```
typedef struct Tnode{  
    int data; /*节点的键值*/
```

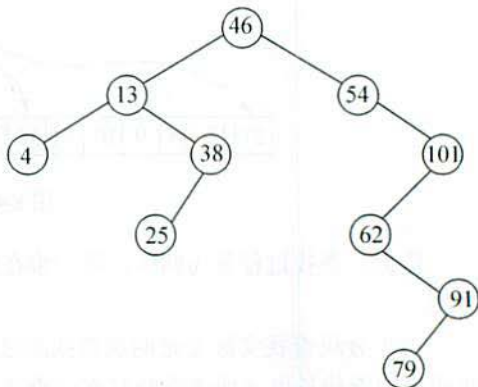


图 8-50 一棵二叉排序树

```

    struct Tnode *lchild,*rchild;           /*指向左、右子树的指针*/
}Tnode, *Bitree;

```

【函数】 二叉排序树的查找算法。

```

Bitree SearchBST(Bitree root,int key,Bitree *father)
/*在 root 指向根的二叉排序树中查找键值为 key 的节点*/
/*若找到, 则返回该节点的指针, 否则返回 NULL*/
{
    Bitree p = root; *father = NULL;
    while (p && p->data!=key) {
        *father = p;
        if (key>p->data) p = p->lchild;
        else p = p->rchild;
    }/*while*/
    return p;
}/*SearchBST*/

```

3) 二叉排序树中插入节点的操作

二叉排序树是通过依次输入数据元素并把它们插入到二叉树的适当位置上构造起来的, 具体的过程是: 每读入一个元素, 建立一个新节点。若二叉排序树非空, 则将新节点的值与根节点的值相比较, 如果小于根节点的值, 则插入到左子树中, 否则插入到右子树中; 若二叉排序树为空, 则新节点作为二叉排序树的根节点。设关键字序列为{46, 25, 54, 13, 29, 91}, 则整个二叉排序树的构造过程如图 8-51 所示。

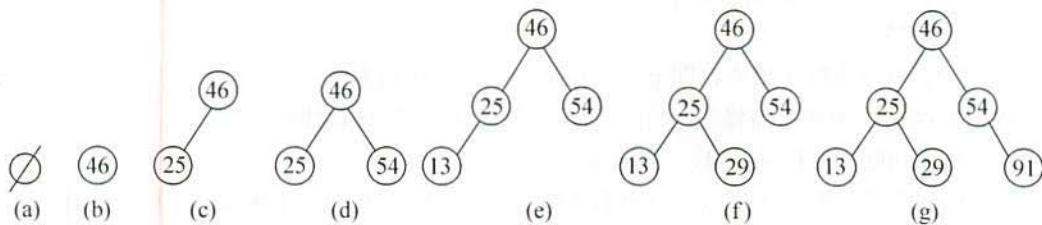


图 8-51 二叉排序树的构造过程

【函数】 二叉排序树的插入算法。

```

int InsertBST(Bitree *root,int e)
/*在*root 指向根的二叉排序树中插入一个键值为 e 的节点, 若插入成功返回 0, 否则返回-1*/
{
    Bitree s,p,f;
    s = (Bitree)malloc(sizeof(TNode));

```



```

if (s == NULL) return -1;
s->data = e; s->lchild = NULL; s->rchild = NULL;
p = SearchBST(*root, e, &f);          /*寻找插入位置*/
if (p != NULL) return -1;             /*键值为 e 的节点已在树中, 不再插入*/
if (f == NULL) *root = s;             /*若为空树, 键值为 e 的节点为树根*/
else if (e < f->data) f->lchild = s;    /*作为左孩子节点插入*/
      else f->rchild = s;              /*作为右孩子节点插入*/
return 0;
} /*InsertBST*/

```

从上面的插入过程还可以看到, 每次插入的新节点都是二叉排序树上新叶子节点, 则在插入操作时, 不必移动其他节点, 仅需改动某个节点的指针。这就相当于在一个有序序列上插入一个记录而不需要移动其他记录。它表明, 二叉排序树具有类似于折半查找的特性, 可采用链表存储结构, 因此是动态查找表的一种适宜表示。

另外, 由于一棵二叉排序树的形态完全由输入序列决定, 所以在输入序列已经有序的情况下, 所构造的二叉排序树是一棵单枝树。从二叉排序树的查找过程可知, 这种情况下的查找效率和顺序查找的效率是相同的。

4) 二叉排序树中删除节点的操作

在二叉排序树中删除一个节点, 不能把以该节点为根的子树都删除, 只能删除这个节点并仍旧保持二叉排序树的特性, 也就是说删除二叉排序树上一个节点相当于删除有序序列中的一个元素。

假设在二叉排序树删除节点*p (p 指针指向被删除节点), *f 为其双亲节点, 则该操作过程可分为 3 种情况:

① 若*p 节点为叶子节点, 即 p->lchild 及 p->rchild 均为空, 则由于删去叶子节点后不破坏整棵树的结构, 因此只需修改*p 节点的双亲节点*f 的相应指针即可:

f->lchild (或 f->rchild) = NULL;

② 若*p 节点只有左子树或者只有右子树, 此时只要将*p 的左子树或右子树接成其双亲节点*f 的左子树 (或右子树), 即令

f->lchild (或 f->rchild) = p->lchild;

或

f->lchild (或 f->rchild) = p->rchild;

③ 若*p 节点的左、右子树均不空, 则删除*p 节点时应将其左子树、右子树连接到适当的位置, 并保持二叉排序树的特性。可采用如下两种方法进行处理: 一是令*p 的左子树为*f 的左 (或右) 子树, 而将*p 的右子树下接到*p 的中序遍历的直接前驱节点*s (*s 节点是*p 的左子树

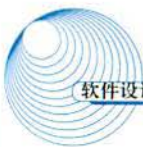
中最右边的节点)的右孩子指针上;二是用*p的中序直接前驱(或后继)节点*s代替*p节点,然后删除*s节点。

【函数】 二叉排序树的删除操作,用*p的中序前驱*s的元素值代替*p节点的元素值,然后删除*s节点。

```

int DeleteBST(Bitree *root,int e)
/*在*root指向根的二叉排序树中删除键值为e的节点,若成功则返回0,否则返回-1*/
{
    Bitree s,p,f;
    p = SearchBST(*root,e,&f);          /*寻找待删除节点*/
    if (p == NULL) return -1;           /*键值为e的节点不在树中,无法删除*/
    if (!p->lchild) {                    /*待删节点无左子树*/
        if (f == NULL) {                 /*待删除的节点是树根*/
            *root = p->rchild; free(p); return 0;
        }
        if (p == f->lchild) f->lchild = p->rchild;
        else f->rchild = p->rchild;
        free(p); return 0;
    }
    else
        if (!p->rchild) { /*待删节点无右子树*/
            if (f == NULL) { /*待删除的节点是树根*/
                *root = p->lchild; free(p); return 0;
            }
            if (p == f->lchild) f->lchild = p->lchild;
            else f->rchild = p->lchild;
            free(p); return 0;
        }
        else { /*待删节点既有左子树又有右子树*/
            q = p; s = p->lchild;
            while (s->rchild) {q = s; s = s->rchild;}
            p->data = s->data;
            if (q != p) q->rchild = s->lchild;
            else q->lchild = s->lchild;
            free(s); return 0;
        }
    }
}
/*DeleteBST*/
  
```

从二叉排序树的定义可知,中序遍历二叉排序树可得到一个关键字有序的序列。这也说明,



一个无序序列可以通过构造一棵二叉排序树而变成一个有序序列,构造树的过程即为对无序序列进行排序的过程。

【例 8.4】 编写一个程序,从正文文件 in.txt 读入一篇英文短文,统计并输出该短文中不同单词出现的次数,然后将单词按词典序输出。

由于不知道文件中不同单词的数目,因此可采用二叉排序树存储读入的单词并通过查找二叉排序树对单词进行计数。二叉链表中节点结构定义为

word	count	left	right
------	-------	------	-------

其中: word 用于存放一个单词, count 是该单词出现的次数。left 和 right 是指针, left 指向在 word 之后出现但比 word 小的第一个单词的节点, right 指向在 word 之后出现但比 word 大的第一个单词的节点。因此,当正文扫描结束后,按照单词出现的先后次序构造了一个二叉排序树。对二叉排序树进行中序遍历,便可以得到一个依词典序递增的单词序列。

【例 8.4 程序】 利用二叉排序树统计正文中的单词并按词典序对单词进行排序。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define INF "in.txt"
typedef struct Node {
    char *word; /*用于存放一个单词*/
    int count; /*计数器,单词在文中出现的次数*/
    struct Node *left,*right;
}Node,*BiTree;

int searchBst(BiTree T,char *keyword,BiTree *p)
/*在二叉排序树中(*T 指向树根节点)查找单词 keyword,若找到,令 p 指向该节点并返回 1*/
/*否则,p 指向查找路径上出现的最后一个节点并返回 0 */
{ int cmpres = 0;
  BiTree ptr;
  *p = NULL;ptr = T;
  while (ptr) {
    cmpres = strcmp(ptr->word,keyword);
    if (cmpres == 0) /*查找成功,令 p 指向该节点,返回 1*/
    { *p = ptr; return 0; }
    else { *p = ptr;
          ptr = cmpres > 0 ? ptr->left : ptr->right;
        }
  }
}/*while*/
```

```

    return -1;                                /*查找失败*/
}/* searchBst */

int insertBst(BiTree *t,char *keyword)
/*在二叉排序树中(*T 指向树根节点)插入单词为 keyword 的节点*/
/*若该单词节点已在树中, 则计数器增 1; 否则, 在树中插入一个单词为 keyword 的节点*/
{
    BiTree s,p;
    if (searchBst(*t,keyword,&p) == -1) { /*找不到单词 keyword 的节点*/
        s = (Node *)malloc(sizeof(Node));
        if (!s) {printf("存储分配失败!\n"); return -1;}
        s->word = (char *)malloc(strlen(keyword));
        strcpy(s->word,keyword);
        s->left = NULL; s->right = NULL; s->count = 1;
        if (p == NULL) *t = s; /*keyword 是第一个出现的单词,令 s 作为根节点*/
        else if (strcmp(p->word,keyword) < 0) /*keyword 比当前节点的单词大*/
            p->right = s; /*将 keyword 插入当前节点的右子树上*/
        else p->left = s; /*将 keyword 插入当前节点的左子树上*/
    }/*if*/
    else p->count++;
    return 0;
}/*insertBst*/

void InOrder(BiTree root) /*中序遍历 root 指向根的二叉排序树*/
{
    if (root) {
        InOrder(root->left);
        printf(" %s (%d)\n", root->word, root->count);
        InOrder(root->right);
    }/*if*/
}/* InOrder */

void main(void)
{
    char ch,*word,buffer[100];
    FILE *fin;
    BiTree root = NULL;
    int i = 0,tag; /*i 用于计算每个单词的长度, tag 标识当前读入的是英文字母*/
    fin = fopen(INF,"r");
    if (!fin) {printf("open file %s error!\n",INF); return;}
    while (!feof(fin)) {

```




```
ch = fgetc(fin);
if ((ch>='a' && ch<='z') || (ch>='A' && ch<='Z')) {
    buffer[i++] = ch; tag = 1;
}
else if (tag) { /*将存储在 buffer 中的单词插入二叉排序树*/
    buffer[i] = '\0';
    word = (char *)malloc(strlen(buffer));
    strcpy(word, buffer);
    if (insertBst(&root, word) == -1) return;
    i = 0; tag = 0;
} /*if tag*/
} /*while*/
fclose(fin);
InOrder(root);
}
```

2. 平衡二叉树

平衡二叉树又称为 AVL 树, 它或者是一棵空树, 或者是具有下列性质的二叉树: 它的左子树和右子树都是平衡二叉树, 且左子树和右子树的深度之差的绝对值不超过 1。若将二叉树节点的平衡因子 (Balance Factor, BF) 定义为该节点的左子树的深度减去其右子树的深度, 则平衡二叉树上所有节点的平衡因子只可能是 -1、0 和 1。只要树上有一个节点的平衡因子的绝对值大于 1, 则该二叉树就是不平衡的。

分析二叉排序树的查找过程可知, 只有在树的形态比较均匀的情况下, 查找效率才能达到最佳。因此, 希望在构造二叉排序树的过程中, 保持其为一棵平衡二叉树。

使二叉排序树保持平衡的基本思想是: 每当在二叉排序树中插入一个节点时, 首先检查是否因插入而破坏了平衡。若是, 则找出其中的最小不平衡二叉树, 在保持二叉排序树特性的情况下, 调整最小不平衡子树中节点之间的关系, 以达到新的平衡。所谓最小不平衡子树指离插入节点最近且以平衡因子的绝对值大于 1 的节点作为根的子树。

1) 平衡二叉树上的插入操作

一般情况下, 假设由于在二叉排序树上插入节点而失去平衡的最小子树根节点的指针为 *a*, 也就是说, *a* 所指节点是离插入节点最近且平衡因子的绝对值超过 1 的祖先节点, 那么, 失去平衡后进行调整的规律可归纳为以下 4 种情况:

(1) 单向右旋平衡处理: 由于在 **a* 的左子树的左子树上插入新节点, **a* 的平衡因子由 1 增至 2, 导致以 **a* 为根的子树失去平衡, 所以需进行一次向右的顺时针旋转操作, 如图 8-52 所示。

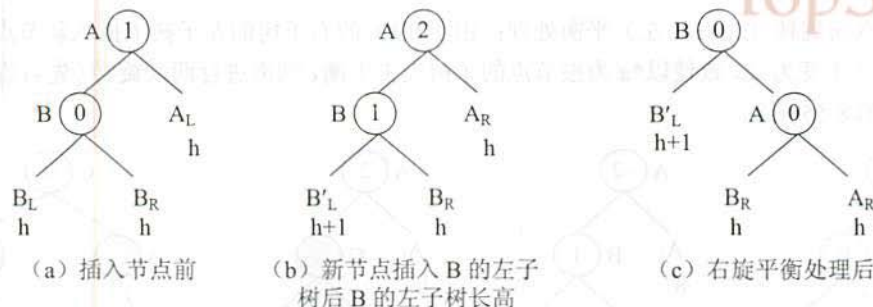


图 8-52 单向右旋平衡处理示意图

(2) 单向左旋平衡处理: 由于在*a 的右子树的右子树上插入新节点, *a 的平衡因子由-1 变为-2, 导致以*a 为根的子树失去平衡, 所以需进行一次向左的逆时针旋转操作, 如图 8-53 所示。

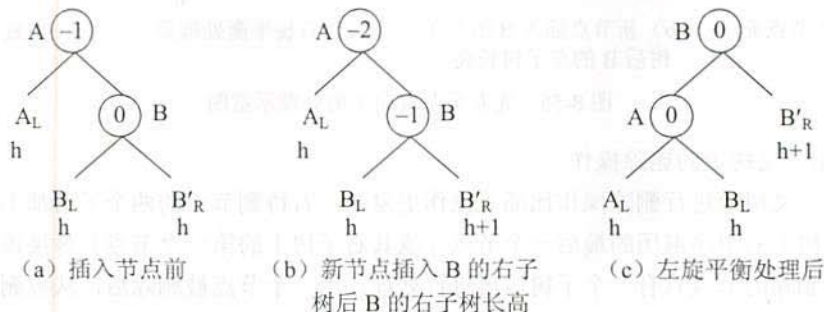


图 8-53 单向左旋平衡处理示意图

(3) 双向旋转(先左后右)平衡处理: 由于在*a 的左子树的右子树上插入新节点, *a 的平衡因子由 1 增至 2, 致使以*a 为根节点子树失去平衡, 所以需进行两次旋转(先左旋后右旋)操作, 如图 8-54 所示。

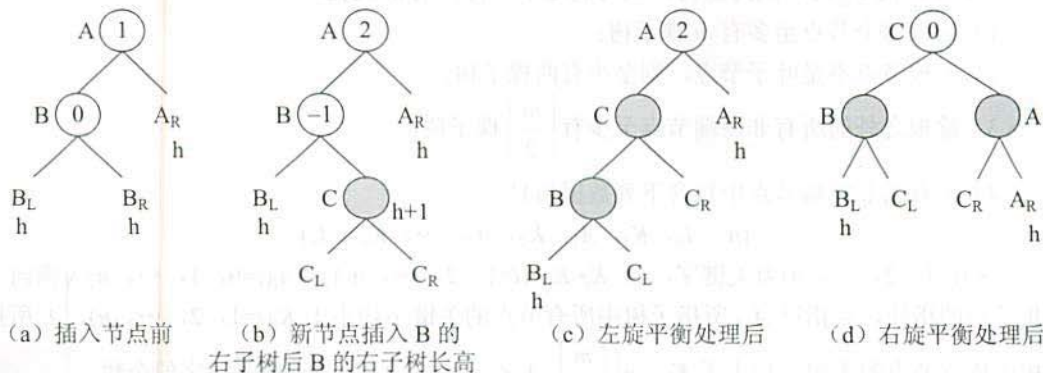


图 8-54 先左后右双向平衡处理示意图

(4) 双向旋转(先右后左)平衡处理: 由于在*a 的右子树的左子树上插入新节点, *a 的平衡因子由-1 变为-2, 致使以*a 为根节点子树失去平衡, 则需进行两次旋转(先右旋后左旋)操作, 如图 8-55 所示。

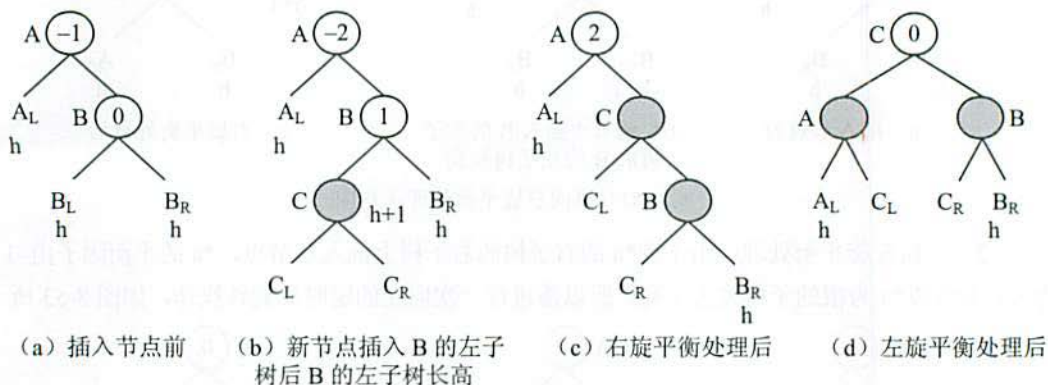


图 8-55 先右后左双向平衡处理示意图

2) 平衡二叉树上的删除操作

在平衡二叉树上进行删除操作比插入操作更复杂。若待删节点的两个子树都不为空, 就用该节点左子树上的中序遍历的最后一个节点(或其右子树上的第一个节点)替换该节点, 将情况转化为待删除的节点只有一个子树后再进行处理。当一个节点被删除后, 从被删节点到树根的路径上所有节点的平衡因子都需要更新。对于每一个位于该路径上的平衡因子为 ± 2 的节点来说, 都要进行平衡处理。

3. B_树

一棵 m 阶的 B_树, 或为空树, 或为满足下列特性的 m 叉树:

- (1) 树中每个节点至多有 m 棵子树;
- (2) 若根节点不是叶子节点, 则至少有两棵子树;
- (3) 除根之外的所有非终端节点至少有 $\left\lceil \frac{m}{2} \right\rceil$ 棵子树;
- (4) 所有的非终端节点中包含下列数据信息

$$(n, A_0, K_1, A_1, K_2, A_2, \dots, K_n, A_n)$$

其中: $K_i(i=1, 2, \dots, n)$ 为关键字, 且 $K_i < K_{i+1}(i=1, 2, \dots, n-1)$; $A_i(i=0, 1, \dots, n)$ 为指向子树根节点的指针, 且指针 A_{i-1} 所指子树中所有节点的关键字均小于 $K_i(i=1, 2, \dots, n)$, A_n 所指子树中所有节点的关键字均大于 K_n , $n \left(\left\lceil \frac{m}{2} \right\rceil - 1 \leq n \leq m-1 \right)$ 为节点中关键字的个数。

(5) 所有的叶子节点都出现在同一层次上, 并且不带信息(可以看作是外部节点或查找失败的节点, 实际上这些节点不存在, 指向这些节点的指针为空)。

一棵4阶的B₊树如图8-56所示。

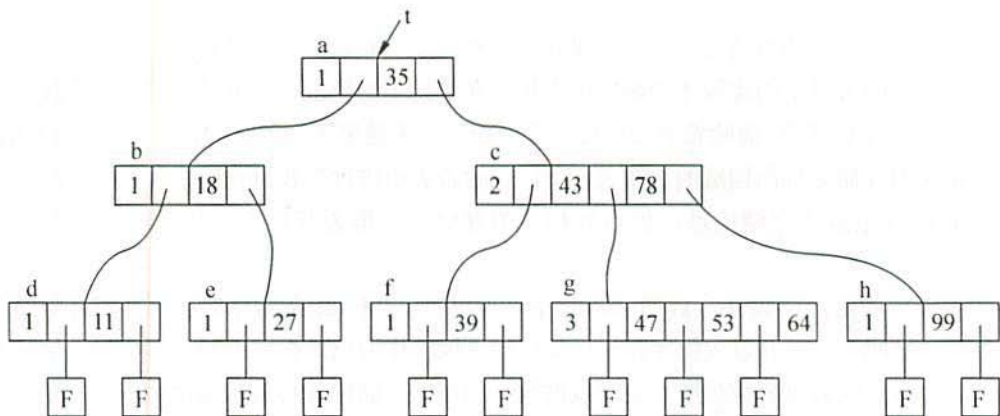


图 8-56 一棵4阶B₊树

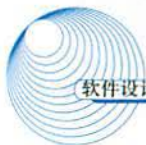
由B₊树的定义可知, 在B₊树上进行查找的过程是: 首先在根节点所包含的关键字中查找给定的关键字, 若找到则成功返回; 否则确定待查找的关键字所在的子树并继续进行查找, 直到查找成功或查找失败(指针为空)时为止。

B₊树上的插入和删除运算较为复杂, 因为要保证运算后节点中关键字的个数大于等于 $\left\lceil \frac{m}{2} \right\rceil - 1$, 因此涉及到节点的“分裂”及“合并”问题。

在B₊树中插入一个关键字时, 不是在树中增加一个叶子节点, 而是首先在低层的某个非终端节点中添加一个关键字, 若该节点中关键字的个数不超过 $m-1$, 则完成插入; 否则, 要进行节点的“分裂”处理。所谓“分裂”, 就是把节点中处于中间位置上的关键字取出来插入到其父节点中, 并以该关键字为分界线, 把原节点分成两个节点。“分裂”过程可能会一直持续到树根。

同样, 在B₊树中删除一个节点时, 首先找到关键字所在的节点, 若该节点在含有信息的最后一层, 且其中关键字的数目不少于 $\left\lceil \frac{m}{2} \right\rceil - 1$, 则完成删除; 否则需进行节点的“合并”运算。

若待删除的关键字所在节点不在含有信息的最后一层上, 则将该关键字用其在B₊树中的后继替代, 然后再删除其后继元素, 即将需要处理的情况统一转化为在含有信息的最后一层再进行删除运算。



8.5.4 哈希表及其查找

1. 哈希表的定义

前面讨论的几种查找方法,由于记录在存储结构中的相对位置是随机的,所以在查找时都要通过一系列的关键字的比较才能确定被查记录在结构中的位置。也就是说,这类查找都是以关键字的比较为基础的,而哈希表则通过一个以记录的关键字为自变量的函数(称为哈希函数)得到该记录的存储地址而构造的查找表,所以在哈希表中进行查找操作时,须用同一哈希函数计算得到待查记录的存储地址,然后到相应的存储单元里去获得有关信息再判定查找是否成功。

根据设定的哈希函数 $H(\text{key})$ 和处理冲突的方法,将一组关键字映射到一个有限的连续的地址集(区间)上,并以关键字在地址集中的“像”作为记录在表中的存储位置,这种表称为哈希表,这一映射过程称为哈希造表或散列,所得的存储位置称为哈希地址或散列地址。

对于某个哈希函数 H 和两个关键字 K_1 和 K_2 ,如果 $K_1 \neq K_2$,而 $H(K_1) = H(K_2)$,这种现象称为冲突。具有相同函数值的关键字对该哈希函数来说称为同义词。

一般情况下,冲突只能尽可能减少而不能完全避免。因为哈希函数是从关键字集合到地址集合的映像。通常关键字集合比较大,它的元素包含所有可能的关键字,而地址集合的元素仅为哈希表中的地址值。假设关键字集合为某种高级语言的所有标识符,如果一个标识符对应一个存储地址,那就不会发生冲突了,但这是不可能也没有必要的,因为存储空间难以满足,而且任何一个源程序都不会使用这么多标识符。因此在一般情况下,哈希函数是一个压缩映像,冲突是不可避免的。所以在建造哈希表时不仅要设定一个“好”的哈希函数,而且要设定一种处理冲突的方法。

对于哈希法,主要考虑两个问题:其一是如何构造哈希函数,其二是如何解决冲突。对于哈希函数的构造,应解决好两个主要问题:

- (1) 哈希函数应是一个压缩映像函数,它应具有较大的压缩性,以节省存储空间;
- (2) 哈希函数应具有较好的散列性,虽然冲突是不可避免的,但应尽量减少。

2. 哈希函数的构造方法

要减少冲突,就要设法使哈希函数尽可能均匀地把关键字映射到符号表存储区的各个存储地址上,这样就可以提高查找效率。构造哈希函数时,一般都要对关键字进行计算。为了尽量避免产生相同的哈希函数值,应使关键字的所有组成部分都能起作用。

常用的哈希函数构造方法有直接定址法、数字分析法、平方取中法、折叠法、随机数法

和除留余数法等。

3. 处理冲突的方法

解决冲突就是为出现冲突的关键字找到另一个“空”的哈希地址。在处理冲突的过程中可能得到一个地址序列 H_i ($i=1, 2, \dots, k$)。常见的处理冲突的方法有:

(1) 开放定址法

$$H_i = (H(\text{key}) + d_i) \% m \quad i=1, 2, \dots, k \quad (k \leq m-1)$$

其中: $H(\text{key})$ 为哈希函数; m 为哈希表表长; d_i 为增量序列。

常见的增量序列有 3 种:

- ① $d_i=1, 2, 3, \dots, m-1$, 称为线性探测再散列;
- ② $d_i=1^2, -1^2, 2^2, -2^2, 3^2, \dots, \pm k^2$ ($k \leq m/2$), 称为二次探测再散列;
- ③ d_i 为伪随机序列, 称为随机探测再散列。

最简单的产生探测序列的方法是进行线性探测。也就是发生冲突时, 顺序地到存储区的下一个单元进行探测。

例如, 某记录的关键字为 key , 哈希函数值 $H(\text{key})=j$ 。若在哈希地址 j 发生了冲突 (即此位置已存放了其他元素), 则对哈希地址 $j+1$ 进行探测, 若仍然有冲突, 再对地址 $j+2$ 进行探测, 依次类推, 直到将元素存入哈希表。

例如, 设关键码序列为 47、34、13、12、52、38、33、27、3, 哈希表表长为 11, 哈希函数为: $\text{Hash}(\text{key})=\text{key} \bmod 11$, 则

$$\text{Hash}(47) = 47 \bmod 11 = 3, \quad \text{Hash}(34) = 34 \bmod 11 = 1,$$

$$\text{Hash}(13) = 13 \bmod 11 = 2, \quad \text{Hash}(12) = 12 \bmod 11 = 1,$$

$$\text{Hash}(52) = 52 \bmod 11 = 8, \quad \text{Hash}(38) = 38 \bmod 11 = 5,$$

$$\text{Hash}(33) = 33 \bmod 11 = 0, \quad \text{Hash}(27) = 27 \bmod 11 = 5,$$

$$\text{Hash}(3) = 3 \bmod 11 = 3$$

使用线性探测法解决冲突构造的哈希表如下:

哈希地址	0	1	2	3	4	5	6	7	8	9	10
关键字	33	34	13	47	12	38	27	3	52		

关键字由哈希函数得到的 47、34、13、52、38、33 的哈希地址没有冲突, 元素直接存入。

对于元素 12, 其哈希地址为 1, 但是该地址中已经存入元素 34, 因此由 $H_1 = (\text{Hash}(12) + 1) \bmod 11 = 2$, 再试探哈希地址 2, 但该地址已被元素 13 占用, 发生冲突; 再计算 $H_2 = (\text{Hash}(12) + 2) \bmod 11 = 3$, 发生冲突 (地址 3 被元素 47 占用); 再计算 $H_3 = (\text{Hash}(12) + 3) \bmod 11 = 4$, 空闲, 因此将元素 12 存入哈希地址为 4 的单元。元素 27 和 3 也是通过解决冲突后存入的。

线性探测法可能使第 i 个哈希地址的同义词存入第 $i+1$ 个哈希地址, 这样本应存入第 $i+1$ 个哈希地址的元素变成了第 $i+2$ 个哈希地址的同义词……因此, 可能出现很多元素在相邻的哈希地址上“聚集”起来的现象, 大大降低了查找效率。为此, 可采用二次探测法或随机探测再散列法, 以改善“聚集”问题。

那么在查找时就有 3 种可能: 第一种情况是在某一位置上查到了关键字等于 key 的记录, 查找成功; 第二种情况是按探测序列查不到关键字为 key 的记录而又遇到了空单元, 这时表明元素不在表中, 表示查找失败; 第三种情况是查遍全表, 未查到指定关键字且符号表存储区已满, 需进行溢出处理。

线性探测法思路清楚, 算法简单, 但也存在以下缺点。

① 溢出处理需另编程序: 一般可另外设立一个溢出表, 专门用来存放上述哈希表中放不下的记录。实现溢出表的最简单的结构是顺序表, 查找方法可用顺序查找。

② 线性探测法很容易产生聚集现象: 所谓聚集现象就是存入哈希表的记录在表中连成一片。当哈希函数不能把关键字很均匀地散列到哈希表中时, 尤其容易产生聚集现象。产生聚集现象后, 会增加探测的次数, 从而降低了查找效率。

可以采取多种方法减少聚集现象的产生, 二次探测再散列和随机探测再散列是两种有效的方法。

(2) 链地址法

链地址法是一种经常使用且很有效的方法。它在符号表的每一个记录中增加一个链域, 链域中存放下一个具有相同哈希函数值记录的存储地址。利用链域就把若干个发生冲突的记录链接在一个链表内。当链域的值为 NULL 时, 表示已没有后继记录了。因此, 对于发生冲突时的查找和插入操作就跟线性表一样了。

例如, 哈希表表长为 11、哈希函数为 $\text{Hash}(\text{key}) = \text{key} \bmod 11$, 对于关键码序列 47、34、13、12、52、38、33、27、3, 使用链地址法构造的哈希表如图 8-57 所示。

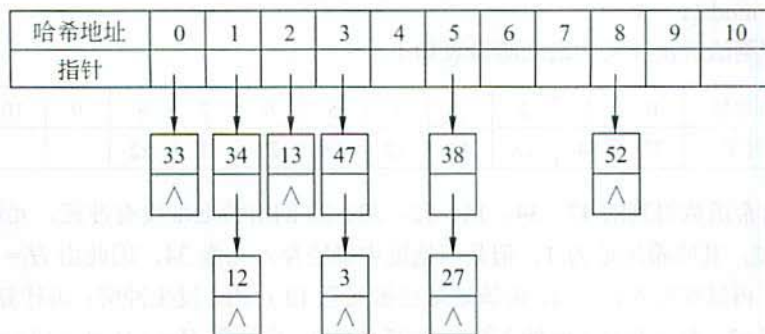


图 8-57 用链地址法解决冲突构造哈希表

在图 8-57 所示的哈希表中进行成功查找的平均查找长度 ASL 为:

$$ASL = (6 \times 1 + 3 \times 2) / 9 \approx 1.34$$

(3) 再哈希法

$$H_i = RH_i(\text{key}) \quad (i=1, 2, \dots, k)$$

RH_i 均是不同的哈希函数, 即在同义词发生地址冲突时计算另一个哈希函数地址, 直到冲突不再发生。这种方法不易产生聚集现象, 但增加了计算时间。

(4) 建立一个公共溢出区

无论由哈希函数得到的哈希地址是什么, 一旦发生冲突, 都填入到公共溢出区中。

3. 哈希表的查找

从哈希表的查找过程可知:

(1) 虽然哈希表在关键字与记录的存储位置之间建立了直接映像, 但由于“冲突”的产生, 使得哈希表的查找过程仍然是一个给定值和关键字进行比较的过程。因此, 仍需以平均查找长度衡量哈希表的查找效率。

(2) 查找过程中需和给定值进行比较的关键字的个数取决于下列 3 个因素: 哈希函数、处理冲突的方法和哈希表的装填因子。

在一般情况下, 冲突处理方法相同的哈希表, 其平均查找长度依赖于哈希表的装填因子。哈希表的装填因子定义为

$$\alpha = \frac{\text{表中装入的记录数}}{\text{哈希表的长度}}$$

α 标志哈希表的装满程度。直观地看, α 越小, 发生冲突的可能性就越小; 反之, α 越大, 表中已填入的记录越多, 再填记录时, 发生冲突的可能性就越大, 则查找时, 给定值需与之进行比较的关键字的个数也就越多。

8.6 排序

8.6.1 排序的基本概念及运算

(1) 排序: 假设含 n 个记录的文件内容为 $\{R_1, R_2, \dots, R_n\}$, 其相应的关键字为 $\{k_1, k_2, \dots, k_n\}$ 。经过排序确定一种排列 $\{R_{j_1}, R_{j_2}, \dots, R_{j_n}\}$, 使得它们的关键字满足如下递增 (或递减) 关系: $k_{j_1} \leq k_{j_2} \leq \dots \leq k_{j_n}$ (或 $k_{j_1} \geq k_{j_2} \geq \dots \geq k_{j_n}$), 这样的运算称为排序。

若在待排序的一组序列中, k_i 和 k_j 的关键字相同, 即 $k_i = k_j$, 且在排序前 k_i 领先于 k_j , 那么当排序后, 如果 k_i 和 k_j 的相对次序保持不变, k_i 仍领先于 k_j , 则称此类排序方法为稳定的。若

在排序后的序列中有可能出现 k_j 领先于 k_i 的情形, 则称此类排序为不稳定的。

(2) 内部排序: 指待排序记录全部存放在内存中进行排序的过程。

(3) 外部排序: 指待排序记录的数量很大, 以至内存不能容纳全部记录, 在排序过程中尚需对外存进行访问的排序过程。

在排序过程中需进行下列两种基本操作: 比较两个关键字的大小; 将记录从一个位置移动到另一个位置。前一种操作对大多数排序方法来说都是必要的, 而后一种操作可以通过改变记录的存储方式来予以避免。

8.6.2 简单排序

1. 直接插入排序

直接插入排序是一种简单的排序方法, 具体做法是: 在插入第 i 个记录时, R_1, R_2, \dots, R_{i-1} 已经排好序, 这时将 R_i 的关键字 k_i 依次与关键字 $k_{i-1}, k_{i-2}, \dots, k_1$ 进行比较, 从而找到应该插入的位置, 然后将 R_i 插入, 插入位置及其后的记录依次向后移动。

直接插入排序法在最好情况下 (待排序列已按关键码有序), 每趟排序只需作 1 次比较且不需要移动元素, 因此 n 个元素排序时的总比较次数为 $n-1$ 次, 总移动次数为 0 次。在最坏情况下 (元素已经逆序排列), 进行第 j 趟排序时, 待插入的记录需要同前面的记录进行 j 次比较, 移动记录的次数为 $j+1$ 。因此, 总比较次数为 $\sum_{i=2}^n i = \frac{(n+2)(n-1)}{2}$, 总移动次数为 $\sum_{i=2}^n (i+1) = \frac{(n+3)(n-2)}{2}$ 。

由此, 直接插入排序是一种稳定的排序方法, 其时间复杂度为 $O(n^2)$ 。排序过程中仅需要一个元素的辅助空间, 空间复杂度为 $O(1)$ 。

2. 冒泡排序

n 个记录进行冒泡排序的方法是: 首先将第一个记录的关键字和第二个记录的关键字进行比较, 若为逆序, 则交换两个记录的值, 然后比较第二个记录和第三个记录的关键字, 依次类推, 直至第 $n-1$ 个记录和第 n 个记录的关键字进行过比较为止。上述过程称作第一趟冒泡排序, 其结果是关键字最大的记录被放置到第 n 个记录的位置上。然后进行第二趟冒泡排序, 对前 $n-1$ 个记录进行同样的操作, 其结果是关键字次大的记录被放置到第 $n-1$ 个记录的位置上。最多进行 $n-1$ 趟, 所有记录有序排列。若在某趟冒泡排序过程没有进行相邻位置的元素交换处理, 则可结束排序过程。

【算法】 冒泡排序算法。

```
void bubblesort(int data[], int n)
```

```
/*将数组 data 中的 n 个整数按非递减有序的方式进行排列*/
```

```

{ int i,j,tag,temp;                /*tag 等于 1 表示上一趟排序过程中存在交换处理*/
  for(i = 0,tag = 1;tag == 1 && i < n-1;i++){
    tag = 0;
    for(j = 0;j < n-i-1;j++){
      if (data[j] > data[j+1]){
        temp = data[j]; data[j] = data[j+1]; data[j+1] = temp;
        tag = 1;
      }/*if*/
    }/*for*/
  }/*bubblesort*/

```

冒泡排序法在最好情况下(待排序列已按关键字有序),只需作 1 趟排序,元素的比较次数为 $n-1$ 且不需要交换元素,因此总比较次数为 $n-1$ 次,总交换次数为 0 次。在最坏情况下(元素已经逆序排列),进行第 j 趟排序时,最大的 $j-1$ 个元素已经排好序,其余的 $n-(j-1)$ 个元素需要进行 $n-j$ 次比较和 $n-j$ 次交换,因此总比较次数为 $\sum_{j=1}^{n-1} (n-j) = \frac{n(n-1)}{2}$,总交换次数为

$$\sum_{j=1}^{n-1} (n-j) = \frac{n(n-1)}{2}。$$

由此,冒泡排序是一种稳定的排序方法,其时间复杂度为 $O(n^2)$ 。排序过程中仅需要一个元素的辅助空间用于元素的交换,空间复杂度为 $O(1)$ 。

3. 简单选择排序

n 个记录进行简单选择排序的基本方法是:通过 $n-i$ ($1 \leq i \leq n$) 次关键字之间的比较,从 $n-i+1$ 个记录中选出关键字最小的记录,并和第 i 个记录进行交换,当 i 等于 n 时所有记录有序排列。

【算法】 简单排序算法。

```

void selectsort(int data[],int n)
/*将数组 data 中的 n 个整数按非递减有序的方式进行排列*/
{ int i,j,k,temp;
  for(i = 0;i < n-1;i++){
    k = i;
    for(j = i+1;j < n;j++)      /*找出最小关键字的下标*/
      if (data[j] < data[k]) k = j;
    if (k != i) {
      temp = data[i]; data[i] = data[k]; data[k] = temp;
    }/*if*/
  }

```



```

    }/*for*/
}/*selectsort*/

```

简单选择排序法在最好情况下（待排序列已按关键码有序）不需要移动元素，因此 n 个元素排序时的总移动次数为 0 次。在最坏情况下（元素已经逆序排列）每趟排序移动记录的次数都为 3 次（两个数组元素交换值），共进行 $n-1$ 趟排序，总移动次数为 $3(n-1)$ 。无论在何种情况下，元素的总比较次数为 $\sum_{i=1}^{n-1} (n-i) = \frac{n(n-1)}{2}$ 。

由此，简单选择排序是一种不稳定的排序方法，其时间复杂度为 $O(n^2)$ 。排序过程中仅需要一个元素的辅助空间用于数组元素值的交换，空间复杂度为 $O(1)$ 。

8.6.3 希尔排序

希尔排序又称“缩小增量排序”，是对直接插入排序方法的改进。

希尔排序的基本思想：先将整个待排记录序列分割成若干子序列，然后分别进行直接插入排序，待整个序列中的记录基本有序时，再对全体记录进行一次直接插入排序。具体做法是：先取定一个小于 n 的整数 d_1 作为第一个增量，把文件的全部记录分成 d_1 个组，将所有距离为 d_1 倍数的记录放在同一个组中，在各组内进行直接插入排序；然后取第二个增量 $d_2 < d_1$ ，重复上述分组和排序工作，依次类推，直至所取的增量 $d_i = 1 (d_i < d_{i-1} < \dots < d_2 < d_1)$ ，即所有记录放在同一组进行直接插入排序为止。

增量序列为 5、3、1 时，希尔插入排序过程如下所示。

【初始关键字】：48 37 64 96 75 12 26 48 54 03

	48				12				
		37				26			
			64				48		
				96				54	
					75				03
第一趟排序结果：	12	26	48	54	03	48	37	64	96
	12			54			37		75
		26			03			64	
			48			48			96
第二趟排序结果：	12	03	48	37	26	48	54	64	96
第三趟排序结果：	03	12	26	37	48	48	54	64	75

【函数】用希尔排序方法对整型数组进行非递减排序。

```

void shellsort(int data[],int n)
{ int *delta,k,i,t,dk,j;
  k = n;
  /*从 k=n 开始, 重复 k=k/2 运算, 直到 k 等于 0, 所得 k 值的序列作为增量序列存入 delta*/
  delta = (int *)malloc(sizeof(int)*(n/2));
  i = 0;
  do{
    k = k/2;    delta[i++] = k;
  }while(k > 0);
  i = 0;
  while ((dk = delta[i])>0)  {
    for(k = delta[i];k < n;++k)
      if (data[k] < data[k-dk]) { /*将元素 data[k]插入到有序增量子表中*/
        t = data[k];             /*备份待插入的元素, 空出一个元素位置*/
        for(j = k-dk;j >= 0 && t < data[j];j -= dk)
          data[j+dk] = data[j]; /*寻找插入位置的同时元素后移*/
        data[j+dk] = t;         /*找到插入位置, 插入元素*/
      }/*if*/
    ++i; /*取下一个增量值*/
  }/*while*/
}/* shellsort */

```

8.6.4 快速排序

(1) 快速排序的基本思想: 通过一趟排序将待排的记录分割为独立的两部分, 其中一部分记录的关键字均比另一部分记录的关键字小, 然后再分别对这两部分记录继续进行排序, 以达到整个序列有序。

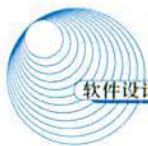
(2) 一趟快速排序的具体做法: 附设两个指针 low 和 high, 它们的初值分别指向文件的第一个记录和最后一个记录。设枢轴记录 (通常是第一个记录) 的关键字为 pivotkey, 则首先从 high 所指位置起向前搜索, 找到第一个关键字小于 pivotkey 的记录并与枢轴记录互相交换, 然后从 low 所指位置起向后搜索, 找到第一个关键字大于 pivotkey 的记录并与枢轴记录互相交换, 重复这两步直至 low 等于 high 时为止。

【函数】 用快速排序方法对整型数组进行非递减排序。

```

void quicksort(int data[],int low,int high)
/*用快速排序方法对数组元素 data[low..high]作非递减排序*/
{ int i,pivotkey,j;

```

```

if (low < high) {
    /*以数组的第一个元素为基准进行划分*/
    pivotkey = data[low]; i = low; j = high;
    while(i < j) {
        /*从数组的两端交替地向中间扫描*/
        while(i < j && data[j] >= pivotkey) j--;
        if (i < j)
            data[i++] = data[j];
        /*比枢轴元素小者移到低下标端*/
        while (i < j && data[i] <= pivotkey) i++;
        if (i < j)
            data[j--] = data[i];
        /*比枢轴元素大者移到高下标端*/
    }
    data[i] = pivotkey;
    /*枢轴元素移到正确的位置*/
    quicksort(data, low, i-1);
    /*对前半个子表递归排序*/
    quicksort(data, i+1, high);
    /*对后半个子表递归排序*/
}
/*if*/
/* quicksort */

```

在所有同数量级($O(n \log n)$)的排序方法中,快速排序被认为是平均性能最好的一种。但是,若初始记录序列按关键字有序或基本有序时,快速排序则蜕化为冒泡排序,此时,算法的时间复杂度为 $O(n^2)$ 。

8.6.5 堆排序

对于 n 个元素的关键字序列 $\{K_1, K_2, \dots, K_n\}$, 当且仅当满足下列关系时称其为堆:

$$\begin{cases} k_i \leq k_{2i} \\ k_i \leq k_{2i+1} \end{cases} \quad \text{或} \quad \begin{cases} k_i \geq k_{2i} \\ k_i \geq k_{2i+1} \end{cases}$$

若将此序列对应的一维数组(即以一维数组作为序列的存储结构)看成是一个完全二叉树,则堆的含义表明,完全二叉树中所有非终端节点的值均不大于(或不小于)其左、右孩子节点的值。因此,在一个堆中,堆顶元素(即完全二叉树的根节点)必为序列中的最小元素(或最大元素),并且堆中任一棵子树也都是堆。若堆顶为最小元素,则称为小顶堆。若堆顶为最大元素,则称为大顶堆。

堆排序的基本思想:对一组待排序记录的关键字,首先按堆的定义排成一个序列(即建立初始堆),从而输出堆顶的最小关键字(对于小顶堆而言),然后将剩余的关键字再调整成新堆,便得到次小的关键字,如此反复,直到全部关键字排成有序序列为止。

初始堆的建立方法是:将待排序的关键字分放到一棵完全二叉树的各个节点中(此时完全二叉树并不一定具备堆的特性),显然,所有 $i > \left\lfloor \frac{n}{2} \right\rfloor$ 的节点 K_i 都没有子节点,以这样的 K_i 为根

的子树已经是堆, 因此初始建堆可从完全二叉树的第 $i \left(i = \left\lfloor \frac{n}{2} \right\rfloor \right)$ 个节点 K_i 开始, 通过调整, 逐步使以 $K_{\lfloor \frac{n}{2} \rfloor}$, $K_{\lfloor \frac{n}{2} \rfloor - 1}$, $K_{\lfloor \frac{n}{2} \rfloor - 2}$, \dots , K_2 , K_1 为根的子树满足堆的定义。在对 K_i 为根的子树建堆的过程中, 可能需要交换 K_i 和 K_{2i} (或 K_{2i+1}) 的值, 如此以来, 以 K_{2i} (或 K_{2i+1}) 为根的子树可能不再满足堆的定义, 则应继续以 K_{2i} (或 K_{2i+1}) 为根进行调整, 如此层层地递推下去, 可能会一直延伸到树叶时为止。这种方法就像过筛子一样, 把最小的关键字一层一层地筛选出来, 最后输出堆顶的最小元素。

【函数】 将一个整型数组中的元素调整成大根堆。

```
void heapadjust(int data[], int s, int m)
/*data[s..m]所构成的一个元素序列中, 除了 data[s]外, 其余元素均满足堆的定义*/
/*调整元素 data[s]的位置, 使 data[s..m]成为一个大根堆*/
{ int t, j;
  t = data[s];                                /*备份元素 data[s], 为其找到适当位置后再插入*/
  for(j = 2*s+1; j <= m; j = j*2+1) {          /*沿值较大的孩子节点向下筛选*/
    if (j < m && data[j] < data[j+1]) ++j;      /*j 是值较大的元素的下标*/
    if (!(t < data[j])) break;
    data[s] = data[j]; s = j;                  /*用 s 记录待插入元素的位置 (下标) */
  } /*for*/
  data[s] = t;                                /*将备份元素插入由 s 所指出的插入位置*/
} /*heapadjust*/
```

调整成新堆: 假设输出堆顶元素之后, 以堆中最后一个元素替代之, 那么根节点的左、右子树均为堆, 此时只需自上至下进行调整即可。

【函数】 用堆排序方法对整型数组进行非递减排序。

```
void heapsort(int data[], int n)                /*数组 data[0..n-1]中的 n 个元素进行堆排序*/
{ register int i;
  int t;
  for(i = n/2-1; i >= 0; --i)                  /*把 data[0..n-1]调整为大根堆*/
    heapadjust(data, i, n-1);
  for(i = n-1; i > 0; --i)
  { t = data[0];
    data[0] = data[i];                          /*堆顶元素 data[0]与序列的最后元素 data[i]交换*/
    data[i] = t;
    /*待排元素的个数减 1, 将 data[0..i-1]重新调整为大根堆*/
    heapadjust(data, 0, i-1);
  }
```



```

    }/*for*/
}/* heapsort */

```

建立初始堆的过程如图 8-58 所示, 调整为新堆的过程如图 8-59 所示。

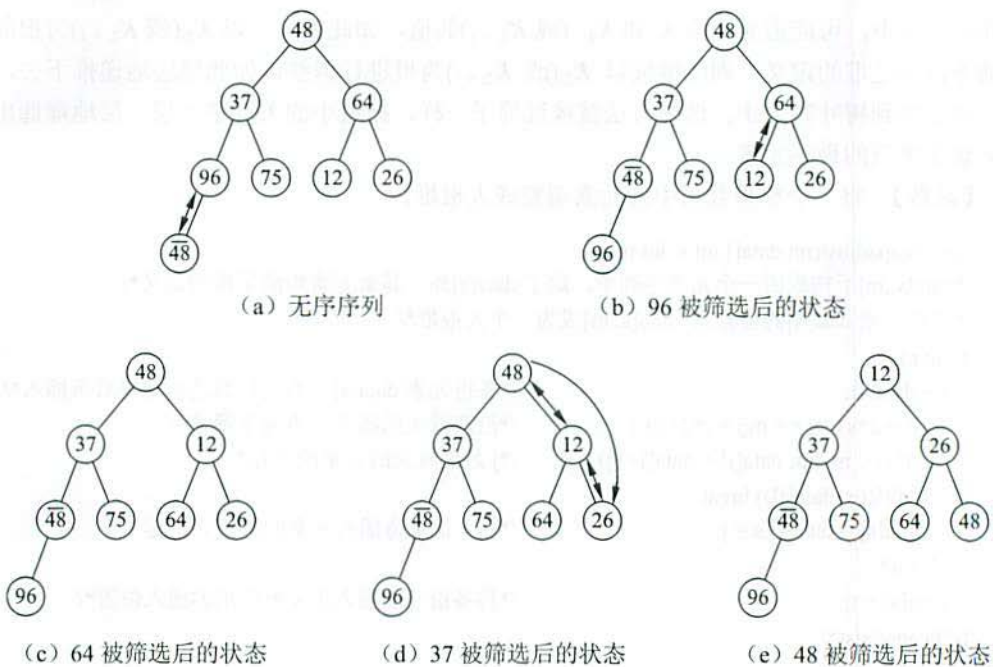


图 8-58 初始堆建立过程示意图

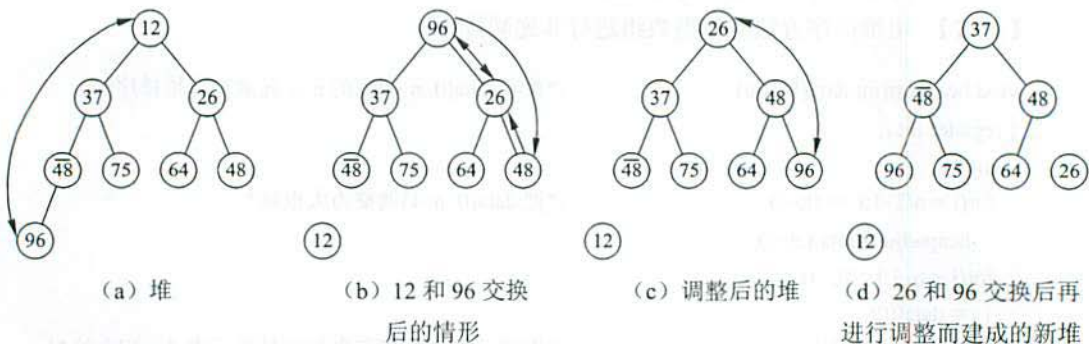


图 8-59 调整为新堆的过程示意图

对于记录数较少的文件来说, 堆排序的优越性并不明显, 但对大量的记录来说堆排序是很



有效的。堆排序的整个算法时间是由建立堆和不断调整堆这两部分时间代价构成的。可以证明,堆排序算法的时间复杂度为 $O(n \log n)$ 。此外,堆排序只需要一个记录大小的辅助空间。堆排序是一种不稳定的排序方法。

8.6.6 归并排序

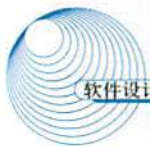
所谓“归并”,是将两个或两个以上的有序文件合并成为一个新的有序文件。从线性表的讨论可知,将两个有序表合并为一个有序表,无论是顺序存储结构还是链式存储结构,都是容易实现的。利用归并的思想可以进行排序。归并排序是把一个有 n 个记录的无序文件看成是由 n 个长度为 1 的有序子文件组成的文件,然后进行两两归并,得到 $\left\lceil \frac{n}{2} \right\rceil$ 个长度为 2 或 1 的有序文件,再两两归并,如此重复,直至最后形成包含 n 个记录的有序文件为止。这种反复将两个有序文件归并成一个有序文件的排序方法称为两路归并排序。

【函数】 两路归并。

```
void Merge(int data1[],int data2[],int s,int m,int n)
/*将分别有序的 data1[s..m]和 data1[m+1..n]归并为有序的 data2[s..n]*/
{ int i,j,k;
  for(i = m+1,k = s; s <= m && i <= n; ++k) /*将 data1 中记录由小到大地并入 data2*/
    if (data1[s] < data1[i]) data2[k] = data1[s++];
    else data2[k] = data1[i++];
  for(j = s; j <= m; ++j, ++k) /*将剩余的 data1[s..m]复制到 data2*/
    data2[k] = data1[j];
  for(j = i; j <= n; ++j, ++k) /*将剩余的 data1[i..n]复制到 data2*/
    data2[k] = data1[j];
} /*Merge*/
```

【函数】 设数组 $data1[]$ 中有 n 个元素,可分割为 $k = \lceil n / len \rceil$ 个段,每段分别有序,分段情况如下: $data1[0..len-1], data1[len..2*len-1], \dots, data1[(l-1)*len..i*len-1], \dots$, 下面的函数中通过调用函数 Merge 将相邻两个分段归并为一个有序段,调用 $k/2$ 次 Merge 后,两两归并的结果就存放在 $data2[]$ 中。

```
void Msort(int data1[],int data2[],int n,int len)
{ int start_p,end_p;
  start_p = 0; /*从下标为 0 的数组元素开始*/
  while (start_p + len < n) { /*每次取相邻的两个有序分段进行合并*/
    end_p = start_p + 2 * len - 1;
    if (end_p >= n) end_p = n - 1; /*若仅剩余一个有序分段,退出*/
```

```

Merge(data1,data2,start_p,start_p+len-1,end_p);
/*调用 merge 将两个有序段合并为一个有序段, 合并结果在 data2 中*/
start_p = end_p+1;
}/*while*/
if(start_p < n)                                /*直接将剩下的一个有序段中的元素复制到 data2 中*/
    for(;start_p < n;start_p++)
        data2[start_p] = data1[start_p];
}/*Msort*/

```

【函数】 对数组 data1[0...n-1]中的 n 个元素进行归并排序。首先, 数组中的每个元素自成一个长度为 1 的有序分段, 共有 n 个分段。然后, 将有序分段两两合并, 使得分段数目逐渐减少, 合并所得分段的长度逐渐加长, 当合并到只有一个分段且长度为 n 时, 元素得到有序排列。

```

void mergesort(int data1[],int n)
{ int length = 1,k = 0;
  int *data2;
  data2 = (int *)malloc(sizeof(int)*n);
  if(data2 == NULL) return;

```

/*第一趟, 有序分段在 data1 中, 两路归并后, 结果存放在 data2 中; 第二趟, 对存放在 data2 中的有序分段进行两路归并, 再将结果存放到 data1 中。归并过程在 data1 和 data2 之间交替进行。length 用于记录每个分段的长度, k 用于控制交替情况*/

```

while(length < n) {
    if(k == 0) Msort(data1,data2,n,length);
    else      Msort(data2,data1,n,length);
    length *= 2; /*每次完成一趟归并, 所得有序分段的长度翻一倍*/
    k = 1 - k;
}/*while*/
if(k == 1)/*若归并趟数为奇数, 排序结果在 data2 中, 因此再将结果复制到 data1 中*/
    for(k = 0;k < n;++k)
        data1[k]=data2[k];
}/*mergesort*/

```

8.6.7 基数排序

基数排序的思想是按组成关键字的各个数位的值进行排序, 它是分配排序的一种。

基数排序中把一个关键字 K_i 看成一个 d 元组, 即

$$K_i^1, K_i^2, \dots, K_i^d$$

其中, $0 \leq K_i^j < r$, $i=1 \sim n$, $j=1 \sim d$ 。这里的 r 称为基数。若关键字是十进制, 则 $r=10$; 若关键字是八进制的, 则 $r=8$ 。 d 是关键字的位数。 d 值取所有待排序的关键字位数的最大值, 其他不足 d 位的关键字则在前面补零。

在 $K_i^1, K_i^2, \dots, K_i^d$ 中, K_i^1 称为最高有效位, K_i^2 称为次高有效位, K_i^d 称为最低有效位。基数排序可以从最高有效位开始, 也可以从最低有效位开始。

基数排序的基本思想: 设立 r 个队列, 队列的编号分别为 $0, 1, 2, \dots, r-1$ 。首先按最低有效位的值, 把 n 个关键字分配到这 r 个队列中; 然后从小到大将各队列中关键字再依次收集起来; 接着再按次低有效位的值把刚收集起来的关键字再分配到 r 个队列中。重复上述收集过程, 直至最高有效位。这样得到了一个从小到大的关键字序列。为了减少记录移动的次數, 队列可以采用链式存储分配, 称为链队列。每个链队列设有两个指针, 分别指向队头和队尾。

分析基数排序算法可知, 对于 n 个记录, 执行一次分配和收集的时间为 $O(n+r)$ 。如果关键字有 d 位, 则要执行 d 遍。所以总的运算时间为 $O(d(n+r))$ 。可见对于不同的基数 r 所用的时间是不一样的。当 r 或 d 较小时, 这种排序方法较为节省时间。另外, 基数排序适用于链式分配的记录的排序, 其要求的附加存储量是 r 个队列的头、尾指针, 所以附加存储量为 $2r$ 个存储单元。由于待排序记录是以链表方式存储的, 相对于顺序分配而言, 还增加了 n 个指针域的空间。基数排序是一种稳定的排序方法。

8.6.8 内部排序方法的比较和选择

综合比较所讨论的各种排序方法, 大致结果如表 8-2 所示。

表 8-2 各种排序方法的性能比较

排序方法	最好时间	平均时间	最坏时间	辅助空间	稳定性
直接插入	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
简单选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
希尔排序	—	$O(n^{1.25})$	—	$O(1)$	不稳定
快速排序	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(n \log n)$	不稳定
堆排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	不稳定
归并排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	稳定
基数排序	$O(d(n+rd))$	$O(d(n+rd))$	$O(d(n+rd))$	$O(rd)$	稳定

迄今为止, 已有的排序方法远远不止上述几种, 人们之所以热衷于研究各种排序方法, 不仅是由于排序在计算机运算中所处的重要位置, 而且还因为不同的方法各有优缺点, 可根据需要运用到不同的场合。选取排序方法时需要考虑的因素有: 待排序的记录个数 n ; 记录本身的



大小;关键字的分布情况;对排序稳定性的要求;语言工具的条件和辅助空间的大小。

依据这些因素,可以得到以下几点结论:

(1) 若待排序的记录数目 n 较小时,可采用插入排序和选择排序。由于直接插入排序所需的记录移动操作较简单选择排序多,因而当记录本身信息量较大时,用简单选择排序方法较好。

(2) 若待排序记录按关键字基本有序,则宜采用直接插入排序或冒泡排序。

(3) 当 n 很大且关键字的位数较少时,采用链式基数排序较好。

(4) 若 n 较大,则应采用时间复杂度为 $O(n\log n)$ 的排序方法,例如快速排序、堆排序或归并排序。快速排序是目前内部排序方法中被认为是最好的方法,当待排序的关键字为随机分布时,快速排序的平均运行时间最短;但堆排序只需 1 个辅助存储空间,并且不会出现在快速排序中可能出现的最坏情况。这两种排序方法都是不稳定的排序方法。若要求排序稳定,可选择归并排序。通常可将归并排序和直接插入排序结合起来使用。先利用直接插入排序求得较长的有序子文件,然后再两两归并。因为直接插入排序是稳定的,所以改进的归并排序是稳定的。

前面讨论的内部排序算法(除基数排序外)都是在一维数组上实现的。当记录本身信息量较大时,为避免耗费大量的时间移动记录,可以采用链表作为存储结构。

8.6.9 外部排序

外部排序就是对大型文件的排序,待排序的记录存放在外存。在排序的过程中,内存只存储文件的一部分记录,整个排序过程需要进行多次的内外存间的数据交换。

常用的外部排序方法是归并排序,这种方法一般分为两个阶段:在第一阶段,把文件中的记录分段读入内存,利用某种内部排序方法对这段记录进行排序并输出到外存的另一个文件中,在新文件中形成许多有序的记录段,称为归并段;在第二阶段,对第一阶段形成的归并段用某种归并方法进行一趟趟地归并,使文件的有序段逐渐加长,直到将整个文件归并为一个有序段时为止。下面简单介绍常用的多路平衡归并方法。

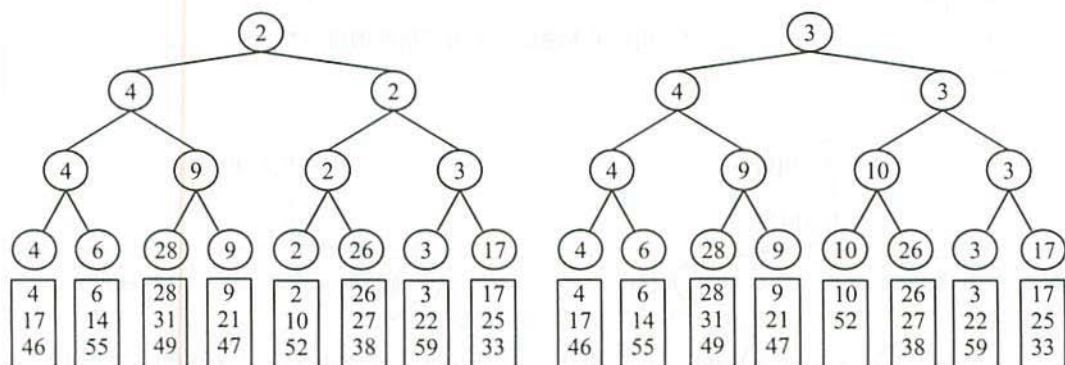
k 路平衡归并是指文件经外部排序的第一个阶段后,已经形成了由若干个初始归并段构成的文件。在这个基础上,反复将每次确定的 k 个归并段归并为一个有序段,将一个文件上的记录归并到另一个文件上。重复这个过程,直到文件中的所有记录都归并为一个有序段。

设已经得到 8 个初始归并段,如图 8-60 所示,其中 b_i 表示第 i 个归并段。

b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7
4	6	28	9	2	26	3	17
17	14	31	21	10	27	22	25
46	55	49	47	52	38	59	33

图 8-60 初始归并段

在树形选择排序中, 首先对 n 个记录的关键字进行两两比较, 然后在 $\left\lceil \frac{n}{2} \right\rceil$ 个较小者之间再进行两两比较, 如此重复, 直到选出最小关键字的记录为止, 该过程一棵有 n 个叶子节点的完全二叉树表示, 如图 8-61 (a) 所示。在树中, 每个非终端节点中的关键字等于其左、右孩子节点中较小的关键字, 则树根节点中的关键字即为所有叶子中的最小关键字。在输出最小关键字之后, 更新最小关键字所在的叶子节点数据, 然后从该叶子节点出发, 和其左(兄弟)节点的关键字进行比较, 修改从叶子节点到根的路径上各节点的关键字, 则根节点的关键字即为次小关键字, 如图 8-61 (b) 所示。重复该过程, 就可完成对所有记录的排序。



(a) 树形选择排序找出最小关键字

(b) 树形选择排序找出次小关键字

图 8-61 树形选择排序

由于在图 8-61 所示的树中, 每个非终端节点记录了其左、右孩子中的“优胜者”, 所以我们称其为“胜者树”。反之, 若在双亲节点中记录比较后的失败者, 而让胜者去参加更上一层的比较, 便可得到一棵“败者树”。这样, 当优胜者到达父节点时, 立刻就知道原先在此比较的失败者并与失败者进行比较, 再次记录新的失败者并让优胜者去进行更上一层的比较。在败者树中, 每个节点只需和其父节点进行比较, 而在胜者树中, 向上调整时节点需与兄弟节点比较, 那么就需得到兄弟节点的位置信息, 因此败者树更易于编程。

为了简便起见, 设每个记录为一个整数, 败者树用数组 $ls[]$ 表示, $ls[i]$ 的值为败者所在归并段的段号, 令 $ls[1]$ 是树根节点, $ls[0]$ 是 $ls[1]$ (根) 的父节点, $ls[0]$ 中存储每次选出的优胜者所在归并段的段号, 输出时则取 $ls[0]$ 指示的归并段的当前记录。例如, 图 8-62 所示的是一棵实现 8 路归并的败者树, 叶子节点的数据来自各个归并段; 败者树的根节点 $ls[1]$ 的父节点 $ls[0]$ 中存储了优胜者 (最小记录) 所在的归并段, 图 8-63 所示的是输出一个记录后, 重新调整后的败者树。

【函数】 败者树的调整。


```

}/*Adjust*/

```

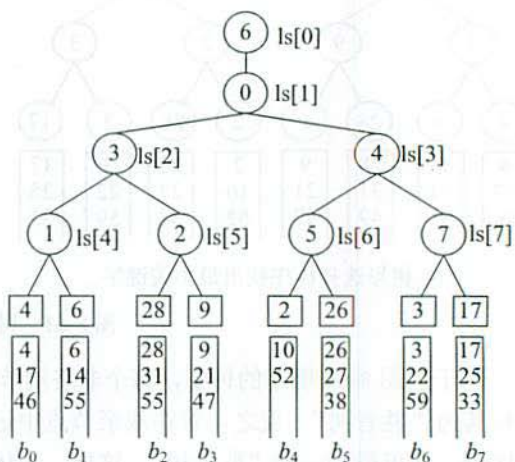


图 8-63 利用败者树找出 8 路归并段的次小关键字

```
#define K 8
#define MINKEY -1 /*比所有关键字都小的一个值*/
#define MAXKEY 10000 /*比所有关键字都大的一个值*/
void K_merge(int ls[K])
/*ls[0]~ls[k-1]是败者树的内部节点。b[0]~b[k-1]分别存储 K 个初始归并段的当前记录*/
/*函数 Get_next(i)用于从第 i 个归并段读取并返回当前记录*/
/*若该归并段已空，则置返回值为 MAXKEY*/
{ int b[K+1],i,q;
```

```

for(i = 0; i < K; i++)
    b[i] = Get_next(i); /*分别读取 K 个归并段的第一个关键字*/
b[K] = MINKEY; /*创建败者树*/
for(i = 0; i < K; ++i) ls[i] = K; /*设置 ls 中败者的初值*/
for(i = K-1; i >= 0; --i) /*依次从 b[K-1], b[K-2], ..., b[0] 出发调整败者*/
    Adjust(ls, i);
while (b[ls[0]] != MAXKEY) {
    q = ls[0]; /*q 是当前最小关键字所在的归并段*/
    printf("%d", b[q]);
    b[q] = Get_next(q);
    Adjust(ls, q); /*q 是调整败者树, 选择新的最小关键字*/
} /*while*/
} /*K_merge*/

```




第9章 常用算法设计方法

要使计算机能完成人们预定的工作,首先必须为如何完成预定的工作设计一个算法,然后再根据算法编写程序。计算机程序要对问题的每个对象和处理规则给出正确详尽的描述,其中程序的数据结构和变量用来描述问题的对象,程序结构、函数和语句用来描述问题的算法。算法和数据结构是程序的两个重要方面。

9.1 算法和算法设计基本概念

9.1.1 算法

算法(Algorithm)是对特定问题求解步骤的一种描述,它是指令的有限序列,其中每一条指令表示一个或多个操作。此外,一个算法还具有下列5个重要特性:

- (1) 有穷性。一个算法必须总是(对任何合法的输入值)在执行有穷步之后结束,且每一步都可在有穷时间内完成。
- (2) 确定性。算法中每一条指令必须有确切的含义,读者理解时不会产生二义性。并且在任何条件下算法只有唯一的一条执行路径,即对于相同的输入只能得出相同的输出。
- (3) 可行性。一个算法是可行的,即算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现的。
- (4) 输入。一个算法有零个或多个输入,这些输入出自于某个特定的对象的集合。
- (5) 输出。一个算法有一个或多个输出。这些输出是同输入有着某些特定关系的量。

9.1.2 算法设计

通常求解一个问题可能会有多种算法可供选择,选择的主要标准首先是算法的正确性和可靠性、简单性和易理解性。其次是算法所需要的存储空间少和执行更快等。

算法设计是一件非常困难的工作,通常设计一个“好”的算法应考虑达到以下目标:

- (1) 正确性(Correctness)。算法应当满足具体问题的需求。
- (2) 可读性(Readability)。算法主要是为了人的阅读与交流,其次才是机器执行。可读性好有助于人对算法的理解;晦涩难懂的程序容易隐藏较多错误而且难以调试和修改。
- (3) 健壮性(Robustness)。当输入数据非法时,算法也能适当地做出反应或进行处理,而

不会产生莫名其妙的输出结果。

(4) 效率与低存储量需求。通俗地说,效率指的是算法执行时间。对于同一个问题如果有多个算法可以解决,执行时间短的算法效率高。存储量需求指算法执行过程中所需要的最大存储空间。效率与低存储量需求这两者都与问题的规模有关。算法效率用时间复杂度来度量,存储量需求用空间复杂度来度量。

经常采用的算法设计技术主要有迭代法、穷举搜索法、递推法、贪心法、回溯法、分治法和动态规划法等。另外,为了以更简洁的形式设计和描述算法,在设计算法时又常常采用递归技术,用递归描述算法。

9.1.3 算法效率的度量

算法执行时间需通过依据该算法编制的程序在计算机上运行时所消耗的时间来度量。而度量一个程序的执行时间通常有两种方法:

(1) 事后统计法。因为很多计算机内部都有计时功能,有的甚至可以精确到 ms 级,不同算法的程序可以通过一组或若干组相同的统计数据以分辨优劣。但这种方法有两个缺陷:一是必须先运行依据算法编制的程序;二是所得时间的统计量依赖于计算机的硬件、软件等环境因素,有时容易掩盖算法本身的优劣。因此人们常常采用另一种事前分析估算的方法。

(2) 事前分析估算方法。一个用高级程序语言编写的程序在计算机上运行时所消耗的时间取决于下列因素:

- ① 依据的算法选用何种策略;
- ② 问题的规模;
- ③ 书写程序的语言,对于同一个算法,实现语言的级别越高,执行效率就越低;
- ④ 编译程序所产生的机器代码的质量;
- ⑤ 机器执行指令的速度。

一个算法是由控制结构(顺序、分支和循环 3 种)和原操作(指基本数据类型的操作)构成的,则算法时间取决于两者的综合效果。为了便于比较同一问题的不同算法,通常的做法是,从算法中选取一种对于所研究的问题(或算法类型)来说是基本操作的原操作,以该基本操作重复执行的次数作为算法的时间度量。

一般情况下,算法中基本操作重复执行的次数是问题规模 n 的某个函数 $f(n)$,算法的时间度量记作

$$T(n)=O(f(n))$$

它表示随着问题规模 n 的增大,算法执行时间的增长率和 $f(4n)$ 的增长率相同,称作算法的渐进时间复杂度(Asymptotic Time Complexity),简称时间复杂度。



9.1.4 算法的存储空间需求

类似于算法的时间复杂度, 算法所需的存储空间用空间复杂度 (Space Complexity) 来度量, 记作

$$S(n)=O(f(n))$$

其中 n 为问题的规模。一个上机执行的程序除了需要存储空间来寄存本身所用指令、常数、变量和输入数据外, 也需要一些对数据进行操作的工作单元和存储一些为实现计算所需信息的辅助空间。若输入数据所占空间只取决于问题本身, 和算法无关, 则只需要分析除输入和程序之外的额外空间, 否则应同时考虑输入本身所需空间 (和输入数据的表示形式有关)。

9.2 迭代法、穷举搜索法、递推法

9.2.1 迭代法

迭代法是用于解决数值计算问题中的非线性方程 (组) 求解或最优解 (近似根) 的一种算法设计方法。它的主要思想是从某个点出发, 通过某种方式求出下一个点, 使得其离要求的点 (方程的解) 更近一步; 当两者之差接近到可接受的精度范围时, 就认为找到了问题的解。由于它是不断进行这样的过程, 因此称为迭代法, 同时从中也可以看出使用迭代法必须保证其收敛性。

在使用迭代法的过程中, 应该注意两种异常情况: 如果方程无解, 那么近似根序列将不会收敛, 迭代过程会成为“死循环”。因此在使用时应先判断其是否有解, 并应对迭代的次数进行限制; 方程虽然有解, 但迭代公式选择不当, 或迭代的初始近似根选择不合理, 也会导致迭代失败。

9.2.2 穷举搜索法

穷举搜索法是对可能是解的众多候选解按某种顺序进行逐一枚举和检验, 并从中找出那些符合要求的候选解作为问题的解。

这种方法通常需要用多重循环 (循环次数取决于变量个数) 来实现, 对每个变量的每个值都测试是否满足所给定的条件, 如果是则找到了问题的一个解。

这种方法简单易行, 尤其是对于一时想不出更好的解法的问题, 不失为一种补救的好方法。但是, 这种方法只能解决变量个数非常有限, 而且每个变量可取值个数也很有限的情况, 否则会造成“指数爆炸”, 在计算机上难以实现。

9.2.3 递推法

设要求问题规模为 N 的解, 当 $N=1$ 时, 解或为已知, 或能非常方便地得到解。能采用递推法构造算法的问题有重要的递推性质: 即当得到问题规模为 $i-1$ 的解后, 由问题的递推性质, 能从已求得的规模为 $1, 2, \dots, i-1$ 的一系列解, 构造出问题规模为 i 的解。这样, 程序可从 $i=0$ 或 $i=1$ 出发, 重复地, 由已知至 $i-1$ 规模的解, 通过递推, 获得规模为 i 的解, 直至得到规模为 N 的解。

【例 9.1】 编写程序, 对给定的 $n(n \leq 100)$, 计算并输出 k 的阶乘 $k!$ ($k=1, 2, \dots, n$) 的全部有效数字。

由于要求的整数可能大大超出一般整数的位数, 程序用一维数组存储长整数, 存储长整数数组的每个元素只存储长整数的一位数字。如有 m 位长整数 N 用数组 $a[]$ 存储:

$$N = a[m] \times 10^{m-1} + a[m-1] \times 10^{m-2} + \dots + a[2] \times 10^1 + a[1] \times 10^0$$

并用 $a[0]$ 存储长整数 N 的位数 m , 即 $a[0]=m$ 。按上述约定, 数组的每个元素存储 k 的阶乘 $k!$ 的一位数字, 并从低位到高位依次存于数组的第二个元素、第三个元素……例如, $5! = 120$, 在数组中的存储形式为:

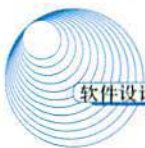
3	0	2	1	...
---	---	---	---	-----

首元素 3 表示长整数是一个 3 位数, 接着是低位到高位依次是 0、2、1, 表示长整数是 120。计算阶乘 $k!$ 可采用对已求得的阶乘 $(k-1)!$ 连续累加 $k-1$ 次后求出。例如, 已知 $4! = 24$, 计算 $5!$, 可对原来的 24 再累加 4 次 24 后得到 120。细节见以下程序。

【例 9.1 程序】

```

#include <stdio.h>
#include <malloc.h>
#define MAXN 1000
void pnext(int a[], int k) /*已知 a 中的(k-1)!, 求出 k!*/
{
    int *b, m = a[0], i, j, r, carry;
    b = (int *)malloc(sizeof(int) * (m+1));
    for(i = 1; i <= m; i++) b[i] = a[i];
    for(j = 1; j < k; j++) { /*控制累加 k-1 次*/
        for(carry = 0, i = 1; i <= m; i++) {
            r = (i <= a[0] ? a[i] + b[i] : a[i]) + carry;
            a[i] = r % 10; carry = r / 10;
        }
    }
}
  
```

```
        if(carry)    a[++m] = carry;
    }
    free(b);
    a[0]=m;
}
void write(int *a, int k)
{   int i;
    printf("%4d!=", k);
    for(i = a[0]; i > 0; i--)    printf("%d",a[i]);
    printf("\n\n");
}
void main()
{   int a[MAXN], n, k;
    printf("Enter the number n:"); scanf("%d",&n);
    a[0] = 1; a[1] = 1; write(a,1);
    for(k = 2; k <= n; k++) {
        pnext(a,k); write(a,k);  getchar();
    }
}
```

递推法是一种简单有效的方法,一般用这种方法编写的程序的执行效率很高,可以解决有前效相关性、但前效相关顺序确定而且个数不多且是定数的问题。

递推法与递归法的关系是:任何可以用递推法解决的问题,都可以很方便地利用递归法解决。反之,许多用递归法解决的问题,却不能用递推法解决。这是因为递归法利用递归时的栈,可以有任意长度和顺序的前效相关性,这是递推法所不具备的。

9.3 递归法

递归是设计和描述算法的一种有力的工具,由于它在复杂算法的描述中被经常采用,为此在进一步介绍其他算法设计方法之前先讨论它。

能采用递归描述的算法通常有这样的特征:为求解规模为 N 的问题,设法将它分解成一些规模较小的问题,然后从这些小问题的解方便地构造出大问题的解,并且这些规模较小的问题也能采用同样的分解和综合方法,分解成规模更小的问题,并从这些更小问题的解构造出规模稍大问题的解。特别地,当规模 $N=1$ 时,能直接得到解。

【例 9.2】 阶乘函数。

阶乘函数可递归地定义为：

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n > 0 \end{cases}$$

阶乘函数的自变量 n 的定义域是非负整数。递归式的第一式给出了这个函数的一个初始值，是非递归定义的。每个递归函数都必须有非递归定义的初始值，否则，递归函数就无法计算。递归式的第二式是用较小自变量的函数值来表示较大自变量的函数值的方式来定义 n 的阶乘。 $n!$ 可以递归地计算如下：

```
int Factorial(int n)
{ if(n == 0) return 1;
  if(n > 0) return n * Factorial(n-1);
}
```

【例 9.3】 斐波那契 (Fibonacci) 数列。

无穷数列 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ……称为 Fibonacci 数列。它可以递归地定义为：

$$F(n) = \begin{cases} 1 & n = 0 \\ 1 & n = 1 \\ F(n-1) + F(n-2) & n > 1 \end{cases}$$

这是一个递归关系式，它说明当 n 大于 1 时，这个数列的第 n 项的值是它前面两项之和。它用两个较小的自变量的函数值来定义一个较大自变量的函数值，所以需要两个初始值 $F(0)$ 和 $F(1)$ 。

第 n 个 Fibonacci 数可递归地计算如下：

```
int Fibonacci(int n)
{ if(n == 0) return 0;
  if(n == 1) return 1;
  if(n > 1) return fib(n-2) + fib(n-1);
}
```

递归算法的执行过程分递推和回归两个阶段。在递推阶段，把较复杂的问题（规模为 n ）的求解推到比原问题简单一些的问题（规模小于 n ）的求解。例如上例中，求解 $Fibonacci(n)$ ，把它推到求解 $Fibonacci(n-2)$ 和 $Fibonacci(n-1)$ 。也就是说，为计算 $Fibonacci(n)$ ，推到计算 $Fibonacci(n-2)$ 和 $Fibonacci(n-1)$ ，而计算 $Fibonacci(n-2)$ 和 $Fibonacci(n-1)$ ，又把它们推到计算 $Fibonacci(n-4)$ 和 $Fibonacci(n-3)$ 。依次类推，直至计算 $Fibonacci(0)$ 和 $Fibonacci(1)$ ，分别能立即得到结果 0 和 1。在递推阶段，必须要有终止递归的情况，例如在函数 $Fibonacci(n)$ 中，当 n 为 0 和 1 的情况。在回归阶段，当获得最简单情况的解后，逐级返回，依次获得稍复杂问题的解。



例如得到 Fibonacci(0)为 0 和 Fibonacci(1)为 1 后,返回获得 Fibonacci(2)的结果……得到了 Fibonacci($n-2$)和 Fibonacci($n-1$)的结果后,返回获得 Fibonacci(n)的结果。

在编写递归函数时要注意,函数中的局部变量和参数只是局限于当前调用层的,当递推进入“简单问题”层时,原来层次上的参数和局部变量便被隐蔽起来。在一系列“简单问题”层,它们各有自己的参数和局部变量。

【例 9.4】整数划分问题。

将一个正整数 n 表示成一系列正整数之和,

$$n = n_1 + n_2 + \cdots + n_k, \text{ 其中, } n_1 \geq n_2 \geq \cdots \geq n_k \geq 1, k \geq 1。$$

正整数 n 的一个这种表示称为正整数 n 的一个划分。正整数 n 的不同的划分个数成为正整数 n 的划分数,记作 $p(n)$ 。

例如正整数 6 有如下 11 种不同的划分,所以 $p(6)=11$ 。

6;

5+1;

4+2, 4+1+1;

3+3, 3+2+1, 3+1+1+1;

2+2+2, 2+2+1+1, 2+1+1+1+1;

1+1+1+1+1+1。

在正整数 n 的所有不同的划分中,将最大加数 n_1 不大于 m 的划分个数记作 $q(n, m)$ 。可以建立如下递归关系:

(1) $q(n, 1)=1, n \geq 1$ 。当最大加数 n_1 不大于 1 时,任何正整数 n 只有一种划分形式,即 $n = \overbrace{1+1+\cdots+1}^n$ 。

(2) $q(n, m)=q(n, n), m \geq n$ 。最大加数 n_1 实际上不能大于 n , 因此 $q(n, m)=1$ 。

(3) $q(n, n)=1+q(n, n-1)$ 。正整数 n 的划分由 $n_1=n$ 的划分和 $n_1 \leq n-1$ 的划分组成。

(4) $q(n, m)=q(n, m-1)+q(n-m, m), n > m > 1$ 。正整数 n 的最大加数 n_1 不大于 m 的划分由 $n_1=m$ 的划分和 $n_1 \leq m-1$ 的划分组成。

以上的关系实际上给出了计算 $q(n, m)$ 的递归式:

$$q(n, m) = \begin{cases} 1 & n=1, m=1 \\ q(n, n) & n < m \\ 1+q(n, n-1) & n=m \\ q(n, m-1)+q(n-m, m) & n > m > 1 \end{cases}$$

据此,可设计计算 $q(n, m)$ 的递归函数如下。正整数 n 的划分数 $p(n)=q(n, n)$ 。

```

int q(int n, int m)
{
    if((n < 1) || (m < 1)) return 0;
    if((n == 1) || (m == 1)) return 1;
    if(n < m) return q(n,n);
    if(n==m) return q(n,m-1)+1;
    return q(n,m-1) + q(n-m, m);
}
  
```

【例 9.5】 背包问题。

有不同价值、不同重量的物品 n 件, 求从这 n 件物品中选取一部分物品的选择方案, 使选中物品的总重量不超过指定的限制重量, 但选中物品的价值之和为最大。

设 n 件物品的重量分别为 w_0, w_1, \dots, w_{n-1} , 物品的价值分别为 v_0, v_1, \dots, v_{n-1} 。采用递归寻找物品的选择方案。设前面已有了多种选择的方案, 并保留了其中总价值最大的方案与数组 `option[]`, 该方案的总价值存于变量 `maxv`。当前正在考察新方案, 其物品选择情况保存于数组 `cop[]`。假定当前方案已考虑了前 $i-1$ 件物品, 现在要考虑第 i 号物品; 当前方案已包含的物品的重量之和为 tw , 至此, 若其余物品都选择是可能的话, 本方案能达到的总价值的期望值设为 tv 。算法引入 tv 是当一旦当前方案的总价值的期望值也小于前面方案的总价值 `maxv` 时, 继续考察当前方案变成无意义的工作, 应终止当前方案, 立即去考察下一个方案。因为当方案的总价值不比 `maxv` 大时, 该方案不会被再考察, 这同时保证函数后找到的方案一定会比前面的方案更好。

对于第 i 件物品的选择考虑有两种可能:

- (1) 考虑物品 i 被选择, 这种可能性仅当包含它不会超过方案总重量限制时才是可行的。选中后, 继续递归去考虑其余物品的选择。
- (2) 考虑物品 i 不被选择, 这种可能性仅当不包含物品 i 也有可能找到价值更大的方案的情况。

按以上思想写出递归算法如下:

【算法】 背包问题, 找最佳方案。

```

try(物品 i, 当前选择已达到的重量和 tw, 本方案可能达到的总价值 tv)
{
    /*考虑物品 i 包含在当前方案中的可能性*/
    if(包含物品 i 是可接受的) {
        将物品 i 包含在当前方案中;
        if(i < n-1) try (i+1, tw+物品 i 的重量, tv);
        else /*又一个完整的方案, 因它比前面的方案好, 以它作为最佳方案*/
            以当前方案作为临时最佳方案保存;
    }
}
  
```




恢复物品 i 不包含状态:

```
}
/*考虑物品  $i$  不包含在当前方案中的可能性*/
if(不包含物品  $i$  仅是可考虑的)
    if( $i < n-1$ ) try( $i+1$ , tw, tv-物品  $i$  的价值);
else /*又一个完整方案, 因它比前面的方案好, 以它作为最佳方案*/
    以当前方案作为临时最佳方案保存;
}
```

由于递归算法结构清晰, 可读性强, 且容易用数学归纳法证明算法的正确性, 因此它为设计算法、调试程序带来很大方便。然而递归算法的运行效率较低, 无论是耗费的计算时间还是占用的存储空间都比非递归算法要多。若在程序中消除算法的递归调用, 则其运行时间可大为节省。因此, 有时希望在一个递归算法中消除递归调用, 使其转换为一个非递归算法。通常, 消除递归是采用一个用户定义的栈来模拟系统的递归调用工作栈, 从而达到将递归算法改为非递归算法的目的。仅仅是机械地模拟还不能达到减少计算时间和存储空间的目的, 还需要根据具体程序的特点对递归调用工作栈进行简化, 尽量减少栈操作, 压缩栈存储空间已达到节省计算时间和存储空间的目的。

9.4 分治法

9.4.1 分治法的基本思想

任何一个可以用计算机求解的问题所需的计算时间都与其规模有关。问题的规模越小, 解题所需的计算时间往往也越少, 从而也较容易处理。例如, 对于 n 个元素的排序问题, 当 $n=1$ 时, 不需任何计算; 当 $n=2$ 时, 只要做一次比较即可; 当 $n=3$ 时, 只要做两次比较即可……而当 n 较大时, 问题就不那么容易处理了。要想直接解决一个较大的问题, 有时是相当困难的。分治法的设计思想是, 将一个难以直接解决的大问题, 分解成一些规模较小的相同问题, 以便各个击破, 分而治之。如果规模为 n 的问题可分解成 k 个子问题, $1 < k \leq n$, 且这些子问题互相独立且与原问题相同。递归地求解这些问题, 然后将各子问题的解合并得到原问题的解。它的一般算法设计模式如下:

```
Divide-and-Conquer(P)
{
    if( $|P| \leq n_0$ ) Adhoc(P);
    divide P into smaller subinstances
     $P_1, P_2, \dots, P_k$ ;
```

```

for(i = 1; i <= k; i++)
    yi = Divide-and-Conquer(Pi);
return Merge(y1,...,yk);
}
  
```

其中, $|P|$ 表示问题 P 的规模, n_0 为阈值, 表示当问题 P 的规模不超过 n_0 时, 问题已容易解出, 不必再继续分解。Adhoc(P) 是该分治法中的基本子算法, 用于直接解小规模的问题 P 。因此, 当 P 的规模不超过 n_0 时, 直接用算法 Adhoc(P) 求解。算法 Merge(y_1, y_2, \dots, y_k) 是该分治法中的合并子算法, 用于将 P 的子问题 P_1, \dots, P_k 的解 y_1, \dots, y_k 合并为 P 的解。

9.4.2 分治法的典型实例

以上讨论的是分治法的基本思想和一般原则, 下面用一些具体例子来说明如何针对具体问题用分治思想来设计有效算法。

【例 9.6】 最大子段和问题。给定由 n 个整数 (可能为负整数) 组成的序列 a_1, a_2, \dots, a_n , 求该序列形如 $\sum_{k=i}^j a_k$ 的子段和的最大值。当所有整数均为负整数时定义其最大子段和为 0。依此定义, 所求的最优值为

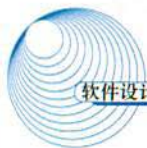
$$\max \left\{ 0, \max_{1 \leq i \leq j \leq n} \sum_{k=i}^j a_k \right\}$$

例如, 当 $(a_1, a_2, a_3, a_4, a_5, a_6) = (-2, 11, -4, 13, -5, -2)$ 时, 最大子段和为 $\sum_{k=2}^4 a_k = 20$ 。

如果将所给的序列 $a[1:n]$ 分为长度相等的两段 $a[1:n/2]$ 和 $a[n/2+1:n]$, 分别求出这两段的最大子段和, 则 $a[1:n]$ 的最大子段和有 3 种情形:

- (1) $a[1:n]$ 的最大子段和与 $a[1:n/2]$ 的最大子段和相同;
- (2) $a[1:n]$ 的最大子段和与 $a[n/2+1:n]$ 的最大子段和相同;
- (3) $a[1:n]$ 的最大子段和为 $\sum_{k=i}^j a_k$, 且 $1 \leq i \leq n/2, n/2+1 \leq j \leq n$ 。

(1) 和 (2) 这两种情形可递归求得。对于情形 (3), 容易看出, $a[n/2]$ 与 $a[n/2+1]$ 在最优子序列中。因此我们可以在 $a[1:n/2]$ 中计算出 $s1 = \max_{1 \leq i \leq n/2} \sum_{k=i}^{n/2} a[k]$, 并在 $a[n/2+1:n]$ 中计算出 $s2 = \max_{n/2+1 \leq i \leq n} \sum_{k=n/2+1}^i a[k]$ 。则 $s1+s2$ 即为出现情形 (3) 的最优值。据此可设计出求解最大子段和的分治算法如下:



```
int MaxSubSum(int a, int left, int right)
{
    int sum = 0;
    if(left == right)    sum = a[left] > 0 ? a[left] : 0;
    else {
        int center = (left + right)/2;
        int leftsum = MaxSubSum(a, left, center);
        int rightsum = MaxSubSum(a, center+1, right);
        int s1 = 0, lefts = 0;
        for(int i = center; i >= left; i--) {
            lefts += a[i];
            if(lefts > s1) s1 = lefts;
        }
        int s2 = 0, rights = 0;
        for(int i = center+1; i <= right; i++) {
            rights += a[i];
            if(rights > s2) s2 = rights;
        }
        sum = s1 + s2;
        if(sum < leftsum)    sum = leftsum;
        if(sum < rightsum)    sum = rightsum;
    }
    return sum;
}

int MaxSum(int n, int a)
{
    return MaxSubSum(a, 1, n);
}
```

【例 9.7】二分搜索技术。

二分搜索算法是运用分治策略的典型例子。

给定已排好序的 n 个元素 $a[0:n-1]$ ，现在要在这 n 个元素中找出一定元素 x 。

首先容易想到的是用顺序搜索方法，逐个比较 $a[0:n-1]$ 中的元素，直至找到元素 x 或搜索遍整个数组后确定 x 不在其中。这个方法没有很好的利用 n 个元素已排序这个条件，因此在最坏情况下，顺序搜索方法需要 $O(n)$ 次比较。

二分搜索算法充分利用了元素间的次序关系，采用分治策略，可在最坏情况下用 $O(\log n)$ 时间完成搜索任务。

二分搜索算法的基本思想是将 n 个元素分成个数大致相同的两部分, 取 $a[n/2]$ 与 x 作比较。如果 $x=a[n/2]$, 则找到 x , 算法终止。如果 $x<a[n/2]$, 则在数组 a 的左半部分继续搜索, 如果 $x>a[n/2]$, 则在数组 a 的右半部分继续搜索。

【例 9.8】合并排序。

合并排序算法是用分治策略实现的对 n 个元素进行排序的算法。其基本思想是: 当 $n=1$ 时终止排序; 否则将待排序元素分成大小大致相同的两个子集合, 分别对两个子集合进行排序, 最终将排好序的子集合合并成为所要求的排好序的集合, 合并排序算法的算法框架可递归地描述如下:

```
void MergeSort(int a[], int left, int right)
{
    if(left < right) {           // 至少有 2 个元素
        int i = (left + right)/2; // 取中点
        MergeSort(a, left, i);
        MergeSort(a, i+1, right);
        Merge(a, b, left, i, right); // 合并到数组 b
        Copy(a, b, left, right);    // 复制回数组 a
    }
}
```

其中, 算法 Merge 合并两个排好序的数组段到一个新的数组 b , 然后又 Copy 函数将合并后的数组段再复制回数组 a 中。Merge 和 Copy 显然可在 $O(n)$ 时间内完成, 因此合并排序算法对 n 个元素进行排序, 在最坏情况下所需的计算时间 $T(n)$ 满足

$$T(n) = \begin{cases} O(1) & n \leq 1 \\ 2T(n/2) + O(n) & n > 1 \end{cases}$$

解此递归方程可知 $T(n) = O(n \log n)$ 。由于排序问题的计算时间下界为 $\Omega(n \log n)$, 故合并排序算法是一个渐进最优算法。

【例 9.9】为参加网球比赛的选手安排比赛日程。

没有 $n(=2^k)$ 位选手参加网球循环赛, 循环赛共进行 $n-1$ 天, 每位选手要与其他 $n-1$ 位选手赛一场, 且每位选手每天赛一场, 不轮空。试按此要求为比赛安排日程。

设 n 位选手被顺序编号为 $1, 2, \dots, n$ 。比赛的日程表是一个 n 行 $n-1$ 列的表, 第 i 行 j 列的内容是第 i 号选手第 j 天的比赛对手。用分治法设计日程表, 就是从其中一半选手 (2^{m-1} 位) 的比赛日程, 导出全体 (2^m 位) 选手的比赛日程。从众所周知的只有两位选手的比赛日程出发, 反复这个过程, 直至为 n 位选手安排好比赛日程为止。

以下程序。

(c) 8 位选手比赛日程表

图 9-1 超干比宾柱及排小总图

```

    for(j = 1; j <= twoml-1; j++)
        a[i][j] = a[i-twoml][j] + twoml;
/* 填日程表的右上角 */
a[1][twoml] = twoml + 1;          /* 填日程表右上角的第1列 */
for(i = 2; i <= twoml; i++)    a[i][twoml] = a[i-1][twoml] + 1;
/* 填日程表右上角的其他列, 参照前一列填当前列 */
for(j = twoml + 1; j < twom; j++) {
    for(i = 1; i < twoml; i++)    a[i][j] = a[i+1][j-1];
    a[twoml][j] = a[1][j-1];
}
/* 填日程表的右下角 */
for(j = twoml; j < twom; j++)
    for(i = 1; i <= twoml; i++)
        a[a[i][j]][j] = i;
for(i = 1; i <= twom; i++) {
    for(j = 1; j < twom; j++)    printf("%4d", a[i][j]);
    printf("\n");
}
printf("\n");
}
}

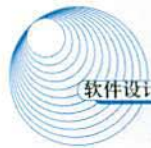
```

9.5 动态规划法

9.5.1 动态规划法的基本思想

动态规划算法与分治法类似, 其基本思想也是讲待求解问题分解成若干个子问题, 先求解子问题, 然后从这些子问题的解得到原问题的解。与分治法不同的是, 适合于用动态规划法求解的问题, 经分解得到的子问题往往不是独立的。若用分治法来解这类问题, 则分解得到的子问题数目太多, 以至于最后解决原问题需要耗费指数级时间。然而不同子问题的数目常常只有多项式量级。在用分治法求解时, 有些子问题被重复了许多次。如果我们能够保存已解决的子问题的答案, 而在需要时再找出已求得的答案, 这样就可以避免大量的重复计算, 从而得到多项式时间的算法。为了达到这个目的, 我们可以用一个表来记录所有已解决的子问题的答案。不管该子问题以后是否被用到, 只要它被计算过, 就将其结果填入表中。这就是动态规划法的基本思路。

动态规划算法通常用于求解具有某种最优性质的问题。在这类问题中, 可能会有许多可行



解, 每个解都对应于一个值, 我们希望找到具有最优值(最大值或最小值)的那个解。设计一个动态规划算法, 通常可按以下几个步骤进行:

- (1) 找出最优解的性质, 并刻画其结构特征;
- (2) 递归地定义最优值;
- (3) 以自底向上的方式计算出最优值;
- (4) 根据计算最优值时得到的信息, 构造一个最优解。

步骤(1)~(3)是动态规划算法的基本步骤。在只需要求出最优值的情形下步骤(4)可以省略。若需要求出问题的一个最优解, 则必须执行步骤(4)。此时, 在步骤(3)中计算最优值时, 通常需记录更多的信息, 以便在步骤(4)中根据所记录的信息, 快速构造出一个最优解。

9.5.2 动态规划法的典型实例

【例 9.10】 最大子段和问题。

最大子段和问题的描述见“例 9.6”。

分析“例 9.6”中给出的最大子段和的分治算法, 可以发现若记 $b[j] = \max_{1 \leq i \leq j} \left\{ \sum_{k=i}^j a[k] \right\}$,

$1 \leq j \leq n$, 则所求的最大子段和为

$$\max_{1 \leq i \leq j \leq n} \sum_{k=i}^j a[k] = \max_{1 \leq j \leq n} \max_{1 \leq i \leq j} \sum_{k=i}^j a[k] = \max_{1 \leq j \leq n} b[j]$$

由 $b[j]$ 的定义易知, 当 $b[j-1] > 0$ 时, $b[j] = b[j-1] + a[j]$, 否则 $b[j] = a[j]$ 。由此可得计算 $b[j]$ 的动态规划递归式

$$b[j] = \max \{b[j-1] + a[j], a[j]\}, \quad 1 \leq j \leq n$$

据此, 可设计出计算最大子段和的动态规划算法如下:

```
int MaxSum(int n, int a)
```

```
{
    int sum = 0, b = 0;
    for(int i = 1; i <= n; i++) {
        if(b > 0) b += a[i];
        else b = a[i];
        if(b > sum) sum = b;
    }
    return sum;
}
```

上述算法显然需要 $O(n)$ 计算时间和 $O(n)$ 空间。

【例 9.11】 求两字符序列的最长公共字符序列。

这里所说的字符序列的子序列可能与读者理解的不同。这里的字符子序列是指从给定字符序列中随意地（不一定是连续的）去掉若干字符（可能一个也不去掉）后所形成的字符列。令给定地字符序列 $X = "x_0x_1 \cdots x_{m-1}"$ ，序列 $Y = "y_0y_1 \cdots y_{k-1}"$ 是 X 的子序列，存在 X 的一个严格递增下标序列 $\langle i_0, i_1, \cdots, i_{k-1} \rangle$ ，使得对所有的 $j = 0, 1, \cdots, k-1$ 有 $x_{i_j} = y_j$ 。例如， $X = "ABCBDAB"$ ， $Y = "BCDB"$ 是 X 的一个子序列。

给定两个序列 A 和 B ，称序列 Z 是 A 和 B 的公共子序列，是指 Z 同时是 A 和 B 的子序列。问题要求已知两序列 A 和 B 的最长公共子序列。

如果采用列举 A 的所有子序列，并一一检查其是否又是 B 的子序列，并随时记录所发现的公共子序列，最终求出最长公共子序列。这种方法因为或是太多而不可取。

考虑最长公共子序列问题如何分解称子问题。设 $A = "a_0a_1 \cdots a_{m-1}"$ ， $B = "b_0b_1 \cdots b_{n-1}"$ ，并设 $Z = "z_0z_1 \cdots z_{k-1}"$ 为它们的最长公共子序列。不难证明有以下性质：

(1) 如果 $a_{m-1} = b_{n-1}$ ，则 $z_{k-1} = a_{m-1} = b_{n-1}$ ，且 $"z_0z_1 \cdots z_{k-2}"$ 是 $"a_0a_1 \cdots a_{m-2}"$ 和 $"b_0b_1 \cdots b_{n-1}"$ 的一个最长公共子序列；

(2) 如果 $a_{m-1} \neq b_{n-1}$ ，则若 $z_{k-1} \neq a_{m-1}$ ，蕴含 $"z_0z_1 \cdots z_{k-1}"$ 是 $"a_0a_1 \cdots a_{m-2}"$ 和 $"b_0b_1 \cdots b_{n-1}"$ 的一个最长公共子序列；

(3) 如果 $a_{m-1} \neq b_{n-1}$ ，则若 $z_{k-1} \neq b_{n-1}$ ，蕴含 $"z_0z_1 \cdots z_{k-1}"$ 是 $"a_0a_1 \cdots a_{m-1}"$ 和 $"b_0b_1 \cdots b_{n-2}"$ 的一个最长公共子序列。

这样在找 A 和 B 的公共子序列时，如果有 $a_{m-1} = b_{n-1}$ ，则进一步解决一个子问题，找 $"a_0a_1 \cdots a_{m-2}"$ 和 $"b_0b_1 \cdots b_{n-1}"$ 的一个最长公共子序列；如果 $a_{m-1} \neq b_{n-1}$ ，则要解决两个子问题，找出 $"a_0a_1 \cdots a_{m-2}"$ 和 $"b_0b_1 \cdots b_{n-1}"$ 的一个最长公共子序列，以及找出 $"a_0a_1 \cdots a_{m-1}"$ 和 $"b_0b_1 \cdots b_{n-2}"$ 的一个最长公共子序列，再取两者中较长者作为 A 和 B 的最长公共子序列。

定义 $c[i][j]$ 为序列 $"a_0a_1 \cdots a_{i-1}"$ 和 $"b_0b_1 \cdots b_{j-1}"$ 的最长公共子序列的长度，计算 $c[i][j]$ 可以递归地表述如下：

$c[i][j] = 0$ ，若 $i = 0$ ，或 $j = 0$ ；

$c[i][j] = c[i-1][j-1] + 1$ ，若 $i, j > 0$ ，且 $a[i-1] = b[j-1]$

$c[i][j] = \max(c[i][j-1], c[i-1][j])$ ，若 $i, j > 0$ ，且 $a[i-1] \neq b[j-1]$

按此计算式可以写出计算两个序列最长公共子序列的长度函数，将例 9.12 程序的函数 `lcs_len`。

由 $c[i][j]$ 的产生仅依赖于 $c[i-1][j-1]$ 、 $c[i-1][j]$ 和 $c[i][j-1]$ 可知，利用函数 `lcs_len` 所产生的数组 $c[i][j]$ ，可以从 $c[m][n]$ 开始，跟踪 $c[i][j]$ 的产生过程，逆向构造出最长公共子序列。

【例 9.11 程序】

```

#include <stdio.h>
#include <string.h>
#define N 100
char a[N], b[N], str[N];
/* 计算两个序列的最长公共子序列的长度 */
int lcs_len(char *a, char *b, int c[][N])
{
    int m = strlen(a), n = strlen(b), i, j;
    for(i = 0; i <= m; i++) c[i][0] = 0;
    for(j = 1; j <= n; j++) c[0][j] = 0;
    for(i = 1; i <= m; i++)
        for(j = 1; j <= n; j++)
            if(a[i-1] == b[j-1]) c[i][j] = c[i-1][j-1] + 1;
            else if(c[i-1][j] >= c[i][j-1]) c[i][j] = c[i-1][j];
            else c[i][j] = c[i][j-1];
    return c[m][n];
}
/* 构造最长公共子序列函数 */
char *build_lcs(char s[], char *a, char *b)
{
    int k, i = strlen(a), j = strlen(b), c[N][N];
    k = lcs_len(a, b, c);
    s[k] = '\0';
    while(k > 0)
        if(c[i][j] == c[i-1][j]) i--;
        else if(c[i][j] == c[i][j-1]) j--;
        else {
            s[--k] = a[i-1]; i--; j--;
        }
    return s;
}
void main()
{
    printf("Enter two string(<%d)! \n", N);
    scanf("%s %s", a, b);
    printf("LCS = %s\n", build_lcs(str, a, b));
}

```

【例 9.12】 凸多边形的最优三角剖分。

一个由 n 个顶点构成的凸多边形 P 可用按顺时针方向列出其所有顶点来表示, 如图 9-2 的凸多边形可以表示成 $P = \langle v_0, v_1, v_2, v_3, v_4, v_5, v_6 \rangle$ 。给定两个不相邻的顶点 v_p 和 v_q , 线段 (v_p, v_q) 为多边形的弦。多边形的最优三角剖分是指已经凸多边形 $P = \langle v_0, v_1, \dots, v_{n-1} \rangle$, 要求给出一种三角剖分方案, 使得剖分所用的弦长之和最小。

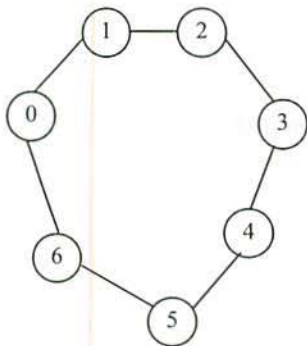


图 9-2 一个由 7 个顶点构成的凸多边形

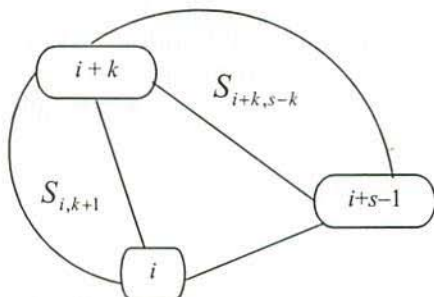


图 9-3 三角剖分示意图

考虑一个 n 个顶点的凸多边形的一种最优三角剖分, 例如该最优剖分包含三角形 $\Delta v_0 v_k v_{n-1}$, $0 < k < n-1$ 。该三角剖分的最优值由三角形 $\Delta v_0 v_k v_{n-1}$ 的弦长值和与对两个多边形 $\langle v_0, v_1, \dots, v_k \rangle$ 和 $\langle v_k, v_{k+1}, \dots, v_{n-1} \rangle$ 的三角剖分中的最优值之和。不难看出, 最优三角剖分应是对所有可能的 k 取其最小值。

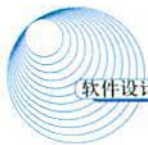
现定义 $S_{i,s}$ 表示自顶点 i 开始, 并由 s 个顶点组成的多边形 $\langle v_i, v_{i+1}, \dots, v_{i+s-1} \rangle$ 。为求解 $S_{i,s}$, 必须考虑以下 3 种可能形式的选择, 见图 9-3。

- (1) 选顶点 v_{i+s-2} , 构成 $\Delta v_i v_{i+s-2} v_{i+s-1}$ 和子问题 $S_{i,s-1}$;
- (2) 选顶点 v_{i+1} , 构成 $\Delta v_i v_{i+1} v_{i+s-1}$ 和子问题 $S_{i+1,s-1}$;
- (3) 对 2 和 $s-3$ 之间的某个 k , 选顶点 v_{i+k} , 构成 $\Delta v_i v_{i+k} v_{i+s-1}$ 和两个子问题 $S_{i,k+1}$ 和 $S_{i+k,s-k}$ 。

在这里, 若约定当子问题的大小 $s < 3$ 时, 值为 0; 而当 $s=3$ 时, 直接由其弦长之和确定, 可把上述 3 种可能合而为一。即, 对 1 和 $s-2$ 之间的某个 k , 解两个子问题 $S_{i,k+1}$ 和 $S_{i+k,s-k}$ 。

对于三角剖分问题, 如果采用递归法求解, 在递归过程中会反复地求同样的子问题, 解题所费时间会随问题规模底大小呈幂次增加。为此采用动态规划方法, 记子问题 $S_{i,s}$ 的最优三角剖分值为 $c[i][s]$, 对于 $s \geq 4$, $c[i][s]$ 的计算公式为:

$$c[i][s] = \min(c[i][k+1] + c[i+k][s-k] + d(v_i, v_{i+k}) + d(v_{i+k}, v_{i+s-1})), \quad 1 \leq k \leq s-2$$



这里的 $d(v_p, v_q)$ 是顶点 v_p 和 v_q 之间的弦长, 如果它们是相邻顶点, 则其值约定为 0。由于给定问题的解依赖于更小问题的解, 所以填表顺序是按子问题规模从小到大的顺序填写。

根据以上分析, 得到求凸多边形的最优三角剖分的算法如下:

【例 9.12 算法】

```
{ 输入多边形的顶点个数 n;
  按逆时针顺序输入各顶点的坐标;
  计算各弦的长度, 填于表 d[i][j];      /* 其中相邻顶点的长度为 0 */
  for(i = 0; i < n; i++) c[i][2] = 0.0;
  for(i = 0; i < n; i++) {              /* 计算 c[i][3], 并形成  $s_{i,3}$  的弦链表 */
    计算 c[i][3];
    形成  $s_{i,3}$  的弦链表:
    }
    for(s = 4; s <= n; s++)
  for(i = 0; i < n; i++) {              /* 计算 c[i][s] */
    for(t = MAX, kk = 1; kk <= s-2; kk++) {
      temp = c[i][(kk+1) % n] + c[(i+kk) % n][(n+s-kk) % n] +
              d[i][(i+kk) % n] + d[(i+kk) % n][(i+s-1) % n];
      if(temp < t) {                    /* 调整 t */
        k = kk;      t = temp;
      }
    }
    c[i][s] = temp;
    将弦  $\langle v_i, v_{i+k} \rangle$  及  $S_{i,k+1}$  和  $S_{i+k,s-k}$  的剖分弦链接在一起;
    if(s == n) break;                  /* 计算结束, 最优值在 c[0][m] */
  }
  输出结果;
}
```

9.6 回溯法

回溯法有“通用的解题法”之称, 用它可以系统地搜索一个问题的所有解或任一解。回溯法是一个既带有系统性又带有跳跃性的搜索算法。它在包含问题的所有解的解空间树中, 按照深度优先的策略, 从根节点出发搜索解空间树。算法搜索至解空间树的任一节点时, 总是先判断该节点是否肯定不包含问题的解。如果肯定不包含, 则跳过对以该节点为根的子树的系统搜索, 逐层向其祖先节点回溯。否则, 进入该子树, 继续按深度优先的策略进行搜索。回溯法在

用来求问题的所有解时,要回溯到根,且根节点的所有子树都已被搜索遍才结束。而回溯法在用来求问题的任一解时,只要搜索到问题的一个解就可结束。这种以深度优先的方式系统地搜索问题的解的算法称为回溯法,它适用于解一些组合数较大的问题。

9.6.1 回溯法的算法框架

1. 问题的解空间

应用回溯法解问题时,首先应明确定义问题的解空间。问题的解空间应至少包含问题的一个(最优)解。例如,对于有 n 种可选择物品的 0-1 背包问题,其解空间由长度为 n 的 0-1 向量组成。该解空间包含了对变量的所有可能的 0-1 赋值。当 $n=3$ 时,其解空间是

$$\{(0,0,0), (0,1,0), (0,0,1), (1,0,0), (0,1,1), (1,0,1), (1,1,0), (1,1,1)\}$$

定义了问题的解空间后,还应将解空间很好地组织起来,使得用回溯法能方便地搜索整个解空间。通常我们将解空间组成树或图的形式。例如,对于 $n=3$ 时的 0-1 背包问题,其解空间用一棵完全二叉树表示,如图 9-4 所示。

解空间树的第 i 层到第 $i+1$ 层边上的标号给出了变量的值。从树根到叶的任一路径表示解空间的一个元素。例如,从根节点到节点 H 的路径对应于解空间中的元素 $(1,1,1)$ 。

2. 回溯法的基本思想

确定了解空间的组织结构后,回溯法就从开始节点(根节点)出发,以深度优先的方式搜索整个解空间。这个开始节点就成为一个活节点,同时也成为当前的扩展节点。在当前的扩展节点处,搜索向纵深方向移至一个新节点。这个新节点就成为一个新的活节点,并成为当前扩展节点。如果在当前扩展节点处不能再向纵深方向移动,则当前的扩展节点就成为死节点。换句话说,这个节点不再是一个活节点。此时,应往回移动(回溯)至最近的一个活节点处,并使这个活节点成为当前的扩展节点。回溯法即以这种工作方式递归地在解空间中搜索,直至找到所要求的解或解空间中已无活节点时为止。

例如,对于 $n=3$ 时的 0-1 背包问题,考虑下面的具体实例: $w=[16,15,15]$, $p=[45,25,25]$, $c=30$ 。其中 w 表示物品的重量, p 表示物品的价值, c 表示背包能够容纳的最大重量。我们从图 9-4 的根节点开始搜索其解空间。

(1) 开始时根节点是唯一的活节点,也是当前的扩展节点。在这个扩展节点处,按照深度优先策略移至节点 B 或节点 C 。假设选择先移至节点 B 。此时节点 A 和节点 B 是活节点,节点 B 成为当前的扩展节点。由于选取了 w_1 (节点 A 到 B 的边上标记为 1,表示选择物品),故在节点 B 处剩余背包容量是 $r=14$,获取的价值是 45。

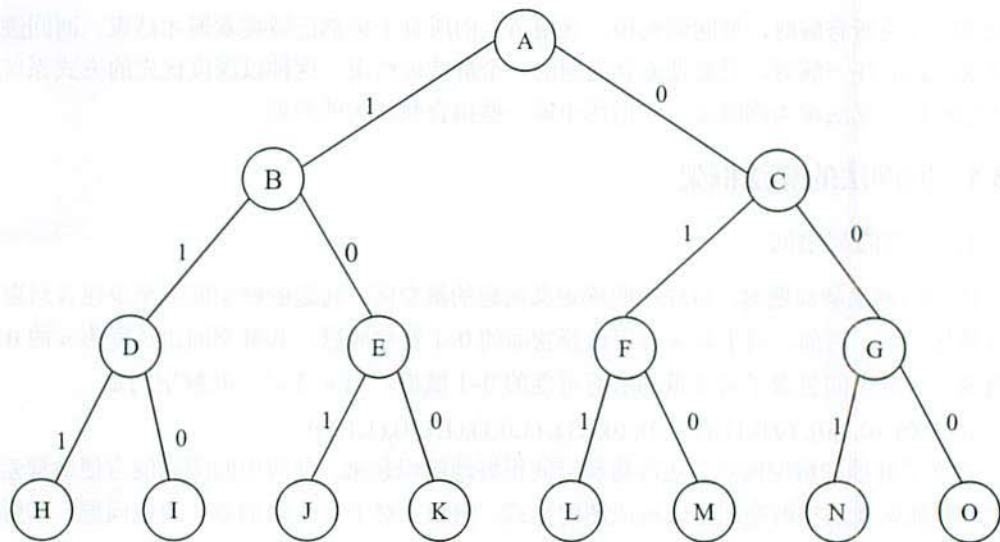
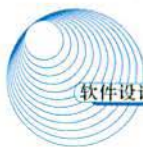


图 9-4 0-1 背包问题的解空间树

(2) 从节点 B 处, 我们可以移至节点 D 或 E。由于移至节点 D 至少需要 $w_2=15$ 的背包容量, 而现在的背包容量是 $r=14$, 故移至节点 D 导致一个不可行的解。而搜索至节点 E 不需要占用背包容量 (节点 B 到节点 E 的边上标记为 0, 表示不需要选择物品), 因而是可行的。从而选择移至节点 E。此时, E 成为新的扩展节点, 节点 A、B 和 E 是活节点。在节点 E 处, $r=14$, 获取的价值为 45。

(3) 从节点 E 处可以移至节点 J 或 K。移至节点 J 导致一个不可行解, 而移至节点 K 是可行的, 于是移至节点 K, 它成为一个新的扩展节点。由于节点 K 是一个叶节点, 故得到一个可行解, 这个解对应的价值为 45。解 x 的取值是由根节点到叶节点 K 的路径所唯一确定, 即 $x=(1,0,0)$ 。由于在节点 K 除已不能再向纵深扩展, 所以节点 K 成为死节点。返回到节点 E, 此时在 E 处也没有可扩展的节点, 它也成为死节点。

(4) 返回节点 B 处, 节点 B 同样也成为死节点, 从而节点 A 再次成为当前扩展节点。节点 A 还可以继续扩展, 从而达到节点 C。此时 $r=30$, 获取的价值为 0。

(5) 从节点 C 可移至节点 F 或 G。假设移至节点 F, 它成为新的扩展节点。节点 A、C 和 F 是活节点。在节点 F 处, $r=15$, 获取的价值为 25。从节点 F 移至节点 L 处, 此时 $r=0$, 获取的价值为 50。由于 L 是一个叶节点, 而且是迄今为止找到的获取价值最高的可行解, 因此记录这个可行解。节点 L 不可扩展, 返回到节点 F 处。

按此方式继续搜索, 可搜索遍整个解空间。搜索结束后找到的最好解就是 0-1 背包问题的最优解。

综上所述，运用回溯法解题通常包含以下 3 个步骤：

- (1) 针对所给问题，定义问题的解空间；
- (2) 确定易于搜索的解空间结构；
- (3) 以深度优先的方式搜索解空间。

9.6.2 回溯法的典型实例

【例 9.13】 在 $9(3 \times 3)$ 个方格的方阵中要填入数字 1 到 $N(N \geq 10)$ 内的某 9 个数字，每个方格填一个整数，使所有相邻两个方格内的两个整数之和为质数。试求出所有满足这个要求的各种数字填法。

可用试探法找到问题的解，即从第一个方格开始，为当前方格寻找一个合理的整数填入，并在当前位置正确填入后，为下一方格寻找可填入的合理整数。如不能为当前方格找到一个合理的可填整数，就要回退到前一方格，调整前一方格的填入数。当为第 9 格也填入合理的整数后，就找到了一个解，将该解输出，并调整第 9 格的填数去找下一个解。

为找到一个满足要求的 9 个数的填法，从还未填一个数开始，按某种顺序（如从小到大的顺序）每次在当前位置填入一个整数，然后检查当前填入的整数是否能满足要求。在满足要求的情况下，继续用同样的方法为下一方格填入整数。如果最近填入的整数不能满足要求，就改变填入的整数。如对当前方格试尽所有可能整数，都不能满足要求，就得回退到前一方格，并调整前一方格填入的整数。如此重复执行扩展、检查或调整、检查，直到找到一个满足问题要求的解，将解输出。

【例 9.13 算法 1】 回溯法找一个解的算法。

```

{ int m = 0, ok = 1; /*空状态是一个合理的情况*/
int n = 8;
do{ /*重复执行以下操作*/
    if(ok) 扩展;
    else    调整;
    ok = 检查前 m 个整数填方的合理性;
} while(!ok || m!=n) && (m != 0);
if(m != 0) 输出解; /*找到了解*/
else 输出无解报告;
}
  
```

如果程序要找全部解，则在将找到的解输出后，应继续调整最后位置上填放的数，试图去找下一个解。相应的算法如下：

【例 9.13 算法 2】 回溯法找全部解的算法。



```
{ int m = 0, ok = 1; /*空状态是一个合理的情况*/
int n = 8;
do{                               /*重复执行以下操作*/
    if (ok)
        if(m == n) {
            输出解:
        调整:
        }
    else 扩展:
    else 调整:
    ok = 检查前 m 个整数填放的合理性;
} while(m! = 0);
}
```

为了确保程序能够终止,调整时,必须保证曾被放弃过的填数序列不会被再次试验,即要求按某种有序模型生成填数序列。给解的候选者设定一个被检验的顺序,按这个顺序逐一形成候选者并检验。从小到大或从大到小都是可以采用的办法。如扩展时,先在新位置填入整数 1,调整时,找当前候选解中下一个还未被使用过的整数。将上述扩展、调整、检验都编写成函数,见下面找全部解的程序。

【例 9.13 程序】

```
#include <stdio.h>
#define N 12
void write(int a[])
{ int i, j;
  for(i = 0; i < 3; i++) {
    for(j = 0; j < 3; j++) printf("%3d", a[3*i + j]);
    printf("\n");
  }
  scanf("%*c");
}
int b[N+1];
int a[10];
int isPrime(int m)
{ int i;
  int primes[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, -1 };
  if(m == 1 || m % 2 == 0) return 0;
```

```

for(i = 0; primes[i] > 0; i++)
    if(m == primes[i])    return 1;
for(i = 3; i*i <= m; ) {
    if(m % i == 0)    return 0;
    i += 2;
}
return 1;
}

int checkMatrix[][3] = { {-1}, {0, -1}, {1, -1},
                        {0, -1}, {1, 3, -1}, {2, 4, -1},
                        {3, -1}, {4, 6, -1}, {5, 7, -1} };

int selectNum(int start)
{ int j;
  for(j = start; j <= N; j++)
    if(b[j])    return j;
  return 0;
}

int check(int pos)
{ int i, j;
  if(pos < 0)    return 0;
  for(i = 0; (j = checkMatrix[pos][i]) >= 0; i++)
    if(!isPrime(a[pos] + a[i]))    return 0;
  return 1;
}

int extend(int pos)
{ a[++pos] = selectNum(1);  b[a[pos]] = 0;
  return pos;
}

int change(int pos)
{ int j;
  while(pos >= 0 && (j = selectNum(a[pos] + 1)) == 0)
    b[a[pos--]] = 1;
  if(pos < 0)    return -1;
  b[a[pos]] = 1;
  a[pos] = j;
  b[j] = 0;
  return pos;
}

```



```

void find()
{ int ok = 1, pos = 0;
  a[pos] = 1;  b[a[pos]] = 0;
  do
  { if(ok)
    { if(pos == 8) {
      write(a);
      pos = change(pos);
    }
    else pos = extend(pos);
    else pos = change(pos);
    ok = check(pos);
  } while(pos >= 0);
}
void main()
{ int i;
  for(i = 1; i <= N; i++) b[i] = 1;
  find();
}

```

【例 9.14】 n 皇后问题。

这是来源于国际象棋的一个问题。 n 后问题要求在一个 $n \times n$ 格的棋盘上放置 n 个皇后, 使得它们彼此不受攻击。按照国际象棋的规则, 一个皇后可以攻击与之处在同一行或同一列或同一条斜线上的其他任何棋子。因此, n 后问题等价于要求在一个 $n \times n$ 格的棋盘上放置 n 个皇后, 使得任何 2 个皇后不能被放在同一行或同一列或同一条斜线上。

求解过程从空配置开始, 在第 1 列至第 m 列为合理配置的基础上, 再配置第 $m+1$ 列。直至第 n 列配置也是合理时, 就找到了一个解。接着改变第 n 列配置, 希望获得下一个解。另外, 在任一列上可能有 n 种配置。开始时, 配置在第 1 行, 以后改变时顺次选择第 2 行, 第 3 行, ..., 直到第 n 行。当第 n 行配置也找不到一个合理的配置时, 就要回溯, 去改变前一列的配置。得到求解皇后问题算法如下:

【算法】

```

{ 输入棋盘大小值 n;                      /* 一个 n*n 的棋盘 */
  m = 0;                                  /* 从空配置开始 */
  good = 1;                               /* 空配置中皇后不相互攻击 */
  do                                      /* 找循环解 */
  { if(good)

```

```

if(m == n) {
    输出解;
    改变之, 形成下一个候选解;
}
else 扩展当前候选解至下一列;
    else 改变之, 形成下一个候选解;
    } while(m != 0);
}
    
```

在编写程序之前, 先确定表示棋盘的数据结构。比较直观的方法是采用一个二维数组, 但仔细考察就会发现, 这种表示方法给调整候选解及检查其合理性带来困难。更好的方法是尽可能直接表示那些常用信息。对于本题来说, “常用信息”并不是皇后的具体位置, 而是“一个皇后是否已经在某行和某条斜线合理地安置好了”。因在每一列上恰好放一个皇后, 引入一个一维数组(设数组名为 col), 值 col[j]表示在棋盘第 j 列、col[j]行有一个皇后。如 col[3]的值为 4, 就表示在棋盘的 3 列、第 4 行有一个皇后。另外, 为了使程序在找完了全部解后回溯到最初位置, 设定 col[0]的初值为 0。当回溯到第 0 列时, 说明程序已求得全部解(或无解), 结束程序执行。

为使程序在检查皇后配置的合理性方面简易方便, 引入以下 3 个工作数组:

- (1) 数组 a[], a[k]表示第 k 行上还没有皇后;
- (2) 数组 b[], b[k]表示第 k 条右高左低斜线上没有皇后;
- (3) 数组 c[], c[k]表示第 k 条左高右低斜线上没有皇后。

棋盘中同一右高左低斜线上的方格, 它们的行号与列号之和均相同; 同一左高右低斜线上的方格, 它们的行号与列号之差均相同。

初始时, 所有行和斜线上均没有皇后, 从第 1 列的第 1 行配置第 1 个皇后开始; 在第 m 列 col[m]行放置了一个合理的皇后后, 准备考察第 m+1 列时, 在数组 a[], b[], c[]中为第 m 列, col[m]行的位置设定有皇后标志; 当从第 m 列回溯到第 m-1 列, 并准备调整第 m-1 列的皇后配置时, 清除在数组 a[], b[], c[]中设置的关于第 m-1 列, col[m-1]行有皇后的标志。一个皇后在 m 列, col[m]行方格内配置是合理的, 由数组 a[], b[], c[]对应位置的值都为 1 来确定。细节见以下程序。

【例 9.14 程序 1】

```

#include <stdio.h>
#include <stdlib.h>
#define MAXN 20
int n, m, good;
    
```




```
int col[MAXN+1], a[MAXN+1];
int b[2*MAXN+1], c[2*MAXN+1];
void main()
{ int j;
  char awn;
  printf("Enter n. "); scanf("%d", &n);
  for(j = 0; j <= n; j++) a[j] = 1;
  for(j = 0; j <= 2*n; j++) b[j] = c[j] = 1;
  m = 1; col[1] = 1; good = 1; col[0] = 0;
  do /* 寻找解 */
  { if(good)
    { if(m == n) { /* 找到一个解 */
      printf("列\t行");
      for(j = 1; j <= n; j++)
        printf("%3d\t%d\n", j, col[j]);
      printf("Enter a character(Q/q for exit)! \N");
      scanf("%c", &awn);
      if(awn == 'Q' || awn == 'q') exit(0);
      while(col[m] == n) {
        m--; /* 清除关于第 m-1 列, col[m-1]行有皇后标志 */
        a[col[m]] = b[m+col[m]] = c[n+m-col[m]] = 1;
      }
      col[m]++; /* 调整第 m 列的皇后配置*/
    }
    else { /* 在第 m 列, col[m]行位置设定有皇后标志 */
      a[col[m]] = b[m+col[m]] = c[n+m-col[m]] = 0;
      col[++m] = 1; /* 第 m+1 列, 从第 1 行开始配置 */
    }
    else { /* 回溯调整 */
      while(col[m] == n) { /* 回溯 */
        m--; /* 清楚关于第 m-1 列, col[m-1]行有皇后标志 */
        a[col[m]] = b[m+col[m]] = c[n+m-col[m]] = 1;
      }
      col[m]++; /* 调整第 m 列的皇后配置 */
    }
    good = a[col[m]] && b[m+col[m]] && c[n+m-col[m]];
  } while(m != 0)
}
```

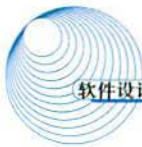
用回溯法找解的算法也常常被编写成递归函数。下面两程序中的函数 `queen_all` 和函数 `queen_one` 能分别用来解皇后问题的全部解和一个解。

【例 9.14 程序 2】 递归方法找皇后问题全部解。

```
#include <stdio.h>
#include <stdlib.h>
#define MAXN 20
int n;
int col[MAXN+1], a[MAXN+1];
int b[2*MAXN+1], c[2*MAXN+1];
void queen_all(int k, int n) /* 在 n×n 棋盘的第 k 列上找合理的配置*/
{ int i, j;
  char awn;
  for(i = 1; i <= n; i++) /* 第 k 列共有 n 种配置, 分别在 1 至 n 行上 */
    if(a[i] && b[k+1] && c[n+k-i]) { /* 测试合理性 */
      col[k] = i;
      a[i] = b[k+i] = c[n+k-i] = 0; /* 置对应位置有皇后 */
      if(k == n) { /* 找到一个解 */
        printf("列\t行");
        for(j = 1; j <= n; j++)
          printf("%3d\t%d\n", j, col[j]);
        printf("Enter a character(Q/q for exit)! \n");
        scanf(" %c", &awn);
        if(awn == 'Q' || awn == 'q') exit(0);
      }
      else queen_all(k+1, n); /* 递归至第 k+1 列 */
      a[i] = b[k+i] = c[n+k-i] = 1; /* 恢复对应位置无皇后 */
    }
}

void main()
{ int j;
  printf("Enter n.\n"); scanf("%d", &n);
  for(j = 0; j <= n; j++) a[j] = 1;
  for(j = 0; j <= 2*n; j++) b[j] = c[j] = 1;
  queen_all(1, n);
}
```

采用递归方法找一个解与找全部解稍有不同。在找一个解的算法中, 递归算法必须确定当



前候选解最终是否能真正成为解。当它能成为最终解时,递归函数就不再递归试探,立即返回;若不能成为解,就得继续试探。设函数 `queen_one` 返回 1 表示找到解,返回 0 表示当前候选解不能成为解。细节见以下函数。

【例 9.14 函数】 递归方法找皇后的一个解。

```
#define MAXN 20
int n;
int col[MAXN+1], a[MAXN+1];
int b[2*MAXN+1], c[2*MAXN+1];
void queen_one(int k, int n)          /* 在 n×n 棋盘的第 k 列上找合理的配置*/
{ int i, found;
  i = found = 0;
  while(!found && i < n) {            /* 未找到解,且还未试尽所有可能循环 */
    i++;
    if(a[i] && b[k+i] && c[n+k-i]) {  /* 测试合理性 */
      col[k] = i;
      a[i] = b[k+i] = c[n+k-i] = 0;
      if(k == n) return 1;            /* 找到解 */
      else found = queen_one(k+1, n); /* 递归至 k+1 列,返回找到与否信息 */
      a[i] = b[k+i] = c[n+k-i] = 1;
    }
  }
  return found;
}
```

9.7 贪心法

贪心法是一种不追求最优解,只希望得到较为满意解的方法。贪心法一般可以快速得到满意的解,因为它省去了为找最优解要穷尽所有可能而必须耗费的大量时间。贪心法常以当前情况为基础作最优选择,而不考虑各种可能的整体情况,所以贪心法不要回溯。

例如平时购物找钱时,为使找回的零钱的硬币数最少,不考虑找零钱的所有各种方案,而是从最大面值的币种开始,按递减的顺序考虑各币种,先尽量用大面值的币种,当不足大面值币种的金额时才去考虑下一种较小面值的币种。这就是在采用贪心法。这种方法在这里总是最优,是因为银行对其发行的硬币种类和硬币面值的巧妙安排。如果只有面值分别为 1, 5 和 11 单位的硬币,而希望找回总额为 15 单位的硬币。按贪心算法,应找 1 个 11 单位面值的硬币和 4 个 1 单位面值的硬币,共找回 5 个硬币。但最优的解答应是 3 个 5 单位面值的硬币。

【例 9.15】活动安排问题。

活动安排问题是可以利用贪心算法有效求解的一个很好的例子。该问题要求高效地安排一系列争用某一公共资源的活动。贪心算法提供了一个简单、有效的方法使得尽可能多的活动能兼容地使用公共资源。

设有 n 个活动的集合 $E = \{1, 2, \dots, n\}$ ，其中每个活动都要求使用同一资源，如演讲会场等，而在同一时间内只有一个活动能使用这一资源。每个活动 i 都有一个要求使用该资源的起始时间 s_i 和一个结束时间 f_i ，且 $s_i < f_i$ 。如果选择了活动 i ，则它在时间区间 $[s_i, f_i]$ 内占用资源。若区间 $[s_i, f_i]$ 与区间 $[s_j, f_j]$ 不相交，则称活动 i 和活动 j 是相容的。也就是说，当 $s_i \geq f_j$ 或 $s_j \geq f_i$ 时，活动 i 和活动 j 相容。活动安排问题就是要在所给的活动集合中选出最大的相容活动子集。

在下面所给出的该活动安排问题的贪心算法 GreedySelector 中，各活动的起始时间和结束时间存储于数组 s 和 f 中且按结束时间的非降序 $f_1 \leq f_2 \leq \dots \leq f_n$ 排列。如果所给出的活动未按此序排列，我们可以用 $O(n \log n)$ 的时间将它重排。

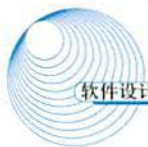
```

void GreedySelector(int n, int s[], int f[], bool A[])
{
    A[1] = true;
    int j = 1;
    for(int i = 2; i <= n; i++) {
        if(s[i] >= f[j]) {
            A[i] = true;
            j = i;
        }
        else A[i] = false;
    }
}
  
```

算法 GreedySelector 中用集合 A 来存储所选择的活动。活动 i 在集合 A 中，当且仅当 $A[i]$ 的值为 true。变量 j 用以记录最近一次加入到 A 中的活动。由于输入的活动是按其结束时间的非降序排列的， f_j 总是当前集合 A 中所有活动的最大结束时间，即：

$$f_j = \max_{k \in A} \{f_k\}$$

贪心算法 GreedySelector 一开始选择活动 i ，并将 j 初始化为 i 。然后依次检查活动 i 是否与当前已选择的所有活动相容。若相容则将活动 i 加入到已选择活动的集合 A 中，否则不选择活动 i ，而继续检查下一活动与集合 A 中活动的相容性。由于 f_i 总是当前集合 A 中所有活动的



最大结束时间, 故活动 i 与当前集合 A 中所有活动相容的充分必要条件是 its 开始时间 s_i 不早于最近加入集合 A 中的活动 j 的结束时间 f_j , 即 $s_i \geq f_j$ 。若活动 i 与之相容, 则 i 成为最近加入集合 A 中的活动, 因而取代活动 j 的位置。由于输入的活动是以其完成时间的非降序排列的, 所以算法 GreedySelector 每次总是选择具有最早完成时间的相容活动加入集合 A 中。按这种方法选择相容活动就为未安排活动留下尽可能多的时间。也就是说, 该算法的贪心选择的意义是使剩余的可安排时间段极大化, 以便安排尽可能多的相容活动。

【例 9.16】 用贪心法解装箱问题。

装箱问题可简述如下: 设有编号为 $0, 1, \dots, n-1$ 的 n 种物品, 体积分别为 v_0, v_1, \dots, v_{n-1} 。将这 n 种物品装到容量都为 V 的若干箱子里。约定这 n 种物品的体积均不超过 V , 即对于 $0 \leq i < n$, 有 $0 < v_i \leq V$ 。不同的装箱方案所需要的箱子数目可能不同, 装箱问题要求装完这 n 种物品使用的箱子数要最少。

若考察将 n 件物品的集合划分成 n 个或小于 n 个物品的所有子集, 最优解就可以找到, 但所有可能的划分的总数太大。对适当大的 n , 找出所有可能的划分要花费的时间是无法承受的。为此, 对装箱问题采用非常简单的近似算法, 即贪心法。该算法依次将物品放到它第一个能放进去的箱子中, 该算法虽不能保证找到最优解, 但还是能找到非常好的解。设 n 件物品的体积是按从大到小的顺序排列的, 即有 $v_0 \geq v_1 \geq \dots \geq v_{n-1}$ 。如不满足上述要求, 只要先对这 n 件物品按它们的体积从大到小排序, 然后按排序结果对物品重新编号即可。装箱算法简单描述如下:

【算法】 一个简单的装箱算法。

```
{ 输入箱子的容积;
  输入物品种类 n;
  按体积从大到小排列, 输入各物品的体积;
  预置已用箱子链为空;
  预置已用箱子计数器 box_count 为 0;
  for(i = 0; i < n; i++) { /* 物品 i 按以下步骤装箱 */
    从已用的第一只箱子开始顺序寻找能放入物品 i 的箱子 j;
    if(已用箱子都不能再放物品 i) {
      另用一只箱子, 并将物品 i 放入该箱子;
      box_count++;
    }
    else 将物品 i 放入箱子 j;
  }
}
```

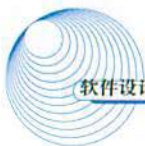
上述算法能求出需要的箱子数 `box_count`，并能求出各箱子所装物品。下面的例子说明该算法不一定能找到最优解。设有 6 种物品，它们的体积分别为 60, 45, 35, 20, 20, 20 单位体积，箱子的容量为 100 单位体积。按上述算法计算，需 3 只箱子。各箱子所装物品分别为：第一只箱子装物品 1, 3；第二只箱子装物品 2, 4；第三只箱子装物品 6。而最优解为两只箱子，如分别装物品 1, 4, 5 和 2, 3, 6 等。

若每只箱子所装物品用链表来表示，链表首节点指针存于一个结构中，结构记录尚剩余的空间量和该箱子所装物品链表的首指针。另外将全部箱子的信息也构成链。以下是按以上算法编写的程序。

【例 9.16 程序】 用贪心法解装箱问题。

```

#include <stdio.h>
#include <stdlib.h>
typedef struct ele {
    int vno;
    struct ele *link;
}ELE;
typedef struct hnode {
    int remainder;
    ELE *head;
    struct hnode * next;
}HNODE;
void main()
{ int n, i, box_count, box_volume, *a;
  HNODE *box_h, *box_t, *u;
  ELE *p, *q;
  printf("输入箱子容积\n"); scanf("%d", &box_volume);
  printf("输入物品种类\n"); scanf("%d", &n);
  a = (int *)malloc(sizeof(int) * n); /* 存储物品体积信息的数组 */
  printf("请按从大到小的顺序输入各物品的体积");
  for(i = 0; i < n; i++) scanf("%d", a-i);
  box_h = box_t = NULL; /* 预置已用箱子链为空 */
  box_count = 0; /* 预置已用箱子计数器 box_count 为 0 */
  for(i = 0; i < n; i++) { /* 物品 i 按以下步骤装箱 */
    /* 从第一只箱子开始顺序寻找能放入物品 i 的箱子 j */
    p = (ELE *)malloc(sizeof(ELE));
    p->vno = i;
    for(u = box_h; u != NULL; u = u->next)
  
```

```
if(u->remainder >= a[i]) break; /* 找到还可装物品 i 的箱子 */
if(u == NULL) { /* 已用箱子都不能装物品 i */
    u = (HNODE *)malloc(sizeof(HNODE)); /* 使用一只新的箱子 */
    u->remainder = box_volume = a[i];
    u->head = NULL;
    if(box_h == NULL) box_h = box_t = u;
    else box_t = box_t->next = u;
    u->next = NULL; box_count++;
}
else u->remainder -= a[i]; /* 将物品 i 放入箱子 j */
for(q = u->head; q != NULL && q->link != NULL; q = q->link);
if(q == NULL) { /* 新启用的箱子 */
    p->link = u->head; u->head = p;
}
else { p->link = u->head; u->head = p; }
}
/* 输出结果 */
printf("共使用%d 只箱子", box_count);
printf("各箱子装物品情况如下: ");
for(u = box_h, i = 1; u != NULL; u = u->next, i++) { /* 第 i 只箱子情况 */
    printf("第%2d 只箱子还剩余容积%4d, 所装物品有: \n", i, u->remainder);
    for(p = u->head; p != NULL; p = p->link)
        printf("%4d", p->vno + 1);
    printf("\n");
}
}
```

9.8 分支限界法

分支限界法类似于回溯法,也是一种在问题的解空间树 T 上搜索问题解的算法。但在一般情况下,分支限界法与回溯法的求解目标不同。回溯法的求解目标是找出 T 中满足约束条件的所有解,而分支限界法的求解目标则是找出满足约束条件的一个解,或是在满足约束条件的解中找出使某一目标函数值达到极大或极小的解,即在某种意义下的最优解。

由于求解目标不同,导致分支限界法与回溯法在解空间树 T 上的搜索方式也不相同。回溯法以深度优先的方式搜索解空间树 T ,而分支限界法则以广度优先或以最小耗费优先的方式搜索解空间树 T 。分支限界法的搜索策略是,每一个活节点只有一次机会成为扩展节点。活节点

一旦成为扩展节点,就一次性产生其所有儿子节点。在这些儿子节点中,那些导致不可行解或非最优解的儿子节点被舍弃,其余儿子节点被加入活节点表中。此后,从活节点表中取下一节点成为当前扩展节点,并重复上述节点扩展过程。这个过程一直持续到找到所需的解或活节点表为空时为止。人们已经利用分支限界法解决了大量离散最优化的实际问题。

从活节点表中选择下一扩展节点的不同方式导致不同的分支限界法。最常用的有以下两种:

1. 队列式(FIFO)分支限界法

队列式分支限界法将活节点表组织成一个队列,并按队列的先进先出原则选择下一个节点作为扩展节点。

2. 优先队列式分支限界法

优先队列式分支限界法将活节点表组织成一个优先队列,并按优先队列中规定的节点优先级选取优先级最高的下一个节点作为扩展节点。

优先队列中规定的节点优先级通常用一个与该节点相关的数值 p 来表示。节点优先级的高低与 p 值的大小相关。最大优先队列规定 p 值较大的节点优先级较高。在算法实现时通常用一个最大堆来实现最大优先队列,用最大堆的 Deletemax 运算抽取堆中下一个节点成为当前扩展节点。类似地,最小优先队列规定 p 值较小的节点优先级较高。在算法实现时常用一个最小堆来实现最小优先队列,用最小堆的 Deletemin 运算抽取堆中下一个节点成为当前扩展节点。

例如 $n=3$ 时的 0-1 背包问题的一个实例: $w=[16,15,15]$, $p=[45,25,25]$, $c=30$, 其解空间树见图 9-4 所示。

用队列式分支限界法解此问题时,用一个队列来存储活节点表。算法从根节点 A 出发。

(1) 初始时活节点队列为空。

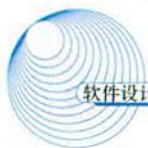
(2) 节点 A 是当前扩展节点,它的 2 个儿子节点 B 和 C 均可谓可行节点,故将这 2 个儿子节点按照从左到右的顺序加入活节点队列,并且舍弃当前扩展节点 A。

(3) 按照先进先出的原则,下一个扩展节点是活节点队列的队首节点 B。扩展节点 B 得到其儿子节点 D 和 E。由于 D 是不可行节点,故被舍去。E 是可行节点,被加入活节点队列。此时活节点队列中的元素是 C、E。

(4) C 成为当前扩展节点,它的 2 个儿子节点 F 和 G 均为可行节点,因此被加入活节点队列。此时活节点队列中的元素是 E、F、G。

(5) 扩展下一个节点 E,得到节点 J 和 K。J 是不可行节点,因而被舍去。K 是一个可行的叶节点,表示所求问题的一个可行解,其价值为 45。此时活节点队列中的元素是 F、G。

(6) 当前活节点队列的队首节点 F 成为下一个扩展节点。它的 2 个儿子节点 L 和 M 均为



叶节点。L 表示获得价值为 50 的可行解, M 表示获得价值为 25 的可行解。

(7) G 是最后一个扩展节点, 其儿子节点 N 和 O 均为可行叶节点。最后活节点队列为空, 算法终止。算法搜索得到最优值为 50。

9.9 概率算法简介

概率算法的一个基本特征是对所求解问题的同一实例用同一概率算法求解两次, 可能得到完全不同的效果。这两次求解所需的时间甚至所得到的结果可能会有相当大的差别。一般情况下, 可将概率算法大致分为数值概率算法、蒙特卡罗(Monte Carlo)算法、拉斯维加斯(Las Vegas)算法和舍伍德(Sherwood)算法 4 类。

(1) 数值概率算法常用于数值问题的求解。这类算法所得到的往往是近似解, 且近似解的精度随计算时间的增加不断提高。在多数情况下, 要计算出问题的精确解是不可能的或没有必要的, 因此用数值概率算法可得到相当满意的解。

(2) 蒙特卡罗算法用于求问题的精确解。用蒙特卡罗算法能求得问题的一个解, 但这个解未必是正确的。求得正确解得概率依赖于算法所用的时间。算法所用的时间越多, 得到正确解得概率就越高。蒙特卡罗算法的主要缺点也在于此, 一般情况下, 无法有效地判定所得到的解是否肯定正确。

(3) 拉斯维加斯算法不会得到不正确的解。一旦用拉斯维加斯算法找到一个解, 这个解就一定是正确解。拉斯维加斯算法找到正确解的概率随着它所用的计算时间的增加而提高。对于所求解问题的任一实例, 用同一拉斯维加斯算法反复对该实例求解足够多次, 可使求解实效的概率任意小。

(4) 舍伍德算法总能求得问题的一个解, 且所求得解总是正确的。当一个确定性算法在最坏情况下的计算复杂度与其在平均情况下的计算复杂度有较大差别时, 可在这个确定性算法中引入随机性将它改造成一个舍伍德算法, 消除或减少问题的好坏实例间的这种差别。舍伍德算法的精髓不时避免算法的最坏情况行为, 而是设法消除这种最坏情形行为与特定实例之间的关联性。

第 10 章 面向对象技术

面向对象 OO (Object-Oriented) 方法是一种非常实用的软件开发方法, 它一出现就受到软件技术人员的青睐, 现在已经成为计算机科学研究的一个重要领域, 并逐渐称为软件开发的一种主要方法。面向对象方法以客观世界中的对象为中心, 其分析和设计思想符合人们的思维方式, 分析和设计的结果与客观世界的实际比较接近, 容易被人们所接受。在面向对象方法中, 分析和设计的界线并不明显, 它们采用相同的符号表示, 能方便地从分析阶段平滑地过渡到设计阶段。此外, 在现实生活中, 用户的需求经常会发生变化, 但客观世界的对象以及对象间的关系相对比较稳定, 因此用面向对象方法分析和设计的结果也相对比较稳定。

10.1 面向对象的基本概念

Peter Coad 和 Edward Yourdon 提出用下面的等式识别面向对象方法:

面向对象 = 对象 (object)

+ 分类 (classification)

+ 继承 (inheritance)

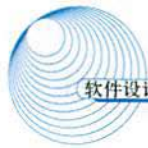
+ 通过消息的通信 (communication with messages)

可以说, 采用这 4 个概念开发的软件系统是面向对象的。

1. 对象

在面向对象的系统中, 对象是基本的运行时的实体, 它既包括数据 (属性), 也包括作用于数据的操作 (行为)。所以一个对象把属性和行为封装为一个整体。封装是一种信息隐蔽技术, 它的目的是使对象的使用者和生产者分离, 使对象的定义和实现分开。从程序设计者来看, 对象是一个程序模块; 从用户来看, 对象为他们提供了所希望的行为。在对象内的操作通常叫做方法。一个对象通常可由对象名、属性和操作 3 部分组成。

在现实世界中, 每个实体都是对象, 如学生、汽车、电视机、空调等都是现实世界中的对象。每个对象都有它的属性和操作, 如电视机有颜色、音量、亮度、灰度、频道等属性, 可以有切换频道、增大/减低音量等操作。电视机的属性值表示了电视机所处的状态, 而这些属性只能通过其提供的操作来改变。电视机的各组成部分, 如显像管、电路板、开关等都封装在电视机机箱中, 人们不知道也不关心电视机是如何实现这些操作的。



2. 消息

对象之间进行通信的一种构造叫做消息。当一个消息发送给某个对象时,包含要求接收对象去执行某些活动的信息。接收到信息的对象经过解释,然后予以响应。这种通信机制叫做消息传递。发送消息的对象不需要知道接收消息的对象如何对请求予以响应。

3. 类

一个类定义了一组大体上相似的对象。一个类所包含的方法和数据描述一组对象的行为和属性。把一组对象的共同特征加以抽象并存储在一个类中的能力,是面向对象技术最重要的一点;是否建立了一个丰富的类库,是衡量一个面向对象程序设计语言成熟与否的重要标志。

类是在对象之上的抽象,对象是类的具体化,是类的实例(instance)。在分析和设计时,我们通常把注意力集中在类上,而不是具体的对象。我们也不必为每个对象逐个定义,只需对类作出定义,而对类的属性的不同赋值即可得到该类的对象实例。

有些类之间存在一般和特殊关系,即一些类是某个类的特殊情况,某个类是一些类的一般情况。这是一种“is-a”关系,即特殊类是一种一般类。例如“汽车”类、“轮船”类、“飞机”类都是一种“交通工具”类。特殊类是一般类的子类,一般类是特殊类的父类。同样“汽车”类还可以有更特殊的类,如“轿车”类、“货车”类等。在这种关系下形成一种层次的关联。

通常把一个类和这个类的所有对象称为“类及对象”或对象类。

4. 继承

继承是父类和子类之间共享数据和方法的机制。这是类之间的一种关系,在定义和实现一个类的时候,可以在一个已经存在的类的基础上来进行,把这个已经存在的类所定义的内容作为自己的内容,并加入若干新的内容。图 10-1 表示了父类 A 和它的子类 B 之间的继承关系。

一个父类可以有多个子类,这些子类都是父类的特例,父类描述了这些子类的公共属性和操作。一个子类可以继承它的父类(或祖先类)中的属性和操作,这些属性和操作在子类中不必定义,子类中还可以定义自己的属性和操作。

图 10-1 中的 B 只从一个父类 A 得到继承,叫做“单重继承”。如果一个子类有两个或更多个父类,则称为“多重继承”。

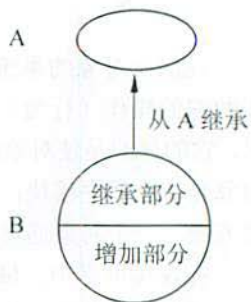


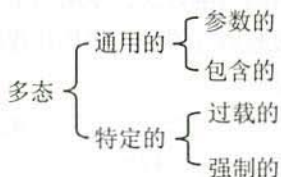
图 10-1 类的继承关系

5. 多态

在收到消息时,对象要予以响应。不同的对象收到同一消息可以产生完全不同的结果,这一现象叫做多态(polymorphism)。在使用多态的时候,用户可以发送一个通用的消息,而实现的细节则由接收对象自行决定。这样,同一消息就可以调用不同的方法。

多态的实现受到继承的支持,利用类的继承的层次关系,把具有通用功能的消息存放在高层次,而不同的实现这一功能的行为放在较低层次,在这些低层次上生成的对象能够给通用消息以不同的响应。

多态有几种不同的形式,Cardelli 和 Wegner 把它分为 4 类,如下所示。其中,参数多态和包含多态称为通用的多态,过载多态和强制多态称为特定的多态。



参数多态是应用比较广泛的多态,被称为最纯的多态。包含多态在许多语言中都存在,最常见的例子就是子类型化,即一个类型是另一个类型的子类型。过载(overloading)多态是同一个变量被用来表示不同的功能而通过上下文以决定一个名所代表的功能。

6. 动态绑定(dynamic binding)

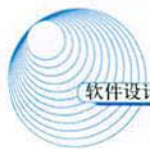
绑定是一个把过程调用和响应调用所需要执行的代码加以结合的过程。在一般的程序设计语言中,绑定是在编译时进行的,叫做静态绑定。动态绑定则是在运行时进行的,因此,一个给定的过程调用和代码的结合直到调用发生时才进行的。

动态绑定是和类的继承以及多态相联系的。在继承关系中,子类是父类的一个特例,所以父类对象可以出现的地方,子类对象也可以出现。因此在运行过程中,当一个对象发送消息请求服务时,要根据接收对象的具体情况将请求的操作与实现的方法进行连接,即动态绑定。

10.2 面向对象程序设计

面向对象程序设计(Object-Oriented Programming, OOP)的实质是选用一种面向对象程序设计语言(Object-Oriented Programming Language, OOPL),采用对象、类及其相关概念所进行的程序设计。

定义什么是面向对象、什么不是面向对象的程序设计的工作是困难的,这不依赖于程序设计语言(可用 C++ 语言编写纯 C 语言程序),而依赖于程序设计风格。当然好的 OOPL 必须至



少支持:

- 被封装的对象;
- 类和实例概念;
- 类间的继承性;
- 多态。

10.2.1 面向对象的好处

1. 面向对象是程序设计新范型

程序设计范型 (Programming Paradigm) 是人们在程序设计时所采用的基本方式模型。程序设计范型决定了程序设计时采用的思维方式、使用的工具, 同时又有一定的应用范畴。

程序设计范型的发展经历了过程程序设计、模块化程序设计、函数程序设计、逻辑程序设计, 发展到现在的面向对象程序设计范型。

面向对象程序设计范型是在上述的范型之上发展起来的, 它的关键在于加入类和继承性。类和继承性的设置, 进一步地提高了抽象的程度。

2. 面向对象的好处

Ian Graham 在 *Object-Oriented Methods Principles & Practice* 一书中, 将面向对象的好处总结如下:

(1) 对象技术解决了产品质量和生产率之间的权衡。精心设计的面向对象系统能够作为那些基本上用可重用构件组装而成的系统的基础, 从而创造了更高的生产率; 重用以前项目中经过测试的那些已经存在的类会使系统具有更高的质量。

(2) 面向对象程序设计, 特别是继承机制, 使得系统具有很高的灵活性和易扩充性。

(3) 面向对象是一个能管理复杂性并增强伸缩性的工具。

(4) 根据面向对象的观点, 以现实世界对应物为基础, 把某一领域分割成各种对象进行分析与设计, 常常比自顶向下进行功能分解的分析及设计更自然合理。

(5) 从概念模型化到分析、设计、编码可以实现无缝传递。

(6) 通过封装进行的信息隐蔽有助于建立安全的系统。

10.2.2 面向对象程序设计语言

在程序设计语言中引入面向对象概念的方法产生较早, 其根源可以追溯到 20 世纪 60 年代后期, 那时 Kristen Nygaard 和 Ole-Johan Dahl 在挪威计算中心开发出了一种叫做 Simula 67 的语言。这种语言在用于模拟的语言 Simula 的基础上, 第一次引入了类、协同程序 (Coroutines)

和子类的概念,非常类似于今天的面向对象程序设计语言。

20 世纪 70 年代中期,在 Xerox 公司的 Palo Alto 研究中心(Xerox PARC)中产生了第一个完整、健全的面向对象程序设计语言:Smalltalk,以后又进行了改进和扩充,形成了 Smalltalk-80。这种语言中的每一个元素都分别是一个对象,从语言本身的实现、程序设计环境,到所支持的程序设计风格,都是面向对象的。Simula 67 和 Smalltalk 的开发,是目前面向对象程序设计和面向对象程序设计语言研究中大多数工作的起点。

到今天为止,已经有大约 100 多种面向对象的和基于对象的程序设计语言。本节以综述的形式介绍几种目前比较流行的或者独特的 OOPs,其中包括 Smalltalk、Eiffel、C++和 Java。

1. Smalltalk

Smalltalk 语言起源于 20 世纪 60 年代末期,主要设计者 Alan Kay 还在犹他大学念研究生时,就认识到可以用 Simula 语言中的概念来充实他在图形方面的研究工作。Alan Kay 到 Xerox PARC 工作后,将这些早期思想发展为基于图形的交互式个人工作站概念,并萌发了设计一种新语言支持这些概念的想法。为 Smalltalk 的开发作出重要工作的另外两人是 Adale Goldberg 和 Dan Ingans。

Smalltalk 在 Xerox PARC 历经了多次重大修改,最终形成了 Smalltalk-80,事实上,Smalltalk 并不是一种单纯的程序设计语言,而是反映面向对象程序设计思想的程序设计环境。这个系统在系统本身的设计中强调了对象概念的归一性,引入了类、方法、实例等概念和术语,应用了单重继承和动态绑定,成为 OOPs 发展过程中的一个引人注目的里程碑。

在 Smalltalk-80 中,除了对象之外没有其他形式的数据,对一个对象的唯一操作就是向它发消息。在这种语言中,连类也被看成是对象——类是元类的实例。因此,定义一个类就是向元类发一条消息,请求元类执行它的 new 来生成一个实例,也就是生成这个类,而消息中的参数,就是关于这个类的说明。Smalltalk-80 全面支持面向对象的概念,表 10-1 给出了它关于这些概念所采用的术语,可以看出二者是完全一致的,表明了这种语言对 OOP 的深刻影响。

表 10-1 Smalltalk-80 关于面向对象概念的术语

面向对象概念	Smalltalk-80 术语
对象	对象
类	类
方法	方法
实例变量	实例变量
消息	消息
子类	子类
继承性	继承性

2. Eiffel

Eiffel 语言是继 Smalltalk-80 之后的另一个“纯”OOP。这种语言是由 OOP 领域中著名的专家 Bertrand Meyer 等人 20 世纪 80 年代后期在 ISE 公司(Interactive Software Engineering Inc.)



开发的,它的主要特点是全面的静态类型化、有大量的开发工具、支持多继承。在众多的 OOPs 中,Eiffel 是较早(1988 年)对多重继承提供支持的,它的一些实现策略(如同名冲突处理、异常处理等)已经对后来的 OOPs 的设计和实现产生了影响。

Eiffel 也全面支持面向对象的概念,表 10-2 给出了它关于这些概念所采用的术语。

3. C++

C++语言是一种面向对象的强类型化语言,由 AT&T 的 Bell 实验室于 1980 年推出,目前已被国外许多主要的计算机和软件生产企业选为替代 C 语言或与 C 语言并存的基本开发工具。

C++语言是 C 语言的一个向上兼容的扩充,而不是一种新语言。C++是一种支持多范型的程序设计语言,它既支持面向对象的程序设计,也支持面向过程的程序设计。它融合了 Simula 67 的几种面向对象机制——类、继承、虚拟函数,借鉴了 ALGOL 68 中的操作符过载、变量声明位置不受限制等特性,形成了一种“比 Smalltalk 更接近于机器、比 C 语言更接近于问题”的 OOP。

C++支持基本的面向对象概念:对象、类、方法、消息、子类和继承性。它同时支持静态类型和动态类型。表 10-3 给出了 C++中关于这些概念所采用的术语。

表 10-3 C++关于面向对象概念的术语

面向对象概念	Eiffel 术语	面向对象概念	Eiffel 术语
对象	对象	消息	函数调用
类	类	子类	派生类
方法	成员函数	继承性	派生
实例变量	数据成员		

C++完全支持多继承,并且通过使用 try/throw/catch 模式提供了一个完整的异常处理机制。

C++不提供自动的无用存储单元收集。这必须通过程序员来实现,或者通过编程环境提供合适的代码库来予以支持。

4. Java

Java 语言起源于 Oak 语言,Oak 语言被设计成能运行在设备(如微波炉、摄影机等)的嵌入芯片上。

表 10-2 Eiffel 关于面向对象概念的术语

面向对象概念	Eiffel 术语
对象	对象
类	类
方法	例程
实例变量	属性
消息	例程施用
子类	后代
继承性	继承性

表 10-4 Java 关于面向对象概念的术语

面向对象概念	Eiffel 术语	面向对象概念	Eiffel 术语
对象	对象	消息	函数调用
类	类	子类	派生类
方法	方法	继承性	派生
实例变量	属性		

Java 编译成伪代码 (P 码或者字节代码), 这需要一个虚拟机来对其进行解释。Java 的虚拟机在几乎每一种平台上都可以运行。这实质上使得开发是与及其独立无关的, 并且提供了通用的可移植性。

Java 把类的概念和接口 (Interface) 的概念区分开来, 并试图通过只允许接口的多继承来克服多继承的危险。

Java 的异常处理机制与 C++ 的 try/throw/catch 相类似, 但更加严密。在 Java 中, 通过声明轻型线程来处理并发性, 这些线程通过副作用和同步协议进行通信。

Java Beans 是组件, 即类和其所需资源的集合, 它们主要被设计用来提供定制的 GUI 小配件。

Java 有自己的对象请求代理技术——RMI (远程方法调用)。这使得应用能够跨越网络来调用执行在其他 Java 应用程序内的方法。从第 2 版起, Java 也包含了一个兼容 CORBA 的、语言独立的对象请求代理。Java 也包含了用于管理与关系数据库进行交互的类 JDBC 和用于基本图形的类。

10.2.3 程序设计语言中的 OOP 机制

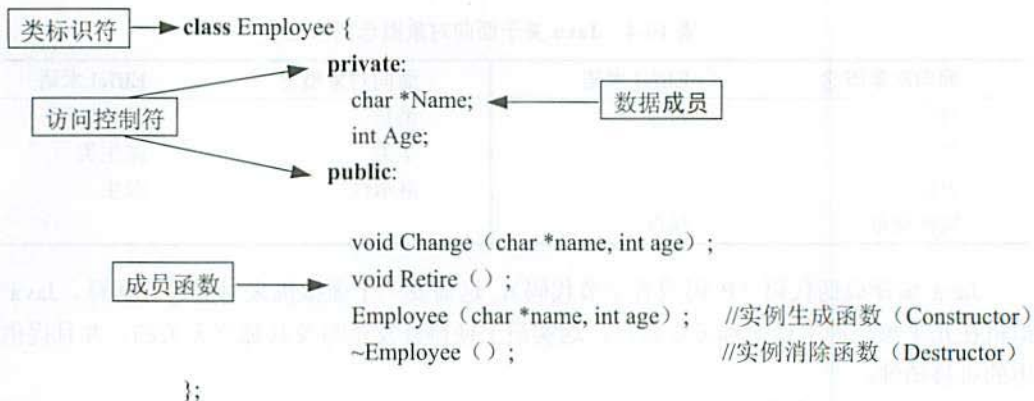
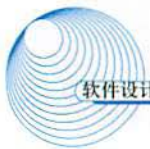
作为一种程序设计范型, OOP 主要解决程序设计方面的问题, 因而与程序设计语言的发展密切相关。特定的 OOP 概念一般是通过 OOPLs 中特定的语言机制来体现的。本节用 C++ 语言、从程序设计的角度进一步讨论这些概念。

OOP 现在已经扩展到系统分析和软件设计的范畴, 出现了面向对象分析和面向对象设计的概念, 这两部分内容见本章的 10.3 和 10.4 节。

1. 类

通常, 在介绍 OOP 的书籍或文章中总是先引入对象的概念, 然后从对对象进行抽象的角度来引入类的概念。但是, 当我们设计和实现一个面向对象的程序时, 首先接触到的不是对象, 而是类和类层次结构。

【例 10.1】 雇员类的定义。



类具有实例化功能,包括实例生成(由类的 Constructor 完成)和实例消除(由类的 Destructor 完成)。类的实例化功能决定了类及其实例具有下面的特征:

(1) 同一个类的不同实例具有相同的数据结构,承受的是同一方法集合所定义的操作,因而具有规律相同的行为;

(2) 同一个类的不同实例可以持有不同的值,因而可以具有不同的状态;

(3) 实例的初始状态(初值)可以在实例化时确定。

2. 继承和类层次结构

孤立的类只能描述实体集合的特征同一性,而客观世界中实体集合的划分通常还要考虑实体特征方面有关联的相似性。在 OOP 中使用继承机制解决这一问题。

在 OOPs 中,继承一般通过定义类之间的关系来体现。

在一个面向对象系统中,子类与父类之间的继承关系构成了这个系统的类层次结构,可以用树(对应于单继承)或格(对应于多继承)这样的图来描述。

【例 10.2】 经理类的定义。

```
class Manager : public Employee {  
    int Level;  
public:  
    void ChangeLevel (int n);  
    Manager (char *name, int age, int level);  
    ~Manager ();  
};
```

当执行一个子类的实例生成方法时,首先在类层次结构中从该子类沿继承路径上溯至它的一个基类,然后自顶向下地执行该子类所有父类的实例生成方法;最后执行该子类实例生成方

法的函数体。当执行一个子类的实例消除方法时,顺序恰好与执行该子类实例生成方法相反:先执行该子类实例消除方法的函数体,再沿继承路径自底向上地执行该子类所有父类的实例消除方法。

与一般数据类型的实例化过程相比,类的实例化过程是一种实例的合成过程,而不仅仅是根据单个类型进行的空间分配、初始化和联编。指导编译程序进行这种合成的,就是类层次结构。

OOPLs 中的继承机制体现了一条重要的面向对象程序设计原则:开发人员在构造程序时不必从零开始,而只需对差别进行程序设计。支持继承也是 OOPLs 与传统程序设计语言在语言机制方面最根本的区别。

3. 对象、消息传递和方法

对象是类的实例。尽管对象的表示在形式上与一般数据类型十分相似,但是它们之间存在一种本质区别:对象之间通过消息传递方式进行通信。

消息传递原是一种与通信有关的概念,OOP 使得对象具有交互能力的主要模型就是消息传递模型。对象被看成用传递消息的方式互相联系的通信实体,它们既可以接收、也可以拒绝外界传来的消息。一般情况下,对象接收它能够识别的消息,拒绝它不能识别的消息。对于一个对象而言,任何外部的代码都不能以任何不可预知或事先不允许的方式与这个对象进行交互。

发送一条消息至少应给出一个对象的名字和要发给这个对象的那条消息的名字。通常,消息的名字就是这个对象中外界可知的某个方法的名字。在消息中,经常还有一组参数(也就是那个方法所要求的参数),将外界的有关信息传给这个对象。

对于一个类来说,它关于方法界面的定义规定了实例的消息传递协议,而它本身则决定了消息传递的合法范围。由于类是先于对象构造而成的,所以一个类为它的实例提供了可以预知的交互方式。例如,假设 `m1` 是类 `Manager` 的一个实例(或对象),当外界要求把这个对象所代表的那位经理的级别改变为 2 时,就应以下面的方式向这个对象发出一条消息:

```
m1.ChangeLevel(2);
```

4. 对象自身引用

对象自身引用(Self-Reference)是 OOPLs 中的一种特有结构。这种结构在不同的 OOPLs 中有不同的名称,在 C++ 和 Java 中称为 `this`,在 Smalltalk-80、Object-C 和其他一些 OOPLs 中则称为 `self`。

以 C++ 语言为例,对于类 `c` 和方法 `c::m`,在 `c::m` 方法体中出现的 `c` 的成员名 `n` 将被编译程序按 `this->n` 来对待。这里的 `this` 是一个类型为 `c*` 的指针(在 Java 语言中 `this` 是一个引用),它的值由语言中的消息传递机制提供。



对象自身引用的值和类型分别扮演了两种意义的角色：对象自身引用的值使得方法体中引用的成员名与特定的对象相关，对象自身引用的类型则决定了方法体被实际共享的范围。

对象自身引用机制使得在进行方法的设计和实现时并不需要考虑与对象联系的细节，而是从更高一级的抽象层次，也就是类的角度来设计同类型对象的行为特征，从而使得方法在一个类及其子类的范围内具有共性。在程序运行过程中，消息传递机制和对象自身引用将方法与特定的对象动态地联系在一起，使得不同的对象在执行同样的方法体时，可以因对象的状态不同而产生不同的行为，从而使得方法对具体的对象具有个性。

5. 重置

重置(Overriding)的基本思想是，通过一种动态绑定机制的支持，使得子类在继承父类界面定义的前提下，用适合于自己要求的实现去替换父类中的相应实现。

在 OOPs 中，重置机制有相应的语法供开发人员选择使用。在 C++ 语言中，通过虚拟函数(Virtual Function)的定义来进行重置的声明，通过虚拟函数跳转表(Virtual Functions Jump Tables, Vtbl)结构来实现重置方法体的动态绑定。在 Java 语言中，通过抽象方法(Abstract Method)来进行重置的声明，通过方法查找(Method Lookup)实现重置方法体的动态绑定。

下面以 C++ 语言为例，说明重置机制的使用。

【例 10.3】 使用重置修改类 Employee 和 Manager。

```
class Employee {
protected:
    char *Name;
    int Age;
public:
    void Change (char * name, int age) ;
    virtual void Retire () ;    //虚拟函数
    Employee (char *name, int age) ;
    ~Employee () ;
};

class Manager : public Employee {
    int Level;
public:
    void Retire () ;           //重置从父类继承的 Retire 方法
    virtual void ChangeLevel (int n) ;//虚拟函数，供 Manager 的子类重置
    Manager (char *name, int age, int level) ;
    ~Manager () ;
};
```

6. 类属类

类属是程序设计语言中普遍注重的一种参数多态机制，在 OOPs 中也不例外。本节主要介绍与类有关的类属。

在 C++ 语言中，类属有专门的术语：template。

【例 10.4】类属类。

```
template <class T> class Vector {  
    T *v;  
    int sz;  
public:  
    T& operator[] (int i); // 运算符重载  
    Vector (int VectorSize);  
    /* ..... */  
};
```

其中<class T>中的 class 用来说明 T 是一个类型变元，单并不意味着 T 一定是一个类，也可以是基本类型。

类属类可以看成是类的模板。一个类属类是关于一组类的一个特性抽象，它强调的是这些类的成员特征中与具体类型无关的哪些部分，而与具体类型相关的那些部分则用变元来表示。这就使得对类的集合也可以按照特性的相似性再次进行划分。类属类的一个重要作用，就是对类库的建设提供了强有力的支持。

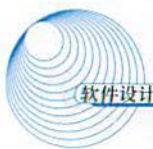
重置和类属都是一种多态机制，多态的基本概念以及多态的分类见本章 10.1 节，对于其他种类的多态不在本节进行介绍。

7. 无实例的类

在前几节中我们曾指出，类是对象的模板，对象是类的实例。那么是否每个类都至少有一个实例？如果在类之间没有定义继承关系，回答是肯定的。这是因为若存在没有实例的类，那么这样的类对程序的行为没有任何贡献，因而是冗余的。相反，如果存在继承关系，那么的确有可能在类层次结构的较高层次上看到始终没有实例的类。

要创建无实例的类，仍然需要语言的支持。在 C++ 和 Java 语言中，抽象类就是这样的类。在 C++ 中通过在类中定义纯虚拟函数来创建一个抽象类，在 Java 中通过在类中定义抽象方法来创建一个抽象类，或者直接将一个类声明为抽象类。

【例 10.5】用 C++ 语言创建抽象类。



```
class Person {    // 抽象类
protected:
    char *Name;
    /* ..... */
public:
    void Change (char *name, int age) ;
    virtual void PrintOn () = 0; // 纯虚拟函数
    Person () ;
    Person (char *name, int age, char *addr, char *tele) ;//重载的构造函数
    ~Person () ;
};
```

【例 10.6】用 Java 语言创建抽象类。

```
public class Person {    // 抽象类
    protected char *Name;
    /* ..... */
    public void Change (char *name, int age) {.....}
    public abstract void PrintOn () {.....} // 抽象方法
    public Person () {.....}
    public Person (char *name, int age, char *addr, char *tele) {.....}
}
```

或者将整个类命名为抽象类，而不管这个类中是否包含抽象方法。

```
abstract class Person {.....}
```

10.2.4 面向对象的程序

1. 类库

类库是一种预先定义的程序库，可以由开发人员自己扩充。

类库以程序模块的形式，按照类层次结构把一组类的定义和实现组织在一起。通常，这一组类预先提供的是一些低层功能，如输入/输出例程、图形操作原语、基本数据结构等，这些功能覆盖了传统例程库所提供的功能。

开发人员可以直接使用类库中的类，其方式与使用语言中的基本类型完全相同。开发人员可以扩充类库，办法是定义和实现类库中已有类的子类，再将这样的子类加入类库。

要想发挥面向对象的软件构造方式的优点，开发人员必须知道类库是怎样组织起来的。衡量一个开发人员的好坏，要看他是否知道如何来最好地发挥已有类库的优点，看他有没有能力将已有的类库与新的问题紧密地匹配起来，还要看他不得不另外编写的代码是不是最少。

大多数程序设计语言都有类库。最早引入类库的语言是 Smalltalk。Smalltalk 类库中基本的类层次结构片断如图 10-2 所示(为了清晰起见,图中略去了表示继承方向的箭头)。

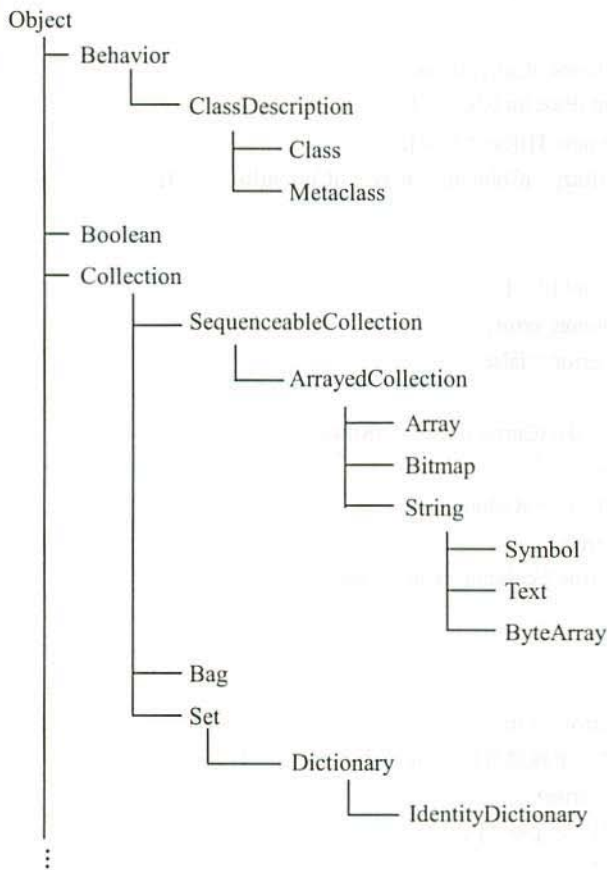


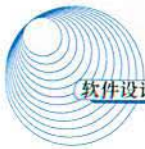
图 10-2 Smalltalk 类库中基本的类层次结构

2. 面向对象程序设计实例

【例 10.7】C++语言本身不提供对数组下标越界的判断。为了解决这一问题,本例的程序中定义了相应的模板类,使得对于任意类型的二维数组,可以在访问数组元素的同时,对行下标和列下标进行越界判断,并给出相应的提示信息。

【例 10.7 程序】

```
#include <iostream.h>
```

```
template <class T> class Array;
template <class T> class ArrayBody {
friend class Array<T>;
    T* tpBody;
    int iRows, iColumns, iCurrentRow;
    ArrayBody (int iRsz, int iCsz) {
        tpBody = new T[iRsz * iCsz];
        iRows = iRsz; iColumns = iCsz; iCurrentRow = -1;
    }
public:
    T& operator[] (int j) {
        bool row_error, column_error;
        row_error = column_error = false;
        try {
            if (iCurrentRow < 0 || iCurrentRow >= iRows)
                row_error = true;
            if (j < 0 || j >= iColumns)
                column_error = true;
            if (row_error == true || column_error == true)
                throw 'e';
        }
        catch (char) {
            if (row_error == true)
                cerr << "行下标越界[" << iCurrentRow << "]" ";
            if (column_error == true)
                cerr << "列下标越界[" << j << "]" ";
            cout << "\n";
        }
        return tpBody[iCurrentRow * iColumns + j];
    }
    ~ArrayBody () { delete[] tpBody; }
};

template <class T> class Array {
    ArrayBody<T> tBody;
public:
    ArrayBody<T> & operator[] (int i) {
        tBody.iCurrentRow = i;
        return tBody;
    }
};
```

```

    }
    Array (int iRsz, int iCsz) : tBody (iRsz, iCsz) {}
};

void main ()
{
    Array<int> a1 (10, 20) ;
    Array<double> a2 (3, 5) ;
    int b1;
    double b2;
    b1 = a1[-5][10]; // 有越界提示: 行下标越界[-5]
    b1 = a1[10][15]; // 有越界提示: 行下标越界[10]
    b1 = a1[1][4]; // 没有越界提示
    b2 = a2[2][6]; // 有越界提示: 列下标越界[6]
    b2 = a2[10][20]; // 有越界提示: 行下标越界[10] 列下标越界[20]
    b2 = a2[1][4]; // 没有越界提示
}

```

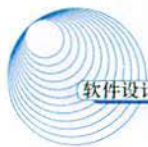
10.3 面向对象开发技术

众所周知,采用系统化的开发方法开发软件(特别是大规模软件)系统才可获很好的系统。什么是“好”系统呢?这个问题应从使用者(外部)和开发者(内部)两个角度来回答。从使用者观点出发,需要系统具有易学易用、界面友好、当正确使用时能快速给出正确结果、效率高等优点,还要求系统安全可靠等;从系统开发者和管理者角度出发,要求系统易于修改和扩充、易于理解、易于测试和重用、易于与其他系统兼容和管理等。虽然不同的系统所强调的特性可能不同,但上述所要求的系统特性是基本特性。

相对而言,面向对象的开发技术更有利于开发具有上述特性的软件系统。本章简要介绍面向对象开发技术的基本概念和相关步骤。

10.3.1 面向对象分析

同其他分析方法一样,面向对象分析(Object-Oriented Analysis, OOA)的目的是为了获得对应用问题的理解。理解的目的是确定系统的功能、性能要求。面向对象分析法与功能/数据分析法之间的差别是前期的表述含义不同。功能/数据分析法分开考虑系统的功能要求和数据及其结构,面向对象分析方法是將数据和功能结合在一起作为一个综合对象来考虑。面向对象分析技术可以将系统的行为和信息间的关系表示为迭代构造特征。



面向对象分析包含认定对象、组织对象、描述对象间的相互作用、定义对象的操作、定义对象的内部信息 5 个活动。

1. 认定对象

在应用领域中,按自然存在的实体确立对象。在定义域中,首先将自然存在的“名词”作为一个对象,这通常是研究问题、定义域实体的良好开始。通过实体间的关系寻找对象常常没有问题,而困难在于寻找(选择)系统关心的实质性对象,实质性对象是系统稳定性的基础。例如在银行应用系统中,实质性对象应包含客户账务、清算等,而门卫值班表不是实质性对象,甚至可以不包含在该系统中。

2. 组织对象

分析对象间的关系,将相关对象抽象成类,其目的是为了简化关联对象,利用类的继承性建立具有继承性层次的类结构。抽象类时可从对象间的操作或一个对象是另一个对象的一部分来考虑,如房子由门和窗构成,门和窗是房子类的子类。由对象抽象类,通过相关类的继承构造类层次,所以说系统的行为和信息间的分析过程是一种迭代表征过程。

3. 描述对象间的相互作用

描述出各对象在应用系统中的关系。如一个对象是另一个对象的一部分,一个对象与其他对象间的通信关系等。这样可以完整地描述每个对象的环境,由一个对象解释另一个对象,以及一个对象如何生成另一个对象,最后得到对象的界面描述。

4. 定义对象的操作

当考虑对象的界面时,自然要考虑对象的操作。其操作有从对象直接标识的简单操作,如创建、增加、删除等;也有更复杂的操作,如将几个对象的信息连接起来。一般而言,避免对象太复杂比较好,当连接的对象太复杂时,可将其标识为新对象。

5. 定义对象的内部信息

当确定了对象的操作后,再定义对象的内部,对象内部定义包括其内部数据信息、信息存储方法,继承关系以及可能生成的实例数等属性。

分析阶段最重要的是理解问题域的概念,其结果将影响整个工作。经验表明,从应用定义域概念标识对象是非常合理的,完成上述工作后写出规范文档,文档确定每个对象的范围。

早期面向对象的目标之一是简化模型与问题域之间的语义差距。事实上,面向对象分析的基础是软件系统结构,这依赖于人类看待现实世界的方法。当人们理解求解问题的环境时,常

采用对象、分类法和层次性这类术语。面向对象分析与功能/数据分析方法相比,面向对象的结果比较容易理解和管理。面向对象分析方法的另一个优点是便于修改,早期阶段的修改容易提高软件的可靠性。

10.3.2 面向对象设计

面向对象设计(Object-Oriented Design, OOD)的含义是设计分析模型和实现相应源代码,在目标代码环境中这种源代码可被执行。通常情况是,由概念模型生成的分析模型被装入到相应的执行环境中时还需要修改。

同所有其他设计活动一样,要使系统的结构优化且效率高,做好相互间的平衡是困难的。分析模型已经提供了概念结构,它将试图长期保存。另外,设计期必须充分考虑系统的稳定性,如目标环境所需的最大存储空间、可靠性和响应时间等,所有这些都会影响系统的结构。

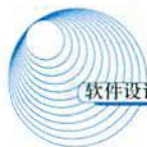
对象标识期间的目标是分析对象,设计过程也是发现对象的过程,我们称之为再处理。因此,必须有从分析模型到设计模型到程序设计语言的线性转换规则。对象可以用预先开发的源代码实现,我们称这样的部分为构件(Component)。由于这些构件是功能和数据的组装,因此,常常用于简化面向对象环境的产生。

如前所述,面向对象系统的开发需要面向对象的程序设计风格,这与OOP无直接关系。面向对象是一种程序设计风格,而不只是一种具有构造继承性、封装性和多态的程序设计语言族的命名。

10.3.3 面向对象测试

就测试而言,用面向对象方法开发的系统测试与其他方法开发的系统测试没有什么不同,在所有开发系统中都是根据规范说明来验证系统设计的正确性。程序验证应尽可能早地开始。程序调试步骤是从最底层开始,从单元测试、综合测试到系统测试。单元测试是系统构件的分体测试,将测试好的系统构件接起来看它们之间相互作用的正确性称综合测试,最后是整个系统的测试,包括软件系统所在相关环境的测试。通常综合测试是一种“主攻”活动,在系统开发期是非常关键的。这一阶段应随时连接已开发的每一部分,再看它们的实际工作,这种“主攻”活动在面向对象系统中是一种实质性的、渐渐增长的测试策略。

面向对象的系统由一些对象和它们相互间的通信组成,这些对象包括了它们的数据属性和操作(动作),比传统系统开发方法中独立工作的子程序大。对象的操作局限于特定数据,一个对象的所有操作自然由同一个设计者开发。这导致单元测试比传统系统的单元大,综合测试尽可能在早期阶段处理,因为通信是系统开发的实质。所有对象有预定义的界面,这也有利于综合测试。当综合测试进行到较高层次时,那么越来越多的对象就会被逐步连接起来。面向对象的综合测试是由底向上的测试。



一般来说,对面向对象软件的测试可分为下列4个层次进行:

(1) 算法层。测试类中定义的每个方法,基本上相当于传统软件测试中的单元测试。

(2) 类层。测试封装在同一个类中的所有方法与属性之间的相互作用。在面向对象软件中类是基本模块,因此可以认为这是面向对象测试中所特有的模块测试。

(3) 模板层。测试一组协同工作的类之间的相互作用。大体上相当于传统软件测试中的集成测试,但是也有面向对象软件的特点(例如,对象之间通过发送消息相互作用)。

(4) 系统层。把各个子系统组装成完整的面向对象软件系统,在组装过程中同时进行测试。

软件工程中传统的测试用例设计技术,如逻辑覆盖、等价类划分、边界值分析等方法,仍然可以作为测试类中每个方法的主要技术。面向对象测试的主要目标也是用尽可能低的测试成本和尽可能少的测试用例发现尽可能多的错误。但是面向对象程序中特有的封装、继承和多态等机制,也给面向对象测试带来一些新特点,增加了测试和调试的难度。

10.4 面向对象分析与设计方法

在实现一个系统之前应当十分重视对系统需求的理解和交流,这一点已经是公认的准则。结构化分析方法产生于20世纪60年代,基本思想是对一个系统的功能构件进行分解,将一种更易于管理、更加确定的方法引入系统分析阶段。面向对象分析强调的则是对一个系统中对象的特征和行为的定义。

目前,国际上已经出现了多种面向对象的方法,例如 Peter Coad 和 Edward Yourdon 的 OOA 和 OOD 方法、Booch 的 OOD 方法、OMT (Object Modeling Technique, 面向对象建模技术) 方法,以及 UML (Unified Modeling Language, 统一建模语言)。本节将对这4种方法进行简要介绍。

10.4.1 Peter Coad 和 Edward Yourdon 的 OOA 和 OOD 方法

1. OOA (Object-Oriented Analysis)

OOA 模型由下列5个层次和5个活动组成。

5个层次是主题层、对象类层、结构层、属性层、服务层。

5个活动是标识对象类、标识结构、定义主题、定义属性、定义服务。

在这种方法中定义了两种对象类之间的结构,一种称为分类结构,一种称为组装结构。分类结构就是一般与特殊关系,是一种“is-a”关系。组装结构则反映了对象之间的整体与部分关系,例如“计算机”对象由“显示器”、“主机箱”、“键盘”、“鼠标”等对象组成,而“主机箱”对象又可由“主板”、“CPU”、“内存”、“风扇”等对象组成。它实际上是一种“has-a”的关系。

主题提供了控制读者在一段时间内考虑和理解模型的范围的一种机制,在 OOA 中将与某一主题有关的对象用主题框围起来,其目的是减少读者理解系统的复杂程度。

OOA 在定义属性的同时,还要识别实例连接。实例连接是一个实例对象与另一个实例对象的映射关系(或者说是一种简单的对应关系)。例如一个公司有多个职员,一个职员只能在一家公司工作,那么“公司”类的实例与“职员”类的实例间就有 1 对多的实例连接关系。OOA 在定义服务的同时要识别消息连接。当一个对象需要向另一对象发送消息时,它们之间就存在消息连接。

需要强调的是这 5 个活动不是必须顺序进行的,有些分析员喜欢先识别对象类,然后定义属性、服务,再识别结构和主题;有的则喜欢先识别对象类、结构和主题,再定义属性和服务。总之,5 个活动都完成了,OOA 的模型就建立了。5 个层次并不是构成软件系统的层次,而是一旦建立了模型,就可以在 5 个层次上进行表示和复审。

2. OOD (Object-Oriented Design)

OOA 中的 5 个层次和 5 个活动继续贯穿在 OOD 过程中。OOD 模型由 4 个部分和 4 个活动组成如图 10-3 所示。

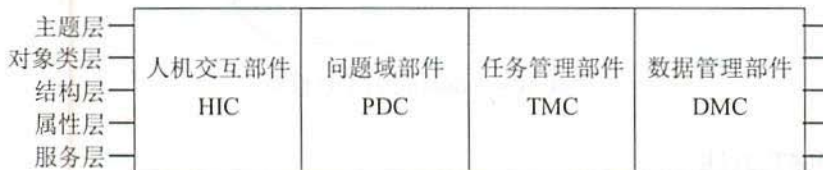


图 10-3 Coad & Yourdon 的 OOD 模型

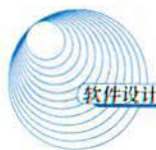
4 个活动是设计问题域部件、设计人机交互部件、设计任务管理部件、设计数据管理部件。

(1) 设计问题域部件: OOA 的结果恰好就是 OOD 的问题域部件,分析的结果在 OOD 中可以被改动或增补,但基于问题域的总体组织框架是长时间稳定的。

(2) 设计人机交互部件: 人机交互部件在上述结果中加入人机交互的设计和交互的细节,包括窗口和输出报告的设计。可以用原型来帮助实际交互机制的开发和选择。

(3) 设计任务管理部件: 这部分主要是识别事件驱动任务,识别时钟驱动任务,识别优先任务和关键任务,识别协调者,审查每个任务并定义每个任务。

(4) 设计数据管理部件: 数据管理部件提供了在数据管理系统中存储和检索对象的基本结构,其目的是隔离数据管理方案(如普通文件、关系数据库、面向对象数据库等)对其他部分的影响。



10.4.2 Booch 的 OOD 方法

Booch 认为软件开发是一个螺旋上升的过程,在螺旋上升的每个周期中有以下步骤:

- (1) 标识类和对象;
- (2) 确定它们的含义;
- (3) 标识它们之间的关系;
- (4) 说明每一个类的界面和实现。

Booch 的 OOD 模型如图 10-4 所示。

除了类图、对象图、模块图、进程图外,Booch 的 OOD 中还使用了两种动态描述图,一种是刻画特定类实例的状态转换图,另一种是描述对象间事件变化的时序图。



图 10-4 Booch 的 OOD 模型

10.4.3 OMT 方法

对象建模技术(Object Modeling Technique, OMT)是由 J.Rumbaugh 等人提出的。OMT 定义了 3 种模型,它们是对象模型、动态模型和功能模型,OMT 用这 3 种模型来描述系统。OMT 方法有 4 个步骤:分析、系统设计、对象设计和实现。OMT 方法的每一步都使用这 3 种模型,通过每一步对 3 种模型不断地精化和扩充。

1. 对象模型、动态模型和功能模型

1) 对象模型

对象模型描述系统中对象的静态结构、对象之间的关系、对象的属性、对象的操作。对象模型表示静态的、结构上的、系统的“数据”特征。对象模型为动态模型和功能模型提供了基本的框架。对象模型用包含对象和类的对象图来表示。

OMT 的对象模型中除了对象、类、继承外,还有一些其他的概念,下面介绍几个主要的概念。

(1) 链(Link)和关联(Association)。链表示实例对象间的物理或概念上的连接。例如在表 10-5 中, Joe Doe 为 Simplex 公司工作, 工资 2000 元。关联描述具有公共结构和公共语义的一组链, 例如关联 works-for 描述了一组某人为某公司工作的链。实际上链是关联的一个实例。链可以有属性, 称为链属性, 链属性表示关联中链的性质。图 10-5 给出了链、关联、链属性的一个实例。图中的实心圆是关联的阶(也称为重数), 阶指出一个类的多少个实例可以与所关联的类的一个实例相关。实心圆表示 0 或多个, 空心圆表示 0 或 1 个, 没有圆表示 1 个。

表 10-5 人员登记表

Person	salary	company
Joe Doe	2000	Simplex
Mary Brown	1500	Simplex
Jean Smith	2200	United Widgets

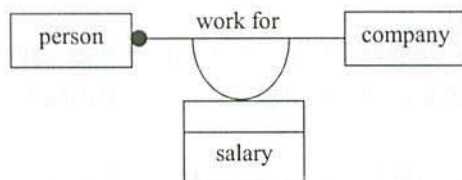


图 10-5 链、关联、链属性

(2) 泛化(Generalization)。泛化是一个类与它的一个或多个细化种类之间的关系, 即一般与特殊的关系。被细化的类称为父类, 每个细化的种类称为子类, 子类可以继承父类的性质。

(3) 聚集(Aggregation)。聚集是一种整体与部分的关系, 在这种关系中表示整体的对象与表示部分的对象关联。图 10-6 给出了泛化和聚集的实例。

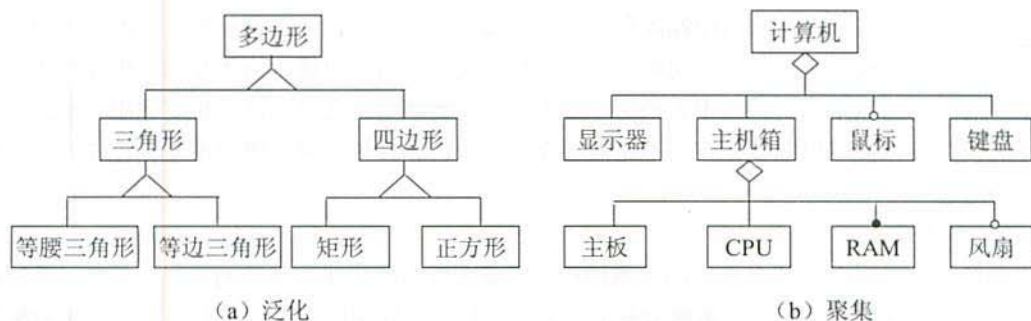


图 10-6 泛化和聚集



(4) 模块 (Module)。模块是组合类、关联和泛化的一种逻辑结构。模块给出了某个主题的视图。

2) 动态模型

动态模型描述与时间和操作顺序有关的系统特征——激发事件、事件序列、确定事件先后关系以及事件和状态的组织。动态模型表示瞬时的、行为上的、系统的“控制”特征。动态模型用状态图来表示。每张状态图显示了系统中一个类的所有对象所允许的状态和事件的顺序。

3) 功能模型

功能模型描述与值的变换有关的系统特征——功能、映射、约束和函数依赖。功能模型用数据流图来表示。

4) 对象模型、动态模型和功能模型之间的关系:

(1) 与功能模型的关系: 对象模型展示了功能模型中的动作者、数据存储和流的结构; 动态模型展示了执行加工的顺序。

(2) 与对象模型的关系: 功能模型展示了类上的操作和每个操作的变量, 因此它也表示了类之间的“供应者—客户”关系; 动态模型展示了每个对象的状态以及它接收事件和改变状态时所执行的操作。

(3) 与动态模型的关系: 功能模型展示了动态模型中未定义的不可分解的动作和活动的定义; 对象模型展示了是谁改变了状态和承受了操作。

2. OMT 的步骤

1) 分析

分析是 OMT 方法的第一步, 其目的是建立可理解的现实世界模型。分析从问题陈述入手, 通过与客户的不间断交互以及对现实世界背景知识的了解, 对能反映系统的 3 个本质特征 (对象类及它们之间的关系、动态的控制流和受约束的数据的函数变换) 进行分析, 并构造出现实世界的模型。分析模型必须简洁、明确地抽象目标系统必须做的事情, 而不是如何做这些事情。模型中的对象应该是应用领域中的概念, 而不是诸如数据结构之类的计算机实现中的概念。一个好的模型应该被应用领域专家而不是被程序员所理解和评价。分析模型不应包含任何与实现有关的考虑。

2) 系统设计

系统设计阶段确定整个系统的体系结构, 形成求解问题和建立解答的高层次策略。系统设计人员必须形成的决策有: 将系统分解成子系统、标识问题中固有的并发性、将子系统分配到处理器和任务、选择数据存储管理的手段、处理对全局资源的访问、选择软件中控制的实现、处理边界条件、设置折中的优先级等。

3) 对象设计

在分析的基础上,对象设计阶段建立基于分析模型的设计模型,并考虑实现的细节。设计人员根据系统设计期间建立的策略把实现细节加入到设计模型中。对象设计的焦点是实现每个类的数据结构及所需的算法。分析模型中的对象类仍然是有意义的,但它们中都加入了数据结构和算法。

对象设计期间,设计者必须履行下列步骤:综合考虑3个模型以获得类的操作;设计实现操作的算法,优化数据访问路径,实现与外部交互的控制;调整类结构以增加继承,设计关联,确定对象表示,将类和关联包装到模块中。

4) 实现

实现阶段将对象设计阶段开发的对象类及其关系转换成特定的程序设计语言、数据库或硬件的实现。程序设计是开发周期中相对较小且机械的部分,因为在对象设计中已形成了所有重要的决策。目标语言在某种程度上能影响设计决策,但设计不应依赖于程序设计语言的细节。在实现期间重要的是应该有一些好的软件工程准则作指导。

10.4.4 UML 概述

统一建模语言(Unified Modeling Language, UML)是面向对象软件的标准化建模语言。由于其简单、统一又能够表达软件设计中的动态和静态信息,目前已经成为可视化建模语言事实上的工业标准。

从企业信息系统到基于Web的分布式应用,甚至严格的实时嵌入式系统都适合用UML来建模。它是一种富有表达力的语言,可以描述开发所需要的各种视图,然后以此为基础装配系统。UML由3个要素构成:UML的基本构造块、支配这些构造块如何放置在一起的规则和运用与整个语言的一些公共机制。限于篇幅,下面仅对UML中的基本构造块进行讨论。

UML的词汇表包含3种构造块:事物、关系和图。事物是对模型中最具有代表性的成分的抽象;关系把事物结合在一起;图聚集了相关的事物。

1. 事物

UML中有4种事物:结构事物、行为事物、分组事物和注释事物。

1) 结构事物 (Structural Thing)

结构事物是UML模型中的名词。它们通常是模型的静态部分,描述概念或物理元素。结构事物包括类(Class)、接口(Interface)、协作(Collaboration)、用例(Use case)、主动类(Active Class)、构件(Component)和节点(Node)。

各种结构事物的图形化表示如图10-7所示。

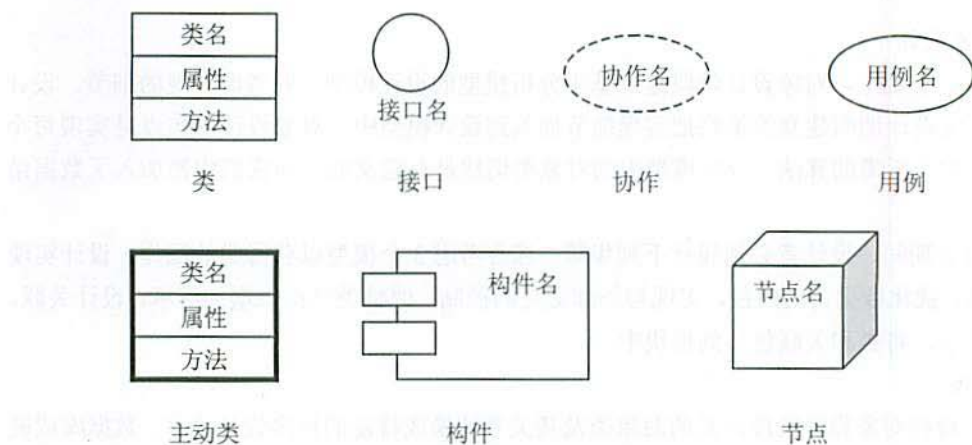
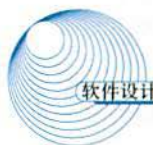


图 10-7 行为事物的图形表示

2) 行为事物 (Behavior Thing)

行为事物是 UML 模型的动态部分。它们是模型中的动词，描述了跨越时间和空间的行为。共有两类主要的行为事物：交互 (Interaction) 和状态机 (State Machine)。

(1) 交互由在特定语境中共同完成一定任务的一组对象之间交换的消息组成。一个对象群体的行为或单个操作的行为可以用一个交互来描述。交互涉及一些其他元素，包括消息、动作序列（由一个消息所引起的行为）和链（对象间的连接）。在图形上，把一个消息表示为一条有向直线，通常在表示消息的线段上总有操作名。如图 10-8 所示。

(2) 状态机描述了一个对象或一个交互在生命期内响应事件所经历的状态序列。单个类或一组类之间协作的行为可以用状态机来描述。一个状态机涉及到一些其他元素，包括状态、转换（从一个状态到另一个状态的流）、事件（触发转换的事物）和活动（对一个转换的响应）。在图形上，把状态表示为一个圆角矩形，通常在圆角矩形中含有状态的名称及其子状态，如图 10-9 所示。

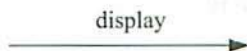


图 10-8 消息

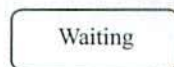


图 10-9 状态

3) 分组事物 (Grouping Thing)

分组事物是 UML 模型的组织部分。它们是一些由模型分解成的“盒子”。在所有的分组事物中，最主要的分组事物是包 (Package)。包是把元素组织成组的机制，这种机制具有多种用途。结构事物、行为事物甚至其他分组事物都可以放进包内。包不像构件（仅在运行时存在），它纯粹是概念上的（即它仅在开发时存在）。包的图形化表示如图 10-10 所示。

4) 注释事物 (Annotational Thing)

注释事物是 UML 模型的解释部分。这些注释事物用来描述、说明和标注模型的任何元素。注解 (Note) 是一种主要的注释事物。注解是一个依附于一个元素或者一组元素之上, 对它进行约束或解释的简单符号。注解的图形化表示如图 10-11 所示。

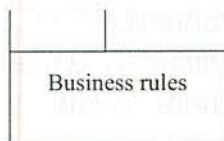


图 10-10 包

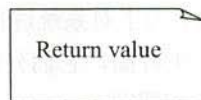


图 10-11 注解

2. 关系

UML 中有 4 种关系: 依赖、关联、泛化和实现。

1) 依赖 (Dependency)

依赖是两个事物间的语义关系, 其中一个事物 (独立事物) 发生变化会影响另一个事物 (依赖事物) 的语义。在图形上, 把一个依赖画成一条可能有方向的虚线, 如图 10-12 所示。

2) 关联 (Association)

关联是一种结构关系, 它描述了一组链, 链是对象之间的连接。聚集 (Aggregation) 是一种特殊类型的关联, 它描述了整体和部分间的结构关系。关联和聚集的图形化表示如图 10-13 和图 10-14 所示。在关联上可以标注重复度 (Multiplicity) 和角色 (Role)。



图 10-12 依赖



图 10-13 关联



图 10-14 聚集

3) 泛化 (Generalization)

泛化是一种特殊/一般关系, 特殊元素 (子元素) 的对象可替代一般元素 (父元素) 的对象。用这种方法, 子元素共享了父元素的结构和行为。在图形上, 把一个泛化关系画成一条带有空心箭头的实线, 它指向父元素, 如图 10-15 所示。

4) 实现 (Realization)

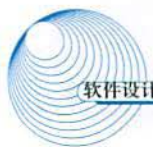
实现是类元之间的语义关系, 其中一个类元指定了由另一个类元保证执行的契约。在两种地方要遇到实现关系: 一种是在接口和实现它们的类或构件之间; 另一种是在用例和实现它们的协作之间。在图形上, 把一个实现关系画成一条带有空心箭头的虚线, 如图 10-16 所示。



图 10-15 泛化



图 10-16 实现



这4种关系是UML模型中可以包含的基本关系事物。它们也有变体,例如,依赖的变体有精化、跟踪、包含和延伸。

3. UML 中的图

图(Diagram)是一组元素的图形表示,大多数情况下把图画成顶点(代表事物)和弧(代表关系)的连通图。为了对系统进行可视化,可以从不同的角度画图,这样图是对系统的投影。

UML 提供了9种图,它们分别是:类图、对象图、用例图、序列图、协作图、状态图、活动图、构件图和部署图。

1) 类图

类图(Class Diagram)展现了一组对象、接口、协作和它们之间的关系。在面向对象系统的建模中所建立的最常见的图就是类图。类图给出系统的静态设计视图。包含主动类的类图给出了系统的静态进程视图。

类图中通常包括类、接口、协作、依赖、泛化和关联关系等内容(如图10-17所示)。

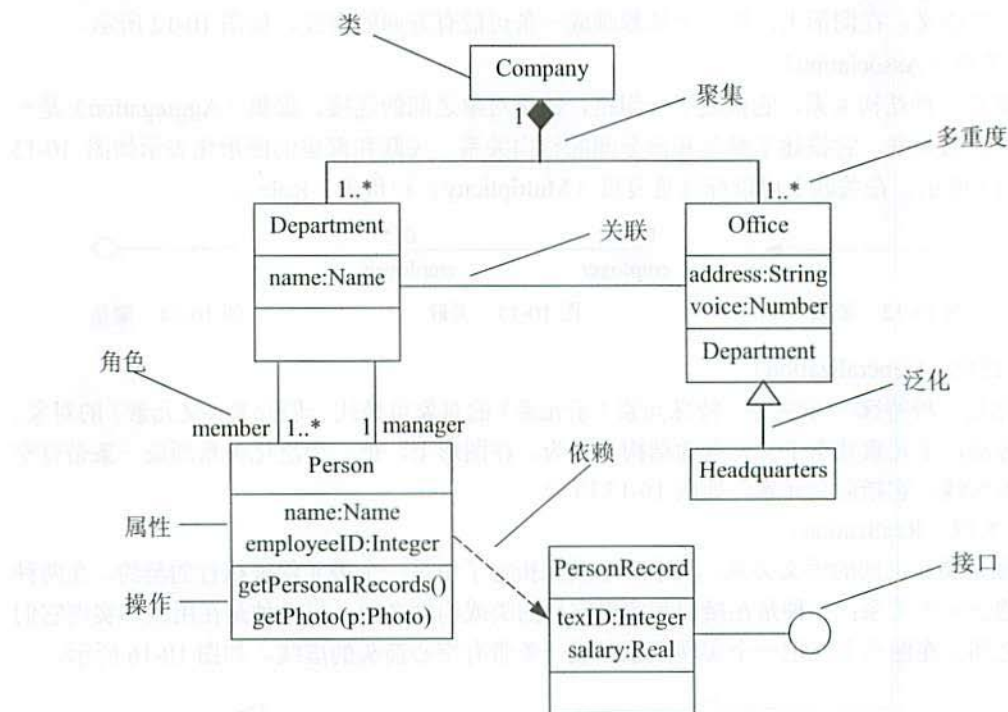


图 10-17 UML 类图

类图中也可以包含注解和约束,类图还可以含有包或子系统,二者都用于把模型元素聚集成更大的组块。

类图用于对系统的静态设计视图建模。这种视图主要支持系统的功能需求,即系统要提供给最终用户的服务。当对系统的静态设计视图建模时,通常以下述3种方式之一使用类图。

(1) 对系统的词汇建模。对系统的词汇建模涉及到做出这样的决定:哪些抽象是考虑中的系统的一部分,哪些抽象处于系统边界之外。用类图详细描述这些抽象和它们的职责。

(2) 对简单的协作建模。协作是一些共同工作的类、接口和其他元素的群体,该群体提供的一些合作行为强于所有这些元素的行为之和。例如当对分布式系统的事务语义建模时,不能仅仅盯着一个单独的类来推断要发生什么,而要有相互协作的一组类来实现这些语义。用类图对这组类以及它们之间的关系进行可视化和详述。

(3) 对逻辑数据库模式建模。将模式看作为数据库的概念设计的蓝图。在很多领域中,要在关系数据库或面向对象数据库中存储永久信息。可以用类图对这些数据库的模式建模。

2) 对象图

对象图(Object Diagram)展现了一组对象以及它们之间的关系。对象图描述了在类图所建立的事物的实例的静态快照。

对象图一般包括对象和链。与类图一样,对象图给出系统的静态设计视图或静态进程视图,但它们是从真实的或原型案例的角度建立的。这种视图主要支持系统的功能需求,即系统应该提供给最终用户的服务。利用对象图可以对静态数据结构建模。

当对系统的静态设计视图或静态进程视图建模时,主要是使用对象图对对象结构进行建模。对对象结构建模涉及到在给定时刻抓取系统中的对象的快照。对象图表示了交互图表示的动态场景的一个静态画面,可以使用对象图可视化、详述、构造和文档化系统中存在的实例以及它们之间的相互关系。

3) 用例图

用例图(Use Case Diagram)展现了一组用例、参与者(Actor)以及它们之间的关系。用例图通常包括用例、参与者、扩展关系、包含关系。如图10-18所示。

用例图用于对系统的静态用例视图进行建模。这个视图主要支持系统的行为,即该系统在它的周边环境的语境中所提供的外部可见服务。

当对系统的静态用例视图建模时,可以用下列两种方式来使用用例图。

(1) 对系统的语境建模。对一个系统的语境进行建模,包括围绕整个系统画一条线,并声明有哪些参与者位于系统之外并与系统进行交互。在这里,用例图说明了参与者以及他们所扮演的角色的含义。

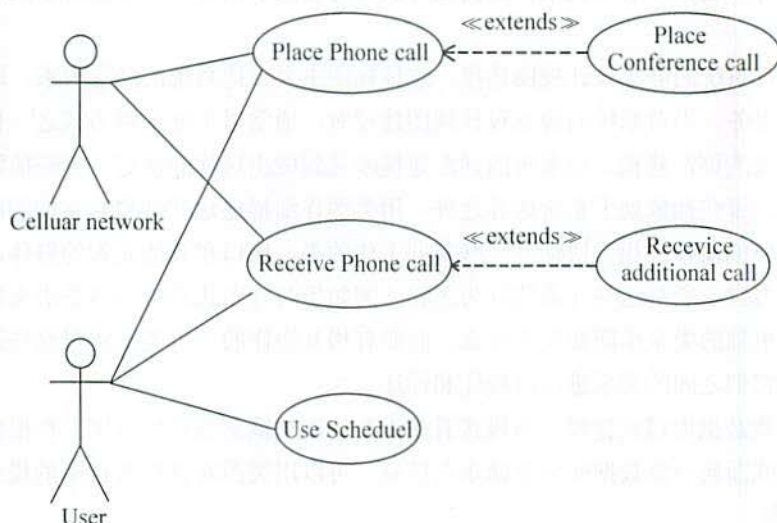


图 10-18 UML 用例图

(2) 对系统的需求建模。对一个系统的需求进行建模,包括说明这个系统应该做什么(从系统外部的一个视点出发),而不考虑系统应该怎样做。在这里,用例图说明了系统想要的行为。通过这种方式,用例图使我们能够把整个系统看作一个黑盒子。你可以观察到系统外部有什么,系统怎样与哪些外部事物相互作用,但却看不到系统内部是如何工作的。

4) 交互图

序列图和协作图均被称为交互图,它们用于对系统的动态方面进行建模。一张交互图显示的是一个交互,由一组对象和它们之间的关系组成,包含它们之间可能传递的消息。顺序图是强调消息时间顺序的交互图;协作图则是强调接收和发送消息的对象的结构组织的交互图。

交互图用于对一个系统的动态方面建模。在多数情况下,它包括对类、接口、构件和节点的具体的或原型化的实例以及它们之间传递的消息进行建模,所有这些都位于一个表达行为的脚本的语境中。交互图可以单独用来可视化、详述、构造和文档化一个特定的对象群体的动态方面,也可以用来对一个用例的特定的控制流进行建模。

交互图一般包含对象、链和消息。

(1) 序列图。序列图(Sequence Diagram)是场景(Scenario)的图形化表示,描述了以时间顺序组织的对象之间的交互活动。如图 10-19 所示,形成序列图时,首先把参加交互的对象放在图的上方,沿 x 轴方向排列。通常把发起交互的对象放在左边,较下级对象依次放在右边。

然后,把这些对象发送和接收的消息沿 y 轴方向按时间顺序从上到下放置。这样,就向提供了控制流随时间推移的清晰的可视化轨迹。

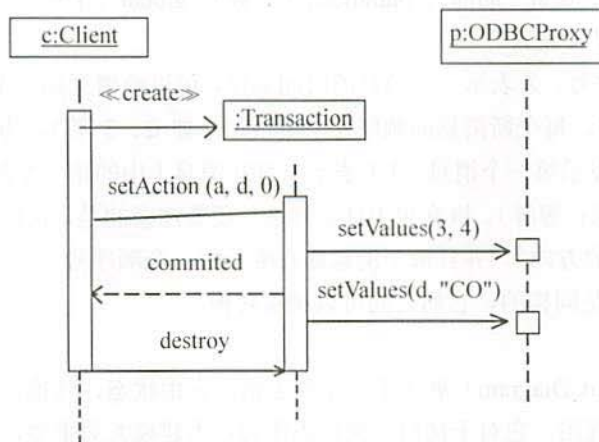


图 10-19 UML 序列图

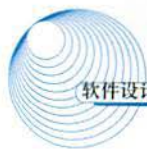
序列图有两个不同于协作图的特征:

① 序列图有对象生命线。对象生命线是一条垂直的虚线,表示一个对象在一段时间内存在。在交互图中出现的大多数对象存在于整个交互过程中,所以,这些对象全都排列在图的顶部,其生命线从图的顶部画到图的底部。但对象也可以在交互过程中创建,它们的生命线从接收到构造型为 `create` 的消息时开始。对象也可以在交互过程中撤销,它们的生命线在接收到构造型为 `destroy` 的消息时结束(并且给出一个大 X 的标记表明生命的结束)。

② 序列图有控制焦点。控制焦点是一个瘦高的矩形,表示一个对象执行一个动作所经历的时间段,既可以是直接执行,也可以是通过下级过程执行。矩形的顶部表示动作的开始,底部表示动作的结束(可以由一个返回消息来标记)。还可以通过将另一个控制焦点放在它的父控制焦点的右边来显示(由循环、自身操作调用或从另一个对象的回调所引起的)控制焦点的嵌套(其嵌套深度可以任意)。如果想特别精确地表示控制焦点在哪里,也可以在对象的方法被实际执行(并且控制还没传给另一个对象)期间,将那段矩形区域阴影化。

(2) 协作图。协作图(Collaboration Diagram)强调收发消息的对象的结构组织。协作图强调参加交互的对象的结构组织。产生一张协作图,首先要将参加交互的对象作为图的顶点。然后,把连接这些对象的链表示为图的弧。最后,用对象发送和接收的消息来修饰这些链。这就提供了在协作对象的结构组织的语境中观察控制流的一个清晰的可视化轨迹。

协作图有两个不同于序列图的特性:



① 协作图有路径。为了指出一个对象如何与另一个对象链接,你可以在链的末端附上一个路径构造型(如构造型 `local`, 表示指定对象对发送者而言是局部的)。通常只需要显式地表示以下几种链的路径: `local` (局部)、`parameter` (参数)、`global` (全局), 以及 `self` (自身), 但不必表示 `association` (关联)。

② 协作图有顺序号。为表示一个消息的时间顺序,可以给消息加一个数字前缀(从1号消息开始),在控制流中,每个新消息的顺序号单调增加(如 2、3 等)。为了显示嵌套,可使用带小数点的号码(1 表示第一个消息;1.1 表示嵌套在消息1中的第一个消息;1.2 表示嵌套在消息1中的第二个消息;等等)。嵌套可为任意深度。还要注意的,沿同一个链可以显示许多消息(可能发自不同的方向),并且每个消息都有唯一的一个顺序号。

序列图和协作图是同构的,它们之间可以相互转换。

5) 状态图

状态图(Statechart Diagram)展现了一个状态机,它由状态、转换、事件和活动组成。状态图关注系统的动态视图,它对于接口、类和协作的行为建模尤为重要,它强调对象行为的事件顺序。

状态图通常包括简单状态和组合状态、转换(事件和动作)。如图 10-20 所示。

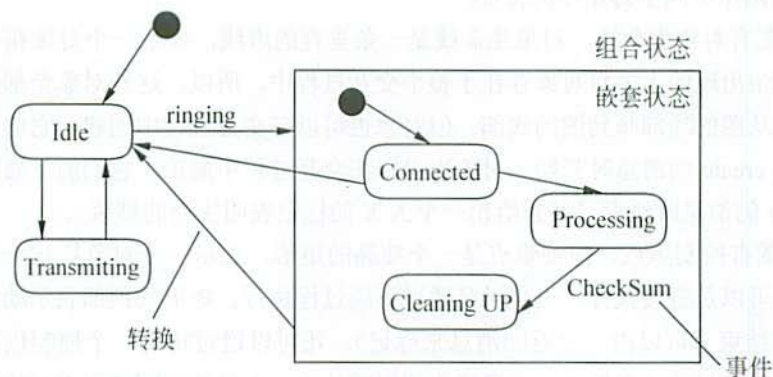


图 10-20 UML 状态图

可以用状态图对系统的动态方面建模。这些动态方面可以包括出现在系统体系结构的任何视图中的任何一种对象的按事件排序的行为,这些对象包括类(各主动类)、接口、构件和节点。

当对系统、类或用例的动态方面建模时,通常是对反应型对象建模。

一个反应型或事件驱动的对象是这样一个人,其行为通常是由对来自语境外部的做

出反应来刻画的。反应型对象在接收到一个事件之前通常处于空闲状态。当它接收到一个事件时,它的反应常常依赖于以前的事件。在这个对象对事件做出反应后,它就又变成闲状态,等待下一个事件。对于这种对象,将着眼于对象的稳定状态、能够触发从状态到状态的转换的事件以及当每个状态改变时所发生的动作。

6) 活动图

活动图(Activity Diagram)是一种特殊的状态图,它展现了在系统内从一个活动到另一个活动的流程。活动图专注于系统的动态视图。它对于系统的功能建模特别重要,并强调对象间的控制流程。

活动图一般包括活动状态和动作状态、转换和对象。

用活动图建模的控制流中,会发生一些事情。你可能要对一个设置属性值或返回一些值的表达式求值。你也可能要调用对象上的操作,发送一个消息给对象,甚至创建或销毁对象,这些可执行的原子计算被称作动作状态,因为它们是该系统的状态,每个原子计算都代表一个动作的执行。动作状态不能被分解。动作状态是原子的,也就是说事件可以发生,但动作状态的工作不能被中断。最后,动作状态的工作所占用的执行时间一般被看作是可忽略的。

活动状态能够进一步被分解,它们的活动由其他的活动图表示。活动状态不是原子的,它们可以被中断。一般来说,还要考虑到它需要花费一段时间来完成。可以把一个动作状态看作一个活动状态的特例。类似地,可以把一个活动状态看作一个组合,它的控制流由其他的活动状态和动作状态组成。

活动图可以表示分支和汇合。当对一个系统的动态方面建模时,通常有两种使用活动图的方式。

(1) 对 workflow 建模。此时所关注的是与系统进行协作的参与者所观察到的活动。workflow 常常位于软件系统的边缘,用于可视化、详述、构造和文档化开发系统所涉及的业务过程。在活动图的这种用法中,对对象流的建模是特别重要的。

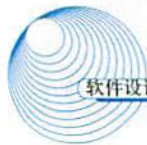
(2) 对操作建模。此时是把活动图作为流程图使用,对一个计算的细节部分建模。在活动图的这种用法中,对分支、分叉和汇合状态的建模是特别重要的。用于这种方式的活动图语境包括该操作的参数和它的局部对象。

7) 构件图

构件图(Component Diagram)展现了一组构件之间的组织和依赖。构件图专注于系统的静态实现视图。它与类图相关,通常把构件映射为一个或多个类、接口或协作。

8) 部署图

部署图(Deployment Diagram)展现了运行处理节点以及其中的构件的配置。部署图给出了体系结构的静态实施视图。它与构件图相关,通常一个节点包含一个或多个构件。



10.5 设计模式

10.5.1 设计模式的要素

“每一个模式描述了一个在我们周围不断重复发生的问题，以及该问题的解决方案的核心。这样，你就能一次又一次地使用该方案而不必做重复劳动。”设计模式的核心在于提供了相关问题的解决方案。

设计模式一般有 4 个要素：

1) 模式名称 (Pattern name)

一个助记名，它用一两个词来描述模式的问题、解决方案和效果。命名一个新的模式增加了设计词汇。设计模式允许我们在较高的抽象层次上进行设计。基于一个模式词汇表，我们就可以讨论模式并在编写文档时使用它们。模式名可以帮助我们思考，便于我们与其他人交流设计思想及设计结果。找到恰当的模式名也是我们设计模式工作的难点之一。

2) 问题 (Problem)

描述了应该在何时使用模式。它解释了设计问题和问题存在的前因后果，它可能描述了特定的设计问题，如怎样用对象表示算法等。也可能描述了导致不灵活设计的类或对象结构。有时候，问题部分会包括使用模式必须满足的一系列先决条件。

3) 解决方案 (Solution)

描述了设计的组成成分，它们之间的相互关系及各自的职责和协作方式。因为模式就像一个模板，可应用于多种不同场合，所以解决方案并不描述一个特定而具体的设计或实现，而是提供设计问题的抽象描述和怎样用一个具有一般意义的元素组合（类或对象组合）来解决这个问题。

4) 效果 (Consequences)

描述了模式应用的效果及使用模式应权衡的问题。尽管我们描述设计决策时，并不总提到模式效果，但它们对于评价设计选择和理解使用模式的代价及好处具有重要意义。软件效果大多关注对时间和空间的衡量，它们也表述了语言和实现问题。因为复用是面向对象设计的要素之一，所以模式效果包括它对系统的灵活性、扩充性或可移植性的影响，显式地列出这些效果对理解和评价这些模式很有帮助。

设计模式确定了所包含的类和实例，它们的角色、协作方式以及职责分配。每一个设计模式都集中于一个特定的面向对象设计问题或设计要点，描述了什么时候使用它，在另一些设计约束条件下是否还能使用，以及使用的效果和如何取舍。按照设计模式的目的可以分为 3 大类，如表 10-6 所示。

表 10-6 设计模式分类

	创建型	结构型	行为型
类	Factory Method	Adapter_Class	Interpreter Template Method Chain of Responsibility Command
对象	Abstract Factory	Adapter_Object Bridge Composite Decorator Facade Flyweight Proxy	Iterator Mediator Memento Observer State Strategy Visitor
	Builder		
	Prototype		
	Singleton		

创建型模式与对象的创建有关；结构型模式处理类或对象的组合；行为型模式对类或对象怎样交互和怎样分配职责进行描述。

10.5.2 创建型设计模式

创建型模式抽象了实例化过程。它们帮助一个系统独立于如何创建、组合和表示它的那些对象。一个类创建型模式使用继承改变被实例化的类，而一个对象创建型模式将实例化委托给另一个对象。

随着系统演化得越来越依赖于对象复合而不是类继承，创建型模式变得更为重要。当这种情况发生时，重心从对一组固定行为的硬编码（hard-coding）转移为定义一个较小的基本行为集，这些行为可以被组合成任意数目的更复杂的行为。这样创建有特定行为的对象要求的不仅仅是实例化一个类。

在这些模式中有两个不断出现的主旋律。第一，它们都将关于该系统使用哪些具体的类的信息封装起来。第二，它们隐藏了这些类的实例是如何被创建和放在一起的。整个系统关于这些对象所知道的是由抽象类所定义的接口。因此，创建型模式在什么被创建，谁创建它，它是怎样被创建的，以及何时创建这些方面给予你很大的灵活性。它们允许你用结构和功能差别很大的“产品”对象配置一个系统。配置可以是静态的（即在编译时指定），也可以是动态的（在运行时）。

【例 10.8】Singleton 模式。

通常情况下，用户可以对应用系统进行配置，并将配置信息保存在配置文件中。应用系统在启动时首先将配置文件加载到内存中，这些内存配置信息应该有且仅有一份。应用单身模式（Singleton）以保证 Configure 类只能有一个实例。这样，Configure 类的使用者无法定义该类的



多个实例, 否则会产生编译错误。

【例 10.8 程序】

```
#include <iostream.h>
class Configure {
protected:
    Configure () {};           // 构造函数
public:
    static Configure* Instance () ;
public:
    int GetConfigureData () { return data; } // 获取配置信息
    int SetConfigureDate (int m_data)
{ data = m_data;    return data; } // 设置配置信息
private:
    static Configure* _instance;
    int data;                // 配置信息
};

Configure::_instance = NULL;

Configure * Configure::Instance () {
    if ( _instance == NULL ) {
        _instance = new Configure () ;
        // 加载配置文件并设置内存配置信息, 此处省略
    }
    return _instance;
}

void main () {
    Configure* t = NULL;
    t = Instance () ;
    int d = t->GetConfigureData () ;
    // 获取配置信息后进行其他工作, 此处省略
}
```

10.5.3 结构型设计模式

结构型模式涉及到如何组合类和对象以获得更大的结构。结构型类模式采用继承机制来组合接口或实现。一个简单的例子是采用多重继承方法将两个以上的类组合成一个类, 结果这个

类包含了所有父类的性质。这一模式尤其有助于多个独立开发的类库协同工作。其中一个例子是类形式的 Adapter 模式。一般来说,适配器使得一个接口与其他接口兼容,从而给出了多个不同接口的统一抽象。为此,类适配器对一个 adaptee 类进行私有继承。这样,适配器就可以用 adaptee 的接口表示它的接口。

结构型对象模式不是对接口和实现进行组合,而是描述了如何对一些对象进行组合,从而实现新功能的一些方法。因为可以在运行时刻改变对象组合关系,所以对象组合方式具有更大的灵活性,而这种机制用静态类组合是不可能实现的。

Composite 模式是结构型对象模式的一个实例。它描述了如何构造一个类层次式结构,这一结构由两种类型的对象所对应的类构成。其中的组合对象可以组合基元对象以及其他的组合对象,从而形成任意复杂的结构。在 Proxy 模式中,proxy 对象作为其他对象的一个方便的替代或占位符。它的使用可以有多种形式。例如它可以在局部空间中代表一个远程地址空间中的对象,也可以表示一个要求被加载的较大的对象,还可以用来保护对敏感对象的访问。Proxy 模式还提供了对对象的一些特有性质的一定程度上的间接访问,从而它可以限制、增强或修改这些性质。

Flyweight 模式为了共享对象定义了一个结构。至少有两个原因要求对象共享:效率和一致性。Flyweight 的对象共享机制主要强调对象的空间效率。使用很多对象的应用必须考虑每一个对象的开销。使用对象共享而不是进行对象复制,可以节省大量的空间资源。但是仅当这些对象没有定义与上下文相关的状态时,它们才可以被共享。Flyweight 的对象没有这样的状态。任何执行任务时需要的其他一些信息仅当需要时才传递过去。由于不存在与上下文相关的状态,因此 Flyweight 对象可以被自由地共享。

如果说 Flyweight 模式说明了如何生成很多较小的对象,那么 Facade 模式则描述了如何用单个对象表示整个子系统。模式中的 facade 用来表示一组对象, facade 的职责是将消息转发给它所表示的对象。Bridge 模式将对象的抽象和其实现分离,从而可以独立地改变它们。

Decorator 模式描述了如何动态地为对象添加职责。Decorator 模式是一种结构型模式。这一模式采用递归方式组合对象,允许添加任意多的对象职责。例如,一个包含用户界面组件的 decorator 对象可以将边框或阴影这样的装饰添加到该组件中,或者可以将窗口滚动和缩放这样的功能添加到组件中。我们可以将一个 decorator 对象嵌套在另外一个对象中就可以很简单地增加两个装饰,添加其他的装饰也是如此。因此,每个 decorator 对象必须与其组件的接口兼容并且保证将消息传递给它。Decorator 模式在转发一条信息之前或之后都可以完成它的工作(比如绘制组件的边框)。许多结构型模式在某种程度上具有相关性。

10.5.4 行为设计模式

行为模式涉及到算法和对象间职责的分配。行为模式不仅描述对象或类的模式,还描述它



们之间的通信模式。这些模式刻画了在运行时难以跟踪的复杂的控制流。它们将你的注意力从控制流转移到对象间的联系方式上来。

行为类模式使用继承机制在类间分派行为。本章包括两个这样的模式。其中 `TemplateMethod` 较为简单和常用。模板方法是一个算法的抽象定义,它逐步地定义该算法,每一步调用一个抽象操作或一个原语操作,子类定义抽象操作以具体实现该算法。另一种行为类模式是 `Interpreter`。它将一个文法表示为一个类层次,并实现一个解释器作为这些类的实例上的一个操作。

行为对象模式使用对象复合而不是继承。一些行为对象模式描述了一组对等的对象怎样相互协作以完成其中任一个对象都无法单独完成的任务。这里一个重要的问题是对等的对象如何互相了解对方。对等对象可以保持显式的对对方的引用,但那会增加它们的耦合度。在极端情况下,每一个对象都要了解所有其他的对象。`Mediator` 在对等对象间引入一个 `mediator` 对象以避免这种情况的出现。`mediator` 提供了松耦合所需的间接性。

`Chain of Responsibility` 提供更松的耦合。它让你通过一条候选对象链隐式的向一个对象发送请求。根据运行时刻情况任一候选者都可以响应相应的请求。候选者的数目是任意的,你可以在运行时刻决定哪些候选者参与到链中。

`Observer` 模式定义并保持对象间的依赖关系。典型的 `Observer` 的例子是 `Smalltalk` 中的模型/视图/控制器,其中一旦模型的状态发生变化,模型的所有视图都会得到通知。

其他的行为对象模式常将行为封装在一个对象中并将请求指派给它。`Strategy` 模式将算法封装在对象中,这样可以方便地指定和改变一个对象所使用的算法。`Command` 模式将请求封装在对象中,这样它就可作为参数来传递,也可以被存储在历史列表里,或者以其他方式使用。`State` 模式封装一个对象的状态,使得当这个对象的状态对象变化时,该对象可改变它的行为。`Visitor` 封装分布于多个类之间的行为,而 `Iterator` 则抽象了访问和遍历一个集合中的对象的方式。

【例 10.9】Observer 模式。

在一公文处理系统中,开发者定义了一个公文类 `OfficeDoc`,其中定义了公文具有的属性和处理公文的相应方法。当公文的内容或状态发生变化时,关注此 `OfficeDoc` 类对象的相应的 `DocExplorer` 对象都要更新其自身的状态。一个 `OfficeDoc` 对象能够关联一组 `DocExplorer` 对象。当 `OfficeDoc` 对象的内容或状态发生变化时,所有与之相关联的 `DocExplorer` 对象都将得到通知,这种应用被称为观察者模式。

【例10.9程序】

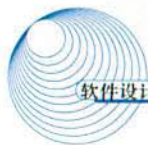
```
#include <iostream>
const OBS_MAXNUM=20; //最多与 OfficeDoc 对象相关联的 DocExplorer 对象的个数
class OfficeDoc;
```

```

class DocExplorer{ //关注 OfficeDoc 公文对象的类
public:
    DocExplorer (OfficeDoc *doc); //构造函数
    virtual void update (OfficeDoc *doc) =0; //更新自身状态的函数
    //其他相关属性和方法省略
};

class OfficeDoc{ //公文类
private:
    DocExplorer *myObs[OBS_MAXNUM];
    //关注此公文类的 DocExplorer 类对象指针数组
    int index; //与 OfficeDoc 对象关联的 DocExplorer 对象的个数
public:
    OfficeDoc () {
        index=0;
    }
    void attach (DocExplorer *o) {
        //将一 DocExplorer 对象与 OfficeDoc 对象相关联
        if (index >= OBS_MAXNUM || o == NULL) return;
        for (int loop = 0; loop < index; loop++)
            if (myObs[loop] == o) return;
        myObs[index] = o;
        index++;
    }
    void detach (DocExplorer *o) {
        //解除某 DocExplorer 对象与 OfficeDoc 对象的关联
        if (o == NULL) return;
        for (int loop = 0; loop < index; loop++) {
            if (myObs[loop] == o) {
                if (loop <= index-2) myObs[loop] = myObs[index-1];
                myObs[index-1] = NULL;
                index--;
                break;
            }
        }
    }
private:
    void notifyObs () { //通知所有的 DocExplorer 对象更改自身状态

```

```
        for (int loop = 0; loop < index; loop++) {  
            myObs[loop]-> update (this); //DocExplorer 对象更新自身状态  
        }  
    }  
};  
//其他公文类的相关属性和方法  
};  
DocExplorer::DocExplorer (OfficeDoc *doc) { //DocExplorer 类对象的构造函数  
    doc-> attach (this); //将此 DocExplorer 对象与 doc 对象相关联  
}
```

第 11 章 标准化基础知识

11.1 标准化的基本概念

11.1.1 标准、标准化的概念

标准(Standard)是对重复性事物和概念所做的统一规定。标准以科学、技术和实践经验的综合成果为基础,以获得最佳秩序和促进最佳社会效益为目的,经有关方面协商一致,由主管或公认机构批准,并以规则、指南或特性的文件形式发布,作为共同遵守的准则和依据。规范(Specification)、规程(Code)都是标准的一种形式。

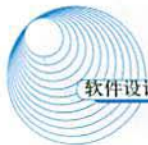
标准化(Standardization)是在经济、技术、科学及管理社会实践中,以改进产品、过程和服务的适用性,防止贸易壁垒,促进技术合作,促进最大社会效益为目的,对重复性事物和概念通过制定、发布和实施标准,达到统一,获最佳秩序和社会效益的过程。

11.1.2 标准化的范围 and 对象

标准化涉及的范围包括生产、经济、技术、科学及管理社会实践中具有的重复性事物和概念以及需要建立统一技术要求的各个领域。在这些领域中,凡具有多次重复使用和需要制定标准的具体产品,以及各种定额、规划、要求、方法、概念等,都可称作为标准化的对象。如产品的品种、规格、型式;产品的性能、质量;包装的尺寸、材质、样式;信息表示、信息处理技术;试验方法、测试方法、检验方法、操作方法、抽样方法等各类方法;量值、单位、术语、符号、代码、标志;日常操作、安全操作、劳动保护、安全卫生、保护环境等方面的规范、规程或规定;市场调查、研制开发、物资采购、加工工艺、质量控制、质量检验、销售、售后服务等各阶段的管理事项,等等。

标准化对象一般可分为两大类:一类是标准化的具体对象,即需要制定标准的具体事物;另一类是标准化总体对象,即各种具体对象的全体所构成的整体,通过它可以研究各种具体对象的共同属性、本质和普遍规律。

对一个企业来说,企业的经济活动、技术活动、科研活动和管理活动的全过程及其要素都可作为标准化的范围和对象。企业的活动及其要素具有重复出现的特性,例如,同一产品的反复投入生产、同一检验方法的反复多次进行、同一概念的多次使用、同一类技术活动(如产品设计)同时或相继发生、同一管理事项的重复进行等。这些事物和概念的多次重复活动便产生



了按统一标准进行的客观需要和要求,制定标准可以总结以往的经验,选择最佳方案,作为今后实践的目标和依据。如果同一事物和概念在反复多次进行时没有统一的标准可遵循,就失去了共同的客观依据,企业活动就会因失去准绳而产生混乱。如果不善于利用标准化这个有效工具,每次遇到问题都从头研究、讨论、做出决定,这样不仅浪费时间和精力,影响效率,而且常常丧失机遇。如果对重复进行的活动进行标准化实施,遇到同类问题照标准执行,就可避免因人事变动等因素造成同一问题前后处理不一致而带来其他的问题。这样,领导者、技术人员或管理人员可以把主要精力放在研究和处理企业根本性的、方向性的大问题或新问题上,而由标准化的实施保证企业各项工作的正常进行,使企业活动纳入高效率的轨道。

11.1.3 标准化的实质

标准化的实质是通过制定、发布和实施标准,达到统一。统一的目的是为了保证事物发展所必需的秩序和效率,对事物的形成、功能或其他特性,确定适合于一定时期和一定条件的一致规范,并使这种一致规范与其取代的对象在功能上达到等效。统一是标准的本质特征,任何标准,都是在一定条件下的“统一规定”。统一的基本含义一般包括以下要点:

(1) 对生产、使用、科研、管理等有关方面进行认真讨论,通过充分协商达成一致的意见,共同认可后,经过批准实施。这样制定的标准具有科学性和民主性,具有突出的科学性和民主性的标准,在实施中将具有权威性。

(2) 为了经济而有效地满足需要,对处于自然状态的标准化对象的结构、形式、规格或其他性能进行筛选提炼,剔除其中多余的、低效的、可替换的环节,合理精简并确定出满足要求所必需的高效环节,保持整体构成精简而合理,使其功能和效率(满足全面需要的能力)达到最佳。

(3) 对标准的水平、质量指标等各项内容确定的一致规范,一般是反映一定时期的水平,因此经统一而确立的一致性仅适用于一定时期和一定条件。随着时间的推移和条件的改变,还需确立新的更高水平的一致性,原先的统一就要由新的统一所代替,以保持标准的先进性和合理性。

(4) 不同层次的标准是在不同范围内进行统一的,不同类型的标准则是从不同角度(或侧面)进行统一。同一功能的同一对象在同一范围、同一标准化层次上只能选择确定一个统一的规范(包含被取代对象所具备的必要功能),而不能制定出两个或多个规范。

(5) 统一并不意味着绝对统一,即全都统一到只有一种。统一不排斥多样性,有时只限定一个范围,有时则规定几种情况。统一并不限制技术发展,它也有最大自由度原则,例如技术参数要选择要为技术发展留有空间。在同一标准中,统一规定的内容,可根据需要是一种或多种;又如产品的分等分级,产品的参数系列,等等。

(6) 标准是科学、技术和实践经验的结晶,其统一的内容和基础是科学、技术和实践经验

的综合成果。统一需要建立在科学试验或验证的数据基础上,通过总结、分析并综合国内外有关科学、技术和实践经验的成果以及多年积累的经验,并使之科学化和条理化;标准需要吸收新的科研成果、新的技术成就和新的实践经验,以便通过标准的形式去推广新成果、新技术和新经验。

11.1.4 标准化的目的

标准化的目的之一是建立最佳秩序,即建立一定环境和一定条件的最合理秩序。通过标准化在社会生产组成部分之间进行协调,确立共同遵循的准则,建立稳定和最佳的生产、技术、安全、管理等秩序,使生产活动和经营管理活动井然有序,避免混乱,达到高效率。标准化的另一目的就是获得最佳效益。一定范围的标准,是按一定范围内的技术效益和经济效果的目标制定出来的,它不仅考虑了标准在技术上的先进性,还考虑到经济上的合理性以及企业的最佳经济效益。标准虽然不是商品,但它却能加速商品的生产和流通,能极其显著地提高劳动生产率和资源的转化效率,实现商品生产的合理化、高效率化和低成本,给企业带来客观的利润。一些工业发达国家把标准化作为企业经营管理、获取利润、进行竞争的法宝和秘密武器。特别是一些著名公司,往往都建立标准化体系,以保证其利润和竞争目标的实现。随着商品经济的发展和市场的扩大,特别是专业化生产和国际协作的出现,为标准的应用创造了广阔的空间,并为一体化的世界经济准备了一套保证公平贸易的准则。

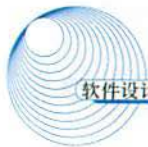
11.2 标准化过程模式

标准是标准化活动的产物,其目的和作用都是通过制定和贯彻具体的标准来体现的。标准化不是一个孤立的事物,而是一个活动过程。一个过程可分为几个子过程,其最后一个子过程,是一个过程的终结,也是下一个过程的开始。通过总结前一个过程的经验和教训,并依据客观环境的新变化和新要求,提出标准修订的新目标。这是一个不断循环、螺旋式上升的运动过程。每完成一个循环,标准的水平将提高一步。

标准化活动过程一般包括标准产生(调查、研究、形成草案、批准发布)子过程;标准实施(宣传、普及、监督、咨询)子过程和标准更新(复审、废止或修订)子过程等。

11.2.1 标准的制定

制定标准的过程,实际上就是总结和积累人类社会实践经验的过程,每一个新标准的产生,都标志着某一领域或某项活动的经验被规范化。标准的产生一般包括调查研究、制定计划(立项)、起草标准、征求意见、审查、批准发布等标准生成阶段。这个过程中产生的问题,有程序性的问题(如征求意见、审查等);更有实质性的问题,即标准的适用性、可行性、先进性



方面的问题。这个过程随标准化对象的不同而不同。起草标准时一般需要吸收新的科研成果、新的技术成就和新的实践经验,以便通过标准的形式去推广新成果、新技术和新经验。为保证标准的编写质量,便于资料管理,体现其严肃性,标准文件有统一的格式。标准从制定到批准发布的一整套工作程序和审批制度,体现了标准产生的科学规律,表现出其法律特性。ISO 和 IEC 是两个国际标准化组织,为规范国际标准的产生过程,发布了导则性文件。我国依据该导则性文件制定了相应的国家标准,以规范国家标准的制定程序。

11.2.2 标准的实施

标准的实施过程,实际上就是推广和普及已被规范化的实践经验的过程。标准的实施过程包括哪些活动内容,没有统一的规定,一般有标准的宣传、贯彻执行和监督检查等。标准化活动是一项有组织的活动,并且始终是一个组织的行为。通常,国家、区域和行业的标准化管理组织以及标准化团体,通过宏观管理(标准规划、计划和协调);健全以国际标准、区域性标准、国家标准、行业标准为主的标准文献资料;建立健全行业和企业标准备案管理制度,提高生产、经销和服务单位执行标准的自觉性。我国强制性标准的实施是通过强制性的监督检查来推动,并依法开展标准的实施与监督。对于推荐性标准的实施,尚无明确有效的措施,需要建立和完善社会主义市场经济体制下的技术法规。《标准化法》、《国家标准管理办法》等法规,规定了我国标准化工作的方针、政策、任务和标准化体制等。它们是我国推行标准化,实施标准化管理和监督的重要依据。企业根据企业和市场的需求及技术发展趋势的需要,开展标准立项的前期研究,为标准制定提供可靠的依据;开展标准化审查,制止不符合强制性标准规定、不符合标准化技术政策的技术和产品;确定重点标准的实施与监督的项目计划,对各个环节进行全面动态跟踪管理。

11.2.3 标准的更新

已经实现了标准化的事物,实施一段时间后,有可能突破原先的规定,有新的需求,使某些环节的标准失去意义,需要对它再制定标准。标准的更新是实践经验的深化和提高的过程。通过信息反馈总结经验和问题,依据客观环境的新变化和新要求,提出标准修订的新目标,更新标准。人类社会实践是一个永不止息的活动,标准化也是一个永无止境的过程。

1. 标准复审

已经发布实施的现有标准(包括已确认或修改补充的标准),经过一段时间后,需要对其内容再次审查,以确保其有效性、先进性和适用性。自标准实施之日起至标准复审重新确认、修订或废止的时间,称为标准的有效期(也称为标龄)。由于各个国家的情况不同,标准有效期也不同。例如,ISO 标准每5年复审一次,平均标龄为4.92年;1988年发布的《中华人民共

和国标准化法实施条例》中规定,标准实施后的复审周期一般不超过5年,即我国的国家标准有效期一般为5年。标准复审工作由各主管部门或标准化技术委员会组织进行,对需要复审的标准要收集实施中的问题并进行分类整理。复审时可采用会议审查或函审,经复审的标准要用书面形式报告复审结果,如复审简况、处理意见、复审结论和依据等,以确认现行标准是继续有效、需要修订或者废止,确保标准的时效性。

2. 标准确认

经复审后的标准,若其内容仍符合当前科学技术水平并适合经济建设的需要,无需修改或只需作编辑性修改,可确认为继续有效。经确认的标准,在发布时,不改变标准的顺序号和年号。当标准再版时在标准封面写明××××年确认字样。

经过复审后的标准,若其内容需完善和补充,只要对标准条文、图、表做少量修改补充,仍可继续使用的,则采用标准修改单的形式,对需要修改补充的内容予以完善后,按照标准的制定程序,经标准化主管单位批准发布。

3. 标准修订

经复审后的标准,若其内容需要做较大修改才能适应生产和使用的需要以及科学技术发展的需要,则应作为修订项目。标准修订的程序按制定标准的程序执行。修订后的标准顺序号不变,年号改为重新修订发布时的年号。

11.3 标准的分类

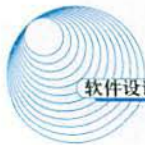
标准化工作是一项复杂的系统工程,标准为适应不同的要求从而构成一个庞大而复杂的系统,为便于研究和应用的目的,可以从不同的角度和属性将标准进行分类。

11.3.1 根据适用范围分类

根据标准制定的机构和标准适用的范围,可分为国际标准、国家标准、行业标准、企业(机构)标准及项目(课题)标准。

1. 国际标准

国际标准(International Standard)是指国际标准化组织(ISO)、国际电工委员会(IEC)所制定的标准,以及ISO出版的《国际标准题内关键词索引(KWIC Index)》中收录的其他国际组织制定的标准。国际标准在世界范围内统一使用,各国可以自愿采用,不强制使用,但由于国际标准集中了一些先进工业国家的技术经验,各个国家为外贸上的利益和本国利益考虑往往都积极采用国际标准。



2. 国家标准

国家标准(National Standard)是由政府或国家级的机构制定或批准的、适用于全国范围的标准,是一个国家的标准体系的主体和基础,国内各级标准必须服从且不得与之相抵触。常见的国家标准有:

(1) 中华人民共和国国家标准 GB 是我国最高标准化机构中华人民共和国国家技术监督局所公布实施的标准简称为“国标”。

(2) 美国国家标准 ANSI 是美国国家标准协会 ANSI (American National standards institute) 制定的标准。

(3) 英国国家标准 BS (British Standard) 是英国标准学会 (BSI) 制定的标准。

(4) 日本工业标准 JIS (Japanese Industrial Standard) 是日本工业标准调查会 (JISC) 制定的标准。

3. 区域标准

区域标准 (Regional Standard) (也称地区标准) 泛指世界上按地理、经济或政治划分的某一区域标准化团体所通过的标准。它是为了某一区域的利益建立的标准。通常,地区标准主要是指太平洋地区标准会议 (PASC)、欧洲标准化委员会 (CEN)、亚洲标准咨询委员会 (ASAC)、非洲地区标准化组织 (ARSO) 等地区组织所制定和使用的标准。

4. 行业标准

行业标准 (Specialized Standard) 由行业机构、学术团体或国防机构制定,并适用于某个业务领域的标准,如:

(1) 美国电气和电子工程师学会标准 IEEE, IEEE 通过的标准常常要报请 ANSI 审批,使其具有国家标准的性质。因此,IEEE 公布的标准常冠有 ANSI 字头。例如,ANSI/IEEE Str 828—1983 (软件配置管理计划标准)。

(2) 中华人民共和国国家军用标准 GJB, 这是由我国国防科学技术工业委员会批准,适用于国防部门和军队使用的标准。例如,1988 年发布实施的 GJB 473—88 (军用软件开发规范)。我国各主管部、委(局)批准发布,在其范围内统一使用的标准。

(3) 美国国防部标准 DOD-STD (Department Of Defense-Standards), 适用于美国国防部门。美国军用标准 MIL-S (Military-Standards), 适用于美军内部。

5. 企业标准

企业标准 (Company Standard) 由企业或公司批准、发布的标准,某些产品标准由其上级

主管机构批准、发布。

例如，美国 IBM 公司通用产品部（General Products Division）1984 年制定的“程序设计开发指南”，仅供该公司内部使用。

6. 项目规范

项目规范（Project Specification）由某一科研生产项目组织制定，且为该项任务专用的软件工程规范。例如，计算机集成制造系统（CIMS）的软件工程规范。

7. 我国标准分类

根据《中华人民共和国标准化法》的规定，我国标准分为国家标准、行业标准、地方标准和企业标准等 4 类。这 4 类标准主要是适用的范围不同，不是标准技术水平高低的分级。

（1）国家标准：由国务院标准化行政主管部门制定的需要全国范围内统一的技术要求，称为国家标准。

（2）行业标准：没有国家标准而又需在全国某个行业范围内统一的技术标准，由国务院有关行政主管部门制定并报国务院标准化行政主管部门备案的标准，称为行业标准。

（3）地方标准：没有国家标准和行业标准而又需在省、自治区、直辖市范围内统一的工业产品的安全、卫生要求，由省、自治区、直辖市标准化行政主管部门制定并报国务院标准化行政主管部门和国务院有关行业行政主管部门备案的标准，称为地方标准。

（4）企业标准：企业生产的产品没有国家标准、行业标准和地方标准，由企业自行组织制定、作为组织生产依据的相应标准，或者在企业内制定适用的、比国家标准、行业标准或地方标准更严格的企业（内控）标准，并按省、自治区、直辖市人民政府的规定备案的标准（不含内控标准），称为企业标准。

11.3.2 根据标准的性质分类

根据标准的性质可分为技术标准、管理标准、工作标准。

1. 技术标准

技术标准（Technique Standard）是针对重复性的技术事项而制定的标准，是从事生产、建设及商品流通时需要共同遵守的一种技术依据。按其标准化对象特征和作用，可分为基础标准、产品标准、方法标准、安全卫生与环境保护标准等；按其标准化对象在生产流程中的作用，可为零部件标准、工装标准、设备维修保养标准及检查标准等；按标准的强制程度，可分为强制性与推荐性标准；按标准在企业中的适用范围，又可分为公司标准、工厂标准和科室标准等。



2. 管理标准

管理标准 (Administrative Standard) 是管理机构为行使其管理职能而制定的具有特定管理功能的标准, 主要用于规定人们在生产活动和社会实践中的组织结构、职责权限、过程方法、程序文件、资源分配以及方针、目标、措施、影响管理的因素等事宜, 它是合理组织国民经济, 正确处理各种生产关系, 正确实现合理分配, 提高生产效率和效益的依据。在实际工作中通常按照标准所起的作用不同, 将管理标准分为技术管理标准、生产组织标准、经济管理标准、行政管理标准、业务管理标准和工作标准等。

3. 工作标准

工作标准 (Work Standard) 是为协调整个工作过程, 提高工作质量和效率, 针对具体岗位的工作制定的标准; 是对工作的内容、方法、程序和质量要求所制定的标准。工作标准的内容包括: 各岗位的职责和任务、每项任务的数量、质量要求及完成期限, 完成各项任务的程序和方法, 与相关岗位的协调、信息传递方式, 工作人员的考核与奖罚方法等。对生产和业务处理的先后顺序、内容和要达到的要求所做的规定称为工作程序标准。工作程序标准是工作标准的一种, 其目的是使各项工作条理化、标准化和规范化, 以求得最佳工作秩序、工作质量和工作效率。以管理工作为对象所制定的标准, 称为管理工作标准。管理工作标准的内容主要包括: 工作范围、内容和要求, 与相关工作的关系、工作条件、工作人员的职权与必备条件, 工作人员的考核、评价及奖惩办法等。

11.3.3 根据标准化的对象和作用分类

根据标准的对象和作用, 标准可分为基础标准、产品标准、方法标准、安全标准、卫生标准、环境保护标准、服务标准等。

1. 基础标准

基础标准 (Basic Standard) 是在一定范围内作为其他标准的基础并普遍通用, 具有广泛指导意义的标准。如名词、术语、符号、代号、标识、方法、模数、公差与配合、基本参数系列、产品系列型号、产品环境条件、可靠性要求等。一般, 在某领域中基础标准是覆盖面最大的标准, 也是该领域中所有标准的共同基础。

2. 产品标准

产品标准 (Product Standard) 是为保证产品的适用性, 对产品必须达到的某些或全部特性要求所制定的标准。产品标准是一定时期和一定范围内具有约束力的产品技术准则; 是产品生

产、质量检验、选购验收、使用维护和洽谈贸易的技术依据。产品标准的主要内容包括：产品的适用范围；产品的品种、规格和结构形式；产品的主要性能；产品的试验；检验方法和验收规则；产品的包装、储存和运输等方面的要求。产品标准可分为完全的产品标准和不完全的产品标准（品种标准）：完全的产品标准是规定产品术语、符号、技术性能、试验方法、检验、包装、储运等全部要求的标准。不完全的产品标准中只规定品种、规格、型式与尺寸、参数系列等。

3. 方法标准

方法标准（Method Standard）是以各种方法为对象而制定的标准。方法标准一般包括两类：一类以试验、检查、分析、抽样、统计、计算、测定、作业等方法为对象制定的标准。例如试验方法、检查方法、分析方法、测定方法、抽样方法、设计规范、计算方法、工艺规程、作业指导书、生产方法、操作方法及包装、运输方法等。另一类是为合理生产优质产品，并在生产、作业、试验、业务处理等方面为提高效率而制定的标准。

4. 安全标准

安全标准（Safety Standard）是以保护人的安全或物品的安全为对象和目的而制定的标准。安全标准一般有两种形式：一种为专门目的安全标准；另一种是在产品标准或工艺标准中列出有关安全的要求和指标。例如，安全标准可包括劳动安全标准、锅炉和压力容器安全标准、电气安全标准和消费品安全标准等。安全标准一般均为强制性标准，由国家通过法律或法令形式规定强制执行。

5. 卫生标准

卫生标准（Hygienic Standard）是为保护人的健康，对食品、医药及其他方面的卫生要求而制定的标准。卫生标准是专门以卫生（工业卫生、劳动卫生等）要求为对象、目的制定的标准，如食品卫生标准，规定了食品中有害物质或病菌的限量。

6. 环境保护标准

环境标准（Environment Protection Standard）是为保护环境不受污染和有利于生态平衡，对大气、水体、土壤、噪声、振动、电磁波等环境质量、污染管理、监测方法及其他事项而制定的标准。例如，水质标准、噪声标准等。

7. 服务标准

服务标准（Service Standard）是为提高服务质量，对某项服务工作要达到的要求所制定的



标准。服务包括技术服务即产品售后服务,也包括第三产业即旅游、商业、交通运输、金融、电信、信息、保险、医疗等服务行业。

11.3.4 根据法律的约束性分类

根据标准的法律约束性,可分为强制性标准、推荐性标准。

1. 强制性标准

根据《标准化法》的规定,企业和有关部门对涉及其经营、生产、服务、管理有关的强制性标准都必须严格执行,任何单位和个人不得擅自更改或降低标准。对违反强制性标准而造成不良后果以至重大事故者,由法律、行政法规规定的行政主管部门依法根据情节轻重给予行政处罚,直至由司法机关追究刑事责任。

强制性标准是国家技术法规的重要组成,它符合世界贸易组织贸易技术壁垒协定关于“技术法规”定义,即“强制执行的规定产品特性或相应加工方法的包括可适用的行政管理规定在内的文件。技术法规也可包括或专门规定用于产品、加工或生产方法的术语、符号、包装标志或标签要求”,为使我国强制性标准与 WTO/TBT 规定衔接,其范围限制在国家安全、防止欺诈行为、保护人身健康与安全、保护动物植物的生命和健康以及保护环境等方面。

2. 推荐性标准

在生产、交换、使用等方面,通过经济手段或市场调节而自愿采用的一类标准称为推荐性标准。这类标准,不具有强制性,任何单位均有权决定是否采用,违犯这类标准,不构成经济或法律方面的责任。应当指出的是,推荐性标准一经接受并采用,或由各方商定后同意纳入经济合同中,就成为各方必须共同遵守的技术依据,具有法律上的约束性。

推荐性标准是指导性标准,由公认机构批准的,非强制性的,为了通用或反复使用的目的,为产品或相关生产方法提供规则、指南或特性的文件。标准也可以包括或专门规定用于产品、加工或生产方法的术语、符号、包装标准或标签要求。由于推荐性标准是协调一致的文件,不受政府和社会团体的利益干预,能更科学地规定特性或指导生产,《标准化法》鼓励企业积极采用推荐性标准。为了防止企业利用标准欺诈消费者,要求采用低于推荐性标准的企业标准组织生产的企业向消费者明示其产品标准水平。

11.4 标准的代号和编号

1. 国际标准 ISO 的代号和编号

国际标准 ISO 的代号和编号的格式为:ISO+标准号+[杠+分标准号]+冒号+发布年号(方括

号中的内容可有可无), 例如, ISO 8402: 1987 和 ISO 9000-1: 1994, 是 ISO 标准的代号和编号。

2. 国家标准的代号和编号

我国国家标准的代号由大写汉字拼音字母构成, 强制性国家标准代号为 GB, 推荐性国家标准的代号为 GB/T。

国家标准的编号由国家标准的代号、标准发布顺序号和标准发布年代号(四位数组成):

(1) 强制性国家标准: GB ××××× — ××××。

(2) 推荐性国家标准: GB/T ××××× — ××××。

3. 行业标准的代号和编号

(1) 行业标准代号: 行业标准代号由汉字拼音大写字母组成, 由国务院各有关行政主管部门提出其所管理的行业标准范围的申请报告, 国务院标准化行政主管部门审查确定并正式公布该行业标准代号。已正式公布的行业代号有: QJ (航天)、SJ (电子)、JB (机械)、JR (金融系统) 等。

(2) 行业标准的编号: 行业标准的编号由行业标准代号、标准发布顺序及标准发布年代号(四位数)组成, 表示方法如下:

- 强制性行业标准编号: ×× ×××× — ××××;

- 推荐性行业标准编号: ××/T ×××× — ××××。

4. 地方标准的代号和编号

(1) 地方标准的代号: 地方标准代号由大写汉字拼音 DB 加上省、自治区、直辖市行政区划代码的前两位数字(如北京市 11、天津市 12、上海市 31 等), 再加上斜线 T 组成推荐性地方标准; 不加斜线 T 为强制性地方标准, 表示方法如下:

- 强制性地方标准: DB××;

- 推荐性地方标准: DB××/T。

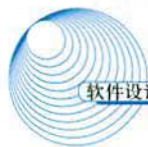
(2) 地方标准的编号: 地方标准的编号由地方标准代号、地方标准发布顺序号、标准发布年代号(四位数)3 部分组成, 表示方法如下:

- 强制性地方标准: DB×× ××× — ××××;

- 推荐性地方标准: DB××/T ××× — ××××。

5. 企业标准的代号和编号

(1) 企业标准的代号: 企业标准的代号由汉字大写拼音字母 Q 加斜线再加企业代号组成, 企业代号可用大写拼音字母或阿拉数字或两者兼用所组成。企业代号按中央所属企业和地方企



业分别由国务院有关行政主管部门或省、自治区、直辖市政府标准化行政主管部门会同同级有关行政主管部门加以规定。例如:Q/×××。企业标准一经制定颁布,即对整个企业具有约束性,是企业法规性文件,没有强制性企业标准和推荐企业标准之分。

(2) 企业标准的编号:企业标准的编号由企业标准代号,标准发布顺序号和标准发布年代号(四位数)组成,表示方法:Q/××× ××××—××××。

11.5 国际标准和国外先进标准

由于国际贸易广泛开展,产品在国际市场上的竞争越来越激烈,要求产品不但要具有高的质量,好的性能,还要具有广泛的通用性、互换性。这就需要采用统一的、先进的标准,按照国际上统一的标准生产,避免采用的标准不一致,带来国际贸易障碍。国际标准和国外先进标准集中了一些先进工业国家的技术经验,加之各国考虑外贸上的利益,世界各国都积极采用国际标准或先进的标准。许多国家直接把国际标准作为本国标准使用。采用国际标准和国外先进标准是我国一项重大技术经济政策,是促进技术进步,提高产品质量,扩大对外开放,加快与国际惯例接轨,也是发展社会主义市场经济的一项重要措施。

11.5.1 国际标准

国际标准是指国际标准化组织(ISO)、国际电工委员会(IEC)所制定的标准,以及 ISO 出版的《国际标准题内关键词索引(KWIC Index)》中收录的其他国际组织制定的标准。1983 年 3 月出版的《KWIC 索引(第一版)》中共收录了 24 个国际组织制定的 7600 个标准,其中 ISO 标准占 68%,IEC 标准占 18.5%,其他 22 个国际组织的标准共 968 个,占 13.5%。1989 年《KWIC 索引(第二版)》,共收录了 ISO 与 IEC 制定的 800 个标准,以及其他 27 个国际组织的 1200 多条标准。ISO 推荐列入 KWIC 索引的 27 个国际组织。一些未列入 KWIC Index 的国际组织所制定的某些标准也被国际公认。这 27 个国际组织制定的标准化文献主要有国际标准、国际建议、国际公约、国际公约的技术附录和国际代码,也有经各国政府认可的强制性要求。对国际贸易业务服务和信息交流具有重要影响。

11.5.2 国外先进标准

国外先进标准是指国际上有权威的区域性标准、世界上经济发达国家的国家标准和通行的团体标准以及包括知名企业标准在内的其他国际上公认的先进标准。主要有:

(1) 国际上有权威的区域性标准,如欧洲标准化委员会(CEN)、欧洲电工标准化委员会(CENELEC)、欧洲广播联盟(EBU)、亚洲大洋洲开放系统互联研讨会(AOW)、亚洲电子数据交换理事会(ASEB)等制定的标准。

(2) 世界经济技术发达国家的国家标准,如美国国家标准(ANSI)、德国国家标准(DIN)、英国国家标准(BS)、日本国工业标准(JIS)、瑞典国家标准(SIS)、法国国家标准(NF)、瑞士国家标准(SNV)、意大利国家标准(UNI)、俄罗斯国家标准(ТОСТ)等。

(3) 国际公认的行业性团体标准,如美国材料与实验协会标准(ASTM)、美国石油学会标准(API)、美国军用标准(MIL)、美国电气制造商协会标准(NEMA)、美国电影电视工程师协会标准(SMPTE)、美国机械工程师协会标准(ASME)、英国石油学会(IP)等。

(4) 国际公认的先进企业标准,如美国IBM公司、美国HP公司、芬兰诺基亚公司、瑞士钟表公司等企业标准等。

11.5.3 采用国际标准和国外先进标准

采用是指采纳和应用。采用国际标准和国外先进标准是把国际标准和国外先进标准或其内容,通过分析研究,不同程度地订入(编入)我国标准,并贯彻执行。将国际标准和国外先进标准订入国家标准的主要方法有:

(1) 认可法:由国家标准机构直接宣布某项国际标准为国家标准,其具体办法是发布认可公告或通知,公告和通知中一般不附带国际标准的正文,也不在原标准文本上加注采用国家的编号。

(2) 封面法:在国际标准上加上采用国国家标准的编号,并附简要说明和要求,如说明对原标准做了哪些编辑修改,以及如何贯彻等要求。

(3) 完全重印法:将国际标准翻译或不作翻译,采用原标准标题,重新印刷作为国家标准,并可在国际标准正文前面,加一篇引言,做一些说明或解释、要求。

(4) 翻译法:国家标准采用国际标准的译文,可以用两种文字(原文和译文)或一种文字出版,采用时,也可在前言中说明被采用国际标准做了哪些编辑性修改,或做一些要求说明。

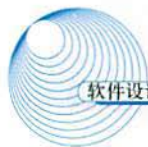
(5) 重新制定法:根据某项国际标准,重新起草国家标准,即把国际标准“熔入”国家标准之中,或做层次上的修改或做结构上的变动,但一般要保留国际标准的主要指标,或基本上保留原结构格局。

(6) 包括与引用法:制定国家标准时,完全引用或部分引用国际标准的内容。根据国际标准的“包容”情况及专业深度,可包括其国际标准的一部分,其余根据需要补充新的内容和指标。可以选择相关部分进行贯彻,其余部分不贯彻。

11.5.4 采用程度的概念

采用国际标准或国外先进标准的程度,分为等同采用、等效采用和非等效采用。

(1) 等同采用:指国家标准等同于国际标准,仅有或没有编辑性修改。编辑性修改是指不改变标准技术的内容的修改。如纠正排版或印刷错误;标点符号的改变;增加不改变技术内容



的说明、指示等。因此,可以认为等同采用就是指国家标准与国际标准相同,不做或稍做编辑性修改,编写方法完全相对应。

(2) 等效采用:指国家标准等效于国际标准,技术内容上只有很小差异。编辑上不完全相同,编写方法不完全相对应。如奥地利标准 ONORMS 5022 内河船舶噪声测量标准中,包括一份试验报告的推荐格式,而相应的国际标准 ISO 2922 中没有此内容。

(3) 非等效采用:指国家标准不等效于国际标准,在技术上有重大技术差异。即国家标准中有国际标准不能接受的条款,或者在国际标准中有国家标准不能接受的条款。在技术上有重大差异的情况下,虽然国家标准制定时是以国际标准为基础,并在很大程度上与国际标准相适应,但不能使用“等效”这个术语。通常包括以下3种情况:

① 国家标准包含的内容比国际标准少:国家标准较国际要求低或选国际标准中部分内容。国家标准与国际标准之间没有互相接受条款的“逆定理”情况。

② 国家标准包含的内容比国际标准多:国家标准增加了内容或类型,且具有较高要求等,也没有“逆定理”情况。

③ 国家标准与国际标准有重叠:部分内容是完全相同或技术上相同,但在其他内容上却互不包括对方的内容。

采用国际标准或国外先进标准,按国家标准 GB161 的规定编写。采用程度符号用缩写字母表示,等同采用 idt 或 IDT 表示,等效采用 eqv 或 EQV,非等效采用 neq 或 NEQ 表示。

(1) 等同采用:GB ××××—×××× (idt ISO ××××—××××)。

(2) 等效采用:GB ××××—×××× (eqv ISO ××××—××××)。

(3) 非等效采用:GB ××××—×××× (neq ISO××××—××××)。

11.5.5 采用国际标准和国外先进标准的原则

(1) 根据我国国民经济发展的需要确定一定时期采用国际标准和国外先进标准的方向、任务。国民经济发展的需要是采用国际标准和国外先进标准的出发点。当国民经济处于建立社会主义经济体系初期,采用国际标准和国外先进标准就是要从战略上、从国家长远利益上考虑突出国际标准中的重大基础标准、通用方法标准的采用问题。当国民经济发展到一定阶段,如产品质量要赶超世界先进水平时,对国际标准和国外先进标准中的先进产品标准和质量标准就成为采用的重要对象。

(2) 很多国际标准是国际上取得多年实际经验后被公认的,一般来说不必都去进行实践验证。为加快采用国际标准和国外先进标准的速度,一般都简化制定手续,基本上采取“先拿来用,然后实践验证,再补充修改”的模式。

(3) 促进产品质量水平的提高是当前采用国际标准和国外先进标准的一项重要原则。产品质量问题首先有标准问题,只有采用了先进的国际标准或先进的国外标准,才能提高我国的标

准水平。只有提高了标准水平,才能有力地促进产品质量的提高。要赶超世界先进水平就要采用国际标准和国外先进标准。

(4) 要紧密结合我国实际情况、自然资源和自然条件,须符合国家的有关法令、法规和政策,做到技术先进、经济合理、安全可靠、方便使用、促进生产力发展。

(5) 对于国际标准中的基础标准、方法标准、原材料标准和通用零部件标准,需要先行采用。通过的基础标准、方法标准,以及有关安全、卫生、环境保护等方面的标准,一般应与国际标准协调一致。

(6) 在技术引进和设备进口中采用国际标准,应符合《技术引进和设备进口标准化审查管理办法(试行)》中的规定。例如,原则上不引进和进口英制设备,等等。

(7) 当国际标准不能满足要求,或尚无国际标准时,应参照上述原则,积极采用国外先进标准。

11.6 信息技术标准化

信息技术标准化是围绕信息技术开发、信息产品的研制和信息系统建设、运行与管理而开展的一系列标准化工作。其中主要包括信息技术术语、信息表示、汉字信息处理技术、媒体、软件工程、数据库、网络通信、电子数据交换、电子卡、管理信息系统、计算机辅助技术等方面标准化。

11.6.1 信息编码标准化

编码是一种信息表现形式。在一定条件下,它对事物或概念的描述,比自然语言要直接、简洁、准确和有力。例如,交通管理人员使用的不同颜色的信号灯,就是一种特定的信息代码,用它表示行驶、慢驶、停止等交通指令信息十分方便、准确、迅速。编码作为一种形式,应具有-致性,在一定范围内为人们共同接受,否则就不能发挥作用。例如,邮政编码必须在一个国家范围内一致;公共汽车交通路线编码,必须在一个城市或地区范围内一致;学生学号编码必须在学校系统内一致。要保证信息编码的一致性,就要对编码对象的确定、对象特性的选择、编码方法和代码设计进行标准化。

编码是一种信息交换的技术手段。对信息进行编码实际上是对文字、音频、图形、图像等信息进行处理,使之量化,从而便于利用各种通信设备进行信息传递和利用计算机进行信息处理。作为一种信息交换的技术手段,必须保证信息交换的一致性。例如,计算机内部的所有数据都是用二进制数表示的,但人们向计算机输入的信息,则是人类语言中数字、文字和专用符号,经计算机处理后的输出也必须是人们所能识别的字符。每个字符所对应的二进制数,便是该字符的代码。各计算机所定义的输入/输出符号集和每个符号的代码,便是该计算机的编码系



统。只有具有相同编码系统的计算机,才可能接受用户编写的同一符号的程序。为了统一编码系统,人们借助了标准化这个工具,制定了各种标准代码,如国际上比较通用的 ASCII 码(美国信息交换标准代码)。

11.6.2 条码标准化

条码是一种特殊的代码。国家标准《GB 12905—91 条码系统通用术语 条码符号术语》中定义,条码是“一组规则排列的条、空及其对应字符组成的标记,用以表示一定的信息”。条码中的条、空分别由两种不同的深浅的颜色(通常为黑、白色)表示,并满足一定的光学对比度要求,其目的是便于光电扫描设备识读后将数据输入计算机。条码中的字符供人们直接识读,或通过键盘向计算机输入数据。目前国际上广泛使用的条码是国际物品编码协会的标准化条码 EAN。我国于 1991 年 4 月正式加入国际物品编码协会,我国国家标准 GB 904—91 通用商品条码的结构与 EAN 条码结构相同,由 13 位数字码以及对应的条码组成,前缀码(3 位)、制造厂商代码(4 位)、商品代码(5 位)、检验码(1 位),3 位前缀码是标识国家或地区的代码。我国的国家代码为“690”。

11.6.3 汉字编码标准化

汉字编码是对每一个汉字按一定的规律用若干个字母、数字、符号表示出来。汉字编码的方法很多,主要有数字编码,如电报码、四角号码字典中的汉字号码都是用数字对汉字进行编码;拼音编码,即用汉字的拼音字母对汉字进行编码;字形编码,即用汉字的偏旁部首和笔画结构与各个英文字母相对应,再用英文字母的组合代表相应的汉字。对每一种汉字编码,计算机内部都有一种相应的二进制内部码,不同的汉字编码,在使用上不能替换。若把各种编码方案都存入计算机供人们选择,在技术上和使用效果上则是困难的和不经济的。我国在汉字编码标准化方面取得的突出成就就是信息交换用汉字编码字符集国家标准的制定。该字符集共有 6 集。其中,GB 2312—80 信息交换用汉字编码字符集是基本集,收入常用基本汉字和字符 7445 个。GB 7589—87 和 GB 7590—87 分别是第二辅助集和第四辅助集,各收入现代规范汉字 7426 个。GB/T 12345—90 是辅助集,它与第三辅助集和第五辅助集分别是与基本集、第二辅助集和第四辅助集相对应的繁体字的汉字字符集。除汉字编码标准化外,汉字信息处理标准化的内容还包括:汉字键盘输入的标准化;汉字文字识别输入和语音识别输入的标准化;汉字输出字体和质量的标准;汉字属性和汉语词语的标准化等。

11.6.4 软件工程标准化

随着软件工程学科的发展,人们对计算机软件的认识逐渐深入。软件工作的范围从只是使用程序设计语言编写程序,扩展到整个软件生存期。软件工程的目的是改善软件开发的组织,降低开发成本,缩短开发时间,提高工作效率,提高软件质量。它在内容上包括软件开发的软

件概念形成、需求分析、计划组织、系统分析与设计、结构程序设计、软件调试、软件测试和验收、安装和检验、软件运行和维护,以及软件运行的终止。同时还有许多技术管理工作,如过程管理、产品管理、资源管理,以及确认与验证工作,如评审与审计、产品分析等。软件工程最显著的特点就是把个别的、自发的、分散的、手工的软件开发变成一种社会化的软件生产方式。软件生产的社会化必然要求软件工程实行标准化。软件工程标准的类型也是多方面的,常常是跨越软件生存期各个阶段。所有这些方面都应逐步建立标准或规范。软件工程标准化的主要内容包括过程标准(如方法、技术、度量等)、产品标准(如需求、设计、部件、描述、计划、报告等)、专业标准(如道德准则、认证等)、记法标准(如术语、表示法、语言等)、开发规范(准则、方法、规程等)、文件规范(文件范围、文件编制、文件内容要求、编写提示)、维护规范(软件维护、组织与实施等)以及质量规范(软件质量保证、软件配置管理、软件测试、软件验收等)等。

我国1983年5月成立“计算机与信息处理标准化技术委员会”,下设13个分技术委员会,其中程序设计语言分技术委员会和软件工程技术委员会与软件相关。我国推行软件工程标准化工作的总原则是向国际标准靠拢,对于能够在我国适用的标准全部按等同采用的方法,以促进国际交流。虽然我国的软件工程标准化工作仍处于起步阶段,但在提高我国软件工程水平,促进软件产业的发展以及加强与国外的软件交流等方面必将起到应有的作用。现已得到国家批准的软件工程国家标准有:

1. 基础标准

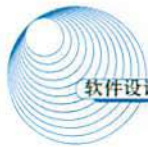
- (1) 信息处理—程序构造及其表示法的约定 GB/T 13502—92;
- (2) 信息处理系统—计算机系统配置图符号及其约定 GB/T 14085—93;
- (3) 软件工程技术术语标准 GB/T 11457—89;
- (4) 软件工程标准分类法 GB/T 15538—95。

2. 开发标准

- (1) 软件开发规范 GB 8566—88 ;
- (2) 计算机软件单元测试 GB/T 15532—95;
- (3) 软件维护指南 GB/T 14079—93。

3. 文档标准

- (1) 计算机软件产品开发文件编制指南 GB 8567—88;
- (2) 计算机软件需求说明编制指南 GB/T 9385—88;
- (3) 计算机软件测试文件编制指南 GB/T 9386—88;



4. 管理标准

- (1) 计算机软件配置管理计划规范 GB/T 12505—90;
- (2) 计算机软件质量保证计划规范 GB/T 12504—90;
- (3) 计算机软件可靠性和可维护性管理 GB/T 14394—93;
- (4) 信息技术、软件产品评价、质量特性及其使用指南 GB/T 16260—96。

11.7 标准化组织

11.7.1 国际标准化组织

ISO 和 IEC 是世界上两个最大、最具有权威的国际标准化组织。目前,由 ISO 确认并公布的国际标准化组织还有国际计量局(BIPM)、联合国教科文组织(UNESCO)、世界卫生组织(WHO)、世界知识产权组织(WIPO)、国际信息与文献联合会(FID)、国际法制计量组织(OIML)等 27 个国际组织。

1. 国际标准化组织 ISO

国际标准化组织 ISO(International Organization for Standardization)是世界上最大的非政府性的,由各国标准化团体(ISO 成员团体)组成的世界性联合专门机构。它成立于 1947 年 2 月,其宗旨是世界范围内促进标准化工作的发展,以利于国际资源的交流和合理配置,扩大各国在知识、科学、技术和经济领域的合作;其主要活动是制定国际标准,协调世界范围内的标准化工作,组织各成员国和技术委员会进行交流,以及与其他国际性组织进行合作,共同研究有关标准问题,出版 ISO 国际标准。制定国际标准的工作通常由 ISO 的技术委员会完成,各成员团体若对某技术委员会确立的项目感兴趣,均有权参加该委员会的工作的。与 ISO 保持联系的各国际组织(官方的或非方的)也可参加有关工作。此外,ISO 还负责协调世界范围内的标准化工作,组织各成员国和技术委员会进行情报交流,并和其他国际性组织保持联系和合作,共同研究感兴趣的有关标准化问题。在电工技术标准化方面,ISO 与 IEC 保持密切合作关系。ISO 的工作语言是英文、法文、俄文,会址设在日内瓦。

ISO 的成员团体分正式成员和通讯成员。正式成员是指由各国最有代表性的标准化机构代表其国家或地区参加,并且只允许每个国家有一个组织参加。通讯成员是尚未建立全国性标准化机构的国家,一般不参与 ISO 的技术工作,但可参会了解工作进展,当条件成熟时,可以通过一定程序成为正式成员。

成员全体大会是 ISO 的最高权力机构。理事会是 ISO 常务机构,由正、副主席、司库和

18 个理事国代表组成, 每年召开一次会议, 理事会成员任期 3 年, 每年改选 1/3 的成员。理事会下设若干专门委员会。其中之一是技术委员会 (TS), 技术委员会完成 ISO 的技术工作, ISO 按专业性质设立技术委员会, 各技术委员会根据工作需要可设立若干分委员会 (SC), TC 和 SC 下面可设立若干工作组 (WG)。TC 和 SC 的成员分为积极参加成员 (P 成员) 和观察员 (O 成员) 两种。P 成员可参与 TC、SC 的技术工作, 而 O 成员则只能了解工作进度和得到技术组织的信息资料, 不参加技术工作。每个 TC 或 SC 均从 P 成员中任命一个成员主持秘书处并领导该委员会或分委员会。ISO 现有技术组织 2871 个, 其中技术委员会 (TC) 191 个、分技术委员会 (SC) 572 个、工作组 2063 个, 临时专题小组 45 个。

2. 国际电工委员会 IEC

国际电工委员会 IEC (International Electrotechnical Commission) 成立于 1906 年, 是世界上最早的非政府性国际电工标准化机构, 是联合国经社理事会 (ECOSOC) 的甲级咨询组织。自 1947 年 ISO 成立后, IEC 曾作为一个电工部并入 ISO, 但在技术上和财务上仍保持独立。1976 年双方又达成新协议, IEC 从 ISO 中分离出来, 两组织各自独立, 自愿合作, 互为补充, 共同建立国际标准化体系, IEC 负责有关电气工程及电子领域国际标准化工作, 其他领域则由 ISO 负责。

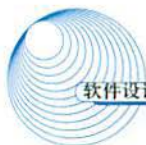
IEC 的工作领域包括电工领域各个方面, 如电力、电子、电讯和原子能方面的电工技术等。理事会是 IEC 的最高权力机构, 会址设在日内瓦。IEC 理事会下设执行委员会和合格评定局。执行委员会负责管理技术委员会 (TC) 和技术咨询委员会。合格评定局管理各认证委员会在组织上自成体系。它是世界范围的自愿认证机构, 其宗旨是促进国家或国际间的自由贸易。按照严格的认证程序, 以国际标准为依据对电工产品生产厂的技术力量和管理水平, 实行全面的审核和评审; 对要求认证的生产的元器件, 按标准要求进行测试检验。对符合质量要求的产品授以合格证书, 以确保产品质量达到和保持标准要求的质量水平。

11.7.2 区域标准化组织

区域标准化组织是指同处一个地区的某些国家组成的标准化组织。区域是指世界上按地理、经济或民族利益划分的区域。参加组织的机构有的是政府性的, 有的是非政府性的, 为发展同一地区或毗邻国家间的经济及贸易, 维护该地区国家的利益, 协调本地区各国标准和技术规范而建立的标准化机构。其主要职能是制定、发布和协调该地区的标准。

(1) 欧洲标准化委员会 (CEN) 成立于 1961 年, 是欧洲经济共同体 (EEC)、欧洲自由联盟 (EFTA) 所属国家的标准化机构所组成, 主要任务是协调各成员国的标准, 制定必要的欧洲标准 (EN), 实行区域认证制度。

(2) 欧洲电工标准化委员会 (CEN EL EC) 成立于 1972 年, 是由欧洲电工标准协调委员会 (CEN EL) 和欧洲电工协调委员会共同市场小组 (CEN EL COM) 合并组成, 主要是协调



各成员国电器和电子领域的标准,以及电子元器件质量认证,制定部分欧洲标准(EN)。

(3) 亚洲标准咨询委员会(ASAC)成立于1967年,由联合国亚洲与太平洋经社委员会协商建立,主要是ISO、IEC标准的基础上,协调各成员国标准化活动,制定区域性标准。

(4) 国际电信联盟ITU(International Telecommunication Union):ITU于1865年5月在巴黎成立,1947年成为联合国的专门机构,是世界各国政府的电信主管部门之间协调电信事务的一个国际组织,研究制定有关电信业务的规章制度,通过决议提出推荐标准,收集有关情报。ITU的目的和任务是维持和发展国际合作,以改进和合理利用电信,促进技术设施的发展及有效应用,以提高电信业务的效率。

11.7.3 行业标准化组织

行业标准化组织是指制定和公布适应于某个业务领域标准的专业标准化团体,以及在其业务领域开展标准化工作的行业机构、学术团体或国防机构。

(1) 美国电气电子工程师学会IEEE(Institute of Electrical and Electronics Engineers)是由美国电气工程师学会(AIEE)和美国无线电工程师学会(IRE)于1963年合并而成,是美国规模最大的专业学会。IEEE主要制定标准内容有电气与电子设备、试验方法、元器件、符号、定义以及测试方法等。近年该学会专门成立了软件标准分技术委员会(SESS),积极开展了软件标准化活动,取得了显著成果,受到了软件界的关注。IEEE通过的标准常常要报请ANSI审批,使具有国家标准的性质。因此,IEEE公布的标准常冠有ANSI字头。例如,ANSI/IEEE Str 828—1983软件配置管理计划标准。

(2) 美国国防部批准、颁布适用于美国军队内部使用的标准代号为DOD(采用公制计量单位的以DOD表示)和MIL。

(3) 我国国防科学技术工业委员会批准、颁布适合于国防部门和军队使用的标准,代号为GJB。例如,1988年发布实施的GJB 473—88军用软件开发规范。

11.7.4 国家标准化组织

国家标准化组织是指在国家范围内建立的标准化机构,以及政府确认(或承认)的标准化团体,或者接受政府标准化管理机构指导并具有权威性的民间标准化团体。这些组织主要有:

(1) 美国国家标准学会ANSI(American National Standards Institute):ANSI是非赢利性质的民间标准化团体,但它实际上已成为美国国家标准化中心,美国各界标准化活动都围绕它开展。通过它使政府有关系统和民间系统相互配合,起到了政府和民间标准化系统之间的桥梁作用。ANSI协调并指导美国全国的标准化活动,给标准制定、研究和使用单位以帮助,提供国内外标准化情报。ANSI本身很少制定标准,主要是将其他专业标准化机构的标准经协商后冠以ANSI代号,成为美国国家标准。

(2) 英国标准学会 BSI (British Standards Institution): BSI 是世界上最早的全国性标准化机构,它是政府认可的、独立的、非赢利性民间标准化团体。主要任务是为增产节约努力协调生产者和用户之间的关系,促进生产,达到标准化;制定和修订英国标准,并促进其贯彻执行;以学会名义,对各种标志进行登记,并颁发许可证;必要时采取各种行动,保护学会利益;对外代表英国参加国际或区域标准化活动。

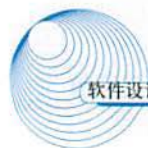
(3) 德国标准化学会 DIN (Deutsches Institut für Normung): DIN 始建于 1917 年,当时称为德国工业标准委员会 (NADI),1926 年改为德国标准委员会 (DNA),1975 年又改名为联邦德国标准化学会 (DIN)。DIN 是一个经注册的公益性民间标准化团体,前联邦政府承认它为联邦德国和西柏林的标准化机构。德国统一后,其活动已遍及全德国。DIN 在国际上一直享有盛誉,其制定的标准在很多国家被广泛采用。

(4) 法国标准化协会 AFNOR (Association Française de Normalisation): AFNOR 成立于 1926 年。它是一个公益性的民间团体,也是一个被政府承认,为国家服务的组织。1941 年 5 月 24 日颁布的一项法令确认 AFNOR 接受法国政府的标准化管理机构“标准化专署”指导,按政府指导开展工作,并定期向标准化专员汇报工作。AFNOR 负责标准的制定、修订工作,宣传、出版、发行标准,实施产品质量认证。

11.8 ISO 9000 标准简介

11.8.1 ISO 9000 标准

ISO 9000 标准是一系列标准的统称。ISO 9000 系列标准由 ISO/TC 176 制定。TC 176 是 ISO 的第 176 个技术委员会(质量管理和质量保证技术委员会),专门负责制定质量管理和质量保证技术的标准。经过 TC 176 多年的协调以及有关国家质量管理专家近 10 年的不懈努力,总结了美国、英国、加拿大等工业发达国家的质量保证技术实践的经验,于 1986 年 6 月 15 日正式发布了 ISO 8402《质量—术语》标准,又于 1987 年 3 月正式公布了 ISO 9000~ISO 9004 五项标准,这五项标准与 ISO 8402:1986 一起统称为 ISO 9000:1987 系列标准。随着工业、经济的不断发展,国际贸易交往的日益频繁,产品质量成为各方面关注的焦点。因此,质量管理的对象也从硬件逐渐扩展到了软件、文档和服务。按照全面质量管理的 PDCA 工作模式,TC 176 于 1990 年在第九届年会上提出了《90 年代国际质量标准的实施策略》,决定对 ISO 9000 标准进行修改。通过对 ISO 9001/2/3/4 标准的技术内容修订(局部修改),形成 1994 版。在 ISO 9000—1:1994 中增加了过程和过程网络等基本概念,为进一步修改提供了理论基础。之后,引入与 PDCA 循环模式统一的过程方法模式,从总体结构和原则到具体的技术内容对 ISO 9000 系列标准进行了全面的修改,形成 2000 年版。2000 年 12 月 15 日,ISO 9000:2000 系列标准正式发布实施。



ISO 9000 系列标准的质量管理模式为企业注入新的活力和生机,给质量管理体系提供评价基础,为企业进行世界贸易带来质量可信度。从 ISO 9000 系列标准的演变过程可见,ISO 9001:1987 系列标准从自我保证的角度出发,更多关注的是企业内部的质量管理和质量保证;ISO 9001:1994 系列标准则通过 20 个质量管理体系要素,把用户要求、法规要求及质量保证的要求纳入标准的范围中;ISO 9001:2000 系列标准在标准构思和标准目的等方面体现了具有时代气息的变化,过程方法的概念,顾客需求的考虑,以及将持续改进的思想贯穿于整个标准,把组织的质量管理体系满足顾客要求的能力和程度体现在标准的要求之中。

11.8.2 ISO 9000: 2000 系列标准文件结构

ISO 9000: 2000 族标准现有 13 项标准,由 4 个核心标准,1 个支持标准,6 个技术报告和 3 个小册子构成,见表 11-1。

表 11-1 ISO 9000: 2000 系列标准文件结构

核心标准	ISO 9000: 2000《基本原理和术语》
	ISO 9001: 2000《质量管理体系 要求》
	ISO 9004: 2000《质量管理体系 业绩改进指南》
	ISO 19011: 2000《质量和环境管理审核指南》
其他标准	ISO 10012《测量设备的质量保证要求》
技术报告	ISO 10006《项目管理指南》
	ISO 10007《技术状态管理指南》
	ISO 10013《质量管理体系文件指南》
	ISO 10014《质量经济性指南》
	ISO 10015《教育和培训指南》
	ISO 10017《统计技术在 ISO 900 中的应用指南》
小册子	质量管理原理
	选择和使用指南
	小型企业的应用指南

11.8.3 ISO 9000: 2000 核心标准简介

1. ISO 9000: 2000《质量管理体系 基础和术语》

该标准描述了质量管理体系的基础,并规定了质量管理体系的术语和基本原理。术语标准是讨论问题的前提,统一术语是为了明确概念,建立共同的语言。

该标准在总结了质量管理经验的基础上,明确了一个组织在实施质量管理中必须遵循的 8

项质量管理原则，也是 ISO 9000: 2000 系列标准制定的指导思想和理论基础。该标准提出的 10 个部分 87 个术语，在语言上强调采用非技术性语言，使所有潜在用户易于理解。为便于使用，在标准附录中，推荐了以“概念图”方式来描述相关术语的关系。

2. ISO 9001: 2000《质量管理体系 要求》

该标准提供了质量管理体系的要求，供组织证实其具有提供满足顾客要求和适用法规要求产品的能力时使用。组织通过有效地实施体系，包括过程的持续改进和预防不合格产品，使顾客满意。该标准是用于第三方认证的唯一质量管理体系要求标准，通常用于企业建立质量管理体系以及申请认证。它主要通过对申请认证组织的质量管理体系提出各项要求来规范组织的质量管理体系。主要分为质量管理体系、管理职责、资源管理、产品实现、测量分析和改进 5 大模块，构成一种过程方法模式的结构，符合 PDCA 循环规则，且通过持续改进的环节使质量管理体系的水平达到螺旋式上升的效应，其中每个模块中又分有许多分条款。

3. ISO 9004: 2000《质量管理体系 业绩改进指南》

该标准给出了改进质量管理体系业绩的指南，描述了质量管理体系应包括持续改进的过程，强调通过改进过程，提高组织的业绩，使组织的顾客及其他相关方满意。

该标准是和 ISO 9001: 2000 协调一致并可一起使用的质量管理体系标准，两个标准采用相同的原则，但应注意其适用范围不同，而且 ISO 9004 标准不拟作为 ISO 9001 标准的实施指南。通常情况下，当组织的管理者希望超越 ISO 9001 标准的最低要求，追求增长的业绩改进时，一般以 ISO 9004 标准作为指南。

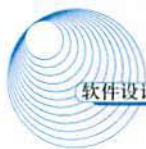
4. ISO 19011: 2001《质量管理体系和环境管理体系审核指南》

该标准提供了质量管理体系和环境管理体系审核的基本原则、审核方案的管理、环境质量管理体系的实施以及对环境和质量管理体系评审员资格要求提供了指南。

该标准是 ISO/TC 176 与 ISO/TC 207（环境管理技术委员会）联合制定的，按照“不同管理体系，可以共同管理和审核”的原则，在术语和内容方面，兼容了质量管理体系和环境管理体系两方面特点。

11.8.4 ISO 9000: 2000 系列标准确认的八项原则

ISO 9000 系列质量管理体系在 ISO 9000: 2000 和 ISO 9004: 2000 标准中提及的八项质量管理原则是该标准中一个非常重要的内容，是整个 ISO 9000 系列质量管理体系标准的精髓和纲领。它是标准的理论基础，又是组织领导者进行质量管理的基本原则。增加了贯彻八项质量管理的基本原则的要求，并且以此为主线贯彻于各个过程中，体现在标准的全部内容上。八项质



量管理原则如下所述。

1. 以顾客为中心

“组织依存于顾客。因此,组织应了解顾客当前的和未来的需求,满足顾客要求并争取超越顾客期望。”顾客是每一个组织存在的基础,顾客的要求是第一位的,组织应全面地调查和研究顾客的需求和期望,并把它转化为质量要求,采取有效措施使其实现。例如,通过对市场获得顾客的质量评价,测量顾客的满意度,并根据结果来指导组织的运作:提高组织资源使用的有效性来增加顾客的满意度;确保顾客满意和其他相关方的利益(例如所有者、雇员、供方、金融机构、所在社区和整个社会)。

2. 领导作用

“领导者确立一致的组织宗旨和方向。他们应当创造并保持员工能够充分参与实现组织目标的内部环境。”领导者具有决策和领导一个组织的关键作用。为了营造一个良好的环境,领导者应关注顾客要求,清晰地规划组织的未来,建立质量方针和质量目标,确保建立和实施一个有效的质量管理体系,以统一的方式来评估、安排和实施活动,并随时将组织运行的结果与目标比较,根据情况决定实现质量方针和目标的措施,决定持续改进的措施。考虑所有利益相关方的需要,包括顾客、员工、供方、所在社区和整个社会。在组织的各个层次树立价值共享和道德伦理的观念,向员工提供所需的资源和培训,赋予其在履行职责和义务的自由度,鼓舞、奖励和承认员工的贡献,让员工了解组织目标并追求组织的成功。在领导作风上要做到透明、务实和以身作则。

3. 全员参与

“各级人员是组织之本,只有他们的充分参与,才能使他们的才干为组织带来最大的收益。”组织的质量管理不仅需要最高管理者的正确领导,还有赖于全员的参与。全体职工是每个组织的基础,让他们了解其贡献的重要性和组织中的角色,承担起解决问题的责任,使得他们的才干为组织带来收益。所以要对职工进行质量意识、职业道德、以顾客为中心的意识 and 敬业精神的教育,激发他们的积极性和责任感,在团队(组织)中自由地分享知识和经验,公开地讨论问题和观点,主动地寻求机会来加强他们的技能、知识和经验,积极为组织的持续改进做出贡献,进一步促进组织目标的创新和创造力。

4. 过程方法

“将相关的资源和活动作为过程进行管理,可以更高效地得到期望的结果。”在应用于质量管理体系时,ISO 9000:2000 系列标准建立了一个过程模式,将“管理职责”、“资源管理”、“产

品实现”、“测量、分析和改进”作为体系的4大主要过程,描述其相互关系、并以顾客要求为输入,提供给顾客的产品为输出,通过信息反馈来测定顾客满意度,评价质量管理体系的业绩。应用过程方法有助于获得不断改进、协调一致、并可预测的结果,以便提供重点及有优先次序的改进方案,达到有效地使用资源、降低成本、缩短周期和提高质量的目的。过程方法的原则不仅适用于某些简单的过程,也适用于由许多过程构成的过程网络。

5. 管理的系统方法

“识别、理解并管理作为体系的相互关联的过程,有助于提高组织的有效性和效率。”针对设定的目标,识别、理解体系的各个过程之间的内在关联性,采用科学的方法以协调和整合过程,构造和实施质量管理体系的方法,既可用于新建体系,也可用于现有体系的改进。这种方法的实施,能够深入理解为实现目标所必需的作用和责任;提供对过程能力及产品可靠性的信任;便于将注意力集中于重点过程;为持续改进打好基础;使顾客满意,使利益相关方对组织的协调性、有效性和效率建立信心,最终使组织获得成功。

6. 持续改进

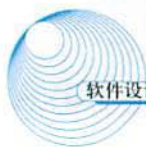
“组织总体业绩的持续改进应是组织的一个永恒的目标。”在质量管理体系中,改进是指产品质量、过程及体系有效性和效率的提高。持续改进包括:了解现状,建立目标,寻找、评价和实施解决办法,测量、验证和分析结果,把更改纳入文件等活动。把产品、过程和体系的持续改进作为组织内每个成员的目标,根据组织的战略意图协调各层次上的改进活动,建立指导和测量跟踪持续改进的目标,提供持续改进方法和工具来持续改进组织的业绩。在竞争激烈的市场环境中,通过改进组织能力,对机遇的快速灵活反应,可增强竞争优势。

7. 基于事实的决策方法

“有效的决策是建立在对数据和信息进行分析的基础上。”对数据和信息的逻辑分析或直觉判断是有效决策的基础。以事实为依据,通过参照实际记录来证明以往决策的有效性,对各种意见和决定加以评审、质疑和修改,可防止决策失误。以事实为依据,要求数据和信息具有足够的精确度、可靠性和可获取性,需要使用正确的方法分析数据和信息。统计技术是最重要的工具之一,可用来测量、分析和说明产品和过程的变异性,为持续改进的决策提供依据。

8. 互利的供方关系

“组织与其供方是相互依存的,通过互利的关系可增强双方创造价值的 ability。”供方提供的产品将对组织向顾客提供满意的产品产生重要影响,因此处理好与供方的关系,影响到组织能否持续稳定地提供顾客满意的产品。对供方不能只讲控制(或提要求)不讲合作互利,需要在



对短期的收益和长期的利益综合平衡的基础上建立相互关系,对市场或顾客的需求和期望的变化,一起做出灵活快速的反应。特别是对关键的供方,更要建立合作关系,开展联合开发和改进活动,开放式地进行交流,激励并承认供方的改进和成就。建立互利关系,双方共享经验和资源,分享信息和对未来的规划。这对组织和供方都有利。

11.9 能力成熟度模型 CMM 简介

软件质量是人们实践产物的属性和行为,是一个很复杂的事物性质和行为,可以通过一些方法和人们的活动来改进质量。概括地说,通过控制软件生产过程、提高软件生产者组织性和软件生产者个人能力来改进软件质量。已经应用的软件质量改进方法有很多,软件能力成熟度模型 CMM (Capability Maturity Model) 是一个目前国际上较流行、较实用的软件生产过程行业标准模型,用于定义和评价软件开发过程的成熟度,并提供怎样做才能提高软件质量的指导。通过创建规范的软件过程(开发和维护软件及其相关产品的一组活动、方法、实践和改革)、软件管理过程(为使软件工程过程顺利进行而进行的管理活动的集合)以及软件企业的过程(企业对软件的组织活动,是以企业为主的活动)三者的有机结合,达到管理并控制软件产品的质量。

CMM 是 Carnegie Mellon 大学软件工程研究所(CMU/SEI)在与企业界和政府合作的基础上开发出来的模型。CMM 以几十年产品质量概念和软件工业的经验及教训为基础,为软件企业的软件能力不断走向成熟提供了有效的步骤和阶梯式的进化框架。它指明了一个成熟的软件企业在软件开发方面需要管理的主要工作和这些工作之间的关系,以及以怎样的先后次序,一步一步地达到预定的目标,从而得到持续的过程改进,实现企业高效率、低成本地交付高质量软件产品的战略目标。

CMM 为软件企业的过程能力提供了一个阶梯式的进化框架,将软件过程改进的进化步骤分成 5 个成熟度等级,每一个级别定义一组过程能力目标,并描述要达到这些目标应该采取的实践活动,为不断改进过程奠定了循序渐进的基础。第一级实际上是一个起点,任何准备按 CMM 体系进化的企业都自然处于这个起点上,并通过这个起点向第二级迈进。除第一级外,每一级都设定了一组目标,如果达到了这组目标,则表明达到了这个成熟级别,可以向上一个级别迈进。CMM 体系不主张跨越级别的进化,因为从第二级起,对低级别的实现是实现高级别的基础。

1. 初始级

在初始级,企业一般缺少有效的管理,不具备稳定的软件开发与维护的环境。软件过程是未加定义的随意过程,项目的执行随意甚至是混乱的,几乎没有定义过程的规则(或步骤)。软件过程在实际的工作过程中经常改变(过程是随意的),其成果是不稳定的,不可预见的,

不可重复的。也就是说,软件的计划、预算、功能和产品的质量都是不可确定和不可预见的。项目的成功完全依赖个人的能力和他们先前的经验、知识以及他们的进取心和积极程度。当项目遇到危机时,通常会放弃原定的计划而只专注于编程与测试。有些企业虽然制定了一些软件工程规范,但这些规范未能覆盖基本的关键过程要求,并且执行没有政策、资源等方面的保证时,那么它仍然被视为初始级。

2. 可重复级

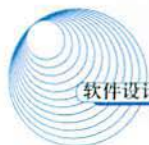
在可重复级,企业建立了基本的项目管理过程的政策和管理规程,对成本、进度和功能进行监控,以加强过程能力。对新项目的计划和管理是基于以往的相似或同类项目的成功经验,以确保再一次的成功。一个可管理的过程则是一个可重复的过程,一个可重复的过程则能逐渐进化和成熟。第二级的焦点集中在软件管理过程上,包括了需求管理、项目管理、质量管理、配置管理和子合同管理等方面。软件项目的计划和跟踪与监控的稳定实施,表现出一个按计划执行的且阶段可控的软件开发过程,并表明软件开发过程是相对稳定的,过程建立在项目一级。项目的成功依赖于个人的能力以及管理层的支持。

3. 已定义级

在定义级,企业全面采用综合性的管理及工程过程来管理,对整个软件生命周期的管理与工程化过程都已标准化,并综合成软件开发企业标准的软件过程。企业标准软件过程通过证明是正确且实用的,所有开发的项目须根据标准过程,剪裁出与项目适宜的过程,并执行这些过程。企业标准软件过程被应用到所有的工程中,用于编制和维护软件。有的项目也可根据实际情况,对软件开发组织的标准软件过程进行剪裁。建立了软件工程过程小组,长期承担评估与调整软件过程的任务,以适应未来软件项目的要求。企业内部的所有人对于所定义的软件过程的活动、任务有深入了解,以项目组的方式进行工作,形成产品团队。

4. 已管理级

在管理级,企业开始定量地认识软件过程,软件质量管理和软件过程管理是量化的管理。对软件过程与产品质量建立了定量的质量目标,制定了软件过程和产品质量的详细而具体的度量标准,实现了度量标准化。通过一致的度量标准来指导软件过程,保证所有项目对生产率和质量进行度量,并作为评价软件过程及产品的定量基础。量化控制使得软件开发真正成为一种工业生产活动。软件过程按照明确的度量标准度量和操作,软件过程以及软件产品的质量的一些趋势就可以得以控制和预见。经度量后一旦发现质量超出或违反标准的,可以采用一些方法及时改进。每个人都了解个人的作用与企业的关系,在项目中存在强烈的群体工作意识。



5. 优化级

在优化级,企业将会把工作重点放在对软件过程改进的持续性、预见性及自身增强上,防止缺陷及问题的发生,不断地提高过程处理能力。通过来自过程执行的质量反馈和吸收新方法和新技术的定量分析来改善下一步的执行过程,即优化执行步骤,使软件过程能不断地得到改进。根据软件过程的效果,进行成本/利润分析,从成功的软件过程中吸取经验,把最好的创新成果迅速向全企业推广,对失败的案例进行分析以找出原因并预先改进,把失败的教训告知全体组织以防止重复以前的错误,不断提高产品的质量和生产率。整个企业都存在自觉的强烈的团队意识,每个人都致力过程改进,而要力求减少错误率。优化级的目标是达到一个保持持续不断的软件过程改进的境界。如果一个企业达到了这一级,那么表明企业能够根据实际的项目性质、技术等因素,不断调整软件生产过程以求达到最佳。

11.10 ISO/IEC 15504 过程评估标准简介

ISO/IEC15504 由 ISO/IEC JTC1/SC7/WG10 与其项目组 SPICE (Software Process Improvement and Capability Etermination, 软件过程改进和能力确定) 和国际项目管理机构共同完成,并收集整理了来自 20 多个国家的工业、政府以及大学专家的意见和建议,同时得到世界各地软件工程师的帮助,包括与美国 SEI、加拿大贝尔合作。

ISO/IEC 15504 提供了一个软件过程评估的框架。它可以被任何软件企业用于软件的设计、管理、监督、控制以及提高获得、供应、开发、操作、升级和支持的能力。ISO/IEC 15504 提供了一种有组织的、结构化的软件过程评估方法,以便实施软件过程的评估。在 ISO/IEC 15504 中定义的过程评估办法旨在为描述工程评估结果的通用方法提供一个基本原则,同时也对建立在不同但兼容的模型和方法上的评估进行比较。过程评估有两个主要的使用环境:软件过程改进或者软件过程能力评定。在软件过程改进环境中,过程评估提供了诸多方法,用于在企业内部根据所选择的过程的性能来描述当前实践的特性。根据企业的商业需要对结果进行分析以确定该过程内在的优点、不足和风险,从而可以判断过程是否有效地实现其目标以及导致质量低下、耗时过多、成本过高的主要原因。评估结果可以为软件过程改进的优先顺序提供相应的基础和依据。在确定软件过程能力的环境中,评估是通过与目标过程能力的对比分析评估过程的能力,从而确定所评估的过程对实现有关项目的风险情况。在 ISO/IEC 15504 文件中涉及了过程评估的各个方面,其文档主要包括以下几个部分。

1. 概念和绪论指南

这部分给出了关于软件过程改进和过程评估概念及其在过程能力确定方面的总体信息。它

描述了 ISO/IEC 15504 文档的各部分是如何组织在一起的，并为选择和使用各部分提供指南。此外，本部分还解释了 ISO/IEC 15504 中所包含的要求对执行评估的适用性；支持工具的建立与选择以及在附加过程的建立和发展方面所起的作用。

2. 过程和过程能力参考模型

该部分从内容上说是在比较高的层次上详细定义了一个用于过程评估的二维参考模型。此模型中描述了过程和过程能力。通过将过程中的特点与不同的能力等级相比较，我们可以用此模型中定义的一系列过程和框架对过程能力加以评估。

3. 实施评估

为了确保等级评定的一致性和可重复性（即标准化），ISO/IEC 15504 为软件过程评估提供了一个框架并为进行评审提出了最低要求。这些要求有助于确保评估输出内在的一致性，并为评级和验证与要求的一致性提供了依据。该部分以及与该部分有关的内容详细定义了实施评估时的需要，这样得到的评估结果才有可重复性、可信性以及可持续性。

4. 评估实施指南

通过这部分内容，可以指导使用者如何进行软件过程评估。这个具有普通意义的指导可适用于所有企业，同时也适用于采用不同的方法、技术以及支持工具的过程评估。它包括如何选择并使用兼容的评估，如何选择用于支持评估的方法，如何选择适合于评估的工具与手段。该部分内容对过程评估做了概述并且以指南形式对用于评估的兼容模型、文件化的评估过程和工具的使用和选择等方面的需求做了解释。

5. 评估模型和标志指南

这部分内容为支持过程评估提出了一个评估模型的范例，此评估模型与第二部分所描述的参考模型相兼容，具体表述了任何兼容评估模型都期望具有的核心特征。该指南是以此评估模型中所包含的指示标志的形式给出的，这些指示标志可在过程改进程序中加以使用，还有助于评价和选择评估模型、方法或工具。采用这种方式并结合可靠的方法，有可能对过程能力做出一致的且可重复的评估。

6. 评估师能力指南

这部分提供了关于评估师进行软件过程评估的资格和准备的指南。它详细说明了一些可用于验证评估师胜任能力和相应的教育、培训和经验，还包括可能用于验证胜任能力和证实受教育程度、培训情况和经验的一些机制。



7. 过程改进应用指南

该部分提供了关于使用软件过程评估作为首要方法去理解一个企业软件过程的当前状态和使用评估结果去形成并优化改进方案方面的指南。该过程改进指南涉及过程改进综述、过程改进方法、文化问题、管理等专题,包括一个过程量度的总体框架。指南用于指导在连续循环里进行软件过程改进时把软件过程评估当成改进框架和方法的一部分使用。一个企业可以根据它的具体情况和需要从参考模型中选择所有的或部分软件过程用于评估或改进。

8. 确定供方能力应用指南

该部分内容为过程能力确定目的而进行的过程评审提供应用指南。它讲述了为对过程能力加以判断,应如何定义输入和如何运用评估结果。这不仅可直接用于对当前状况进行判断,而且也可以对复杂情况加以判断,例如对未来的预测。该部分中关于过程能力的判断方法不仅适合于任何希望确定其自身软件过程的过程能力的企业,也同样适应于对供应商的能力进行判断。

9. 词汇

本部分定义了 ISO/IEC TR 15504 整个技术报告中使用的术语。术语首先按字母排列顺序以便于参考,然后,术语再按逻辑类进行分类以便于理解(将相关的术语安排在一类)。

第 12 章 知识产权基础知识

12.1 知识产权的概念与特点

12.1.1 知识产权的概念

知识产权（也称为智慧财产权）是现代社会发展不可缺少的一种法律制度。知识产权保护制度为促进知识的积累与交流，丰富人们的精神生活，提高全民族的科学文化素质，推动经济的发展以及社会进步起到了极其重要的作用。知识产权是指人们基于自己的智力活动创造的成果和经营管理活动中的经验、知识而依法享有的权利。我国《民法通则》的规定，知识产权是指民事权利主体（公民、法人）基于创造性的智力成果。

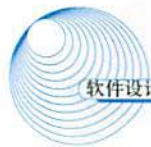
根据有关国际公约规定（世界知识产权组织公约第二条），知识产权的保护对象包括下列各项有关权利：

- 文学、艺术和科学作品；
- 表演艺术家的表演以及唱片和广播节目；
- 人类一切活动领域的发明；
- 科学发现；
- 工业品外观设计；
- 商标、服务标记以及商业名称和标志；
- 制止不正当竞争；
- 在工业、科学、文学艺术领域内由于智力创造活动而产生的一切其他权利。

在世界贸易组织协议的知识产权协议中，第一部分第一条所规定的知识产权范围，还包括“未披露过的信息专有权”，这主要是指工商业经营者所拥有的经营秘密和技术秘密等商业秘密。知识产权保护制度是随着科学技术的进步而不断发展和完善的。随着科学技术的迅速发展，知识产权保护对象的范围不断扩大，不断涌现新型的智力成果，如计算机软件、生物工程技术、遗传基因技术、植物新品种等，这些都是当今世界各国所公认的知识产权的保护对象。知识产权可分为工业产权和著作权两类。

1. 工业产权

根据《保护工业产权巴黎公约》第一条的规定，工业产权包括专利、实用新型、工业品外观设计、商标、服务标记、厂商名称、产地标记或原产地名称、制止不正当竞争等内容。此



外,商业秘密、微生物技术、遗传基因技术等也属于工业产权保护的对象。近年来,在一些国家可以通过申请专利,对计算机软件进行专利保护。对于工业产权保护的对象,可以分为“创造性成果权利”和“识别性标记权利”。发明、实用新型和工业品外观设计等属于创造性成果权利,它们都表现出比较明显的智力创造性。其中,发明和实用新型是利用自然规律做出的解决特定问题的新的技术方案,工业品外观设计是确定工业品外表的美学创作,完成人需要付出创造性劳动;商标、服务标记、厂商名称、产地标记或原产地名称以及我国反不正当竞争法第5条中规定的知名商品所特有的名称、包装、装潢等为识别性标记权利。

2. 著作权

著作权(也称为版权)是指作者对其创作的作品享有的人身权和财产权。人身权包括发表权、署名权、修改权和保护作品完整权等;财产权包括作品的使用权和获得报酬权,即以复制、表演、播放、展览、发行、摄制电影、电视、录像或者改编、翻译、注释、编辑等方式使用作品的权利,以及许可他人以上述方式使用作品并由此获得报酬的权利。关于著作权保护的对象,按照《保护文学艺术作品伯尔尼公约》第二条规定,包括:文学、科学和艺术领域内的一切作品,不论其表现形式或方式如何,诸如书籍、小册子和其他著作,讲课、演讲和其他同类性质作品,戏剧或音乐作品,舞蹈艺术作品和哑剧作品,配词或未配词的乐曲,电影作品以及使用电影摄影艺术类似的方法表现的作品,图画、油画、建筑、雕塑、雕刻和版画,摄影作品以及使用与摄影艺术类似的方法表现的作品,与地理、地形建筑或科学技术有关的示意图、地图、设计图、草图和立体作品等。

有些智力成果可以同时成为这两类知识产权保护的客体,例如,计算机软件和实用艺术品属著作权保护的同时,权利人还可以通过申请发明专利和外观设计专利,获得专利权,成为工业产权保护的对象。在美国和欧洲的一些国家,如果计算机软件自身包含技术构成,软件又能实现某方面的技术效果,如工业自动化控制等,则不应排除专利保护。按照世界知识产权组织公约,科学发现也被列为知识产权。我国《民法通则》第97条规定了科学发现权的法律地位,但很难将其归属工业产权或著作权。可见新产生的一些知识产权不一定就归为这两个类别,知识产权所保护的对象是依赖人类智力劳动创造的,特别是高科技创新产业迸发出的呈现千姿百态的知识财产,都属人类智力劳动的成果,法律都赋予它们民事权利。

12.1.2 知识产权的特点

1. 无形性

知识产权是一种无形财产权。知识产权的客体指的是智力创作性成果(也称为知识产品),是一种没有形体的精神财富。它是一种可以脱离其所有者而存在的无形信息,可以同时为多个主体所使用,在一定条件下不会因多个主体的使用而使该项知识财产自身遭受损耗或者消失。

2. 双重性

某些知识产权具有财产权和人身权双重性，例如著作权，其财产权属性主要体现在所有人享有的独占权以及许可他人使用而获得报酬的权利，所有人可以通过独自实施获得收益，也可以通过有偿许可他人实施获得收益，还可以像有形财产那样进行买卖或抵押；其人身权属性主要是指署名权等。有的知识产权具有单一的属性，例如，发现权只具有名誉权属性，而没有财产权属性；商业秘密只具有财产权属性，而没有人身权属性；专利权、商标权主要体现为财产权。

3. 确认性

无形的智力创作性成果不像有形财产那样直观可见，因此，智力创作性成果的财产权需要依法审查确认以得到法律保护。例如，我国的发明人所完成的发明，其实用新型或者外观设计已经具有价值和使用价值，但是，其完成人尚不能自动获得专利权，完成人必须依照专利法的有关规定，向国家专利局提出专利申请，专利局依照法定程序进行审查，申请符合专利法规定条件的，由专利局做出授予专利权的决定，颁发专利证书，只有当专利局发布授权公告后，其完成人才享有该项知识产权。又如，商标权的获得，大多数国家（包括中国）都实行注册制，只有向国家商标局提出注册申请，经审查核准注册后，才能获得商标权。文学艺术作品以及计算机软件的著作权虽然是自作品完成其权利即自动产生，但有些国家也要实行登记或标注版权标记后才能得到保护。

4. 独占性

由于智力成果具有可以同时被多个主体所使用的特点，因此，法律授予知识产权一种专有权，具有独占性。未经权利人许可，任何单位或个人不得使用，否则就构成侵权，应承担相应的法律责任。法律对各种知识产权都规定了一定的限制，但这些限制不影响其独占性特征。少数知识产权不具有独占性特征，例如技术秘密的所有人不能禁止第三人使用其独立开发完成的或者合法取得的相同技术秘密，可以说，商业秘密不具备完全的财产权属性。

5. 地域性

知识产权具有严格的地域性特点，即各国主管机关依照本国法律授予的知识产权，只能在其本国领域内受法律保护，例如中国专利局授予的专利权或中国商标局核准的商标专用权，只能在中国领域内受保护，其他国家则不给予保护，外国人在我国领域外使用中国专利局授权的发明专利，不侵犯我国专利权。所以，我国公民、法人完成的发明创造要想在外国受保护，必须在国外申请专利。著作权虽然自动产生，但它受地域限制，我国法律对外国人的作品并不都给予保护，只保护共同参加国际条约国家的公民作品。同样，公约的其他成员国也按照公约规定，对我国公



民和法人的作品给予保护。还有按照两国的双边协定,相互给予对方国民的作品保护。

6. 时间性

知识产权具有法定的保护期限,一旦保护期限届满,权利将自行终止,成为社会公众可以自由使用的知识。至于期限的长短,依各国的法律确定。例如,我国发明专利的保护期为 20 年,实用新型专利权和外观设计专利权的期限为 10 年,均自专利申请日起计算;我国公民的作品发表权的保护期为作者终生及其死亡后 50 年。我国商标权的保护期限自核准注册之日起 10 年内有效,但可以根据其所有人的需要无限地延长权利期限,在期限届满前 6 个月内申请续展注册,每次续展注册的有效期为 10 年,续展注册的次数不限。如果商标权人逾期不办理续展注册,其商标权也将终止。商业秘密受法律保护的期限是不确定的,该秘密一旦为公众所知悉,即成为公众可以自由使用的知识。

12.1.3 我国保护知识产权的法规

目前,我国已形成了比较完备的知识产权保护的法律体系,保护知识产权的法律主要有:

- 《中华人民共和国著作权法》;
- 《中华人民共和国专利法》;
- 《中华人民共和国继承法》;
- 《中华人民共和国公司法》;
- 《中华人民共和国合同法》;
- 《中华人民共和国商标法》;
- 《中华人民共和国产品质量法》;
- 《中华人民共和国反不正当竞争法》;
- 《中华人民共和国刑法》;
- 《中华人民共和国计算机信息系统安全保护条例》;
- 《中华人民共和国计算机软件保护条例》;
- 《中华人民共和国著作权法实施条例》等。

12.2 计算机软件著作权的主体与客体

12.2.1 计算机软件著作权的主体

计算机软件著作权的主体指享有著作权的人。根据著作权法和《计算机软件保护条例》的规定,计算机软件著作权的主体包括公民、法人和其他组织。著作权法和《计算机软件保护条

例》未规定对主体的行为能力限制,同时对外国人、无国籍人的主体资格,奉行“有条件”的国民待遇原则。

1. 公民

公民(即指自然人)通过以下途径取得软件著作权主体资格:

- 公民自行独立开发软件(软件开发者);
- 订立委托合同,委托他人开发软件,并约定软件著作权归自己享有;
- 通过转让途径取得软件著作权财产权主体资格(软件权利的受让者);
- 公民之间或与其他主体之间,对计算机软件进行合作开发而产生的公民群体或者公民与其他主体成为计算机软件作品的著作权人;
- 根据《继承法》的规定通过继承取得软件著作权财产权主体资格。

2. 法人

法人是具有民事权利能力和民事行为能力,依法独立享有民事权利和承担义务的组织。计算机软件的开发往往需要较大投资和较多的人员,法人则具有资金来源丰富和科技人才众多的优势,因而法人是计算机软件著作权的重要主体。法人取得计算机软件著作权主体资格一般通过以下途径:

- 由法人组织并提供创作物质条件所实施的开发,并由法人承担社会责任;
- 通过接受委托、转让等各种有效合同关系而取得著作权主体资格;
- 因计算机软件著作权主体(法人)发生变更而依法成为著作权主体。

3. 其他组织

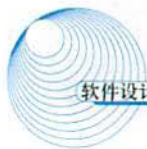
其他组织是指除法人以外的能够取得计算机软件著作权的其他民事主体。包括非法人单位、合作伙伴等。

12.2.2 计算机软件著作权的客体

计算机软件著作权的客体是指著作权法保护的计算机软件著作权的范围(受保护的客体)。根据《著作权法》第三条和《计算机软件保护条例》第二条的规定,著作权法保护的计算机软是指计算机程序及其有关文档。著作权法对计算机软件的保护是指计算机软件的著作权人或者其受让者依法享有著作权的各项权利。

1. 计算机程序

根据《计算机软件保护条例》第三条第一款的规定,计算机程序是指为了得到某种结果而



可以由计算机等具有信息处理能力的装置执行的代码化指令序列,或者可被自动转换成代码化指令序列的符号化语句序列。计算机程序包括源程序和目标程序,同一程序的源程序文本和目标程序文本视为同一软件作品。

2. 计算机软件的文档

根据《计算机软件保护条例》第三条第二款的规定,计算机程序的文档是指用自然语言或者形式化语言所编写的文字资料和图表,用来描述程序的内容、组成、设计、功能规格、开发情况、测试结果及使用方法等。文档一般以程序设计说明书、流程图、用户手册等表现。

12.3 计算机软件受著作权法保护的条件

《计算机软件保护条例》规定,依法受到保护的计算机软件作品必须符合下列条件:

1. 独立创作

受保护的软件必须由开发者独立开发创作,任何复制或抄袭他人开发的软件不能获得著作权。当然,软件的独创性不同于专利的创造性。程序的功能设计往往被认为是程序的思想概念,根据著作权法不保护思想概念的原则,任何人可以设计具有类似功能的另一件软件作品。但是如果用了他人软件作品的逻辑步骤的组合方式,则对他人软件构成侵权。

2. 可被感知

受著作权法保护的作品应当是作者创作思想在固定载体上的一种实际表达。如果作者的创作思想未表达出来不可以被感知,就不能得到著作权法的保护。因此,《计算机软件保护条例》规定,受保护的软件必须固定在某种有形物体上,例如固定在存储器、磁盘、磁带等设备上,也可以是其他的有形物,如纸张等。

3. 逻辑合理

逻辑判断功能是计算机系统的基本功能。因此,受著作权法保护的计算机软件作品必须具备合理的逻辑思想,并以正确的逻辑步骤表现出来,才能达到软件的设计功能。毫无逻辑性的计算机软件,不能计算出正确结果,也就毫无价值。

根据《计算机软件保护条例》第六条的规定,除计算机软件的程序和文档外,著作权法不保护计算机软件开发所用的思想、概念、发现、原理、算法、处理过程和运算方法。也就是说利用已有的上述内容开发软件,并不构成侵权。因为开发软件时所采用的思想、概念等均属计算机软件基本理论的范围,是设计开发软件不可或缺的理论依据,属于社会公有领域,不能为

个人专有。

12.4 计算机软件著作权的权利

12.4.1 计算机软件的著作人身权

《中华人民共和国著作权法》规定，软件作品享有两类权利，一类是软件著作权的人身权（精神权利）；另一类是软件著作权的财产权（经济权利）。《计算机软件保护条例》规定，软件著作权人享有发表权和开发者身份权，这两项权利与软件著作权人的人身权是不可分离的。

1. 发表权

发表权是指决定软件作品是否公之于众的权利，即指软件作品完成后，以复制、展示、发行或者翻译等方式使软件作品在一定数量不特定人的范围内公开。发表权具体内容包括软件作品发表的时间、发表的形式以及发表的地点等。

2. 开发者身份权（也称为署名权）

开发者身份权是指作者为表明身份在软件作品中署自己名字的权利。署名可有多种形式，既可以署作者的姓名，也可以署作者的笔名，或者作者自愿不署名。对一部作品来说，通过署名即可对作者的身份给予确认。我国著作权法规定，如无相反证明，在作品上署名的公民、法人或非法人单位为作者。因此，作品的署名对确认著作权的主体具有重要意义。开发者的身份权，不随软件开发者的消亡而丧失，且无时间限制。

12.4.2 计算机软件的著作财产权

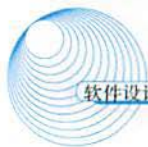
著作权中的财产权是指能够给著作权人带来经济利益的权利。这种经济利益的实现，要依靠著作权人对作品使用才能获得。财产权通常是指由软件著作权人控制和支配，并能够为权利人带来一定经济效益的权利。《计算机软件保护条例》规定，软件著作权人享有下述软件财产权。

（1）使用权：即在不损害社会公共利益的前提下，以复制、修改、发行、翻译、注释等方式合作软件的权利。

（2）复制权：即将软件作品制作一份或多份的行为。复制权就是版权所有人决定实施或不实施上述复制行为或者禁止他人复制其受保护作品的权利。

（3）修改权：即对软件进行增补、删节，或者改变指令、语句顺序等以提高、完善原软件作品的作法。修改权即指作者享有的修改或者授权他人修改软件作品的权利。

（4）发行权：发行指为满足公众的合理需求，通过出售、出租等方式向公众提供一定数量



的作品复制件。发行权即以出售或赠与方式向公众提供软件的原件或者复制件的权利。

(5) 翻译权: 翻译是指以不同于原软件作品的一种程序语言转换该作品原使用的程序语言, 而重现软件作品内容的创作。简单地说, 也就是指将原软件从一种程序语言转换成另一种程序语言的权利。

(6) 注释权: 软件作品的注释是指对软件作品中的程序语句进行解释, 以便更好地理解软件作品。注释权是指著作权人对自己的作品享有进行注释的权利。

(7) 信息网络传播权: 以有线或者无线信息网络方式向公众提供软件作品, 使公众可在其个人选定的时间和地点获得软件作品的权利。

(8) 出租权: 即有偿许可他人临时使用计算机软件的复制件的权利, 但是软件不是出租的主要标的的除外。

(9) 使用许可权和获得报酬权, 即许可他人以上述方式使用软件作品的权利(许可他人行使软件著作权中的财产权)和依照约定或者有关法律规定获得报酬的权利。

(10) 转让权, 即向他人转让软件的使用权和使用许可权的权利。软件著作权人可以全部或者部分转让软件著作权中的财产权。

12.4.3 软件合法持有人的权利

根据《计算机软件保护条例》的规定, 软件的合法复制品所有人, 享有下述权利:

- 根据使用的需要把软件装入计算机等能存储信息的装置内;
- 根据需要进行必要的复制;
- 为了防止复制品损坏而制作备份复制品。这些复制品不得通过任何方式提供给他人使用, 并在所有人丧失该合法复制品所有权时, 负责将备份复制品销毁;
- 为了把该软件用于实际的计算机应用环境或者改进其功能性能而进行必要的修改。但是, 除合同约定外, 未经该软件著作权人许可, 不得向任何第三方提供修改后的软件。

12.4.4 计算机软件著作权的行使

1. 软件经济权利的许可使用

软件经济权利的许可使用是指软件著作权人或权利合法受让者, 通过合同方式许可他人使用其软件, 并获得报酬的一种软件贸易形式。许可使用的方式可分为以下几种:

(1) 独占许可使用: 权利人通过书面合同授权, 被授权方可以根据合同规定的方式、条件和时间确定独占性, 权利人不得将软件使用权授予第三方, 权利人自己不能使用该软件。

(2) 独家许可使用: 权利人通过书面合同授权, 被授权方可以根据合同规定的方式、条件和时间确定独占性, 权利人不得将软件使用权授予第三方, 权利人自己可以使用该软件。

(3) 普通许可使用: 权利人通过书面合同授权, 被授权方可以根据合同规定的方式、条件和

时间确定独占性，权利人可以将软件使用权授予第三方，权利人自己可以使用该软件。

(4) 法定许可使用和强制许可使用：在法律特定的条款下，不经软件著作权人许可，使用其软件。

2. 软件经济权利的转让使用

软件经济权利的转让使用是指软件著作权人将其享有的软件著作权中的经济权利全部转移给他人。软件经济权利的转让将改变软件权利的归属，原始著作权人的主体地位随着转让活动的发生而丧失，软件著作权受让者成为新的著作权主体。《计算机软件保护条例》规定，软件著作权转让必须签订书面合同。同时，软件转让活动不能改变软件的保护期。转让方式包括出买、赠与、抵押、赔偿等，可以定期转让或者永久转让。

12.4.5 计算机软件著作权的保护期

根据《著作权法》和《计算机软件保护条例》的规定，计算机软件著作权的权利自软件开发完成之日起产生，保护期为 50 年。保护期满，除开发者身份权以外，其他权利终止。一旦计算机软件著作权超出保护期，软件就进入公有领域。计算机软件著作权人的单位终止和计算机软件著作权人的公民死亡均无合法继承人时，除开发者身份权以外，该软件的其他权利进入公有领域。软件进入公有领域后成为社会公共财富，公众可无偿使用。

12.5 计算机软件著作权的归属

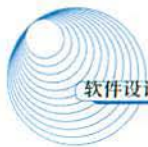
12.5.1 软件著作权归属的基本原则

我国著作权法对著作权的归属采取了“创作主义”原则，明确规定著作权属于作者，除非另有规定。《计算机软件保护条例》第九条规定“软件著作权属于软件开发者，本条例另有规定的情况除外。”这是我国计算机软件著作权归属的基本原则。

计算机软件开发者是计算机软件著作权的原始主体，也是享有权利最完整的主体。软件作品是开发者从事智力创作活动所取得的智力成果，是脑力劳动的结晶。其开发创作行为使开发者直接取得该计算机软件的著作权。因此，《计算机软件保护条例》第九条明确规定“软件著作权属于软件开发者”，即以软件开发的事实来确定著作权的归属，谁完成了计算机软件的开发工作，软件的著作权就归谁享有。

12.5.2 职务开发软件著作权的归属

职务软件作品是指公民在单位任职期间为执行本单位工作任务所开发的计算机软件作品。



《计算机软件保护条例》第十三条做出了明确的规定,即:公民在单位任职期间所开发的软件,如果是执行本职工作的结果,即针对本职工作中明确指定的开发目标所开发的;或者是从事本职工作活动所预见的结果或者自然的结果,则该软件的著作权属于该单位;或者主要使用了单位的专用设备、未公开的专门信息等物资技术条件所开发并由法人或者其他组织承担责任的软件。根据《计算机软件保护条例》规定,可以得出这样的结论,当公民作为某单位的雇员时,如其开发的软件属于执行本职工作的结果,该软件著作权应当归单位享有。若开发的软件不是执行本职工作的结果,其著作权就不属单位享有。如果该雇员主要使用了单位的设备,按照《计算机软件保护条例》第十三条第三款的规定,不能属于该雇员个人享有。

对于公民在非职务期间创作的计算机程序,其著作权属于某项软件作品的开发单位,还是从事直接创作开发软件作品的个人,可按照《计算机软件保护条例》第十三条规定的3条标准确定。

1. 所开发的软件作品不是执行其本职工作的结果

任何受雇于一个单位的人员,都会被安排在一定的工作岗位和分派相应的工作任务。完成分派的工作任务就是他的本职工作。本职工作的直接成果也就是其工作任务的不断完成。当然具体工作成果又会产生许多效益、产生范围更广的结果。但是该条标准指的是雇员本职工作最直接的成果。若雇员开发创作的软件不是执行本职工作的结果,则构成非职务计算机软件著作权的条件之一。

2. 开发的软件作品与开发者在单位中从事的工作内容无直接联系

如果该雇员在单位担任软件开发工作,引起争议的软件作品不能与其本职工作中明确指定的开发目标有关,软件作品的内容也不能与其本职工作所开发的软件的功能、逻辑思维和重要数据有关。雇员所开发的软件作品与其本职工作没有直接的关系,则构成非职务计算机软件著作权的第二个条件。

3. 开发的软件作品未使用单位的物质技术条件

开发创作软件作品所使用的物质技术条件,即开发软件作品所必需的设备、数据、资金和其他软件开发环境,不属于雇员所在的单位所有。没有使用受雇单位的任何物质技术条件构成非职务软件著作权的第三个条件。

雇员进行本职工作以外的软件开发创作,必须同时符合上述3个条件,才能算是非职务软件作品,雇员个人才享有软件著作权。常有软件开发符合前两个条件,但使用了单位的技术情报资料、计算机设备等物质技术条件的情况。处理此种情况较好的方法是对该软件著作权的归属应当由单位和雇员双方协商确定,如对于公民在非职务期间利用单位物质条件创作的与单位

业务范围无关的计算机程序，其著作权属于创作程序的作者，但作者许可第三人使用软件时，应当支付单位合理的物质条件使用费，如计算机机时费等。若通过协商不能解决，按上述3条标准做出界定。

12.5.3 合作开发软件著作权的归属

合作开发软件是指两个或两个以上公民、法人或其他组织订立协议，共同参加某项计算机软件的开发并分享软件著作权的形式。《计算机软件保护条例》第十条规定：“由两个以上的自然人、法人或者其他组织合作开发的软件，其著作权的归属由合作开发者签订书面合同约定。无书面合同或者合同未做明确约定，合作开发的软件可以分割使用的，开发者对各自开发的部分可以单独享有著作权；但是，行使著作权时，不得扩展到合作开发的软件整体的著作权。合作开发的软件不能分割使用的，其著作权由合作开发者共同享有，通过协商一致行使；如不能协商一致，又无正当理由，任何一方不得阻止他方行使除转让权以外的其他权利，但是所得收益应合理分配给所有合作开发者。”根据此规定，对合作开发软件著作权的归属应掌握以下4点：

(1) 由两个以上的单位、公民共同开发完成的软件属于合作开发的软件。对于合作开发的软件，其著作权的归属一般是由各合作开发者共同享有；但如果有软件著作权的协议，则按照协议确定软件著作权的归属。

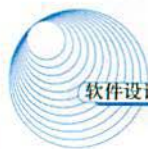
(2) 由于合作开发软件著作权是由两个以上单位或者个人共同享有，因而为了避免在软件著作权的行使中产生纠纷，规定“合作开发的软件，其著作权的归属由合作开发者签订书面合同约定”。

(3) 对于合作开发的软件著作权按以下规定执行：“无书面合同或者合同未作明确约定，合作开发的软件可以分割使用的，开发者对各自开发的部分可以单独享有著作权；但是，行使著作权时，不得扩展到合作开发的软件整体的著作权。合作开发的软件不能分割使用的，其著作权由合作开发者共同享有，通过协商一致行使；如不能协商一致，又无正当理由，任何一方不得阻止他方行使除转让权以外的其他权利，但是所得收益应合理分配给所有合作开发者。”

(4) 合作开发者对于软件著作权中的转让权不得单独行使。因为转让权的行使将涉及软件著作权权利主体的改变，所以软件的合作开发者在行使转让权时，必须与各合作开发者协商，在征得同意的情况下方能行使该项专有权利。

12.5.4 委托开发软件著作权的归属

委托开发的软件作品属于著作权法规定的委托软件作品。委托开发软件作品著作权关系的建立，一般由委托方与受委托方订立合同而成立。委托开发软件作品中，委托方的责任主要是提供资金、设备等物质条件，并不直接参与开发软件作品的创作开发活动。受托方的主要责任是根据委托合同规定的目标开发出符合条件的软件。关于委托开发软件著作权的归属，



《计算机软件保护条例》第十一条规定:“接受他人委托开发的软件,其著作权的归属由委托者与受委托者签订书面合同约定;无书面合同或者合同未作明确约定的,其著作权由受托人享有。”根据该条的规定,委托开发的软件著作权的归属按以下标准确定:

(1) 委托开发软件作品系根据委托方的要求,由委托方与受托方以合同确定的权利和义务的关系而进行开发的软件。因此,软件作品著作权归属应当作为合同的重要条款予以明确约定。对于当事人已经在合同中约定软件著作权归属关系的,如事后发生纠纷,软件著作权的归属仍应当根据委托开发软件合同来确定。

(2) 若在委托开发软件活动中,委托者与受委托者没有签订书面协议,或者在协议中未对软件著作权归属做出明确的约定,则软件著作权属于受委托者,即属于实际完成软件的开发者。

12.5.5 接受任务开发软件著作权的归属

根据社会经济发展的需要,对于一些涉及国家基础项目或者重点设施的计算机软件,往往采取由政府有关部门或上级单位下达任务方式,完成软件的开发工作。对于下达任务开发的软件著作权的归属关系,《计算机软件保护条例》第十二条做出了明确的规定:“由国家机关下达任务开发的软件,著作权的归属与行使由项目任务书或者合同规定;项目任务书或者合同中未作明确规定,软件著作权由接受任务的法人或者其他组织享有。”根据该规定,国家或上级下达任务开发的软件著作权归属应按以下两条标准确定:

(1) 下达任务开发的软件著作权的归属关系,首先应以项目任务书的规定或者双方的合同约定为准。

(2) 下达任务的项目任务书或者双方订立的合同中未对软件著作权归属做出明确的规定或者约定的,其软件著作权属于接受并实际完成开发软件任务的单位。

12.5.6 计算机软件著作权主体变更后软件著作权的归属

计算机软件著作权的主体,因一定的法律事实而发生变更。如作为软件著作权人的公民的死亡,单位的变更,软件著作权的转让以及人民法院对软件著作权的归属做出裁判等。软件著作权主体的变更必然引起软件著作权归属的变化。对此,《计算机软件保护条例》也做了一些规定。因计算机软件主体变更引起的权属变化有以下几种:

1. 公民继承的软件权利归属

《计算机软件保护条例》第十五条规定:“在软件著作权的保护期内,软件著作权的继承者可根据《中华人民共和国继承法》的有关规定,继承本条例第八条项规定的除署名权以外的其他权利。”按照该条的规定,软件著作权的合法继承人依法享有继承被继承人享有的软件著作权的使用权、使用许可权和获得报酬权等权利。继承权的取得、继承顺序等均按照继承法的规定。

定进行。

2. 单位变更后软件权利归属

《计算机软件保护条例》第十五条规定：“软件著作权属于法人或其他组织的，法人或其他组织变更、终止后，其著作权在本条例规定的保护期内由承受其权利义务的法人或其他组织享有。”按照该条的规定，作为软件著作权人的单位发生变更（如单位的合并、破产等），而其享有的软件著作权仍处在法定的保护期限内，可以由合法的权利承受单位享有原始著作权人所享有的各项权利。依法承受软件著作权的单位，成为该软件的后续著作权人，可在法定的条件下行使所承受的各项专有权利。一般认为，“各项权利”包括署名权等著作人身权在内全部权利。

3. 权利转让后软件著作权归属

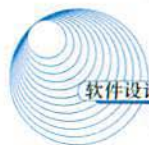
《计算机软件保护条例》第二十条规定：“转让软件著作权的，当事人应当订立书面合同。”计算机软件著作权财产权按照该条的规定发生转让后，必然引起著作权主体的变化，产生新的软件著作权归属关系。软件权利的转让应当根据我国有关法规以签订、执行书面合同的方式进行。软件权利的受让者可依法行使其享有的权利。

4. 司法判决、裁定引起的软件著作权归属问题

计算机软件著作权是公民、法人和其他组织享有的一项重要的民事权利。因而在民事权利行使、流转的过程中，难免发生涉及计算机软件著作权作为标的物的民事、经济关系，也难免发生争议和纠纷。争议和纠纷发生后由人民法院的民事判决、裁定而产生软件著作权主体的变更，引起软件著作权归属问题。因司法裁判引起软件著作权的归属问题主要有4类：一类是由人民法院对著作权归属纠纷中权利的最终归属做出司法裁判，从而变更了计算机软件著作权原有归属；第二类是计算机软件的著作权人为民事法律关系中的债务人（债务形成的原因可能多种多样，如合同关系或者损害赔偿关系等），人民法院将其软件著作财产权判归债权人享有抵债；第三类是人民法院做出民事判决判令软件著作权人履行民事给付义务，在判决生效后执行程序中，其无其他财产可供执行，将软件著作财产权执行给对方折抵债务；第四类是根据破产法的规定，软件著作权人被破产还债，软件著作财产权作为法律规定的破产财产构成的“其他财产权利”，作为破产财产由人民法院判决分配。

5. 保护期限届满权利丧失

软件著作权的法定保护期限可以确定计算机软件主体能否依法变更。如果软件著作权已过保护期，该软件进入公有领域，便丧失了专有权，也就没有必要改变权利主体了。根据软件保护条例的规定，计算机软件著作权主体变更必须在该软件著作权的保护期限内进行，转让活动的发



生不改变该软件著作权的保护期。这也就是说,转让活动也不能延长该软件著作权的保护期限。

12.6 计算机软件著作权侵权的鉴别

侵犯计算机软件著作权的违法行为的鉴别,主要依靠保护知识产权的相关法律来判断。违反著作权、计算机软件保护条例等法律禁止的行为,便是侵犯计算机著作权的违法行为,这是鉴别违法行为的本质原则。对于法律规定不禁止,也不违反相关法律基本原则的行为,不认为是违法行为。在法律无明文具体条款规定的情况下,违背著作权法和计算机软件保护条例等法律的基本原则,以及社会主义公共生活准则和社会善良风俗的行为,也应该视为违法行为。在一般情况下,损害他人著作财产权或人身权的行为,总是违法行为。

12.6.1 计算机软件著作权侵权行为

根据《计算机软件保护条例》第二十三条的规定,凡是行为人主观上具有故意或者过失对著作权法和计算机软件保护条例保护的计算机软件著作权人身权和财产权实施侵害行为的,都构成计算机软件的侵权行为。该条规定的侵犯计算机软件著作权的情况,是认定软件著作权侵权行为的法律依据。计算机软件著作权侵权行为主要有以下几种:

1. 未经软件著作权人的同意而发表或者登记其软件作品

软件著作权人享有对软件作品公开发表权,未经允许著作权人以外的任何其他人都无权擅自发表特定的软件作品。如果实施这种行为,就构成侵犯著作权人的发表权。

2. 将他人开发的软件当作自己的作品发表或者登记

此种行为主要侵犯了软件著作权的开发者身份权和署名权。侵权行为人欺世盗名,剽窃软件开发者的劳动成果,将他人开发的软件作品假冒为自己的作品而署名发表。只要行为人实施了这种行为,不管其发表该作品是否经过软件著作权人的同意,都构成侵权。

3. 未经合作者的同意将与他人合作开发的软件当作自己独立完成的作品发表或者登记

此种侵权行为发生在软件作品的合作开发者之间。作为合作开发的软件,软件作品的开发者身份为全体开发者,软件作品的发表权也应由全体开发者共同行使。如果未经其他开发者同意,又将合作开发的软件当作自己的独创作品发表,即构成本条规定的侵权行为。

4. 在他人开发的软件上署名或者更改他人开发的软件上的署名

这种行为是指在他人开发的软件作品上添加自己的署名,或者替代软件开发者的署名以及或

者将软件作品上开发者的署名进行更改的行为。这种行为侵犯了软件著作人的开发者身份权及署名权。此种行为与第2条规定行为的区别主要是对已发表的软件作品实施的行为。

5. 未经软件著作权人或者其合法受让者的许可, 修改、翻译其软件作品

此种行为是侵犯了著作权人或其合法受让者的使用权中的修改权、翻译权。对不同版本计算机软件, 新版本往往是旧版本的提高和改善。这种提高和改善实质上是对原软件作品的修改、演绎。此种行为应征得软件作品原版本著作权人的同意, 否则构成侵权。如果征得软件作品著作权人的同意, 因修改和改善新增加的部分, 创作者应享有著作权。

6. 未经软件著作权人或其合法受让者的许可, 复制或部分复制其软件作品

此种行为侵犯了著作权人或其合法受让者的使用权中的复制权。计算机软件的复制权是计算机软件最重要的著作财产权, 也是通常计算机软件侵权行为的对象。这是由于软件载体价格相对低廉, 复制软件简单易行效率极高, 而销售非法复制的软件即可获得高额利润。因此, 复制是常见的侵权行为, 是防止和打击的主要对象。当软件著作权经当事人的约定合法转让给转让者以后, 软件开发未经允许不得复制该软件, 否则也构成本条规定的侵权行为。

7. 未经软件著作权人及其合法受让者同意, 向公众发行、出租其软件的复制品

此种行为侵犯了著作权人或其合法受让者的发行权与出租权。

8. 未经软件著作权人或其合法受让者同意, 向任何第三方办理软件权利许可或转让事宜

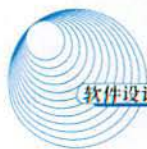
这种行为侵犯了软件著作权人或其合法受让者的使用许可权和转让权。

9. 未经软件著作权人及其合法受让者同意, 通过信息网络传播著作权人的软件

这种行为侵犯了软件著作权人或其合法受让者的信息网络传播权。

10. 侵犯计算机软件著作权存在着共同侵权行为

二人以上共同实施《计算机软件保护条例》第二十三条和第二十四条规定的侵权行为, 构成共同侵权行为。对行为人并没有实施《计算机软件保护条例》第二十三条和第二十四条规定的行为, 但实施了向侵权行为人进行侵权活动提供设备、场所或解密软件, 或者为侵权复制品提供仓储、运输条件等行为, 构成共同侵权应当在行为人之间具有共同故意或过失。其构成的要件有两个: 一是行为人的过错是共同的, 而不论行为人的行为在整个侵权行为过程中所起的作用如何; 二是行为人主观上要有故意或过失的过错。如果这个要件具备, 各个行为人实施的侵权行为虽然各不相同, 也同样构成共同侵权。两个要件如果缺乏一个, 不构成共同的侵权, 或者



是不构成任何侵权。

12.6.2 不构成计算机软件侵权的合理使用行为

我国《计算机软件保护条例》第八条第四项和第十六条规定,获得使用权或使用许可权(视合同条款)后,可以对软件进行复制而无需通知著作权人,亦不构成侵权。对于合法持有软件复制品的单位、公民在不经著作权人同意的情况下,亦享有复制与修改权。合法持有软件复制品的单位、公民,在不经软件著作权人同意的情况下,可以根据自己使用的需要将软件装入计算机,为了存档也可以制作备份复制品,为了把软件用于实际的计算机环境或者改进其功能时也可以进行必要的修改,但是备份制品和修改后的文本不能以任何方式提供给他人,超过以上权利,即视为侵权行为。区分合理使用与非合理使用的判别标准一般有:

- 软件作品是否合法取得。这是合理使用的基础。
- 使用目的是非商业营业性,如果使用的目的是为商业性营利,就不属合理使用的范围。
- 合理使用一般为少量的使用,所谓少量的界限根据其使用的目的以行业惯例和人们一般常识综合确定。超过通常被认为的少量界限,即可被认为不属合理使用。

我国《计算机软件保护条例》第十七规定:“为了学习和研究软件内含的设计思想和原理,通过安装、显示、传输或者存储软件的方式使用软件的,可以不经软件著作权人许可,不向其支付报酬。”

12.6.3 计算机著作权软件侵权的识别

计算机软件明显区别于其他著作权法保护的客体,它具有以下特点:

1. 技术性

计算机软件的技术性是指其创作开发的高技术性。具有一定规模的软件的创作开发,一般开发难度大、周期长、投资高,需要良好组织,严密管理且各方面人员配合协作,借助现代化高技术和高科技工具生产创作的。

2. 依赖性

计算机程序的依赖性是指人们对其的感知依赖于计算机的特性。著作权保护的其他作品一般都可以依赖人的感觉器官所直接感知。但计算机程序则不能被人们所直接感知,它的内容只能依赖计算机等专用设备才能被充分表现出来,才能被人们所感知。

3. 多样性

计算机程序的多样性是指计算机程序表达的多样性。计算机程序的表达较著作权法保护的

其他对象特殊,其既能以源代码表达,还可以以目标代码和微码等表达,表达形式多样。计算机程序表达的存储媒体也多种多样,同一种程序分别可以被存储在纸张、磁盘、磁带、光盘和集成电路上等。计算机程序的载体大多数精巧灵便。此外,计算机程序的内容与表达难于严格区别界定。

4. 运行性

计算机程序的运行性是指计算机程序功能的运行性。计算机程序不同于一般的文字作品,它主要的功能在于使用。也就是说计算机程序的功能只能通过对程序的使用、运行才能充分体现出来。计算机程序采用数字化形式存储、转换,复制品与原作品一般无明显区别。

根据计算机软件的特点,对计算机软件侵权行为的识别可以将发生争议的某一计算机程序与比照物(权利明确的正版计算机程序)进行对比和鉴别,从两个软件的相似性或完全相同来判断,做出侵权认定。软件作品常常表现为计算机程序的不唯一性,两个运行结果相同的计算机程序,或者两个计算机软件的源代码程序不相似或不完全相似,前者不一定构成侵权,而后者不一定不构成侵权。识别侵权(盗版)软件可以采取以下简单方法和步骤:

(1) 对被识别的软件与正版软件直接进行目录、文件名对比以及部分内容对比。如果这两者完全一致,就可以初步认定没有手续而拥有该软件并进行使用或销售者为软件侵权者;如果并非完全一致,而是大部分一致,就需要继续进行识别。

(2) 对两套软件同时或先后进行安装,观察其安装过程中的屏幕显示,包括软件信息以及使用功能键后的屏幕显示等是否相同。如果相同,则可认定这两套软件的安装手段一致。

(3) 对其安装成功后的目录,以及各文件信息进行对比。观察屏幕显示,通过表观现象,包括文件名、文件长度、文件建立(或修改)的时间、文件属性4个部分进行识别。一般情况下,侵权销售者经过修改的盗版软件与正版软件可能不完全一致,只是修改少数部分,而绝大部分文件的表观现象都是一致的。

(4) 对盗版软件与正版软件的使用过程进行对比,即对使用过程中的屏幕显示、功能、功能键、使用方法等进行对比。

(5) 对盗版软件与正版软件源程序进行对比。软件生产开发者一般不向外公布其源程序,计算机软件以源程序方式向外传播的情况较少,大多是以目标程序的形式向外传播。

12.7 软件著作权侵权的法律责任

当侵权人侵害他人的著作权财产权或著作人身权,造成权利人财产上的或非财产的损失,侵权人不履行赔偿义务,法律即强制侵权人承担赔偿责任的民事责任。



1. 民事责任

侵犯计算机著作权以及有关权益的民事责任是指公民、法人或其他组织因侵犯著作权发生的后果依法应承担的法律责任。我国《计算机软件保护条例》第二十三条规定了侵犯计算机著作权的民事责任,即侵犯著作权或者与著作权有关的权利的,侵权人应当按照权利人的实际损失给予赔偿;实际损失难以计算的,可以按照侵权人的违法所得给予赔偿。赔偿数额还应当包括权利人为制止侵权行为所支付的合理开支。权利人的实际损失或者侵权人的违法所得不能确定的,由人民法院根据侵权行为的情节,判决给予五十万元以下的赔偿。有下列侵权行为的,应当根据情况,承担停止侵害、消除影响、公开赔礼道歉、赔偿损失等民事责任:

- 未经软件著作权人许可发表或者登记其软件的。
- 将他人软件当作自己的软件发表或者登记的。
- 未经合作者许可,与他人合作开发的软件当作自己单独完成的作品发表或者登记的。
- 在他人软件上署名或者涂改他人软件上的署名的。
- 未经软件著作权人许可,修改、翻译其软件的。
- 其他侵犯软件著作权的行为。

2. 行政责任

我国《计算机软件保护条例》第二十四条规定了相应的行政责任,即对侵犯软件著作权行为,著作权行政管理部门应当责令停止违法行为,没收违法所得,没收、销毁侵权复制品,并可处以每件一百元或者货值金额二至五倍的罚款。有下列侵权行为的,应当根据情况,承担停止侵害、消除影响、公开赔礼道歉、赔偿损失等行政责任:

- 复制或者部分复制著作权人软件的。
- 向公众发行、出租、通过信息网络传播著作权人的软件的。
- 故意避开或者破坏著作权人为保护其软件而采取得技术措施的。
- 故意删除或者改变软件权利管理电子信息的。
- 许可他人行使或者转让著作权人的软件著作权的。

3. 刑事责任

侵权行为触犯刑律的,侵权者应当承担刑事责任。我国《刑法》第二百一十七条、第二百一十八条和第二百二十条的规定,构成侵犯著作权罪、销售侵权复制品罪的,由司法机关追究刑事责任。

12.8 计算机软件的商业秘密权

12.8.1 商业秘密的概念

关于商业秘密的法律保护,各国采取不同的立法例,有的制定单行法,有的规定在反不正当竞争法中,有的适用一般侵权行为法。我国反不正当竞争法规定了商业秘密的保护问题。

1. 商业秘密的定义

《反不正当竞争》中商业秘密定义是指“不为公众所知悉、能为权利人带来经济利益、具有实用性并经权利人采取保密措施的技术信息和经营信息。”经营秘密和技术秘密是商业秘密的基本内容。经营秘密,即未公开的经营信息,是指与生产经营销售活动有关的经营方法、管理方法、产销策略、货源情报、客户名单、标底和标书内容等到专有知识。技术秘密,即未公开的技术信息,是指与产品生产和制造有关的技术诀窍、生产方案、工艺流程、设计图纸、化学配方、技术情报等专有知识。

2. 商业秘密的构成条件

商业秘密的构成条件是:商业秘密必须具有未公开性,即不为公众所知悉;商业秘密必须具有实用性,即能为权利人带来经济效益;商业秘密必须具有保密性,即采取了保密措施。

3. 商业秘密权

商业秘密是一种无形的信息财产。与有形财产相区别,商业秘密不占据空间,不易为权利人所控制,不发生有形损耗,其权利是一种无形财产权。商业秘密的权利人与有形财产所有权人一样,依法享有占有、使用和收益的权利,即有权对商业秘密进行控制与管理,防止他人采取不正当手段获取与使用;有权依法使用自己的商业秘密,而不受他人干涉;有权通过自己使用或者许可他人使用以至转让所有权,从而取得相应的经济利益;有权处分自己的商业秘密,包括放弃占有、无偿公开、赠与或转让等。

4. 商业秘密的丧失

一项商业秘密受到法律保护的依据,是必须具备上述构成商业秘密的3个条件,当缺少上述3个条件之一都会造成商业秘密丧失保护。

5. 计算机软件与商业秘密

《反不正当竞争法》保护计算机软件,是以计算机软件中是否包含着“商业秘密”为必要



条件的。而计算机软件是人类知识、智慧、经验和创造性劳动的成果,本身就具有商业秘密的特征,即包含着技术秘密和经营秘密。即使是软件尚未开发完成,在软件开发中所形成的知识内容也可构成商业秘密。

12.8.2 计算机软件商业秘密的侵权

侵犯商业秘密,是指行为人(负有约定的保密义务的合同当事人、实施侵权行为的第三人、侵犯本单位商业秘密的行为人)未经权利人(商业秘密的合法控制人)的许可,以非法手段(包括直接从权利人那里窃取商业秘密并加以公开或使用、通过第三人窃取权利人的商业秘密并加以公开或使用)获取计算机软件商业秘密并加以公开或使用的行为。根据我国《反不正当竞争法》第十条的规定,侵犯计算机软件商业秘密的具体表现形式主要有:

(1) 盗窃、利诱、胁迫或其他不正当手段获取权利人的计算机软件商业秘密。盗窃商业秘密,包括单位内部人员盗窃、外部人员盗窃、内外勾结盗窃等手段;以利诱手段获取商业秘密,通常指行为人向掌握商业秘密的人员提供财物或其他优惠条件,诱使其向行为人提供商业秘密;以胁迫手段获取商业秘密,是指行为人采取威胁、强迫手段,使他人受强制的情况下提供商业秘密;以其他不正当手段获取商业秘密。

(2) 披露、使用或允许他人使用以不正当手段获取的计算机软件商业秘密。披露是指将权利人的商业秘密向第三人透露或向不特定的其他人公开,使其失去秘密价值;使用或允许他人使用是指非法使用他人商业秘密的具体情形。如果以非法手段获取商业秘密的行为人,如果将该秘密再行披露或使用,即构成双重的侵权;倘若第三人从侵权人那里获悉了商业秘密而将秘密披露或使用,同样构成侵权。

(3) 违反约定或违反权利人有关保守商业秘密的要求,披露、使用或允许他人使用其所掌握的计算机软件商业秘密。合法掌握计算机软件商业秘密的人,可能是与权利人有合同关系的对方当事人,也可能是权利人的单位工作人员或其他知情人,他们违反合同约定或单位规定的保密义务,将其所掌握的商业秘密擅自公开,或自己使用,或许可他人使用,即构成侵犯商业秘密。

(4) 第三人在明知或应知前述违法行为的情况下,仍然从侵权人那里获取、使用或披露他人的计算机软件商业秘密。这是一种间接的侵权行为。

12.8.3 计算机软件商业秘密侵权的法律责任

根据我国《反不正当竞争法》和《刑法》的规定,计算机软件商业秘密的侵权者将承担行政责任、民事责任以及刑事责任:

1. 侵权者的行政责任

我国《反不正当竞争法》第二十五条规定了相应的行政责任,即对侵犯商业秘密的行为,

监督检查部门应当责令停止违法行为,而后可以根据侵权的情节依法处以 1 万元以上 20 万元以下的罚款。

2. 侵权者的民事责任

计算机软件商业秘密的侵权者的侵权行为对权利人的经营造成经济上的损失时,侵权者应当承担经济损害赔偿的民事责任。我国《反不正当竞争法》第二十条规定了侵犯商业秘密的民事责任,即经营者违反该法规定,给被侵害的经营者造成损害的,应当承担损害赔偿责任。被侵害的经营者的合法权益受到损害的,可以向人民法院提起诉讼。

3. 侵权者的刑事责任

侵权者以盗窃、利诱、胁迫或其他不正当手段获取权利人的计算机软件商业秘密;披露、使用或允许他人使用以不正当手段获取的计算机软件商业秘密;违反约定或违反权利人有关保守商业秘密的要求,披露、使用或允许他人使用其所掌握的计算机软件商业秘密,其侵权行为对权利人造成重大损害的,侵权者应当承担刑事责任。我国《刑法》第二百一十九条规定了侵犯商业秘密罪,即实施侵犯商业秘密行为,给商业秘密的权利人造成重大损失的,处 3 年以下有期徒刑或者拘役,并处或者单处罚金;造成特别严重后果的,处 3 年以上 7 年以下有期徒刑,并处罚金。

12.9 专利权概述

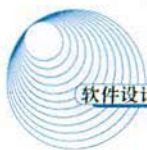
12.9.1 专利权的保护对象与特征

发明创造是产生专利权的基础。发明创造是指发明、实用新型和外观设计,是我国专利法主要保护的客体。我国《专利法实施细则》第二条第一款规定:“专利法所称的发明,是指对产品、方法或者其改进所提出的技术方案。”实用新型(也称小发明)则因国而异,我国《专利法实施细则》第二条第二款规定:“实用新型是指对产品的形状、构造或者其组合所提出的新的技术方案。”外观设计是指对产品的形状、图案、色彩或者它们的结合所做出的富有美感的并适于工业应用的新设计。

专利的发明创造是无形的智力创造性成果,不像有形财产那样直观可见,必须经专利主管机关依照法定程序审查确定,在未经审批以前,任何一项发明创造都不得成为专利。

下列各项属于专利法不适用的对象,因此不授予专利权:

- (1) 对违反国家法律、社会公德或者妨害公共利益的发明创造。
- (2) 科学发现,即人们通过自己的智力劳动对客观世界已经存在的但未揭示出来的规律、



性质和现象等的认识。

(3) 智力活动的规则和方法,即人们进行推理、分析、判断、运算、处理、记忆等思维活动的规则和方法。

(4) 病的诊断和治疗方法。即以活的人或者动物为实施对象,并以防病治病为目的,是医护人员的经验体现,而且因被诊断和治疗的对象不同而有区别,不能在工业上应用,不具有实用性。

(5) 动物和植物品种。但是动物和植物品种的生产方法,可以依照专利法规定授予专利权。

(6) 用原子核变换方法获得的物质,即用核裂变或核聚变方法获得的单质或化合物。

12.9.2 授予专利权的条件

授予专利权的条件是指一项发明创造获得专利权应当具备的实质性条件。一项发明或者实用新型获得专利权的实质条件为新颖性、创造性和实用性。

1. 新颖性

新颖性是指在申请日以前没有同样的发明或实用新型在国内外出版物公开发表过,在国内公开使用过或以其他方式为公众所知,也没有同样的发明或实用新型由他人向专利局提出过申请并且记载在申请日以后公布的专利申请文件中。在某些特殊情况下,尽管申请专利的发明或者实用新型在申请日或者优先权日前公开,但在一定的期限内提出专利申请的,仍然具有新颖性。我国专利法规定申请专利的发明创造在申请日以前6个月内,有下列情况之一的,不丧失新颖性:

- 在中国政府主办或者承认的国际展览会上首次展出的。
- 在规定的学术会议或者技术会议首次发表的。
- 他人未经申请人同意而泄露其内容的。

2. 创造性

创造性是指同申请日以前已有的技术相比,该发明有突出的实质性特点和显著的进步,该实用新型有实质性特点和进步。例如,申请专利的发明解决了人们渴望解决但一直没有解决的技术难题;申请专利的发明克服了技术偏见;申请专利的发明取得了意想不到的技术效果;申请专利的发明在商业上获得成功。一项发明专利是否具有创造性,前提是该项发明具备新颖性。

3. 实用性

实用性是指该发明或者实用新型能够制造或者使用,并且能够产生积极的效果。即不造成环境污染、能源或者资源的严重浪费,损害人体健康。如果申请专利的发明或者实用新型缺乏技术手段、申请专利的技术方案违背自然规律、利用独一无二自然条件所完成的技术方案,则

不具有实用性。

我国专利法的规定,外观设计获得专利权的实质条件为新颖性和美观性。新颖性是指申请专利的外观设计与其申请日以前已经在国内外出版物上公开发表的外观设计不相同或者不近似;与其申请日前已在国内公开使用过的外观设计不相同或者不近似。美观性是指外观设计被使用在产品上时能使人产生一种美感,增加产品对消费者的吸引力。

12.9.3 专利的申请

1. 专利申请权

公民、法人或者其他组织依据法律规定或者合同约定享有的就发明创造向专利局提出专利申请的权利(专利申请权)。一项发明创造产生的专利申请权归谁所有,主要有由法律直接规定的情况和依合同约定的情况。专利申请权可以转让,不论专利申请权在那一个时间段转让,原专利申请人便因此丧失专利申请权,由受让人获得相应的专利申请权。专利申请权可以被继承或赠与。如专利申请人死亡后,其依法享有的专利申请权可以作为遗产,由其合法继承人继承。

2. 专利申请人

专利申请人是指对某项发明创造依法律规定或者合同约定享有专利申请权的公民、法人或者其他组织。专利申请人包括:职务发明创造的单位;非职务发明创造的专利申请人完成发明创造的发明人或者设计人;共同发明创造的专利申请人是共同发明人或者设计人。或者其所属单位;委托发明创造的专利申请人合同约定的人;受让人。

3. 专利申请的原则

专利申请人及其代理人在办理各种手续时都应当采用书面形式。一份专利申请文件只能就一项发明创造提出专利申请,即“一份申请一项发明”原则。两个或者两个以上的人分别就同样的发明创造申请专利的,专利权授给最先申请人。

4. 专利申请文件

发明或者实用新型申请文件包括请求书、说明书、说明书摘要、权利要求书。外观设计专利申请文件包括请求书、图片或照片。

5. 专利申请日

专利申请日(也称关键日),它是专利局或者专利局指定的专利申请受理代办处收到完整



专利申请文件的日期。如果申请文件是邮寄的,以寄出的邮戳日为申请日。

6. 专利申请的审批

专利局收到发明专利申请后,一个必要程序是初步审查,经初步审查认为符合本法要求的,自申请日起满18个月,即行公布(公布申请),专利局可根据申请人的请求,早日公布其申请。自申请日起3年内,专利局可以根据申请人随时提出的请求,对其申请进行实质审查。实质审查是专利局对申请专利的发明的新颖性、创造性和实用性等依法进行审查的法定程序。

我国专利法规定:“实用新型和外观设计专利申请经初步审查没有发现驳回理由的,专利局应当做出授予实用新型专利权或者外观设计专利权的决定,发给相应的专利证书,并予以登记和公布。”由此规定可知,对实用新型和外观设计专利申请只进行初步审查,不进行实质审查。

7. 申请权的丧失与恢复

专利法及其实施细则有许多条款规定,如果申请人在法定期间或者专利局所指定的期限内未办理相应的手续或者没有提交有关文件,其申请就被视为撤回或者丧失提出某项请求的权利,或者导致有关权利终止后果。因耽误期限而丧失权利之后,可以在自障碍消除后2个月内,最迟自法定期限或者指定期限届满后2年内或者自收到专利局通知之日起2个月内,请求恢复其权利。

12.9.4 专利权行使

1. 专利权的归属

根据《中华人民共和国专利法》的规定,执行本单位的任务或者主要是利用本单位的物质条件所完成的职务发明创造,申请专利的权利属于该单位。申请被批准后,专利权归该单位持有(单位为专利权人)。执行本单位的任务所完成的职务发明创造是指:

- 在本职工作中做出的发明创造。
- 履行本单位交付的本职工作之外的任务所做出的发明创造。
- 工作变动(辞职、退休或者调离)后短期内做出的,与其在原单位承担的本职工作或者原单位分配的任务有关的发明创造。

本单位的物质技术条件包括本单位的资金、设备、零部件、原材料或者不对外公开的技术资料等。

非职务发明创造,申请专利的权利属于发明人或者设计人;在中国境内的外资企业和中外合资经营企业的工作人员完成的职务发明创造,申请专利的权利属于该企业,申请被批准后,专利权归申请的企业或者个人所有;两个以上单位协作或者一个单位接受其他单位委托的研究、设计任务所完成的发明创造,除另有协议的以外,申请专利的权利属于完成或者共同完成

的单位,申请被批准后,专利权归申请的单位所有或者持有。

2. 专利权人的权利

专利权是一种具有财产权属性的独占权以及由其衍生出来和相应处分权。专利权人的权利包括:独占实施权、转让权、实施许可权、放弃权、标记权等。专利权人有缴纳专利年费(也称专利维持费)和实际实施已获专利的发明创造的两项基本义务。

专利权人通过专利实施许可合同将其依法取得的对某项发明创造的实施权转移给非专利权人行使。任何单位或者个人实施他人专利的,除《中华人民共和国专利法》第十四条规定的以外,都必须与专利权人订立书面实施许可合同,向专利权人支付专利使用费。被许可人无权允许合同规定以外的任何单位或者个人实施该专利。专利实施许可的种类包括独占许可、独家许可、普通许可、部分许可。

12.9.5 专利权的限制

根据《中华人民共和国专利法》的规定,发明专利权的保护期限为自申请日起 20 年;实用新型专利权和外观设计专利权的保护期限为自申请日起 10 年。发明创造专利权的法律效力所及的范围为:

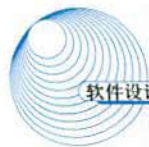
- 发明或者实用新型专利权的保护范围以其权利要求的内容为准,说明书及附图可以用于解释权利要求。
- 外观设计专利权的保护范围以表示在图片或者照片中的该外观设计专利产品为准。

公告授予专利权后,任何单位或个人认为该专利权的授予不符合专利法规定条件的,可以向专利复审委员会提出宣告该专利权无效的请求。专利复审委员会对这种请求进行审查,做出宣告专利权无效或维持专利权的决定。我国专利法规定提出无效宣告请求的时间(启动无效宣告程序的时间)始于“自专利局公告授予专利权之日起”。

专利权因某种法律事实的发生而导致其效力消灭的情形称为专利权终止。导致专利权终止的法律事实有:

- 保护期限届满。
- 在专利权保护期限届满前,专利权人以书面形式向专利局声明放弃专利权。
- 在专利权的保护期限内,专利权人没有按照法律的规定交年费。专利权终止日应为上一年度期满日。

专利法允许第三人在某些特殊情况下,可以不经专利权人许可而实施其专利,且其实施行为并不构成侵权的一种法律制度。专利权限制的种类包括强制许可、不视为侵犯专利权的行为、国家计划许可。



12.9.6 专利侵权行为

专利侵权行为是指在专利权的有效期限内,任何单位或者个人在未经专利权人许可,也没有其他法定事由的情况下,擅自以营利为目的实施专利的行为。专利侵权行为主要包括:

- 为生产经营目的制造、使用、销售其专利产品,或者使用其专利方法以及使用、销售依照该专利方法直接获得的产品。
- 为生产经营目的制造、销售其外观设计专利产品。
- 进口依照其专利方法直接获得的产品。
- 产品的包装上标明专利标记和专利号。
- 将非专利产品冒充专利产品的或者将非专利方法冒充专利方法等。

对未经专利权人许可,实施其专利的侵权行为,专利权人或者利害关系人可以请求专利管理机关处理。在专利侵权纠纷发生后,专利权人或者利害关系人既可以请求专利管理机关处理,又可以请求人民法院审理。侵犯专利权的诉讼时效为2年,自专利权人或者利害关系人知道或者应当知道侵权行为之日起计算。如果诉讼时效期间届满,专利权人或者利害关系人不能再请求人民法院保护,同时也不能再向专利管理机关请求保护。

12.10 企业知识产权的保护

从事计算机软件科研开发、技术贸易的高新技术企业通常需要不断开发和积累本企业的技术成果,以保持企业技术的优势,需要保护好企业拥有的知识产权,以能够利用本企业的知识产权为自身带来经济效益,否则将会影响企业的生存和发展。高新技术企业大都是以知识创新开发产品,当知识产品进入市场后,则完全依赖于对其知识产权的保护,如果没有保护或保护不好,将影响企业大规模的知识产权开发投资,企业也不可能更好的生存与发展。知识产权是一种无形的产权,是企业的重要财富,应当把保护软件知识产权作为现代企业制度的一项基本内容。

12.10.1 知识产权管理

企业往往会面临软件知识产权的权利归属、软件人才的管理、软件技术的保密以及软件成果的有效利用等一系列的问题,这些问题的解决不可能仅仅依靠企业的某些领导的行政干预,更有效的途径是依靠企业自身的管理制度才能解决问题。软件的权利归属不清、人才外流、管理不善而导致软件知识产权泄密和流失的问题,软件的研发及经营活动中不规范的问题,只有通过企业自身的知识产权管理制度来有效地控制和管理,以保护企业有关知识产权。企业知识产权管理制度,不仅应当形成激励机制,还应当建立明确的约束机制,并将可能发生的问题防

患于未然。通过制度管理,掌握主动权,以确保企业生存与发展,在技术发展处于不败的地位。

专利权、商标权、著作权等知识产权是企业的经营资源,应建立承担企业知识产权管理的知识产权机构,有效地管理和利用知识产权。参与到软件立项、开发、销售及售后服务的每一环节,把握住企业的技术开发动向以及其他公司的技术开发动向,以知识产权的角度向开发人员、研究人员、管理人员提出建议,并为对外事务提供法律保障。

12.10.2 知识产权的保护和利用

目前,计算机技术和软件技术的知识产权法律保护已形成以《著作权法》保护为主,《著作权法》(包括《计算机软件保护条例》)、《专利法》、《商标法》、《反不正当竞争法》《合同法》实施交叉和重叠保护为辅的趋势。例如,源程序及设计文档作为软件的表现形式用著作权法保护,同时作为技术秘密又受《反不正当竞争法》的保护。由于软件具有技术含量高的特点,使得对软件法律保护成为一种综合性的保护,对于企业来说,仅依靠某项法律或法规是不能解决软件的所有知识产权问题的,应在保护企业计算机软件成果知识产权方面实施综合性的保护。例如,在新技术的开发中重视技术秘密的管理,也应重视专利权的取得,而在命名新产品名称时,也应重视商标权的取得,以保护企业的知识产权。企业软保护件成果知识产权的一般途径有:

1. 明确软件知识产权归属

明确知识产权是归企业还是制作、设计、开发人员所有,避免企业内部产生权属纠纷。

2. 及时对软件技术秘密采取保密措施

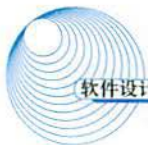
对企业的软件产品或成果中的技术秘密,应当及时采取保密措施,以便把握市场优势。一旦发生企业“技术秘密”被泄露的情况,则便于认定为技术秘密,依法追究泄密行为人的法律责任,保护企业的权益。

3. 依靠专利保护新技术和新产品

专利知识产权是企业重要的经营资源,依靠专利保护企业开发出的新技术或者新产品,使得企业在市场竞争中占据优势来发展自己的产业。我国采用的是先申请原则,如果有相同技术内容的专利申请,只有最先提出专利申请的企业或者个人才能获得专利权。企业的软件技术或者产品构成专利法律要件的,应当尽早办理申请专利权登记事宜,不能因企业自身的延误,造成企业软件成果新颖性的丧失,而失去申请专利的时机。

4. 软件产品进入市场之前的商标权和商业秘密保护

企业的软件产品已经冠以商品专用标识或者服务标识,要尽快完成商标或者服务标识的登



记注册,保护软件产品的商标专用权。

5. 软件产品进入市场之前进行申请软件著作权登记

申请软件著作登记以起到公示的作用,软件著作权登记只要求软件的独创性,并不以软件的技术水平作为著作权是否有效的条件,不能等到软件达到某种技术水平后再进行登记,若其他企业或者个人抢先登记,则不利于企业权益的保护。

12.10.3 建立经济约束机制规范调整各种关系

软件企业需要按照经济合同规范各种经济活动,明确权利与义务的关系。建立企业内部以及企业与外部的各种经济约束机制。从目前存在的比较突出的问题来看,软件企业应建立以下各项合同规范:

1. 劳动关系合同

软件企业与企业职工、外聘人员之间应建立合法的劳动关系,以及应就企业的商业秘密(技术秘密和经营秘密)的保密事宜进行约定,建立劳动利益关系合同以及保守企业商业秘密的协议。一些目前不宜马上实行劳动合同的单位,也要通过建立或者健全本单位的有关规章制度的方式进行过渡,以鼓励企业员工的创造性劳动,明确企业开发过程中产生的软件技术成果归属关系,以预防企业技术人员流动时造成的技术流失和技术泄密等问题。

2. 软件开发合同

软件企业与外单位合作开发、委托外单位开发软件时,应建立软件权利归属关系等事宜的协议,可按照有关规定签订软件开发合同,约定软件开发各方面尚未开发的软件享有的权利与义务的关系,以及软件技术成果开发完成后的权利归属关系和经济利益关系等。如果软件开发方在合作中发现了合同的缺陷,应及早对合同进行补充、完善。

3. 软件许可使用(或者转让)合同

软件企业在经营本企业的软件产品时,应当建立“许可证”(或是转让合同)制度,用软件许可合同(授权书)或者转让合同的方式,来明确规定软件使用权的许可(转让)方式、条件、范围、时间等事宜,避免因合同条款的约定不清楚、不明确而导致当事人之间发生扯皮等不愉快的事情,或者因合同条款无法界定而引发的软件侵权纠纷。

1998-2007 年全国计算机技术与软件专业资格(水平)考试真题及答案汇总

2007 年下半年软考网络工程师试题 Word版
2007 年下半年信息系统项目管理师试题 Word版
2007 年下半年信息系统监理师试题 Word版
2007 年下半年信息处理技术员试题 Word版
2007 年下半年系统分析师试题 Word版
2007 年下半年网络管理员试题 Word版
2007 年下半年数据库系统工程师试题 Word版
2007 年下半年软件设计师试题 Word版
2007 年下半年嵌入式系统设计师试题 Word版
2007 年下半年多媒体应用设计师试题 Word版
2007 年下半年电子商务设计师试题 Word版
2007 年下半年电子商务技术员试题 Word版
2007 年下半年程序员试题 Word版
2007 年上半年系统分析师试题 PDF版
2007 年上半年信息系统监理师试题 PDF版
2007 年上半年信息系统管理工程师试题 PDF版
2007 年上半年信息处理技术员试题 PDF版
2007 年上半年网络管理员试题 PDF版
2007 年上半年数据库系统工程师试题 PDF版
2007 年上半年网络工程师试题 PDF版
2007 年上半年网络工程师试题答案 Doc
2007 年上半年软件设计师试题 PDF版
2007 年上半年软件设计师试题答案 Doc
2007 年上半年软件评测师试题 PDF版
2007 年上半年程序员题 PDF版
2006 年下半年信息系统项目管理师试题 PDF版
2006 年下半年系统分析师试题 PDF版
2006 年下半年信息系统监理师试题 PDF版
2006 年下半年网络工程师试题 PDF版
2006 年下半年网络工程师试题及答案 Word版
2006 年下半年软件设计师试题 PDF版
2006 年下半年软件设计师试卷及答案 Word版
2006 年下半年嵌入式系统设计师试题 PDF版

2006 年下半年电子商务设计师试题 PDF版
2006 年下半年网络管理员试题 PDF版
2006 年下半年程序员试题 PDF版
2006 年上半年程序员试题 Word版
2006 年上半年网络工程师试题(含答案) Word版
2006 年上半年软件设计师试题及答案 Word版
2005 年下半年软件设计师试题及答案 Word版
2005 年下半年网络工程师试题及答案 Word版
2005 年下半年网络管理员试题
2005 年上半年网络工程师试题(含答案) Word版
2005 年上半年软件设计师试题(含答案) Word版
2005 年上半年程序员上午试题(word版)
2005 年上半年网络管理员试题
2005 年上半年网络管理员上午试题(word版)
2005 年上半年网络管理员下午试题(word版)
2005 年上半年网络工程师上午试题(word版)
2005 年上半年网络工程师下午试题(word版)
2005 年上半年软件设计师上午试题(word版)
2005 年上半年软件设计师下午试题(word版)
2004 年下半年网络工程师试题(含答案) Doc
2004 年下半年软件设计师试题(含答案) Word版
2004 年下半年网络工程师上午试题(word版)
2004 年下半年网络工程师下午试题(word版)
2004 年下半年程序员上午试题(word版)
2004 年上半年网络工程师试题(含答案) Word版
2004 年上半年软件设计师试题含答案 Word版
2004 年下半年网络管理员员试题
2004 年上半年软件设计师(高级程序员)上午试题(word版)
2004 年上半年软件设计师(高级程序员)下午试题(word版)
2004 年上半年程序员上午试题(word版)
2004 年上半年程序员下午试题(word版)
2003 年高级程序员试题(含答案) Word版
2003 年度网络设计师试题及答案 Word版
2003 年网络设计师上(下)午试题(word版)
2003 年程序员考试上(下)午试题(word版)

2003 年初级程序员上午试题(word版)
2003 年初级程序员下午试题(word版)
2002 年度网络设计师级试题及答案 Word版
2002 年度高级程序员级试题答案
2002 年网络程序员试题
2002 年系统设计师(高级程序员)上午试题(word版)
2002 年系统设计师(高级程序员)下午试题(word版)
2002 年系统分析员级上(下)午试题(word版)
2002 年网络程序员级上(下)午试题(word版)
2001 年度网络设计师级试题及答案 Word版
2001 年度高级程序员级试题及答案 Word版
2001 年高级程序员级下午试题(word版)
2001 年高级程序员级上午试题(word版)
2001 年网络设计师考试上午试题(word版)
2001 年网络设计师考试上午试题(word版)
2001 年程序员上午试题(word版)
2001 年初级程序员级下午试题(word版)
2001 年度网络程序员级试题
2001 年系统分析员级上午试题(word版)
2000 年高级程序员级试题及答案 Word版
2000 年初级程序员级上午试题(word版)
2000 年初级程序员级下午试题(word版)
2000 年初高程序员级上午试题(word版)
2000 年初高程序员级下午试题(word版)
2000 年程序员上午试题(word版)
1999 年初级程序员上午试题(word版)
1999 年初级程序员级下午试题(word版)
1999 年程序员上午试题(word版)
1999 年程序员下午试题(word版)
1999 年高级程序员上午试题及答案(word版)
1999 年高级程序员下午试题(word版)
1998 年高级程序员级上午试题(word版)
1998 年高级程序员级下午试题(word版)

计算机技术与软件专业资格考试推荐视频教程下载汇总

数据结构视频教程 清华大学严蔚敏主讲 全 48 讲 ASF格式

数据结构C语言版视频教程 全 52 讲

操作系统原理视频教程 浙江大学徐宗元教授主讲(全 32 讲)

数据结构与算法 浙大徐镜春教授主讲(全 40 讲)

计算机网络基础 陆魁军主讲(全 32 讲)

数据库系统概论 浙大张军主讲(全 32 讲)

软件工程基础 浙大陈天洲主讲(全 32 讲)

面向对象程序设计 浙大毛根生主讲(全 30 讲)

程序设计语言视频教程:

孙鑫C++视频教程 rmvb格式 全 20CD

中山大学蔡培兴 C++语言视频教程 全 51 讲

Java视频教程 孙鑫Java无难事(全 12CD)

Java视频教程 即学即会java

中山大学汇编语言视频教程 全 51 讲

汇编语言程序设计视频教程(郝玉洁主讲 全 36 讲)

其他专项视频教程系列:

[程序员考试视频教程] 硬件基础 二进制编码

[程序员考试视频教程] 图的遍历

[程序员考试视频教程] 数据库 关系运算

[程序员考试视频教程] 操作系统 作业调度算法

[网络工程师培训视频教程] 帧中继拥塞控制

[网络工程师培训视频教程] 以太网CSMA/CD

[网络工程师培训视频教程] 奈氏准则和香农公式

[网络工程师培训视频教程] 滑动窗口模型

[网络工程师培训视频教程] TCP数据传输

[信息系统项目管理师视频教程]前言

[信息系统项目管理师视频教程]论文写作 注意事项及例题

[信息系统项目管理师视频教程]项目管理案例分析 例题讲解

[信息系统项目管理师视频教程]项目组合管理 盈亏平衡点

[信息系统项目管理师视频教程]项目质量管理 软件质量特性

[信息系统项目管理师视频教程]项目进度管理 关键路径

[信息系统项目管理师视频教程]项目沟通管理 项目管理信息系统

[信息系统项目管理师视频教程]项目风险分析 决策树分析
[信息系统项目管理师视频教程]项目范围管理 实现范围变更
[信息系统项目管理师视频教程]项目成本管理 挣值法的例子
[信息系统项目管理师视频教程]项目成本管理 影响成本估算的因素
[信息系统项目管理师视频教程]成本效益分析 动态投资回收期
[软件设计师视频教程] 最小生成树
[软件设计师视频教程] 知识产权-著作权法解读
[软件设计师视频教程]数据流图设计(例题讲解)
[软件设计师视频教程] 数据安全-对称加密
[软件设计师视频教程] 面向对象继承访问控制
[软件设计师视频教程] 操作系统习题讲解
[软件设计师视频教程] **Web Service**及应用

.NET 语言(C#/VB)电子资料、开发工具下载汇总

Visual Studio 相关电子资料、软件汇总:

[VS代码辅助工具Visual Assist X 10.4 完美版+特别文件](#)

[CodeSmith 4.1.2 专业版 最新完美版 .NET代码模板生成工具](#)

[Altova MissionKit 2008 for Enterprise Software Architects完美版](#)

[正则表达式辅助生成工具RegexBuddy 3.0.5 破解版](#)

[Pro Visual Studio 2005 Team System](#)

[Microsoft Visual Studio 2005 Unleashed](#)

[Visual Studio Team System Better Software Development for Agile Team](#)

.NET 开发语言电子资料汇总:

[Pro C# 2008 and the .NET 3.5 Platform](#)

[Apress出版 Accelerated C# 2008](#)

[Pro LINQ:Language Integrated Query in C# 2008](#)

[MS Press - Introducing Microsoft LINQ](#)

[LINQ for Visual C# 2005 \(07 年 6 月出版\)](#)

[LINQ for VB 2005 \(07 年 6 月最新PDF文字版\)](#)

[Wrox C# 入门经典](#)

[C# 设计模式](#)

[C# 网络核心编程](#)

[Windows应用高级编程 C#编程篇](#)

[C#高级编程\(第三版\)](#)

[数据结构与算法 C#语言版](#)

[C#字符串和正则表达式参考手册](#)

[O'Reilly 正则表达式参考手册 第二版 2007 年最新出版](#)

[Programming Microsoft Windows with C#](#)

[C# 2005 图解教程](#)

[Visual C# 2005 Express Edition编程初学者指南](#)

[Programming .NET Framework with C#](#)

[C#语言参考](#)

[C#应用程序开发](#)

[Client Side Reporting with Visual Studio in C#](#)

[Wrox Professional VB 2005 with .NET 3.0](#)

Build A Program Now Visual Basic 2005
.NET游戏编程入门经典—VB.NET篇
O'Reilly Visual Basic 2005 Cookbook
.NET Insight for Classic VB Developers
Fast Track Visual Basic.NET
How to Code .NET
ADO.NET全攻略
Apress Professional ADO.NET 2.0
Oreilly .NET and XML
.NET组件编程 (第二版)
Wrox Beginning Visual C++ 2005
Visual C++.NET专业项目
精通.Net核心技术原理与构架
Cross-Platform Web Services Using C# and Java
Advanced C# Programming

.NET、ASP.NET 控件及源码大汇总

[CuteEditor 6.0 在线HTML编辑器的领航者](#)

[ComponentArt.Charting.WebChart.dll](#)

[ComponentArt.Web.UI 2007.2 源代码+实例+DLL](#)

[ComponentArt.WebUI.2007.1 破解DLL](#)

[ComponentArt.WebUI.2007.1 源代码](#)

[ComponentArt.Web.UI.2006.2](#)

[ComponentArt.Web.UI.2006.2 源代码](#)

[ComponentArt.Web.UI.2006.1](#)

[Infragistics NetAdvantage for ASP.NET 2007 Vol 2](#)

[Infragistics NetAdvantage AppStylist 2007 Vol 2](#)

[Infragistics TestAdvantage WinForms 2007 For CLR2](#)

[Infragistics TestAdvantage WinForms 2007 for CLR1.x](#)

[Infragistics NetAdvantage for Windows Forms 2007 Vol 2](#)

[Infragistics NetAdvantage 2007 for WPF](#)

[Infragistics NetAdvantage 2006 Vol2 CLR1.x](#)

[Infragistics NetAdvantage 2006 Vol2 for CLR2](#)

[Infragistics NetAdvantage 2006 Vol2 CLR1.x](#)

[ComponentOne Studio 2007 v1.5 for ASP.NET 2.0](#)

[ComponentOne Studio 2007 v1.5 for ASP.NET 1.x](#)

[ComponentOne Studio 2006 v2 for ASP.NET 2.0](#)

[ComponentOne Studio 2006 v2 for ASP.NET 1.x](#)

[ComponentOne Studio for Mobile Devices 2007 v1.5 CLR1x](#)

[ComponentOne Studio for Mobile Devices 2006 v2 CLR2](#)

[ComponentOne Studio for Mobile Devices 2006 v2 CLR1.x](#)

[ComponentOne Studio 2007 v1.5 for .NET CLR2](#)

[ComponentOne Studio 2007 v1.5 for .NET CLR1.x](#)

[ComponentOne Studio for .NET 2006 v2 CLR2](#)

[ComponentOne Studio for .NET 2006 v2 CLR1.x](#)

[ComponentOne Studio for ActiveX 2007 v1.5](#)

[ComponentOne Studio for ActiveX 2006 v2 CLR2](#)

[ComponentOne Studio for ActiveX 2006 v2 CLR1.x](#)

[Telerik RadWindow for ASP.NET 2.0 v1.8.2.0](#)

[Telerik RadUpload for ASP.NET 2.0 v2.3.2.0](#)

Telerik RadTreeView for ASP.NET 2.0 v6.2.2.0

Telerik RadTabStrip for ASP.NET 2.0 v3.5.2.0

Telerik RadToolBar for ASP.NET 2.0 v1.5.2.0

Telerik RadSplitter for ASP.NET 2.0 v1.2.2.1

Telerik RadSpell for ASP.NET 2.0 v3.1.2.0

Telerik RadRotator for ASP.NET 2.0 v2.6.2.0

Telerik RadPanelbar for ASP.NET 2.0 v4.2.2.0

Telerik RadMenu for ASP.NET 2.0 v4.2.2.0

Telerik RadInput for ASP.NET 2.0 v2.0.2.0

Telerik RadGrid for ASP.NET 2.0 v4.6.2.0

Telerik RadEditor for ASP.NET 2.0 v7.1.2.0

Telerik RadComboBox for ASP.NET 2.0 v2.7.2.0

Telerik RadDock for ASP.NET 2.0 v1.8.2.0

Telerik RadChart for ASP.NET 2.0 v3.2.1.0

Telerik RadCalendar for ASP.NET 2.0 v2.1.2.0

Telerik RadAjax for ASP.NET 2.0 v1.7.2.0

telerik r.a.d.upload

telerik r.a.d.window

telerik r.a.d ToolBar

telerik r.a.d.Chart

telerik r.a.d.combobox

DotNetBar for VS2005 6.8.0.1

DotnetCharting 4.3 破解DLL

DotNET Charting WebForms

dotnetCharting.WinForms

TeeChart for .NET 3.2.2763.26084 完美DLL

TeeChart for .NET 3.2.2699.17379 完美DLL

DevExpress 7.3.4 完美破解DLL

Dxperience 7.3.5 完美破解DLL

DevExpress.LocalizationCHS.Dll

NickLee.Web.UI

SolpartWebControls

AspNetPager 6.0 for ASP.NET 1.x 自定义分页控件

AspNetPager 6.0 for ASP.NET 2.0 自定义分页控件

数据操作类 Socut.Data.dll for .NET 2.0 v3.1

数据操作类 Socut.Data.dll for .NET 1.x v3.1

Developer Express for .NET v7.3.5.0 全套完美无限制版

ASP.NET、Ajax、Silverlight 学习电子资料汇总

Silverlight 电子资料汇总:

O'Reilly Silverlight 1.1 简介

Wrox出版 Silverlight 1.0 (彩页染色代码、全面解析)

Silverlight 1.0 Development with JavaScript

Sams出版 Silverlight 1.0 Unleashed

O'Reilly Essential Silverlight

XAML简明教程 CHM+PDF版

ASP.NET 1.x/2.0/3.5 电子资料汇总:

Pro ASP.NET 3.5 in C# 2008

Beginning ASP.NET 3.5 in VB 2008 从入门到精通

Wrox ASP.NET 2.0 MVP Hacks and Tips

Professional ASP.NET 2.0 Design

ASP.NET 2.0 入门经典

Wrox ASP.NET 2.0 Visual Web Developer 2005 Express Edition Starter

Beginning ASP.NET 2.0 in C# 2005 From Novice to Professional

Wrox Professional ASP.NET 2.0

Wrox Professional ASP.NET 2.0 XML

Wrox Professional ASP.NET 2.0 Security Membership and Role Management

Wrox Beginning ASP.NET 2.0 and Databases

ASP.NET开发人员手册

ASP.NET 2.0 网络编程入门到精通

ASP.NET Web应用程序开发新思维

ASP.NET 2.0 高级应用程序设计专家教程

ASP.NET XML高级编程 C#编程篇

ASP.NET程序开发 C#篇

ASP.NET XML深入编程技术

ASP.NET 2.0 Cookbook

ASP.NET 2.0 Everyday Apps for Dummies

Pro ASP.NET for SQL Server

ASP.NET 从入门到精通

Wrox Beginning ASP.NET 1.1 with Visual C#.NET 2003

ASP.NET 2.0 揭秘

Build Your Own ASP.NET 2.0 Web Site Using C# and VB

开发Microsoft ASP.NET 2.0 网络应用程序

开发ASP.NET 2.0 核心参考

Building Websites with VB.NET and DotNetNuke 4

Wrox出版 Professional DotNetNuke 4.0

Professional DotNetNuke ASP.NET Portals

ASP.NET 视频教程系列汇总:

天轰穿ASP.NET2.0 视频教程(全 106 讲, 共七部分)

VS2005 环境下开发ASP.NET 2.0 Web应用程序视屏教程(swf)

[ASP.NET视频]Data 数据访问与操作

[ASP.NET视频]Masterpages 母版页

[ASP.NET视频]Caching 缓存机制

[ASP.NET视频]Contact页

[ASP.NET视频]ASP.NET详细功能介绍

[ASP.NET视频]Localization 本地化

[ASP.NET视频]Membership and Roles management

[ASP.NET视频]Profiles and Themes

[ASP.NET视频]Tips and Tricks

[ASP.NET视频]Web Parts和Personalization详解

Ajax, ASP.NET Ajax 电子资料汇总:

Ajax基础教程

Ajax宝典

Wrox Beginning Ajax

Ajax in Practice

Ajax模式最佳实践教程

Wrox Professional Rich Internet Applications AJAX and Beyond

O'Reilly Ajax on Java

Practical JavaScript DOM Scripting and Ajax Projects

Creating Web Pages with Asynchronous Javascript and XML

O'Reilly Securing Ajax Applications

Advanced Ajax Architecture and Best Practices

Beginning ASP.NET 2.0 AJAX

Introducing Microsoft ASP.NET AJAX
Wrox Professional ASP.NET 2.0 AJAX
ASP.NET AJAX Programmer's Reference



Java 语言及其相关开发技术电子资料汇总

[Java 编程初步 傻瓜书](#)

[数据结与算法 Java语言版](#)

[JSF JavaServer Faces in Action Manning](#)

[Wrox Professional Java JDK 6 Edition](#)

[Java 2 宝典](#)

[侯捷java编程思想 PDF中文版](#)

[Learning Java \(第三版\)](#)

[Beginning Java Programming for Dummies 第二版](#)

[Java 2 核心编程](#)

[Java How to Program \(第六版\)](#)

[Java All-In-One案头参考傻瓜书 \(第二版\)](#)

[21 天自学 Java 6 \(2007 年 5 月更新出版\) PDF](#)

[Java咖啡馆](#)

[Thinking in Java\(第四版\)](#)

[深入学习JFC SWING - Java基础类组件集](#)

[J2EE全实例教程](#)

[Java信息系统设计与开发实例\(第二版\)](#)

[Java优化编程](#)

[Java信息系统设计与开发实例\(第二版\)](#)

[Tricks of the Java Programming](#)

[Wrox Professional Java Native Interfaces with SWT JFace](#)

[Java Swing 第二版 PDF文字版 O'Reilly出版](#)

[O'Reilly - Java Database Programming with JDBC](#)

[JDBC与Java数据库程序设计](#)

[Learning JQurey \(2007 年 7 月最新出版\)](#)

[J2EE设计开发编程指南](#)

[Java Web Services简明教程](#)

[O'Reilly Java and XML \(第二版PDF\)](#)

[O'Reilly Java and XML \(第三版PDF\)](#)

[Java技术XML高级编程](#)

[Expert One-on-One J2EE Design and Development](#)

[Expert One-on-One J2EE Development without EJB](#)

[JBoss - A Developer's Notebook](#)

The Java Programming Language (第四版)
Spring in Action (第二版)
Professional Java Development with the Spring Framework
Core Java Server Faces 第二版
精通Enterprise JavaBeans
Enterprise JavaBeans EJB 第四版
J2EE应用与BEA WebLogic Server (第二版PDF)
O'Reilly Java Web Services
Ant权威指南
Ajax和Java框架高级编程
Java Web Services简明教程
Cross-Platform Web Services Using C# and Java
O'Reilly Ajax on Java
O'Reilly Java and XSLT
O'Reilly Java and XML Binding
O'Reilly Java and SOAP
Design Patterns Java Companion
J2EE Java黑客大曝光 开发安全的Java应用程序
J2ME API 速查手册
精通J2ME无线编程
J2ME开发大全
Java网页开发的艺术
Java编程高手
Java 5.0 Tiger程序高手秘笈
Java2 网络协议技术内幕(附源码)
Using Enterprise JavaBeans 2
Java技术实用教程
企业级Java安全性(构建安全的J2EE应用)
Java语言集成开发环境Eclipse中文教程
NetBeans IDE 5.5 企业版高级开发教程
Using Enterprise JavaBeans 2
Borland JBuilder Developer's Guide
Eclipse精要与高级开发技术
Java 6 3D游戏开发

视频教程系列:

J2EE开发IDE Eclipse视频教程 全 9CD 完整版

Java视频教程 孙鑫Java无难事 (全 12CD)

J2EE高级开发视频教程第 01 讲

J2EE高级开发视频教程第 02 讲

J2EE高级开发视频教程第 03 讲

J2EE高级开发视频教程第 04 讲

J2EE高级开发视频教程第 05 讲

J2EE高级开发视频教程第 06 讲

J2EE高级开发视频教程第 07 讲

J2EE高级开发视频教程第 08 讲

J2EE高级开发视频教程第 09 讲

J2EE高级开发视频教程第 10 讲

J2EE高级开发视频教程第 11 讲

SQL 语言及各种数据库管理系统电子书汇总

SQL 语言，数据库基础电子资料：

[SQLite权威指南](#)

[SQL 语法大全中文版](#)

[SQL 语言案头完全参考手册](#)

[SQL - A Practical Introduction](#)

[O'Reilly SQL Tuning](#)

[O'Reilly The Art of SQL](#)

[数据库综合资料库](#)

[数据库设计指南](#)

[Wrox Beginning Database Design](#)

[SQL Puzzles and Answers](#)

[SQL Queries for Mere Mortals](#)

[SQL Puzzles and Answers](#)

[Apress出版 The Berkeley DB Book](#)

[数据库系统概论 浙江大学张军教授主讲\(全 32 讲\)](#)

MS SQL Server 电子资料：

[Transact-SQL Cookbook](#)

[SQL Server 2005 宝典](#)

[Microsoft SQL Server 2005 完全参考](#)

[O'Reilly Learning SQL on SQL Server 2005](#)

[Beginning SQL Server 2005 Programming](#)

[Pro SQL Server 2005 High Availability](#)

[Beginning SQL Server 2005 Administration](#)

[SQL Server 2005 Unleashed](#)

[Pro SQL Server 2005](#)

[A Developer's Guide to SQL Server 2005](#)

[Pro T-SQL 2005 Programmer's Guide](#)

[Beginning Transact-SQL with SQL Server 2000 and 2005](#)

[SQL Server 2005 报表服务](#)

[Wrox Professional SQL Server 2005 Programming](#)

[Scaling Out SQL Server 2005 权威指南](#)

[Sql Server 2005 Performance Optimiztion and Tuning Handbood](#)

[Microsoft SQL Server 2005 编程傻瓜书](#)

[Pro SQL Server 2005 Assemblies](#)

[MS SQL Server 2005 Reporting Essentials](#)

[SQL Server 2005 工具箱内幕](#)

[SQL Server 2005 管理员手册](#)

[SQL Server 2005 工具箱内幕](#)

[SQL Server 2005 数据挖掘](#)

[Pro SQL Server 2005 Service Broker](#)

[Pro SQL Server 2005 Replication](#)

[Sql server 2005 的XML最佳实施策略](#)

[Microsoft SQL Server Black Book](#)

[MS SQL Server2000 宝典](#)

[SQL Server 2000 存储过程和XML编程](#)

[SQL Server 2005 高级数据分析视频教程系列](#)

[SQL Server 2005 盛宴系列视频 全 52 讲](#)

MySQL 电子资料:

[SQL for MySQL Developers](#)

[MySQL教程](#)

[MySQL 5 权威指南\(第三版\)](#)

[MySQL培训经典教程](#)

[MySQL Essential Skills](#)

[MySQL Administrators Guide](#)

[MySQL权威指南 中文版+英文版](#)

[MySQL 4.1.0 中文参考手册](#)

[MySQL in a Nutshell](#)

[Export MySQL](#)

[MySQL and PHP from Scratch](#)

其他数据库电子资料:

[Microsoft Access 2007 初学者指南 2007 年 6 月](#)

[Microsoft Access 2007 宝典](#)

[Microsoft Office Access 2007 VBA宝典](#)

[Wrox出版 Expert Access 2007 Programming](#)

[Access 2007 窗体、报表和查询](#)

[Microsoft Access 2007 数据分析](#)

[Oracle Automatic Storage Management](#)

[Pro Oracle Spatial for Oracle Database 11g](#)

[Oracle 9i 数据库管理员指南](#)

[Wrox Professional Oracle 8i Programming](#)

[O'Reilly Oracle Security](#)

[PL/SQL Study Guide](#)

[Sybase实用教程](#)

[PostgreSQL 对象关系数据库开发](#)

[PostgreSQL 必备参考手册](#)

[PostgreSQL 7 数据库开发指南](#)

[PostgreSQL 8 for Windows 2007 年 3 月最新出版](#)

[Crystal Reports 10 完全参考](#)

[Crystal Reports 10 水晶报表 10 傻瓜书](#)

HTML、CSS、JavaScript 等 Web 开发技术电子资料汇总

CSS、HTML、xHTML

CSS权威指南

The CSS Anthology (第二版) CSS设计大师设计思路与实践

HTML & XHTML 权威指南(英文CHM版+中文PDF版)

Building a Web Site 傻瓜书

HTML 4 傻瓜书 第五版

css禅意花园 (高级CSS开发)

CSS与DHTML精髓

CSS网页设计师

CSS Hacks and Filter

CSS Mastery高级Web开发标准

CSS 设计艺术

CSS代码效果对比学习

CSS Web Design傻瓜书

Pro CSS Techniques

CSS HTML高级设计模式 Pro CSS and HTML Design Patterns

Build your Own WebSite - The Right Way Using HTML and CSS

Mastering Integrated HTML and CSS

CSS Web Development从入门到精通

Web Publishing with HTML and CSS in One Hour a Day

Designing with Web Standards 网站重构 英文+中文版

Beginning HTML with CSS and XHTML

XHTML Moving toward XML

Building Smart Web 2.0 Applications

How to Do Everything With HTML

Spatial Data on the Web:Modeling and Management

AdvancED DOM Scripting Dynamic Web Design Techniques

精通XHTML程序设计高级编程

XHTML实例精解

XHTML技术内幕

完美HTML设计 - 使用CSS不用Table (第二版)

250 HTML and Web Design Secrets

JavaScript

JavaScript and Ajax for the Web 第六版

Return on Design

Wrox出版 Silverlight 1.0 (彩页染色代码、全面解析)

Silverlight 1.0 Development with JavaScript

Learn Javascript 高清晰PDF珍藏版

JavaScript 宝典

JavaScript宝典(第六版)

JavaScript实例宝典 PDF文字版

Wrox Beginning JavaScript(第三版)

Professional JavaScript for Web Developers

JavaScript for breakfast

JavaScript in 10 Simple Steps or Less

JavaScript开发人员参考

JavaScript参考大全第二版

JavaScript权威指南(第四版)

JavaScript快速查询手册

Practical JavaScript DOM Scripting and Ajax Projects

JavaScript傻瓜书 第四版

高级Javascript 第二版

Build Your Own Database Driven Website

上百个超酷JS广告代码收集汇总 (第一辑)

上百个超酷JS广告代码收集汇总 (第二辑)

上百个超酷JS广告代码收集汇总 (第三辑)

XML 及其相关技术电子书及视频汇总

[Altova MissionKit 2008 for Enterprise Software Architects完美版](#)

[XML宝典](#)

[Wrox Beginning XML 第四版 2007 出版](#)

[Wrox XML高级编程 2007 版](#)

[Wrox Professional XML Database](#)

[Wrox Professional XML 第二版](#)

[无废话XML](#)

[XAML简明教程 CHM+PDF版](#)

[XML傻瓜书 第四版 PDF文字版](#)

[XML宝典\(第二版\) PDG](#)

[XML编程从入门到精通](#)

[O'Reilly Learning XML](#)

[O'Reilly Learning XML \(第二版\)](#)

[XML终极教程 PDF版](#)

[XML轻松学习手册](#)

[XML Pocket Reference \(第二版\)](#)

[XML编程](#)

[XML in a Nutshell \(第二版\)](#)

[XML 21 天自学教程\(第三版\)](#)

[实战XML\(第二版\)](#)

[轻松搞定XML](#)

[XML实用大全](#)

[XML解决方案开发实务](#)

[XSLT Quickly](#)

[XML参考手册\(第 4 版\)](#)

[XML in Theory and Practice](#)

[Real World Xml Web Services](#)

[Practical Transformation with XSLT and XPath](#)

[Navigating XML With XPath 1.0/2.0 Kick Start](#)

[O'Reilly XSLT Cookbook](#)

[XML之使用SOAP开发Web服务](#)

[XML编程 XQuery专家教程](#)

[XPath,XLink,XPointer,and XML学习手册](#)

O'Reilly XSLT

Beginning XSLT 2.0 从入门到精通

XML学习系列之 XSL-FO权威指南

XML,XSLT,Java and JSP

SVG编程指南

XML Security

Beginning RSS and Atom Programming

O'Reilly XQuery 2007 年最新版

孙鑫xml视频教程 全三讲

PHP 语言、Apache 相关电子书及视频下载汇总

[PHP Designer 2008 专业版+特别文件 完美版](#)

[PHP Designer 2007 专业版+特别文件 完美版](#)

[Practical Web 2.0 Applications with PHP\(Apress 2008 最新版\)](#)

[Practical Apache Struts2 Web 2.0 Projects](#)

[O'Reilly Learning PHP & MySQL 第二版](#)

[PHP 5 傻瓜书](#)

[Wrox Beginning PHP 5](#)

[PHP 5 Advanced](#)

[PHP 5 与MySQL编程初学者指南](#)

[PHP 5 和MySQL 5 从入门到精通 PDF文字版](#)

[PHP 4.1 从入门到精通](#)

[PHP技术内幕](#)

[PHP最新参考手册](#)

[PHP程序设计](#)

[PHP经典 100 例](#)

[Object Oriented PHP Concepts Techniques and Code](#)

[The PHP Anthology 第二版](#)

[PHP API使用完全指南](#)

[Wiley出版 Making Use of PHP](#)

[PHP实例教程](#)

[PHP in Action](#)

[PHP+MySQL网络开发技术](#)

[Dreamweaver CS3 with CSS, Ajax, and PHP](#)

[AJAX and PHP Building Responsive Web Applications](#)

[Beginning Ajax with PHP](#)

[PHP Programming with PEAR](#)

[PHP MySQL and Apache自学教程](#)

[PHP Apache和MySQL网页开发初步](#)

[24 小时学会使用PHP MySQL Apache](#)

[PHP MySQL 网络应用程序开发核心](#)

[Professional LAMP - Linux,Apache,MySQL and PHP 5 Web Development](#)

[Setting Up LAMP - Getting Linux Apache MySQL and PHP Working Together](#)

[Beginning PHP,Apache,MySQL Web Developmnet](#)

PHP Data Objects for MySQL

MySQL and PHP from Scratch

Extending and Embedding PHP

Wiley出版 Secure PHP Development

O'Reilly Building Tag Clouds in Perl and PHP

Wrox Professional Apache Tomcat 5

PHP MySQL编程初学者指南

Beginning PHP and Oracle (PDF文字版)

Advanced PHP for Web Professionals

Apache Server 2.0 实用指南

Apache管理员手册

Apache使用指南与实现原理

PHP专业项目实例开发 中文PDF版

PHP高级开发技术与实例 中文PDF影版

PHP 5 for Flash

O'Reilly - Tomcat权威指南

Wrox Professional Apache Tomcat 6

Pro Jakarta Tomcat 5

Foundations of PEAR - Rapid PHP Development

黑客基地PHP开发中级提高班 PHP视频教程 全 11 讲完整版

一周学会PHP编程 台湾中原大学PHP教程 第一讲

一周学会PHP编程 台湾中原大学PHP教程 第二讲

一周学会PHP编程 台湾中原大学PHP教程 第三讲

一周学会PHP编程 台湾中原大学PHP教程 第四讲

一周学会PHP编程 台湾中原大学PHP教程 第五讲

上百个 **Linux**、**BSD**、**Unix** 学习电子书+视频下载汇总

Linux 学习必备系列之 Linux 各发行版本资料汇总

[Linux宝典 2007 版](#)

[Linux宝典 2005 版](#)

[Beginning Ubuntu Linux](#)

[Ubuntu Linux宝典](#)

[Ubuntu部落](#)

[Ubuntu中文Wiki离线PDF版](#)

[Ubuntu Unleashed](#)

[Ubuntu Linux for Non-Geeks 第一版](#)

[Ubuntu Linux for Non Geeks 第二版](#)

[Moving to Ubuntu Linux](#)

[O'Reilly Ubuntu Hacks](#)

[Debian GNU Linux安装与基本配置](#)

[Debian GNU/Linux宝典](#)

[O'Reilly Learning Debian GNU/Linux](#)

[Redhat Linux 学习指南 第二版](#)

[Learning Red Hat Linux 第三版](#)

[Red Hat Linux 9 魔鬼式培训教程](#)

[Redhat Linux 9 从入门到精通](#)

[Fedora 6 and Red Hat Enterprise Linux宝典](#)

[Red Hat Linux网络管理工具](#)

[Red Hat Fedora Linux宝典](#)

[Red Hat Linux Fedora24 小时自学教程](#)

[Red Hat Linux Fedora for Dummies](#)

[Red Hat Fedora Linux 2 案头完全参考傻瓜书](#)

[红帽企业版Linux国际化支持语配置指南](#)

[红帽企业版 5.0 快速布置指南](#)

[红帽企业Linux安装指南](#)

[RedHat Linux AS 4 平台下Oracle 9](#)

[Red Hat Enterprise Linux 4 傻瓜书](#)

[Fedora 7 Unleashed](#)

[Redhat Fedora core 6 unleashed](#)

[Fedora Core 5 初学者指南](#)

[Redflag HA Cluster 4.1 完全参考](#)

[Redflag Linux Server 4.0 用户手册](#)

[Redflag Data Center 5.0 系统管理](#)

[Redflag Linux Desktop 5 用户手册](#)

[Freebsd简明教程](#)

[FreeBSD 6 Unleased](#)

[FreeBSD使用大全\(第二版\)](#)

[FreeBSD完全手册\(第三版\)](#)

[FreeBSD Handbook PDF 中文版+英文版](#)

[BSD FreeBSD Architecture Handbook](#)

[Absolute BSD - The Ultimate Guide to FreeBSD](#)

[Designing BSD Rootkits - An Introduction to Kernel Hacking](#)

[FreeBSD 6.0 架设管理与应用](#)

[Beginning SUSE Linux 第二版](#)

[SuSe Linux](#)

[SuSe Linux初学者从入门到精通](#)

[SuSe Linux 10 宝典](#)

[SuSe Linux 10 完全参考](#)

[SuSe Linux 10 傻瓜书](#)

[Suse Linux 10 新手指南](#)

[SUSE Linux 企业服务器权威指南](#)

[Suse linux 9.3 用户手册](#)

[Suse linux 9.3 管理员手册](#)

[Knoppix Hacks](#)

Linux 学习必备系列之 基础应用资料汇总

[Linux新手管理指南](#)

[Linux是如何工作的](#)

[Linux简明教程 第四版](#)

[Linux傻瓜书 第六版](#)

[Linux All-In-One Desk Reference for Dummies 2006 版](#)

[Linux All-In-One Desk Reference for Dummies 2005 版](#)

[Learning the Vi Editor \(第六版\)](#)

[GNU Emacs 参考手册](#)

[Linux Cookbook](#)
[Linux for Non-Geeks](#)
[Linux文件查找命令find、xargs详述](#)
[Linux Desktop Hacks](#)
[LINUX 24 学时教程](#)
[Linux案头参考\(第二版\)](#)
[O'Reilly Linux简明教程 第五版](#)
[Linux系统管理白皮书](#)
[Linux系统一本通](#)
[Linux实用培训学习教程](#)
[Linux命令参考大全](#)
[送给初学Linux的穷人Linux系统指令大全](#)
[Linux命令完全参考](#)
[Linux命令字典](#)
[Linux Complete Command Reference](#)
[Linux故障排除宝典](#)
[Linux桌面系统提速法宝](#)
[从Windows转向Linux基础教程](#)
[让Linux像Windows一样方便](#)
[Learning the UNIX Operating System 第四版](#)
[UNIX和Linux权威教程 \(第三版\)](#)
[Unix完全参考\(第二版\)](#)
[Unix傻瓜书](#)
[Unix简明教程 第四版](#)
[Unix教程网络篇](#)
[UNIX for OpenVMS Users 第三版](#)
[Linux网络管理员手册](#)

Linux 学习必备系列之 高级应用资料汇总

[A Beginner's Guide to LVM](#)
[Novell出版 Linux防火墙 第三版 chm格式](#)
[No Starch出版 Linux Firewalls](#)
[2 小时玩转iptables企业版](#)
[Securing Optimizing Linux The Hacks Solution](#)
[SUSE Linux 企业服务器权威指南](#)

[Red Hat Linux网络管理工具](#)

[Linux进程管理教程](#)

[Linux下安装Oracle完全参考](#)

[RedHat Linux AS 4 平台下Oracle 9](#)

[Redflag HA Cluster 4.1 完全参考](#)

[Hack Proofing Linux](#)

[Linux Server Hacks](#)

[Understanding Linux Network Internals](#)

[Linux Power Tools](#)

[Linux On The Mainframe](#)

[Linux Network Servers](#)

[O'Reilly Building Embedded Linux Systems](#)

[Advanced Linux Networking](#)

[Linux Security Cookbook](#)

[保护Linux系统 - Linux安全生存指南](#)

[SELinux NSAs - Open Source Security Enhanced Linux](#)

[Hardening Linux](#)

[Building Secure Servers With Linux](#)

[详细剖析Linux和Unix两系统病毒威胁](#)

[Linux黑客大曝光 - Linux安全机密与解决方案](#)

[使用Ipfilter建立FreeBSD加固防火墙](#)

[Mastering FreeBSD and OpenBSD Security](#)

[User Mode Linux](#)

[Linux Appliance Design](#)

[Linux Device Drivers](#)

[Multitool Linux Practical Uses for Open Source Software](#)

[高性能Linux集群](#)

[Linux网络构架设计与实现](#)

[Building Embedded Linux Systems](#)

[Unix Linux管理自动化](#)

[Linux Power Tools](#)

[Building Applications with the Linux Standard Base](#)

[Hacking Linux Exposed](#)

[Building Secure Servers with Linux](#)

[SSH - Unix Secure Shel tool](#)

[UNIX 系统安全工具](#)

[Unix for Oracle DBAs Pocket Reference](#)

[Unix 网络安全实用教程](#)

[Unix备份与恢复全攻略](#)

[UNIX Power Tools \(第三版\)](#)

[Practical Unix and Internet Security \(第三版\)](#)

Linux 学习必备系列之 Linux 环境编程资料汇总

[Understanding The Linux Kernel 第一版](#)

[Understanding The Linux Kernel 第二版](#)

[Understanding The Linux Kernel 第三版](#)

[Linux内核精要](#)

[Understanding the Linux Kernel - 理解Linux内核](#)

[Linux内核源代码情景分析 中文版 \(上下册\)](#)

[O'Reilly Bash Cookbook](#)

[101 个超酷Shell脚本](#)

[Bash快速参考](#)

[Bash Beginners Guide](#)

[Perl入门及高级编程](#)

[Perl语言编程](#)

[Perl指南](#)

[Perl编程思想](#)

[O'Reilly 精通Perl编程](#)

[O'Reilly Perl and XML](#)

[Perl 5 21 天自学教程](#)

[轻松学习Linux编程](#)

[Linux应用开发基础](#)

[Linux 网络编程](#)

[Python简明教程](#)

[Unix编程艺术 The Art of Unix Programming](#)

[Linux编程白皮书](#)

[A Practical Guide to Linux Commands Editors and Shell Programming](#)

[Linux案头参考\(第二版\)](#)

[Linux与Unix Shell编程指南](#)

[Linux Shell Scripting with Bash](#)

Unix Shell Programming(第三版)
Linux Debugging And Performance Tuning
Linux系统分析与高级编程技术
十分钟Unix自学教程 第二版
Korn Shell:Unix and Linux Programming Manual
Unix shell范例教程 (第四版)
Unix环境高级编程
Unix环境高级编程 第二版
Unix Systems Programming
Linux编程从入门到精通
Linux实例编程
Linux环境编程 GCC完全参考
Linux应用程序开发指南 使用Gtk+ Gnome库
Linux C高级程序员指南
Sams Mono Kick Start - Linux环境的.NET编程

Linux 学习必备系列之 Linux 视频教程系列

Redhat认证 RHCE视频教程 全七部分 高清AVI格式
楚广明 24 小时学通Linux 第一讲 基础安装与GNU历史
楚广明 24 小时学通Linux 第二讲 基本概念与命令
楚广明 24 小时学通Linux 第三讲 系统设置
楚广明 24 小时学通Linux 第四讲
楚广明 24 小时学通Linux 第五讲 DNS服务器
楚广明 24 小时学通Linux 第六讲
楚广明 24 小时学通Linux 第七讲 Apache与Tomcat整合实验
SUSE Linux Enterprise 10 精妙解决方案视频教程
FreeBSD 6.2 服务器架设视频教程 (全五部分)
高效架设Redhat Linux服务器全过程视频教程
楚广明主讲 FreeBSD视频教程 swf版
红旗Linux安装全过程视频教程
debian etch安装全程攻略视频
Arch Linux安装全程攻略视频
Linux基础教程视频(全四讲)
Linux从入门到精通视频教程(swf版) 基础安装配置
Linux从入门到精通视频教程(swf版) 服务器配置

[Linux从入门到精通视频教程\(swf版\) 安全配置](#)
[Linux下访问NTFS分区配置全过程视频教程](#)
[黑客基地Linux视频教程系列之linux简介](#)
[黑客基地Linux视频教程系列之文件系统](#)
[黑客基地Linux视频教程系列之在字符界面下安装](#)
[黑客基地Linux视频教程系列之图形界面安装linux并配置](#)
[黑客基地Linux视频教程系列之体验linux单操作系统的安装](#)
[黑客基地Linux视频教程系列之linux与windows双系统安装](#)
[黑客基地Linux视频教程系列之GRUB的配置方法](#)
[黑客基地Linux视频教程系列之GRUB相关问题解决方法](#)
[黑客基地Linux视频教程系列之桌面简介与网络配置](#)
[黑客基地Linux视频教程系列之VI编辑器使用](#)
[黑客基地Linux视频教程系列之linux中的用户管理](#)
[黑客基地Linux视频教程系列之文件系统常用命令](#)
[黑客基地Linux视频教程系列之网络基础配置](#)
[黑客基地Linux视频教程系列之软件包安装使用](#)
[黑客基地Linux视频教程系列之目录文件的操作命令](#)
[黑客基地Linux视频教程系列之linux下的日志文件](#)
[黑客基地Linux视频教程系列之linux中的进程管理](#)
[黑客基地Linux视频教程系列之linux下压缩文件使用教程](#)
[黑客基地Linux视频教程系列之shell编程简介](#)
[黑客基地Linux视频教程系列之DHCP服务器](#)
[黑客基地Linux视频教程系列之samba服务器\(二讲\)](#)
[黑客基地Linux视频教程系列之DNS服务器设置详解\(三讲\)](#)
[黑客基地Linux视频教程系列之linux下架设apache服务器\(四讲\)](#)
[黑客基地Linux视频教程系列之linux下简单的网络安全](#)
[黑客基地Linux视频教程系列之iptables防火墙简单使用](#)
[黑客基地Linux视频教程系列之linux内核与总结](#)
[边学边用linux视频教程 第 01 讲\(全 24 讲\)](#)
[边学边用linux视频教程 第 02 讲\(全 24 讲\)](#)
[边学边用linux视频教程 第 03 讲\(全 24 讲\)](#)
[边学边用linux视频教程 第 04 讲\(全 24 讲\)](#)
[边学边用linux视频教程 第 05 讲\(全 24 讲\)](#)
[边学边用linux视频教程 第 06 讲\(全 24 讲\)](#)
[边学边用linux视频教程 第 07 讲\(全 24 讲\)](#)

边学边用linux视频教程 第 08 讲(全 24 讲)
边学边用linux视频教程 第 09 讲(全 24 讲)
边学边用linux视频教程 第 10 讲(全 24 讲)
边学边用linux视频教程 第 11 讲(全 24 讲)
边学边用linux视频教程 第 12 讲(全 24 讲)
边学边用linux视频教程 第 13 讲(全 24 讲)
边学边用linux视频教程 第 14 讲(全 24 讲)
边学边用linux视频教程 第 15 讲(全 24 讲)
边学边用linux视频教程 第 16 讲(全 24 讲)
边学边用linux视频教程 第 17 讲(全 24 讲)
边学边用linux视频教程 第 18 讲(全 24 讲)
边学边用linux视频教程 第 19 讲(全 24 讲)
边学边用linux视频教程 第 20 讲(全 24 讲)
边学边用linux视频教程 第 21 讲(全 24 讲)
边学边用linux视频教程 第 22 讲(全 24 讲)
边学边用linux视频教程 第 23 讲(全 24 讲)
边学边用linux视频教程 第 24 讲(全 24 讲)

全国计算机技术与软件专业技术资格（水平）考试指定用书

根据人事部、信息产业部文件，计算机技术与软件专业技术资格（水平）考试纳入全国专业技术人员职业资格证书制度的统一规划。通过考试获得证书的人员，表明其已具备从事相应专业岗位工作的水平和能力，用人单位可根据工作需要从获得证书的人员中择优聘任相应专业技术职务（技术员、助理工程师、工程师、高级工程师）。计算机技术与软件专业实施全国统一考试后，不再进行相应专业技术职务任职资格的评审工作。

本系列推荐书目（清华大学出版社出版）

软件设计师教程（指定教材）

软件设计师备考训练

软件设计师考试辅导

软件设计师考试科目1：计算机与软件工程知识——考点解析及模拟训练

跨越软件设计师必备训练

软件设计师考试全真模拟试题及解析

以及往年试题分析与解答

ISBN 978-7-302-12957-8



9 787302 129578 >

定价：60.00元