普通高等教育“十一五”国家级规划教材
计算机科学与技术系列教材　信息技术方向

计算机系统平台

张　丽　李晓明　编著

清华大学出版社
北　京

内 容 简 介

这是一本从一个新的视角——平台角度来诠释计算机系统的教材。本书从读者熟悉的计算机操作平台入手，逐步深入到计算机硬件、计算机软件、计算机网络、设施和环境等几个层面。主要内容涉及操作系统功能及实现机制、计算机组成、信息表示、计算机网络服务及基础支持机制，以及机房设备与管理等。

本书深入浅出，内容面宽但有层次感，按照教育部《高等学校计算机科学与技术本科专业规范（信息技术方向）》的培养目标和规格要求，围绕规范要求的知识点来组织内容。因此，适合作为计算机专业信息技术方向的本科学生的专业基础课教材，同时也适合作为其他计算机相关专业的选修课教材，本书还可作为希望全面了解计算机系统的其他专业人员自学使用。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目（CIP）数据

计算机系统平台 / 张丽，李晓明编著．—北京：清华大学出版社，2009.7
（计算机科学与技术系列教材　信息技术方向）
ISBN 978-7-302-19722-5

Ⅰ．计…　Ⅱ．①张…②李…　Ⅲ．电子计算机－高等学校－教材　Ⅳ．TP3

中国版本图书馆 CIP 数据核字（2009）第 037198 号

责任编辑：张瑞庆　顾　冰
责任校对：时翠兰
责任印制：

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
社　　总　　机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185×260　　**印　张**：22.5　　**字　数**：543 千字
版　　次：2009 年 7 月第 1 版　　**印　次**：2009 年 7 月第 1 次印刷
印　　数：1～0000
定　　价：.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：　-

计算机科学与技术系列教材 信息技术方向

编 委 会

主 任

陈道蓄

副主任

李晓明 陈 平

委 员

（按姓氏笔画为序）

马殿富 王志坚 王志英 卢先和

张 刚 张彦铎 张瑞庆 杨 波

陈 峻 周立柱 孟祥旭 徐宝文

袁晓洁 高茂庭 董 东 蒋宗礼

PREFACE

序　言

随着高等教育规模的扩大以及信息化在社会经济各个领域的迅速普及，计算机类专业在校学生数量已在理工科各专业中遥遥领先。但是，计算机和信息化行业是一个高度多样化的行业，计算机从业人员从事的工作性质范围甚广。为了使得计算机专业能更好地适应社会发展的需求，从2004年开始，教育部高等学校计算机科学与技术教学指导委员会组织专家对国内计算机专业教育改革进行了深入的研究与探索，提出了以“培养规格分类”为核心思想的专业发展思路，将计算机科学与技术专业分成计算机科学(CS)、软件工程(SE)、计算机工程(CE)和信息技术(IT)四个方向，并且自2008年开始进入试点阶段。

以信息化技术的广泛应用为动力，实现信息化与工业化的融合，这是我们面临的重大战略任务。这一目标的实现依赖于培养出一支新一代劳动大军。除了计算机和网络等硬件、软件的研制开发生产人员外，必须要有更大量的专业人员从事信息化系统的建设并提供信息服务。

信息技术方向作为计算机科学与技术专业中分规格培养的一个方向，其目标就是培养在各类组织机构中承担信息化建设任务的专业人员。对他们的能力、素质与知识结构的要求尽管与计算机科学、软件工程、计算机工程等方向有交叉，但其特点也很清楚。信息技术方向培养能够熟练地应用各种软、硬件系统知识构建优化的信息系统，实施有效技术管理与维护。他们应该更了解各种计算机软、硬件系统的功能和性能，更善于系统的集成与配置，更有能力管理和维护复杂信息系统的运行。在信息技术应用广泛深入拓展的今天，这样的要求已远远超出了传统意义上人们对信息中心等机构技术人员组成和能力的理解。

信息技术在国外也是近年来才发展起来的新方向。其专业建设刚刚开始起步。本系列教材是国内第一套遵照教育部高等学校计算机科学与技术教学指导委员会编制的《高等学校计算机科学与技术专业发展战略研究报告暨专业规范(试行)》(以下简称专业规范)，针对信息技术方向需要组织编写的教材，编委会成员主要是教育部高等学校计算机科学与技术教学指导委员会制定专业规范信息技术方向研究组的核心成员。本系列教材的着重点是信息技术方向特色课程，即与计算机专业其他方向差别明显的课程的教材建设，力图通过这些教材，全面准确地体现专业规范的要求，为当前的试点工作以及今后信息技术方向更好的发展奠定良好的基础。

PREFACE

参与本系列教材编写的作者均为多年从事计算机教育的专家，其中多数人直接参与了计算机专业教育改革研究与专业规范的起草，对于以分规格培养为核心的改革理念有着深刻的理解。

当然，信息技术方向是全新的方向，这套教材的实用性还需要在教学实践中检验。本系列教材编委和作者按照信息技术方向的规范在这一新方向的教材建设方面做了很好的尝试，特别是把重点放在与其他方向不同的地方，为教材的编写提出了很高的要求，也有很大的难度，但对这一新方向的建设具有重要的意义。我希望通过本系列教材的出版，使得有更多的教育界的同仁参与到信息技术方向的建设中，更好地促进计算机教育为国家社会经济发展服务。

2009.6.1

中国科学院院士

教育部高等学校计算机科学与技术教学指导委员会主任

FOREWORD

前　言

“十五”期间，教育部高等学校计算机科学与技术教学指导委员会编制了《高等学校计算机科学与技术专业发展战略研究报告暨专业规范》(试行)，后简称《规范》。“计算机系统平台”是其中信息技术专业方向的建议课程之一。本教材是参照《规范》附录 2.4A 和 2.4B 的内容编写成的教科书。

按照《规范》建议的，“计算机系统平台”这门课的前导课程有“信息技术概论”、“信息技术应用数学入门”和“程序设计与问题求解”，后续可有“计算机网络与互联网”、“Web 系统与技术”、“应用集成原理与工具”和“系统管理与维护”等。开设课程的时间最早可以安排在一年级下学期，比较合适的是二年级上学期，但不要晚于二年级下学期。这是我们编写这本教材时考虑的一个基本定位。

从对信息技术专业方向人才培养的定位出发(见《规范》2.4 节)，我们力求内容的实用性。这种实用性，不仅体现在着重于基本概念及其关系的描述，不限于过于细节的讨论，而且还体现在一种问题引导的陈述方式。在这种方式下，一些关键技术产生所针对实际问题的背景得到了强调，从而有可能促进学生的思考，加深对技术的理解。我们追求内容深度和广度的平衡，以及表述的简洁和严谨。读者也许能体会到，本教材在具有一般可读性的同时，有些内容需要仔细推敲才能真正读懂，尽管没完全懂也不一定影响后续内容的学习。

以“计算机系统平台”为书名的教材，应该说还不多见。可以说这也正是信息技术方向所追求教学内容的一个特色。按照《规范》，信息技术方向的基本目标是围绕社会中各种组织机构(以及个人)的信息化需求，通过对计算技术的选择、应用和集成，创建优化的信息系统并对其运行实行有效的技术维护和管理。我们理解，“系统集成”是上述文字蕴涵的一个要义。所谓“系统集成”，大致上说就是让各种相关技术协同发挥作用以实现某种功能或目标的过程。如何进行有效的系统集成？过去几十年来人们的实践告诉我们，根据普遍的需求构建通用的平台，让满足特殊需求的系统集成在这样的平台上来进行是一条行之有效的技术路线。

那么，到底什么是平台？一般地讲，平台就是满足一类不同事物的共同需求，能够使那些事物方便地在其上形成、存在、运行与展现的基础设施。在生活

FOREWORD

中，火车站的月台就是一个平台，它支持各种火车的停靠、加水加油、旅客的上下车等。在工程中，造船厂的船坞就是一个平台，如果没有它上面的各种基本设施，一条船的建造不是不可能，但会麻烦很多。在计算机中，打开机箱，看到的上面有一个个插槽的主板就是一个平台，借助于它，各种配置的计算机就能够方便地构成。我们常常会听到的软件开发平台也是这个意思，即一整套基础软件工具和环境，可能是很通用的(例如 Java)，也可能是相对比较专用的(或者说面向领域的，例如 Weblogic)，使得开发软件变得容易起来。当谈到软件的时候，常常也区别“软件开发平台”和“软件运行平台”，有些只是开发平台(例如 C 语言编译器)，另一些则只是运行平台(例如操作系统)，还有些则二者兼任(例如整个 Java 体系)。

根据信息技术方向的定位，本教材不讨论平台构建本身，而是从不同的角度或层面看“计算机系统平台”的含义，了解并掌握它们所提供的支撑功能，感受我们能在上面做哪些事情；当然也会扼要地介绍一些典型功能实现的基本原理。为此，全书分成五篇共 9 章进行介绍。

第一篇包含一章，介绍计算机操作运行平台，基本出发点是想回答这样的问题：当我们买来了一台计算机，上面已经装了操作系统，但还没有任何应用程序，此时能做些什么呢？取决于不同类型的用户，能做的事情有所不同。普通用户能做的就是进行各种配置，以及安装所需的应用软件；对于具有程序开发能力的用户，则还可以直接利用操作系统提供的函数编写程序。这一章的目的就是在不涉及实现细节的情况下使读者对操作系统有一个初步但实实在在的概念。

第二篇包含第 2、3 章，介绍计算机的硬件组成和计算机中信息的表示。由于本教材的定位不在于使读者对计算机原理有系统的了解，硬件部分的介绍主要就是结合 PC 的结构，从应用的角度给予介绍。在信息表示方面也类似，主要介绍二进制和典型信息在计算机中的编码，希望学生建立起的概念是：任何形式的信息都可以用 0、1 字符串编码。

第三篇由第 4～6 章构成，集中讲计算机软件平台。第 4 章介绍操作系统功能实现的基本原理。一个安装操作系统的具体例子则放到了附录中。第 5 章介绍计算机系统初始化的过程和 Shell 编程，前者在于加深读者对操作系统运行过程的理解，后者在于对操作系统作为一个平台所提供的功能的理解。第 6 章是对应用程序开发平台的简要介绍。具体的应用程序开发平台有很多，我们只是从“平台”的概念出发，解释它们的共性内容，即它们一般能为程序员提供的方便和设施，包括集成环境，预先实现的应用程序库接口等。

第四篇包含第 7、8 章，讲计算机网络平台。第 7 章是网络基础概念与知识，第 8 章是典型网络应用。同样，我们不追求系统介绍网络实现的原理，也不在于详细介绍各种网络应用的功能，重点在于从使用和维护的角度，介绍所涉及的各种配置及其功能表现的机制和基本原理。

第五篇也只有一章。一方面，我们将视野放开，离开常见的PC类计算机，介绍在企业和机构中常会用到的大型服务器和集群相关的概念，以及路由器和交换机等网络互联设备。另一方面，按照《规范》的要求，信息技术方向的学生希望能在大型计算机设施的运行和维护方面具有竞争力，我们特别对机房设施及其相关要求进行了一定的介绍。

如同《规范》中所论述的，信息技术是当代计算机学科发展的一个重要方向，计算机平台技术是该方向的一个核心内容。然而，如何根据信息技术的人才培养定位，编写出一本这样的教材来，对我们是一个挑战。本教材的构思来源于对《规范》中信息技术方向的理解，基本内容主要来自作者多年讲授计算机相关课程的教学实践，并根据《规范》的要求进行了适应性调整。尽管如此，鉴于作者的学识水平以及对信息技术人才知识结构的理解所限，书中定有不少欠妥甚至谬误之处，谨盼读者不吝指正。同时我们也意识到，将原本在传统计算机专业的几门课的内容抽取一部分成为一门课，对教师也是一个挑战。我们欢迎使用本教材的教师不仅能给教材提意见，而且也能将自己讲授课程的体会和经验与我们交流。

编者

2009年4月

CONTENTS

目 录

第一篇 计算机操作平台 1

第1章 操作系统 3

1.1 终端用户的操作系统平台 4

1.1.1 操作系统界面 4

1.1.2 操作系统平台的基本功能 6

1.1.3 操作系统平台的配置 11

1.2 程序员的操作系统平台 13

1.2.1 程序员的操作系统界面 13

1.2.2 基本的系统调用 14

1.3 操作系统存在的必要性 17

1.3.1 操作系统的来历 17

1.3.2 计算机系统的层次结构 19

1.3.3 屏蔽硬件的利与弊 22

1.4 常见操作系统及分类 23

1.4.1 Windows 操作系统 24

1.4.2 Linux 操作系统 25

1.4.3 操作系统分类 28

复习题 33

讨论 33

实验 33

第二篇 计算机硬件平台 35

第2章 计算机硬件组成 37

2.1 计算机组成结构 39

2.2 总线 44

2.3 存储器 47

2.4 中央处理器 CPU 53

2.5 输入输出系统 59

2.5.1 外围设备 60

2.5.2 驱动程序 78

2.5.3 外设与主机信息传送的控制方式 80

2.6 计算机系统结构 82

CONTENTS

2.6.1 计算机系统结构类型 82
2.6.2 计算机性能的提高 82
复习题 87
练习题 87
讨论 88
实验 88

第3章 信息表示 89
3.1 进位制及其转换 89
3.1.1 进位制 90
3.1.2 数的进制转换 92
3.1.3 二进制数的运算 98
3.2 数字的编码 101
3.2.1 原码 101
3.2.2 补码 102
3.2.3 反码 105
3.2.4 移码 105
3.2.5 小数的表示 106
3.3 字符编码 106
3.3.1 简单字符的编码——ASCII码 107
3.3.2 汉字字符的编码——GB2312 109
3.3.3 字符的统一码——Unicode 110
3.4 机器指令 112
3.5 汇编语言和高级语言 113
复习题 116
练习题 116
讨论 117

第三篇 计算机软件平台 119

第4章 操作系统的内部实现机制 121
4.1 进程的管理 121
4.1.1 进程概念的引入 121
4.1.2 进程的状态 125
4.1.3 进程的调度 128
4.1.4 进程的同步与互斥 130
4.1.5 进程间的通信 132

4.1.6 线程 135
4.2 内存的管理 137
4.2.1 固定分区 138
4.2.2 动态分区 139
4.2.3 页式内存管理 141
4.2.4 段式内存管理 144
4.2.5 虚拟内存管理 145
4.3 信息存储的管理 152
4.3.1 外存管理和文件的物理组织 154
4.3.2 文件操作 156
4.3.3 文件保护 161
4.3.4 虚拟文件系统 164
4.4 外设的管理 167
4.4.1 外设访问接口 168
4.4.2 设备驱动程序 169
4.4.3 设备分配 170
4.4.4 中断技术 171
4.4.5 缓冲技术 173
复习题 176
讨论 177
实验 177
第 5 章 系统初始化及 Shell 编程 178
5.1 计算机系统初始化 178
5.1.1 启动过程要做什么 178
5.1.2 计算机启动过程 181
5.1.3 操作系统的启动过程 183
5.2 CMOS 设置 185
5.3 操作系统安装 190
5.3.1 安装前的准备工作 191
5.3.2 安装过程要点 192
5.4 Shell 编程 195
5.4.1 Shell 的启动 195
5.4.2 Shell 命令 196
5.4.3 编写 Shell 脚本 199
复习题 215
练习题 216

实验 216

第6章 应用软件开发平台 217

6.1 高级程序设计语言 217

6.2 开发工具和开发环境 219

6.2.1 单独的工具 219

6.2.2 集成的开发环境 226

6.2.3 可构建的集成开发环境 232

6.2.4 更广泛的、更易使用的开发平台 242

6.3 开发平台中的可重用代码 242

6.3.1 函数 243

6.3.2 类 243

6.3.3 组件 244

6.3.4 框架 244

复习题 246

讨论 246

实验 246

第四篇 计算机网络平台 247

第7章 网络平台的服务 249

7.1 Web服务 251

7.1.1 万维网概述 251

7.1.2 Web服务器的性能提升 258

7.2 电子邮件服务 261

7.3 FTP服务 266

复习题 268

讨论 268

实验 268

第8章 网络平台基础知识 269

8.1 计算机网络的层次结构 269

8.1.1 分层动机 269

8.1.2 分层与参考模型 270

8.2 将计算机接入网络 273

8.2.1 实现物理连接 273

8.2.2 进行网络信息配置 276

8.3 网络基础支持服务 287

8.3.1 DHCP 服务 287
8.3.2 域名系统 289
8.3.3 网络管理 294
复习题 297
讨论 297
实验 297

第五篇 设施与环境平台 299

第 9 章 计算机系统基础设施 301
9.1 主要设备 301
9.1.1 服务器 301
9.1.2 网络设备 308
9.2 计算机机房 313
9.2.1 机房的位置 314
9.2.2 机房的环境条件 314
9.2.3 机房的电气 318
9.2.4 机房的干扰及防护 320
复习题 324
讨论 325
实验 325

附录 A Fedora 9 安装过程 326
A.1 检测光盘 326
A.2 识别环境 326
A.3 网络配置 327
A.4 时区配置 329
A.5 root 密码设置 329
A.6 磁盘分区 330
A.7 配置引导装载程序 334
A.8 选择要安装的软件 335
A.9 安装完毕 337

参考文献 338

第一篇　计算机操作平台

操作系统为用户提供了一个操控计算机的平台。通过它，普通终端用户和程序员用户都可以方便地控制计算机这个复杂并具有超强能力的设备。操作系统界面也因此成为用户对计算机系统的最直接感知。为了让读者更容易地理解计算机系统，本篇就从用户最熟悉的操作平台开始对计算机系统的介绍。本篇只包含第1章。它首先与读者一起查看计算机操作平台的外观，并体会怎样按照自己的喜好来配置平台。然后将浏览操作平台的基本功能。在此基础上，还将介绍操作系统的发展过程，以便从中总结出操作系统产生的原因及其任务和重要性。最后，介绍众多操作系统中两个常见并各具特色的典型系统Windows和Linux，同时简介操作系统的分类。本篇作为开篇，希望通过与后面各篇内容的结合，力图让读者逐步从表面到内部、从现象到原理了解和理解计算机系统，从而可以更好地操控它，使之效率达到最高。

第1章　操作系统

按下电源开关启动计算机，通常看到的是一幅美丽的背景图案，上面有一些小图标和菜单。单击这些图标或菜单，计算机就会为用户做一些事情，如播放一段音乐或弹出一个可以输入文字的编辑框。当然，在看到美丽的图案之前，一般要经过或长或短的一段等待，并按提示在屏幕上出现的对话框中输入正确的用户名和密码。不过，如果计算机是新买来的，而且没有预装操作系统(这种情况目前已经很少见了)，情况就不同了。那时按下电源按钮，计算机只会在屏幕上出现几个英文字母，然后就不动了。这时，用户会被告知需要安装一个操作系统，如 Windows 或 Linux。在成功安装操作系统之后，就可以看到上述图景，开始使用计算机了。从这里得出一个事实：没有安装操作系统的计算机是没法儿用的。那么操作系统是什么呢？为什么一定要安装操作系统呢？

实际上，不带操作系统的计算机也是非常能干的，它可以做很多事情。但是，向它发号施令却非常复杂。因此，用户必须有一个助手帮忙才能比较方便地控制它。这就好像是用户得到了一个神话世界中非常能干的精灵，但是，必须有懂咒语的人帮忙，才能让精灵为用户做事。操作系统就好像是那个懂咒语的人，而计算机就是精灵。用户通过用鼠标单击菜单，或通过键盘输入命令，让操作系统得知用户的需求。然后，操作系统就会用计算机能懂的方式命令计算机为用户做事。也就是说，实际上干活的还是计算机那一堆硬邦邦的部件。但是，操作系统隔离了操控那些部件的复杂性，将计算机能做的事情以一种比较明了的方式呈现给用户，为用户提供了一个操控计算机的平台。有了它，用户不需要知道计算机内部的各个部件都是干什么的，都在干什么，也不需要知道怎样控制和使用它们。

那些硬邦邦的部件被称为计算机硬件。而操作系统则被称为软件。每个计算机系统都是由“硬件”和“软件”这两大部分组成的。硬件就是那些看得见、摸得着的实实在在的部分，如显示器、主机、硬盘。从设计和生产的角度看，计算机的各个硬件部件还可以不断细分，例如主机包含中央处理器、内存等，而中央处理器又包含寄存器、算术逻辑部件等，到最后就都是原子了。这就好像人的身体可以被层层细分为系统、器官、组织、细胞一样。而计算机软件则好像人的思想，是看不见摸不着的，是计算机的灵魂。软件实际上是人们编制好的、给计算机的命令清单以及计算机完成这些任务必需的一些信息。虽然软件看不到、摸不着，但是它可以通过一些形式表现出来。就好像人的思想可以描述在纸上，用文字表达出来一样。软件在计算机中被记录在由某种存储介质构成的硬件上，如硬盘、U 盘等。操作系统就是软件中的一种，称为系统软件。与之相对应的，还有应用软件，例如 Office、浏览器、媒体播放器等。与人类不同的是，现在的计算机还不能独立思考，不能产生思想。目前它的全部智慧还是来自于人类。也就是说现在所有的软件还都是人类设计的，计算机只是根据这些软件的指示来完成人类交给的任务。计算机还不能自己产生软件，做自己想做的事情。不过，人们正在努力让计算机学会思考。从计算机的发展过程来看，软件在计算机系统中所占的比重越来越大。计算机系统也从具有更多机械特性转变为具有更多智能特性。但是，无论何时计算机硬件始终都是计算机系统的物质基础，是软件运行的前提条件，就像人的身体是

一切智慧产生的物质基础一样。因此，我们将在第二篇中介绍计算机硬件。

这一章我们先来看看计算机系统中最重要的软件，操作系统向我们展现的计算机操作平台。

1.1 终端用户的操作系统平台

世界上有很多的厂家在生产计算机。同时也有很多公司和组织在为这些计算机编制操作系统。因此，就有了很多种不同的操作系统。在这些操作系统中，有些从用户体验到具体实现都非常的不同，如我们常见的 Linux 操作系统和 Windows 操作系统。但也有些看起来非常相似，如 UNIX 操作系统和 Linux 操作系统。不过既然都是操作系统，那么它们就有很多本质上相同的内容。例如，它们都会提供与用户交互的手段，让用户通过它们使用计算机；它们都要管理磁盘，让用户存储信息等等。就好像每个火车站的站台都不同，但是它们都是站台，都是帮助人们上下火车的。因此，它们有些本质的东西是相同的，例如，一定在某个醒目的地方写着这一站的名字、在火车轨道旁边留有人们站立的地方。

下面就来了解操作系统这个操作平台大致是什么样子、在这个平台上人们一般可以做什么事情、操作系统到底给我们帮了什么忙。

不过，在进入正题之前，先要来分析一下计算机系统的用户。实际上计算机的用户大致可以分为两类。一类是普通用户，确切来说应该称为非专业终端用户。他们对计算机知识的了解可能很少，使用计算机或是工作需要，如银行的业务处理，或是为了个人休闲娱乐，如上网聊天、看电影、打游戏等。终端用户一般使用专用的业务软件，或像 QQ、Realplayer、IE 等一类的应用程序。除了终端用户外，操作系统平台还有另外一类服务对象，他们就是编写终端用户所用应用软件的程序员。程序员一般对计算机知识掌握比较多，熟悉一种或多种程序设计语言。显然程序员与终端用户的背景和需求都是不一样的，所以操作系统也就以不同于终端用户的方式与程序员交流。因而程序员看到的操作系统平台与终端用户的也是不一样的。本节先来看看普通用户眼里的操作系统是什么样子的。

1.1.1 操作系统界面

无论什么样的操作系统，既然它的任务是帮助我们使用计算机，那么它首先就必须有一个与我们交互的界面，让我们把需要告诉它。

早期的操作系统与人交互的方式是像图 1-1 所示的黑乎乎的命令行界面。当人们需要计算机做事情时，就用键盘输入相应的命令。例如，在 DOS(早期 PC 上的操作系统)上，如果我们想列出磁盘上的所有文件，就要输入 dir 并回车。当然，不同操作系统能接受的命令是不同的，需要分别学习。现在的操作系统尽管都有了图形界面，但同时仍然都保留着这种键盘命令形式的交互方式。例如，图 1-1 实际上就是现在流行的 Linux 操作系统的命令行界面，而 Windows 操作系统也有一个命令提示符窗口(通过“开始”菜单选择“运行”，输

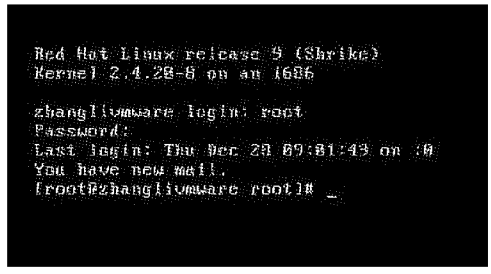


图 1-1 命令行界面

入 cmd 得到),它也是一种命令行界面。

显然键盘命令不够方便。如果不知道操作系统命令,就只能对着黑乎乎屏幕上那几个小小的命令提示符发呆了。同时,键盘命令又很难记。就算记住了,时间长不用,也会想不起来了。本来人们对计算机就有一种敬畏感,觉得它深不可测。键盘命令这种操作方式更是加深了人们的这种感觉。所以,这种交互界面不利于计算机的普及。为此,现在的操作系统都采用了另外一种很友好的交流方式,即图形用户界面,如图 1-2 所示。操作系统的工作界面不再是一个黑乎乎只有几个提示字符、等待用户命令的黑屏幕,而是一幅色彩鲜艳的图画,上有各种图标或者菜单条。人们只要用鼠标单击图标或菜单,操作系统就会安排计算机完成相应的任务。人们不再需要记住命令。常用命令的图标或菜单项都可以直接在屏幕上找到。而且图形界面上通常都有进入帮助的图标或菜单项。用户大多可以通过帮助文档自学使用计算机。

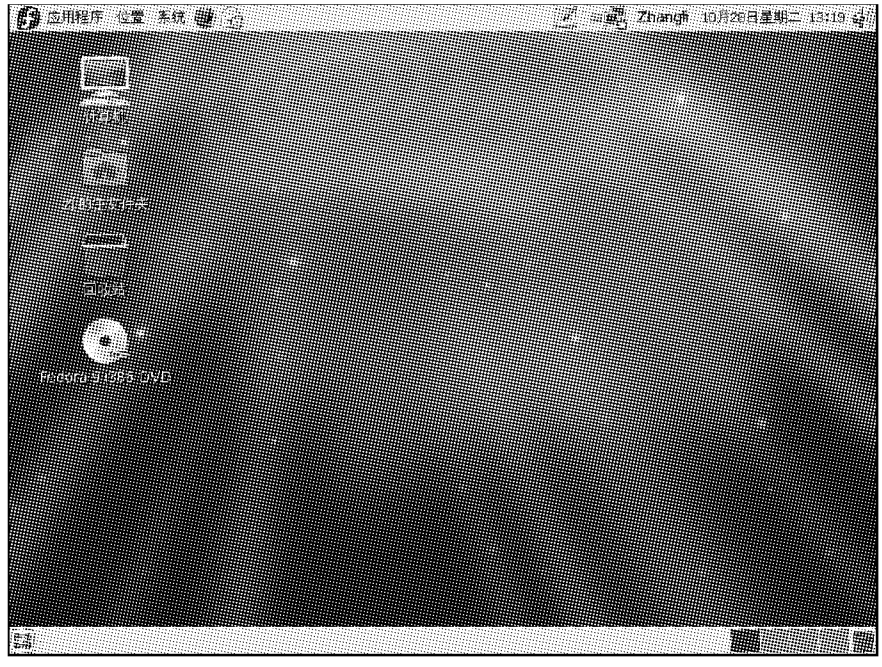


图 1-2　图形用户界面

图形界面使操作计算机变得十分简单,只要点点鼠标,计算机就可以按要求工作了,因此非常令人喜欢。于是越来越多的人开始使用计算机,图形界面的发明为计算机的普及立下了汗马功劳。

通过键盘输入的那些命令的集合与在图形界面上可以实施的操作的集合(例如“拖曳”、“剪切”、“粘贴”等)在操作系统的术语中被称为联机接口,或称为交互式接口。因为用户是在计算机前联机控制的。所谓接口就是人和计算机相互作用的交界面。与联机接口相对应,操作系统还有一种称为脱机控制的用户接口方式。这种交流方式一般巨/大型机的操作系统才会采用。所谓脱机就是用户不在计算机前联机控制,而是把需要操作系统做的事情列个清单一起交给它,然后就让操作系统去做了。这种接口也被称为批处理用户接口。因为它通常用来成批处理任务。后面介绍的 Shell 编程就是脱机控制计算机的一种接口方式。

小提示

可以通过操作系统提供的键盘命令、图形界面或者脱机控制接口告诉操作系统我们的要求，操作系统根据我们的要求控制计算机完成相应的任务。

1.1.2 操作系统平台的基本功能

1. 运行应用程序

有了操作系统这个平台我们能做什么、又不能做什么呢？比如说，想听一段音乐，操作系统能满足我们的要求吗？回答是，它不能直接满足我们的要求。必须有一个能够播放音乐的应用程序，如叫做“豪杰音频解霸”这样的软件。操作系统可以做的事情是让计算机运行这个软件。这样我们就能听到音乐了。

实际上，直接满足用户需求的是计算机上各种各样的应用软件，如文字编辑软件、视频播放软件、财务系统软件等。但是这些软件必须在操作系统这个平台上运行。它们必须通过操作系统来使用计算机的硬件(至于为什么要这样，会在后面的章节中逐步讲解)。因此，操作系统实际上是一个应用软件的运行平台。

在操作系统上运行应用程序非常容易。通常只要输入程序名或者用鼠标单击应用程序名即可。因为这是操作系统最基本的功能。不过，现在多数应用程序在第一次运行之前都需要有一个安装过程。安装过程所做的事情通常是把相关文件复制到特定目录下，并在相关的系统配置文件或者用户配置文件中填写必要的信息。

图 1-3 是 Fedora 9(Linux 操作系统的发行版之一)的(桌面环境的)图形界面(通常称为桌面)。要运行安装好的程序，只要在图 1-3 所示桌面左上角的菜单中用鼠标选定相应的应

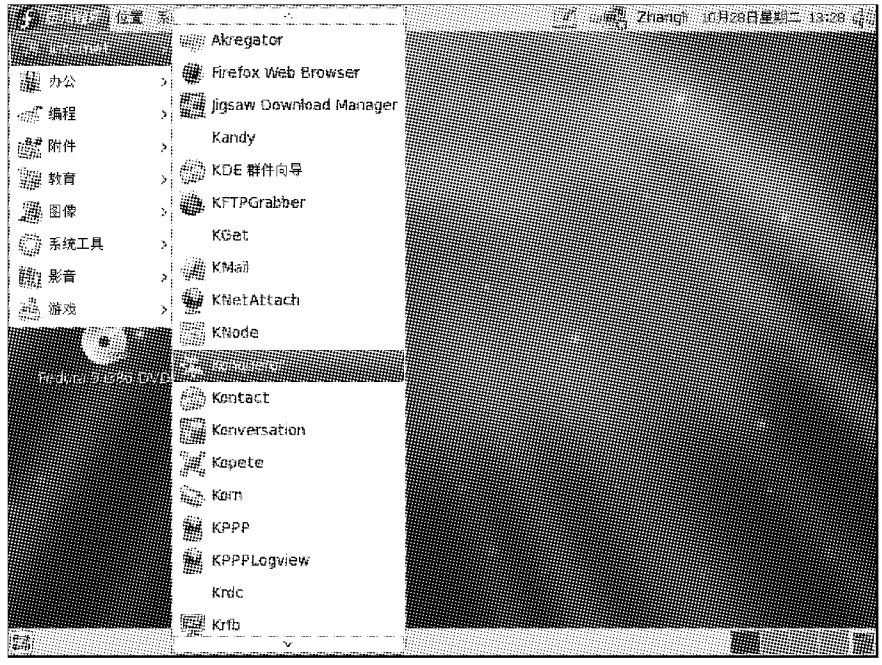


图 1-3 执行应用程序

用程序名，单击鼠标左键即可。例如，要想运行浏览器 Konqueror，只要将鼠标移动到屏幕左上角菜单上的“应用程序”，这时就会出现下拉菜单；再将鼠标移动到下拉菜单的 Internet 项上，屏幕上就会出现下一级菜单；这时再将鼠标移动到其中的 Konqueror 项上，然后单击左键，浏览器 Konqueror 就会运行起来。

除了直接在菜单上用鼠标选定之外，也可以在操作系统的命令行界面用输入程序名的方式运行应用程序。例如，在 Linux 的命令行界面中（确切地说是 Shell），如果要执行浏览器 Konqueror，它的可执行文件存储在/usr/bin 目录下（通常 Linux 下安装的应用程序的可执行文件都存放在/usr/bin 目录下），那么只要在命令提示符后输入如下命令：

```
/usr/bin/konqueror
```

如果当前目录就是/usr/bin，那么只要输入：

```
./konqueror
```

其中，“./”表示当前目录。也就是说在程序名前面要加上完整的目录。如果不想在程序名前加上目录，那么就需要在配置文件中设置路径。具体怎样设，可以去查找相关书籍或者 Linux 的帮助。其他操作系统运行应用程序的方式都与此类似。

2. 管理存储的信息

人们常常会用计算机保存一些信息，如下载一些好听的歌曲保存在硬盘上，或者写一份简历备用，或者把数码相机中的照片复制到计算机硬盘的某个目录下。通常这种事情我们都需要操作系统的帮助。

为了方便地在像光盘、硬盘、U 盘这样的存储器上存取信息，操作系统为用户提供了创建、删除文件，以及更改文件名、移动文件位置等手段。这样我们就不需要了解访问这些存储设备的细节了。操作系统要求我们把信息以文件的形式存储。相关的信息存储在一个文件中，用一个文件名标识，可以通过文件名向操作系统说明要访问的信息。在 Fedora 9 的图形界面上，用鼠标双击桌面左上角的“计算机”图标，然后再在出现的界面上单击“文件系统”，就会出现如图 1-4 所示的界面。

图 1-4 显示的是文件系统根目录下的所有内容，包括其下保存的目录和文件。操作系统用户无须知道文件在磁盘上的具体存放位置，只知道文件的名字和逻辑位置即可。这里的逻辑位置是指文件在目录树中的位置。在多数操作系统中，文件并不是简单地罗列在系统中，而是被组织到若干目录中。目录可以再被组织到上一级目录中。这样就形成了层次组织结构——目录树。每个目录都有名字。目录也被称为文件夹。这样组织的好处，我们将在操作系统的内部实现机制章节中说明。

小提示

操作系统负责管理外存上长期存储的信息。它为用户提供创建、复制、删除文件的功能。
通常系统中的文件都以目录树的形式组织起来。

用鼠标双击图 1-4 中的某一个目录，就会出现新的窗口显示目录中的内容。用鼠标左键单击选中某个文件，然后单击鼠标右键，屏幕上就会出现如图 1-5 所示的下拉菜单（要在

图 1-4　文件系统根目录界面

“查看”菜单的子菜单中选择“显示隐藏文件”项才能看到图 1-5 中这些以“.”开头的配置文件和文件夹）。

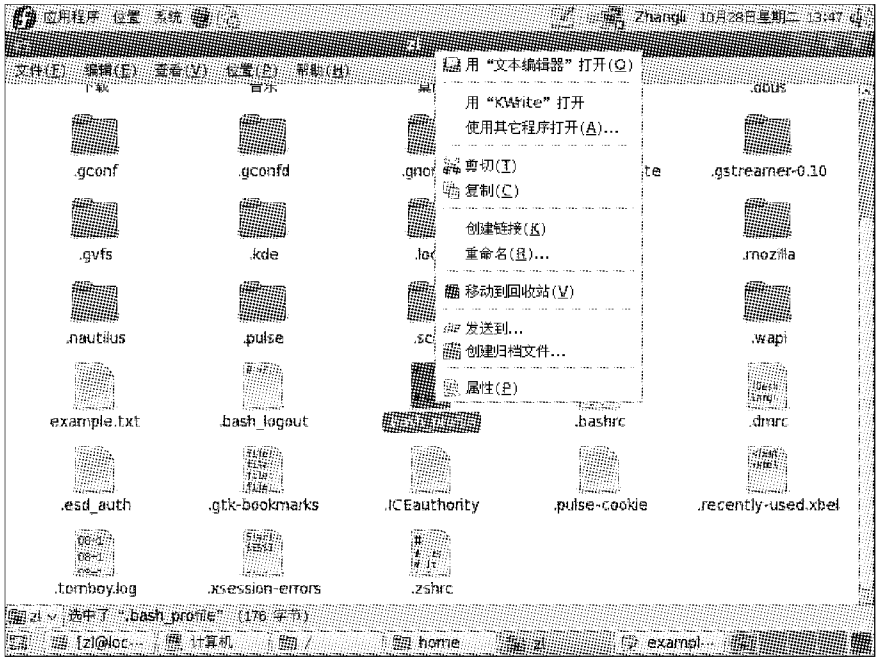


图 1-5　文件及目录操作下拉菜单

通过该下拉菜单中的“剪切”、“复制”选项可以复制和剪切文件。当用鼠标右键单击某

个目录窗口中的空白处时，出现的菜单中会有“粘贴到文件夹”字样。这个选项和“剪切”或“复制”选项结合就可以把文件移动或复制到其他目录中。菜单中的“重命名”可以用来更改文件或目录的名字。

通常文件名分为两部分，由句点“.”分开，如 kunming.jpg。句点前面的部分通常由用户指定。而句点后面的部分则由处理文件的应用程序自动生成，被称为文件名的后缀。例如，用 Word 软件编写的文档后缀通常是 doc，而数码相机中存储的照片文件后缀名通常是 jpg。不同的后缀代表不同的文件类型，实际上意味着信息内容存储格式的不同。所以，不同类型的文件要由不同的应用软件来解释。一般一个应用软件只能解释相近的几种文件类型。只有选择合适的软件才能查看到正确的文件内容。例如，用 Word 软件可以打开 doc 文档，但是不能打开后缀为 ppt 的文档。TXT 类型的文件只能由文本编辑器一类软件解释，而 JPEG 文件则由图片编辑或者查看软件才能打开。

操作系统的文件管理只为终端用户提供了文件的创建和移动等操作。如果想查看或更改文件的内容，那么就要借助于应用软件。对于文件内容的增、删、改任务，操作系统所提供帮助是启动处理该文件的应用程序。当我们用鼠标双击图 1-5 中的某个文件时，操作系统就会启动能够处理该类型文件的应用程序。例如，双击图 1-5 中的 example.txt 文件时，系统就会启动文本编辑软件，如 gedit，来显示文件的内容。因为 txt 后缀表示文件是一个以文本形式保存的文件，因而要用文本编辑类软件打开。通常操作系统会有个表格记录某种后缀的文件需要用什么应用程序来打开。因而，文件名的后缀不能随意更改。如果文件的类型在表格中没有记录，当用户双击要打开的文件时，操作系统会显示出系统中可以处理文件的所有应用程序，让用户选择。用户选择后，操作系统会记住这种类型文件曾经用该应用程序打开过。下次用户再要打开该类型的文件时，操作系统就直接启动它记录的应用程序。

除了使用 Fedora 9 桌面上的“计算机”图标来管理文件之外，用户还可以通过菜单上的“应用程序”→“系统工具”→“文件浏览器”选项来查看和管理文件。

前面这些图形界面下的操作，在 Linux 的命令行界面下都有相对应的命令。例如，命令 cp 可以把一个文件复制另外一个地方，命令 mv 可以移动文件的位置（其实，前面提到的这些对于文件的操作，严格地来说，并不是 Linux 操作系统直接提供的，而是由应用程序借助于操作系统的接口函数提供的。人们看到的 Linux 图形界面，实际上叫做 Linux 桌面环境，它本身就是由一组应用程序构成的。Linux 操作系统本身也提供对文件的这些操作，不过不以图形界面的方式提供。管理文件的这些应用程序是 Linux 操作系统最基本文件操作的外部封装，通常都作为基本套件随 Linux 操作系统发布。对于 Windows 这样的操作系统，文件管理的功能则是由操作系统本身的图形界面直接支持的。从这可以看出操作系统之间确实有很大不同，但是它们的功能和目标确实是一致的）。

3. 显示系统状态

运行应用程序、管理存储的信息，有了这两项功能，普通用户的需求基本上也就满足了。不过，有时候用户也需要对系统目前的状态有个了解和掌握。因此，操作系统也都提供了这部分功能。例如，在 Linux 命令行界面下输入 top 命令，就可以看到系统中正在运行的进程的情况。在图形界面的菜单上选择“系统监视器”，如图 1-6 所示，也可以达到同样的目的（从图上可以看到 Fedora 9 提供了两个系统监视器，它们有不同的显示形式和内容）。进程是指程序的一次执行过程（第 4 章会详细说明这个概念）。在命令行界面输入可执行程序

名，或者图形窗口中双击可执行程序名，程序开始执行，也就产生了一个进程。Top 命令会显示出系统中进程使用 CPU 的时间、占用的内存量、进程的状态，以及进程是由哪个用户启动的等信息。

另外，每个操作系统都有日志，里面记录着系统中发生的重要事情。一旦系统运行有什么不正常的地方，用户就可以通过命令来查看日志的内容，推断到底哪里出了问题。在图 1-6 所示界面中，选择“系统工具”→“系统日志”命令就可以查看系统的日志。

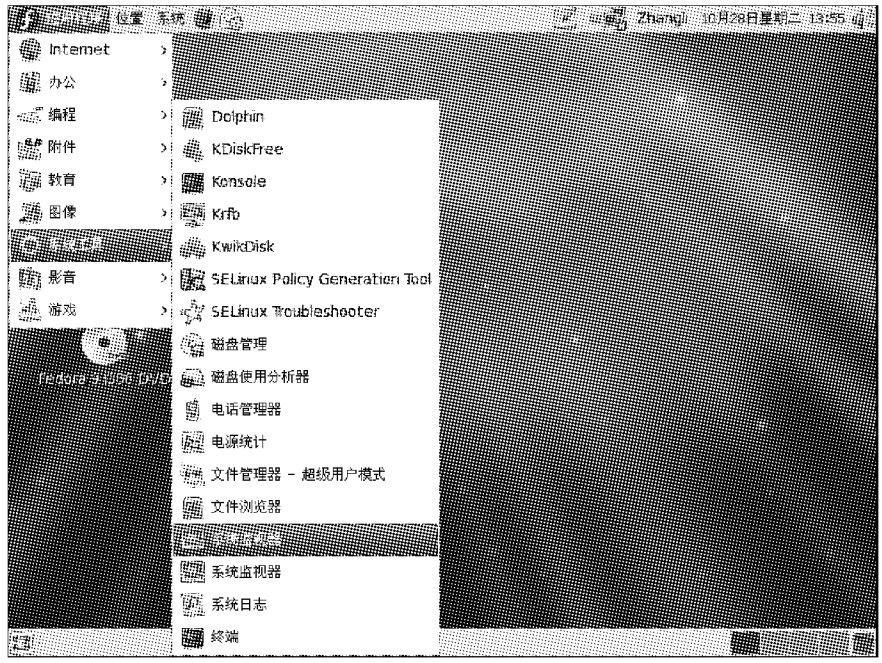


图 1-6　系统监视器

其实，在操作系统这个平台上，可以做很多事情。这里介绍的只是操作系统最最基本的几项功能。这就好像站台的最基本功能是让旅客安全地上下火车，除此之外，它还有好多设施，如显示时间的表、提供开水的饮水机等，提供必需的或者额外的功能来方便旅客。操作系统也一样，它还有很多其他基本的和额外的功能。不同的操作系统在其他功能上会有不同的表现，就好像不同站台的辅助功能也或多或少地不同一样。可以查阅操作系统的使用手册来了解和学习更多的关于操作系统功能的内容。后面我们会介绍操作系统针对普通用户之外的程序员用户的基本功能。对于普通用户来讲，操作系统的目标就是通过尽可能友好的界面让用户可以方便地使用计算机。

小提示

运行应用程序、管理存储的信息、显示系统的运行状态是操作系统平台提供给终端用户最基本的三项功能。

1.1.3　操作系统平台的配置

每一个操作系统版本发布之后，它的功能就是确定的了。就像站台修好了，它能做的事也就固定了一样。但是，每个操作系统都给用户提供了一定选择，让用户来配置自己的操作平台。这就好像是工作台虽然搭建完毕，但是，要调整工具的摆放位置以适应自己的工作方式；也可以把工作流程表贴在工作台边上，这样工作时查看起来会方便一些；或者还会对工作台进行一些美化工作。对操作系统平台进行配置就是做类似的工作。

现在的操作系统都是支持多用户的系统。因此，操作系统平台上可以进行的配置一般也分为两大类别：一是某个用户自己的环境的设置，另外一类是针对操作系统平台的整体环境，对系统的所有用户都起作用。

每个用户可以对自己的个人操作环境进行设置。例如，在图形界面下，用户可以选择自己喜欢的桌面背景、设置屏幕保护出现的时间以及使用的屏幕保护程序等。在支持多个命令行界面(Shell)的 Linux 操作系统中，用户可以配置自己登录后默认使用的是哪一个命令行界面。

除了定制自己的操作平台外观外，用户也可以对运行环境进行配置。运行环境通常由一些称为环境变量的信息来描述。例如，操作系统都有一个名为 PATH 的环境变量，它定义了用户执行的应用程序或命令在系统中的查找顺序。第 1.1.2 节的“运行应用程序”中讲过在命令行界面运行应用程序时，必须给出程序所在的完整目录名，如/usr/bin/konqueror，因为只有给出完整的路径名，操作系统才能在磁盘上找到这个应用程序(为什么要这样，在操作系统实现机制的文件系统中有详细的解释)。但是如果我们把/usr/bin/加入到 PATH 环境变量中，那么就可以只输入程序或命令名，即 konqueror，就可以运行这个程序了。这是因为操作系统如果在当前工作目录下找不到输入的程序或命令名，那么它会查看 PATH 环境变量，在里面给出的路径中逐个查找是否有这个程序或命令，当它查找到/usr/bin/目录时就会找到 konqueror，因此就能运行了。

在 Linux 的命令行界面(bash Shell)中，可以在命令提示符后输入如下命令来设置环境变量：

```
export PATH=$PATH:/home/mypath
```

这条命令就把/home/mypath 目录添加到 PATH 环境变量中，$PATH 表示取原来 PATH 环境变量的值，冒号是多个路径之间的分隔符。所以这条命令就是在原来 PATH 变量设置的路径中再增加目录/home/mypath。设置完之后，可以用命令

```
echo $PATH
```

显示环境变量 PATH 的值。你会发现在后面确实多了一个目录/home/mypath。这样我们再运行/home/mypath 中的程序时，就不用输入目录名了。也可以把 export 那条设置路径的命令写入到自己主目录下的一个名为. bash_profile 的文件中。这样每次命令行界面(bash Shell)启动时就会将环境变量的值设置好。

在 Windows 操作系统中设置环境变量就更容易了。例如，在 Windows XP 中，右击“我的电脑”→“属性”→“高级”→“环境变量”命令。就会看到如图 1-7 所示的配置环境变量的窗口。从图上可以看出环境变量被分为两部分，即用户自己的环境变量和系统整体的环境

变量。单击下面的“新建”、“编辑”和“删除”按钮就可以对环境变量进行增、改、删操作。从这个图中可以看到环境变量不止 PATH 一个。系统或者其他应用程序在运行中会使用这些环境变量中的一个或者几个。

图 1-7　Windows XP 设置环境变量

系统管理员可以对操作系统平台的整体环境进行配置。通常他可以做的事情包括：增、删、改可进入系统的用户账号；设置随系统启动的服务；安装和删除系统中可用的应用程序；增、删系统支持的硬件设备；设置主机的名字；设置主机所用的网络地址；设置系统日期与时间；设置文件和目录的权限；设置系统的默认登录界面是命令行方式还是图形窗口等。

当然，系统管理员还可以设置系统的整体运行环境，即设置系统的环境变量。设置方式同用户自己设置的类似。例如，在 Linux 操作系统下，管理员同样可以用命令设置环境变量。不过，一般管理员都用配置文件来完成。管理员要把环境变量设置命令添加到/etc/profile 文件中，而不是用户主目录下的.bash_profile。在 Linux 操作系统中，系统的配置文件都在/etc 目录下。当用户登录，系统启动命令行界面(Shell)时，会先读取/etc/profile 文件执行其中的命令，然后再读取用户主目录下的./bash_profile 文件。从图 1-7 中已经看到了在 Windows 下设置系统环境变量的情形。

系统管理员的配置工作就好像火车站台建设完毕之后，站台管理员对站台进行的进一步配置工作一样。他可能需要调整站台上时钟的时间，他还可能规定什么样的人什么时间可以进入站台，如只有持有即将出发列车车票的旅客可以进入站台，或者所有人都可以进入。系统管理员可以和必须做的配置和管理工作是如此地多，以至于可以有一门叫做“系统管理与配置”课程来专门讲这些工作。

1.2　程序员的操作系统平台

1.2.1　程序员的操作系统界面

程序员与终端用户不同。程序员懂得计算机的专业知识，但这并不是他们最重要的区别，关键是程序员的需求与终端用户不同。他们不像终端用户只要能运行程序就行了，他们需要调动和使用系统的软硬件资源来实现应用软件的功能。例如，使用显示器显示结果，或者从键盘读取用户的输入，或者把信息存放在硬盘中。而且对这些资源的调配和使用的具体细节必须是以程序设计语言的形式体现在应用软件的代码中。按照目前计算机系统的设计，操作系统是计算机硬件和应用程序之间的桥梁（为什么这样设计，在第 1.2.2 节说明）。要使用任何硬件资源都必须经过操作系统。所以，应用程序要想使用资源，必须给操作系统下命令。这些命令要由程序员放在应用程序的代码中，因而这些命令必须以程序语句的形式出现。因此，操作系统和程序员之间的交流不能采用终端用户那样通过用户输入命令或者单击鼠标来实现。操作系统必须提供程序语言格式的命令，让程序员能够把它们写在程序中。为此，操作系统提供了第二类接口，即程序员级接口，用来和程序员（实际上是他们写的应用程序）交流。

在有些操作系统中程序员级接口在形式上和一般机器指令相似，因此被称为广义命令。在有些操作系统中，程序员级接口的形式与函数形式是一样的。因此，在这些系统中，程序员级接口被称为系统调用。系统调用和广义指令都可以被放在程序代码中。这样当程序执行到这些广义指令或系统调用时，操作系统就会按照要求替应用程序完成相应的任务，如向屏幕上写文字。

程序员级接口提供的功能与操作系统的功能是相对应的。操作系统能做什么，当然就会提供给程序员相应的控制指令了。操作系统的功能非常强大，因而程序员可以使用的系统调用也是多之又多。不过不同操作系统提供的程序员级接口的功能、名称和数量并不相同。所以，程序员为一个操作系统写的应用程序通常不可以在另一个操作系统上不加修改地运行。

系统调用可以看作是一组特殊的函数。在程序员看来，调用系统调用和调用普通函数没有区别。但实际上它们是有本质区别的。为什么这样说呢？因为操作系统有一个目标，就是要保证计算机系统的安全，不能让恶意用户或粗心程序员的程序把系统搞乱或崩溃。为了实现这个目标，计算机的指令按照重要性被分成若干级别。目前较多的做法是分为两级，即核心级指令和用户级指令。那些比较重要的、会对系统产生致命影响的指令，如设置一些特殊寄存器（什么是寄存器，在第 2 章有介绍）的值、访问存储器特殊空间以及控制外设等，都属于核心级指令。这些重要的核心级指令只能由操作系统来执行。操作系统在某种程度上就好像是计算机的操作员，替用户操作计算机。为了保证系统的安全，关键的操作一定要由操作员亲自执行。而不太重要的指令，则归为用户级。这些指令可以由用户程序直接向计算机发布。也就是影响不大的操作可以由用户自己来做。相对于此，处理机的工作状态也被分为两种：核心态和用户态。只有处于核心态时，处理机才能执行核心级指令。处理机在执行操作系统代码时，就处于核心态。而处于用户态的 CPU 只能执行用户态指令。

处理机的工作状态通过设置硬件标记来改变的。同样这个硬件标志只能由操作系统来控制。当用户程序调用系统调用时，操作系统会使 CPU 的状态转为核心态。这样就可以执行核心指令了。系统调用中就可以向 CPU 发布核心级指令了。而调用用户自己写的函数时，操作系统会让 CPU 处于用户态。因而用户的函数中只能使用用户级的指令。所以说，系统调用和普通函数是有本质区别的。

用户程序如果想使用核心指令必须通过操作系统，即通过系统调用或者广义指令。为什么这样就安全了呢？这就好像是复印机的操作员怕你因为不会用而弄坏了复印机，所以如果你要复印，就必须向他提出要求，由他来操作一样。这样只要他的操作是正确的，就不会出问题。因此复印机就安全了，不用担心哪一天来了个鲁莽的用户一通乱按而被弄坏了。不过，不同的是，在这种情况下，复印机不存在状态转换，永远属于核心态，只能由操作员来使用。而 CPU 的工作状态则可以在核心态和用户态之间转换。系统调用的代码是由操作系统提供的。在用户程序执行到系统调用时，处理机的状态就变成核心态，系统调用中的代码就可以对特殊资源进行操纵了。操作系统的相关代码执行完毕后，系统调用结束，程序控制转回到用户程序，处理机也转换为用户态，应用程序也就不能够再对敏感资源进行访问了。通过这种方式，操作系统可以保护系统中的敏感资源，保证核心级指令不会被乱用。

下面通过 Linux 系统的系统调用来了解程序员级操作系统接口的形式和功能。

小提示

操作系统与程序员的接口称为广义指令或者系统调用。
计算机指令通常被分为核心级和用户级两个级别。
系统调用可以使用核心级指令。
普通用户程序只能使用用户级指令。

1.2.2 基本的系统调用

1. 执行应用程序

程序员在程序中可以使用系统调用执行另外一个应用程序。例如，下面这段 C 语言代码可以让当前的程序执行另外一个应用程序 Konqueror。

```
#include<unistd.h>
main()
{
    execl("/usr/bin/Konqueror","Konqueror",(char*)0);
}
```

这段代码中其实只有一条语句，就是系统调用 execl。它的作用就是执行一个应用程序。应用程序的名字、位置以及参数在系统调用 execl 后面括号中给出。这里，要执行的程序是浏览器 Konqueror。在 execl 后面括号中的第一个参数/usr/bin/Konqueror，表示要执行的程序的位置。第二个参数是应用程序的名字，这里为 Konqueror。如果应用程序执行时需要参数，那么可以在程序名，即 Konqueror 和(char *)0 之间添加以字符串形式表示的参数。程序员可以把这条代码放在程序中，只要把各个参数调整成合适的内容，即把 Konqueror 换成其要执行应用程序的名字、填写正确应用程序的位置以及所需参数。这样，当他的应用

程序执行到这条语句时，就会转去执行设定的应用程序。

不过，程序员在程序中直接使用“执行应用程序”接口的时候比较少，因为它们的代码本身就是一个应用程序，这种情况就是在应用程序中再去执行其他的应用程序。通常程序员会在程序中使用一个名为 fork 的系统调用创建另外一个进程，让它来执行另外的应用程序，或者其他代码。这样就可以实现任务的并发处理。关于进程和并发处理在操作系统的内部实现机制中会详细说明。例如，在需要并发处理的程序中，多会出现类似下面的代码段：

```
⋮
int pid;
pid=fork();        /* create a new process */
if (pid !=0) { /* original process */
    /*原进程的任务处理代码段*/
}else { /* newly created process */
    /*新进程的任务处理代码段*/
}
⋮
```

这里系统调用 fork 会创建出一个新的进程，它执行 else 语句后面的代码段，这段代码可以是处理一个任务的语句段，或者是使用系统调用 execl 执行另外一个应用程序。而原来的进程则执行 if 后面的代码段。这样，一个进程就变成了两个，它们就可以并发执行任务了。

关于程序执行的系统调用还有很多，这里就不一一介绍了。

2. 文件访问

管理存储的信息目前来说就是管理和访问文件。Linux 下与文件相关的常用系统调用有 create、open、close、read、write、link、mkdir、chdir 等。它们分别被用来创建、打开、关闭、读、写、链接文件，以及创建目录，改变当前工作目录。例如，如果程序员在程序中添加了如下语句：

```
fd=open ("hello.c",O_RDONLY);
```

程序就会通过操作系统以只读方式打开 hello. c 文件。而如果在其后加上语句

```
nbytes=read (fd,buf,10);
```

则表示要从刚才打开的 hello. c 文件中读出 10 个字节的内容放到变量 buf 中。而程序中如果出现语句

```
i=mkdir("/home/zl",0644);
```

则会在/home 目录下创建一个新目录 zl，后面的 0644 表示目录的权限，具体含义就不做介绍了。当然这条语句能否让应用程序成功创建该目录还要取决于执行这个程序的用户是否有在/home 目录下创建新目录的权限。

其他文件相关的系统调用的具体功能和用法，就不一一做介绍了。这里只是让大家知道程序员可以通过系统调用来操纵存储的文件。

3. 系统状态显示

终端用户可以通过系统提供的命令和工具查看系统的运行状态。而对于程序员来说，

想让程序在运行中了解系统的状况也不是个难事儿。系统日志以文件形式存储在磁盘特定的目录下。例如，Linux 中保存在/var/log 目录下的文件 messages 就是一个日志文件。日志文件通常不只有一个。例如，Linux 的/var/log/wtmp 也是个日志文件，里面记录着每个用户登录、注销以及系统的启动、停机等事件。而前面提到的文件 messages 则包含有与硬件和服务有关的信息，如某个服务启动成功或者失败。如果查看 Linux 的/var/log 目录，你还会发现很多日志文件。这些日志都以明文的形式存储。因此，程序员可以在程序中使用访问文件的系统调用直接读取这些日志的内容。而系统调用 syslog 则可以让程序向日志文件中填写新的日志内容。

在 Linux 中有一个特殊的文件夹/proc。这个目录下的所有文件都是特殊文件。这些文件的内容囊括了关于系统执行的各个方面的信息，如系统的 CPU 使用情况、每个进程执行的命令行、已经打开的文件号以及进程状态等。每个进程的信息以各种特殊文件的形式存在/proc 下以进程号为名称的子目录下。/proc 目录下还有系统中各种资源的管理信息、各种设备的信息、文件系统的信息等。例如，在命令提示符后面输入命令：

```
cat /proc/cpuinfo
```

(cat 命令的作用是显示其后文件名所表示文件的内容)，就会看到如图 1-8 所示的关于 CPU 的信息，这就是/proc/cpuinfo 文件的内容。因此，如果程序员想让应用程序在执行过程中了解系统进程的运行状况，通过访问文件的系统调用读取/proc 下的文件即可。实际上，/proc 下的这些文件并没有存放在磁盘上。文件中所有的内容都是在用户读/写这些文件时，系统根据内存中的管理信息动态生成的。因而，它们可以实时地反映系统的运行状况。此外，系统调用 sysinfo 可以让程序获得诸如系统已用内存大小、可用内存大小、系统当前进程数，以及系统启动至今的运行时间等相关的系统信息。

```
processor  :0
vendor_id : GenuineIntel
cpu family      : 6
model           : 13
model name      : Intel(R) Pentium(R) M processor 1.60GHz
stepping        : 8
cpu MHz         : 797.973
cache size      : 0 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug : no
fpu             : yes
fpu_exception : yes
cpuid level     : 2
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat clflush dts acpi mmx fxsr sse sse2 ss
bogomips : 1599.07
```

图 1-8　/proc/cpuinfo 文件的内容

每个操作系统都提供了各种让应用程序了解系统运行状态的系统调用/命令。这些系统调用的数量和形式可能会不同。但是，无论怎样操作系统最终都是通过它们满足程序员

的需求。

小提示

操作系统让应用程序使用的功能都通过系统调用提供给程序员。
程序员需要了解怎样正确使用系统调用。

操作系统通过系统调用提供给程序员的功能要比终端用户界面提供的多很多。操作系统所能做的事情都会通过程序员级接口反映出来。也就是说，系统调用的功能是和操作系统的功能相对应的。道理其实很明白，操作系统可以做的事情，一定会提供给用户一个接口，让用户可以指挥它来做这些事。如果不提供接口，那些功能用户就没法使用了。不过，终端用户主要是通过应用程序来使用计算机，因此，操作系统的很多功能他们都不需要直接使用。例如，操作系统会管理内存，因而它提供了系统调用 brk 让应用程序来申请内存，还提供了系统调用 mmap 可以让应用程序把一个打开的文件内容映射到自己的用户空间。但是，普通用户对内存是不感兴趣的，对内存的使用完全由应用程序根据自己的需要来掌控。所以，终端用户界面就不会提供类似使用内存的功能。终端用户通过程序员用系统调用编写的应用程序来间接使用这些功能。

图 1-9 总结了操作系统这个用户计算机操作平台与用户及计算机硬件的关系。操作系统工作在计算机硬件之上。终端用户通过操作系统的文本界面或者图形界面操纵计算机。应用程序运行在操作系统平台之上。应用程序的设计和开发者使用操作系统的程序员级接口实现应用程序对计算机硬件的控制。

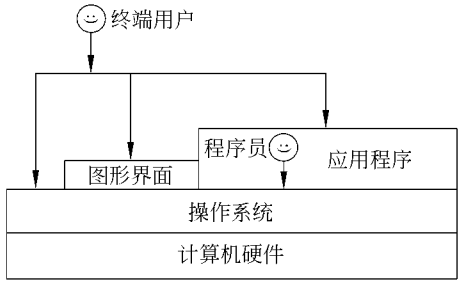


图 1-9　计算机操作平台示意图

1.3　操作系统存在的必要性

前面对操作系统有了一个表面上的接触，也明白了一个事实，即对计算机的操控都是通过操作系统这个平台来完成的。现在，读者可能会问：为什么要通过操作系统这个平台来操控计算机？这样做有必要吗？直接面对计算机的硬件不行吗？这一节就来回答这些问题。

1.3.1　操作系统的来历

要讲操作系统平台的必要性，就要从它的来历说起。不过这可有点儿话长，要从计算机出现的时候说起。

计算机刚出现时，它还不能读程序。程序是由程序员兼操作员用接插板或开关板解释给计算机的。也就是人们只能通过扳动计算机庞大面板上无数的开关来向计算机输入信息，弄不好可能还需要重新搭电路。那时计算机也没有显示设备，只能由信号灯来显示运算的结果。人们写程序也只能使用二进制的机器指令，连汇编语言都没有。

后来人们发明了用带孔卡片表示程序、用卡片机读取的输入方式。这时计算机做事的

流程是：操作员把代表程序的带孔卡片放在卡片机上，控制计算机读入并执行；执行完毕，操作员取下打印机的结果，换上另外一个程序的卡片，继续控制计算机执行。慢慢地人们发现一些问题：当操作员换卡片时，计算机就没事做了；而且操作员必须始终在计算机前监视着，时刻准备程序执行完毕，换下一批卡片。操作员换卡片的时间通常还要包括从称为输入室的地方取来卡片、从打印机上撕下计算结果放到输出室去。

为了减少计算机的等待时间，提高这个昂贵设备的利用率，计算机的设计者就写了一个专门的程序来控制需要执行的程序。这个程序被称为监控程序。操作员首先让计算机把监控程序读入内存并执行。然后，操作员把要执行的程序成批交给计算机。监控程序控制计算机读入一个用户程序，之后监控程序让出处理器，让处理器执行新装入的这个程序。不过监控程序并没有结束退出，因为接下来它还要管事呢。等装入的程序执行完，计算机的控制权自动转交回监控程序。它继续控制计算机读取下一个程序并执行，如此往复。这时人们也开始用汇编语言和高级语言写程序了(第 2 章中会说明什么是汇编语言、高级语言以及它们的关系)。汇编语言源程序需要用汇编程序汇编成二进制机器指令才能被计算机执行。高级语言源程序需要用编译程序编译成汇编语言，再汇编成二进制机器指令才能被计算机执行。如果程序是用汇编语言或者高级语言写的，那么程序员需要写一个说明书，里面用监控程序能够读懂的命令告诉他需要用什么汇编/编译程序来汇编/编译这些源程序。这样监控程序就会命令计算机读入指定的汇编/编译程序并执行。汇编/编译结束后，监控程序再控制计算机执行汇编/编译后的程序。

在监控程序的参与下，程序的切换不再需要人的介入。这样切换时耽误的时间就大大减少了。尽管执行监控程序也需要时间，但是比起人来，它的动作还是要快多了。计算机的效率因此大大提高了。因为在整个系统运行期间监控程序都要控制计算机，因此它始终被放在内存中，从来不会被清出内存。为了避免被其他程序破坏，监控程序被放在内存中特定的地方，那段内存空间不允许其他程序使用。

可以看出，监控程序与在其控制之下执行的其他程序的作用和地位是不一样的。从这时开始，计算机中执行的程序分化成两种类型。一种是用户请求执行的程序，我们把它称为应用程序。另外一种就是像监控程序这样的，并不是用户请求执行的，而是为了提高系统效率，或者为运行用户程序做准备工作而运行的程序。这类程序被称为系统程序。

尽管在监控程序的控制下，计算机的效率有了很大提高。但是计算机的 CPU 还是没有被完全利用起来。CPU 不用再等待慢吞吞的人，但它还是有很大一部分时间是在等待。例如，当程序需要打印机输出结果时，CPU 必须等待慢悠悠的打印机把结果打印完毕，才能继续执行下一条指令。为此，人们又对系统的执行过程进行了改进。在打印机输出结果时，CPU 不再等待打印结束，而是把控制权转回给监控程序。监控程序再读取另外一个程序到内存中执行。打印机完成工作后，它向 CPU 发一个信号，CPU 再转回到这个程序做后面的工作。这样 CPU 的利用率就又被提高了。在这种办法下，计算机上同时存在了两个未执行完的用户程序，一个在用打印机，另一个在 CPU 上执行。显然，这样的程序可以不只有两个，因为可以有很多种外设。这时，内存中也不再是只有监控程序和一个应用程序，而是可能同时存在多个程序。有的程序正在 CPU 上执行，有的程序正在使用打印机，有的程序则正在等待键盘输入。这些程序可能分别执行到其代码的某一个地方。这种提高效率的方式使监控程序的工作异常复杂起来。它需要知道现在有多少程序已经开始执行，都存放在

内存中的什么地方，分别都执行到了什么地方，它们之间是否有关联，都打开了哪些文件，谁在使用什么外设等。监控程序还要决定 CPU 现在应该执行哪个程序：是继续执行当前这个程序，还是执行刚刚完成输出操作的程序，还是重新装入一个程序来执行。除此之外，因为内存中同时放了几个未执行完的程序，所以监控程序必须要保证它们之间不能互相干扰，一个程序不能改变另外一个程序在内存中的代码或数据。这些都要求计算机的设计者仔细设计监控程序，确保不能出现差错，因为监控程序决定了整个计算机系统能否工作的问题。

与此同时，计算机硬件在不断发展，CPU 变得越来越快，外设的种类和数量也越来越多。监控程序必须充分利用这些硬件资源，保证系统的性能。因而它要做的工作也越来越多，整个计算机系统也越来越离不开它。这时的监控程序已经不再是一个简单的程序，而是由一组复杂程序构成的一个软件系统，这就是我们常说的操作系统。

从名字上可以看出操作系统的主要任务是帮助人们操作计算机。有了操作系统的帮助，尽管计算机已经变得很复杂，人们还是可以很方便地使用它。而对于计算机系统来讲，操作系统的另一个任务就是提高整个系统的效率。硬件再好，管理不行，利用不充分，效率还是上不去。

到底什么是操作系统？确切地说，现在还没有一个严格的定义。从前面的介绍中，可以得出操作系统是一个系统软件。它负责管理和控制计算机系统中的各种资源，协调计算机系统中的各个组成部分的关系，使系统能够高效运转，并为用户提供操作计算机的方便手段。从上面这个定义可以看出三点内容。首先，操作系统也是计算机程序，只不过它的作用和地位比较特殊。第二，它的任务是为用户使用计算机提供基础支持。所有计算机用户都是通过操作系统的帮助和计算机打交道的。操作系统让用户能够更方便地使用计算机。除此之外，操作系统还负责提高整个计算机系统的效率。它通过调度计算机系统的资源，把它们合理地分配给需要的用户，来达到充分利用系统中的资源、保证系统的效率，并尽量提高用户满意程度的目的。总结起来，操作系统的任务就是方便用户使用计算机和提高系统效率。

小提示

操作系统是作用和地位比较特殊的计算机程序。
方便用户使用计算机和提高系统效率是操作系统的两大任务。

1.3.2　计算机系统的层次结构

本节换一个角度来讲解操作系统的作用。

从操作系统来历的介绍中，以及日常接触的计算机世界，可以发现计算机系统的功能越来越强，但是，系统本身也变得越来越复杂。任何一个系统，当它变得复杂起来的时候，必须有一个有效的组织和管理结构，它才可能继续发展壮大下去。人类社会就是一个非常庞大、复杂的系统。人类社会发展到今天，其中贡献最大的一件事情就是社会分工。在社会分工下，尽管我们每天都需要食物，但是多数人并不需要自己去种庄稼，粉碎稻谷，甚至不用亲手把面粉制成馒头。同样，拥有汽车的人也不用自己去炼钢、设计汽车、生产零件、组装。社会

分工使每个人各司其职，专注去做自己擅长的事情。这样每个人的工作能力和水平都得以加强，整个社会的效率也因此大大提高。人类社会因而得以持续不断地向前发展。很难想象，如果每个人都需要亲自驯养家禽，种植粮食棉花、织布，人类社会能够发展到现在的样子。计算机系统同人类社会一样经历了从简单到复杂的过程，只不过计算机的历程要简短多了。在这个短暂而又迅猛的发展过程中，计算机系统中也像人类社会一样逐步出现分工，并一直继续到目前的状态。

在计算机的发展过程中，它的功能在不断地增强。很多以前不能完成的事情慢慢都可以做到了，如三维动画模拟、播放电影等。但是，计算机也因此变得越来越复杂，绝大多数这些复杂的任务都不再是一个程序员单枪匹马就能驾驭计算机完成的了，因为很难有人能够精通计算机的全部知识。即使有这样的人，包揽全部任务从头开始把每一步工作都写成代码几乎也是不可能的事情了。例如，要写一个能够显示照片的软件。那么，程序中需要有从硬盘上读取信息的代码。这要求程序员了解硬盘上信息的存储格式，以及向硬盘驱动器发布命令的方式。不同硬盘的控制方式不一样，所以程序员需要了解所使用硬盘的细节知识。为了把照片显示出来，程序员还要知道怎样和显卡打交道。不同显卡的控制方式也不一样。所以程序员同样需要了解所使用显卡的细节知识。为了让这个程序能够在另外一台计算机上运行。程序员需要了解新机器的硬盘和显卡，然后更改代码，重新编译程序。如果程序员用高级语言写程序，还要自己写个编译程序……这确实不是一个人可以完成的事情。

除了一个人了解整个系统的知识难度很大之外，人们还发现其实有时一个程序员完成全部程序也是完全没有必要的，因为很多不同用途的程序中有些代码的功能是相同的。比如，尽管文字编辑软件功能与照片显示软件不同，但是它也需要从硬盘上读取信息，并通过显卡显示文字。不过，每个程序员对这些相同功能的实现却可能不同。精通硬盘知识的人所写的访问硬盘信息的代码可能效率就比较高，缺陷也比较少，但是他写的显示代码可能就很差。而另外一个程序员写的程序则可能正好相反。

看到这些现象，人们开始分工。计算机的任务被分解成若干小任务，由不同的程序来完成。一些程序专门为另外一些程序提供服务。这些程序互相配合，共同完成一个大任务。每个程序员专注于编写自己精通的程序。例如，熟悉某种显卡硬件的程序员就专心写这个显卡的控制代码，而编写照片显示软件和文字编辑软件的程序员都直接在程序中引用前面那个人编写的显卡控制代码。结果是程序写得又快又好。

相对于人类社会的分工，计算机系统的分工看起来要简单得多。系统各部分之间的关系可以抽象成一种层次关系，如图 1-10 所示。在这个层次结构中，处于下层的部分为上一层提供服务，而上层部分则依靠下层的工作，加上自己的努力来完成某个任务。

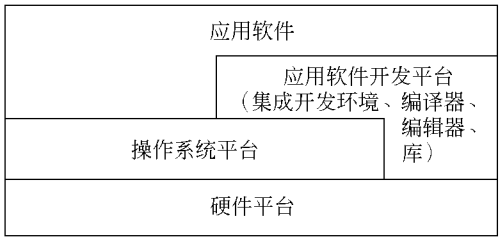


图 1-10　计算机系统层次结构图

在计算机系统层次结构中，位于最底层的是实实在在的硬件。它提供实际的计算功能。这一层也被称为裸机。在计算机硬件之上就是操作系统，它负责管理计算机的软硬件资源，比如分配 CPU、内存，控制外设等。在操作系统之上是一些系统提供的工具软件，比

如编译程序、编辑程序、集成开发环境，以及各种函数库、类库等。它们构成了应用软件开发平台。最上层是我们最常接触的应用程序。从图上可以看出，操作系统是各种应用程序和硬件之间的桥梁。它把各种硬件的复杂特性给遮盖起来。应用程序访问硬件时，只要向操作系统指明要做的动作并提供必要的数据即可。而其他与此有关的细节则不用应用程序操心，都由操作系统代理了。就像顾客想吃西红柿炒鸡蛋，他只要把菜名告诉厨师，就可以等着吃了。至于怎样做，就完全不用他操心了，厨师都一手包办了。

可以看出，计算机系统的这种分工之所以叫做层次，是因为上一层的工作是在下一层提供的服务的基础上完成的。正是这种层次结构，使计算机系统能够完成越来越复杂的工作。

小提示

计算机系统中软件和硬件被组织成一种层次结构，处于下层的软/硬件为上一层提供服务，上层的软件需要依靠下层的服务来完成任务。层次结构使计算机系统更容易设计、实现和管理，推动了计算机系统的发展。

不同的计算机用户接触到的层次，能够看到和理解的计算机系统是不一样的。对于终端用户，即非专业用户，如银行柜台后面坐着的业务员，他们通常只看到最上层的应用程序，如财务系统软件、聊天软件、电子邮件客户端等。他们也会接触到操作系统提供的一些简单计算机管理操作，如复制文件、创建一个文件夹等。这些用户只需要掌握最简单的操作系统命令和操作。一般他们都通过操作系统的图形界面使用操作系统。通常那些讲授 Windows 操作的培训班针对的就是这一类用户。

终端用户使用的应用软件是由一批特定的程序员来开发的。有很大一批程序员专注于开发这些终端用户使用的应用软件。这些应用软件是软件中数量比重最大的一部分。它们也是计算机中直接创造价值、完成任务的软件，是人们购买计算机的驱动力。这些程序员需要利用编辑软件来编写源程序，并通过编译程序编译他们写的源程序代码。现在更多的程序员使用集成了编辑器和编译器的开发环境来编写应用软件。这些应用程序中可能会使用到具体的硬件，如向磁盘写一些数据、要喇叭发出声音等。不过，程序员并不需要了解这些硬件的具体使用办法。操作系统向应用程序提供访问这些硬件的接口。设计和编制应用软件的程序员所要做的事情就是了解操作系统提供的这些接口的功能、用法以及各种开发工具软件的用法。

而应用软件开发平台的设计和开发人员，他们需要了解操作系统的程序员级接口，通过它来使用系统的资源。有些开发工具，如编译器，有时需要绕开操作系统直接控制硬件，因而编译器的开发者有时也会看到裸机，需要了解硬件的细节。

操作系统帮助终端用户和程序员来控制计算机硬件，因而操作系统的设计和开发者看到的是实实在在的硬件，他们需要了解各种硬件的特性，想办法把这些硬件充分利用起来。操作系统的设计者除了保证上层的用户能够方便地使用计算机外，还要努力使整个系统性能达到最高。因此，操作系统设计和开发者要做的工作非常多，他们的工作可以用艰苦卓绝来形容。

小提示

计算机系统主要被划分为四层，从下到上依次是计算机硬件平台、操作系统平台、应用软件开发平台、应用软件。

不同的计算机用户接触不同层次的计算机系统。终端用户使用应用软件和操作系统终端用户接口。

应用程序员接触应用软件开发平台以及操作系统平台。

应用软件开发平台设计和开发者接触操作系统以及部分硬件。

操作系统的设计和开发者需要了解计算机硬件的全部细节。

1.3.3 屏蔽硬件的利与弊

从前面的介绍，可以知道操作系统位于计算机应用软件和计算机硬件之间，它就像一个操作员完全代劳用户使用硬件。这样用户可以不用了解硬件的细节，只要向操作系统发布命令即可使用各种硬件了。这被称为硬件屏蔽。这样做到底是好是坏，可以通过生活中一个类似的事例来分析。

人们经常会需要复印资料。当然，用户可以自己用复印机复印。不过，这需要用户对复印机的操作略知一二。如果新换了一个没用过的品牌或型号的复印机，那么用户可能还要对着说明书仔细熟悉一会儿。通常，用户并不这样做，而是到复印室去解决这个问题。复印室里有很多复印机，不过我们并不能接触到它们。在复印机和用户之间通常隔着一个柜台。柜台上粘着一张价格表。上面写着提供的复印服务种类以及相应价格，如 A4 纸每页 0.08 元。我们要做的事情就是把要复印的资料交给柜台里面的操作员，并且告诉他印多大的幅面以及印数。然后，操作员就会按照要求去复印。他要做的工作可能包括：打开复印机的电源开关(可能还要先接上电源线)、等待打印机出现准备就绪的指示标志、打开复印机上面的盖子、从资料中取出一页放在上面、盖好盖子、把空白纸张放到进纸盒中、设置复印的张数、按复印按钮。然后，复印机的输出口会出来一页印好的纸。如果复印的效果不好，操作员会按控制面板上的一些按钮来调整设置，如墨色的深浅等，直到印出满意的副本。在这之后他打开盖板，取出已经被复印过的一页，放入要复印的下一页，如此往复直到全部印完。最后他把复印好的资料交给我们，我们则按照规定付钱给他。

计算机的操作系统就好比柜台里面为我们复印的操作员，他必须熟悉复印机，即硬件的具体用法。而操作系统之上的应用软件开发平台和应用软件就好像是要复印的我们。柜台上那张说明服务项目和价格的纸就像操作系统所提供的系统调用的接口说明。我们向操作员要求具体服务的过程就是进行系统调用。价目表上每个服务说明，如复印 A4 纸，实际对应于一个特定的功能，也对应于操作员的一整套动作。而交给操作员的原始材料和取走的复印件，就是系统调用的输入和输出参数。

有了复印室提供的复印服务，我们不再需要了解复印机的操作过程，只要知道复印室这套服务的使用过程，如找到复印室、向操作员说明自己的要求、付费等。这样我们需要了解的信息就少了。复印室可能会换一批新的复印机。这些复印机的操作与以前的可能不同。例如，新复印机不再需要操作员一页页地放资料，也不用打开机盖，只要把资料一起放在一个输入槽中，复印机就可以自动一页一页复印。这需要操作员重新熟悉新机器的操作流程。但是，这不影响价目表上的服务项目。作为用户的我们不会感觉到任何变化。同样，在计算

机系统中，硬件不但复杂而且还会不断升级变化，如果应用软件要直接操控这些硬件，那么编写应用软件的程序员就需要不断地掌握这些变化资源的控制细节。而对程序员来说，这显然是个比较头疼的问题。有了操作系统的介入，当具体硬件设备有变化时，操作系统访问硬件的操作会发生变化，即操作系统的代码实现需要改变。但是，只要硬件功能不变，操作系统提供给应用软件的接口就不变，因而应用软件也不需要改变，程序员也不用学习新知识。这就是操作系统屏蔽硬件带来的好处之一，方便用户。

此外，复印室之所以不让客户接触复印机的原因之一是怕机器损坏，耽搁了复印室的工作。比如，如果把带有钉书钉的资料放入复印机的输入槽，就会使复印机出现故障。通常，操作系统不让用户软件直接接触硬件也是这个原因。万一哪个程序员乱写了一通程序，把外设搞乱了，会大大影响系统的工作进程。这是操作系统屏蔽硬件带来的好处之二，保护硬件。

再有，复印室的操作员有权决定复印的顺序。当然，通常他要相对公平才行，一般先来客户的活儿肯定要先做。但是，在一些特殊时候，他也可以做一些改变。例如，一个先来的客户要复印一本书，在他后面又来了一个人要复印 10 页纸。这时操作员就可以先复印后者的 10 页，然后再复印前者的书。如果是用户亲自操作复印机，那么后来的人要等多久，就可能要取决于先来人的性格和心情了。同样道理，现在的系统都支持多道程序。在这种多个程序同时运行的情况下，就会出现竞争资源的情况。而操作系统这个操作员，可以综合多方面的因素，如公平性以及系统效率等，来分配资源。这是屏蔽硬件的好处之三。

不过，操作系统通过提供接口屏蔽硬件细节也会带来一个小麻烦，那就是灵活性差了。还是拿复印室这个例子来说。假定复印室只提供单色复印服务，那我们就不可能要求印彩色。而实际上复印机具有彩色复印功能，只是因为操作员不会用，或者他懒得去做，或者觉得没有人会需要这种服务。同样道理，有些硬件可能有一些用户需要的特殊功能。但是，操作系统并没有提供使用这些功能的接口，这样就不能使用这些功能。

小提示

操作系统屏蔽硬件可以方便用户使用硬件，还可以起到保护硬件的作用，同时还可以有效分配硬件资源，但是，降低了设备使用的灵活性。

1.4　常见操作系统及分类

从计算机产生至今，世界上出现了各种各样，不下百余个不同的操作系统。这些操作系统从本质上来说功能基本相同，即前面提到的两大功能：为用户提供操作计算机的接口以及管理计算机资源以提高系统的效率。但是它们在实现细节以及具体表现上却有很大的区别。这主要是因为各个操作系统的目标重点、用户群，以及主要所针对硬件的不同造成的。这些操作系统各有长处，也有着各自的不足。它们也因此在不同的应用领域中占领着自己的一块市场阵地。对于这些操作系统，很难说哪一个更好，哪一个略差一筹。对于具体的用户来说，适合需求的应该就是最好的。

在这些操作系统中，我们比较熟悉的是微软公司的 Windows，以及自由软件基金会

FSF 资助的 Linux。

1.4.1 Windows 操作系统

Windows 操作系统恐怕是个人计算机用户最熟悉的操作系统之一了。它是微软公司推出的系列操作系统。其实比微软操作系统起步早、性能强的操作系统有很多。但是，微软的操作系统定位很准确，它盯住了巨大的个人计算机用户市场。微软能够有今天的发展，得益于它把握住两个重要的机会。一是在 1980 年 IBM 推出第一台个人计算机时，微软成为 IBM 的合作者，使它的微机操作系统 MS-DOS 成为 IBM 个人机的操作系统。这个机会加上微软的不断努力，MS-DOS 最终成为 16 位操作系统的标准。但是，MS-DOS 是文本命令行界面，用户要对着黑乎乎的屏幕输入命令。如果微软满足 DOS 的成就，那么就不会有今天的 Windows 了，微软可能也不是现在的样子了。在 Apple 公司推出世界上第一个成功的商用图形用户界面系统时，微软公司发现了第二个重要的发展契机——图形用户界面。认识到图形用户界面的重要性以及广阔的市场前景，微软很快推出了 Windows 这个带有图形界面的操作系统。

开始的 Windows 1.0 并不令人满意，也没有引起用户的注意。但是随后微软在 1990 年 5 月份发布的 Windows 3.0 取得了巨大的成功。在此之后，微软又推出了 Windows 3.2、Windows 95、Windows 98。这一系列产品奠定了微软公司在个人计算机操作系统领域的垄断地位，以至于现在有些用户居然抱怨 Linux 的图形桌面环境与 Windows 的风格不一样。

在一系列 Windows 中，Windows 95 之前的版本都是作为一个应用程序运行在 DOS 基础上的(与现在 Linux 下的图形桌面环境类似)。从 Windows 95 开始，Windows 操作系统成为真正的图形界面操作系统。Windows 不再需要 DOS 的支持，图形界面完全由操作系统内核支持。

在个人计算机市场站稳脚跟的微软公司，在 1992 年开始立项研发服务器级产品 Windows NT。在此之前，服务器级操作系统一直都是另一个操作系统 UNIX 的天下。依仗在个人计算机领域的声望及其产品的性能，Windows NT 成功进入了服务器操作系统市场。在 2000 年发布的 Windows NT 5.0 更名为 Windows 2000。在此之后，人们又看到了基于 NT 版内核的 Windows XP。2003 年，微软把它的服务器产品统一在 Windows Server System 品牌下。我们见到的 Windows 2003 就是它的产物。微软目前最新的操作系统是 64 位的 Windows Vista。

Windows 是一个非常出色的操作系统。它有很多优点，如图形界面友好、网络支持便捷、多媒体功能出色、硬件支持灵活、应用程序众多等。总之，它非常好地贯彻了操作系统就应该让用户方便地使用计算机的原则。因而它获得了成功。

小提示

友好的图形界面以及在个人计算机操作系统领域的捷足先登促成了 Windows 操作系统的成功。

1.4.2　Linux 操作系统

Linux 和 Windows 虽然都是操作系统，但是却有着很大的不同。可以说，二者从出生到性格都有着极大的差异。它们不但外表看起来不一样，而且内部实现也有很大差别。

实际上，在众多操作系统中，Linux 是非常不一样的一个，可以说是独树一帜。这样说不是指 Linux 的内部实现技术，而是指它非常独特的运作模式。为此，我们要多花一些笔墨来说说这个比较有个性的 Linux。

Linux 的个性首先在于它是开放源码的。任何人都可以得到它的代码。其次，它是在 Linus 领导下，由全世界的计算机爱好者志愿设计和开发的。第三，它采用了完全不同于商用版权的 GPL 许可证(什么意思？在后面详述)。

开放源码为 Linux 带来了三大好处。首先，这意味着任何人都可以知道 Linux 操作系统是怎样实现的。这样每个人都对 Linux 很放心，不用担心里面藏着什么危险的后门。对于没有自主设计开发操作系统的国家来说，这一点尤为重要。其次，这使 Linux 操作系统成为世界上计算机精英智慧的集成者。Linux 吸引了世界上顶尖计算机技术人的注意力。开放源码机制使这些人只要愿意，只要他们的工作做得足够好，那么他们对 Linux 操作系统的改进就可以被写进 Linux 的新版本。任何一家公司都没有能力说他们聚集了最适合设计和开发其产品的全世界顶级高手，而只有开源的 Linux 操作系统可以这样说。这直接促成了 Linux 操作系统的高性能。第三，尽管 Linux 操作系统的性能一点儿也不比其他操作系统差，但是它很便宜。花几十元就可以买到一个 Linux 操作系统，甚至可以直接从网络上免费下载。而这绝对不会使我们受到侵权的指责。总之一句话，Linux 操作系统的特点就是物美价廉加放心。因此，Linux 操作系统非常被看好。从软件到硬件，很多厂商都推出了基于 Linux 操作系统的产品。我国也正在进行大力推广 Linux 操作系统的工作。因为在信息化的现代社会中，各行各业对计算机的依赖越来越强，而每台计算机上都要有操作系统。仔细想想这可不是个小事情。无论从经济还是安全角度，我国都需要一个在掌控之中的操作系统。但开发操作系统又不是一朝一夕的事情。所以，在这种形势下，Linux 操作系统正好可以解决我们的燃眉之急。

小提示

Linux 操作系统的三个优势：开放源码、性能高、价格便宜。

1. Linux 操作系统的起源

要说 Linux 操作系统，还要从 UNIX 说起。UNIX 是什么？它也是一个操作系统。它是 AT&T 贝尔实验室的两个工程师创造出来的。其中一个人叫做 Thompson。因为他对自己计算机上的操作系统很不满意，所以就想自己写一个能联机玩游戏的操作系统。最终他和同事 Ritchie 实现了这个天才想法，开发出了一个全新的操作系统，就是 UNIX 操作系统。后来很多厂商和大学一起对 UNIX 进行了开发和完善，形成了不同版本的 UNIX 系统。BSD UNIX 和 UNIX System V 是其中两大主流分支。慢慢地，UNIX 变成了一个非常重要的操作系统，占据了统治地位，成为工作站上的主流操作系统。

一开始，UNIX 也是开放源码的，所以很多大学采用 UNIX 源码作为操作系统课程和实验的教材。但是，随着 UNIX 的成功，巨大的商机掀起了 UNIX 的商品化热潮。UNIX 最终不再免费开放源码，演变成了一个非常昂贵的商用系统，代码也受到了版权的保护。这样学校就不能再用最新的 UNIX 源码为学生讲解操作系统原理了。

为了让学生能够有一个合适的操作系统代码了解和掌握操作系统原理，荷兰著名教授 Andrew S. Tanenbaum 写了一个在 PC 上运行的小型类 UNIX 操作系统 Minix。但是，Minix 只是一个教学工具。它的功能和性能与 UNIX 相比有很大差距。它并没有实用价值，所以让人有一种纸上谈兵的感觉。为此，芬兰学生 Linus Torvalds 就萌生了以 Minix 为基础开发一个真正可以实用的类 UNIX 内核的想法(内核？你可以理解成是操作系统的核心部分。一般内核是要常驻内存的)。Linus 的这个意见并没有得到 Tanenbaum 教授的支持。但是，Linus 并没有因此而气馁。他随后开始自己动手实现这个计划。因为这个新操作系统基本上实现的是 UNIX，所以 Linus 把它称为 Linux，即 Linus 的 UNIX。在 Linux 第一个版本基本实现后，Linus 就将其源码放到了当时还不像现在这么流行的网络上，希望大家来参与开发和完善。他的想法得到了很多人的热烈响应，尤其是与一个叫做自由软件基金会(FSF)的组织的想法不谋而合。这个基金会支持很多开放源码的项目。他们也正想支持一个开放的，类 UNIX，但又不是 UNIX 的操作系统项目。他们把这个项目称为 GNU (GNU 是 GNU is Not UNIX 的缩写，有点儿绕啊)。于是由 Linus 主持的 Linux 内核的开发、改进与维护成为 FSF 的主要项目之一。这样在 Linus 的带领下，FSF 的支持下，Linux 在一群志愿者通过网络的松散合作之下一步步地完善起来，最终形成一个能够在 PC 以及大型机上运行的、充满无限魅力的操作系统[2]。

Linus 领导下开发和维护的是 Linux 的内核。而我们实际上使用的是各个厂商或组织在这个内核基础上添加各种应用软件封装的操作系统软件包。这些软件包被称为 Linux 的发行版本。现在常见的发行版本有 Feroda、Redhat、Redflag、Suse 等。这些发行版本在安装程序、安装界面、软件包的数目、软件包的安装和管理方式方面会有不同。但是它们的内核来源都只有一个，就是 Linus 等人主持并负责维护的内核版本。用户可以从相关网站上获得最新版本的 Linux 内核。甚至可以找出其中的不足，加以改进，并且以补丁的形式提交给 Linux 的维护者。现在 Linux 内核的各个部分都有一个专门的维护者负责。这些人都是操作系统设计和开发者中的精英。这些维护者再向 Linus 负责。如果你的补丁通过了这些维护者的审查，被认为是合理有效的，那么就有可能被加入到新版本的 Linux 内核中。你的名字也会随你的补丁进入 Linux 的世界。

2. Linux 操作系统的版权

这里还要说说 Linux 的版权问题。既然是开放源码，是不是就没有版权的说法了呢？也不是这样的。Linux 是有版权的，只不过这个版权归公众所有，由自由软件基金会管理。自由软件基金会为所有 GNU 软件制定了一个公用许可证制度，称为 GPL(General Public License)，也叫 copyleft。它与常听到的 copyright，即版权，完全不同。copyleft 允许用户对作品进行复制和更改。用户可以免费使用 GNU 软件，可以通过源代码重新构建 GNU 软件。GPL 允许用户免费取得源代码，并且加以发布甚至出售。

那么 GPL 对用户有什么义务要求呢？GPL 规定每个 GNU 软件以及在 GNU 软件基础上加以修改得到的软件，必须声明该软件出自或源于 GNU。最重要的是，开发者必须保

证用户能够获得这些衍生软件的源代码。并且用户能够用这个源代码重新构建出一模一样的软件。也就是说提供给用户的源代码应该就是所发布或出售软件的源代码，而不是演示版或者别的什么版本的源代码。这种机制力争保证开源的代码会被延续下去，而不至于到了什么时候就变成商用而结束开源。FSF 要竭力避免 Linux 重蹈 UNIX 的覆辙。不过，如果仅仅调用 GNU 软件提供的接口进行的项目开发，则不受 GPL 的约束[2]。

现在越来越多的厂商和组织在基于 GNU 软件做项目，尤其是基于 Linux 开发产品。商家的加入将市场竞争、经济利益带入开源世界，给 GPL 带来了很多挑战。例如，一些商家虽然在 GNU 软件基础上开发软件，但是为了自己的经济利益，他们隐瞒这个事实，不愿公开源码。针对这种违反 GPL 的情况应该怎么办？随着环境的复杂化，GPL 的一些条款细节还表现出自相矛盾的地方。针对这些新出现的问题，人们正在不断地修改和完善 GPL。

小提示

开发源码、使用 GPL 许可证、在 Linus 主持下由全世界的计算机爱好者志愿开发维护是 Linux 的三大特别之处。

Linux 的版权归公众所有，由 FSF 维护。

3. Linux 的用户界面

Linux 的默认用户接口/界面是由一个称为 Shell 的命令行解释程序提供的。Shell 的英文原意就是“外壳”。这就是说 Shell 是 Linux 操作系统的外壳，是用户看到的操作系统的外部样子。Linux 操作系统启动时，把所有的初始化准备工作都做完之后，就会执行一个等待用户登录的程序。当用户用正确的用户名和密码成功登录之后，系统就会执行 Shell 程序，来等待用户输入控制命令。

Shell 程序运行起来的样子并不好看。用户只会看到一个黑乎乎的屏幕，上面有很少的一些字符串简单地说明着一些事情。前面给出的图 1-1 就是 Linux 的 Shell 界面。这种界面也被称为命令行界面或者字符界面。屏幕上闪烁的光标前通常是一个短短的字符串，如图 1-1 中的[root@zhanglivmware root]#。这被称为命令提示符。命令提示符的末尾有时候是“#”。这表示当前用户是以特权账号身份登录的。而如果末尾符号是“$”，则表示是普通用户。命令提示符中@前面给出了当前登录用户的账号，图 1-1 中是 root。紧接着@的是机器名，这里是 zhanglivmware。在机器名后面空格隔开的是当前所在的目录名，图 1-1 中是 root。

Shell 为用户提供了两种与计算机交互的方式。一种是联机命令。例如，命令 ls 可以查看目录下的文件，pwd 可以显示当前工作目录的名称。用户在命令提示符后输入这些命令之后，Shell 程序就会解释并执行这些命令。因此说 Shell 是命令解释程序。其实，某些命令是由外部其他程序完成的，如复制命令 cp 和移动命令 mv。这些命令是存在于文件系统中某个目录下的单独程序。当用户输入这些命令时，Shell 就会去运行这些程序。

此外，Shell 还提供了自己的编程语言用于对命令的编辑。用户可以利用这些语句来编写程序。通常把这类程序叫做 Shell 脚本。在命令行下输入 Shell 脚本文件的名字，Shell 就会运行这个程序，执行其中的命令。这就是 Shell 提供的另外一种交互方式。它可以看

作是一种脱机控制方式。Shell 编程语言支持高级语言里出现的绝大多数程序控制结构，如循环(while 语句)、分支(if 语句)以及变量等。Shell 提供的是一个解释型的程序设计语言。也就是说，Shell 脚本程序不是全部被编译成可执行文件后再执行，而是被逐条翻译执行的。我们将在后面章节中专门介绍 Shell 脚本的编写。

就像操作系统一样，Shell 有很多不同的版本。常见的 Shell 版本如下：

- Bourne Shell(sh)：由贝尔实验室开发。
- BASH(bash)：GNU 的 Bourne Again Shell，是 GNU 操作系统上默认的 Shell。
- Korn Shell(ksh)：对 Bourne Shell 的发展，大部分与 Bourne Shell 兼容。
- C Shell(csh)：Sun 公司 Shell 的 BSD 版本。

用户可以选择运行自己喜欢的 Shell，一般在每个账号的配置文件中记录用户要使用的 Shell 程序。当用户登录后，系统就会根据配置文件运行相应的 Shell。

除了 Shell 外，Linux 用户也可以通过图形界面来控制计算机。但是图形界面并不是 Linux 提供的默认用户界面，而且与通常使用的 Windows 操作系统的图形界面也完全不同。Windows 的图形界面是操作系统的默认界面，是被操作系统内核支持的。而 Linux 下的图形界面只是一个应用程序。这个应用程序的名字叫做视窗系统。Linux 下的视窗系统不只有一个。但是其中最成功、最流行的是 X-window，也被称为 X 或 X11。实际上，X 只是一个工具套件和架构规范，而不是一个应用软件。人们在这个规范上开发出了很多 X-window 应用程序。在 Linux 下见到的图形界面实际上是桌面环境系统。它们通常是以 X-window 系统为基础构建的。桌面环境通常是一组有着共同外观和操作感的应用程序、程序库以及创建新应用程序的方法。目前，Linux 下最常见的桌面环境是 Gnome 和 KDE。

小提示

Linux 默认的用户界面是 Shell。Shell 是一个命令解释程序。
Shell 为用户提供两种与计算机交互的方式：命令和 Shell 脚本。
Linux 下的图形用户界面也是一个用户级的应用程序。
用户使用的图形用户界面实际上是桌面环境系统。
目前 Linux 下常见的桌面环境有 Gnome 和 KDE。

1.4.3 操作系统分类

为了对众多操作系统进行说明和研究，人们从不同角度对操作系统加以分类。不过，仁者见仁，智者见智，角度不同，目的不同，分类的结果也不一样。因而目前并不存在统一的分类标准。通常，按照管理的硬件规模，操作系统可以分为嵌入式操作系统、微机操作系统、中小型机操作系统和大/巨型机操作系统。而按照响应和处理用户任务的方式则可以分为批处理操作系统、分时操作系统和实时操作系统。如果按照管理资源的内容则有单机操作系统、网络操作系统、分布式操作系统等。随着各个操作系统功能的逐步完善，操作系统之间的区别也变得越来越不明显。

下面简单介绍一些主要的操作系统类型以及典型的操作系统实例，以便能对分类以及相关术语有个简要了解。

小提示

有很多种不同的操作系统。没有统一的操作系统分类标准。从不同角度可以得出不同的分类结果。

按照响应和处理用户任务的方式，可以把操作系统分为批处理操作系统、分时操作系统和实时操作系统。

1. 批处理操作系统

批处理操作系统是在简单监控程序之后，分时操作系统出现之前，产生的一类操作系统。所谓批处理就是指计算机采用批量的方式处理用户提交的任务。在计算机出现的初期，用户不能直接控制计算机。用户需要把程序、相关数据以及控制程序执行的一些说明交给操作员。操作员把一批任务集中起来输入计算机。这些任务被称为作业。操作员把作业输入计算机后，操作系统根据程序员提供的作业说明书控制计算机完成作业。执行完毕，操作员把结果返回给用户。在操作系统的控制下，这些作业的执行过程形成一个自动转接的、连续处理的作业流。人们把这种操作系统称为批处理系统。

批处理系统省去了人工切换作业的时间，提高了计算机系统的使用率。但是，程序员只能通过作业说明书来控制计算机。用户把作业交给系统后，就不能对作业运行中出现的意外情况进行干预了。批处理系统的主要优点是系统的吞吐量大、资源利用率高、操作系统的开销较小。它的缺点是作业处理的平均周转时间较长、用户交互能力较弱等。

现在已经很少见到单一的批处理系统了。批处理的概念和技术已经被融合在现代操作系统中，成为一种不可缺少的功能服务模块。现今的批处理也与传统的批处理有了很大的不同。原来的批处理系统中作业只能顺序执行、用户不能干预，现在的批处理则支持可控的顺序执行和有限的用户干预，有些还支持高级逻辑编程和控制的功能。UNIX 和 Linux 都是支持批处理的操作系统。它们的批处理功能通过 Shell 编程来实现。

2. 分时操作系统

使用批处理操作系统，用户与计算机的交互很少，所以会感觉很不方便。用户更希望操作系统能够提供直接与计算机面对面、你来我往的操作方式。可是，相对于计算机的运算速度，人的操作速度又实在太慢了。如果支持人与计算机交互，那么计算机就要经常等待用户的反应。为了不浪费资源，操作系统的设计者想出了一个两全其美的好办法：让一台计算机同时给多个用户服务。每个用户有一个终端。这些终端都连接到同一台计算机上。这台计算机的 CPU 轮流查看每个用户现在要它做什么。如果需要它做事情，它就飞快地去做，然后再去查看下一个用户。等 CPU 巡视一圈回头再来查看第一个用户的终端时，可能他已经有了新的命令给它，或者他还没有把命令输入进去呢。为了公平起见，CPU 为每个用户服务的时间长度都相同。这段时间被称为时间片。如果一个用户的时间片用完了，那么 CPU 就会放下他的工作去为下一个用户服务。这样可以保证别的用户不会等待太长时间而有所察觉。因为 CPU 的速度实在比人快得多，而且时间片又足够短，所以每个终端前的用户都感觉不到 CPU 在完成它的任务之前或之后还为其他人做了很多事情。他感觉计算机就是在为他一个人服务。采用这种方式完成任务的操作系统被称为分时操作系统。早期的计算机工作站就是这种情形。那时的计算机非常昂贵，也没有个人计算机。这样可以大大提高计算机的利用率，做到物有所值。那时到机房看到的是成排的显示器和键盘，可以供

几十人同时上机工作，而实际上整个机房只有一台被称为工作站的计算机。

现在几乎所有的操作系统都采用分时机制，而且不只对交互程序采用分时办法。计算机上的多数任务都采用这种方式来分享 CPU。即使现在我们用的个人计算机是绝对的每个人独享，但是分时技术仍然存在其中。例如，你在用编辑软件写文章时，打开了媒体播放器一边听歌一边写。可实际上你的计算机只有一个 CPU。这时 CPU 就是采用分时机制来运行编辑软件和媒体播放器。因为 CPU 的速度足够快，所以你根本没有感觉到它的分时工作。听到的歌曲不是时断时续的、编辑软件也运用自如。当然，这都要感谢操作系统繁忙而有序的指挥工作。

3. 实时与嵌入式操作系统

如果把计算机用在工业或者军事控制中，如飞机自动驾驶、导弹制导等，那么计算机必须能够根据实际情况做出及时的反应才行。而批处理和分时这两种处理任务的方式都可能会耽误事情。因此，在这些情况下，操作系统必须改变行事方法，在规定时间内做出合适的响应。能够完成实时任务的操作系统，被称为实时操作系统。为了支持实时性，实时操作系统显然要采取不同的方式来实现。

大部分实时系统是工作在那些嵌入或者隐藏到其他应用系统中的结合微处理器或者微控制器的系统电路上。这样的计算机系统被称为嵌入式计算机系统。它们不再以物理上独立的设备或者装置出现。我们可以在很多地方看到嵌入式系统的影子，如移动电话、能听懂语音指令的玩具、能够自动调节温度的冰箱，以及电话交换设备、印刷机、零售设备、智能卡等。因此实时性和嵌入式常常是同一个系统的两个特性，所以我们常听到实时与嵌入式操作系统的称呼。受应用环境和成本的限制，嵌入式系统的硬件条件相对都比较弱。例如，这些系统的 CPU 处理能力都比较低，通常可以使用的内存也非常小。所以在这些环境条件限制之下的操作系统的设计与实现必然与普通计算机上的有很大不同。

现在世界上有很多种不同的实时与嵌入式操作系统，例如 WindRiver 公司的 VxWorks，微软公司的 Windows CE，以及开源 Linux 的嵌入式版本，如美国墨西哥理工学院开发的 RT-Linux、Lineo 公司的 μCLinux 等。

4. 通用操作系统

如果一个操作系统兼有批处理、分时处理和实时处理三者或者其中两者的功能，那么这个操作系统就被称为通用操作系统。UNIX 和 Linux 都属于通用操作系统。

这些操作系统根据任务需求的不同来安排处理方式。例如，在具有批处理和分时处理功能的操作系统中，当有用户交互时，系统就采用分时方式对其进行响应；当系统没有或者较少用户交互时，就去处理批处理作业。同样，操作系统可以同时支持分时处理和实时处理：当有实时任务发生时，系统优先响应实时任务；没有实时事件发生时，系统其他任务就分时使用 CPU。

小提示

批处理操作系统是指采用批量的方式处理用户提交任务的操作系统。
批处理操作系统提供作业控制语言让用户描述作业的执行步骤。
分时操作系统是指让多个用户/任务按照时间段轮流分享 CPU 的操作系统。
实时与嵌入式操作系统是指工作在嵌入式系统中能够及时对外界情况做出响应的操作系统。

附文：

计算机操作系统发展历史

文/维基百科

操作系统的历史在某种意义上来说也是计算机的历史。操作系统提供对硬件控制的调用和应用程序所必需的功能。

背景

早期的计算机没有操作系统。用户有单独的机器，他(她)会带着记录有程序和数据的卡片(punch card)或较后期的打孔纸带去操作机器。程序读入机器后，机器就开始工作直到程序停止。由于程序难免有误，所以机器通常都会中途崩溃。程序一般通过控制板的开关和状态灯来调试。据说图灵能非常熟练地用这种方法操作 Manchester Mark Ⅰ机器。

后来，机器引入帮助程序输入输出等工作的代码库。这是现代操作系统的起源。然而，机器每次只能执行一件任务。在英国剑桥大学，这些任务的磁带从前是排成一排挂在衣钩上的，衣钩的颜色代表任务的优先级。

概念意义上的操作系统和通俗意义上的操作系统差距越来越大。通俗意义上的操作系统为了方便而把最普通的包和应用程序的集合包括在操作系统内。随着操作系统的发展，一些功能更强的“第二类”操作系统软件也被包括进去。在今天，没有图形界面和各种文件浏览器已经不能称为一个真正的操作系统了。

大型机时代

早期的操作系统非常多样化，生产商生产出针对各自硬件的系统。每一个操作系统都有很不同的命令模式、操作过程和调试工具，即使它们来自同一个生产商。最能反映这一状况的是，厂家每生产一台新的机器都会配备一套新的操作系统。这种情况一直持续到 20 世纪 60 年代 IBM 公司开发了 System/360 系列机器。尽管这些机器在性能上有明显的差异，但是它们有统一的操作系统——S/360(在开发 S/360 过程中遇到的问题在 Fred Brks 写的软件工程经典之作《人月神话》中详细地叙述了)。

S/360 的成功陆续地催化出 MFT、MVT、SVS、MVS、MVS/XA、MVS/ESA、S/390 和 z/S。

小型机和 UNIX 的崛起

UNIX 操作系统是由 AT&T 公司开发出来的。由于它的早期版本是完全免费的，可以轻易获得并随意修改，所以它得到了广泛的接受。后来，它成为开发小型机操作系统的起点。由于早期的广泛应用，它已经成为操作系统的典范。不过，它始终属于 AT&T 公司，只有那些能负担得起许可费的企业才用得起，这限制了它的应用范围。

早期的操作系统是可以被用户软件所利用的功能的集合。一些有能力的公司发展更好的系统，但他们不支持其他公司硬件的特性。

20 世纪 60 年代末 70 年代初，几种硬件支持相似的或提供端口的软件可在多种系统上运行。早期的系统已经利用微程序来在他们的系统上实现功能。事实上，除了 360/165 和

360/168 外，360/40 之后的大部分 360 系列的机器都实行微程序设计。

个人计算机时代：Apple，DOS 和以后

微型处理器的发展使计算机的应用普及至中小企业及个人爱好者。而计算机的普及又推动了硬件组件公共接口的发展(如 S-100，SS-50，Apple Ⅱ，ISA 和 PCI 总线)，并逐渐地要求有一种“标准”的操作系统去控制它们。在这些早期的计算机中，主要的操作系统是 8080/8085/Z-80 CPU 用的 Digital Research's CP/M-80，它建立在数码设备公司(Digital Research)几个操作系统的基础上，主要针对 PDP-11 架构。在此基础上又产生了 MS-DOS(或 IBM 公司的 PC-DOS)。这些计算机在 RAM(只读存储器)都有一个小小的启动程序，可以把操作系统从磁盘装载到内存。IBM-PC 系列的 BIOS 是这一思想的延伸。自 1981 年第一台 IBM-PC 诞生以来，BIOS 的功能得到不断地增强。

随着显示设备和其处理成本的降低，很多操作系统都开始提供图形用户界面。如许多 UNIX 提供的 X Window 一类的系统、微软的 Windows 系统、苹果公司的 Mac 系统和 IBM 公司的 OS/2 等。最初的图形用户界面是由 Xerox Pal Alt 研究中心 20 世纪 70 年代初期研发出来的，之后被许多公司模仿，继承发展。

操作系统列表

早期操作系统(专利保护)：TRS-DOS、ROM OS's、TI99-4、Commodore PET、VIC-20、第一套 IBM-PC、Apple、Sinclair Micro。

非 UNIX 商业操作系统：CPM 操作系统、MP/M-80、UCSD P-system、Mini-FLEX、SSB-DOS、CP/M-86、DR-DOS、FreeDOS、MS-DOS、PC-DOS、Mach、Multics、OS-9。

UNIX 及类似系统：UNIX、微软 Xenix、Cromix、UNIflex、OS-9、IBM 的 AIX、BSD、FreeBSD、NetBSD、OpenBSD、Digital UNIX、DNIX、HP 的 HP-UX、GNU/Hurd、SGI 的 IRIX、Linux(或称 GNU/Linux)、MacOS X、OSF/1、SCO 的 SCO UNIX Sun 的 SunOS 即 Solaris、System V、Ultrix、UniCOS、LindowsOS、OS/390、z/OS、Syllable。

其他：Acorn：RiscOS、Arthur，Amiga：AmigaOS，Atari ST：TOS、MultiTOS、MiNT。

苹果电脑(Apple/Macintosh)：Apple DOS、ProDOS、Mac OS、Mac OS X。

Be：BeOS 、BeIA。

Digital/康柏(Compaq)：AIS、OS-8、RSTS/E、RSX-11、RT-11、TOPS-10、TOPS-20、VMS(OpenVMS)。

IBM：OS/2、AIX、OS/400、OS/390、VM/CMS、DOS/VSE、OS/360、MFT、MVT、SVS、MVS、TPF、ALCS、z/OS。

微软(Microsoft)：MS-DOS，Xenix：Windows 1.0、Windows 3.1、Windows 95、Windows 98、Windows ME、Windows Longhorn、Windows NT：Windows 3.5、Windows 4、Windows 2000、Windows XP、Windows Server 2003。

Novell：NetWare。

NeXT：NeXTStep(Mac OS X)、Plan 9、Inferno。

Prime Computer：Primos。

西门子：BS2000。

个人电子助理(PDA)操作系统：Palm OS、Pocket PC、EPOC、Microsoft Windows CE。

复习题

1. 操作系统平台与终端用户的接口有哪几种形式？
2. 操作系统平台提供给终端用户的基本功能有哪些？除了书上提到的，你还知道哪些？
3. 操作系统的程序员级接口与终端用户接口有什么不同？为什么？
4. 系统调用与普通函数的区别是什么？
5. 什么是操作系统？操作系统的主要任务是什么？
6. 请画出计算机系统的层次结构图，并说明终端用户接触层次结构中的哪些层？应用程序员会接触哪些层？
7. 操作系统向用户屏蔽硬件细节有什么优缺点？
8. Windows 操作系统有什么优势？它为什么能成功？
9. Linux 有版权吗？
10. GPL 对于用户有什么义务要求？
11. Linux 操作系统有什么特点？
12. Shell 是什么？
13. 什么是 Linux 的桌面环境系统？
14. 什么是分时操作系统？你知道的操作系统中哪些是分时操作系统？
15. 什么是实时与嵌入式操作系统？你知道的操作系统中哪些是实时与嵌入式操作系统？

讨论

1. 为什么要使用操作系统平台？而不是让用户直接面对裸机？
2. 计算机系统为什么要组织成层次结构？它的实质是什么？
3. 同样是设计和开发程序，操作系统的设计和开发者的工作与应用程序员有什么不同？为什么？
4. 你认为教学版软件和实用版软件有何区别？
5. 查看有关 GPL 的文档和资料，看看目前 GPL 存在哪些问题。对于这些问题，你有什么建议吗？

实验

1. 在 Linux 操作系统上，练习复制文件、创建目录以及运行程序的命令。
2. 查阅相关 Linux 命令帮助资料。登录 Linux 操作系统，查看系统中有多少进程在运行。指出 init、cron、inetd 这些进程的 ID 号是多少。查看系统中有多少 sh、bash、csh 进程在运行。显示系统中进程的层次结构。
3. 请查阅有关 Linux 的日志命令。然后登录 Linux 系统，查看系统中的用户登录、注销及系统的启动、停机，以及系统上正在和曾经运行的进程的情况。
4. 查阅有关资料，在 Linux 系统的/proc 目录下找到系统内存状态的信息，如系统中空闲内存、已用物理内存和交换内存的总量等。
5. 在 Linux 操作系统中，查看自己账号主目录下的.bash_profile 文件的内容。并设置环境变量，使在当前目录下运行程序时，不用在程序名前再输入表明当前目录的符号“./”。
6. 尝试在 Linux 和 Windows 下分别打开一个小文件和一个长度超过 100M 的大文件，统计打开文件所用的时间，体会两个操作系统的区别。

第二篇　计算机硬件平台

上一篇中对计算机系统的操作平台——操作系统有了一个初步认识。在进一步了解操作系统处理事务的原理及策略以便更好地使用这个平台之前，需要先了解计算机硬件平台的一些基础知识，因为硬件是计算机做一切工作的基础。尽管操作系统平台已经为我们屏蔽了硬件的许多细节，但是了解必要的硬件知识对我们理解系统的行为、合理使用和配置系统，以及正确使用应用软件是非常必要的。本篇首先将介绍计算机系统的硬件组成，即计算机是由哪些实实在在的部件构成的，以及各主要部件的工作原理。介绍的主要部件包括计算机的存储器、中央处理器(CPU)，以及外围设备，还有负责把这些部件联系起来的总线。这些硬件之间的关系也是比较重要的内容，即这些组成部件之间是怎样合作来形成完整、功能强大的计算机系统。最后还会简单介绍计算机系统结构方面的一些知识，为读者提供了解计算机系统的另外一个角度。

除了计算机系统的硬件组成外，本篇还将介绍各种信息在计算机中表示方式，即软件和各种数据在计算机硬件上记录的基本形式。正是这些表示方法使计算机软件和数据能够以数字化的形式表现出来，得以由计算机硬件来承载。因为信息的表示受硬件制约，与硬件自身有一定的关系，因此也把相关内容放在这一篇中介绍。

第 2 章　计算机硬件组成

本章先从整体上介绍计算机，看看计算机的主要组成部分以及这些组成部分之间的逻辑关系。下面将对这些组成部分做详细介绍(参考文献[14-20])。

现在计算机对大多数人来说都不陌生了，图 2-1 是目前较常见的台式机的样子。从外观看来，台式机由几个部件构成：显示器、键盘、机箱、鼠标，有些还可能有一个耳麦。一提到计算机，我们脑海中首先想到的可能就是这些。实际上，最引人注意的显示器、键盘，还有鼠标，从严格意义上来说，并不是计算机系统中最重要的部分。它们只是被用来向人们汇报计算机正在进行的活动，或是将人的命令传达给计算机，就像人类的感觉器官一样。而计算机最主要的工作实际上是由那些被机箱遮盖住的部件完成的。它们才是计算机系统中最重要的部分。

一个运转的计算机系统在某种程度上与一个活着的人有些相像。为了说明这一点，现在假设发生了下面这件小事情。某一天，你的老师突然对你说："你最近学习很不努力！"你的耳朵把这句话传送到你的大脑。你的大脑思考这句话，发现老师说的并不是事实。所以你感觉很委屈，脸涨得红红的，想说点儿什么。现在我们就来把计算机和你类比一下。计算机的键盘或者鼠标就好像你的耳朵(其实麦克风更相像一些)；显示器以及上面显示的内容就好像你的脸和脸色；而与你的大脑类似的东西则在机箱里面，叫做中央处理器(Central Processing Unit，CPU)。

我们把鼠标、键盘和显示器这类设备叫做外围设备，简称外设，也叫输入/输出设备，即 I/O(Input/Output)设备。外设不只有这三种。打印机、绘图仪、数码相机、扫描仪，还有磁盘(严格地说，是磁盘驱动器)都属于外设。就像你的感觉器官除了耳朵、皮肤之外，还有眼睛、嘴和鼻子一样。随着技术的发展，计算机会变得和人越来越像，就像你在科幻片中看到的一样(没有做不到，只有想不到，这句话好像有点儿道理，所以如果你喜欢白日做梦，也不是件坏事情。只不过别把时间都花在做梦上，留出一点儿来做事，有一天你会发现现实与梦境的差距会越来越小)。

还是把话题转回到计算机上。下面到机箱里面看一看，考察一下刚才说的重要部件到底都是什么，长成什么样子。图 2-2 是一个打开的机箱。在图上，最大的一个部件是位于侧

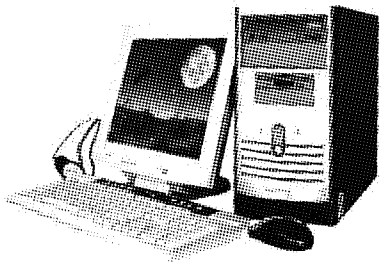


图 2-1　台式机

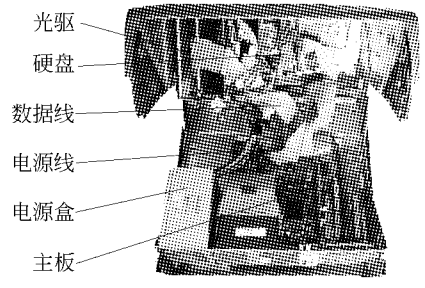


图 2-2　计算机内部图

面的那块电路板，上面布满元器件，还插着形状各异的板卡。它叫做主板。把主板上插着的这些板卡或部件取下来，可以看到在主板上有一些插槽，如图 2-3 所示。电路板或者部件就是通过这些插槽连接到主板上的。插槽的形状不都是一样的，有的长，有的短，有的有几个，有的只有一个。

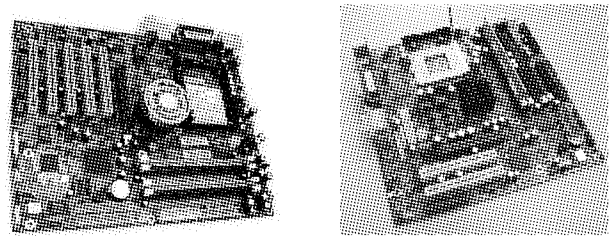


图 2-3　主板

从主板上取下来的部件中有一个方方的，个子比较大的芯片，上面还带着一个风扇。它就是计算机最重要的部件 CPU 了。图 2-4 是一些常见国外品牌的 CPU。图 2-5 是我国的龙芯系列 CPU(左侧是龙芯一号通用嵌入式 CPU 芯片，右侧是龙芯二号增强型处理器“龙芯 2E”。CPU 通过很多密密麻麻的小针脚插在主板的一个插槽上。图 2-3 中左边的主板就插上了一个 CPU，上面带有一个风扇。而图中右边的主板上没有 CPU，所以，我们可以清晰地看到方方的、白色的 CPU 插槽。

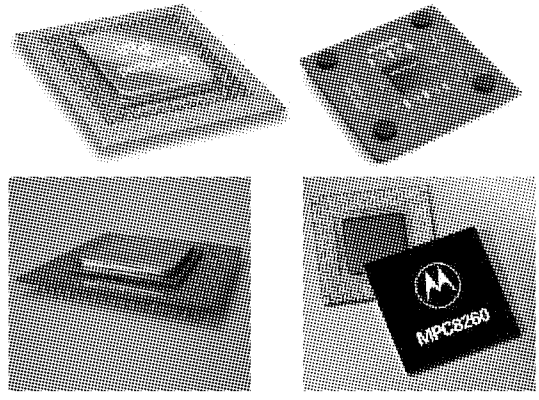


图 2-4　CPU

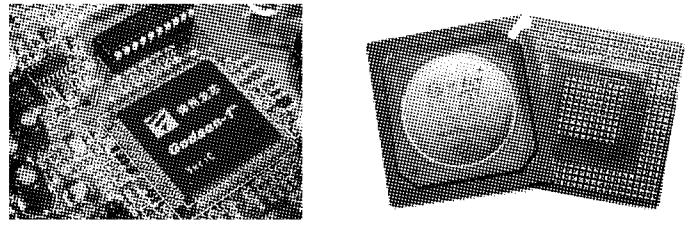


图 2-5　龙芯系列 CPU

从主板上取下的板卡中有一种不起眼的小板卡。通常是一个小小的长条，有时候是两

条，或者更多。它就是内存条。计算机用它来存放 CPU 要执行的命令、用到的数据以及计算的结果，有点儿像 CPU 的草稿纸。图 2-6 是一些内存条的外观。图 2-3 中右侧的主板上就插着一条内存，而左边的则没有，该主板上最长的两个蓝色插槽就是用来插内存条的。

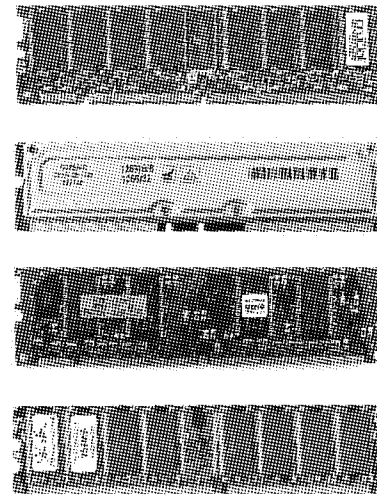


图 2-6　内存条

对了，在机箱的一角还有一个盒子，它通过一组并排的电线和主板相连，并且和其他的一些部件也有类似的线连接，这就是电源盒。它是计算机的动力，有点像人的心脏。有了前面看到的这 4 件东西：主板、CPU、内存以及电源，一个计算机基本上就可以工作了。也就是说这 4 件东西是缺一不可的。与此不同的是，外设是可以相互替换的。比如，用游戏手柄代替键盘，用绘图仪替换显示器。通常把以内存和 CPU 为核心构成的硬件部分并称为主机。从计算机硬件工程的角度看，主板是一个很重要的概念，它上面精心设计安排的布线、插槽、阻容元件等形成了一个能够支撑 CPU 和内存等部件灵活配置、协调运行的平台。

除了这几件最重要的东西外，机箱中还有什么？固定在机箱边上的还有光驱（光盘驱动器）、硬盘（是硬盘和硬盘驱动器的结合体）。它们都通过一排线（我们称之为数据线，见图 2-2 标注）连接到主板上。有的计算机上还有软盘驱动器。当然现在的计算机已经越来越少配备这个部件了，取而代之的是更便于携带，容量也大很多的 U 盘。这些部件除了和主板相连外，分别都还有一组线（称为电源线，见 2-2 标注）同电源盒相连。可以看出这些设备需要独立的供电，不是通过主板获得能量的。主板上除了插有 CPU 和内存条外，还有一些插在插槽上的电路板。其中最常见的就是连接显示器的显示卡。它负责处理与显示有关的一些信息。常见的板卡还有图像采集卡、网卡（网络适配器）等。可以看到所有的部件都连接到了主板上，所以可以很大胆地猜测：所有的部件都是通过主板彼此联系的，事实就是如此。

小提示

以内存和中央处理器（CPU）为核心构成的硬件部分被称为主机。
负责输入输出的设备，即 I/O 设备，被称为外围设备，简称外设。
主板是支撑计算机各个部件灵活配置、协调运行的平台。

2.1　计算机组成结构

现在我们对计算机的主要硬件构成已经有了初步认识。那么这些部件是怎样合作完成我们交给计算机的任务的呢？图 2-7 是计算机系统的一个逻辑组织框图。图中把计算机的硬件组件按照功能分成了五大部分：控制器、运算器、存储器、输入设备和输出设备。控制器和运算器构成 CPU。我们看到机箱里面的 CPU（见图 2-4 和图 2-5）就是封装了这两部分的

一个小盒子。这种计算机结构被称为冯·诺依曼(John Von Neumann)结构,因为是由一个叫冯·诺依曼的人提出来的。在这种结构中,指令(也就是程序)和指令要处理的数据都放在同一个存储器中,指令的内容也可能根据需要在程序运行时改变(这是一个很重要的概念)。指令和数据在控制器的控制之下经过输入设备、运算器最后存放到存储器中。控制器从存储器中读取指令进行翻译,然后按照指令给有关的部件发控制信号。如果需要运算,控制器就命令运算器来做这件事情。运算器从存储器中取出需要加工的数据,然后将结果写回到存储器。输出设备则可以显示运算结果。

最初的冯·诺依曼结构并不是图 2-7 这样的,而是以运算器为中心,输入输出设备如果想和存储器通信必须要经过运算器。现在图 2-7 的计算机系统结构改成了以存储器为中心的结构。在这种结构下,某些时候,如果没有必要的话,输入输出设备可以不通过 CPU 直接与存储器交流。这样就可以把 CPU 解放出来,让它去做必须由它来做的事情,而不是事无巨细,事必躬亲,浪费它的宝贵时间。因此,CPU 可以解释更多的命令,求解更多的问题。CPU 的利用率得以提升,整个计算机系统的效率也就提高了。不过这种系统结构在本质上并没有改变统一存储程序和数据的特性,所以仍然被称为冯·诺依曼结构。

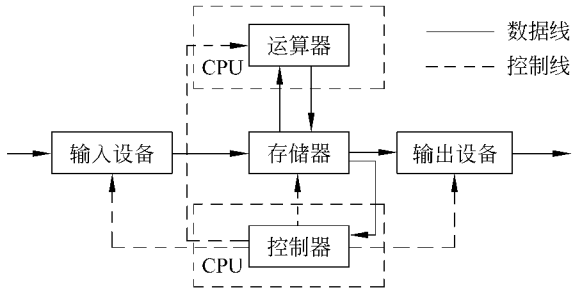


图 2-7　以存储器为中心的计算机结构

在进一步介绍这些部件怎样合作之前,我们要先说说存储器。从名字上,就知道它是用来存储程序和数据的。不过,你知道它是怎样存储程序和数据的吗?从硬件构成上来说,存储器是由很多相同的、有记忆功能的一组一组的器件构成的。记忆功能?噢,其实也就是这些器件在某段时间内可以维持某种状态,比如,保持某种电压的带电状态。这里先把这样的一个器件叫做一个存储单元。存储器的每个单元都有两种可能的状态,比如,电平的高或低。这些单元可以根据要求保持在两种状态中的一种。因此,就可以用一种状态,比如高电平,表示二进制的数码 1;用另外一种状态,如低电平,表示二进制的数码 0。这样如果我们让某个单元带电,那么就说它存储着数码 1,反之,就说它存储着数码 0。这样就好像它能够记住数码 1 和 0 似的,所以说它有记忆功能。因为二进制数的数码只有两个:0 和 1,所以用 n 个这种双状态的存储单元就可以表示 n 位的二进制数了。比如,用 5 个这样的单元,就可以表示 00000 到 11111 范围之内的某个二进制数。

现在你能明白为什么计算机使用二进制数而不是你熟悉的十进制数了吗?十进制数?那可是有 0～9 一共 10 个不同的数码需要表示。按照前面的情况类推,那就需要一种能保持 10 种状态的器件。这样的器件可是很复杂而且很昂贵的呀!其实使用二进制数的好处还不只这一个,如果你深入了解计算机的运算过程就还会有新的发现。这就留给你自己去

发现吧，你可以从计算机组成原理书中找到线索。当然，使用二进制数可能给你带来了一点儿小小的麻烦，因为我们都习惯采用十进制数。小时候开始学算术的时候，用的都是手指头，而手指头有 10 个。不过没关系，后面的章节会介绍二进制数以及计算机中数的表示。这里要感谢计算机系统工程师们的工作：现在程序员并不需要使用二进制数来操作计算机。对于我们来说，这真是一个伟大的进步！

现在我们已经知道存储器是由很多有记忆功能的单元构成的，每个单元可以存储一位二进制数码了。向某个存储单元存储数据也很好理解了，也就是根据情况设置该存储单元的状态就行了。而读取数据也就是识别该单元的状态。在使用这些一模一样的单元之前，我们需要区分开这些单元，不然存进去的东西，取的时候到哪里找啊。所以我们需要把这些单元编上号。实际中，人们并不是把能够存储一个二进制数码的存储单元作为单位来对这些器件进行编号的，因为人们通常会一起需要很多个位来表示一个较大的数，毕竟一位二进制数只能表示 0 和 1 两个数，也太小了。所以，人们把 8 个单元编为一组，然后再进行编号。每个单元中存储的一个二进制数码叫做一个位，也称为比特(bit)。8 位编为一组叫做一个字节(byte)。所以，我们可以说存储器是按照字节编号的。因此，存储器的访问(读/写)是以一个字节为单位的。从现在开始讲存储器的存储单元不再是指能够存储一位的小单元了，而是指 8 个邻接小单元构成的一个小组。每个单元的编号叫做该存储单元的地址，就像小明宿舍的房间号一样。这个编号过程也叫做编址。好了，现在有了这个宿舍号，我们就能够通过它访问存储器中指定的地方了。不过别忘了，通过这个地址我们一次至少可以得到的 0/1 数码是 8 个。

小提示

计算机硬件组成按照功能可以分为控制器、运算器、存储器、输入设备、输出设备五大部分。

存储器是由很多有记忆功能的单元构成的。存储器中存储信息的最小单元只能维持两种状态，分别表示信息 0 和 1。

存储器是按照 8 个位(bit)为一个单位进行编址的。

下面继续讨论计算机结构框图中的其他内容。图 2-8 是对计算机组成结构的进一步细化。从图中可以看出 CPU 里面的内容多了起来。运算器被细化成一个算术逻辑单元

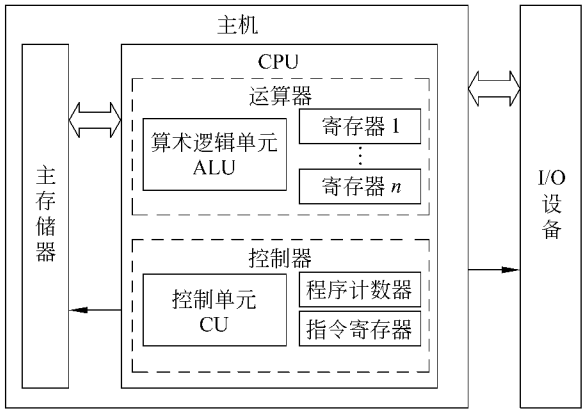


图 2-8　细化的计算机组成结构图

(Arithmetic Logic Unit,ALU)和一组寄存器。而控制器则由一个控制单元(Control Unit,CU)、一个程序计数器(Program Counter,PC)以及一个指令寄存器代替。存储器名字前面也多了一个字,变成主存储器。

这里看到了一些新的事物,比如程序计数器、寄存器,还有算术逻辑单元以及控制单元。先解释一下它们的功能。程序计数器是用来暂存一个地址的。地址——我们刚刚解释过的——就是存储器某个单元的编号。程序计数器暂存的这个地址非常重要。它是 CPU 应该执行的下一条指令在存储器中的位置。程序计数器本身也是由具有记忆功能的元器件构成的,也就是说它实际上也是一种存储部件。但是它不叫存储器,而叫寄存器。所以你能猜到图 2-8 中的存储器为什么变成主存储器了吧?因为可以称为存储器的部件太多了。

从寄存器中存/取数据的速度要比主存储器快很多。但是因为它的造价太高了,所以不能用它来制造容量较大的主存储器。而只能把它用在关键的地方,暂时寄存一些经常用到的信息,比如下一条指令的地址。因而寄存器的容量相对于主存储器来说要小得多。这样做可以使重要的信息能被 CPU 迅速得到,不至于耽误 CPU 的工作,影响整个系统的执行效率。除了程序计数器之外,CPU 里还有一些别的寄存器。比如,用来暂存从主存储器中取出的指令的指令寄存器。而运算器中的寄存器则用来存放运算用到的操作数以及运算的中间结果,以减少访问主存的次数,节省时间。

控制单元是控制器的核心。它从主存储器中读取指令到指令寄存器,对其加以翻译和解释,然后根据指令的内容给其他部件发控制信号。每读取一条指令,控制单元都会将程序计数器加 1,或者按照指令中的要求将其设置成要求的值。所以当 CPU 执行完当前指令时,控制单元就可以按照程序计数器指引读取下一条指令并执行了。

现在你一定发现一个不太小的问题。因为前面说过,主存储器中的程序是在控制器的指挥下通过输入设备导入的,控制器是根据程序中的指令控制其他部件的(包括输入设备),而指令则是控制器从存储器中读取的。这好像是个鸡生蛋、蛋生鸡的过程,那么这个循环是怎么开始的呢?第一条指令是怎么进入存储器的呢?这就是系统的自举问题。噢,这个问题一定是要解决的,不过我们把它先放下,到存储器小节以及计算机系统启动过程一节中再来解决。

下面继续讲 CPU 的事情。在控制单元读取的指令中,有些会要求控制单元做一些协调和控制工作,比如要求计算机跳到程序中的某个位置去执行从那里开始的一段代码。这时控制单元就会按照要求,把跳转的地址放到程序计数器中。而有些指令是要求进行运算的,例如,算算这个月剩下的钱还够吃几次宫保鸡丁的。在这种情况下,控制单元会指示运算器,实际上是里面的算术逻辑单元,来完成这个任务。从名字可以看出,算术逻辑单元是能够进行算术运算和逻辑运算的一个部件。算术运算指对数据进行加减乘除四则运算或者数据格式的转换。而逻辑运算指对数据进行与、或、非,以及移位等运算。算术逻辑单元在进行运算时用到的数据多数是在主存储器中。程序指令中会给出所需数据的地址。算术逻辑单元根据地址到主存储器取操作数,然后加以运算,并把结果写回到主存储器。存放运算结果的存储单元的地址也在指令中给出。有时候指令中也会直接带有要参加运算的数。运算器中有一些寄存器存储经常用到的数据或者运算的中间结果。访问寄存器要比主存储器快,因此,这样可以提高计算机的效率。有些运算和上一次运算的结果有关系,比如,上一次

运算有没有进位会决定这一次加法运算要不要把进位加上。所以，除了参加运算的数据外，算术逻辑单元还需要经常用到一些和运算结果相关的状态值。这些状态值也存储在寄存器中，由相关的部件来设置。这类特殊的寄存器通常叫做状态寄存器。

小提示

寄存器是访问速度比主存储器快的存储器件。CPU 内部有很多寄存器，用来暂存经常用到的数据。

程序计数器是一个位于 CPU 内部的寄存器，它里面存放着下一条要执行的指令在主存中的存放位置。

在向主存储器存数或从其取数的过程中，CPU 和主存储器之间涉及三种信息的传递：地址信息、要存取的数据以及控制信息。控制信息用来指明是读取还是存储数据。为了传送这三种信息，CPU 和主存储器之间有三种连线：地址总线、数据总线以及控制总线。地址总线由一些特殊的信号线和电路构成，当一个地址放到它上面时，它就会打开相应存储单元的门。这时如果控制总线上的信息指示本次操作是写，则放到数据总线的数就会进入该存储单元；如果是读，那么存储单元中的内容就会出现在数据总线上。那么这三类总线在机箱中的什么位置呢？你可能已经想到了。对！当然是在主板上了，这就是主板的作用。可是为什么不说数据线、地址线，而说数据总线、地址总线呢？总线到底是什么意思呢？我们将在总线小节中给出详细的说明。

现在来总结一下，构成计算机系统的这些部件是怎样合作完成任务的。还是通过节俭的小明来总结吧。小明在进入二年级以来，一直处于入不敷出的状态。经过深思熟虑他准备卖掉网球拍。网球场地费太贵了，他的预算中已经好久没有这笔开支了。此外，他准备忍痛割舍他最昂贵的一件不动产——计算机。这台计算机是当初他为了学习程序设计向父母申请资助买的，还赔上了他半年的零花钱。可是他现在发现计算机基本上成了游戏机，在公共机房写程序的性价比似乎更合理。在卖掉计算机之前，小明希望最后再利用一下自己这台聪明的机器，让它算算卖掉这台旧计算机和球拍他可以得到多少钱。于是他写了个小程序。他通过控制器指挥下的输入设备，就是键盘，把程序放到存储器中。然后他命令计算机执行他的程序。控制器把程序第一条指令的地址设置在程序计数器中之后，便开始按照程序计数器的指示位置读取指令放到指令寄存器中，并将程序计数器的值加 1。控制器把取到的指令翻译成发送给运算器的控制信号，告诉运算器把卖掉旧计算机和球拍的钱加在一起。运算器从存储器中取出小明估计旧计算机以及球拍可以卖到的价钱，将它们相加，然后将结果写回到存储器中。小明可以让这个过程变得复杂一些，比如，根据计算机的型号、使用时间等，来查查目前市场上的一个行情，然后推断出可能卖掉的好价钱。不过这需要小明一件一件仔细地写成指令告诉计算机怎么去做。计算机现在还不会主动查看自己的新旧程度，再去查找一个参照表，然后估算出自己的价钱。最后，控制器将再次按照小明的指令通知输出设备，也就是显示器，显示出他可能的收入。结果小明就从显示器上看到了一个可能比他买计算机时少很多的一个数字。

计算机到底是怎样工作的？

2.2 总线

总线在英文中对应的词是 bus。这个英文词的另外一个意思大家都清楚，就是公共汽车。公共汽车是干什么用的？如果公众有一些相同或者基本相同的交通需求，那么就可以设计一路公共汽车来满足那种需求。公共汽车有两个基本的属性，一是为公众所共享，二是路线和车站是固定的。前者引来了“容量问题”，即乘车的人如果太多了可能就有上不去的，后者则引来了“规则问题”，即乘客只能在固定的地方上下车，尽管那些地方可能不是他的出发点或目的地。在学完这一节后，你也许能体会到为什么计算机中的总线也叫做 bus。

现在再看看图 2-7 中的计算机组织结构图。从图上可以看出计算机系统必须通过输入输出设备来和我们沟通。输入输出设备有很多，并且新的外设始终在持续不断地产生。比如，从开始的卡片机、打印机到后来的键盘、显示器，又到今天的摄像头、音箱、数码相机。

输入输出设备必须采用某种方式和现有的计算机系统连接到一起才能工作。如果想增加新的输入输出设备，最简单的办法就是将新设备与每个需要与之通信的部件用相应的线连接起来。比如，在摄像头和控制器之间连一根线、在摄像头和存储器之间再连一根线。如果再接一个绘图仪，就在绘图仪和 CPU 之间连一根线再和存储器之间连一根线。这样很快随着外设的增加，计算机系统中的连线就会变得一片混乱，如图 2-9 所示[1]。如果希望能够随时增添或者减少设备，那就更麻烦了。为此，聪明的设计者想出了图 2-10 所示的连接方式。所有的部件都连接到一根信息传输线上。这根线就称为总线。总线上连接有 I/O 接口，新增加的设备只要连接到 I/O 接口上就可以同其他部件通信了。总线在主板上，主板上的插槽就是各种 I/O 接口。现在如果希望把录像机连接到计算机上，有了总线，事情就变得简单多了。只要在主板的 PCI 插槽上插一块视频采集卡，然后，把录像机用视频线和音频线连接到采集卡上就行了。PCI 插槽是 PCI 总线的一个接口。PCI 总线是标准总线中的一种。制定标准是为了让各个厂商制造的接口卡以及主板能够通用。这样不同厂商生产的外设，只要连接方式符合你计算机的总线标准，那么你就可以使用和替换这些外设了。同样 CPU 和主存也可以挂在总线上同其他设备通信，其实它们也是通过相应的插槽连接到总线上。

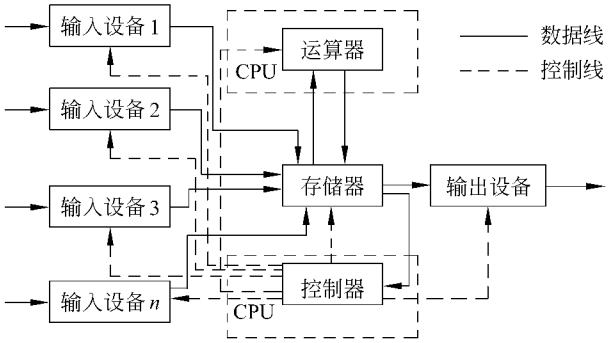


图 2-9 分散式连接外设

总线其实是由许多条传输线构成的，每条线传输一位二进制数码。这样这些线就可以

一起同时传输若干位二进制数码。传输线的条数，也就是一次可以同时传输的二进制数码的位数，称为总线的宽度。目前常见的地址总线宽度为 32 位，还有 64 位。

总线由所有部件共享，所以需要有一个仲裁方法和相应的电子器件来决定在某个时刻由谁来使用总线。我们这里就不介绍总线仲裁的方法和过程了，大家可以查阅计算机组成原理相关书籍去了解。图 2-10 只是总线连接的一种方式，叫做单总线方式。还有很多种其他不同总线连接方式。比如，在 CPU 和主存之间增加一条总线，就形成双总线结构。现在所有的系统都使用总线结构，但是在细节上会有不同。从上述内容看来，总线和公共汽车是不是还真有点儿像呢？

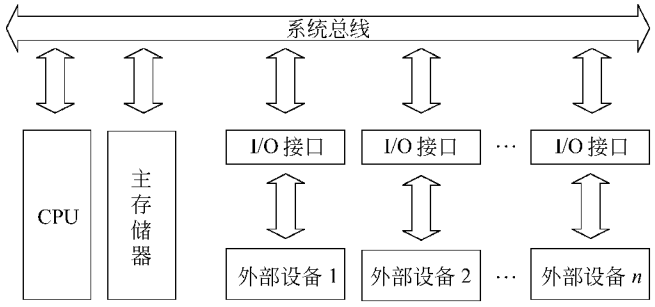


图 2-10　总线连接方式

总线是现代计算机技术中很基本的一个概念，贯穿在计算机系统平台各个层次中，出现在计算机中的很多地方。对于这里讲的纯硬件意义上的总线，按照其连接部件的不同，可以分为片内总线、部件内总线、系统总线和外总线几种[1]。

片内总线是指芯片内部的总线。例如，CPU 内部寄存器和寄存器之间、寄存器和 ALU 之间的总线。

部件内总线是指插件板内各芯片之间传送信息所使用的总线，如显示适配卡中使用的总线。

系统总线是指计算机系统内各功能部件，如 CPU、主存、I/O 等之间的信息传输线。我们前面所说的数据总线、地址总线以及控制总线就是系统总线。它位于主板上，因此也叫做板级总线。

外总线是指计算机系统之间以及计算机系统与其他系统之间的通信总线。

小提示

总线是计算机中不同部件之间的公用信号通道。
总线一次可以同时传输的二进制数码的位数，称为总线的宽度。
按照总线的连接部件的不同，可以把总线分为片内总线、部件内总线、系统总线和外总线几种。

虽然有了作为计算机不同部件之间公用的信号通道——总线，可是，总线应该有多宽，采用什么样的方式和总线相连，还需要做个规定。不然，一家做出的总线一个样儿，最终就会形成一种 I/O 接口配一种总线、各种总线之间互相不兼容的情形。这最后造成的结果跟没有总线也差不多。为此，人们开始统一总线的规格，也就是为总线制定标准。这样只要制

造外设的厂家按照总线接口标准去做，设备就可以连接到总线上，就能够互相通信。

总线技术在不断进步，因而总线的标准也在随时间变化。不长的计算机发展史中还是出现了很多总线标准。下面我们介绍几种常见的总线标准[25]：ISA 总线、PCI 总线、PCI-Express 总线和 USB 总线。这些名字听起来是不是有点儿熟悉？

1. ISA 总线

ISA 总线是工业标准结构(Industrial Standard Architecture)总线的简称。它是 PC 上出现较早的一种总线标准。在当时可是绝对的标准，没有人能与之抗衡。ISA 总线采用 16 位数据线、24 位地址线，最大传输速率为 16MB/s。后来出现的以 ISA 总线为基础的 EISA 总线的最大传输率可达 32MB/s。如果你现在还能找到 386 或者 486 的机器，那么在主板上还会看到用来插声卡、网卡的 ISA 槽。

2. PCI 总线

随着计算机的发展，尤其是图形界面的流行，ISA 总线和 EISA 总线的速率已经渐渐不能满足要求了。在这个时候出现了 PCI 总线标准，即外部组件互连总线(Peripheral Component Interconnect Bus)。PCI 总线开始时的总线宽度是 32 位，带宽是 133MB/s。这个速度显然比 ISA 快多了。在 PCI 发布一年之后，它的带宽又被增加到 266MB/s，总线宽度改为 64 位。PCI 总线的速度最终被提升到 528MB/s。在 PCI 出现的初期，PCI 总线和 ISA 总线并存在同一系统中。所以你在 486 机器上可以看到插在 ISA 槽上的声卡和插在 PCI 槽上的显卡。到后来，使用 ISA 槽的设备越来越少，主板上也就不见了 ISA 槽的踪影。

3. PCI-Express 总线

PCI-Express(简称 PCI-E)是 2001 年提出的一个总线标准。PCI-E 有多种不同速度的接口模式，包括 1X、2X、4X、8X、16X 以及 32X。PCI-E 1X 模式的单向传输速率为 250MB/s。而其他模式，如 8X、16X 的传输速率便是 1X 的 8 倍和 16 倍。这样算下来 16X 接口已能达到双向 8GB/s 的峰值带宽了。这样快的速度能够支持大屏幕、高分辨率的显示，是高端显卡要发挥作用所必需的。

4. USB 总线[26]

USB 总线可不是新技术，早在 20 世纪 90 年代中期就被提出来了。只不过近几年，尤其是 U 盘出现后的这段时间，我们才感觉到它的异彩光芒。USB 总线是通用串行总线(Universal Serial Bus)的简称。这种总线与前面几种最大的区别是，USB 接口都在机箱外露着，不用开机箱就可以直接连接设备。从这一点儿你也能感觉到它的不同，实际上 USB 总线和前面提到的几种总线不是一个级别的标准。前面提到的几种都属于系统总线。而 USB 总线在某种程度上可以看作是系统总线的一个连接低速外设的延长线。USB 总线一端连接在系统总线上(如前面提到的 PCI 总线)，另一端可以连接低速的设备。通过级联，它最多可以连接 127 个设备，不过很少见到有人用它连接这么多设备。USB 总线的另一个特点是支持热插拔。这确实非常方便，总不能为了取下 U 盘关一次机吧。现在的计算机启动可是需要一段时间。USB 1.0/1.1 标准传输速率可达 1.5Mb/s，最大可达 12Mb/s。USB 2.0 标准对于高速设备可支持高达 480Mb/s 的数据传输率，它向下兼容旧的 USB 1.0/1.1 软件和设备。

注意我们讲 USB 的速度时用的单位是 Mb/s，而前面几个总线的速度写的都是 MB/s。这里面大写的 B 和小写的 b 可不是一个含义。大写的 B 是字节 byte 的缩写，而小写的 b 是

位 bit 的缩写，差了 8 倍呢。这两种表示法都可以用。但是选设备时，你一定要看清楚，不要糊涂噢。

最后我们用一个 Pentium CPU 的三总线结构图，如图 2-11 所示，作为这一节的总结。从这个图上可以清楚地看出各种总线的关系和作用。

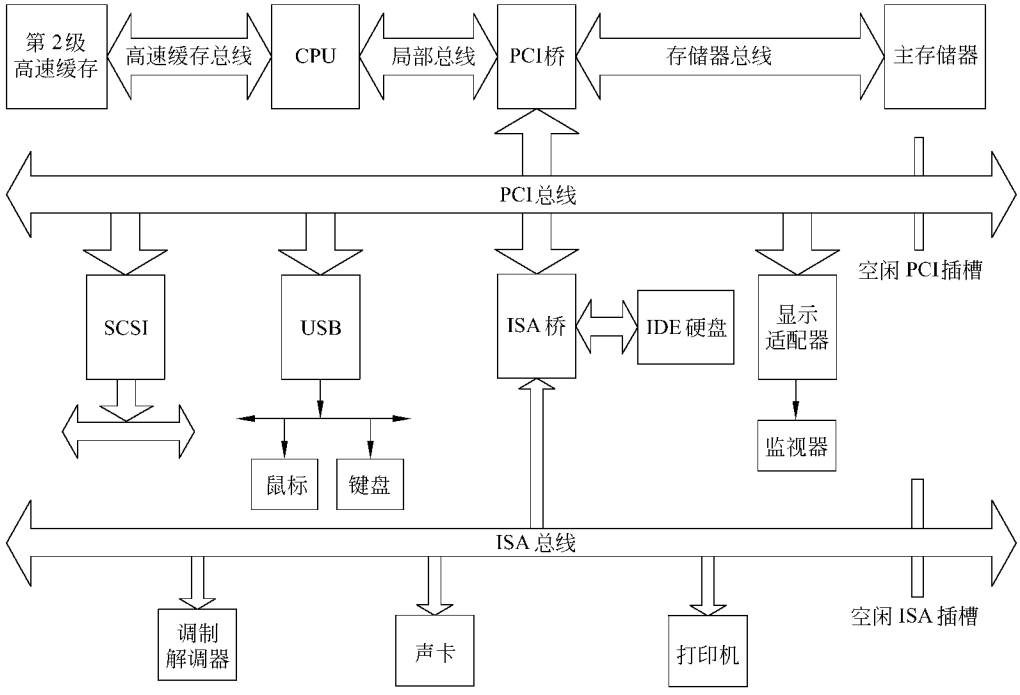


图 2-11　Pentium 的三总线结构

小提示

PCI 总线、PCI-Express 总线和 USB 总线是目前常见的标准总线。
xMB/s 表示每秒 x 兆字节，xMb/s 表示每秒 x 兆位，前者是后者的 8 倍。

2.3　存储器

存储器对我们来说都非常熟悉。一提起来，大家都会说出几种，如硬盘、软盘、光盘、内存条，还有现在使用越来越广泛的 U 盘等。为什么会有这么多种类型的存储器呢？它们可不只是名字不同，显然它们的形状、容量和存取速度都各不相同。为什么不选取一种最好的来用呢？这样我们也就用不着了解这么多种不同的设备，尤其是计算机系统的设计人员更不用设计这么多种不同种类存储器的访问方法了。直接导致这种混乱结果的原因在于：目前还没有一种存储器能够在所有性能指标上都强于其他的类型。

这些不同类型的存储器是用不同的元器件或者材料制成的。我们前面说过，存储器是由一些具有记忆功能的器件构成的。对计算机系统来说，所谓记忆功能就是能够保持两种

不同的状态。这样的元器件或材料有很多。例如，电容可以有带电和没电两种状态，因此它可以用来做存储器。用电容制造的存储产品就是内存条。又比如，磁性材料可以被磁化成不同极性方向，所以可以用两个相反的极性来表示 0 和 1 两种数码。因而磁性材料也可以做成存储器。这种材料做成的产品就是硬盘和软盘。其实可以表示两种状态的材料非常多。比如在一张纸上打一个孔，就可以和没打孔的地方区分开，因而也可以表示两种状态。早期的程序就是在卡片上打孔，用读卡器输入到计算机中的。从这个角度来说，纸张也是一种存储器。只不过它的存储容量太小，成本太高，而且不能重复使用。现在你是不是受到启发，又想到什么也能表示两种状态的材料了？对了，在塑料板上凿个凹坑，也可以和没有凹坑的地方相区分，同样可以表示两种状态。光盘采用的就是这种机制。光盘的介质薄膜上压一些凹坑，表示一个数码，没有凹坑的地方表示另一个数码。和纸张一样，这种方法制成的存储器也不能多次使用。这么看起来具有记忆功能的材料真是不少。

但是，这些材料除了都可以保持两种状态外，其他属性的差别非常大。例如，若用电容来记忆信息，电容里的电量只能保持 1～2ms，所以必须在这段时间内对带电单元重新上电，不然，信息就消失了。如果这种存储器没有了电源，那么电容中的信息很快也就不见了。因此，这种存储器也被称为易失性存储器。它不能用来存储要长期保存的信息。但是这种存储器有个优点，就是它的访问速度很快，所以用它来做内存条。而用磁化磁性材料的原理制成的存储器上被磁化后的极性，如果没有被再次磁化，就不变了。它不需要用电，或者其他的任何帮助，信息就可以长期保持下来了。如果需要重写，用磁头对其进行再次磁化就可以了。可是，对这种存储器的访问比起电容来说就慢多了。而光盘，就是带凹坑的塑料片，它的造价非常便宜，缺点是不能重写（现在有可重写光盘，但重写的效率不高，次数也不能太多）。从上面的分析能够看出，不同材料制成的存储器各有长处和缺点。计算机系统的设计者就利用这些特性，根据需要把它们用在不同的场合。

对于存储器，除了要求它必须能够存储信息外，我们还有哪些性能上的要求呢？

当然，首先考虑的就是存储器的容量，也就是它能够存储多少信息。容量是以字节（B）为单位来记的。容量当然是越大越好。现在常见内存条的存储容量在 512MB（兆字节）左右，1GB 的也能够见到；硬盘的大小则多在百 GB（吉字节）左右；光盘则在 600MB 左右；而 DVD 光盘可达 17GB（1M＝1024×1024，1G＝1024×1024×1024）。

第二个要考虑的指标就是造价，通常用元/位来表示。显然，我们希望这个数值越小越好。存储器的价格可以在一定程度上反映出这个值。目前几百吉字节硬盘的价格一般不足千元，而上吉字节的内存条大概以百元为单位。一张 600MB 可写光盘的价格现在也就几元钱。容量几吉的 U 盘价格一般不足百元。可以看出不同种类存储器在价格上有很大不同。

除了造价，访问速度是存储器另外一个重要的指标。不然，怎么会有人愿意花大价钱去买贵的存储器呢。用电容或者半导体器件制成的存储器的访问速度最快。而且访问这种存储器的任意单元时间都是相同的。地址放到地址总线上，就可以对相应单元的数据进行读写了。而访问磁盘这类存储器要经过磁盘驱动器，经过磁电转换。而且要访问磁盘上某个位置的数据，磁头必须移动到该位置才能读取。而磁头移动的机械速度与比起地址信号通过地址总线到达内存的速度相差太大了，就好像自行车和飞机去比速度。存储器的存取速度通常用带宽来表达，单位是“位/秒”或者“字节/秒”。也就是单位时间内能够从存储器读

写的数据量。提高存储带宽的方法有两种：一是使用存储速度更快的元器件；另一种办法就是增加每次存取的数据量，即每次同时存取更多的字节。增加每次存取的数据量需要增加数据总线的宽度，在读写电路上要有相应的硬件支持。例如，对于磁盘来讲，这需要增加多个磁头，让它们同时工作。硬盘就是这样做的。这好比给一个大的礼堂多修几个出口和相应的通道，这样人们进出的速度就快了。

小提示

存储器种类很多。不同类型的存储器由不同的材料依据不同的原理制成。各种类型的存储器各有优势。

存储器的指标包括容量、速度、价格。

我们已经知道计算机的工作过程：CPU 从主存储器中读取指令，执行，把结果写回主存储器，然后读下一条指令。这是一个不断循环反复的过程。CPU 每取出并执行一条指令所需的全部时间被称为一个指令周期。如果指令周期短，那么单位时间内执行的指令条数就多了，计算机的速度也就快了。也就是说计算机的速度与指令周期的长短是相关的。我们仔细分析一下指令周期，发现它可以分解成以下几部分：CPU 解释和执行指令的时间，从主存储器取指令、取操作数，向主存储器写操作数的时间。从这里可以看出与计算机速度有关的不仅仅是 CPU 解释和执行指令的速度，主存储器的访问速度也是一个重要的影响因素。目前 CPU 的速度正在快速发展，而存储器的速度则远远落后。虽然主存储器比硬盘要快得多，但是和 CPU 的执行速度相比，它还差了很多数量级。存储器的速度已经成为计算机系统的一个瓶颈。

要想使存储器的速度能和 CPU 速度匹配，那么就需要使用最快的存储材料。但是，我们已经知道速度快的存储设备造价又非常高，这样整个计算机的价格又难于接受。怎样解决这个问题呢？计算机科学研究人员发现在计算机程序运行过程中有一个相当普遍的规律，即所谓“局部性原理”，大致意思是在一个时间段内执行的程序代码和用到的数据基本集中在一起(空间上)。基于此，系统设计者就想出了一个综合利用不同存储设备的好办法。在讲这个办法之前，我们先来分析一下所处的环境。因为只有认真分析问题，才能找到切实解决问题的办法。目前的情况很简单：CPU 从存储器上读取程序来执行。CPU 要一条一条来执行指令。好像是废话啊？别急！最显而易见的现象中往往蕴涵着解决问题的关键。CPU 要一条一条来执行指令意味着什么呢？意味着在某个时刻它只需要一条代码，而不是整个程序段。如果你的计算机能够同时运行几个程序，情况可能复杂一些。不过因为你的计算机也还是只有一个 CPU，所以即使在这种情况下在某个时刻 CPU 也只能执行一条指令。只不过此时，存储器中需要同时准备几个不同的程序。尽管如此，在某个时间段内，计算机上安装的所有程序也并不是都要执行。假设现在你想编个程序。假设一边听歌一边写代码，你的效率可能比较高。所以你要运行一个 Visual Basic Studio(一个可以让你写程序的软件)，再运行一个 MP3 播放器。尽管你的存储器中还有 Foxmail(一个收发电子邮件的软件)，但是你并不想运行它，因为你现在不想让邮件打扰你。所以对环境的分析工作得出的结论是，通常 CPU 不会同时需要非常多的信息。这个结论非常有价值，给了系统设计员一个好机会。他们于是给计算机加了一个很快的存储器，以便能接近 CPU 的飞快速度。

但是因为这个高速的存储器很贵，所以它的容量被设计得很小，以便让计算机的价格能够被你接受。但是，你拥有名目繁多的软件、整理不完的文档，以及看不完的电影。这个小小的存储器是不能满足你的要求的。计算机的设计者就在计算机里面加了一个大大的、造价便宜的存储器——硬盘，来存放你所有的软件、文档、电影和歌曲。我们已经知道磁盘的速度是远远跟不上 CPU 速度的。设计者的办法是把你正在运行的软件从磁盘复制到那个小小的、高速的存储器里，然后 CPU 只和这个高速的存储器打交道。我们把这个高速存储器叫做内部存储器，或者内存、主存。它们通常直接插在主板上。相对内存，硬盘被称为外存、辅助存储器或辅存。这样我们用两种不同类型、不同性价比的存储器就把问题解决了：又省钱，又能满足速度和容量上的要求。设计者有时候真的很聪明！因为内存里面的信息是从外存复制来的，当程序执行结束后，需要存储的信息又会被复制到外存，所以内存可以用易失性材料制成。

内存用来存放正在执行的程序段和相关的数据，所以容量也不能太小。太小就要不断地从辅存复制数据，进而影响整个计算机系统的速度。因而内存也用不起最快、最贵的存储材料。正如我们前面说的，内存的速度和 CPU 还是差很远呢。为了让存储器的速度更接近 CPU，设计者把这个好思想进一步发挥。他们用更快、更小、更贵的存储器来存放内存中最近最可能被 CPU 访问的内容。这种存储器被称为高速缓冲存储器，简称缓存。缓存不但速度很快，而且位置也很不错，一般都直接放在 CPU 芯片的内部。买 CPU 的时候，你一定听卖主推销时说：这个 CPU 带一级缓存、二级缓存等。

其实在日常生活中，也利用过这种办法。例如，你在图书馆查资料，准备一篇关于计算机的演讲。你到书架上找到一本书，拿回到书桌前查阅有关 CPU 的内容。在查阅完这部分资料，开始查阅另外一本书时，你并没有把这本书放回书架，而是暂时放在书桌上。因为你想一会儿可能还会用到这本书。这样可以避免再去书架上去取，那就要多花一些时间。当然书桌大小是有限的，只能放一些书，而不能把书架上的书全拿过来。缓存就好像书桌，而内存就好像书架。如果书桌放满了，那你就必须选择一些书把它们放回书架，并取一些新的书。如果放回去的书很快又被用到，那么你还需要到书架上再把它取回来。要提高效率，就要尽量避免这种情况。同样道理，缓存的设计也要考虑类似的问题。

计算机系统就是用这个层次的存储系统结构达到了多快好省的目的。图 2-12 是计算机存储系统的层次结构图。在这个层次结构中，从下到上存储器的速度越来越快，价格越来越高，容量也因此越来越小。最顶层就是前面提到过的寄存器。它的访问速度最快，容量也很小。通常每个寄存器只存储几个字节。在它的下面是缓冲存储器，就是缓存，它的速度介于寄存器与内存之间。接下来是主存。最底层是辅存。它的价格最便宜，容量也最大，用来存储需要长期存储的大量信息。

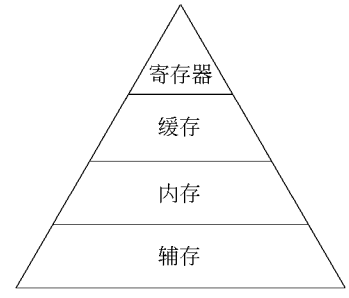


图 2-12　计算机存储系统层次结构

本节主要介绍层次图中上几层存储器的相关知识。这些存储器基本上都在机箱内部的主板上。而位于最底层的辅存，因为要通过驱动器这个输出设备来访问，所以我们把它放到输入输出系统小节中介绍。

内存的地址译码器直接和地址总线相连，地址译码器会把地址翻译成相应单元的选择信号，该信号和读写电路配合就可以访问指定的存储单元。这样的机制使内存的访问速度较快，而且访问每个单元的时间都是一样的。这和磁盘有很大不同，访问磁盘时离磁头近的区域需要的时间就短。因此，内存也被称为随机访问存储器，就是常说的 RAM（Random Access Memory）。图 2-6 是现在常见的一些内存条。RAM 中的内容在计算机掉电之后就不存在了。内存条通过专用的插槽安装在主板上，然后通过总线与其他部件通信。

内存的类型随着制造技术的不断改进，也在不断地变化。早期的内存类型主要有 FPM、EDO、SDRAM、RDRAM 等，目前常见的则是 DDR SDRAM、DDR2 两种内存[28]。我们给出的顺序也是这些内存出现的时间顺序。和这些不同类型内存对应的主板插槽也在变化之中。最早的时候内存通过特定的板卡插在总线插槽中。因为存在一些问题，后来出现了专用的内存插槽。常见的内存插槽有 SIMM、DIMM 和 RIMM 三种类型[27]。这些缩写是不是看得你有点儿晕？下面就介绍一下目前常见的内存类型。如果遇到了其他类型可以去互联网上查查，或许会找到详细的信息。不过在互联网上查资料的时候，建议你多找几份对照来看，因为保不准哪位作者不小心会有笔误。

SDRAM 是同步动态随机存储器（Synchronous Dynamic Random Access Memory）的简称，是前几年曾普遍使用的内存类型，在市场上流行很长时间。如果你对它的内部结构感兴趣，可以去考究一下它名字里面的“同步”和“动态”是什么意思。这里我们就不介绍了。

DDR SDRAM 是双数据率同步动态随机存储器（Double Data Rate Synchronous Dynamic Random Access Memory）的简称，是目前最常用的内存类型。DDR SDRAM 是 SDRAM 的更新换代产品，具有比 SDRAM 多一倍的传输速率和内存带宽，这就是它名字前面 DDR 的含义了。

DDR2（Double Data Rate 2）从名字上来看是下一代 DDR 的意思，是新开发的内存技术标准，具有更快的速度。

SIMM 是单列直插内存模块（Single Inline Memory Module）的简称，是早期常用的一种插槽。这种插槽上可以插以前的 EDO 内存。随着 EDO 内存的退出，这种插槽在主板上也越来越少了，在比较旧的机器上你可能还会看到它。向这种插槽安装内存条时，只要把内存条斜向 45 度插入，然后再扶正就可以了。插槽旁边的两个水平方向的卡子就会把内存条卡住。取下时，只要把两端的卡子掰开，内存条就会被松绑倒下，就可以取出了。

DIMM 是双列直插内存模块（Dual Inline Memory Module）的简称。SDRAM、DDR 和 DDR2 内存都采用 DIMM 插槽。但是这几种内存使用的 DIMM 插槽的针脚结构不同。最主要是内存条上的卡口不同。为了避免将内存错误地反向插入插槽而导致烧毁，内存条有针脚的那个边上都会有卡口，对应于插槽上的凸起，如图 2-13 所示的两个内存条的卡口就不一样。上面的一条有两个卡口，而下面的只有一个卡口。SDRAM 有两个卡口。而 DDR 有一个卡口。DDR2 也有一个卡口，但是和 DDR 的位置不同。这样就可以保证不会把内存条插到错误的插槽上了。

往 DIMM 插槽上安装内存很简单。首先确定内存的方向，就是看看内存条上的卡口和插槽上的凸起。对好之后，用力把内存条插到底就可以了。插槽旁边的竖直方向的卡子就会把内存条卡住。取下时，只要用力压两端的卡子，内存条就会被推出来，如图 2-14 所示。

RIMM 是和 RDRAM 内存对应的插槽，这种内存比较少见，所以这里就不介绍了。

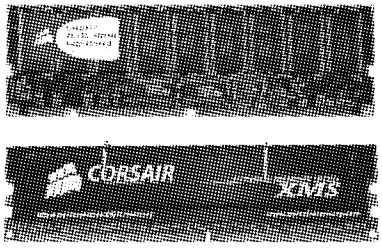


图 2-13　内存条上的卡口

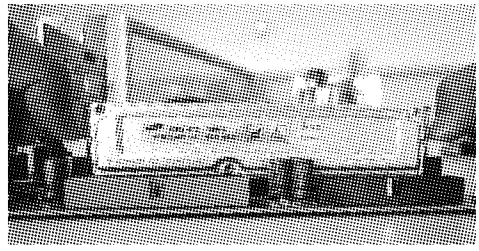


图 2-14　DIMM 内存条的取下

小提示

CPU 每取出并执行一条指令所需的全部时间叫做一个指令周期。
存储器的速度已经成为计算机系统的一个瓶颈。
计算机的存储系统采用层次结构来解决成本与速度的矛盾。
存储器层次结构的最顶端是寄存器，向下依次是缓存、内存和辅存。
现在常见的内存类型是 DDR SDRAM，使用 DIMM 插槽。

除了 RAM 之外，在计算机内部还有另外一种类型的存储器。这类存储器上的内容不会因没有电源而消失。但是，这些内容也不能更改，计算机只能读取。所以这类存储器叫做只读存储器(Read Only Memory，ROM)。

计算机主板上有一个很重要的部件，叫 BIOS ROM 芯片。从名字上看就可以知道它的身份了。BIOS 是基本输入输出系统(Basic Input Output System)的缩写。BIOS ROM 芯片中存储的内容在计算机出厂时就通过某种技术记录在芯片中，在计算机的使用过程中不能再更改。芯片中的内容包括计算机系统的基本输入输出程序、系统信息设置程序、开机上电自检程序和系统启动自举程序等。这是一些非常重要的程序，没有这些程序计算机就启动不起来了。

在计算机开机时，CPU 首先从 BIOS ROM 中读取程序，并开始执行，一步一步将计算机启动起来。其中系统信息设置程序被用来对计算机系统进行基本信息的配置。这些信息包括系统基本情况、系统启动顺序、CPU 特性、软硬盘驱动器、显示器、键盘等部件的信息。用户在 BIOS 程序刚开始运行时按下某个特定键即可使它暂停，来手动更改这些配置信息。这些基本配置信息存储在主板上一块可读写的 CMOS RAM 芯片中。设置 CMOS 中这些参数的过程，习惯上也称为“BIOS 设置”。CMOS 是一种半导体存储器。它由一块单独的电池供电，所以关机以后 CMOS 中的内容也不会消失。到现在为止，我们已经见到了足够多种类的存储器了。

除了系统信息设置程序外，BIOS 中的开机上电自检程序和系统启动自举程序也是在系统启动时执行的。在你按下开机按钮后，计算机首先读取开机上电自检 POST(Power On Self Test)程序对系统内各个设备进行检查，比如，对 CPU、内存、ROM、主板、CMOS 存储器、串并口、显示卡、软硬盘子系统及键盘等进行测试。如果在自检中发现问题，系统将给出提示信息或鸣笛警告。假设你没有给计算机插上内存条，那么开机后，你就会听到喇叭发出特别的声音。这就是由自检程序产生的。不同的问题会导致计算机发出不同的声音。当

然，这些声音很难辨别，只有熟悉的人才能从声音上判断出问题出在哪里。

现在多数的 BIOS ROM 芯片已经不是绝对的只读存储器了，而是可以用特殊方式进行更改的存储器。现在常听说的升级 BIOS 就是用特殊的方式对 BIOS 芯片进行更改。可更改的 BIOS 芯片开始时用的是可擦写可编程只读存储器，简写为 EPROM。而现在则常用一种称为闪存(flash memory)的存储器。这种存储器在不加电的情况下，信息能长期保存，又可在线擦写。其实我们常见的 U 盘也是这种存储器。

现在你知道计算机执行的第一个程序是怎么进到计算机中的吧？它并不是在控制器的控制下进入内存的，而是在计算机出生那天就有了，而且还轻易不能更改。

在以后的章节中我们还会详细介绍 BIOS 的内容以及 CMOS 的设置。

小提示

BIOS ROM 芯片里存储着计算机系统启动时执行的一些重要程序，包括计算机系统的基本输入输出程序、系统信息设置程序、开机上电自检程序和系统启动自举程序。

CMOS 存储器中存放着计算机的基本设置。

为什么存储系统要采用层次结构？

2.4　中央处理器 CPU

CPU 是计算机的核心。前面的介绍使我们对 CPU 已经有一些印象了。现在让我们再详细了解了解。在计算机市场中，我们经常听到一些名词，中关村那些热心的摊主会告诉你他们的 CPU 是 32 位的，支持超标量、流水线，主频是 4G 等。这些专业名词经常会把你弄得一头雾水。那么它们和 CPU 到底有什么关系呢？下面就来逐个探究一下。

图 2-15 和图 2-16 分别是 Intel 公司的 Pentium 4 CPU 的正面和背面。图 2-17 是 Intel 的 P6 CPU 的背面。除了 Intel 之外还有很多著名的电子公司都在生产 CPU。目前我国也正在开展这方面的研发，在不久的将来也能够在这个领域占有一席之地。

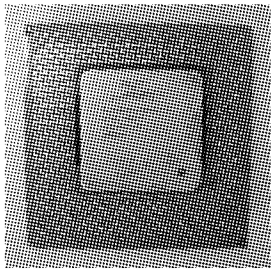


图 2-15　Pentium 4 CPU 正面

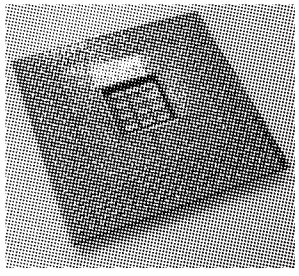


图 2-16　Pentium 4 CPU 背面

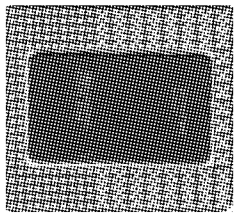


图 2-17　P6 CPU 的背面

从前面几幅图能够看出，CPU 的形状、大小不尽相同。即使是同一个厂家的不同型号的产品也可能不同。从图 2-16 可以看到 P4 背面上有很多从这个小小的方块芯片中引出的针脚。CPU 就是通过这些针脚插到主板上的 CPU 专用插槽，实现同其他部件的通信。仔细看图 2-16 和图 2-17，你会发现一个小的细节，无论是正方形还是矩形的芯片，4 个角中都

有一个角与众不同。比如图 2-16 左上角与其他角就不同，而图 2-17 中 P6 的右下角则干脆少了一块。在插槽上，对应 CPU 这个不同的角也会有一个不同的角，这样安装 CPU 的时候就不用担心插错方向了。

不同型号 CPU 的针脚数可能不同。例如，图 2-16 中的和图 2-17 中的两个 CPU 的针脚数就不同。从这也可以推断出不同 CPU 的内部结构也不相同。你可能想到了，一块主板只能支持特定型号的 CPU。因为主板出厂后上面的 CPU 插槽也就固定了，所以，会看到主板的技术资料中就有一项 CPU 插槽类型，比如是 Socket370 或者 Socket478（这些都是 CPU 插槽的名称）等，然后才是支持的 CPU 类型。有些 CPU 会使用相同的插槽。

别小看 CPU 这个小小的芯片。它里面封装了上亿个元器件，这叫做超大规模集成电路。可想而知，生产这样的器件是很复杂、难度很大的事情。它需要设计以及生产工艺上的全面进步，不是头脑一热就能做成的事情，需要很多人长时间的不懈努力才能实现。

小提示

不同型号 CPU 的形状、大小可能不同，针脚数及其使用的插槽也可能不同。
一个 CPU 芯片中会集成上亿个器件，并且这个数字还在不断增长。

CPU 的速度是计算机效率的一个关键。通常用 CPU 每秒钟执行的指令条数来描述 CPU 的速度，单位记做 MIPS（Million Instruction Per Second），即每秒执行百万条指令。例如，Pentium 4 1.5G 的评测速度可达 2800MIPS 左右。

那么 CPU 的速度到底是由什么决定的呢？这要从指令的执行开始说起。前面说过，CPU 执行完一条指令后再去取第二条指令执行。取指令和执行指令的过程中控制单元要发出很多次控制信号。比如，在取指令过程中，控制单元要将指令的地址发送给主存储器（内存），再向主存储器发送一个读信号，然后再从数据总线上把指令读过来并放到指令寄存器中，最后将程序计数器的值加 1，使其表示下一条指令的地址。这一连串的动作，是有先后顺序的，而且完成每个动作都需要一定的时间。控制信号是从控制器到工作部件单方向的，每个部件做完事情是不向控制器报告的。那么控制器怎样决定什么时候发送下一个命令信号呢？如果命令信号发得太快，前一动作还没有做完，那就会出乱子。可是如果发得太慢，各个部件干完活只能闲等，浪费时间。而实际上，控制单元是在一个时钟脉冲的控制下发布命令信号的。这个脉冲就好像唱歌时的节拍，到一个节拍时，控制单元就发出一个命令信号，就像唱歌到一个节拍唱下一个音一样。设计者精妙地设计节拍的时间以及每个动作完成的时间，让每个命令信号产生的动作刚好在一个节拍时间内完成。

控制 CPU 的这个时钟脉冲来自于主板。主板上有一个器件，一般是石英晶体，能够定期地发出脉冲信号，被称为时钟脉冲。我们已经知道 CPU 是通过芯片上的引脚和主板相连的。这些引脚中有一个就叫做时钟引脚。它负责将主板上的时钟脉冲传递给 CPU。在一秒钟内产生的时钟脉冲数就叫做频率。主板上产生的时钟脉冲叫做外频，是指挥主板上其他部件工作的节拍。因为 CPU 完成操作的速度越来越快，所以时钟脉冲传递给 CPU 的过程中被进行了特殊处理，频率被加倍，这样 CPU 的工作速度就提高了。这个频率就是常说的 CPU 主频。从某种角度上来说，频率越高，CPU 的速度越快，因为它单位时间内发出的命令信号多，完成的动作多，执行的指令也就多了。不过，这只说对了一半。对同一系列

的 CPU 是这样的，比如 Pentium 4 系列，4GHz 主频的 CPU 一定比 1.5GHz 的快。但是，不同系列的 CPU 因为结构不一样，所以即使频率相同，完成的指令数却可能差很多。

小提示

MIPS 是描述 CPU 速度的一个单位，意思是每秒百万条指令。
控制器在时钟节拍的控制下发布控制信号。
CPU 的主频是指每秒钟内产生的控制 CPU 动作的时钟节拍数。
外频是指主板产生的、控制主板上部件工作的、单位时间内的节拍数。

从前面的叙述我们知道提高主频可以提高计算机的速度。除此之外，还有没有别的办法可以提高 CPU 的效率呢？要想回答这个问题，我们还要从 CPU 所做的事情入手。首先来分析一下指令的执行过程。我们已经知道一个指令周期内 CPU 做的事情可分为取指令和执行指令两部分。CPU 取一条指令，然后执行，再去取下一条指令、再执行，如此反复。我们把这个过程叫做指令的串行执行，如图 2-18 所示。

取指令 1	执行指令 1	取指令 2	执行指令 2	取指令…	执行指令…

图 2-18　指令的串行执行

进一步分析这个过程，我们可以发现取指令主要由主存和取指令部件参与，而执行阶段则是运算器在工作。取指令时，运算器就闲着没事做，只是等待取指令动作的完成。而执行指令时，运算器在工作，取指令部件又在空闲。也就是可以这样说，部件的一半时间被浪费了。如果有一种办法能让这些部件同时工作没有空闲，那么整个系统的效率也就提高了。计算机的设计者把这个想法付诸于实践。他们让 CPU 在执行一条指令的同时，把下一条指令也取出来，如图 2-19 所示。从图 2-19 上可以看出，原来 CPU 只能执行两条指令的时间现在可以执行三条指令了。这样继续把指令重叠下去，可以看到，CPU 单位时间内执行的指令条数实际上被加倍了，这被称为指令的流水执行。这样在 CPU 主频没有增加的情况下，执行的指令条数却增加了。这就是前面所说的不同结构 CPU 是不能只用主频来比较速度的原因。

图 2-19　指令的流水执行

事情还可以做得更细一些。因为执行指令时，CPU 需要从内存读操作数或者向内存写结果，这时运算器和取指令的部件都必须等待。所以，如果把执行指令的动作再细分成更小的步骤，让每个部件都没有闲置时间，CPU 的效率还可以更进一步提高。例如，把 CPU 的全部动作分解成取指令、指令译码、计算操作数地址、取操作数、执行指令和写操作数 6 个小步骤。然后把这些动作都重叠起来，那么单位时间内完成的指令条数就更多了。不过你可能已经发现了一个问题。取指令、取操作数以及写操作数，都是访问内存的操作，实际上是不能一起做的。如果一定要一起做，就必须把这三个动作所操作的内存分开，比如把指令和操作数放在不同的存储器中。这样取指令和取操作数就可以重叠着一起做了。可是这样写操作数和读操作数还是不能同时做。实在没办法，那就不一起做吧。这样流水的重叠数可能就减小了。还有一个问题：如果下一条指令的地址要等上一条指令执行完了才能知道，例

如，程序要根据从鼠标得来的消息是按下“开始”还是“退出”按钮，来决定接下来是执行“游戏开始”的那段代码还是“游戏结束”的那段代码，这样流水也必须被打断。所以说，实现流水也不是那么简单地把动作分细就行了。流水的效率也不是动作的粒度有多细就有多高。里面有很多工作要做，有很多细节需要考虑。

超标量其实也很好理解。就是将 CPU 中执行指令必需的一些功能部件，如指令部件、运算器的数量加倍，让 CPU 能够同时取多条指令并且执行多条指令。不过，这说起来简单，做起来也不是很容易。就像前面提到的，有时候指令之间是有前因后果的，不是都能一起同时执行的。那怎么办呢？这就需要另外一个程序把要执行的程序中能够同时执行的指令分出来放在一起。这个事情非常复杂。现在这部分工作一般是由一个叫做编译器的程序来完成的。由此可见，CPU 速度的提高不单单是硬件的事情，还必须有相应软件的配合才行。

小提示

指令流水是指将指令执行过程分割成小的动作，让这些动作在不同的部件上执行，从而让多条指令能够在不同的部件上同时执行。

CPU 的速度与主频有关，但是也受 CPU 结构的影响，比如结构是否采用流水、超标量等技术。

计算机速度的提高需要软件和硬件的配合。

现在来解释一下 32 位或者 64 位的 CPU 指的是什么？这个位数也叫机器字长，指 CPU 一次能够处理的二进制数据的位数。这个数值与运算器中寄存器的位数以及 ALU（还记得吧？算术逻辑单元）有关。一般来说，位数越高，CPU 的效率越高。这个道理很明显。如果你只会算个位数的加法，那么如果我想让你算一个两位数的加法，我就得先让你把个位相加，并把得出的结果以及进位记下来；然后再让你算十位上两个数码的加法，再让你加上进位，得出结果记下来；最后把两次运算的结果拼在一起。哎，累坏我了！还不如我自己算了。所以如果你能算两位数的加法，我直接告诉你要相加的两个数，你就告诉我答案了。多简单，又快又省事儿！同样 CPU 的字长也与此类似。一次可以参加运算的数位越多，那么运算的速度显然也就越快。

数据总线的宽度也会影响计算机的性能。如果计算机能一次处理 32 位数据，即机器字长是 32 位，而数据总线宽度为 8 位，也就是说，一次只能从内存取出 8 位数据，那么 CPU 的一次运算要访问 4 次存储器才能取到操作数。要是这样，CPU 算得再快有什么用啊！

其实地址总线的宽度也和机器字长有关系。因为程序中有时候需要对地址进行运算。比如，程序要求跳转到距离当前指令位置 200 个字节的地方去执行那里的指令。这时 CPU 就需要将当前地址加上 200。如果你写过程序的话，会知道程序中有一类特殊类型的数据，叫做指针，其实就是地址。指针的运算也就是地址的运算。那么这个地址的加法在哪里做呢？当然是在运算器中了。所以，当然是地址有多长，运算器就能处理多长的数据最好，这样免得把地址拆开一部分一部分运算。也就是说地址总线的宽度和机器字长相等最好。

早期计算机有的地址总线宽度就长于机器字长。例如，Intel 80286 的地址总线是 20 位的，而机器字长是 16 位。所以，在 286 的 CPU 中，为了计算 20 位的地址，特意加了一些专门的硬件。那为什么不把它们设计成一样的呢？比如，把地址总线也设置为 16 位不就行了

吗？这样事情就简单多了。为什么一定要 20 位的地址总线？有什么考虑吗？现在我们就来看看地址总线的宽度到底影响着什么。地址总线上传输的是地址，地址总线的宽度也就是二进制地址的位数。一个地址就是表示一个存储单元的门牌号。我们先来看看门牌号的事情。你家的门牌号是多少？是 119 吗？好！那么同样是三位的门牌号码可以有多少呢？1000 个(即 000～999，假设 000 也可以用作门牌号)。所以，如果规定门牌号的位数是三位，那么一共可以表示 1000 个不同的房子。如果有 1001 个房子，用 1000 号表示就可以了。注意这里有个小小的不同，现在的门牌号已经变成 4 位了。所以，可以看出，门牌号的位数决定可以表示房子的最大数目。因此，地址总线的宽度也决定了可以表示的内存单元的数目。20 位地址总线可以表示的内存单元数目为 2^{20}。现在你能明白 80286 为什么费很多力气一定要使用 20 位地址总线了吧？16 位地址宽度意味着可以表示 2^{16}，也就是 64×1024 个内存地址。在这种情况下，即使你给计算机插上一个 1MB 的内存也没有用，因为 CPU 没办法说出 64KB 之外的存储单元的地址。

小提示

机器字长，指 CPU 一次能够处理的二进制数据的位数。它和地址总线宽度以及数据总线宽度的关系会影响计算机的效率。

地址总线的宽度决定可以访问到的内存单元的数目。

寄存器和高速缓存(cache)是两种比较快的存储器。现在的 CPU 中都集成了这两类存储器[3]。当然别的设备中也有这两种性质的存储器。CPU 中的寄存器，我们已经知道的有程序计数器、指令寄存器。还有一些其他称为控制和状态寄存器的部件，包括状态标志寄存器、存储器访问寄存器等。状态标志寄存器主要存放一些状态标志和控制标志，比如，运算结果是否为 0、是否有进位、指针移动的方向等。而存储器访问寄存器则是 CPU 和内存进行数据交换时使用的，有存储器地址寄存器和存储器数据寄存器两种。与控制和状态寄存器相对，另外一些寄存器是可以由程序员控制的，称为用户可见寄存器。它们可以用来存放运算的原始数据和中间结果，也可以用来放指令或者数据的地址。程序员可以通过程序来指定它们的内容。CPU 中寄存器的数量非常多。不同的 CPU 结构，设计者给出的寄存器的种类和数量也可能不同。

缓存被 CPU 用来暂存从内存取出的数据，以弥合 CPU 和内存速度上的巨大差距。早期的缓存并不在 CPU 内部，近几年才把它集成到 CPU 中。这样可以让 CPU 和缓存的通信更快。当然，缓存的容量越大越好，这样可以提高缓存的命中率。什么是命中率？如果在缓存中找到需要的内容，就称为命中。如果缓存中没有需要的内容，那么就需要到内存把相应内容读到缓存中，这样就增加了访问时间。命中率就是指 CPU 要访问的信息已经在缓存中的比率。当然命中率越高越好。如果命中率为 0，缓存也就没用了，反而增加了信息传递的一个过程，画蛇添足了。为了提高缓存的命中率，在设计上要考虑缓存的容量以及将哪些内容存放在缓存中比较合适。

CPU 是整个计算机的核心，它的工作量可想而知。而且通过增加主频、流水、超标量等各种技术，CPU 还在变得越来越忙。因此，CPU 的功率也越来越高，发热量也越来越大。你可不能小瞧了这个问题。如果 CPU 的温度过高，你的计算机就有可能罢工。所以正如

你已经看到的，现在的 CPU 上都有一个风扇，风扇下面还有一个散热片。这两样东西合起来被叫做 CPU 散热器。如果你的散热器不工作了，那你可不要勉强 CPU 工作，否则会得不偿失。除了用散热器外，为了让 CPU 不至于过热，机箱最好放在通风的地方。

小提示

CPU 中的寄存器可以分为用户不可见的控制和状态寄存器以及用户可见寄存器两大类。
用户可见寄存器，程序员可以通过程序访问。
控制和状态寄存器则只能由系统使用。
CPU 中集成了缓存，以弥合和内存的巨大速度差距。
使用计算机时，要防止 CPU 的温度过高。

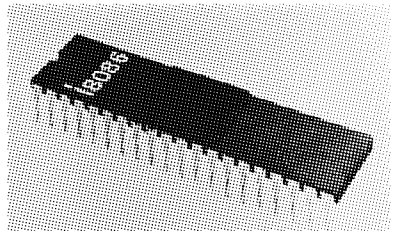
附文：

曾经的辉煌代表产品

摘自：不可磨灭的记忆　CPU 发展经典回顾放送
作者：中关村在线 ZOL 评测中心 2005 年 11 月 23 日 05:10

1978 年 6 月，Intel 推出 4.77MHz 的 8086 微处理器（见附图 1），标志着第三代微处理器问世。它采用 16 位寄存器、16 位数据总线和 29000 个 3 微米技术的晶体管，售价 360 美元。不过当时由于 360 美元过于昂贵，大部分人都没有足够的钱购买使用此芯片的计算机，于是 Intel 在 1 年之后，推出 4.77MHz 的 8 位微处理器 8088。IBM 公司 1981 年生产的第一台计算机就是使用的这种芯片。这也标志着 x86 架构和 IBM PC 兼容计算机的产生。

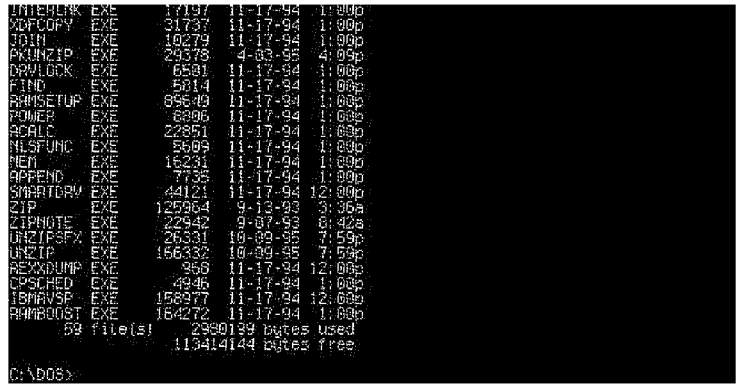
发布的时候，8086 的时钟频率有 4.77MHz、8MHz 和 10MHz 三个版本，包括了具有 300 个操作的指令集。其中 8MHz 版本包含了大约 28000 个晶体管，具备 0.8 MIPs 的能力。



附图 1　Intel 8086

1980 年 10 月，Microsoft 把握了一次绝佳的发展机遇。IBM 在秘密进行代号为“跳棋计划”的开发项目（第一台 IBM PC）过程中，向 Microsoft 提出采购一套操作系统。Paul Allen 抓住机会与 Seattle Computer Products 的 Tim Patterson 签约，向其支付了不到 10 万美元，获得了其 DOS 操作系统的版权并进行了一些修改，从而做成了与这个神秘客户（IBM）的大买卖（见附图 2）。

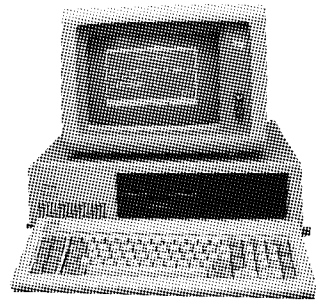
今天的 Windows 系列操作系统仍然兼容 DOS，这个系统对于老一代计算机用户来说再熟悉不过了。



附图 2　Microsoft 的秘密交易

早在 1980 年 7 月，一个负责“跳棋计划”的 13 人小组秘密来到佛罗里达州波克罗顿镇的 IBM 研究发展中心，开始开发后来被称为 IBM PC 的产品。一年后的 8 月 12 日，IBM 公司在纽约宣布第一台 IBM PC 诞生，这个开创计算机历史新篇章的时刻，迄今正好 25 年。

第一台 IBM PC（见附图 3）采用了主频为 4.77MHz 的 Intel 8088，操作系统是 Microsoft 提供的 MS-DOS。IBM 将其命名为“个人电脑（Personal Computer）”，不久“个人电脑”的缩写 PC 成为所有个人电脑的代名词。IBM 原来预计在一年中售出 241 683 台 PC，然而用户的需求被大大低估了，实际上一个月的订货量就超出了预计。



附图 3　IBM PC

2.5　输入输出系统

前面介绍的计算机核心部件决定了计算机的能力。但是，只有超强的能力还不行，计算机要能够与人类沟通，接受人类的指令，并把它们的工作成果显现出来才能为人所用。这就需要借助输入输出设备了，也就是我们常说的外围设备。

外设的种类非常多。常见的有显示器、键盘、鼠标、打印机、音箱、麦克风以及外部存储设备，如硬盘、光盘、U 盘等。这些基本上是目前每台计算机的必备外设了。除此之外，一

些专业用机还会配备扫描仪、绘图仪、刻字机、触摸屏等。数码摄像机、数码相机也都可以看作是计算机的外设。

本节介绍有关这些外设的知识。这包括外设的制造原理、同主机通信的方式方法以及我们通过什么样的形式来控制和使用这些设备。

2.5.1 外围设备

任何一种外设首先必须能够与主机通信。通常外设是通过总线和主机通信的。从图2-10上看到，为了连接到总线，外设都要通过一个叫做I/O接口的部件。接口，顾名思义就是两个“物体”的连接处。对于计算机系统来讲，“物体”既可以是看得见摸得着的硬件也可以是逻辑上的软件。所以接口可以是电路等部件，也可是软件之间的逻辑连接处。我们这里说的I/O接口是指外设同主机相连的部件及其软件控制。

接口不只完成一个连接工作。它还要让主机能够分辨并找到其所连接的外设，因为总线上可能挂有很多设备。当主机想使用一个外设时，会在总线上发出设备的设备码。当接口发现其与自己的设备码相同时，就会给CPU发送一个信号，这样CPU就能找到外设了。接口中一般有三类寄存器，体现了接口的三个功能。它们是命令寄存器、数据寄存器以及状态寄存器。命令寄存器用来存放CPU发给外设的命令。数据寄存器则负责缓存外设与主机之间交换的信息。状态寄存器里面的标志位会反应外设的状态。

本节先从外存说起，然后介绍显示器、打印机这两种常见的输出设备，最后再简单说说常用的输入设备：鼠标和键盘。这一节结束后，你会对我们司空见惯的这些设备有更新的认识。

小提示

外设通过I/O接口同主机相连。
I/O接口是指外设同主机相连的部件及其软件控制。
I/O接口中一般有三类寄存器，分别是命令寄存器、数据寄存器和状态寄存器。

1. 外部存储设备

在存储器一节中，我们已经对存储系统的结构有了一个认识，知道由于各种存储器件在速度、成本上的区别致使存储系统被设计成一种层次结构。从层次结构的顶部开始向下，我们对存储器的存取速度要求越来越低，但是容量要求越来越高，并且希望价格越来越便宜。此外，信息保存时间的要求也越来越长。在这个层次结构的最底层就是外部存储器。外部存储器是目前在存储层次结构中唯一支持信息长期存储的一层。关于层次结构的上几层前面都进行了介绍。本节将详细介绍外部存储器及其辅助访问设备。

现在常见外部存储器有硬盘、软盘、U盘和光盘。从严格意义上来说，软盘、光盘应该叫做存储介质，同纸张的功能类似。而读写这几种介质的设备才是外设，如光盘驱动器。但是这些介质与其读写设备是密不可分的，所以我们都放在一起介绍，并且将它们统称为外设。软盘已经很少使用了，所以就不再介绍。其余三种虽然都是外部存储器，都可以支持大容量长期存储，但是它们具有各自不同的特点，由不同的材料制成，应用场合也各不相同。下面首先从硬盘说起。

1）硬盘

(1) 硬盘的存储原理

硬盘？尽管你已经很熟悉了，可是有关它的内容实在很多也很重要，所以还是要唠叨唠叨，也只好请你耐心地看下去，或许你能发现一些不知道的事情。硬盘是依据什么原理存储信息的？是电磁感应。能讲出这个说明你的物理学得不错。如果没讲出来，也不要灰心，这不影响你成为 IT 精英。确切地说，硬盘存储信息是依据两个原理。一是奥斯特发现的电流的磁效应。也就是电流周围可以产生磁场，而且不同电流方向产生的磁场方向也不同。第二才是电磁感应现象，即法拉第通过大量实验证实的，当闭合电路的一部分导体做切割磁力线运动时，电路中就会有感应电流产生。人们利用第一个原理来记录信息。根据信息是 0 还是 1，人们让闭合线圈中产生不同方向的电流，然后用这个电流产生的磁场去磁化一个可被磁化的材料，如小铁棒。这个铁棒就会随电流方向的不同而被磁化成不同的磁极方向。读取记录的信息时，人们让被磁化的铁棒在闭合线圈中运动，就可以在线圈中产生电流，因为磁铁磁极方向不同，产生电流的方向也会不同，这样就能分辨出记录的是 0 还是 1 了。

既然是电磁转换，硬盘中就应该有磁性介质材料。当然这并不是说硬盘里面是一块大铁棒，那能存多少东西啊？人们是把这个大铁棒粉碎成一个一个的小铁片，每个小铁片可以存储一个信息位。这些小铁片太小了，必须依附在其他物体上。实际上，人们可不是真的去分割铁块，而是在一个圆形的盘片表面涂上一层薄薄的磁性材料。开始时盘片是由塑料制成的，现在则常用铝合金材料，最近也有用玻璃的。从这能看出点儿“硬盘”这个名字的出处。硬盘中负责读取/记录信息的部件被称为叫做读写头。根据前面介绍的原理，读写头的主要构成被设计成线圈。读写头也被称为磁头，但实际上它是缠着线圈的铁芯。磁头与盘片的距离足够近时，它们产生的磁场就可以互相影响，进行电和磁的转换了。磁头下方的磁性材料小区域可以看作是一个小小的磁铁。图 2-20 和图 2-21 分别是硬盘的写入和读出原理示意图。

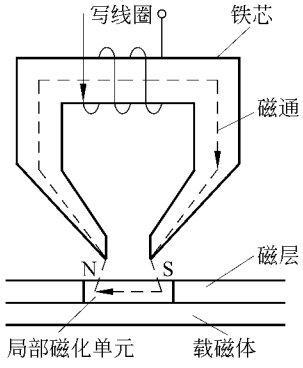


图 2-20　磁存储器写入原理

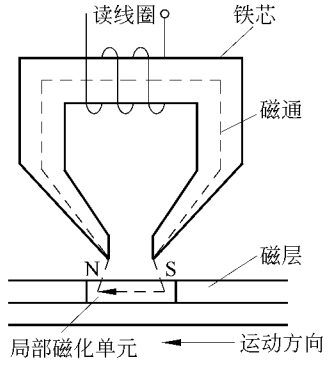


图 2-21　磁存储器读出原理

小提示

硬盘是依据电流的磁效应和电磁感应原理存储信息的。
硬盘盘片表面涂有一层磁性材料，用来存储信息。
磁头的主要结构是线圈和铁芯。

(2) 硬盘的机械构成

我们刚刚说到盘片是圆形的。可是我们看到的硬盘都是像图 2-22 那样的长方形的。其实我们平时看到的硬盘是硬盘驱动器和盘片的结合体,有些里面还集成有硬盘控制器。硬盘控制器是指管理硬盘的芯片。打开硬盘的外壳(千万不要自己打开,后果很严重!),就会见到如图 2-23[22]所示的硬盘内部。可以看到里面确实有圆形的盘片,旁边是一些其他的机械部件。为什么盘片不制成方形的?那样存放信息多规整,行列还清楚。如果盘片是方形的,要想在上面读取信息,那么需要把读写头在盘片上做水平和垂直两个方向的移动,并且要控制它不能移动出盘片的边界。要想信息读取的快,读写头的移动速度必须非常快。让快速移动的读写头在到达盘片边界时快速回头,想想也不是一件容易的事。所以只好另想办法。既然读写头移动不方便,那就移动盘片,而读写头不动。盘片怎样移动?当然就简单的办法就是让它转动,这样又省空间,又好控制,想让它转多快就有多快。既然要转动,那盘片只能做成圆形的了。

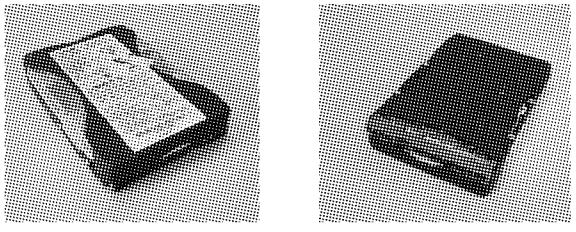


图 2-22 硬盘

从图 2-23 中可以看出圆形的盘片附着在一个中心轴上,它应该是能够转动的。如果盘片转动,而读写头不动,那么就可以读写盘片上位于读写头下方的以中心轴为圆心的一个圆环上的全部信息。如果把读写头沿盘片的半径方向移动一个位置,则可以读取另外一个圆环上的数据。所以在硬盘中存在两种运动,一种是盘片的绕轴转动,一种是磁头的径向运动。这两种运动分别由两个不同的电机控制。

实际上,为了增加存储容量,硬盘中不是只有一个盘片,而是一组盘片,如图 2-24 所示。这些盘片都固定在称为主轴的中心轴上。它们能够围绕主轴同时转动。每个盘片的两面都能存储信息,第一片和最后一片的外面除外,它们作为保护面。硬盘驱动器为每个盘片设了两个磁头,负责从盘片的两个面上读写信息。所有的磁头组成一个磁头组。这些磁头组会

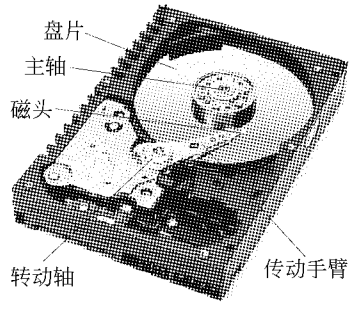


图 2-23 硬盘内部图

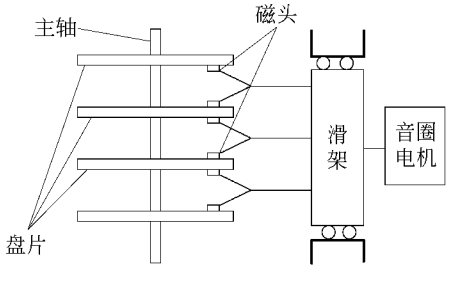


图 2-24 磁盘驱动结构

沿着盘片的径向方向运动。通过磁头组的径向运动以及盘片的圆周运动，盘片上的所有位置就都能够被访问到了。

为了避免划伤盘片，磁头在读写盘片时并不与盘片接触，而是悬浮在盘片上方。磁头依靠盘片高速转动产生的气垫浮力浮起。在盘片高速转动产生足够的浮力之前，磁头还是要和盘片接触。为此盘片上靠近主轴的地方留出了一块空白区，叫做启停区。在读写之前，磁头就放在启停区。图 2-23 的磁头就放在启停区内。在停止使用硬盘前，磁头都会回到启停区。所以，使用硬盘时要避免突然断电。如果你不能保证你的电力供应，那么为计算机准备一个不间断电源是必要的。

现在多数硬盘的磁头、盘片以及相关的运动机构都被密封在盘盒内。这样做可以起到很好的防尘效果，使硬盘不必对环境太挑剔。不过这样的盘盒就不能随意拆卸了。这就是你为什么不要自己随意拆开硬盘的原因。如果你真的自己动手看到了硬盘中的盘片、磁头，那么你的硬盘也就很危险了。密封，加上我们前面提到的技术：盘片固定且绕轴高速旋转、磁头径向运动、磁头接触式启停等，是一种被称为温彻斯特技术的典型特点。这种技术制造的硬盘也被称为温彻斯特硬盘，简称温盘。目前多数硬盘都属于这种类型。

下面把硬盘里面主要的部件总结一下。在图 2-24 中，圆圆的一组就是磁盘盘片。盘片组被固定在主轴上。有一个精度很高的主轴电机会带动盘片高速转动。磁头用于读写数据。音圈电机负责把磁头准确定位在信息的位置上。要提醒大家注意的一点是：磁头以及主轴电机在受到剧烈碰撞时易于损坏，所以使用硬盘时要避免剧烈振动。

(3) 硬盘上的信息组织

盘片制成圆形方便了磁头和盘片之间的运动。但是在圆形盘片上存放信息就有点儿麻烦了，毕竟圆形区域不像方方正正的形状那么好分割。下面就来说说硬盘上的信息是怎样组织的。我们把硬盘每个盘片记录信息的面叫做记录面，如图 2-25 所示。记录面从 0 开始编号，因为硬盘为每个记录面设有一个磁头，所以记录面号也就是磁头号。盘片做圆周运动，磁头不动，就可以访问一个同心圆环上的信息，所以盘片上每个同心圆环被称为一个磁道。所有记录面上的磁头是同时移动的，因而所有盘片上垂直方向在同一位置磁道上的信息，会被同时读出。我们把这些位于垂直方向同一位置上的所有同心圆环称为一个柱面。只有磁头同时读出或者相邻两次读出的数据是连续的、相关的，如同属一个文件，这样磁盘的读取效率才能高。所以磁盘上的数据通常并不以记录面为单位组织，而是按照磁道和柱面进行组织的。磁道和记录面都从 0 开始编号。每个磁道又被分成若干段，每段叫做一个扇区，或者扇段，如图 2-25 右图所示。有的硬盘扇区从 1 开始编号，也有的从 0 开始编号。

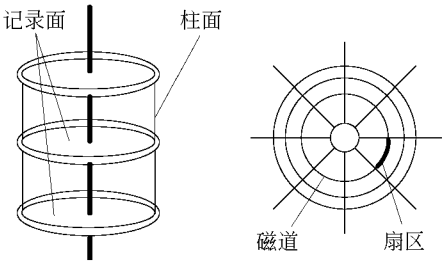


图 2-25　磁盘的数据组织

从上述介绍可以看出，一个数据在硬盘上的位置信息包括柱面(磁道)号、记录面(磁头)号、扇区号。因为计算机上可能有多个硬盘，所以还需要有一个驱动器号来指明数据在哪一个硬盘上。我们把这些位置信息称为磁盘地址，就像内存单元都有一个地址一样。通过这些信息最终可以定位到一个扇区上，因此扇区是磁盘编址的最小单位。

这里你可能会发现一个问题:磁道的周长是不一样的。那么每个磁道上的信息量是否相等呢?这个问题换个说法就是,假定每个扇区中的数据量是固定的,那么每个磁道上的扇区数应该是多少呢?靠近圆心的磁道短,外圈磁道则比较长。如果每个磁道上的扇区数相同,那就意味着外圈磁道上的信息密度小,内圈磁道上的密度大。既然内圈能够以一个大的密度存储信息,那么外圈也能够以这个密度存储信息。所以,外圈使用小密度存储,显然是一种浪费。要充分利用磁盘上的空间,所有的磁道就都应该采用和最内圈相同的密度存储信息。这样外圈磁道存储的信息就要比内圈磁道多。不过,这又带来一个问题。主轴是匀速转动的。这意味着无论哪个磁道,转动一圈的时间是相同的。也就是说所有磁道在磁头下转过一圈的时间是相同的。这样在同样的时间内,磁头如果在外圈磁道上,那么访问的信息量就大,而在内圈则小。因此,这样的磁盘就有一个最大数据传输率和一个最小数据传输率。早期硬盘所有磁道上的扇区数是相同的。而现在的硬盘则多是每个磁道有不同的扇区数。这种技术被称为分区域记录技术。

小提示

温彻斯特技术是目前制造硬盘的常用技术。
温盘的特点如下:
硬盘中有一组盘片被固定在主轴上,随主轴高速转动;
磁头、盘片以及相关机械部件和电路被密封在盘盒内;
磁头做径向运动;
采用接触启停式磁头。
硬盘上的信息按照记录面、磁道、扇区的方式组织,以扇区为单位存取。

(4) 硬盘主要参数

关于硬盘已经介绍得相当多了,加上原来的知识,现在你对硬盘已经有了足够的了解。如果正好计算机上的数据也多得快要放不下了,那么就可以自己去为计算机添一块硬盘了。

你要做的第一件事就是决定新硬盘的容量。相对于内存来说,硬盘的容量可大多了。内存现在刚刚能够以 GB 为单位来衡量。而硬盘目前(2008 年)一般都几百吉字节了。根据对业务的测算,你决定这次要买个 300GB 的硬盘。

于是你信心百倍地来到电脑市场,找到了一家卖硬盘的柜台(当然你完全不必这么辛苦,直接通过网络就可以把这件事情完成,而且可能买到更好的价格。不过暂时假定你想活动一下腿脚,锻炼一下口才,顺便感受一下电脑市场的火热气氛)。你很专业地告诉卖主你要买一个 300GB 的硬盘,而且性能要好。然后你开始等着他向你推荐硬盘,并做好侃价的准备。令你意想不到的是,卖主并没有给你拿出硬盘,而是问你需要什么接口类型的硬盘。这是什么意思呢?噢,我们忘了介绍硬盘的接口了。看来每个领域的知识都是无穷无尽的,所以经验就是,任何时候都不要以为自己什么都知道了。

下面介绍硬盘的接口以及常见的接口类型。硬盘接口主要是指硬盘与主机系统间的连接部件,作用是在硬盘和主机内存之间传输数据。接口类型主要是指接口采用的标准。其实在图 2-11[14]介绍总线的图中,就已经出现了硬盘的接口。我们知道外设都是通过接口连接到总线,然后再与其他部件通信的。不同的硬盘接口决定着硬盘与计算机之间的连接速度。

常见的硬盘接口有 IDE、SCSI 和 SATA 几种类型[21]。

IDE 是集成驱动组件(Integrated Drive Electronics)的缩写。意思是把控制器与盘体集成在一起的硬盘驱动器。IDE 接口也叫 ATA(Advanced Technology Attachment)接口。它是普通 PC 的标准接口。因为 IDE 的价格都比较低,而且兼容性又强,多数主板上都有 IDE 接口,所以多数个人计算机都选用 IDE 接口的硬盘。当然也有一些服务器使用 IDE 硬盘。

SCSI 的全称为小型计算机系统接口(Small Computer System Interface)。它并不是专门为硬盘设计的接口。它可以看作是一条最多可以连接 1 个 SCSI 控制器和 7 个 SCSI 设备的总线。SCSI 接口应用的范围比较广。它可以支持多任务,并且速度要比 IDE 快,对 CPU 的占用也较少,并且支持热插拔。但 SCSI 硬盘价格较高,因此 SCSI 硬盘主要应用于中、高端服务器和高档工作站中。

SATA 接口就是串行 ATA 的意思(Serial ATA)。所以使用 SATA 接口的硬盘又叫串口硬盘。从名字上看 SATA 应该属于 IDE 接口。但是因为主板上目前使用的、标有 IDE 字样的接口插槽都是 40 针的并行 PATA(Parallel ATA)的接口,而 SATA 接口的插槽则完全不同,所以我们也把它单列出来说明一下。SATA 是 2001 年制定的一种新接口标准。它支持热插拔,并且数据传输率比 IDE 要高。

现在你对硬盘接口类型已经略有了解。根据情况,选一个 IDE 硬盘就够了。当然如果主板支持 SATA 接口,也可以考虑是不是买一个这种硬盘。这里假定你买的是 IDE 硬盘。

现在卖主就会给你拿出一些 IDE 硬盘,告诉你不同的价格。相对内存条来说,硬盘的每位价格非常便宜。不同的品牌会略有不同,但大体上相差不多。目前 100～300GB 容量硬盘的价格一般也就几百元。除了容量外,硬盘卖家还会告诉你每个硬盘的转数。从直觉和价格上你能猜到转数越高越好。实际上也是这样。转数就是主轴电机的转速。转速越高,单位时间内盘片在磁头下面通过区域就越多,因而硬盘的速度也越快。一般常见的转速是每分钟 7200 转。

除了前面几个基本参数外,在选硬盘时还要考虑另外几个和硬盘速度有关的参数。

一是平均寻道时间。这是指磁头在接到指令后找到指定磁道所需时间的平均值,单位为毫秒。显然这个时间越低越好。磁头定位好以后,还要等待相应的扇区转到磁头下方,这段时间称为旋转延时。这个时间应该是和转数相关的,所以考虑转数就行了。

在介绍存储系统的层次结构时,我们讲到过缓存。这个思想在硬盘这里又被利用起来。多数硬盘都增加了高速缓存,以提高硬盘的读取速度。现在 8MB 的缓存已经不是什么新鲜事了。显然,大的缓存对提高数据的访问速度是有帮助的。

此外,硬盘的数据传输率也是个非常重要的参数。它是指单位时间内能够从硬盘得到的数据量。不过这个参数有点儿复杂,它是随具体读写的情况变化的。对于采用分区域技术的硬盘来说,数据在硬盘不同磁道、不同扇区会导致数据传输率的不同。而且数据的存放方式也会影响硬盘的数据传输率。如果数据在硬盘上放得很分散,那么磁头就需要不断地移动定位,就会导致读取时间大大增加,降低单位时间内得到的数据量。而数据的存放方式是由操作系统决定的。因此可以看出,单单有好的硬件还不行,还需要好的软件来配合。就好像一个计算中心即使有最好的设备,管理上不去,效率也会受影响一样。在第 4 章中会介绍磁盘的存储管理。因为数据传输率变化太大,所以厂商在标示硬盘参数时更多是采用外

部数据传输率和内部数据传输率。外部数据传输率是指硬盘缓存和计算机之间的数据传输率。而内部数据传输率是指硬盘磁头与缓存之间的数据传输率，也就是将数据从盘片上读取出来，然后存储在缓存内的速度。

好了，经过综合考虑，并且费一番口舌之后，你终于把一块 300GB 的硬盘带回了家。

小提示

常见的硬盘接口有 IDE、SCSI 和 SATA 几种。
硬盘的主要参数有硬盘容量、主轴转速、缓存容量、平均寻道时间及数据传输率。

(5) 硬盘的安装

硬盘买回来之后，要做的事情就是把它安装到计算机上。我们先把机箱打开，找到一个空闲的支架。先别急着把硬盘固定上去。你要先进行跳线。从图 2-26 上可以看到硬盘的跳线针脚。所谓跳线，就是根据硬盘的使用情况，把一个小套子套到不同的针脚上。使用情况指的什么呢？一般系统都会有多块硬盘，系统需要这些硬盘有个主次之分。跳线就是说明这种情况的。一般跳线的说明都在硬盘背面的标签上，所以你要仔细看。因为你的机器上原来有一块硬盘，上面装载着操作系统，所以新买的这块硬盘应该作为辅盘。现在你要根据标签上的说明把小套子放到“辅盘”对应的针脚上。

做完这件事情后，就可以把硬盘固定在支架上了。然后用数据电缆线连接硬盘的 IDE 数据线接口(见图 2-26)和主板上的 IDE 接口。接口上有个小小的凹槽，可以保证不会把线插反。除了接口上的凹槽提示之外，数据线的某一侧通常会有一排红色的标线，它表示第一根数据线，它应该对准硬盘上标有 J1 的一端。插好之后就可以找一个空闲的电源线接口接到硬盘的电源接口(见图 2-26)上了。同样电源线接口带削角的插槽也使用户不用担心把电源线接错。这些工程师真的很聪明！他们就知道写在说明书上你不会有耐心去看。关于硬盘的硬件知识就到这里了。接下来就是通过操作系统格式化、分区等，然后就可以存取数据了。在这里我们就不介绍了，你自己上网去查查相关的资料吧。

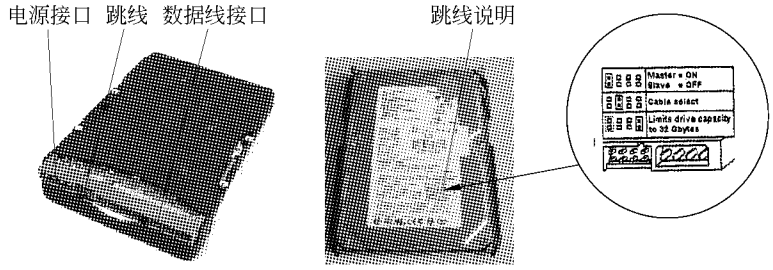


图 2-26 硬盘的跳线

小提示

硬盘使用注意事项：
防止剧烈震动。
防止突然断电。
注意散热。

2）光盘

现在来介绍光盘，当然不只介绍亮晶晶的盘片，还要说说它的读写设备——光盘驱动器。我们经常会接触到光盘，装软件、看影碟、听 CD 等。这些活动涉及的光盘包括 CD、VCD、DVD。这些不同名字的光盘到底是什么意思，有什么区别？要解释这些，先要来看看光盘的存储原理。

(1) 光盘的存储原理[29]

既然叫光盘，那肯定是和光有关系了。光盘主要是利用了光的反射原理。现在常见的光盘，按照是否支持写入，可以分为只读光盘 CD-ROM、一次性可写光盘、可读写光盘。这三种光盘都是利用光来读写的，但是制造材料和原理上略有不同。

我们先从 CD-ROM 说起。这名字的意思就是光盘只读存储器，所以这种光盘是不能写的。从商店里买来的 CD 唱碟和 VCD 影碟等都是这种 CD-ROM。为了表示数据，在 CD-ROM 光盘的表面上有两种不同的状态，一是平的，另一状态是凹坑。当用一束激光照射光盘时，有凹坑的地方和平的地方反射光的强度就会不同。通过对反射光强度的判断，光盘驱动器就可以分辨出光盘表面被照射的地方是凹坑还是平的。这样就可以把 CD-ROM 上用刻痕表示的数据解释出来了。图 2-27 是 CD-ROM 盘表面的剖析图。从图上可以看到中间一层具有不同的凹凸界面。在这层之上镀的是一层反射层，这是光盘看起来亮晶晶的缘故。镀的材料不同，光盘的颜色也不同。最后在外面是一层保护层。在这一层可以印上光盘的说明。

与硬盘使用同心圆环磁道不同，光盘上的光道是螺旋形的，如图 2-28 所示。数据按照内螺旋线顺序存放在光盘上，因此光盘的读取速度比硬盘要慢。光道上每位信息所占用的长度相同，也就是整个光盘的位密度是相同的。

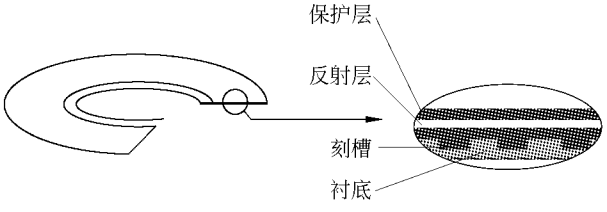


图 2-27　光盘表面剖析图

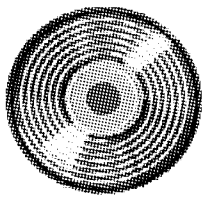


图 2-28　光道

为了降低生产成本、加快生产速度，只读的原版光盘都是批量生产的。先用激光通过软件烧制母版，然后根据母版形成压模，最后用塑料注塑成型机通过压模把数据压制到塑料盘上。因此，在出厂时光盘表面的这些坑就固定了，以后不能再更改，光盘也就只能读，不能写了。

小提示

光盘表面有凹凸不平两种情况，会使反射光的强度发生变化，因而可以表示信息 0 和 1。
光盘上的光道是螺旋形的。
整个光盘的信息位密度是相同的。

CD-ROM 光盘只能读不能写，且要批量生产。这也注定如果你想把自己的美妙歌声录制在光盘上送给朋友，是一件颇为烦琐而且昂贵的事儿。为了方便大家自制光盘，人们开发出了一次性可写光盘，简称 CD-R，也称 WORM(write one read memory)光盘。有了 CD-R 你就可以通过特制的刻录机把你的歌声、程序或者排演的小品存储在上面，送给任何你想送的人。现在最便宜的 CD-R 盘只要 1 元钱多一点儿。但是，这种光盘只能写一次，就像用钢笔在纸上写字，如果写错了，你只能换一张纸。那么这种光盘是怎样存储信息的呢？它是怎样支持写入呢？而为什么又只能写一次呢？道理说出来也不难理解。CD-R 光盘中有一个特殊的染料层[23]。这种染料在高功率激光的照射下会发生化学变化形成类似 CD 盘上的凹坑。不过这种变化是不可逆的，因而这种光盘只能写一次。当读光盘时，用的激光强度要比写的时候低很多，所以不用担心把这种光盘读坏了。从这里可以看出解决问题的关键就是一种特殊的染料。其实事情往往是这样，问题的解决方案说起来、做起来都不难，可是能第一个把方案想出来却不是件很容易的事。这也是要保护知识产权的一个原因吧。

除了 CD-R 外，还存在一种支持读写的光盘。这种光盘被称为 CD-RW，即表示可读可写 CD。它可以作为硬盘的替代。这种光盘可以多次擦写。目前的实现技术主要有热磁反转和相变记录两种。热磁反转光盘称为 MO(magnet optical)光盘。它的记录层采用对温度敏感的磁性材料制成。这种材料在温度达到特定高度时，就会被外界磁场改变磁极方向。低于这个温度时，就保持原来的磁极方向。相变光盘是利用一种存储介质的特殊性质：在特定温度下能够进行晶体态和非晶体态的可逆转变。因为晶体和非晶体对光的反射和折射率不同，因而可以表示出两种不同的状态，这两种状态就可以被解释成 0 和 1 了。

一般光盘的存储容量在 600M 左右。这个容量对多数影片来说都需要两张光盘才能放得下。DVD 通过一系列的技术手段把容量扩展到吉数量级。DVD 采用的技术包括：缩小光盘上光道之间的距离，减小每个凹坑的长度，使用两个盘面记录数据，以及在一个面上制作多个记录层。同一面上多个记录层，听起来有些悬。设计者把下面的记录层用激光可以穿透的、半透明的材料制成。这样激光就可以通过这一层读取它上面一层上的数据。双面双层 DVD 的容量可以达到 17GB。不过这种盘片对生产工艺要求较高，因而生产成本也比较高。

(2) 光盘驱动器[29]

访问光盘上的数据需要有光盘控制器和光盘驱动器的配合。光盘控制器和光盘驱动器一般集成在一起，所以统称光盘驱动器。CD-ROM、可擦写光盘和 DVD 光盘的原理各不相同，所以它们的驱动器也不一样。这里只简单介绍 CD-ROM 光盘的驱动器。

CD-ROM 光盘驱动器主要由主轴电机驱动机构、定位机构、光头装置及电路等构成。和硬盘驱动器的机理有些类似。光驱的工作模式也是通过主轴电机高速转动光盘、光头径向运动定位到数据位置读出数据。

光头主要由激光二极管、光电二极管以及一系列透镜构成。激光二极管会产生一定波长的激光束。激光束照射到光盘上，会被反射回来。反射光照射在光电二极管会被转换成电流。因为光盘上的凹凸会导致反射光的强度不同，进而形成不同强度的电流。光驱再对其进行相关的处理和解释，就可以得到光盘上的数据了。

速度是经常提到的光驱参数之一。这个速度是指光驱的数据传输率。习惯上经常以倍速为单位来说光驱的速度。这是因为光驱的数据传输率是以 150kB/s 为基准成倍增加的。传输率是 150kB/s 的光驱被称为单倍速光驱。这样你就可以自己计算双倍速、16、32 等倍

速光驱的速度了。

提到光驱的速度，还要多说几句。目前使用的光驱设计技术中，光盘的转动方式有两种。一种叫做恒定线速度（Constant Linear Velocity，CLV），另一种称为恒定角速度（Constant Angular Velocity，CAV）。恒定线速度方式下，光头在单位时间内经过的光道长度是固定的。因为每位信息在光道上占用的长度相同，那么光头在单位时间内读取的数据量也是恒定的。可是因为光盘外圈光道的周长要比内圈长，所以为了保持恒定线速度，当光头在外圈光道上时，光盘的转动速度应该比在内圈时快。也就是说主轴电机的转动速度必须不断随光头的位置变化。这对于电机来说是个不小的挑战，会使电机的寿命缩短。所以恒定线速度只用在速度较低的光驱中。恒定角速度则是指光头相对于光盘中心的角速度是恒定的。这也就意味着，无论光头在光盘哪个光道上，光头相对于光盘中心移动的角度是一定的。这样光头在外圈光道上移动的线距离就要长于在内圈时的距离。在数据位密度不变的情况下，光头在外圈单位时间内扫过的数据位就要多于内圈。所以在角速度恒定的情况下，光驱的数据传输率是变化的。厂家给出的光驱数据传输率多数是指访问最外圈光道上数据的传输率。高速光驱多采用恒定角速度的方式。所以有时候你会发现你的高速光驱并不比低速的快，这其中的道理你自己解释一下吧。

CD-ROM 光驱采用的接口主要有两种：IDE 接口和 SCSI 接口。IDE 接口光驱价格相对比较便宜，安装也方便，是目前普遍使用的光驱接口方式。SCSI 接口的光驱价格相对要贵得多，需要 SCSI 接口卡支持，安装也较烦琐，但具有稳定、CPU 耗用低的优点，因而网络服务器广泛采用的都是 SCSI 接口的光驱。

IDE 光驱的安装和前面介绍的 IDE 硬盘类似，相信你已经能够完成这个任务了。

小提示

常见的光盘种类有只读光盘 CD-ROM、一次性可写光盘、可读写光盘。
传输率是 150kB/s 的光驱被称为单倍速光驱。
光盘的转动方式有恒定线速度 CLV 和恒定角速度 CAV 两种。
恒定角速度下，厂家给出的数据传输率多数是指访问最外圈光道上数据的传输率。

3）U 盘

U 盘是最近几年兴起的一个新型存储设备。在 U 盘出现之前，移动存储主要依靠软盘。软盘只有 1.44M 的存储容量，早就满足不了用户的需求了。但是，由于没有找到合适的替代品，所以一直还在勉强使用着。U 盘的出现迅速地结束了软盘的历史。同软盘相比，U 盘体积更小，重量更轻，携带也方便，存储量更大，读取速度快，且价格也不贵。由于 U 盘的上述特点，它成了软盘的最好替代者。

U 盘也有称闪盘的。这两种称呼分别反映了 U 盘的两个结构特性。称其为 U 盘是因为它是通过 USB 接口与主机系统进行数据传输的。U 盘采用快闪存储器（flash）芯片存储数据。快闪存储器芯片要占到 U 盘体积的近 2/3。所以在结构上来说，闪盘应该说是一种更确切的描述。不过多数用户并不关心存储器内部采用什么机制存储数据，倒是通过什么方式把它连接到计算机上更为重要和直观，所以用户更接受 U 盘的称呼，尽管通过 USB 接口连接到计算机的存储设备不止这一种，比如，很多移动硬盘就使用 USB 接口。

闪存和 USB 接口技术都不是新技术。闪存技术应用开始于 20 世纪 80 年代中期，而 USB 接口是 WINTEL 在 20 世纪 90 年代公布的一个公用标准。

闪存主要是利用单晶体管的特性：在通电以后能够改变状态、不通电就固定状态来存储信息的。一般来说，闪存的写入速度比读出速度要慢。所以用闪存来做 MP3 播放器的存储器比较合适，我们一次存储的歌曲一般要听许多次。

目前常见闪存的类型有 NAND 和 NOR 两种[30]。为什么叫这种名字？如果你对电子电路比较感兴趣，可以去考究一下。这两种闪存在基本的数据存储方式和操作机理上完全相同，但是在电荷生成以及存储方案上有所不同，因而也形成一些不同的特性。

NOR 型闪存的随机读取速度较快，但是容量较小，所以多用在手机、掌上电脑等需要直接运行代码、读入速度要求快、容量要求较小、安全性要求较高的场合。NOR 型闪存的擦写次数约在 10 万次左右。

NAND 型闪存的读写是以页和块为单位进行的（一页包含若干字节，若干页组成储存块），这样更适合大容量数据的传输。但由于 NAND 型闪存内部没有存储控制器，并且串行存取数据，因此随机存取速度较慢。出现坏块时也会随机分布且无法修正。因此，它的可靠性相对 NOR 型也要低一些。相对 NOR 型闪存，NAND 型闪存可提供更大的容量，更快的写入速度。所以，NAND 型闪存多用于数据存储相关的领域，如移动存储产品、各种类型的闪存卡、音乐播放器等。它的成本相对较低，擦写次数较高，约在 100 万次。U 盘多采用这种类型的闪存。

U 盘内部的结构很简单，除去一大块闪存外，其余部分就是控制芯片以及一些必需的电路，如供电电路、时钟电路等。

小提示

U 盘通过 USB 接口与主机系统进行数据传输。
U 盘采用快闪存储器（flash）芯片存储数据。

2. 显示设备

显示器是一件非常重要的输出设备。计算机系统中没有一件设备能像显示器这样吸引我们的目光。显示器就像计算机的脸向我们展示计算机的所思所想。实际上，计算机的显示系统由两部分构成。一部分是显示器，也叫监视器。另外一部分是装在机箱里、插在主板上、负责将监视器和主板连接起来的板卡，叫做显示适配器，就是我们常说的显卡。显卡也有集成在主板内部的。

目前常见的显示器可以分为两大类：阴极射线管（Cathode Ray Tube，CRT）显示器和液晶（Liquid Crystal Display，LCD）显示器。其他类型的显示器还有等离子显示器（Plasma Display，PD）、发光二极管（Light Emittling Diode，LED）、电泳显示器等。

原来计算机系统中使用最广泛的是 CRT 显示器。它的成本较低、色彩鲜明而且真实。但是它体积也比较大、很重，搬起来很麻烦，放在桌子上也比较占地方。现在液晶显示器因为体积小、没有辐射和静电而正在抢占 CRT 的阵地。液晶显示器的价格正随着市场的扩大、成本的降低而逐渐降低。但是，液晶显示器的色彩比起 CRT 来还是略逊一筹，所以在图形显示方面还是不能与 CRT 同日而语。因此 CRT 显示器仍然是图形工作站以及游戏

玩家的首选。

1）CRT 显示器的成像原理[24]

在介绍 CRT 显示器的显示原理之前，要先说说三原色原理。我们眼中看到的五彩缤纷的世界是由于不同物体对不同色光反射的不同而形成的。所有的色彩都可以通过选定三种颜色的单色光以适当的比例形成。这三种被选定的颜色称为三原色。三原色必须是独立的，也就是说其中的任一种原色都不能由其他两种混合得到。人的眼睛对红、绿、蓝三种颜色比较敏感，而且这三种颜色能够配出大部分颜色。所以 CRT 显示器就选择用这三种颜色作为原色，也就是我们常在计算机中看到的 R(Red)、G(Green)、B(Blue)三个字母。(哎，用的又是英文字母的首字母。谁让都是讲英语的人研究的呢。现在能做的事情只能是学好英语，并且努力赶上去，将来用汉语拼音的首字母来命名我们的研究成果和产品，让外国人感觉必须要学好汉语。)

还是回来讲显示器的事情。既然讲到了三原色，你可能已经想到显示器实际上只产生三种颜色，只要调整三原色的比例就能形成所有的颜色了。你想对了。显示器就是这样做的。但是显示器是用什么产生这三种颜色的呢？这要借助于显示器两个重要的部件：荧光屏和电子枪。荧光屏就是我们平时看到的显示器前面的玻璃板。所有图像都是通过它显示出来的。为什么叫荧光屏呢？而不是玻璃屏？因为这块玻璃的里面涂满了荧光粉。荧光粉可不是一般的东西。它有一个特性：在受到高速电子束的轰击时，会发出光来，而且光的强弱还会根据电子束的强弱变化。为了能够发出三原色的光，荧光屏上就涂着会产生这三种原色的荧光粉。

那么这三种颜色的荧光粉怎么涂才能实现配色呢？为了在屏幕上显示一幅图像，我们的做法是把图像分割成一个一个细小的点，每个点叫做一个像素。每个点都有一个颜色。荧光屏也被分成一个一个细小的格子，每个格子显示一个像素点。要想在显示器显示一幅图像，只要按照每个点的颜色把它们都显示在屏幕上就行了。显然每个像素点需要有三种颜色的荧光粉以便配色显示各种各样的颜色。在不知道显示具体图像之前，荧光屏上每个点的颜色是不知道的，而且不同图像不同位置的颜色也是不一样的，这荧光粉可怎么涂啊？看样子麻烦来了。好在科学家发现了人眼的一个缺点：在超过一定距离后人眼的分辨能力就不行了，如果把三种颜色不同的光照在距离很近的点上，人眼看起来和混色的效果一样。这被称为空间混色法。这样事情就简单了。把每个像素点分成三个区，涂上产生三种原色的荧光粉就行了。为了产生一个带颜色的像素，只要用三束强度不同的电子束同时分别轰击像素点上的三个不同的荧光粉区就行了。好了，现在问题好像已经解决了。

不过，还有一个小问题，那就是屏幕上应该有多少个像素点？对于一幅图像，把它分得越细，那么用这些点再还原出来的图像和原版就越接近。所以，屏幕上的像素点应该越多越好。早期显示器屏幕上的像素点每行有 800 个，每列有 600 个，也就是一共有 800×600 个像素点。每屏上能够显示的像素点的个数被称为显示器的分辨率。显然，在同样面积下，分辨率越高显示效果越清晰。现在很多显示器的分辨率已经达到 1600×1280。

小提示

CRT 显示器根据三原色原理产生色彩。

显示屏上能够显示的像素点的个数称为显示器的分辨率。

那么在多大的屏幕上显示这些像素点呢？这就是显示器的屏幕尺寸。人们常说一个显示器是15吋的或者18吋的，指的就是屏幕大小。15吋是指屏幕的对角线长度是15英寸。同样的像素点数分布在不同尺寸的屏幕上效果是不一样的。相同的分辨率在不同屏幕尺寸上显示的效果显然不同。所以，人们引入了另外一个参数来说明这件事情。它就是两个像素点之间的距离，被称为点距。这是选择显示器时一个重要的参数。常见的点距有0.28mm、0.27mm、0.26mm、0.25mm等。点距越小，显示器能达到的分辨率就越高，显示的画面就越清晰。

现在假定显示器有800×600的分辨率，也就是屏幕上一共有800×600个像素点。要显示一幅图像，那么需要这些像素点都亮起来。所以我们需要有800×600×3个不同的电子束。这可不是个小数目，对电子枪的要求可够高的。而实际上，CRT显示器并不是这样做的。它只用了一个能产生3束电子束的电子枪。这一次科学家又利用了人眼的一个特点：人眼会把看到的东西在视网膜上保留一段时间，这就是人们常说的视觉暂留。CRT显示器用一个产生3束电子束的电子枪快速地、按次序地轰击屏幕上的所有像素点。因为速度足够快，由于视觉暂留、加上荧光粉的余辉，人们根本没有感觉到这些像素是按顺序一个一个相继发光的。电子束从屏幕左上角开始，沿水平方向从左到右扫描，到达屏幕最右端后即迅速到下一行的左端继续扫描。当电子枪扫描到屏幕的右下角时，再直接回到屏幕的左上角重新开始下一屏的扫描。计算机的显示器多是这样逐行扫描的。而电视机有的是隔行扫描的，就是先扫描完所有奇数行后再回头扫描偶数行。在水平方向上的扫描速度，即单位时间内能够扫描的行数被称为显示器的行频，以kHz为单位。而单位时间内整个屏幕被扫描的遍数被称为显示器的场频，也就是平时说的刷新率。刷新率过低，即扫描的速度太慢，人眼就会感觉到屏幕在闪烁。这样容易造成眼睛疲劳，甚至恶心。早期显示器的刷新率都在60Hz，现在视频电子标准协会VESA规定85Hz为无闪烁的标准场频。

小提示

显示器两个像素点之间的距离，称为点距。
显示器利用了空间混色法以及人眼的视觉暂留特性。
单位时间内完成的对整个屏幕扫描的次数称为显示器的场频，即刷新率。
单位时间内能够扫描的行数被称为显示器的行频。

2）显示适配器

当要显示一幅图像时，计算机会把图像的信息存储在一个显示缓存区中。这个显示缓存区又称为视频存储器（Video RAM，VRAM）。假设图像以像素点阵的方式存储在VRAM中。假定显示器的分辨率是1024×768，可以表示16种颜色。图像的信息在显存中以二进制形式表示，为了区分16种不同的颜色，那么至少需要4位二进制数。因此在显存中一幅图像要占用的空间为1024×768×4/8＝393 216字节，也就是近400KB的空间。现在一般显示器都支持32位颜色，也就是用32位表示颜色，这样能够表示的颜色数可以高达2^{32}种。一幅图像占用的空间也增加到1024×768×32/8＝3 145 728字节，即3M多的空间。显存的存储速度必须满足刷新率的要求。因此，显示器的分辨率、刷新速度、能够表示的颜色等除了跟显示器自身有关系外，还受到显存的限制。它们二者必须配套，才能达到预

期的效果。但是显存并不是放在显示器中的，而是在一个和显示器配套使用的设备——显示适配器中，也就是我们通常所说的显卡。通常它的制造厂家也与显示器是不同的。

显卡通常是一块接口卡，插在主板特定插槽上。也有很多显卡被集成在主板上。图 2-29 是一块常见的显卡。显卡除了装有用来缓存图像信息的显存外，还负责将一些其他格式的图形/图像信息转换成显示器能够识别的信息，实际上 CRT 显示器只能接受模拟信号。显卡上有接口负责连接显示器，所以显卡是显示器和主机系统的连接桥梁。

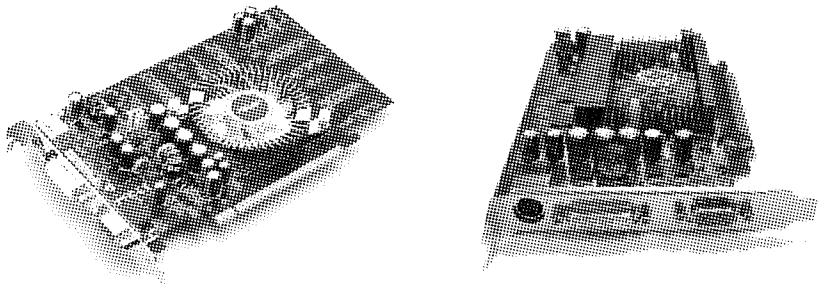


图 2-29　显卡

显卡主要由显示芯片、显存、数据转换器等几部分组成。以前显卡的主要作用就是连接显示器以及向其提供显示数据。例如，CRT 显示器只能接受模拟信号，显卡需要把计算机中按数字信号存储的图像信息转换成模拟信号发送给显示器。而现在显卡上一般会集成处理器，负责处理跟显示有关的一些工作，以便分担主机 CPU 的工作、加快显示的速度。这样对显存的要求就提高了，它需要存储更多的信息，因而现在显卡显存的容量越来越高。

显卡同显示器连接的接口类型主要有两种[31]。比较常见的是 15 针 D-Sub 输入接口，也叫 VGA 接口，如图 2-30 所示。VGA 接口传递给显示器的是经过转换的模拟信号。第二种接口叫 DVI(Digital Visual Interface)数字接口，如图 2-31 所示。DVI 接口传递的是数字信号，因此比较适合连接数字化显示设备。它可以避免用 D-Sub 输入接口连接数字显示设备带来的数/模、模/数两次转换带来数据损失。DVI 接口还支持热插拔。

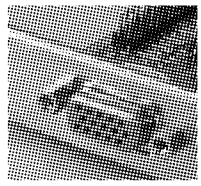


图 2-30　VGA 接口

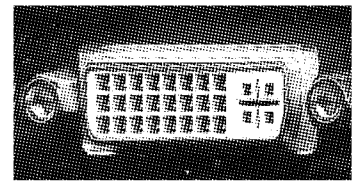


图 2-31　DVI 接口

随着显示要求的不断提高，显卡与主机的通信接口在不断变化。早期的显卡功能比较简单，和主机通信主要采用通用的外设接口，如早期的 ISA 接口、VESA 接口、PCI 接口等等。目前这几种接口的显卡都已经逐渐被淘汰。AGP 接口是为了提高主机和显卡的通信速度，专门为显卡设计的接口标准。而目前渐成主流的显卡接口则是称为 PCI-Express 接口。在总线一节已经介绍过 PCI-E 也是一种通用总线标准，可以提供比 PCI 更高的速率。不过 PCI-E 接口显卡的价格还比较高。

小提示

显示器必须和显卡配合使用。
显卡目前使用 AGP 接口和 PCI-E 接口的较多。

3. 打印机

打印机是最早使用的输出设备，也是目前仍然在使用的设备。打印机按照打印原理可以分为针式打印机、喷墨打印机和激光打印机。其中激光打印机的质量最高，速度最快，价格也最贵。我们先来了解一下激光打印机是怎样工作的。

1）激光打印机的工作原理[32]

激光打印机的核心部件是以碳粉和感光鼓为主的碳粉盒以及激光扫描系统。感光鼓是一个用合金制成的圆筒，表面镀有一层光导体感光材料，通常是硒，所以这个感光鼓也叫硒鼓。

图 2-32 是一个激光打印机的内部结构示意图。当打印机准备好数据开始打印时，首先会给硒鼓充电，让它表明均匀地带有一层电荷(或正或负)。接下来，打印机的控制电路根据要打印的数据形成激光束对硒鼓进行扫描照射。要打印的地方关闭激光，空白的地方打开激光束。硒鼓上被光照射的地方就变成了导体，电荷就跑掉了。这样硒鼓上带电的区域就是要打印的影像。硒鼓转动，激光继续扫描下一行。而前面的静电影像就转到碳粉盒前，这样带电的区域就会吸附上碳粉，在硒鼓表面形成要打印的信息。硒鼓继续转动遇到了打印纸。打印纸在此之前被充上与硒鼓相反的电荷。由于正负电荷相互吸引，硒鼓上吸附的碳粉就传给了打印纸。打印纸再经过高温滚筒的烘烤，碳粉熔化就永久性地印在纸面上。你现在明白为什么每次打印出来的纸都是热乎乎的了吧？然后硒鼓被放电，表面残留的碳粉也被清除掉，打印机重新开始新的打印任务。

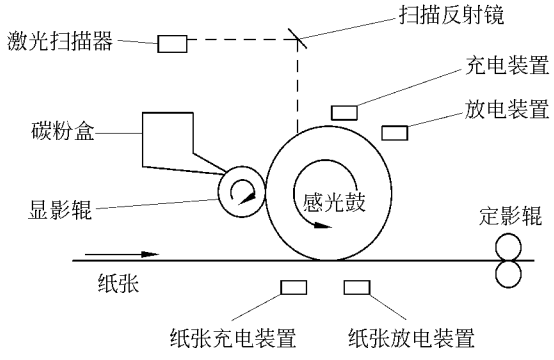


图 2-32　激光打印机内部结构示意图

硒鼓的价钱不便宜，属于耗材类。它的使用寿命是有限的。这是使用激光打印机的日常开销。现在很多打印机支持鼓粉分离的设计技术以降低使用者的成本。这样，当粉盒里的碳粉用完，只要更换粉盒，而不用再更换硒鼓了。那什么时候换硒鼓呢？一般这种鼓粉分离的打印机都带有一个硒鼓计数器。它会告诉我们什么时候硒鼓的寿命到期了。

2）打印机的参数[33,34]

现在来说说打印机的几个关键参数。首先要介绍的参数是分辨率，它的单位是 dpi。没见过这个缩写吧？其实就是每英寸上(可打印)的点数(Dot Per Inch)。在讲鼠标的性能时还会遇到这个单位。当然，固定大小区域内打印的点数越多，图像就越连续，也就越清楚。如果只打印文字，一般不必要求最高的分辨率。因为分辨率高的打印机价格也贵。

第二项参数就是打印速度。现在多用每分钟打印的页数描述，缩写是 PPM(Pages Per Minute)。这个数据也分两种情况。一是指打印机在最快时能够打印的页数。另一则指打印机在连续打印时每分钟的平均打印页数。所以不能光看一个 PPM，一定要弄清楚到底指的是什么。当然无论哪个值，厂家给出的都是最理想情况下的数据，现实往往和理想会有些差距。

除此之外，打印机能够打印的最大幅面也是一个重要的参数，比如最大能够打印 A4 幅面或 A3 幅面。当然支持幅面越大，打印机的价格也越高，从打印机的个头上也能看出这个特点。好在平时并不需要打印很多大幅的图像和文字，偶尔需要可以去找专业文印社。这样可以节约很多资金。这也是合理利用资源的一种办法。

对于激光打印机还要考虑它的首页输出时间(First Paper/Print Output Time，FPOT)，也就是从打印机接收信息开始到第一页打印出来的时间。一般打印机要花 5 秒以上的准备时间。

每台打印机都有自己的内存，也叫做缓冲区。缓冲区越大，一次输入数据就越多。这样可以减少其与主机的通信次数、加快打印速度、减少主机的负担。而且，现在越来越多的打印机以网络连接形式为多台主机服务。大的缓冲区也就更加重要了。

打印机使用的传统接口是并行接口。并行接口和串行接口是最常见的两种通用接口。以前的鼠标就是接在串行接口上的，而打印机则多连接在并行接口上。串行接口是指外设与接口间是一位一位传输数据的，因而速度较慢。而并行接口则是同时传送一个字节或者一个字，也就是一次能传送多位。总线上的数据传输是并行的。因此串行接口从外设串行得来的数据需要进行处理后并行地放到总线上。目前，使用 USB 接口的打印机越来越多。也有打印机使用 SCSI 接口，但是这种接口还是很贵，因而使用并不广泛。

现在很多单位都使用网络打印机，以提高资源的利用率。网络打印一般通过两种形式实现。一是选择一台主机做打印服务器。这台主机通过并行口连接打印机，其他主机通过网络将打印任务传递给打印服务器，再由打印服务器将任务发送给打印机。另外一种则是把打印服务器直接内置在打印机中，打印机提供网络接口连接到网络，网络中的主机将打印任务直接发送给打印机。

小提示

打印机的基本参数包括打印分辨率、打印速度、首页输出时间、内存大小、接口类型等。

网络打印一般有两种形式，一是使用一台主机做打印服务器，二是使用内置打印服务器的打印机。

4. 鼠标

鼠标器是最常用的输入设备。因为拖着一个长长的尾巴，看起来像个老鼠而得名。现在的无线鼠标因为没了尾巴，看起来有点儿名不符实了。小小的鼠标在不长的发展历史里，

也经历了不断创新改革的历程。开始的鼠标叫做机械式鼠标，现在还有人在用。但是用惯了新式光电鼠标的人怎么用都觉得机械式鼠标又沉又笨，尤其是在越来越多的用户进入鼠不离手的境界后，机械鼠的沉重因为累积而被放大。这导致机械式鼠标最终退出舞台。因此，我们这里就不再介绍机械鼠的原理了，直接介绍光电鼠标。

作为输入设备，鼠标要传递给计算机的信息很简单，就是鼠标的移动状况。因此鼠标的任务也就是判断出自己位置的变化，并通知给计算机。光电鼠标通过外界对光的反射判断自己的位移状况。光电鼠标里面有一个发光二极管，会发出光线，所以光电鼠标工作时内部总是亮通通的，有光线从底部射出。早期的光电鼠标必须与一个带格子的鼠标垫配合使用。当鼠标从鼠标垫的格子上经过时，反射光的强度就会发生变化。鼠标内部的逻辑电路可以计算出在两个方向上移动的格数，从而判断出鼠标的位移。但是必须与特制的垫子一起工作很不方便。因而新式光电鼠标不再用鼠标垫做参照物。它将底部表面反射回来的光线传到一个光感应器件内形成图像。光电鼠标每隔很短的时间（几毫秒）就这样拍一次照，然后比较两次图像的区别，从而判断出自己的移动方向和距离[35]。

对鼠标来说，最重要的指标就是它的分辨率。这里分辨率的意思是指鼠标能够感知的移动距离。以前最常用分辨率的描述是 DPI。这个缩写在打印机那里刚刚出现过。在这里表示鼠标每移动一英寸所能检测出的点数。当然对鼠标来说，DPI 越大，用来定位点数就越多，所以定位精度也就越高。机械式鼠标的精度都在 200DPI 以下，而光电鼠标能达到 400DPI，800DPI，甚至更高。因而你会感觉光电鼠标反应更灵敏[36]。

现在多用 CPI(Count Per Inch)来描述鼠标的分辨率。CPI 即每英寸测量次数。也就是鼠标在桌面上每移动 1 英寸距离鼠标所产生的坐标数。坐标数越多，当然鼠标的灵敏度也就越高。CPI 的倒数就是鼠标能够感知的最小移动距离。只有超过这个距离的移动，鼠标才能感觉到。

光电鼠标还有一个特有的性能指标叫做采样频率。它指光感应器每秒钟采集/分析图像的次数，也就是光电鼠标单位时间内照相的次数，单位用“帧/秒”。这个值当然越大越好。不然，如果鼠标位置移动太快了，光电鼠标前后两次照出的照片没有重合的地方，它就不知道怎么办了。这种情况被称为丢帧，会造成光标暂时消失。

早期的鼠标使用串行接口。串口通信的数据传输率太低，而且不支持热插拔。所以现在已经很少使用这种接口了。在串口鼠标之后出现了使用 PS/2 接口的鼠标。PS/2 接口是鼠标和键盘的专用接口，是一种 6 针的圆形接口。PS/2 接口的传输速率比串口稍快一些，是目前应用最为广泛的鼠标接口之一。鼠标和键盘的 PS/2 接口不能通用，一般 PS/2 接口旁边都有相应的图示，并且颜色也不同。通常鼠标接口为绿色，键盘接口为紫色。现在越来越多的新鼠标使用的是 USB 接口[37]。USB 接口比前两种接口的传输速度快很多，而且支持热插拔。可以看到使用 USB 接口的设备越来越多，说不定有一天一个 USB 接口上真的需要连接 127 种外设呢。

小提示

光电鼠标通过外界对自己所发出的光的反射情况来感知自己位置的变化。
光电鼠标如果采样频率太低，就容易导致丢帧。

5. 键盘

作为输入设备，从按键数量来看，键盘会比鼠标输入更多的信息。但是实现起来应该不比鼠标复杂，因为键盘不需要移动。键盘只需要判断出哪个键被按下，然后把这个信息传递给主机就行了。键盘采用很简单的原理得知键被按下的信息。键盘上的每个键对应一个开关。所有的开关排列成一个矩阵结构，每个键开关位于行和列的交叉处，如图 2-33 所示[39]。当有一个键被按下时，该处对应的开关就被闭合，键盘通过扫描每行和每列上的电压变化就能知道哪个位置上的键被按下了。

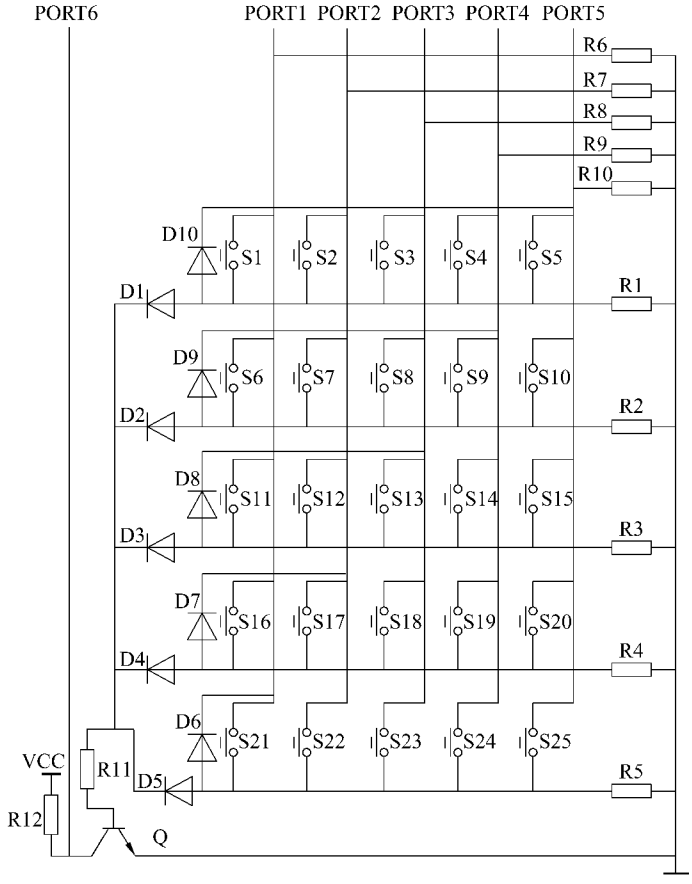


图 2-33　键盘矩阵结构示意图

判断出按键的位置之后，键盘需要把信息传递给主机。但是它不能只简单告诉主机哪行哪列的键被按下了，而是要说清楚按下的键代表什么意思。有两种方式可以把按键对应成字符。一是硬件方式。通过硬件直接得到按下键对应的字符。这种方式实现起来也不难。把每个键对应的字符存储在 ROM 中。然后把每个键的位置信息和对应字符在 ROM 中的地址用硬件做个对应。这样当某个键被按下时，其对应的字符就可以被读出。采用这种方式的键盘识别按键的速度很快。但是，每个键对应的字符不能改变。这种键盘被称为编码键盘。与其相对的就是非编码键盘。得知一个被按下键的位置后，非编码键盘通过软件在一张表中查找对应的字符。这样只要修改表的内容，键对应的字符也就改变了。所有

这些工作，无论软件还是硬件，都是在键盘里面的控制芯片的控制下完成的。

按照按键开关闭合的方式，键盘可以分为接触式和非接触式两种[38]。所谓接触式就是开关是实实在在地碰到一起。早期的机械触点式键盘就是典型的接触式键盘。这种键盘使用电触点接触作为连通标志，使用机械金属弹簧作为弹力机构。这种键盘因为弹簧和电触点都比较容易坏，所以已经被淘汰了。非接触式键盘的典型代表是电容式键盘。它用弹性橡胶制作的弹簧取代了机械金属弹簧，通过按键底部和键盘底部的两个电容极板距离的变化带来的电容量变化来获得按键的信号。一般电容式键盘的价格较高。目前使用较多的是一种被称为薄膜接触式的键盘。这种键盘中间有三层塑料薄膜，上下两层印有电路，在每个按键的下方，这两层薄膜上有对应的触点。中间一层上没有电路，它把上下两层隔开。但是，在每个键对应的地方留有圆孔。这样当某个键被按下时，上下两层薄膜的触点就接触上了。这种键盘也使用弹性橡胶弹簧，因而手感和电容式比较接近。但是它的制造成本较低。

早期的键盘多使用 AT 接口，也被称为“大口”。而现在的键盘则多用 PS/2 接口，俗称“小口”。

键盘的原理我们就说到这里。下面再简单聊几句键盘上的按键布局。不同键盘的按键个数会略有不同，如 104 键、101 键、83 键等。但是键盘中间的 26 个字母基本上都是一致的，而且位置也相对固定。你知道这个字母顺序排列有什么道理吗？好像什么道理也没有，没有一点儿规律可言，很难记。不过我们也都记住了。对于我们现在使用的计算机来说，这种排列确实一点儿道理也没有。实际上，这种排列是从打字机的键盘继承过来的。早期的机械打字机上每个字母按键控制一个金属杆，金属杆的末端刻着一个字母。为了让这些按键敲打出的字母在纸上一个挨着一个，这些金属杆的距离很近。当打字速度快起来的时候，这些金属杆就会相互打架。为此，打字机的设计者经过一番研究，把所有常用字母的距离都排得尽量远。这样就可以减少金属杆互相干涉的情况了。最后就形成了我们今天使用的这个键盘的布局。这种布局使我们的手指在键盘上的移动距离尽可能地长。现在我们已经不用金属杆来打字了，所以这种设计也就完全没有必要了。然而，尽管后来有很多更合理的键盘布局，但是因为人们已经习惯这种键盘而无法推广了。这种现象在软件行业也很多见，尽管很多新设计开发的软件比人们用惯了的一些软件更优秀、更合理，但是都因为无法战胜习惯而夭折。所以抢占市场先机非常重要。

小提示

键盘通过一个排列成矩阵形状的键开关来判断被按下的键。
编码键盘每个键对应的字符不能改变。
薄膜接触式键盘是目前使用较多的键盘。

2.5.2 驱动程序

前面介绍的几种输入输出设备机理和作用都各不相同，而且和主机的连接方式也不同。由此可想而知，主机也必须采用不同的方式操作这些设备。可是计算机怎样能管理和使用这么种类繁多的外设呢？尤其是我们经常听到一个词——即插即用，也就是说把一件外设插到主机上，然后就能够使用它。这些都是怎样实现的呢？实际上，计算机是依靠一个称为驱动程序的软件认识和操纵这些外设的。

驱动程序不是一个软件的名字，而是指一类软件。每个外设都有自己的驱动程序。第1章提到一个事实，用户或者说应用程序并不是直接操纵硬件的，而是通过操作系统平台来进行。例如，当我们准备打印自己写的技术报告时，我们直接用鼠标点了一下 Word 工具栏中的打印机那个小图标，通知打印机打印这篇报告。然后，打印机就开始工作，你的报告就被打印出来了。实际上，我们轻轻地一按，计算机要做很多工作。它要找到现在系统中安装的打印机有哪些？应该使用哪一个？要判断这台打印机是否处于就绪状态。决定好使用的打印机后，主机要和打印机通信，把打印任务发送给打印机。这涉及要了解通过什么接口传递，以及与这个接口通信需要采用什么命令。主机需要知道让这种打印机工作要发送一个什么样的信号，还要判断打印机的缓冲区目前的剩余空间是否够用等一系列事情。这些工作细碎而又烦琐，并且有些具体的动作会因为使用的打印机不同而不同。显然 Word 软件的设计者要想把这件事情做好必须好好了解一下所有的打印机以及接口。这个任务显然会牵扯 Word 软件设计者的很多精力，而 Word 软件的主要功能只是要给我们提供一个好的文档编辑环境。如果每件事情都这样做下去，这个软件恐怕不会写出来了。实际上，按照计算机系统的层次结构，Word 软件只专注实现排版功能，而像怎样使用打印机的事情则交给操作系统来实现。

操作系统为应用程序屏蔽了各种硬件复杂的特性，但是操作系统的设计者也不可能了解所有的外设，尤其是新的外设还在不断地出现。因而操作系统的设计者想了一个办法，让外设的设计者和生产者向操作系统提供访问这些设备的手段，即提供使用和管理外设的软件。这个软件就是驱动程序。毕竟每个外设的设计者熟悉自己外设的属性，知道怎样使用外设更合理、效率最高。因为操作系统要通过驱动程序才能认识和使用各种外设，所以一件设备连接到计算机上之后，在使用之前，必须要做的一件事情就是安装相应的驱动程序，如为新插的显卡安装显卡驱动程序、为声卡安装声卡驱动程序等。

驱动程序就相当于给了操作系统一个设备的使用说明书和工具。当然使用说明书要按照操作系统能够读得懂的格式来写。操作系统有很多种，如 Windows 和 Linux。它们对驱动程序格式的要求也不尽相同，因此每种设备在不同操作系统下都有驱动程序，如同一款显卡就要分别有 Windows 和 Linux 下的驱动程序。在不同操作系统下要安装不同的驱动程序，不能互换。此外，设备不同，说明书当然也不一样。因此每件设备都有自己的驱动程序，多数时候不同型号的同一种设备的驱动程序也不一样，如同一家公司不同型号的激光打印机都分别有自己的驱动程序。如果安装了错误的驱动程序，设备可能就不会正常运行。

可是我们好像并没有给键盘和硬盘安装驱动程序，这些设备一样能够使用。这是因为这些常见设备的驱动程序已经被集成到操作系统中了。有时候新安装一个硬件，如给计算机新换一个显卡，好像没安装驱动程序也能工作。发生这种情况有两种可能的原因。一是这种显卡比较常见，操作系统带有它的驱动程序。操作系统通过一些特殊的方式能够认出这种显卡，然后就自动安装了它的驱动程序。还有一种情况，就是操作系统并没有认出这种显卡，但是它选择并安装了一个最常见、最简单的显卡驱动程序。这时你的显卡能工作，显示器上能够出现图像。但是，显卡并没有正常工作。它的许多能力都没发挥出来，如分辨率本来可以达到 1024×768，但是现在只能是 800×600。要想让显卡正常工作，就需要找到并安装显卡匹配的驱动程序。

如果新装的硬件是很流行的类型，那么操作系统一般会带有它的驱动程序，因而会自动找到并安装它们相应的驱动。但如果是比较新的产品，那就需要你提供驱动程序了。一般当你买到一个硬件时，都会附带着一张有主流操作系统下的驱动程序盘。说明书上会告诉你怎样安装。你只要照着做就行了。但是过一段儿时间如果你的计算机出了问题，你不得不重新安装操作系统，那么这些驱动程序也需要重新安装。你有可能就找不到原来那张盘了。喜欢干净的你早把它当作垃圾处理了。这时候你首先要弄清楚需要驱动程序的硬件的品牌和型号，然后到网络上去找，尤其是到该硬件产品厂家的网站上，一般都会有收获。如果你使用的不是主流操作系统，那么附带光盘上一般不会带有这种操作系统的驱动程序。这时候你也需要这样自己去找，或者向卖硬件的经销商讨要，不过一般他也是从网上下载，所以你还是自己动手丰衣足食吧。互联网真是个好东西！

小提示

应用程序通过操作系统使用外设。
操作系统通过驱动程序识别、管理和使用外设。
必须正确安装驱动程序，外设才能正常工作。

2.5.3 外设与主机信息传送的控制方式

本节介绍主机和外设之间传送数据的几种方式。

第一种方式叫做程序直接控制方式。也就是由程序直接控制主机和外设之间的数据传输。我们是通过程序和外设打交道的。不过这中间要涉及很多不同的程序。例如，通过Word使用打印机，告诉打印机把你的漂亮简历(光漂亮可不行啊，当然如果你是学广告设计的除外)打印出来。应用程序把任务转交给操作系统，如Windows。而操作系统再去寻求驱动程序的帮忙，最终和打印机联系上。下面我们就看看驱动程序是怎样控制数据传输的。首先驱动程序命令CPU初始化外设，也就是告诉外设做好准备工作。在这个例子里外设就是打印机。然后，CPU就开始不断地用测试指令去检测外设的状态，即不停地去看看外设准备好了没有，直到外设准备完毕，给了CPU一个肯定的回答。CPU把数据传送给外设，即把你的简历传送给打印机，并开始等待，直到打印机打印完毕告诉CPU完成任务。在打印机打印期间，Word不能做别的事情，只有等待，实际上是CPU不能做别的事情。打印机打印的速度是很慢的，相信你一定等不及了，想收一下邮件看看。对不起，CPU正在等着打印机完成任务呢，不能干别的，你就等着吧。

显然，使用程序直接控制方式，CPU乃至整个计算机系统的效率都很低。造成这种结果的根本原因是CPU和外设串行完成任务，CPU的速度太快，相对来说外设的速度就太慢了。不仅浪费了CPU的时间还把你的时间也浪费了(当然你可以去喝点儿水、活动活动快直了的脖子，把这个时间利用起来)。所以，通常程序直接控制方式只用在与高速外设之间进行数据传输的情况。

既然程序直接控制方式效率这样低，那么应该怎样改进呢？其实也不难，不让CPU闲等就行了。在外设处理数据期间，让CPU去做别的事情。等外设做完了事情再告诉CPU一下就行了。第二种传输方式，被称为程序中断方式，就是依据这个思想。在这种方式下，当CPU告诉打印机进行打印之后，它便不再不厌其烦地去查看打印机的状态，而是回来继

续做它的事——为你服务。现在，你可以看看邮件了。这时 CPU 就去读取邮件接收程序，为你显示邮件。你也可以继续使用 Word 来写作业。打印机打印完毕会主动跟 CPU 说：我的工作完成了。这种方式就叫做中断。实际上外设是发出了一个中断信号。这个信号由一个叫做中断控制器的芯片来处理。CPU 每次执行完一条指令就来看看有没有中断信号发生。如果有，它就会停下手头的工作，去完成中断要求做的事情，如把另外一些数据交给打印机继续打印。实际上 CPU 并没有三头六臂，它只能一心一意做好一件事情。它的工作之所以能够被外设中断（或者叫打断更确切）是因为 CPU 主动来检查看有没有人想打断它。

程序中断方式的优点在于，如果没有外设想打断 CPU 的工作，那么 CPU 就可以做一点儿别的事，这样它的效率就提高了。当然在每条指令执行完了都要查看中断信号，也增加了 CPU 的负担。但这还是比只等着一个外设，什么都不做好多了。当 CPU 发现有中断信号之后，它会放下手头的工作，去处理中断信号请求的事情。等处理完中断请求，再回头继续做刚才的事情。

因为在主存和外设，尤其是外部存储器，之间可能会传送大量数据。为此人们设计了第三种数据传输方式，称为直接存储器访问（Direct Memory Access，DMA）方式。这种方式更进一步地把 CPU 解放出来。前面两种方式下，数据的传送都有 CPU 参与，需要占用 CPU 内部的寄存器。而 DMA 方式下，CPU 只在传输开始和结束时参与一下，中间过程全交给一个称为 DMA 控制器的硬件来管理。当需要与外设传送数据时，CPU 告诉 DMA 控制器要传输数据的地址和长度，然后 CPU 就去做别的事情去了。DMA 控制器通过对总线的控制在主存和外设之间形成一条数据通道。然后它根据 CPU 给它的地址和长度监控数据的传送。数据传送完毕后，DMA 控制器会给 CPU 发送一个中断信号，通知 CPU 已经完成，这样 CPU 可以继续做后面相关的一些事情。

上面说的这三种数据传输方式的效率对于小型、微型计算机来说已经够用了。但是对于大、中型计算机来说，还远远不够。就拿性能最好的 DMA 方式来说。对于大、中型计算机来说，挂接的外设种类和数目非常多。为每台外设都配备一个 DMA 接口，成本非常高，而且管理这些 DMA 接口也要占用很多 CPU 时间。这些 DMA 接口还会在访问总线上与 CPU 发生矛盾，影响 CPU 的效率。因此在大、中型计算机上使用了一种称为 I/O 通道的方式。这种方式可以看作是进一步扩大 DMA 控制器的职权范围。在这里，通道是负责管理外设以及实现外设和主存之间交换信息的部件。通道有自己专用的指令，主要是输入输出指令。通道可以执行用这些指令编写的程序。它实际上可以看作是一个处理器，只不过它受 CPU 的控制，是从属于 CPU 的一个专用处理器。通过通道管理外设，可以进一步提高 CPU 的利用率。

对于大型机来说，大量的数据输入输出还是超出了通道的工作能力。因此，通道的职能被进一步扩大直到具有全部处理机的能力，也就是使用专门的处理机来处理输入输出事务。这个专用处理机被称为外围处理机或者 I/O 处理机。它独立于主机工作。I/O 处理机可能就是一台小型机或微机，专门负责以前由 I/O 通道完成的对外设的控制，而且还可以做和输入输出数据相关的码制变换、格式处理、数据检错纠错等工作。这样，主机就完全从输入输出工作中解脱出来，与输入输出系统并行工作了。

小提示

主机和外设的数据传输方式有程序直接控制方式、程序中断方式、直接存储器访问 DMA 方式。大型系统中会使用 I/O 通道或者 I/O 处理机来处理输入输出。

2.6 计算机系统结构

计算机系统结构是一个比较广泛的概念。在这里，我们局限在指那些用机器语言(包括汇编语言)写程序，即直接对计算机硬件编程的人看到的计算机的属性，包括功能与性能两个方面，如计算机的指令集及其实现、数据格式、存储器寻址技术以及前面提到过的存储层次结构，还有 I/O 机理等。随着技术的不断进步，计算机系统在性能和结构上都有了很大的变化。下面简单介绍目前人们在计算机系统结构方面做出的努力和成果。

2.6.1 计算机系统结构类型

尽管计算机系统的性能都有了很大的提高，但是我们常用的计算机所遵循的基本结构还是属于冯·诺依曼体系结构。也许你还记得，冯·诺依曼计算机根据程序计数器中的指令地址来取下一条要执行的指令。也就是指令的执行次序由程序计数器控制，所以这种结构也称为控制驱动结构。

人们为了提高计算机的性能，在对新系统结构的不断探索中，提出了一些不同于冯·诺依曼结构的新型结构。按照计算的驱动力来分，这些新型系统结构可主要分为数据驱动、需求驱动和模式匹配驱动三种。

在数据驱动的系统结构中，只要指令需要的操作数已经存在，那么指令就可以执行。也就是指令能否执行不再受程序计数器控制，而是以其执行的条件是否满足为前提。这样，所需数据没有相互关系的指令就可以同时执行了。所谓“数据流计算机”就是采用这种结构思想的一种代表。

所谓需求驱动是指只有需要某条指令的执行结果时才执行该指令。当然如果某条指令的执行需要其他指令的结果，那么它就会启动那些指令。有一种被称为归约机的计算机采用的就是这种结构。

而模式匹配驱动的系统结构则主要适合进行非数值运算的符号演算。例如，人工智能运算的主要对象就是符号而不是数值。所以人工智能计算机采用的就是基于模式匹配驱动的计算机系统结构。

这些新的系统结构听起来就很神奇，但它们都还处于研究和开发阶段，离普遍应用还有相当距离。但计算机技术总是在不断发展的，说不定哪一天一种新的结构就会取代现在的冯·诺依曼结构，我们需要保持关注，不断地更新自己的知识。

2.6.2 计算机性能的提高

计算机系统结构的发展目标就是在可用的工艺条件下，通过结构设计来提高计算机对于它所要执行的典型应用程序的性能，有时还需要考虑性价比。人们不断地改进或者重新设计计算机的系统结构，希望无论用什么技术实现，计算机都能比以前的性能好。

对计算机的设计者来说，快的执行速度一直是一个追求的目标。提高速度有两个途径：要么让每个操作的完成时间变短，要么同时进行多个操作。缩短每个操作的速度可以通过采用新材料和新技术来实现。比如计算机的部件从使用电子管器件转变到晶体管，又到现在的超大规模集成电路，就是技术和材料上的革新。这种办法属于材料科学领域的研究范围。

而第二种方法，同时进行多个操作，则是计算机系统研究者的课题。可以同时执行多个操作的特性被称为并行性。因此，第二种提高速度的方法，换句话说，也就是要提高计算机系统的并行性。并行性中“同时”的范围可有点儿广，它不仅指同一时刻，也指同一段时间内。为了区分这两种并行性，把同一时刻执行多个操作称为同时性。而多个操作在同一时间间隔内发生的性质称为并发性。提高计算机系统的并行性可以有很多种方法和手段。

前面提到过的流水技术就是一种提高并行性的手段。这种办法并没有增加硬件，而是根据每个部件的工作，把每个操作细分，让所有部件都没有闲置。流水的实质思想是调整任务的分配形式，不让任何车间任何人空闲。

当所有部件都调动起来之后，流水提高并行性的能力也就发挥到极点了。要想再进一步提高并行性，提高计算机的速度，就只能增加硬件了。也就是在没有一个人闲着的情况下，要想多出活儿，只能靠增加人手了。增加人手的方式也有很多。

第一种方式是只对工作量大的工序增加人手。对于计算机，就是增加关键部件，比如增加运算单元，而控制单元还保持一个。这些运算单元都执行同一个指令，但是各自处理的操作数不同，因而可以同时得出很多不同的结果。当然为了让这些运算单元能够同时工作，还需要有配套部件保证它们能同时访问到各自的数据。这种结构被称为单指令流多数据流(Single Instruction Multiple Data，SIMD)结构。这种计算机被称为并行计算机。显然这种计算机适合处理对大量数据进行相同操作的问题。

我们还可以扩大所增加硬件的范围，例如，增加成套的取指令部件、执行部件和写回部件。这样计算机就可以同时执行多条指令。这种方式就是前面提到的超标量技术。这好比扩大增派人手的范围，直接增加关键车间。

人手增派的范围还可以继续扩大，你可能已经想到了，那就是把所有车间加倍。让工厂里存在两套或者更多套一模一样的车间体系，这样它们可以同时工作。具体到计算机系统就是增加一个或多个完整的处理器。这些处理器在同一操作系统的控制下，共享同一主存、I/O 通道以及外设。这种计算机被称为多处理机系统，属于多指令流多数据流(Multiple Instruction Multiple Data，MIMD)结构。每个处理器都有自己的控制部件，执行各自的程序。

最后我们把这种资源的增加做得更彻底一些。这次干脆另外增加一些工厂。也就是直接再增加一些计算机，这样多台计算机一起工作，并行性显然会得到大大地提高。这种系统被称为多计算机系统。多计算机系统中的计算机相互独立。

前面我们介绍了目前常见的一些提高并行性的手段。这些技术的最终目标是提高计算机的速度。除了速度之外，人们还在努力让计算机更加聪明。

附文：

中国计算机产业发展大事记

摘自 http://itmus.chinalabs.com/206.html

新华社记者张晓松　王立彬　李佳路

1956年，夏培肃完成了第一台电子计算机运算器和控制器的设计工作，同时编写了中国第一本电子计算机原理讲义。

1957年，哈尔滨工业大学研制成功中国第一台模拟式电子计算机。

1958年，中国第一台计算机——103型通用数字电子计算机研制成功，运行速度每秒1500次。

1959年，中国研制成功104型电子计算机，运算速度每秒1万次。

1960年，中国第一台大型通用电子计算机——107型通用电子数字计算机研制成功。

1963年，中国第一台大型晶体管电子计算机——109机研制成功。

1964年，441B全晶体管计算机研制成功。

1965年，中国第一台百万次集成电路计算机DJS-Ⅱ型操作系统编制完成。

1967年，新型晶体管大型通用数字计算机诞生。

1969年，北京大学承接研制百万次集成电路数字电子计算机——150机。

1970年，中国第一台具有多道程序分时操作系统和标准汇编语言的计算机——441B-Ⅲ型全晶体管计算机研制成功。

1972年，每秒运算11万次的大型集成电路通用数字电子计算机研制成功。

1973年，中国第一台百万次集成电路电子计算机研制成功。

1974年，DJS-130、131、132、135、140、152、153等13个机型先后研制成功。

1976年，DJS-183、184、185、186、1804机研制成功。

1977年，中国第一台微型计算机DJS-050机研制成功。

1979年，中国研制成功每秒运算500万次的集成电路计算机——HDS-9，王选用中国第一台激光照排机排出样书。

1981年，中国研制成功的260机平均运算速度达到每秒100万次。

1983年，“银河Ⅰ号”巨型计算机研制成功，运算速度达每秒1亿次。

1984年，联想集团的前身——新技术发展公司成立，中国出现第一次微机热。

1985年，华光Ⅱ型汉字激光照排系统投入生产性使用。

1986年，中华学习机投入生产。

1987年，第一台国产的286微机——长城286正式推出。

1988年，第一台国产386微机——长城386推出，中国发现首例计算机病毒。

1990年，中国首台高智能计算机——EST/IS4260智能工作站诞生，长城486计算机问世。

1991年，新华社、科技日报、经济日报正式启用汉字激光照排系统。

1992年，中国最大的汉字字符集——6万电脑汉字字库正式建立。

1993年，中国第一台10亿次巨型银河计算机Ⅱ型通过鉴定。

1994年，银河计算机Ⅱ型在国家气象局投入正式运行，用于天气中期预报。

1995年，曙光1000大型机通过鉴定，其峰值运算速度可达每秒25亿次。

1996年，国产联想电脑在国内微机市场销售量第一。

1997年，银河-Ⅲ并行巨型计算机研制成功。

1998年，中国微机销量达408万台，国产占有率高达71.9%。

1999年，银河四代巨型机研制成功。

2000年，我国自行研制成功高性能计算机“神威Ⅰ”，其主要技术指标和性能达到国际先进水平。我国成为继美国、日本之后，世界上第三个具备研制高性能计算机能力的国家。(新华社北京2000年8月17日电)

- 2000年1月28日，中科院计算所研制的863项目曙光2000-Ⅱ超级服务器通过鉴定，其峰值运算速度达到每秒1100亿次，机群操作系统等技术进入国际领先行列。
- 2000年5月，大唐公司提出的TD-SCDMA获得国际电信联盟的批准，成为第一个由中国提出的3G移动通信国际标准。
- 2000年6月15日，中科院软件所在UltraSPARC 64位平台上开发成功第一个64位中文Linux操作系统——Penguin 64。这是当时起点最高的直接针对具体硬件平台开发的中文Linux操作系统。
- 2000年6月24日，国务院发布了《鼓励软件产业和集成电路产业发展的若干政策》的文件(国发{2000}18号文件)。此举对即将加入WTO的中国软件产业和集成电路产业的发展意义重大。
- 2000年8月21日至25日，国际信息处理联合会(IFIP)主办的第16届世界计算机大会(WCC2000)在中国北京召开，江泽民总书记出席开幕式并发表重要讲话。
- 2000年12月28日，全国人民代表大会常务委员会通过《关于维护互联网安全的决定》。这是我国最高立法机构首次针对互联网制定的立法文件。
- 2001年，中国IT产业产值达到1.35万亿元(人民币)，规模直逼排名第二的日本。此时，中国IT产业产值及产品贸易额，均占世界总额的5%以上；其中家电、通信终端、计算机等一些整机产品的产量已跃居世界前列，程控交换机、手机、计算机显示器、彩电、彩管、激光视盘放像机、收录机等产品的产量和出口量排名世界第一。
- 2001年6月1日，由海关总署牵头，12个相关部委联合开发的口岸电子执法系统在全国各口岸全面运行。
- 2001年7月10日，中芯微系统公司宣布研制成功第一块32位CPU芯片“方舟-1”，其主频为200MHz。
- 2001年7月12日，中国移动通信集团宣布在全国25个城市开通GPRS业务。此举标志着中国无线通信进入2.5G时代。
- 2001年10月，由大唐电信与国防科技大学共同研制的、具有自主知识产权的863中国高速信息示范网核心路由器ISR系列正式推向市场。
- 2001年12月11日，国务院批准电信体制改革方案，中国电信宣布一分为二。中国电信北方10省市的资源归重组后的中国网通集团公司，其余的资源归新的中国电

信集团公司。加上中国移动、中国联通、卫星通信以及中国铁通等，中国电信市场垄断局面被彻底打破。

- 2001 年 12 月 20 日，国务院公布《计算机软件保护条例》。
- 2001 年 12 月 22 日，中国联通公司 CDMA 移动通信网一期工程建成，并于 12 月 31 日开通运营。从地域和人口的覆盖来看，中国联通的 CDMA 网是世界上最大的 CDMA 网络。
- 2002 年 4 月，境外权威调查机构(Nielsen/NetRatings)的最新研究表明，中国内地家庭上网人数达 5660 万，超过日本，居世界第二，仅次于美国。
- 2002 年 7 月 3 日，国家信息化领导小组举行第二次会议。会议通过了《国民经济和社会信息化专项规划》和《关于我国电子政务建设的指导意见》，为各级政府的电子政务建设催生了一个大市场。
- 2002 年 9 月 28 日，中科院计算所宣布中国第一个可以批量投产的通用 CPU“龙芯 1 号”芯片研制成功。其指令系统与国际主流系统 MIPS 兼容，定点字长 32 位，浮点字长 64 位，最高主频可达 266MHz。此芯片的逻辑设计与版图设计具有完全自主的知识产权。采用该 CPU 的曙光“龙腾”服务器同时发布。
- 2002 年 11 月 8 日，党的第十六次全国代表大会上提出：“以信息化带动工业化，以工业化促进信息化”，为我国制定了一条新型的工业化发展思路。
- 2002 年 11 月 25 日，高性能嵌入式 32 位微处理器神威Ⅰ号在上海复旦微电子公司研制成功，并一次流片成功。
- 2003 年 4 月 9 日，由苏州国芯、南京熊猫、中芯国际、上海宏力、上海贝岭、杭州士兰、北京国家集成电路产业化基地、北京大学、清华大学等 61 家集成电路企业机构组成的“C * Core(中国芯)产业联盟”在南京宣告成立，谋求合力打造中国集成电路完整产业链。
- 2003 年 7 月 22 日，温家宝总理在国家信息化领导小组第三次会议上提出信息化建设四项重点：一是大力推广应用信息技术；二是加强信息资源开发利用；三是抓紧推行电子政务；四是切实加强信息安全保障工作。
- 2003 年 12 月 9 日，联想承担的国家网格主节点“深腾 6800”超级计算机正式研制成功，其实际运算速度达到每秒 4.183 万亿次，全球排名第 14 位，运行效率 78.5%。
- 2003 年 12 月 28 日，“中国芯工程”成果汇报会在人民大会堂举行，我国“星光中国芯”工程开发设计出 5 代数字多媒体芯片，在国际市场上以超过 40%的市场份额占领了计算机图像输入芯片世界第一的位置。
- 2004 年 3 月 24 日，在国务院常务会议上，《中华人民共和国电子签名法(草案)》获得原则通过，这标志着我国电子业务渐入法制轨道。
- 2004 年 6 月 21 日，美国能源部劳伦斯·伯克利国家实验室公布了最新的全球计算机 500 强名单，曙光计算机公司研制的超级计算机“曙光 4000A”排名第十，运算速度达每秒 8.061 万亿次。

复习题

1. 计算机系统的硬件从功能上可以分为哪些部分？它们各自的功能是什么？
2. 请为计算机的主要部件画一个框图，包括其中的互联关系。
3. 什么叫做存储器的编址？
4. 程序计数器是什么？它的功能是什么？
5. 什么是总线？为什么要使用总线？什么是总线宽度？
6. 什么是系统总线？系统总线又可以分成哪几类？
7. 存储系统为什么采用层次结构？请画出该层次结构图。
8. 考察存储器主要参考哪些指标？
9. 现在常见的内存条类型是什么？使用什么插槽？
10. 计算机系统的启动程序存储在什么地方？
11. 什么是 BIOS？CMOS 和 BIOS 有什么关系？
12. 什么是 CPU 的主频？与 CPU 的速度有什么关系？
13. 什么是机器字长？和地址总线以及数据总线宽度有什么关系？
14. 什么是 I/O 接口？它的作用是什么？
15. 硬盘是依据什么原理存储信息的？
16. 温彻斯特硬盘的特点是什么？
17. 什么是硬盘的记录面、磁道、柱面、扇区？
18. 硬盘的主要参数有哪些？
19. 硬盘的数据传输率指的是什么？
20. 硬盘常用的接口类型有哪些？
21. 使用硬盘要注意哪些事项？为什么？
22. 光盘是依据什么原理存储信息的？
23. 光盘的光道和硬盘的磁道有什么区别？
24. 为什么有人把 U 盘叫做闪盘？
25. 彩色显示器是依据什么原理成像的？
26. 显示器的分辨率和场频指的是什么？
27. 显卡的作用是什么？
28. 打印机的主要参数有哪些？
29. 光电鼠标是怎样感知自己的位置移动的？
30. 什么叫编码键盘？什么叫非编码键盘？
31. 驱动程序的作用是什么？为什么必须为外设安装驱动程序？
32. 外设与主机传输数据通过几种方式？它们具体的做法是怎样？

练习题

1. 你能说出计算机系统还有哪些外设吗？

2. 一个计算机的机器字长为 16 位，地址总线宽度为 16 位，如果以 16 位为单位对内存进行编址，那么这台计算机可以支持多大的内存容量？

3. 一台计算机的高速缓存的存取周期为 60ns，而内存的平均存储周期为 1μs，缓存的命中率为 0.6。假定指令的执行时间与存储器的平均存储周期成正比。那么通过使用缓存可以把指令的执行速度提高多少倍？如果命中率达到 0.75 呢？

4. 请向历史系的同学介绍计算机的工作过程。

5. 若计算机的主频为 4GHz，主要类型指令的平均执行时间和使用频率为，存取指令：0.6μs，35%；加、减、比较、转移指令：0.8μs，50%；乘、除指令：10μs，5%；其他指令：1.4μs，10%，则计算机的速度为多少 MIPS？

6. 请查阅资料解释液晶显示器的成像原理。

7. 一个硬盘，每个盘片的直径为18英寸，最内圈磁道半径为4英寸，最外圈磁道半径为8英寸，磁道密度为100道/英寸，最内圈记录密度为1000位/英寸，所有磁道上的信息量相同，盘片组共有20个记录面，转速为7200转/分，道间移动时间为0.2ms。试计算硬盘的容量、数据传输率和平均存取时间。

8. 假设某设备向CPU传送信息的最高频率为30K次/秒，而相应的中断处理程序的执行时间为40μs，问该外设是否可以用程序中断方式与主机交换信息？

讨论

1. 请说出几种常见的总线标准。目前使用的总线有哪些？
2. 有哪些办法可以提高计算机的速度？

实验

1. 请通过操作系统检查实验室某台计算机的CPU型号、主频，并到网络上查询一下这种CPU的运算速度。
2. 请打开机箱，在里面找到CPU、主板、内存、BIOS芯片，并向老师或者管理员询问其他部件的名称。如果可能自己将计算机拆卸后，再重新组装上。
3. 查看你自己的计算机上显卡的品牌以及型号、驱动程序。将其他计算机上的显卡换到你的计算机上，看看发生了什么？试着安装该显卡的驱动程序。

第3章　信息表示

从前面的介绍中可知：计算机元器件的基本物理特性使得它们容易产生两种明显不同的状态（电路的开与关、磁极的南与北、材料表面的凸与凹等），通常我们就用0和1来抽象表示它们。那么世界上纷繁多样的信息是怎样在计算机中用0和1表示出来的呢？这一章就介绍与此有关的知识。

计算机中只能存储0和1，如果要用它们来表示数值，则需要采用二进制。当然，计算机中表示的信息不一定都是数值。在表示非数值信息时，所谓“进制”的概念是没有意义的，但人们习惯上还是笼统地说“计算机是用二进制来表示信息的”。那么到底什么叫二进制数呢？和我们常用的数的形式之间有什么关系呢？针对这些问题，本章将首先介绍进位制的有关内容。在弄清楚进位制的问题之后，再说明计算机中是怎样用二进制来表示信息的，主要介绍数字和各种字符，其他诸如多媒体信息的表示问题不属于本书的范畴。最后还要介绍指令和程序。

3.1　进位制及其转换

采用进位制来表示数是人类的一个重要的发明。我们在学算算术时，都知道一个道理，“满十进一”。数数的时候数到了十就进一位。这种计数方法叫做十进制。我们把规定计数进位的规则叫做进位制。习惯上用进位制规定的进位的值来命名计数方法。比如“满二进一”的计数方法，就叫做二进制，“满八进一”就叫做八进制。

进位制就好像一种打包方法。假设有一大堆苹果。现在要把它们装到箱子里。把苹果分成10个一组，装在一个个小箱子里，把它们称为1号箱。如果最后一组不到10个，那就把它们放在外边。都装完之后，再把那些1号箱分成10个一组，每组再装到大一点儿的2号箱子里。同样如果最后一组不够10个，就把它们剩在那里，不再装入大一号的箱子。然后再把每10个2号箱装到更大的3号箱中。就这样继续下去，一直到不存在10个一样的箱子。最后清点每种箱子的个数，并把这些数目按照箱子从大到小的顺序写在一起。比如最大的4号箱有6个，没装进4号箱的3号箱有7个，没有没装进3号箱的2号箱，没装进2号箱的1号箱有9个，外面还剩有3个苹果。这样你就得出一个数字67 093，这个数正好是这些苹果的个数，应该说是苹果个数的十进制表示。因为只要有10个一样的箱子，我们就把它装到大一号的箱子中，其实也就是“满十进一”。

如果你不是按照10个一组来装箱，而是按照2个一组来装箱，即只要有两个一样的箱子，就把它装到大一号的箱子中。这样最后清点箱子形成的数字就是苹果个数的二进制表示。以此类推，以8为单位装箱就可以得到这个数目的八进制表示。

日常生活中我们用到较多的计数方法是十进制。有时也会用到其他进制的计数方法。比如，在计时上，采用的就是60进位制，60秒为一分钟，60分钟为一个小时。

下面介绍几种常见的与计算机有关的进位制。

3.1.1 进位制

1. 十进制

十进制是我们最熟悉的进位制。下面通过十进制来了解一些与进位制有关的属性，仍然使用苹果装箱的例子。使用十进制，也就是说把苹果按照 10 个一组进行装箱。如果最后一组苹果不够装满一箱，那这几个苹果就不装箱。那么剩下的苹果数可能是 1～9 的任何一种情况，或者一个不剩。为了表示没有装到 1 号箱中的苹果个数，我们一共需要 10 个符号来区分从“没有”到“九”这 10 种不同的情况。我们用 0～9 这 10 个阿拉伯数字来完成这个任务。这 10 个阿拉伯数字叫做十进制的数码。每个进位制的数码的个数叫做该进位制的基数。比如十进制有 0～9 一共 10 个数码，那么十进制的基数就是 10。十进制数可用英文字母 D 表示，是对应英文单词 decimal 的首字母。

小提示

规定计数进位的规则叫做进位制。
习惯上用进位制规定的进位的值来命名计数方法。
每个进位制所使用的数码个数叫做该进位制的基数。

假定按照 10 个一组为单位进行苹果装箱，最后清点得出的数字是 6789。也就是说最后我们看到 6 个 3 号箱、7 个 2 号箱、8 个 1 号箱，以及 9 个苹果。这也就意味着苹果的个数是 6789。为什么呢？因为每个 1 号箱中可以装 10 个苹果，每个 2 号箱中有 10 个 1 号箱，所以每个 2 号箱中有 10×10 个苹果，每个 3 号箱中有 $10\times10\times10$ 个苹果。所以一共有 $6\times10^3+7\times10^2+8\times10^1+9=$ 6789 个苹果。我们把 6789 这个数的右边最后一位记作位置 1、第二位记作位置 2、第 n 位记作位置 n。可以看出，6789 这个数每个位置和一个特殊的常数有关。例如，位置 4 对应 10^3、位置 2 对应 10、位置 n 对应 10^{n-1}。我们把这个常数叫做该位置的位权。例如，4 号位置的位权是 10^3。按照我们的记法，不同箱子的个数在数字中的位置是固定的。比如 3 号箱的个数出现在位置 4。其实每位上的位权就是对应箱子中苹果的个数。比如，4 号位置的位权就是 3 号箱中的苹果个数 10^3。通常位权用十进制数表示。位权是和数字的进位制有关系的。现在 4 号位置的位权为 10^3 是因为采用的是十进制。也就是说，苹果是按照 10 个一组装箱的，所以 3 号箱中的苹果数是 10^3。如果苹果是按照 2 个一组装箱的，那么 3 号箱中苹果的个数应该是 2^3。你算算对不对？所以，此时 4 号位的位权应该是 2^3，这是因为采用的是二进制计法。

概括一下，用 q 进制表示的数，第 n 位（最右边一位为第 1 位）上的位权是 q^{n-1}。例如，二进制数的第 3 位上的位权为 2^{3-1}，十六进制数的第 2 位上的位权为 16^1。

我们还要说一说小数的问题。看下面的例子：

$$0.1234 = 1\times10^{-1}+2\times10^{-2}+3\times10^{-3}+4\times10^{-4}$$

因此，我们规定小数点右边第一位的位权为 10^{-1}，第二位为 10^{-2}，以此类推。

现在我们把整数和小数的情况概括在一起：

对于用 q 进制表示的数，其数码的顺序定义如下：小数点左边第一位定义为第 1 位，向左第二位定义为第 2 位，以此类推；小数点右边第 1 位定义为第 0 位，右边第二位定义为第

－1 位，以此类推，那么，第 n 位上的位权就是 q^{n-1}。

例如，二进制数小数点后第 3 位上的位权为 2^{-3}，十六进制数小数点后第 2 位上的位权为 16^{-2}。

从上面的分析，可以看出每个数字表示的实际数值（用十进制表示）应该是经过下列运算得出的值：把每个位置上的数码乘以该位置上的位权，然后再把这些乘积求和。比如，

$$6789 = 6\times10^3+7\times10^2+8\times10^1+9= 6789$$

因为现在用到的数字 6789 和它所表示的数值 6789 都是十进制表示，所以当然一样了。如果不是十进制，这个数看起来就会不一样了。

小提示

对于用 q 进制表示的数，其数码的顺序定义如下：小数点左边第一位定义为第 1 位，向左第二位定义为第 2 位，以此类推；小数点右边第 1 位定义为第 0 位，右边第二位定义为第－1 位，以此类推，那么，第 n 位上的位权是 q^{n-1}。

2. 二进制

二进制就是“满二进一”。用苹果装箱的例子说，就是按照 2 个一组为单位进行装箱。装到最后，未装箱的苹果和每种箱子要么不剩，要么只剩一个。如果剩两个，就可以装一个大一号的箱子了。所以只需要两个数码 0 和 1 就够了。因此，二进制的基数为 2。

现在假定只有 2 个苹果，那么我们可以把它们装入到一个 1 号箱中。清点箱子的个数，并按照箱子大小顺序写出来就是 10。别忘了，最后一位应该是没装箱的苹果，如果没剩，应该用 0 表示。所以，十进制的数 2_{10} 在二进制中的表示就是 10_2。为了防止把你弄晕了，我们在每个数字的后面用一个十进制数作为下标指出这个数的进位制。我们也用符号 B（Binary）来表示二进制数。

那么 3_{10} 在二进制中的表示是什么呢？实际也就是比刚才又多了一个苹果。多这一个苹果也不够装一箱，所以只能放在外边。现在重新清点箱子和苹果的个数：1 个 1 号箱和 1 个没装箱的苹果，得出数字 11。也就是说 3_{10} 的二进制表示是 11_2。

现在假设中关村的摊主说这个 MP3 可以少要你 1011_2 元，你知道是多少钱吗？当然如果他这么跟你说，显然他是卖不成这个 MP3 了。现在来算算这个数字换成常用的十进制数到底是多少。

现在的情况就是按照 2 个一组把苹果装箱，最后清点得出的数字是 1011。算算一共有多少个苹果（用十进制表示）。

按照装箱和清点规则，1011 这个数字表示最后看到 1 个 3 号箱、0 个 2 号箱、1 个 1 号箱和 1 个苹果。按照 2 个一组装箱，现在每个 1 号箱中应该有 2 个苹果；每个 2 号箱中有 2 个 1 号箱，所以有 2×2 个苹果；每个 3 号箱中有 2 个 2 号箱，所以有 2×2×2 个苹果。所以一共有

$$1\times2^3+0\times2^2+1\times2^1+1\times2^0= 11(个)$$

其实我们可以换另外一种说法，前面我们已经得出：每个数字表示的数值（用十进制表示）应该是：每个位置上的数码乘以该位置上的位权再求和得出的值。

因为是二进制数，也就是按照 2 个一组进行苹果装箱，那么二进制数第 4 位上的位权就

应该是3号箱子中的苹果数2^3，第n位上的位权应该是$n-1$号箱子中的苹果数2^{n-1}。那么二进制数1011_2表示的数值(用十进制表示)应该是，每个位置上的数码乘以该位置上的位权再求和：

$$1011_2 = 1\times2^3+0\times2^2+1\times2^1+1\times2^0 = 11_{10}$$

同样可以算出二进制数1011_2所表示的数值对应的十进制数就是11_{10}。

3. 八进制

八进制，顾名思义就是“满八进一”。为什么要讲八进制呢？因为八进制和后面马上要说到的十六进制与二进制有很方便的转换关系。所以在计算机领域这两种进位制的使用非常广泛。

计算机里面存储的是二进制数，但是因为二进制数每位只能表示2个数值，所以要表示一个较大的数，就会用很多位。这在机器中问题不大。但是书写起来就很不方便。比如，6789_{10}写成二进制数就是1101010000101_2。够长了吧？(噢，你要仔细检查一下，看看我有没有写错。我是用计算器算出来的，但是这么长串的0和1，抄过来的时候搞得我有点儿头晕)。所以，尽管计算机中实质上用的是二进制，但我们平时书写时常将它们“浓缩”成八进制或十六进制。

八进制通常用字母O(Octal)表示。八进制需要8个数码符号，它们就是0～7。记住，八进制数中肯定不会出现数码8。6789_{10}写成八进制数应该是15205_8。因为八进制也就是按照8个一组进行苹果装箱，所以八进制数中位置n的位权为8^{n-1}。八进制数15205_8表示的十进制数值可以用下面这个式子计算：

$$15205_8 = 1\times8^4+5\times8^3+2\times8^2+0\times8^1+5\times8^0$$

结果应该是6789_{10}，不然就不对了。

4. 十六进制

十六进制也就是“逢十六进一”，那么就需要16个不同的符号。人们用0～9这10个阿拉伯数字再加上字母A～F来表示0～15这16个数值。6789_{10}写成十六进制数就是$1A85_{16}$。是不是感觉有点儿怪怪的。这是个习惯问题，过些天你就不感觉奇怪了，就像中国人原来用一、二、三计数，刚开始用阿拉伯数字1,2,3也会有些不习惯。十六进制也就是按照16个一组进行苹果装箱，所以十六进制数中位置n的位权为16^{n-1}。$1A85_{16}$表示的十进制数值就是$1\times16^3+10\times16^2+8\times16^1+5\times16^0$。通常用H(Hexadecimal)表示十六进制。

3.1.2 数的进制转换

有时候需要察看计算机存储器的内容，尤其是寄存器的内容。例如，程序出错时，有些调试软件会向你显示相关寄存器的内容。不过这些内容不是以十进制的形式表示的，而是二进制。说得更精确一些，多数软件会以十六进制的形式显示寄存器的内容。如果想了解寄存器中的数值到底是多少，就要掌握把这些数转换成十进制表示的方法。或许有时候你还需要知道一个数在计算机中到底应该以什么样的一个0/1串表示的，那么就需要你把它们转换成二进制表示。也就是说，你要掌握数的不同进制表示之间转换的方法。需要提醒的是，这里讨论的只是同一个数的不同表示方法，所谓转换，并不是将一个数变成了另一个数。

实施数制的转换，最简单的办法就是像我一样使用计算器，又快又准确。如果没有计算器，也可以很容易地在网上搜到做这些转换的软件。不过，在碰巧没有计算器，没有软件或

网络时，那只能自力更生，像下面这样做了。好在这个事情并不难。下面就介绍这些进制表示之间的转换。首先看看其他进制表示的数怎样转换成十进制表示。

1. 转换成十进制表示

把二进制、八进制和十六进制数转换成十进制表示非常简单。在前面其实已经做过了。只要把要转换数的每位上的数码乘以该位置上的位权，然后把这些乘积加在一起即可。下面我们通过实例分别看看这几种进制表示的转换。

1）二进制转换成十进制

例如：将二进制数 11111111_2 和 0.1101011_2 转换成十进制表示。

因为二进制数第 n 位上的权为 2^{n-1}，所以，

$$11111111_2 = 1\times2^7+1\times2^6+1\times2^5+1\times2^4+1\times2^3+1\times2^2+1\times2^1+1\times2^0$$
$$= 255_{10}$$

$$0.1101_2 = 1\times2^{-1}+1\times2^{-2}+0\times2^{-3}+1\times2^{-4} = 0.8125_{10}$$

2）八进制转换成十进制

例如：请将八进制数 377_8 和 0.377_8 转换成十进制表示。

因为八进制数第 n 位上的权为 8^{n-1}，所以，

$$377_8 = 3\times8^2+7\times8^1+7\times8^0 = 255_{10}$$

$$0.377_8 = 3\times8^{-1}+7\times8^{-2}+7\times8^{-3} = 0.498046875_{10}$$

3）十六进制转换成十进制

例如：将十六进制数 FF_{16} 和 $F.F_{16}$ 转换成十进制表示。

因为十六进制数第 n 位上的权为 16^{n-1}，所以，

$FF_{16} = 15\times16+15\times16^0 = 255_{10}$（还记得吗？十六进制数码 F 表示十进制数 15）

$F.F_{16} = 15\times16^0+15\times16^{-1} = 15.9375_{10}$

清楚了这几种进制数到十进制数的转换，接下来看看这些不同进制的数如果要转换成二进制表示应该怎样做。

小提示

把二进制、八进制和十六进制等进制的数转换成十进制表示，只要把要转换数的每位上的数码乘以该位置上的位权，然后把这些乘积相加。

2. 转换成二进制表示

首先来看看日常使用的十进制数怎样转换成二进制表示。

1）十进制转换成二进制

我们还是用苹果装箱的事情来说这个转换吧。假设我们有 89_{10} 个苹果，平时我们习惯用十进制表示数目，所以这个 89 是十进制数。现在我们要把这些苹果装箱。装箱和清点规则和前面一样。只不过现在要得出二进制表示，所以装箱单位是 2，即只要有两个一样的箱子就要装入一个大一号的箱子，也就是“满二进一”。这样最后清点出来各种箱子的个数按顺序放在一起形成的数就是 89_{10} 对应的二进制表示。

对于上面这个规则我们都没有异议。那么现在我们的转换问题就变成求各种箱子的个数问题。下面我们就来算算各种箱子的个数。我们可以有两种算法。一种是从计算最大箱

子的个数开始，然后再依次算小一点儿箱子的个数；另一种方式则反过来，先从计算小箱子的个数开始，一直算到最大号的箱子。两种算法得出的结果应该一样。

先来说第一种方法：从最大号箱子的个数算起。

这种办法首先要判断最大的箱子是几号。我们已经知道在以 2 为单位装箱的情况下，n 号箱中苹果的个数是 2^n。也就是每个箱子中的苹果数正好是 2 的 n 次幂。要判断出使用的最大号箱子，就是要看看刚好比 89_{10} 小一点儿的 2 的幂是多少。这种办法需要你对 2 的幂很熟悉。最后我们发现 $2^6<89_{10}<2^7$，所以最大的箱子应该是 6 号箱。装满一个 6 号箱之后，还剩 $89-2^6=25$ 个苹果（我们还是先把十进制数后面的标识省掉吧，以后没有特意声明且没有下标的数都认为是十进制数）。接下来要看看剩下这些苹果能装满的最大箱是几号？因为 $2^4<25<2^5$，所以接下来用到的应该是 4 号箱。现在还剩 $25-2^4=9$ 个苹果。我们继续为这 9 个苹果找最大的箱子，以此类推。最后我们看到的箱子是：1 个 6 号箱，1 个 4 号箱，1 个 3 号箱，还有一个苹果。按照箱子大小顺序把这些个数放在一起，中间没有用到的箱子用 0 表示，最后得到的数字串是 1011001。这个数字串就是 89_{10} 的二进制表示。

这种算法被称为减权定位法。它的较规范的说法是，将要转换的十进制数和相近的权值比较，从中减去刚好比该数小的权值，并确定与该权值对应的数位上的数码为 1。以此类推，直到余数是 0 为止，未被记 1 的数位上的系数均记为 0，即可得到相应的二进制数。

使用减权定位法需要记住每位上的权。相对于此，第二种转换办法显得更容易一些。下面就来介绍这种办法。这种办法就是从小号箱的个数算起。首先我们要算这些苹果能够装满多少个 1 号箱，然后剩多少苹果。剩下的苹果数就是二进制数的最后一位。因为每个 1 号箱中能装两个苹果，89 除以 2 等于 44 余 1，所以可以得出要用到 44 个 1 号箱，外面要剩 1 个苹果。那么外面最后要剩几个 1 号箱呢？还要把这些 1 号箱装到 2 号箱中之后才知道。每个 2 号箱装 2 个 1 号箱，因为 44/2＝22，所以这些 1 号箱正好全装入 22 个 2 号箱中，一个没剩。因此，最后能看到的 1 号箱的个数是 0。我们继续将 2 号箱装入 3 号箱，每个 3 号箱装 2 个 2 号箱，因为 22/2＝11，所以这些 2 号箱正好全装入 11 个 3 号箱中，一个没剩。因此，最后能看到的 2 号箱的个数是 0。以此类推，直到箱子不够再装入大一号的箱子。最后我们清点箱子和苹果的个数，把这些个数按照箱子从大到小的顺序形成一个数字串。这个数字串就是 89 的二进制表示。当然，结果肯定与第一种算法相同。

这种算法其实就是：反复地把十进制数除以 2 直到商为 0，最后形成的二进制数就是从最后一次除法的余数到第一次余数的一个排列。这种算法被称为除基取余法。概括起来就是：除（以）基（数）取余（数），先余为低，后余为高。

我们一般采用下面的竖式来表示上面的过程。

例如：求 89_{10} 的二进制表示。

$$
\begin{array}{r|l|ll}
2 & 89 & 1 & p_1 \\
2 & 44 & 0 & p_2 \\
2 & 22 & 0 & p_3 \\
2 & 11 & 1 & p_4 \\
2 & 5 & 1 & p_5 \\
2 & 2 & 0 & p_6 \\
2 & 1 & 1 & p_7 \\
 & 0 & &
\end{array}
$$

则 $89_{10}= p_7\ p_6\ p_5\ p_4\ p_3\ p_2\ p_1= 1011001_2$。

小数的转换也有两种办法，一种是减权定位法。与前面整数的情况一样。第二种办法略有不同，小数的转换不再采用除法，而是采用乘法，称为乘基取整法。具体的做法是：乘(以)基取整，先整为高，后整为低。也就是用十进制小数部分不断地乘以 2，直到乘积的小数部分为 0，然后把每次乘积的整数部分按照先后顺序排列在一起。我们来看一个例子。

例如：求 0.8125_{10}的二进制表示。

$$
\begin{array}{rll}
 & 0.8125 & \\
\times & 2 & \\
\hline
 & 1.6250 & \text{整数 } 1 \quad p_1 \\
 & 0.6250 & \\
\times & 2 & \\
\hline
 & 1.2500 & \text{整数 } 1 \quad p_2 \\
 & 0.2500 & \\
\times & 2 & \\
\hline
 & 0.5000 & \text{整数 } 0 \quad p_3 \\
 & 0.5000 & \\
\times & 2 & \\
\hline
 & 1.0000 & \text{整数 } 1 \quad p_4
\end{array}
$$

则 $0.8125_{10}= p_1\ p_2\ p_3\ p_4= 0.1101_2$。

如果要转换的十进制数既有整数部分又有小数部分，那么要将这两部分分开，分别用除基取余法和乘基取整法求得整数和小数部分，最后再把它们合在一起。

2) 八进制转换成二进制

八进制数和十六进制数转换成二进制数则比较简单。因为 $2^3=8$，所以三位二进制数正好等于一位八进制数。我们先看看八进制的 8 个数码对应的二进制数。

0	1	2	3	4	5	6	7
000	001	010	111	100	101	110	111

把八进制数转换成二进制数时，直接把这些数码对换成相应的三位二进制数码就可以了。例如：

$$765_8=111\ 110\ 101_2$$

$$0.765_8=0.111\ 110\ 101_2$$

3) 十六进制数转换成二进制数

因为 $2^4=16$，所以一位十六进制数正好等于 4 位二进制数，它们的关系也是一一对应的。

16 个十六进制数码对应的二进制数码如下：

0	1	2	3	4	5	6	7
0000	0001	0010	0111	0100	0101	0110	0111
8	9	A	B	C	D	E	F
1000	1001	1010	1011	1100	1101	1110	1111

把十六进制数转换成二进制数非常简单，只要把每位上的十六进制数码替换成4位二进制数码即可。如：

$$A13F_{16}=1010\ 0001\ 0011\ 1111_2$$

$$0.A13F_{16}=0.1010\ 0001\ 0011\ 1111_2$$

当然和八进制转换成二进制数一样，这需要你熟记这些数码的对应关系。因为十六进制数和二进制数有这样简单的对应关系，而且十六进制数比较简短，比二进制数更容易读写（二进制数那么长的0/1串，人看了不晕才怪呢，除非你是计算机），所以，平时都用十六进制计数法描述计算机中的二进制数。例如，需要显示计算机中寄存器的内容时，通常都是以十六进制表示。

小提示

十进制数转换成二进制数有减权定位法和除基取余法两种方法。

八进制数转换成二进制表示，只要把每位上的八进制数码替换成相应的3位二进制数码即可。

十六进制数转换成二进制表示，只要把每位上的十六进制数码替换成相应的4位二进制数码即可。

3. 转换成八进制表示

有了前面的铺垫，现在你对到八进制数的转换已经能够举一反三，找到一些窍门了吧？

1）十进制数转换成八进制表示

用什么方法？对了，除基取余法，只不过现在的基数是8而已。

例如，求89_{10}的八进制表示。

$$\begin{array}{r|l|ll} 8 & 89 & 1 & p_1 \\ 8 & 11 & 3 & p_2 \\ 8 & 1 & 1 & p_3 \\ & 0 & & \end{array}$$

所以，$89_{10}=p_3\ p_2\ p_1=131_8$。

请把这个八进制数转换成二进制表示，看看和前面直接从89_{10}转换成的二进制数是否相同？

2）二进制数转换成八进制表示

把二进制整数从最后一位开始分成三位一组，然后替换成相应的八进制数码就可以了。例如，求1010000100111111_2的八进制表示。转换如下：

$$1010000100111111_2=1\ 010\ 000\ 100\ 111\ 111_2=120477_8$$

求0.1010000100111111_2的八进制表示。

对于小数要从小数点后开始三位一组来分，最后的一组如果不够三位，则要补0。转换如下：

$$0.1010000100111111_2=0.101\ 000\ 010\ 011\ 111\ 100_2=0.502374_8$$

小提示

二进制整数转换成八进制表示，把二进制整数从最后一位开始向前分成三位一组，然后把每组替换成相应的八进制数码即可。

二进制小数则从小数点后第一位开始向后分成三位一组，如果最后一组不足三位，则用 0 补齐，然后把每组替换成相应的八进制数码即可。

3）十六进制数转换成八进制表示

十六进制数转换成八进制表示，最简单的办法是把它先转换成二进制表示，然后再转换成八进制表示。例如：

求 $A13F_{16}$ 的八进制表示。

$$A13F_{16} = 1010\ 0001\ 0011\ 1111_2 = 1\ 010\ 000\ 100\ 111\ 111_2 = 120477_8$$

求 $0.A13F_{16}$ 的八进制表示。

$$0.A13F_{16} = 0.1010\ 0001\ 0011\ 1111_2 = 0.101\ 000\ 010\ 011\ 111\ 100_2 = 0.502374_8$$

4. 转换成十六进制表示

1）十进制数转换成十六进制表示

除基取余法，基数为 16。

例如，求 89_{10} 的十六进制表示。

16	89	9	p_1
16	5	5	p_2
	0		

所以，$89_{10} = p_2\ p_1 = 59_{16}$。

小提示

把十进制数转换成二进制、八进制和十六进制等进制的表示，可以采用减权定位法或者除基取余法（小数部分：乘基取整法）。

2）二进制数转换成十六进制表示

二进制数转换成十六进制表示，把二进制整数从最后一位开始分成 4 位一组，然后把每组替换成相应的十六进制数码就可以了。例如，求 1010000100111111_2 的十六进制表示。转换如下：

$$1010000100111111_2 = 1010\ 0001\ 0011\ 1111_2 = A13F_{16}$$

二进制小数则从小数点后第一位开始分成 4 位一组，例如，求 0.1010000100111111_2 的十六进制表示。转换如下：

$$0.1010000100111111_2 = 0.1010\ 0001\ 0011\ 1111_2 = 0.A13F_{16}$$

如果最后一组不足 4 位，则用 0 补齐，然后把每组替换成相应的十六进制数码。

小提示

二进制数转换成十六进制表示，把二进制整数从最后一位开始向前分成4位一组，然后把每组替换成相应的十六进制数码即可。

二进制小数则从小数点后第一位开始向后分成4位一组，如果最后一组不足4位，则用0补齐，然后把每组替换成相应的十六进制数码即可。

3）八进制数转换成十六进制表示

八进制数转换成十六进制表示，把它先转换成二进制表示，然后再转换成十六进制表示。例如：求 120477_8 的十六进制表示。

$$120477_8 = 1\ 010\ 000\ 100\ 111\ 111_2 = 1010\ 0001\ 0011\ 1111_2 = A13F_{16}$$

小提示

十六进制数转换成八进制表示，把它先转换成二进制表示，然后再转换成八进制表示。

八进制数转换成十六进制表示，把它先转换成二进制表示，然后再转换成十六进制表示。

3.1.3 二进制数的运算

进入计算机的世界后，我们将频繁地接触二进制数。而在此之前，我们运算的都是十进制数，二进制数的运算则几乎没有接触过。下面把二进制数的运算规则给大家简要概括一下。

二进制数主要涉及的运算有两类：算术运算和逻辑运算。算术运算主要包括加、减、乘、除。而主要的逻辑运算有逻辑与、逻辑或、逻辑非以及逻辑异或。

1. 算术运算

1）加法运算

加法运算很简单，和十进制数的加法运算类似，只是进位时要记住“满二进一”。

例如：计算 $1101_2 + 1011_2$。

$$\begin{array}{r} 1101 \\ +\ 1011 \\ \hline 11000 \end{array}$$

因此，$1101_2 + 1011_2 = 11000_2$。

2）减法运算

减法运算也同样与十进制数的减法运算类似，只是不够减时，从上一位借位是“借一当二”。

例如，计算 $1101_2 - 1011_2$。

$$\begin{array}{r} 1101 \\ -\ 1011 \\ \hline 10 \end{array}$$

因此，$1101_2 - 1011_2 = 10_2$。

3）乘法运算

乘法运算的法则也不变，即分别用乘数的每一位去乘被乘数，然后把这些结果累加。只不过现在乘数与被乘数中只有两种数码 0 和 1。这两个数码又比较特殊：任何数乘以 1 还等于该数，任何数乘以 0 都等于 0。所以，二进制的乘法算起来也比较有意思。

例如，计算 $1101_2 \times 1011_2$。

$$
\begin{array}{r}
1101 \\
\times\ 1011 \\
\hline
1101 \\
1101 \\
0000 \\
+\ 1101 \\
\hline
10001111
\end{array}
$$

所以，$1101_2 \times 1011_2 = 10001111_2$。

4）除法运算

除法运算法则也和十进制数相同，由减法、逐位上商等操作分步完成。

例如，计算 $10001111_2 / 1101_2$。

$$
\begin{array}{r}
1011 \\
\hline
1101 \overline{)\ 10001111} \\
1101 \\
\hline
10011 \\
1101 \\
\hline
1101 \\
1101 \\
\hline
0
\end{array}
$$

所以，$10001111_2 / 1101_2 = 1011_2$。

小提示

二进制数主要的算术运算有加、减、乘、除。

二进制数算术运算的法则和十进制数的运算法则相同，只是进位为二。

2. 逻辑运算

逻辑运算是以前在十进制数中没有接触过的。常见的逻辑运算有逻辑与、逻辑或、逻辑非以及逻辑异或。其实逻辑运算中二进制数的 0 和 1 不再是数值的含义，而是分别代表真和假两个值。1 代表真，而 0 代表假。所以实际上逻辑运算是在真值和假值上的运算。等到你们讲信息技术应用数学中还会涉及这些逻辑运算。对于两位以上的二进制数来说，逻辑运算是在每个操作数对应的位之间进行的，所以也称为按位与、按位或等。下面我们通过具体的示例了解一下这些运算规则的定义。

1）逻辑与

逻辑与的运算符号为 ∧ 或者 &。

逻辑与运算的定义如下：

$$1\&1=1,\quad 1\&0=0,\quad 0\&1=0,\quad 0\&0=0$$

即只有两个参与逻辑与运算的数都是1时，运算结果是1，其余情况结果都为0。其实“逻辑与”有“和”、“并且”的意思。如果1代表真，0代表假，那么如果两件事情A、B，要“A并且B”为真时，显然必须A是真的，并且B也是真的才行。

例如，求1101 & 1011。

$$\begin{array}{rr} & 1101 \\ \& & 1011 \\ \hline & 1001 \end{array}$$

因此，1101 & 1011= 1001。

2）逻辑或

逻辑或的运算符号为∨或者|。

逻辑或运算的定义如下：

$$1|1=1,\quad 1|0=1,\quad 0|1=1,\quad 0|0=0$$

即1和任何数进行逻辑或运算结果都为1，其余情况结果为0。逻辑或有“或者”的意思。如果两件事情A，B，当然如果要“A或者B”为真，那么只要A和B中至少有一个是真的就行了。

例如，求1101 | 1011。

$$\begin{array}{rr} & 1101 \\ | & 1011 \\ \hline & 1111 \end{array}$$

因此，1101 | 1011= 1111。

3）逻辑非

逻辑非是个单值运算符，也就是它只对一个数进行操作。它的符号是～或者在数字的头上划一横线。运算规则为：

$$\sim 1=0,\quad \sim 0 = 1$$

意思很明显，就是取反。例如：

$$\sim 1101=0010$$

4）逻辑异或

异或的符号是⊕。它的运算规则如下：

$$1\oplus 1=0,\quad 1\oplus 0=1,\quad 0\oplus 1=1,\quad 0\oplus 0=0$$

从规则可以看出逻辑异或的意思就是看看两个操作数是否不一样。如果不一样那就是真的，结果为1；如果一样，那就是假的，结果为0。

例如，计算1101⊕1011。

$$\begin{array}{rr} & 1101 \\ \oplus & 1011 \\ \hline & 0110 \end{array}$$

因此，1101⊕1011= 0110。

小提示

二进制数主要的逻辑运算有逻辑与、逻辑或、逻辑非以及逻辑异或。

3.2　数字的编码

我们现在已经知道两件事实：一是计算机中存储的是 0/1 串；另一件事实是我们向计算机中存储的是程序和数据。程序多是用某种计算机程序设计语言编写的，具体体现形式就是一个符号串，包括英文字母、数字、运算操作符以及一些特别规定的符号等。而数据则包括用来运算的数以及我们的一些文档，如最近新写的一首小诗。那么这些日常使用的数字、字母、汉字与 0/1 串有什么关系呢？是怎样互相转换的呢？这就是编码问题。

下面就来说说数字、字母以及汉字在计算机中是怎样表示的。我们先从数字说起。

3.2.1　原码

从上一节的介绍可见，数的表示应该说比较简单。你现在一定在想：这也用说，把各种数字转换成二进制表示就行了。噢，你说的没错。不过还有一个小小的问题。我们平时用的数是分正负的。如果是负数，前面的那个负号怎么表示呢？你肯定会说，这有什么难的？在转换后的二进制数前单独加一位表示符号不就行了吗？如果是负数，符号位就为 1，如果是正数，符号位就为 0。问题很轻松就解决了！这确实是一种办法。这种数据表示法被称为原码，名字的意思很明显。

一般情况下，计算机中的数都是要参加运算的。最常见的运算就是加、减运算了。现在我们来看一看对于用原码表示的数，在计算机中对其进行加法运算时的步骤。假设我们要计算两个数 57_{10} 和 -68_{10} 的和。

这两个数在计算机中应该是什么样子呢？我们先把它们转换成二进制表示：

$$57_{10} = 111001_2$$

$$68_{10} = 1000100_2$$

做这个转换你一定很得心应手了。接下来要做的事情就是加上符号位。现在又来了一件小小的麻烦事。如果直接在 1000100_2 前面加上一个 1 表示负号，变成 11000100，那么怎样能区分它表示的是负的 1000100_2 而不是正的 11000100_2 呢？我们需要把符号位的位置固定下来，比如，在从最右边开始向左数的第 8 位(从 1 开始计数)作为符号位。这样：

$$57_{10} = 111001_2 \quad [57_{10}]_{原} = 0\ 011\ 1001$$

$$68_{10} = 1000100_2 \quad [-68_{10}]_{原} = 1\ 100\ 0100$$

这意味着什么？后面有 7 位可以用来表示数值。前面我们已经讲过，地址总线的宽度会决定最大的地址。同样道理，如果我们已经决定用 8 位来表示数，并且 1 位作为符号位，余下 7 位表示数的绝对值，那么可以表示的数的范围就固定了。现在，你知道为什么会常常听到关于计算机中整数长度的讨论了。就目前这种假设情况，我们能表示的数的范围是什么？你算算。

具体地说，如果把原码的长度确定为 n 位，则数 x 的原码定义如下：

$$[x]_{原}=\begin{cases}x, & 0\leqslant x<2^{n-1}\\ 2^{n-1}-x, & -2^{n-1}<x\leqslant 0\end{cases}$$

现在我们要通过 57 和－68 这两个数的原码来算它们的和，并且把和保存在计算机中，也就是说结果也要转换成原码的形式。这是两个带符号数的加法。算这样的加法，步骤如下：

首先判断两个加数的符号位；
　　如果符号位相同，
　　则将两个数相加，和的符号位与加数相同；
否则（如果符号位不同），
　　比较两个数绝对值的大小；
　　用绝对值大的数减去绝对值小的数，得出结果的绝对值；
结果的符号位与绝对值大的加数相同。

具体到我们的示例上，求和步骤如下：

① 首先通过 57_{10} 和 -68_{10} 的原码判断它们的符号位是否相同。

可以看到 57_{10} 和 -68_{10} 的原码符号位分别为 0 和 1，不相同。

② 接下来比较这两个数的绝对值的大小，也就是原码的数值部分。

可以发现 100 0100 大于 011 1001。

所以，用 100 0100 减去 011 1001，结果为 000 1011。

最后确定结果的符号位，它应该与绝对值大的数，也就是 100 0100 的符号位相同，即为 1。

所以

$$[-68_{10}]_{原}+[57_{10}]_{原}=1\ 000\ 1011=[-11_{10}]_{原}$$

我们可以发现，对于这个求和过程，最后具体运算过程是加法还是减法取决于具体加数的情况。所以，为了实现原码表示数的加法，计算机中需要有能够判断符号位的硬件逻辑，还需要有能计算加法的逻辑电路，以及能够计算减法的逻辑电路。这样的硬件设计是不是有点儿太麻烦了？如果能够不用考虑符号位，并且也不用判断两个数的大小，算加法时什么都不用考虑，只要把计算机中两个数的编码直接相加，就能得出正确结果的编码，那该多好。这个目标听起来有点儿难以实现。不过，科学家真的是非常聪明，他们就真的把这件事做成了。好像只要你能提出要求，他们就会想办法做到。他们是怎么做的呢？请看下一小节。

不过最后还要说一点，除了与数的真值有简单的对应关系，比较容易相互转换之外，原码还有一个优点，就是乘除法的运算规则简单。等学习完后面几种数字的编码，你可以自己想想怎样实现这些编码的乘除法，然后比较一下看看是不是这样。

小提示

原码表示法直接用数字绝对值对应的二进制数表示，并在前面加一位符号位，用 1 表示负号，0 表示正号。

数的原码同其真值有简单的对应关系，比较容易相互转换。

原码实现加减法很不方便，但实现乘除法的规则简单。

3.2.2 补码

前面讨论原码时，我们假设计算机用 8 位来表示一个数。这种情况下，可以出现的不同

0/1 串范围为从 0000 0000～1111 1111，一共有 256 种不同的串。对于原码表示法，第一位是符号位，因此可以表示的正数为 0000 0000～0111 1111，换算成十进制整数就是 0～127；而负数为 1000 0000～1111 1111，即从 0～－127。所以 8 位原码可以表示的整数范围为 127～－127。一共表示了 255 个整数，因为数字 0 占用了两个码：1000 0000 和 0000 0000。

原码就先说到这里。现在回头继续说第 3.2.1 节留下的目标，找到一个数字编码形式，它能够进行加法运算时不用考虑符号位，也不用判断两个数的大小，只要把计算机中两个数的编码直接相加，就能得出和的相应编码结果。

现在我们试着来设计这种编码。还是假定用 8 位来表示整数。我们已经知道 8 位一共可以表示 256 个不同数。如果全用来表示正数，则可以表示 0～255。现在我们希望用它来表示有符号数，并且 0 只占用一个 0/1 串。所以我们确定新的编码所表示的数的范围在 127～－128。那么我们仍然用 0000 0000～0111 1111 这些 0/1 串表示和它们二进制数值相同的数 0～127。而对应 128～255 这 128 个正数的 0/1 串，即 1000 0000～1111 1111，则用来表示－128～－1 这 128 个负数。我们用 128 对应的二进制串 1000 0000 表示－128，用 129 对应的二进制串的 1000 0001 表示－127，以此类推，用 255 对应的二进制串的 1000 0010 表示－1。发现一个规律没有？即 128＝－128＋256，129＝－127＋256，255＝－1＋256。好了，现在假设用这个编码，暂时把这种编码的名字叫做简单码，因为它似乎的确太简单了。现在算算 57_{10} 和 -68_{10} 在这种编码规则下对应的数据表示。

因为　　$57_{10}=111001_2$，

所以　　$[57_{10}]_{简}=0011\ 1001$

因为　　$256-68=188, 188_{10}=1011\ 1100_2$，

所以　　$[-68_{10}]_{简}=1011\ 1100$

现在把 $[57_{10}]_{简}$ 和 $[-68_{10}]_{简}$ 加在一起，看看是什么。

$$\begin{array}{r} 0011\ 1001 \\ +\quad 1011\ 1100 \\ \hline 1111\ 0101 \end{array}$$

$$1111\ 0001_2=245_{10}=256-11=[-11_{10}]_{简}$$

也就是说 $[-68_{10}]_{简}+[57_{10}]_{简}=[-11_{10}]_{简}$。这是巧合吗？下面再找另外两个数试验一下。

这一次我们算算－68 和－57 的和。

因为　　$256-57=199, 199_{10}=1100\ 0111_2$，

所以　　$[-57_{10}]_{简}=1100\ 0111$

$[-68_{10}]_{简}=1011\ 1100$

那么　　$[-68_{10}]_{简}+[-57_{10}]_{简}=1011\ 1100+1100\ 0111=1\ 1000\ 0011$

因为现在的数据只能有 8 位，也就是说存储器中只为每个数留了 8 位的空间，所以，我们最前面的进位 1 扔掉。结果就是 1000 0011。

而　　$1000\ 0011_2=131_{10}=256-125=[-125_{10}]_{简}$

而－68 和－57 的和正好是－125。看起来不像巧合。那么我们就找到了希望的编码：计算两个有符号数的加法只要直接把它们的简单码相加就可以得出和的简单码，而不用考虑加数的符号。

其实，这样的结果完全是由于我们前面发现的那个规律，129 = −127 + 256，即$[-127]_{简}$=129= 256 +(−127)。这里面蕴涵着两个概念，就是“模”与“同余”的概念。模是指一个计量器的容量。比如，我们日常使用的钟表，上面一共用 12 个格表示整点。所以钟表的模就是 12。我们可以用钟表来表示 19 点，可是在表盘上你看到的指针是指向 7 点。我们说 19 和 7 是模 12 同余的。因为 19 和 7 除以 12 的余数都是 7。也就是说，在计量容量为 12 的表盘上，7 点和 19 点的表现是相同的，都是 7。现在我们再来看另外一件重要的事情。假设现在是 7 点，如果需要把时间调整到 3 点，那么有两种办法：可以把表针拨回 4 个小时；也可以把表针向前拨 8 个格。结论是，在只有 12 个格的表盘上，7−4=7+8。现在可以发现−4 和 8 是有一定关系的。仔细再观察一下，可以发现，在表盘上表示 8 和−4 其实是同一个格，都是 8 点。只不过看你从哪个方向数。从 0 点开始顺时针数那个格就可以称为 8 点。如果从 0 开始逆时针数那个格就是−4。实际上，−4 和 8 也是模 12 同余的。我们称−4 和 8 互为补数。这里有两个规律。首先，将负数加上模就可以得出它的正补数，比如，−4+12 = 8。还有，与一个负数的加法可以转换成与这个负数补数的加法，比如，7+(−4)=7+8(mod 12)。为了避免混淆，在算式中加上(mod 12)表示以 12 为模。

现在知道为什么前面的编码能把同负数的加法直接转换成两个数的加法，而不用考虑符号了吧？在这种情况下，因为只有 8 位的存储空间来存放整数，所以只能表示 256 个数，这样放整数的容器的模就是 256。为了把减法转换成加法，我们把负数用它的正补数值表示，即−127 用 129(=256 +(−127))表示。然后，当我们要计算与某个负数的加法时，比如 25 加上−127，则直接把它们的编码表示相加，也就是 25 + 129 = 154。因为我们表示数的范围是−128～127，所以我们会把 154 解释成它对应的负补数，就是−102 (=154−256)。实际上我们是用补数来表示每个数(包括正数，因为正数的补数就是它自己，对吧?)，所以这种编码的学名不叫简单码，而叫做补码。

正数的补码就是它本身。负数的补码等于模加上这个负数，实际上就是模减去负数的绝对值。模是多少呢？对于用 8 位二进制数是 256(=2^8)，对于 n 位二进制数就是 2^n。

总结一下，如果补码的长度定为 n 位，那么数 x 的补码定义为：

$$[x]_{补}=\begin{cases}x, & 0\leqslant x<2^{n-1}\\ 2^n+x, & -2^{n-1}\leqslant x<0\end{cases}$$

从定义可以看出，正数的补码比较好求，而负数的补码要做一个减法运算。

例如：求−68 的补码。

$$68_{10}=1000100_2$$

$$2^8_{10}=1\ 0000\ 0000_2$$

所以

$$[-68_{10}]_{补}=100000000-0100\ 0100=1011\ 1100$$

实际上，通常并不用计算这个二进制减法，而是用另外一种简单的算法。即把 68 对应的二进制数 0100 0100(一定要记住前面加 0，补全 8 位)按位取反(即变成 1011 1011)，然后在末位加 1(1011 1011 + 1 = 1011 1100，对吧?)。这样就把减法换算成取反加一这样简单的运算。这种算法用硬件实现起来非常简单。二进制数还要好多吸引人的运算窍门呢。

再告诉你一种求负数补码的简单办法。首先，把负数的绝对值对应的二进制数求出来(什么办法这一步好像都不能省，好在这对你来说已经轻车熟路了)，并在前面补上必要的

0，凑全位数，比如8位；然后，从后面开始找到第一个1，把它记下来，并把其后的0也都抄下来；最后，把这个1前面的各位取反，即1变0，0变1。我们还是拿－68来试一下。假设这次用16位编码。

第一步，求出二进制表示，并补全位数（千万别忘了）。

$$68_{10}=0000\ 0000\ 0100\ 0100_2$$

第二步，找到第一个1，并将其以及后面的0照抄下来。第三位上的1是从后面数第一个遇到的1，所以将后面的三位100保留下来。

将第三位上的1前面的各位都取反，就得到了补码：

$$[-68]_{补}=1111\ 1111\ 1011\ 1100$$

简单吧！你用补码最原始的定义算一算，验证一下，看看我们算的对不对。

补码这种把减法换成加法的特性，可以大大简化计算机硬件的设计。你可以想象一下，如果用原码，那么设计加法运算的硬件时，必须要有两套电路，一套算加法，一套算减法，而且在运算之前还要有一套电路对加数的符号进行比较，然后决定是用加法电路还是减法电路，最后算好的结果还要根据前面比较的情况来确定符号。多复杂呀！而补码则不用了，只要一套加法电路就行了，根本不用考虑符号，并且减法都可以转换成加法来实现。负数的补码也很好求，取反加1，在电路上实现起来也非常简单。所以，补码是目前计算机中使用很广的一种数字表示方式。

小提示

正数的补码就是它的二进制表示。
负数的补码可以通过将其二进制表示按位取反、末位加一的方法获得。
补码实现加减法的规则比较简单。

3.2.3　反码

第三种数的表示法叫做反码。它的定义很简单：正数的反码等于正数本身，负数的反码等于把负数绝对值的二进制表示每位取反。其实就是求补码中间步骤，按位取反得出的结果。只不过补码多了个加1的步骤。

计算机中很少用反码直接进行运算。

3.2.4　移码

除了上述几种数字编码外，有时也会用到一种叫做移码的表示法。如果移码的长度定为 n 位，那么数 x 的移码定义为：

$$[x]_{移}=2^{n-1}+x,\quad -2^{n-1}\leqslant x<2^{n-1}$$

为什么叫移码呢？因为移码是原来的数加上一个 2^{n-1}，在数轴上也就是所表示数的位置向轴的正方向移动 2^{n-1}，因此而得名。移码和补码有一个很好的关系：一个数的补码和移码的符号位正好相反，而其他位则完全相同。例如，如果采用8位编码，那么，十进制数7的移码表示就是1000 0111，补码是0000 0111。而－7的移码表示为0111 1001，补码则是1111 1001。

3.2.5 小数的表示

前面都是在讲整数怎样表示，关键在于解决符号的表示问题。不过，除了整数，很多地方还要用到小数。小数比整数多了个小数点，它把数值分成整数部分和小数部分。所以，要表示小数就要解决小数点的表示问题。

通常有两种办法可以解决这个问题。一是把小数点固定在某个默认位置。小数点并不在数的表示中出现。这种方法表示的数被称为定点数。通常小数点的固定位置有两种情况。一是默认固定在数的最右边。也就是说，这种数只有整数部分，实际上也就是整数。因此我们也把这种数称为定点纯整数。另外一种就是把小数点固定在符号位之后。那么，这类数也就只有小数部分，因此，也被称为定点纯小数。

定点表示法可以表示数的范围很有限，要么是整数，要么是小数。可是，通常使用的数这两部分都是有的。要表示这样的数，就应该允许小数点位置的变化。如果这样，那就需要在表示中给出小数点位置。这就是第二种小数的表示法，我们将这样表示称为浮点表示，意思是小数点可以浮动位置。这种表示法表示出来的数称为浮点数。

那么怎样表示小数点的位置呢？计算机采纳了科学记数法的方式。小数

$$N1=0.000001234$$

用科学记数法来表示就是：

$$N1=1.234\times10^{-6}$$

科学记数法中乘号前面的小数部分，如 1.234，给出了有效数，而后面的幂 10^{-6}，给出了小数点的位置。计算机中的浮点数也采用这种方式。因此，一个浮点数分为两部分，一部分称为尾数，就是科学记数法中的有效数位部分，如 1.234。另外一部分称为阶码，记录的是后面的幂次，如前面的－6。但是没有记录基数。通常在一个系统中，会使用默认的基数，如 2 或者 4。

浮点数的尾数一般用补码或原码表示。阶码则用补码原码或者移码来表示。与科学记数法不同的是，浮点数的尾数只记录小数部分，省略了科学记数法中的 1 位整数部分(因为对于二进制数来说，这 1 位整数部分的值只能是 1，所以就可以省略了)。另外，阶码只记录了幂的次数。计算机中使用的基数有 2，4，16 等，通常默认的基是 2，而不是像科学记数法通常采用的 10。

前面提了这么多种浮点表示可以采用的编码，如原码、补码，几乎数字的各种编码都可以使用。当然，为了便于软件的移植，大家都用相同的编码最好。这就需要定义标准。为此，IEEE 在 1985 年给出了 IEEE754 标准。这个标准规定浮点数的基数为 2、阶码用移码表示，尾数用原码表示。它规定了单精度和双精度两个基本格式。单精度格式浮点数为 32 位，分别是 1 位符号位、8 位指数位(阶码)、23 位有效位(尾数)。

双精度是什么，就不介绍了。如果你对具体的定义感兴趣请查阅其他计算机原理书，你会发现在数的表示上还有好多要仔细考究的地方。

3.3 字符编码

现在知道数字在计算机中是怎样表示的了。但是我们写的程序在计算机中是怎样存储的呢？程序一般是由英文字母加上一些符号构成的。因为计算机中只能存储 0 和 1，所以

我们必须为每个字母或符号规定一个0/1串表示，我们把这种规定叫做字符的编码。当然每个人都可能会给出一种编码。比如，小明希望用0111 0000表示小写字母a，而用1111 0000表示大写字母F；可是Tom喜欢用0111 0000表示字母T，而用1111 0000表示字母o，用0000 0100表示字母m。显然，要想使不同的计算机能够互相识别各自表示的字母，那么就必须使用相同的编码。这些相同的编码就是编码标准。不过不幸的是，标准也有很多。不同的国家，不同的公司都在制定标准。这一节中我们就介绍三个有代表性的编码标准。它们分别是使用最多的字符编码——ASCII码，汉字编码标准之一——GB2312，以及力争统一全世界字符的统一码——Unicode。

3.3.1　简单字符的编码——ASCII码

使用最广泛的编码标准是美国国家信息交换标准字符码。这一长串名字的英文首字母缩写就是ASCII(American Standard Code for Information Interchange)，所以这种编码被称为ASCII码。这种编码用8位来表示128个字符。这些字符包括：10个阿拉伯数码0～9，52个英文字母a～z和A～Z，34个专用符号和32个控制符号。表示128个不同的符号只要7位二进制数就够了。而8位的0/1串可以有256个不同序列。实际上ASCII码的最高位都为0，也就是说它只用了后面的7位，所以一共只表示了128个不同的字符。具体的编码规则如表3-1所示。有些公司会把最高位进行定义，这样就可以再多表示128个字符，这种编码被称为扩展的ASCII码。

在表3-1中可以看到，ASCII码用十进制数97对应的二进制0/1串，也就是0110 0001，表示小写字母'a'。根据这种编码规则，如果你想在计算机中存储小写字母'a'，计算机就在存储器中存储0110 0001。现在问题出现了。你一定会问："如果存储器某个单元的内容是0110 0001，我是认为它表示的是字母'a'呢？还是数字97呢？"

在实际的计算机系统中，如果计算机处于指令周期的取指令阶段，CPU从存储器取出的内容会被当作机器指令来解释。机器指令也是一种0/1串，我们将在下一小节说明。在执行指令阶段，CPU需要从存储器取操作数进行运算，这时从存储器取出的内容就会被解释成数据。但是，这并不能回答前面提出的问题啊！如果取数据时取出0110 0001，那么可以解释成97，也可以解释成字母'a'。这个问题的解决留给了程序设计。

当用某种程序设计语言写程序设计时，如果需要在存储器中存储数据，那么必须首先申请存储单元，并且要指明这个存储单元准备存储什么类型的数据。用程序设计语言的说法就是，必须首先定义一个变量并且声明变量的类型。变量其实就是给存储空间起的别名，就好像我们不愿意讲"中央大街115号"，而把那间房子起名叫做"空间站"。这样只要我们说"空间站"指的就是"中央大街115号"。定义变量就是申请存储空间，计算机按照我们的要求分配存储单元，并且在我们的程序中，这个存储单元就用我们给出的变量名标识。比如，我们定义一个变量名叫age。声明变量的类型就是说明这个存储单元要用来存储什么类型的数据。例如，"int age;"，表示变量age所对应的存储单元要存储整数。这样当计算机从age这个存储单元取出0110 0001，就会把这个串解释成整数97，而不是字母'a'。与之相对的是，如果定义age时，写的是"char age;"，表示变量age所对应的存储单元要存储字符。那么当计算机从age中取出0110 0001时，就会把它解释成字母'a'，而不是整数97。

表 3-1 ASCII 码表

Dec	Hx	Oct	Char	Dec	Hx	Oct	Char	Dec	Hx	Oct	Char	Dec	Hx	Oct	Char
0	0	000	NUL(null)	32	20	040	SPACE	64	40	100	@	96	60	140	'
1	1	001	SOH(start of heading)	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX(start of text)	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX(end of text)	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT(end of transmission)	36	24	044	$	68	44	104	D	100	64	144	d
5	5	005	ENQ(enquiry)	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK(acknowledge)	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL(bell)	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS(backspace)	40	28	050	(	72	48	110	H	104	68	150	h
9	9	011	TAB(horizontal tab)	41	29	051	)	73	49	111	I	105	69	151	i
10	A	012	LF(NL line feed, new line)	42	2A	052	*	74	4A	112	J	106	6A	152	j
11	B	013	VT(vertical tab)	43	2B	053	+	75	4B	113	K	107	6B	153	k
12	C	014	FF(NP form feed, new page)	44	2C	054	,	76	4C	114	L	108	6C	154	l
13	D	015	CR(carriage return)	45	2D	055	-	77	4D	115	M	109	6D	155	m
14	E	016	SO(shift out)	46	2E	056	.	78	4E	116	N	110	6E	156	n
15	F	017	SI(shift in)	47	2F	057	/	79	4F	117	O	111	6F	157	o
16	10	020	DLE(data link escape)	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1(device control 1)	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2(device control 2)	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3(device control 3)	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4(device control 4)	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK(negative acknowledge)	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN(synchronous idle)	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB(end of trans. block)	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN(cancel)	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM(end of medium)	57	39	071	9	89	59	131	Y	121	79	171	y
26	1A	032	SUB(substitute)	58	3A	072	:	90	5A	132	Z	122	7A	172	z
27	1B	033	ESC(escape)	59	3B	073	;	91	5B	133	[	123	7B	173	{
28	1C	034	FS(file separator)	60	3C	074	<	92	5C	134	\	124	7C	174	|
29	1D	035	GS(group separator)	61	3D	075	=	93	5D	135	]	125	7D	175	}
30	1E	036	RS(record separator)	62	3E	076	>	94	5E	136	^	126	7E	176	~
31	1F	037	US(unit separator)	63	3F	077	?	95	5F	137	_	127	7F	177	DEL

从现在开始，我们认识到一个事实：存储单元中的 0/1 串在不同的情况下可能被解释成不同的事情。所以，如果不知道计算机目前所处的情况，单单看一个存储单元的内容，我们什么信息都得不到。

小提示

为字符规定一个对应的 0/1 序列，叫做为字符编码。

现在存在很多个字符的编码标准。

计算机中的 0/1 串可以被解释成字符、数字、指令等多种情况，计算机系统根据具体情况进行相应的解释。

程序中的数据变量必须声明类型，计算机才能正确解释它的内容。

3.3.2　汉字字符的编码——GB2312

ASCII 码解决了简单字母和符号在计算机中的表示问题。可是，要在计算机中存储的内容还有好多。比如说，你想把最近写的一首小诗存在计算机中，然后把它放到主页上，让大家欣赏欣赏。当然，你的诗是用汉字写的。再比如，要把原版的《一千零一夜》做成电子版。你知道原版的《一千零一夜》应该是用什么文字写的吗？阿拉伯文！它同英文以及汉字可完全不同。那么我们又该怎么办呢？其实，道理很简单，只不过做起来辛苦一点儿。仍然是为每个需要表示的字符定义一个 0/1 串就行了。每一种定义就形成一种编码。当然，在解释这些 0/1 串时，要知道用的是哪种编码方案，用对了才能还原出最初的内容。上网时用户经常会遇到一种情况：网页上显示的都是一些看不懂的符号，将其称为乱码。这时我们需要在浏览器的菜单条中找到“编码”一项（见图 3-1）。然后就会看到很多名称，比如简体中文、阿拉伯文、繁体中文、希伯来文等。我们需要在这些名字中间试验选择，直到网页上的东西看起来像是可以理解的文字为止。从这个菜单上列出的项，我们可以对各种文字的编码

图 3-1　浏览器的“编码”选项

有个初步的认识：编码方案真是够多的！每种文字至少有一种编码。

这里不能把所有编码都介绍，而且说这些干巴巴的编码也实在没有意思。我们就以一个汉字编码方案为例简单介绍一下。就说说图 3-1 中看到的简体中文(GB2312)吧。这是我们国家为了能在计算机内存储和表示汉字，在 1980 颁布的一套国家标准。它的全称是“信息交换用汉字编码字符集 基本集”[40]。它的标准号是 GB2312—80。在这个标准中，汉字以及一些序号、字母等被称为图形字符。GB2312—80 一共对 7445 个图形字符进行了编码，其中包括一般符号、序号、数字、拉丁字母、日文假名、希腊字母、俄文字母、汉语拼音符号、汉语注音字母、汉字等。GB2312 能够表示汉字的数量为 6763 个。每一个图形字符都用两个字节表示。

GB2312—80 定义了一个代码表。代码表为其支持的 7445 图形字符进行了编号。代码表分为 94 个区。每个区中有 94 个图形字符的位置，GB2312 把这些位置称为位。所以，一个区中就有 94 个位(为什么用 94 呢？不知道有没有什么特殊的意义？)。区的编号是从 1～94。位的编号也是从 1～94。所以一个图形字符的编码就可以用它的区号和位置号表示了。区号加上位号，也被称为区位码。比如，汉字“啊”在 16 区的第一位，所以它的区位码是 16-01。写的时候区号和位号用连字符相连。如果你的计算机上有区位码输入法，那么用的就是这个区位码。

但是，GB2312 并没有用区位码做每个图形字符最终编码。GB2312 最终定义并使用的编码被称为国标码。国标码实际上是在区位码的基础上加上一个十六进制数的 2020H。那么最后计算机中存储汉字“啊”用的是哪个 0/1 串？是 16-01 换算成十六进制数加上 2020H 的和的二进制表示？非也！因为计算机系统中可能同时支持 ASCII 码和国标码。这时候就会带来一些麻烦。比如，两个字节单元的内容是 3021H 时，如果解释成国标码，那么表示的就是汉字“啊”(你用区位码 16-01 计算一下看看对不对？)。如果解释成 ASCII 码，那么就是字符'0'和'!'(查一下前面的 ASCII 码表，核实一下)。所以，为了解决这个问题，计算机中并不存储汉字的国标码，而是再把它加上一个十六进制数 8080H。这样就保证两个字节的最高位都是 1，也就能与 ASCII 码分辨开了。我们把最后存在计算机中的这个编码称为机内码。够复杂吧？建议你再仔细看几遍，把这几个编码的关系弄清楚。

小提示

ASCII 码是使用最广泛的编码标准，它使用 7 位编码来表示 128 个字符。
简体中文 GB2312 是我国为了能在计算机内存储和表示汉字，在 1980 年颁布的一套国家标准。

3.3.3 字符的统一码——Unicode

好了，你现在已经清楚了，这些编码不过就是把每个要在计算机中表示的字符规定一个编号而已。所以，我们就不过多地介绍这些编码的具体内容了，也实在是介绍不过来呀。不过还有一个编码一定要讲一下。因为它比较特殊。它就是统一码，即 Unicode。

我们先讲讲 Unicode 出台的背景。你已经看到了，不同的编码实在太多。这除了给我们的学习带来不便之外，最麻烦的是，把一篇文章从一台计算机复制到另外一台上之后，可能就面目全非了。例如，明明你写好的一首诗，放到 Web 服务器上，结果你的女朋友在家里

上网看到却是一堆乱七八糟的东西。这确实比较麻烦。当然，你可能也会因祸得福，当你选定浏览器菜单上的编码项，把那堆乱东西变成一首小诗时，女朋友的目光里可能会充满崇拜：你真的很专业！不过话又说回来，为了以后不再学习这一节的内容，不需要再了解浏览器菜单上编码一项是干什么用的（万一你不知道编码这回事，在女朋友面前，对着你那首面目全非的诗手足无措，该多丢面子！），所以你一定希望世界上只有一种编码。这样你的麻烦事少了，设计和开发浏览器的工程师也终于可以把菜单上的那个编码项去掉了（为了这一项内容，工程师们不知道要花多少时间！）。计算机领域的专家们也想到了这件事。于是他们就开始做这件事情。而且不止一个专家想到了这件事情，所以就出现了两个组织同时开始做全世界统一的，当然希望也是“唯一”的编码[41]。好在他们后来相互知道了这件事情，统一了思想。不然世界上就将出现两个不同的“统一”编码方案。

统一编码方案希望能够给世界上所有的文字都规定一个唯一的号码。因此，统一编码方案的号码空间必须足够大，也就是号码的位数要足够长。在存储器一节中我们已经知道这个道理了。这个统一编码就是 Unicode。Unicode 的学名是 Universal Multiple-Octet Coded Character Set，简称为 UCS。UCS 可以看作是 Unicode Character Set 的缩写。Unicode 的目标就是要给每个字符提供一个唯一的数字，不论是什么平台、什么程序、什么语言。

Unicode 标准已经被工业界的著名公司们采用，如 Apple、HP、IBM、Microsoft、Oracle 等。许多操作系统、所有最新的浏览器和许多其他产品都支持它。

Unicode 给每个字符分配一个数字，这个数字被称为码点。UCS 目前有两种格式：UCS-2 和 UCS-4。UCS-2 就是用两个字节编码，而 UCS-4 就是用 4 个字节编码。UCS-2 的码点是 16 位二进制数。所以这样算来它也只能表示 65 536 个字符，好像并不能代表世界上的所有字符。因为担心最终码点数将超过 65 536，所以 Unicode 设计了 UCS-4。它用 32 位（实际上是 31 位，最高位没用）二进制数来表示字符，所以就有 $2^{31}=2\,147\,483\,648$ 个码位。这个数字看起来应该够用了。如果不够，设计者还可以设计出 UCS-8 或者 UCS-16。当然 UCS-2 的码点是包含在 UCS-4 中，不然就又出现两个编码了。只要把 UCS-2 规定的码点前面加 0 扩充到 4 字节，就是该字符的 UCS-4 编码。目前 UCS-2 的码点还没有被全部分配出去。Unicode 的设计者正仔细地分配这些有限的资源，以免浪费。

Unicode 和别的编码还有一个不同。码点并不是字符在计算机中的编码表示，而只是字符对应的一个数字而已。它的码点其实可以理解成只存在于纸上，它的表示范围实际上并没有限制。那么码点在计算机中怎样表示呢？这个问题看起来很奇怪，因为码点已经是数字了，而且是二进制数字，把它存储在计算机中就行了。可是，UCS 的码点并没有被全部分配出去，所以 16 位/32 位的二进制数并没有完全都用上。而且对于一些国家来讲，尤其是美国人，他们的字符集只要 26 个英文字母加上一些符号，也就是 ASCII 码表就够了。所以如果采用 UCS-2 或 UCS-4，那么在美国的机器上，字符表示的第一个字节都是 0。这对空间是个不小的浪费。为此，这些人就提出了另外一个规范，叫做 UCS 传输格式，缩写为 UTF（UCS Transformation Format）。常见的 UTF 规范包括 UTF-8、UTF-7、UTF-16。顾名思义，UTF-8 就是用 8 位来表示 UCS 码点。UTF-7 就是用 7 位来表示。而 UTF-16 则是用 16 位来表示。UTF-16 和 UCS-2 的位数相同，但它也不是将字符的 UCS-2 码点直接拿来用，也有一些不同[42]。不过目前那些不同的码点还没有被分配出去，所以，可以暂时认

为 UTF-16 和 UCS-2 一样。现在麻烦事又来了，尽管全世界的字符都被统一编码了，但是这些编码在计算机中的表示并不一样。你可能已经注意到了，在浏览器菜单编码项中的 Unicode 后面有个括号里面写着 UTF-8。所以，就目前的发展来看，要想去掉浏览器菜单中的编码项还需要很长一段时间。

对了，顺便说一句，在浏览器菜单的编码项中还有“自动选择”一个选项。猜猜是怎么实现的？实际上在描述每个网页的 html 文件中有一个 content-type 项，会告诉浏览器这个网页用的是什么编码。但是为什么有时还会出现乱码呢？这是因为有些网页并没有给出 content-type 项的内容。那这种情况下，还能不能实现自动选择了呢？答案是肯定的，目前常见的一种做法是“猜”。因为每种文字都有一些出现频率非常高的字符，如英文中的 t，汉字中的“的”等。利用这个特性，统计出现概率最高的码，就可能推测出这个网页使用的编码。当然不是总能猜得准，所以有时候还会看到乱糟糟的东西。

小提示

Unicode 的目标是为世界上的每个字符提供一个唯一的数字。
Unicode 给每个字符分配一个数字，这个数字被称为码点。
UTF 是 Unicode 码在计算机中的表示形式。

3.4 机器指令

通过上面小节，我们知道程序和诗是怎么存储在计算机中的。但是，你一定还有个问题：计算机怎么看懂我们写的程序的呢？回答这个问题之前，先来想想我们是怎样和外国人沟通的。一般我们有两种办法。其一就是我们学会外语，直接和老外讲外语，这样他们就能明白我们说的是什么。还有一种办法就是找一个既懂汉语又懂外语的翻译，他在中间把我们的话翻译成老外能懂的语言。同样道理，如果希望计算机能明白我们的意思，也有两种途径：一是我们讲计算机的语言；再有就是找个翻译，把我们的话翻译成计算机的语言。下面我们先来介绍第一种方法，就是讲计算机的语言。第二种方法将在第 3.5 节中介绍。

计算机能够理解的语言被称为机器语言。机器语言由一条一条语句构成的，这些语句通常被称为机器指令。这些指令是什么样子呢？在 CPU 一节中我们已经了解到，控制器从存储器中取出的程序来执行。在存储器中的指令只能是 0/1 串。要想让计算机能明白我们的意思，程序中只能是计算机指令。你一定发现了，在计算机中除了 0 只有 1，没有其他的。控制器读取一条指令后，比如 01010101，然后把它交给译码器。译码器是会翻译指令的硬件，它会把指令翻译成相应的控制信号，如把 01010101 翻译成给加法部件的启动信号。计算机就这样读懂了这条指令。译码器怎么会翻译？别着急，马上就会介绍。

在介绍译码器之前，我们先来了解指令的格式。机器指令一般分成两部分，一部分叫做操作码，另外一部分叫地址码，如图 3-2 所示。有的计算机的操作码长度是固定的，有的是可以变化的。根据操作码的不同，地址码字段中可能有一个地址也可能有多个地址。例如，若操作码表示加法操作，那么地址码字段中就可能存在三个地址，分别是两个加数和一个存放和的存储单元的地址。那么译码器是怎样把指令，例如 01010101 翻译成加法的呢？其实说起来也不难。假定这条指令中前三位是操作码，也就是 010（当然实际机器的指令长度和

操作码的长度都要比这长，不然三位的操作码字段意味着只能有 8 条不同的指令，那这个计算机的能力也太低了）。控制器会将这三位操作码传递给译码器。译码器的核心是一个译码电路。它有一些输入端和输出端。图 3-3 是译码器的简单示意图。当有指令到来时，指令的操作码，比如 010，就会通过译码电路的输入端传递给译码电路。假定用高电平表示 1，低电平表示 0，那么 010 这种输入会使第二条输入线路上发送一个高电平信号，而第一条和第三条线路上产生一个低电平信号。在这种信号输入的情况下，译码器中搭设的电路就在第一个输出线路上产生一个信号，这个信号就会开启加法运算电路。这个信号还可能同时传递给其他相关的电路，如完成以下功能的电路：取指令中的地址，按照地址读取加数，并把加数传递给加法电路。这个译码过程让你感觉计算机好像能够看懂 010 是让它算加法一样。而实际上完成这一动作的是没有思想的各种电子元器件。图 3-3 只是个简单的译码器逻辑示意图，实际上的译码器比这复杂得多。

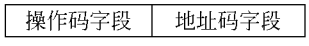


图 3-2　机器指令的一般格式

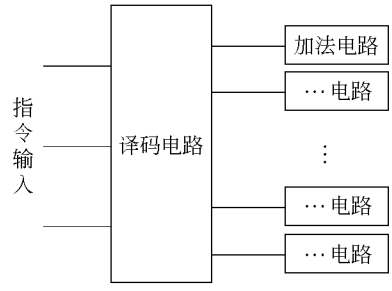


图 3-3　译码器结构示意图

在 CPU 出厂时，译码电路就固定下来了，所以一台计算机能够理解的机器指令在 CPU 出生时就确定了。因此，每个 CPU 都有自己的指令集。CPU 内部的结构不同，它支持的指令集合也不同。当然这些指令集合都是一些 0/1 字符串。如果你想不通过翻译直接让计算机为你做事，那么你就需要了解这些指令及其用法。然后把你要做的事情用这些指令描述出来，这就是程序。最后把你的程序交给计算机去执行就可以了。

小提示

计算机能够理解的语言被称为机器语言。
机器语言是由一条一条语句构成的，这些语句通常被称为机器指令。

3.5　汇编语言和高级语言

用机器语言写的程序就是一堆 0/1 串。这样的程序读起来会相当的费劲儿。如果不小心把哪条指令的 0 和 1 抄错了，那可就惨了：就慢慢在这些 0/1 串中去找错误吧。而且相信我们离开指令表，一条程序都写不出来。程序员不可能记住那么多指令对应的 0/1 串，比电话号码还难记呢。因此，早期的设计人员就给这些指令起了一些好记的名字。比如，01010101 表示加法，那么就用 add 表示（早期的设计人员不是中国人，不然就用 jia 了）。他们把像 add 这样的符号叫做助记符。这样，程序员写程序时，就直接写 add 而不写 0101010

了。这既容易记住又不容易出错。但问题是译码器看不懂了。所以，设计者就专门写了一个程序来把程序员用助记符写的程序翻译成机器指令。这个翻译的过程被称为“汇编”。完成汇编工作的程序就叫汇编程序。所以，这些助记符也叫汇编语言。用汇编语言写的程序则称为汇编语言程序或者汇编语言源程序。

图 3-4 是一段汇编源程序，它的任务是计算几个数的平均值。看起来是不是感觉有点儿晕？但是想象一下把它们换成 0/1 串，相信你会晕得更厉害。如果想运行这个源程序，则首先要对其进行汇编。假定你把这段代码存储在一个叫做 count. asm 的文件中(汇编源程序文件要求以. asm 为后缀)。现在要做的事情就是汇编。假定我们使用的汇编程序的名字叫做 masm. exe。我们在 Windows 的命令行中输入 masm count. asm 命令。就会生成一个叫做 count. obj 的目标文件。这个文件中的内容就是机器指令了。但是，源程序中可能会引用其他库文件中的内容。库文件中一般是一些经常用到的功能的代码段，专门有人把它们写好，这样当你的程序需要这些功能时，就不用自己再写代码，直接引用就行了。不过这就还需要一步连接目标文件和库文件的过程。这用连接程序 link 完成。最后生成一个可执行文件，也就是计算机能够读懂的程序。我们就可以输入文件名，让计算机来执行这些指令了。图 3-5 描述的就是这样一个过程。

小提示

机器指令的助记符形成汇编语言。
把汇编语言翻译成机器语言的过程称为“汇编”。完成汇编工作的程序称为汇编程序。
用汇编语言写的程序称为汇编语言源程序。

汇编语言虽然比机器语言好记好用，但是用起来仍然不太方便。毕竟它是机器语言的助记符，和我们日常用的语言区别很大。比如，算几个数的平均数，我们希望写个算式就行了。为此，人们又设计了高级语言，它更接近于人类的语言(实际上是更接近英语。因为早期的这些开拓者用英语的多)。在高级语言中我们就可以写算式，直接写加减乘除符号等。高级语言有很多种，比如 C、Java、COBOL 语言等，可以说快要数不胜数了。高级语言因为更接近于人的理解，所以离机器也就远了。用高级语言写的程序也需要特殊的程序把其翻译成机器语言，这些翻译程序叫做编译程序或者解释程序。编译程序将高级语言源程序全部翻译成机器语言后保存下来，然后再执行编译产生的可执行程序。这样一个源程序只需要编译一次，以后都执行这次编译产生的可执行程序。C 语言的编译器就是编译程序。而解释程序则把源程序一条语句翻译成机器指令后就立即执行这条指令，然后再翻译执行下一条。解释程序并不把翻译的结果保存下来，所以下一次再要执行这段代码时，还要逐句翻译再执行。因此解释执行的程序速度要比执行编译后的程序要慢。

图 3-6 是求几个数的平均值的 C 语言源程序。看起来比汇编语言的短多了。要想让这段代码被计算机执行，需要对它进行编译。在 Linux 操作系统下进行 C 语言编译的程序叫 gcc。假定这段代码被我们存储在一个名叫 count. c 的文件中(C 语言的源程序要求以. c 为后缀)。我们用 gcc 对它进行编译。编译之后生成一个可执行文件叫 a. out。现在你就可以输入 a. out 运行它，让计算机执行其中的指令了。C 语言的源程序编译后也需要有个链接的过程，只不过 gcc 一起给做了。图 3-7 描述了上述过程要用到的命令。

```
.model small
.stack
.data
        score dw 90,95,54,65,33,55,11,-1
        average dw 0
.code
.startup
        xor ax,ax
        xor dx,dx
        xor cx,cx
        lea si,score
again:
        mov bx,word ptr[si]
        cmp bx,0
        jl over
        add ax,bx
        adc dx,0
        inc cx
        add si,2
        jmp again
over:
        jcxz exit
        div cx
        mov average, ax
exit:
.exit 0
end
```

图 3-4　汇编语言源程序

```
C:\>masm count.asm
C:\>link count.obj
C:\>count.exe
```

图 3-5　汇编语言源程序的汇编

```
#include <math.h>
main()
{
      int a[7],i;
      float av=0;
      for(i=0;i<7;i++)
      {
        scanf("%d",a+i);
        av+=a[i];
       }
       av=av/10;
       printf("%f, \n",av);
}
```

图 3-6　C 语言源程序

```
root#> gcc count.c
root#> a.out
```

图 3-7　C 语言源程序的编译执行

因为汇编语言是和机器语言对应的，所以不同 CPU 支持的汇编指令集合也不同。现在直接用汇编语言写的程序不是很多。但是高级语言的效率要比汇编语言低一些。出于各种考虑，编译程序在编译高级语言写的代码时产生的目标程序会比较庞大，难以优化，运行速度较慢。所以，在一些对速度和空间有严格要求的环境下，汇编语言仍然被广泛使用，如各种工业自动控制用芯片的编程，以及操作系统中底层与设备相关的功能模块。

小提示

高级语言更接近人类语言。
高级语言编写的程序需要用编译或者解释程序翻译成机器语言。

复习题

1. 什么是进位制？什么是进位制的基数？
2. 二进制数每位上的位权是多少？
3. 十进制数怎样转换成二进制表示？
4. 十六进制数怎样转换成二进制表示？
5. 八进制数怎样转换成二进制表示？
6. 每一个二进制数都能准确转换为十进制数吗？每一个十进制数都能准确转换为二进制数吗？如果能，为什么？如果不能，在什么情况下不能？
7. 二进制主要逻辑运算有哪些？它们的运算规则是什么？
8. 什么叫原码？原码有什么优点和缺点？
9. 什么叫补码？为什么现在很多计算机中都使用补码表示数？
10. 怎样求一个负数的补码？
11. 什么是 ASCII 码？
12. 区位码、国标码和机内码有什么关系？
13. Unicode 和 UTF 有什么关系？
14. 计算机怎么知道一个内存单元中的 0/1 串应该解释成指令还是数据？
15. 计算机解释数据时，怎样决定存储单元中 0/1 串应该解释成数字还是字符？
16. 什么是机器语言？
17. 什么是汇编语言？汇编程序和汇编语言程序有什么关系？
18. 什么是高级语言？计算机怎样读懂高级语言？

练习题

1. 16 位补码能够表示的整数的范围是多少？
2. 将下列十进制数转换为二进制表示、八进制表示、十六进制表示(如果不能精确表示，保留小数点后 4 位)。

256　63.25　－1022　16.321

3. 将下列二进制数转换成十进制表示、八进制表示、十六进制表示。

10101110　0111010001　01101101　11111111

4. 将下列十进制整数用 16 位原码表示出来。

789　－543　－128　4095

5. 将下列十进制整数用 16 位补码表示出来。

789　－543　－128　4095

6. 求下列二进制补码表示的十进制整数。

10010001　01110011　11101101　11000000

7. 求下面两个数的 16 位补码，并用补码计算两个数的和、差。

－9837　3456

8. 完成下列二进制运算。

11011101＋11101000　11010010－01110011
101×1101　11110101/101
10101010&11010011　10010101|01010110
～10101101　11101101⊕10100011

9. 对下列 ASCII 码进行译码。

01001110　01101001　01101000　01100001　01101111　00100001
01001010　01101001　01100001　01111001　01101111　01110101　00100001

讨　论

1. 请用 8 位补码计算下列两个数的和，是否能够得出正确的结果？为什么？

$$X=-89 \quad Y=-120$$

2. 请设计用原码进行乘法运算的规则。

3. 补码、原码在负数的表示上不同，在正数表示上相同。如果已知程序中要使用的数据都是正数，你认为用哪一种编码比较合适？是否需要另外设计一种编码？如果需要，请说明原因。

第三篇　计算机软件平台

实际上人们日常很少直接接触计算机内部的硬件平台。从计算机系统层次图上这一点体现得非常清楚。在第1章我们对硬件上的第一层软件平台——操作系统有了一个初步认识。因为操作系统这个平台非常重要，对它的认识直接决定我们使用计算机系统的效率，所以，这一篇我们将进一步了解操作系统的知识。我们将深入到操作系统的内部，去了解这个平台实现的机理，了解它为什么能够帮助我们做那些事情，从而初步了解它的各种外在表现背后的原理。这些知识能帮助我们更有效地使用这个操作平台。

在对操作系统的内部机制以及硬件平台有了足够认识的基础上，本篇还将讲述在计算机启动时发生的一系列内部过程。然后再告诉你怎样通过 Shell 脚本这个操作系统提供的另外一个用户接口来控制计算机。

操作系统是人们操作计算机的平台，也是应用程序的运行平台。但是，对于创造应用程序员的用户来说，只有操作系统平台还是不够的。他们需要有一个能够帮助其开发应用程序的平台。为此，在计算机的发展进程中，逐渐形成了应用软件开发和运行平台。不同类型的应用可能有不同的开发平台，它们本身属

于计算机软件的范畴，向下依托操作系统，向上为应用程序员提供各种软件开发上的便利，以及应用软件在运行时的支持。这个平台包括帮助程序员编辑、编译、调试程序的软件工具，还提供一些可重用的成品或半成品的软件构件。在这个意义上，图 1-9 是比较简化的，在操作系统和应用程序之间还应该有一层，即这里说的“应用软件开发和运行平台”。有时我们听说的“中间件”基本上指的就是这样的平台。本篇的最后将简要介绍应用软件开发平台。

第 4 章　操作系统的内部实现机制

从前面的介绍我们知道了操作系统平台的功能。那么操作系统是怎样实现这些功能的呢？了解操作系统做事的方法和策略，对我们更好地使用这个操作平台非常有帮助。因此，本章就来介绍操作系统的内部实现机制[2-13]。

操作系统所做的最基本的事情是使终端用户能够在计算机硬件上运行应用程序。只要用户输入程序名或者单击程序名就可以运行程序。别看这么简单的一个动作，操作系统为此可做了非常多的事情。为让程序运行起来，操作系统需要把存储在外存（硬盘）上的程序找到并读取到内存中，然后分配 CPU 让其执行该程序。具体来说，这里涉及的工作基本上可以分为四大块：信息存储管理、内存管理、进程管理，当然还有外设管理。因为肯定会涉及外设，至少要使用外存，那可是个地地道道的外设。

操作系统中负责管理信息存储的部分也被称为文件系统。文件系统主要做的事包括显示外存上存储的文件；用户选定文件后，把文件从外存中找到并读出来。目前常用的内存在没电时是不能保存信息的。因而，文件或者说程序都是保存在外存上的。但是外存的存储速度比内存要慢多了。为了弥补外存与 CPU 之间巨大的速度差距，要运行的程序都要先从外存读入到内存。这个读入过程就由操作系统负责完成。到外存什么地方读取程序由文件系统负责。而读后放在内存中什么地方，就由操作系统的内存管理部分来负责。那么进程管理指的又是什么呢？进程管理负责在合适的时候让 CPU 执行已经读入内存的程序。这里面涉及一个概念，就是“进程”。这是个与程序动态执行、一个系统中同时有多个程序在运行，以及 CPU 分配相关的概念。我们马上就会详细说明这个概念。总之，操作系统为了让多个程序能够同时在计算机中好好运行（这几乎是现在每个操作系统都必须支持的功能）就必须管理进程。至于外设管理部分当然是负责同外设打交道了。文件管理部分要访问硬盘上的信息就必须找操作系统的外设管理部分来帮忙。

下面我们就从进程管理开始介绍操作系统做这些事情的过程和策略。

4.1　进程的管理

4.1.1　进程概念的引入

前面已经讲过，为了提高计算机硬件的使用率，尤其是 CPU 的利用率，操作系统都采用了多道程序技术。在这种技术下，当一个程序需要使用或者等待外设时，操作系统会让 CPU 去执行别的程序。这时第一个程序并没有执行完毕，它可能在等待或使用外设。与此同时，另外一个程序也正在 CPU 上运行。这里的另外一个程序甚至可以与刚刚让出 CPU 的那个程序一模一样。例如，如果你连续两次双击了桌面上的 QQ 图标，就会出现这种情况。即你启动了两次 QQ 应用程序。

现在计算机中有多个程序在执行了。我们来想想，操作系统为此要多做哪些事情。首

先，当一个程序因为某些原因不能继续执行而闲置 CPU 时，操作系统应该有一种办法知道这件事情。其次，当 CPU 被让出来之后，操作系统应该决定接下来让哪个程序来使用 CPU。如果是要执行一个新程序，那么操作系统就要把新程序读入到内存中的某个地方，而且要保证新读入的程序不会破坏前面还没有执行完、仍在内存中的程序及其相关数据。如果接下来要执行前面没有执行完的某个程序，那么操作系统要在内存中能够找到这个程序，并且还要知道之前执行到了哪里，即接下来该执行哪条语句。为了完成这些工作，操作系统必须记录一些必要的信息。从前面分析可以看出，至少要记录的信息包括：每个程序在内存中存放的位置，每个程序将要执行的指令的地址，还有系统中正在执行的程序是什么，以及系统中有哪些程序在等待获得 CPU 来执行。

我一直在用“程序”这个词来描述这些事情。如果仔细分析，你会发现在有些情况下，这会带来一些混乱。先来回想一下程序是什么？程序是一段指令代码。这段代码被执行后能够产生一些结果。这段代码可以被执行无数次，不仅是一次结束后可以再来一次，而且可以在上次还没有结束时就启动下一次，造成程序的多次执行同时存在的情形。当然，如果每次执行时的初始条件都一样，那么执行的结果也应该是一样的。那么用“程序”这个概念描述多个程序在计算机中执行的情景会有什么问题呢？我们来看个例子。假设你写了一个小程序。程序的功能很简单：获取计算机系统维护的当前时间；把它换算成一个整数，在屏幕上显示出来，并让用户输入这个数的质因数；然后等待用户输入；最后判断用户输入的结果是否正确并显示。你把这个程序编译链接后生成了一个可执行文件。你给程序起的名字叫“时间与智慧”。然后，你开始运行这个程序。你在非常短的时间间隔内先后两次双击屏幕上显示的“时间与智慧”那个图标。那么这个程序就被启动起来，而且启动了两次。因为你的计算机只有一个 CPU，所以操作系统会首先读入第一次单击的程序的代码，并且开始执行。当执行到等待用户输入质因数的应答时，程序需要等待键盘的输入。这时，操作系统并不让 CPU 去等待，而是让它读取第二个启动的程序并执行（其实计算机的时间片比这短多了，只不过为了说明方便，我们假定时间片非常长，所以才会让第一个程序一直执行到因等待外设而让出 CPU）。计算机同样将这段程序读取到内存中一个区域，并保证它不会破坏先前读入的程序。这个程序再次执行到等待键盘输入而被操作系统收回 CPU。此时恰好有一个键盘输入。这时操作系统的麻烦事情就来了：它把这次输入的结果是交给第一次启动的程序还是第二次启动的程序呢？如果操作系统用程序名字来区分这两次程序的执行，显然是达不到目的的。因为它们都叫“时间与智慧”。所以，操作系统必须能够区分同一程序在系统中的两次执行。用程序名不行的原因在于程序名对应的是一份代码，是静态的。无论程序执行多少次，代码都是这一份，所以用一个程序名就足够标识这份代码了。而操作系统现在关注的是程序的执行过程，它是动态的。一份程序可能在系统中存在多个执行，而且在某个时刻每个执行过程所执行到的指令位置可能不一样，所产生的中间结果也可能不同。所以，我们需要一个新的概念来描述程序的执行过程。为此，人们引入了“进程”的概念。

“进程”的英文是 process。这个英文单词的原意就是过程的意思。现在操作系统不再用程序来描述正在执行的代码了，而是用进程。程序的一个执行过程就对应一个进程。那么像前面例子中“时间与智慧”的两次执行在操作系统中就对应两个不同的进程。操作系统给它们分配两个不同的名字（通常是用号码）以示区别。这样当有个键盘输入到达时，操作

系统可以根据这个输入框所属的进程号码找到正确的进程，程序就能正确执行了。

我们用炒菜的例子再来说明一下进程的概念。程序就好像菜谱，而进程就好比照着菜谱炒菜的过程。假设你找到了一个鸡蛋炒柿子的菜谱。现在你打算用两个锅同时来做两份鸡蛋炒柿子(因为你的锅实在太小了。当然，用炒不同的菜的示例同样可以说明问题，就好像执行不同的程序，这是更普遍的情况。但是，为了更清楚地看到引入进程概念的必要性，我们选用同一个菜谱)。你往第一个锅放上油后，接着往第二个锅里继续倒入少许油。这时第一锅中的油冒出了刺鼻的烟味。你赶快查看菜谱，发现要放入搅拌好的鸡蛋。于是你找到鸡蛋，飞快地倒入第一个锅中。在整个炒菜过程中你要记住两道菜各炒到哪一步了。也就是说当炒菜过程进行到菜谱上的“向锅里加少许盐”时，你要知道是往哪一个锅里放。除此之外，你还要决定什么时候你该关注哪一个锅。这取决于你当前处理的那道菜是否能够暂时离开你等很多因素。如果两个菜同时都需要你，那么你就要有个处理轻重缓急的原则策略。在这个例子中，你的手就好像是计算机的 CPU，而你的大脑就是操作系统。从这个例子，你可以体会出程序和进程有多大的不同。

现在通过你的大脑在炒菜过程中做的事情来分析操作系统为支持多道程序技术而需要做的事情。首先，为了能够让你的手在两个炒菜过程之间有效地切换，你的大脑必须给两个炒菜过程起个不同的名字来区分，比如分别叫做第一锅、第二锅，或者左锅、右锅。此外，它还要记住两道菜都分别进行到菜谱中的哪一步，比如第二锅该放盐了，第一锅该出锅了。同样操作系统也要做类似的事情，它要给每个进程起标识号，然后要记录每个进程接下来要执行的指令的地址。另外，两道菜的原材料要分开放，比如，第一锅的原料放在左边，第二锅的放在右边。不能混放，以防一道菜的材料放到另外一道菜中。同样，操作系统要保证两个进程的程序和数据在内存中不能互相重叠。

操作系统把与进程有关的信息都组织在一起，放在一个称为进程控制块的数据结构中。进程控制块存储在内存中特定的位置。当用户要求执行某个应用程序时，操作系统就会创建一个进程控制块。进程控制块中除了前面讲到的内容外，还有很多其他信息，如进程是由哪个用户启动的、它都使用了什么资源、它正在做什么等。我们将在后续小节中逐步说明这些内容。有了进程的概念以及进程控制块这个描述进程的数据结构，操作系统控制整个系统中多道程序的执行就容易多了。

图 4-1　进程控制块

图 4-1 是进程控制块的示意图。里面简要列出了一些比较重要的信息项。图 4-2 是 Linux 中任务结构 task_struct 的定义。Linux 为每个进程维护一个 task_struct 类型的数据结构，它实际上就是 Linux 的进程控制块。

小提示

进程是程序在计算机中的一个执行过程。
程序是静态的，而进程是动态的。
操作系统用进程控制块来存放进程有关的信息。
每个进程都有一个进程控制块。

```
struct task_struct {
        volatile long state;    /*-1 unrunnable, 0 runnable, >0 stopped */
        unsigned long flags; /* per process flags, defined below */
        ⋮
        volatile long need_resched;
        ⋮
        long counter;
        long nice;
        ⋮
        struct list_head run_list;
        unsigned long sleep_time;
        ⋮
        pid_t pid;
        ⋮
/* filesystem information */
        struct fs_struct *fs;
⋮
/* signal handlers */
        spinlock_t sigmask_lock;    /* Protects signal and blocked */
        ⋮
};
```

图 4-2　Linux 的 task_struct 结构

解决了怎样在系统中描述多道程序的问题，接下来看看为了支持多道程序，操作系统具体需要做哪些事情。首先想到的第一件事就是操作系统要决定在某个时刻应该执行哪个进程。这个工作被称为进程调度。因为两个或多个进程可能会同时需要同一个资源，如打印机或者某一个数据，所以操作系统需要仲裁让哪个进程使用资源。这就是第二个问题，被称为进程间资源的竞争与共享。此外，如果多个进程都要访问同一个数据，如一个进程要更改系统时间，而另外一个进程要显示系统时间，如年月日、时分秒，那么操作系统必须保证第一个进程把年月日、时分秒全部更改完毕，才能让第二个进程显示，不然就出问题了。这就是进程的同步与互斥。另外，有时候进程之间需要通信。例如，一个进程在结束前想把自己的退出原因告诉给另外一个负责监控它的进程。那么，操作系统必须为这两个进程提供通信手段。这么一想，为了支持多个进程的并发执行(多个操作在同一时间间隔内发生的性质称为并发性)，操作系统要做的事情还真不少。下面就来了解操作系统是怎样做这些事情的。

4.1.2　进程的状态

既然系统中有多个进程，操作系统就要知道它们都在干什么，这样才可能管理和安排它们的工作。也就是说操作系统需要了解进程当前处于什么状态。为此，我们来看看将进程的活动归纳成哪些状态就便于操作系统管理了。

首先，被操作系统创建出来的进程可能获得 CPU 正在运行。我们把这种状态称为“执行”。这时其他进程或是在等待 I/O 设备，或已经准备好了在等待 CPU。我们称那些等待 I/O设备的进程处于“阻塞/等待”状态。那些万事俱备只差 CPU 就可以执行的进程则被称为“就绪”。

为了在 CPU 空闲时能够很快地找到合适的进程接着运行，操作系统把处于就绪状态的进程排成一队。当 CPU 空闲出来时，操作系统就在就绪队列中取出一个进程让它运行。处于阻塞状态的进程也被排成一队。当一个正在执行的进程需要等待 I/O 操作时，操作系统就把 CPU 分给就绪队列中的某个进程，让刚才执行的进程到阻塞队列中排队。当某个 I/O 操作完成时，操作系统就从阻塞队列中查找等待这个设备的进程，把它从阻塞队列中取出，让它到就绪队列中去排队等待 CPU 以便继续运行。多数操作系统会为每种常用的设备维护一个等待队列，也就是说等待进程会根据自己等待设备的不同而排在不同的队列中，并不是都排在一个等待队列中。这样当某个设备工作完成时，系统直接到相应的等待队列中取进程就行了。

为了防止某个进程执行起来没完没了，而使其他进程等待的时间过长，一般操作系统都会采取分时机制，为每个进程分配一个可连续占用 CPU 的时间，称为时间片。时间片用完，尽管进程并没有结束，也不是要等待什么 I/O 设备，但操作系统也不让该进程继续执行，而是让其到就绪队列去重新排队，并把 CPU 分给就绪队列中的第一个进程，让它接着从上次停下来的地方继续执行。

系统中新进程的来源有很多。例如，新用户登录时，系统要为用户创建一个进程。当用户运行了某个应用程序时，操作系统也要为这个应用程序创建一个进程。这些进程都是操作系统自动为用户创建的。操作系统还提供系统调用，让用户的应用程序自己创建新进程来完成其任务中的某一个部分。例如，一个监控网络状态的应用程序可以创建一个进程专门负责收集网络的状态数据并放到数据库中，而操作系统自动为这个应用程序创建的进程则只负责从数据库中读取状态信息来显示。又比如，一些服务器运行时，系统为其创建了一个进程。当有客户向其请求服务时，这个进程就会创建一个新进程来为该用户服务，而它自己则又去查看是否有新客户需要服务。它就好像一个接待员，而具体的服务都是由它创建的进程完成的。如果进程 A 创建了另外一个进程 B，那么通常把 A 称为 B 的父进程，而把 B 称为 A 的子进程。操作系统在创建进程时，要为进程创建一个进程控制块以及相关的数据结构。在做完准备工作后，操作系统会把新创建的进程放到就绪队列中排队。

当一个进程执行完其代码的最后一条语句，它就结束了。当系统的一个用户注销时，该用户的进程也将被终止。当一个用户退出应用程序时，系统也把该进程终止。导致进程被终止的原因还有很多，如内存不够、对数据的不正确使用等。在有些系统中，如果父进程被

终止，那么子进程也会被终止。操作系统会收回被终止进程的资源。

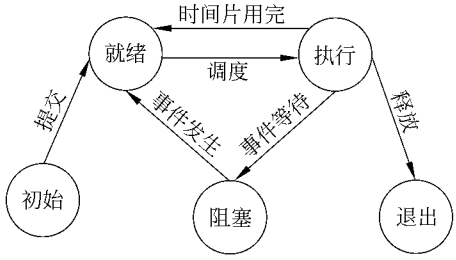


图 4-3　进程的三种基本状态图

“执行”、“就绪”、“阻塞”构成了进程最基本的三种状态。从前面的分析我们可以得出这三种状态之间的转换图，如图 4-3 所示。图中圆圈表示进程的状态，箭头上标注的是引起状态转换的事件。

每个进程的状态被保存在它的进程控制块中。进程排队是怎样实现的呢？进程怎么会自己去排队呢？这只是一种形象的说法。在实现上其实就是操作系统对进程控制块进行组织。比如，让进程到就绪队列中排队，实际上就是把进程控制块放到一个链表中。这个链表中所有进程的状态都是就绪。

链表是什么？链表是一类特殊的数据结构。在数据结构课程中会有详细介绍。链表中通常是一些结构相同的结构体数据。把一些相关的信息组织在一起，用一个名字来标识，就称为结构体。例如，一个学生全部的信息，如姓名、成绩等，放在一起就可以形成一个结构体数据。我们可以把每个结构体数据理解成一张卡片。进程控制块就是一个结构体。在这些结构体中有一个特殊的项，它可以存放一个地址。这个地址是另外一个一模一样结构体在内存中存放的地址，如另外一个进程的控制块的地址。通过这个地址就可以找到另外一个进程控制块。就好像这个项指向了另外一个进程控制块，所以我们把这样的项叫做指针。指针就好像给每张卡片下面加了一个钩，可以用来挂下一张卡片。指针是一种数据类型。我们前面已经说过，内存单元中存放的数据必须事先声明它的数据类型。当 CPU 看到一个声明为指针类型的数据时，就知道这个数据的内容是个地址。这样每个进程控制块中这个项都放着另外一个进程控制块的地址，那么我们只要知道第一个进程控制块中的地址，就可以通过它找到第一个进程控制块，然后就能够从控制块中找到下一个进程控制块的地址。这些链起来的进程控制块就好像是在排队了。图 4-4 是 Linux 中就绪队列的示意图，图 4-5 是该队列的 C 语言定义。

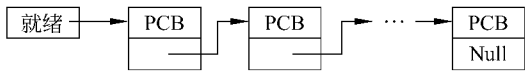


图 4-4　就绪队列

```
struct list_head {
    struct list_head *next;
};
struct task_struct {
    ⋮
    struct list_head run_list;
    ⋮
}
```

图 4-5　Linux 的就绪进程列表

其实系统中的进程不只有这么三种基本状态。操作系统采用的一些管理策略有时会产生一些新的进程状态。例如，下面的策略就增加了一个进程状态。我们先来看这个策略使用的背景。在多道程序系统中，内存中有多个进程。内存大小是有限的。在一定时候就会出现这种情况：系统想再读入一个程序来运行，即创建一个新进程，而内存中已经

没有地方了。这时如果把系统中暂时不能执行的进程从内存中清除，就有空间了。可是这些进程都已经执行一部分了，如果直接把它们扔掉，那么前面的时间就浪费了。最好的办法是把这些进程的控制块以及已经产生的数据先放到外存中。这被称为进程的换出。这样进程在内存中占用的空间就被腾出来了。要知道它让出来的空间可不只是刚刚存储到外存的控制块和数据所占的地方，还有被读入内存要执行的代码。只不过这些代码本来就在外存中，所以换出进程时不用复制这些内容。等到内存中有空闲空间时，系统再把这些换出进程读进内存，并根据情况分配资源。它们就可以从上次暂停的位置继续执行了。这里换出的进程就被称为处于“挂起”状态。通常系统会选择那些处于阻塞状态的进程挂起，因为它们在等待 I/O 设备，即使获得 CPU 也不能执行。当没有阻塞进程时，系统再从就绪的进程中选择。

因为挂起进程的相关信息都在外存，所以不可能直接获得 CPU 进入执行状态。它们只能在重新装入内存后才能被重新调度。挂起进程重新被装入内存的过程被称为“激活”。在激活挂起进程时，当然也是要优先选择装入内存中就能处于就绪状态的进程，而不是那些还会继续阻塞的进程。为此，挂起进程的状态又被划分成两种。那些进入内存就能处于就绪状态的挂起进程被称为处于“挂起就绪状态”，而还在等待某类其他事件的挂起进程称为处于“挂起阻塞状态”。当挂起阻塞进程等待的事件已经发生，它就转入挂起就绪状态。显然挂起就绪状态的进程被激活后马上转为就绪状态，而挂起阻塞状态的进程被激活后只能处于阻塞状态。就绪进程被挂起后即处于挂起就绪状态，而阻塞进程被挂起则处于挂起阻塞状态。图 4-6 是增加了挂起状态的进程状态转换图。系统也可以因为某种原因把正在执行的进程挂起，这时进程也转入挂起就绪状态。

计算机出现初期，系统为程序执行创建进程时，需要将整个程序都读入内存。后来应用程序随功能增强而变得越来越大，内存也变得越来越紧张，就产生了挂起技术。现在操作系统创建进程时不再把整个程序都读入到内存（在内存管理的介绍中会讲这是怎么回事），而且计算机的内存也比以前大了很多。所以现在很少再因为内存不足导致进程挂起了。但是，挂起这个概念仍然有发挥作用的场合。例如，你设置了一个程序，它负责在每天半夜把你的数据都备份一遍。当这个进程完成今天的工作后，它再需要运行的时刻是在 24 小时之后，那么在这 24 小时之内就没有必要把它放在内存中占地方了，所以可以把它挂起。另外在调试程序时，我们有时也需要把一些进程挂起，以便分析它的一些数据。

除了上述状态之外，个别操作系统还会根据需要增加一些状态。比如，Linux 操作系统就设置了一个“僵死”状态。处于“僵死”状态的进程已经执行完毕，但是系统并没有删除它的进程控制块和相关信息，也就是进程的尸体还在。一些需要了解该进程信息的其他进程可以来查看这些信息，然后把这些信息清除。这时僵死进程才真正退出系统。图 4-7 是 Linux 操作系统的进程状态转换图。

小提示

就绪、执行、阻塞是进程的三种基本状态。
各种操作系统会根据自己的策略增减进程的状态。

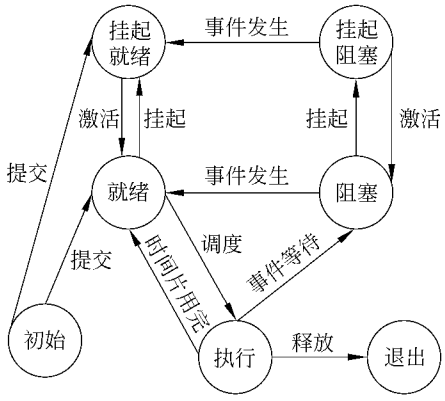


图 4-6 进程的五状态图

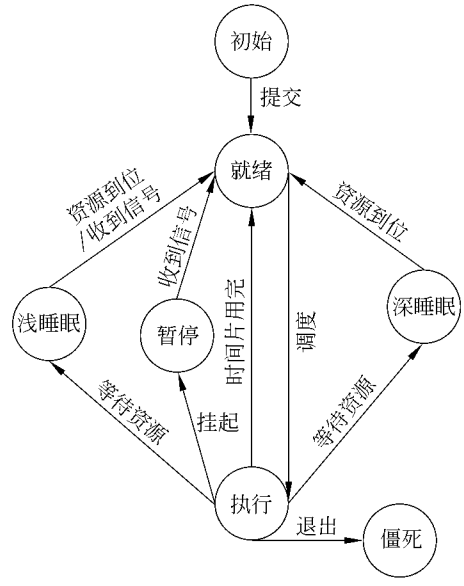


图 4-7 Linux 的进程状态图

4.1.3 进程的调度

既然系统中同时存在处于各种不同状态的多个进程，那么操作系统就需要小心地指挥这些进程。既要保证整个系统的效率，又要尽量让每个进程都满意。

当 CPU 空闲出来时，操作系统要考虑让哪个进程来运行。这项工作被称为进程的调度，有时也叫做处理器调度。最简单的办法就是在所有就绪进程中随机选一个来执行。你可能感觉这种随机抽取太没有确定性了，怎么也应该是让先准备好的进程先得到 CPU，也就是最早来就绪队列排队的进程应该最先运行。这样才公平。我们把这种策略叫做先来先服务。这种方式看起来很公平，但不一定合理。例如，一个进程只需要做很少的工作，如把桌面背景换一下，但在它被创建之前系统刚刚创建了另外一个计算进程，后者需要进行相当复杂的运算，大概需要工作半个小时。按照先来先服务的原则，CPU 空闲时，应该运行这个计算进程，那么后边那个只要运行一秒钟不到的进程就需要等待至少半个小时。这显然让后面的进程感觉有些不舒服。这就和在银行排队交电话费一样，你只需要把钱交上打印个收据就行了。但是不巧你前面的人手里拿了数不清的单据来处理。

分析先来先服务策略，我们发现产生对短进程不公平的现象主要是因为进程获得 CPU 后就一直执行到结束(除非进程遇到 I/O 操作或者主动让出 CPU)。我们把这种 CPU 分配方式叫做非剥夺式。其实这个问题很好解决，前面也提到过，就是让每个进程执行一个固定的时间。如果进程在分给它的时间内没有执行完，那么它也必须让出 CPU 到就绪队列重新排队。系统取出排在就绪队列最前面的进程，让它来运行。这就是轮转式调度策略。这种在进程没有执行完就把 CPU 收回的 CPU 分配方式被称为可剥夺式。有了轮转调度法，现在短进程就不用怕前面的长进程了。当长进程用完自己的时间片后就要到就绪队列末尾重新排队。这样短进程就有机会迅速地执行了。

现在看来事情解决得差不多了。可是各个进程的情况都不一样，例如，前面出现的问题就是由于进程执行时间长短不同造成的。所以，我们还要仔细勘查一下进程之间的区别。我们发现有些进程会经常使用 I/O 设备，而有些进程则使用 CPU 的时间长一些。如果采用轮转法，经常使用 I/O 设备的进程在刚刚获得 CPU 后，只执行了几条指令，就遇到了 I/O 操作，结果不得不让出 CPU 去执行 I/O 操作，等 I/O 操作结束后再重新排队。这些进程在再次得到 CPU 后，又一次在只用了一点点儿时间片的时候让出 CPU 去访问 I/O 设备，如此往复。这就好像你在银行办事时好不容易排到窗口，里面的人给了你一张复杂的表格让你拿到旁边去填。等你填完了，又要到队尾重新排过。当你再次到达窗口，递上填好的表格后，业务员又给了你另外一张表格让你去填。然后你再次重新排队。这样如此往复，经过几个回合后，我猜你肯定会崩溃的。那么这个问题怎么解决呢？好多银行其实已经有很好的解决办法。通常填好表格后，你并不用重新到原来的队伍去排队，而是在另外一个队伍里和所有像你一样填表的人一起排队。业务员会优先服务这个队伍，也就是说只要这个队伍中有人，那么他们就先为这些人服务。理由很简单，因为这些人都是在另外一个队伍中排过队的。有些操作系统也采用了这种策略。除了就绪队列外，操作系统再组织另外一个队列，那些没有用完时间片就因为 I/O 操作让出 CPU 的进程在执行完 I/O 操作后，就进入这个队列。只要这个队列中有进程，CPU 就会分配给它。和银行例子不同的是，操作系统会把每个进程没有用完的时间片记下来。获得 CPU 的进程在用完剩余的时间片后必须到就绪队列中重新排队。这样可以避免前面先来先服务策略中长进程占用 CPU 的问题。

在人人平等的条件下，前面的调度策略可以说工作得不错了。但是，凡事都有个轻重缓急。在某些情况下，是不能一味强调平等的。比如，公路上的汽车具有平等使用道路的权利。但是，消防车是特例，它拥有优先权，因为它做的事情很重要，耽误不得。在计算机中也是这样，有些进程做的事情必须马上做，比如报时程序。操作系统要能够支持这种需求。为此，有些操作系统会为进程定优先级别，如 1，2，3 级等。假定 1 级最高，那么当有 1 级进程要运行时，优先级为 2 的进程必须让出 CPU。同一级别的进程则可以采用先来先服务或轮转策略。使用这种策略时，操作系统通常会让优先级相同的进程在一起排队，即为每个优先级组织一个队列。分配 CPU 时，先看最高优先级队列中有没有进程，如果没有再查看下一级队列。我们把这种策略称为优先级策略。你一定会问，如果高优先级队列中始终有进程，低优先级的进程不是永远得不到机会了？你问得很对，这种现象被称为饥饿。为了避免进程被饿死，必须对前面的策略做些调整，采取一些措施让低优先级的进程能够有机会运行才行。例如，可以让进程的优先级和等待时间挂钩，当低优先级的进程等到一定时间时，它的优先级就升高了。这样它总会等到获得 CPU 的时候。你还能想到什么办法解决这个问题吗？

实际的操作系统在调度进程时，都是根据自己目标和应用环境，采用上述策略中的一种或把几种结合在一起，也有些会使用其他更合适的策略。

和调度策略有关的一些信息也需要保存在每个进程的进程控制块中，如进程的优先级、进程剩余的时间片数等。

操作系统根据一定的策略选择进程是需要时间和空间的。各种调度策略的具体体现就是操作系统中的有关代码。这些代码的执行需要时间。这些代码以及用到的相应数据结构

都要放在内存中。当进程让出 CPU 后，如果它还没有执行完，为了保证它能够再次继续正确地执行，操作系统必须把这次执行的一些有关事项保存下来。例如，一些寄存器，如程序计数器的内容必须保存下来，因为后面的进程会更改它们的值。还记得程序计数器里面放的是什么吗？是下一条应该执行指令的地址。只有把这个值保存起来，才能保证进程下一次能够从正确的指令继续执行。还有状态寄存器的内容，它记录着进程执行到当前位置时产生的一些情况，如是否有进位、是否可以中断等。这也必须保存起来。这些寄存器的内容被保存在进程控制块中或者与当前进程有关的一些内存区域中。这个保存过程被称为进程上下文保存。其实就是保护现场。同样，被选定接下来要执行的进程必须要在上次其被切换下来时的环境中重新执行才行，所以要恢复为它保存起来的上下文。也就是要重新装入以前保存起来的寄存器内容及相关数据。它不能使用刚刚切换下去的进程的环境。这个过程被称为恢复上下文。如果是新进程，没有旧的环境需要恢复，那么就根据需要设置这些寄存器的值，总之不能使用前面进程遗留的寄存器内容。保存和恢复现场的过程也被称为上下文切换。上下文切换涉及读写内存的操作，所以它的时间开销也是不容忽视的。如果操作系统设计得不好，进程频繁地切换，那么很多时间都浪费在进程切换上，计算机就不用干活了。

小提示

CPU 的分配方式分为剥夺式和非剥夺式两种。
先来先服务、轮转法、优先级法是常见的进程调度策略。
进程切换会产生时间和空间上的系统开销。

4.1.4 进程的同步与互斥

支持多道程序，系统的效率提高了。但是进程多了，麻烦事也多了。就好像人手多了，事情就做得快了，可是事情也多起来，尤其是在工具不是很充裕的情况下。组织者必须协调干活的人对工具的使用顺序，才能保证效率，保证不出问题。同样，操作系统除了要保证系统中多个进程有序地运行，充分提高系统的效率外，它还要保证这些进程之间不能因为共享或者竞争资源而造成问题，尤其是要保证程序的运行结果是正确的。前面讲到的上下文切换就是为了达到这个目的而进行的工作之一。但是，只保证进程运行环境互相不影响是不够的。下面我们就来看看还有什么情况需要操作系统协调。

系统中有很多资源，每个进程都可能使用。例如，编辑器需要打印机来打印你的新作，而照片查看软件也需要打印机来打印你的照片。这时操作系统必须保证编辑器进程没有用完打印机之前，照片查看进程只能等待。这样才能保证照片和新作不会被打印在一起。

我们来看另外一个例子。假定一个进程负责收集温度，它每隔一段时间测量一下当前室内的温度，并把它记录下来。另外一个进程则负责显示前一进程测量的结果。这两个进程同时在系统中工作着。可是，当测量进程刚刚把时间从 9 点修改成 10 点，准备把温度从 23 度修改成 27 度时，不幸的事情发生了。它的时间片用完了。这时操作系统毫不客气地把 CPU 收回来分给了显示温度的进程。显示进程很认真地读来了时间和温度，并把它显示出来：10 点的温度是 23。它并不知道这个信息是错的。这种情况显然是不应该出现的。因此，操作系统必须解决这个问题。

分析前面的两个例子，可以看出，问题的根源在于进程在对资源的使用有特殊的要求，即自己没有使用完资源之前，其他进程不能插进来打扰。这被称为资源的互斥访问。前面的打印机（硬件资源）是要求互斥访问的资源，时间和温度（数据，可以看作软件资源）也是这样。

操作系统怎么支持进程互斥地访问资源呢？如果程序能告诉操作系统哪些资源在未用完时不能被其他进程使用，那么操作系统也就好处理这件事情了。为此，操作系统提供手段让应用程序定义互斥资源。应用程序要使用互斥资源时，必须先申请。当互斥资源没有被其他进程使用时，操作系统才会让该进程使用。当然，进程用完互斥资源时，必须在程序中明确说明用完了。这样操作系统才会让其他进程使用这些资源。像打印机这类物理上要求必须互斥访问的硬件资源，操作系统自动实现资源的互斥支持，不用程序员再在程序中明确说明。

程序中访问互斥资源的那段代码被称为临界区，或者敏感区、危险区。也就是说这部分代码如果在执行过程中被其他进程打断就会带来危险。申请使用互斥资源的操作也常被称为加锁，意思是把资源锁住。而告诉操作系统使用完毕释放资源的动作被称为开锁。在使用互斥资源的应用程序中必须首先为互斥资源声明一把锁。就是一个与互斥资源相关联的变量，也称为互斥量。

图 4-8 是前面提到的测温进程和显温进程加上了互斥访问操作后的样子。加上这些语句后，测温进程在更改数据之前，要先申请锁住数据。这样如果它在数据更改一半被切换下来时，显温进程获得 CPU 要读取数据，那么它也要先申请锁住这些数据。可是系统发现数据已经被锁住了，所以显温进程就不能读取这些数据，只好等待。结果它只好让出 CPU，一直到测温进程再次执行并修改完数据，把锁打开之后。这之后显温进程才能读取数据。这样两个程序的运行结果就会是正确的了。加锁和开锁动作是由操作系统实现的。但是程序员必须在程序中正确的地方使用系统调用告诉操作系统来做这件事情。程序员需要很小心地使用这些功能。例如，如果在加锁之后忘了开锁，那么就会使另外一个进程一直等待下去。

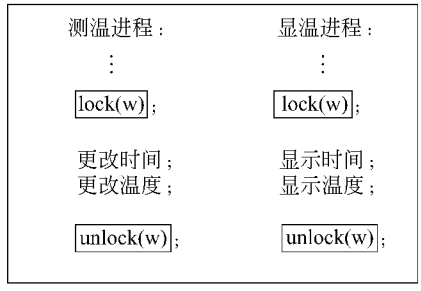


图 4-8　加锁互斥

小提示

操作系统通过提供互斥量及对互斥量进行操作的系统调用来实现进程对资源的互斥访问。

除了竞争资源外，进程之间有时还会有合作关系。进程间协作的典型例子就是负责产生数据的进程和负责打印的进程。负责产生数据的进程把数据放到缓冲区中，打印进程从缓冲区中取走数据并打印。只有缓冲区中有数据，打印进程才能执行，而产生数据的进程只有在打印进程把数据取走，缓冲区空出来时，才能往里面写数据。我们说这两个进程的动作需要协同。这两个进程的问题就是被称为生产者和消费者问题的典型示例。生产数据的进程叫生产者，打印进程被称为消费者。

为了解决生产者和消费者的同步问题，操作系统引入了一个“信号量”的概念。其实，生产者和消费者的同步要求是由于共享资源产生的制约关系。例如，打印进程和计算进程产生同步的需求就是由于共享的缓冲区资源和数据资源。对于产生同步的资源，操作系统让程序为其声明一个信号量。信号量实际上是一个整数。其值就是它对应资源的个数。信号量减少，表明消费资源。信号量增加，表明生产资源。操作系统提供了两种对信号量的操作，分别称为 P 操作和 V 操作。P 操作用来申请消费资源。它所做的事情包括：对信号量的值减 1；然后判断信号量是否小于 0；如果信号量减 1 后小于 0，表明现在没有资源可供进程使用，那么调用 P 操作的进程就要去等待；如果不小于 0，进程就可以继续执行，即使用资源。而 V 操作是对信号量加 1，然后判断信号量是否大于 0；如果加 1 后信号量不大于 0，就说明有进程在等待信号量对应的资源，那么调用 V 操作的进程就要发出信号，通知等待的进程已经有资源可用了；如果信号量大于 0，表明没人在等，那么调用进程就继续执行。有了操作系统提供的信号量和 PV 操作，前面的同步问题就可以很好地解决了。P 和 V 是荷兰语两个词的首字母，因为信号量机制是荷兰人 Dijkstra 在论文中提出来的，他早期的论文多用 P 表示等待、V 表示发信号，于是就这样延用下来。

图 4-9 是一个使用 PV 原语实现计算进程和打印进程同步问题的伪代码。这里信号量 buffer 表示缓冲区的数量，而 data 则表示数据的数量。

```
buffer : =1 ;                    data : =0 ;
P计算 :                          P打印 :
Repeat                           Repeat
    P(buffer) ;                      P(data) ;
    写入数据 ;                       输出数据 ;
    V(data) ;                        V(buffer) ;
Until false ;                    Until false ;
```

图 4-9　PV 原语实现计算、打印进程的同步

你发现了吗？如果把信号量的值设置为 1，就可以实现对资源的互斥访问。

找到了解决进程间竞争和合作问题的协调手段，进程可以在一个和平共处的环境下，友好地利用操作系统中的资源完成各自的任务了。

小提示

操作系统通过提供信号量及相应的 PV 操作来实现进程对资源使用的同步。
信号量也可以实现资源的互斥访问。

4.1.5　进程间的通信

系统中的进程会有相互交换信息的需求。有时要传递的信息很少，如子进程告诉父进程它已经完成任务退出了。有时要传递的信息也可能很多，如负责监测网络状态的进程要把自己收集来的信息交给负责产生统计信息的进程。为了让进程之间互相不干扰，操作系统禁止进程访问分配给其他进程的内存空间。因此，操作系统必须提供手段让进程之间能

够交换信息。信号量机制可以使进程之间传递同步信号，但是对于传递数据显然是无能为力的。为此，操作系统都为进程提供了其他方式的通信手段。常见通信手段有信号、管道、消息队列、共享内存，以及基于网络连接的通信。

对于只是通知某件事发生了这种简单的通信要求，操作系统提供了一种称为信号的通信机制(注意，不是信号量)。从名字上，就可以明白这种通信手段的含义。当发生某件事时，人们会发出特定的信号来通知另外一些人。例如，在古代，如果有敌人入侵，长城的烽火台上就会点起狼粪，另外一个烽火台上的人看到冒出的狼烟就知道敌人来了。当然使用信号，必须事先定义好信号的含义，如燃起狼烟就表示敌人来了。如果一个烽火台想传递胜利的意思，那么就不能再用狼烟了。他们必须再约定另外一个信号，比如可以点起若干支火把或者学几声狼叫。

操作系统提供的信号机制也一样。操作系统事先定义好一些信号表示一些特定的事件(现在多数操作系统也允许用户自己再定义一些专用的信号)。当特定的事件发生时，一个进程就会给另外一个进程发送相应的信号。操作系统为用户提供了发送信号的接口函数，让进程在发送信号时调用。接口函数的工作就是把进程要发送的信号放到目标进程的信号队列中。每个进程都有自己的一个信号队列，里面放着别的进程发给它的、等待它处理的信号。进程在执行过程中的特定时刻，如从系统空间返回到用户空间之前，会检查自己的信号队列看是否有需要处理的信号。如果有信号，进程就先处理信号，执行针对该信号的处理程序，完毕之后再继续执行后面的指令。对应每个信号进程都有一个处理程序。操作系统为信号定义了处理程序，进程可以使用系统定义的处理程序，也可以自己定义处理办法。当进程发现信号队列中有要处理的信号时，就去查找相应的处理程序来执行。处理完毕后，再去处理下一个信号，直到信号队列空了为止。本章后面会讲到和 I/O 设备相关的中断的概念，可以看出，信号的产生和处理有点儿像中断，只不过并不需要硬件的支持。因此，信号也被称为软中断。发送信号时，必须要指明发送给哪一个进程，即发送进程必须知道目标进程的号码。因为进程号是在进程被创建时由操作系统分派的。因此，程序员在写程序代码时是不能知道的。一般只有具有亲缘关系的进程才能在运行中通过特殊的系统调用得到相互的进程号码，所以，信号机制一般用在父子进程或者兄弟进程之间。

小提示

信号是操作系统提供的进程之间通信的一种手段。
信号可以在进程之间传递特定事件发生的信息。
信号一般用在具有亲缘关系的进程之间。

对于要传递大量信息的通信要求，信号机制就力不从心了。为此，操作系统提供了另外一些进程通信手段，常见的有管道、消息队列、共享内存，以及基于网络连接的进程通信方式。下面逐一介绍这些通信手段。

首先说管道。这个名字让你想起了什么？对！输油管道、下水管道。就拿输油管道为例。输油管道的作用是把油田的油输送给用户。油田从管道一端注入原油，用户在另外一端打开管道的阀门就会有油流出来。当然，油田先注入的油会先到达用户。如果管道中没有油，用户就不可能接到任何东西。操作系统中的管道也是类似的一种机制。操作系统为

用户提供建立管道的系统调用函数。需要通信的进程双方调用这些函数在它们之间创建一个管道。一方在创建时声明是管道的上游，而另一方则声明是下游。然后操作系统就会在双方之间建立一个通信管道。上游的进程往管道中写入数据，下游的一方从管道接收数据。可以看出，管道是一种先进先出的单向通信。如果需要双向通信就要建立两个管道。

管道的容量是有限的。如果管道满了，接收方还没有把数据取走，那么发送方就必须等待。这种同步控制由操作系统来负责。当管道满时，操作系统就让发送进程去等待。管道中的数据都是以字节为单位的，也就是管道中是没有格式的字节流。这个意思是说，管道中的数据都被看作是一个个单个字节，管道并不管这些字节之间的逻辑关系。例如，一个进程想给另外一个进程发送一首唐诗，“清明时节雨纷纷，路上行人欲断魂，借问酒家何处有，牧童遥指杏花村。”于是它分 4 次把这首诗放到管道中（为了节约，它没加标点）。接收进程接收数据时，4 句话已经连成一片了。因此它收到的是“清明时节雨纷纷路上行人欲断魂借问酒家何处有牧童遥指杏花村”。于是进程把它解释成一首词：“清明时节/雨/纷纷路上行人/欲断魂/借问酒家何处/有牧童遥指/杏花村”。

而消息队列是操作系统提供的能够保持数据逻辑边界的通信手段。操作系统为进程提供系统调用来创建消息队列。进程可以使用系统调用创建一个消息队列。任何进程都可以向这个队列中发送消息，也可以从队列中接收消息。在这里每个消息都是一个有意义的报文，不再是没有边界的字节流。这样如果一个进程把“清明”那首唐诗分成 4 个消息发送，那它就不会再被断句成词了。如果消息队列已满，那么操作系统就会让写消息的进程去等待。如果从消息队列取消息的进程没有找到需要的消息，那么操作系统也会让它去排队等候。

管道和消息队列都需要数据的复制。发送进程把数据复制到管道或者队列中，接收进程把数据从管道或者队列复制到自己的内存区中。为了省去复制数据的时间，操作系统提供了共享内存的通信方式。操作系统提供系统调用让一个进程声明自己某块内存区可以被共享。而其他进程可以通过系统调用把这段内存映射到自己的空间中。然后，两个进程就都可以直接读写这段内存区间了。共享内存不涉及数据的复制，因此是速度最快的通信方式。但是，共享内存没有同步机制。如果一个进程向共享区域中写内容了，另一方如果不查看的话，它是不会知道的。也就是说操作系统是不会插手帮忙的。所以需要进程自己通过其他手段来实现同步。而管道和消息队列都由操作系统来控制同步。

最后一种通信方式是通过网络连接来实现的。前面提到的几种通信方式要求通信进程都要在同一台计算机上。如今计算机已经通过网络连接起来了。不同计算机上的进程也需要通信。为此，操作系统为不同计算机上的进程建立网络连接，让它们通过网络连接来传递信息。要建立网络连接，进程必须通过系统调用创建一个称为套接字（socket）的抽象结构。进程通过套接字和另外一个进程建立联系。这就叫创建了一个网络连接。每个套接字在网络中都有一个唯一的标识号，通常是由计算机的名称（一般是 IP 地址，在网络中，每台计算机都有一个独一无二的数字标号，称为 IP 地址）和端口号构成。计算机名称用来在网络中定位计算机，而端口号用来区分同一台计算机上的不同进程。主机的名字就好像是公司的通信地址。而端口号就好像同一个公司中不同人的名字。进程通过套接字把消息发送到网络中，消息中指明接收进程所在的主机名以及接收进程正在守候的端口号。网络中的传输设备会根据消息中的主机地址找到目的进程所

在的主机，把消息传递给该主机。然后主机上的操作系统通过消息中的端口号找到计算机上接收这个信息的套接字，并把数据复制到套接字的缓冲区内。这样目的进程就可以从套接字中接收到数据了。这种办法也可用在同一台计算机上的进程之间通信。只不过目的进程所在的主机地址就是本机地址而已。

小提示

可以传送大量信息的进程间通信手段有管道、消息队列、共享内存，以及基于网络连接的进程通信方式。

管道是一种先进先出的单向通信。管道中的信息以字节为单位组织。

消息队列能够保持数据的逻辑边界。

共享内存是最快的一种进程间通信手段，但是进程必须自己实现同步控制。

基于网络连接的进程通信方式可以支持同一主机或者不同主机上的进程之间的通信。

4.1.6　线程

操作系统用多进程并发运行来提高系统的效率。但是，进程切换需要很多时间和空间上的管理开销。因此，怎样降低这些开销，又让系统仍然能并发工作，成为操作系统设计者的工作目标。

解决问题当然从分析问题入手。经过研究人们发现进程切换时做的工作大致可以分成两部分。一部分与处理机相关，如保存和恢复进程上下文、进程状态的改变。而另外一部分只是与进程有关，而与处理机无关，如修改存储管理、文件管理相关的数据结构。而进程切换的目标主要是改变 CPU 所做的事情，实际上是处理机这个资源分配的改变。按照道理应该只有与处理机相关的信息是必须改变的。如果进程切换时只修改与处理机相关的信息，切换开销不就小了吗？那么为什么切换进程时需要改动这么多信息呢？这需要了解一下进程在系统中扮演的角色。在系统中，操作系统是以进程为单位分配资源的，这些资源包括内存空间、处理机、I/O 设备，还有文件等。而处理机分配的改变导致了进程的切换。进程的切换又使系统中所有与其相关的资源状态信息发生改变。可见处理机是不同于其他资源的。其实处理机分配的改变，就是计算机执行任务的改变。可以看出，进程同时是执行任务的单元体。这样一来，进程实际上同时扮演了两个角色，资源分配的单元体以及执行任务的单元体。

要想处理机切换时不改变其他资源的状态信息，那么必须把其他资源与处理机的分配载体分开。这样就必须改变一个事实：处理机的分配单位和其他资源的分配单位不能再是同一个实体。为此，操作系统的设计者采用了另外一个分配处理机的单位，称为线程。有了线程之后，进程变成除了处理机之外的其他资源的分配单位。也就是它放弃了作为执行任务的单元体的角色。而线程是执行任务的单元体。这样，除了处理机之外的资源仍然以进程为单位分配。处理机则以线程为单位分配。系统在创建进程时需要同时为其创建一个或者更多线程。同一进程的所有线程共享分配给进程的资源。程序员可以把程序要完成的任务分给几个线程来完成。操作系统调度这些线程在一个或者多个处理机上并发运行。这样，当处理机在同一个进程的线程之间切换时，就只需要切换与处理机有关的信息，各种其他资源的相关信息就不用变动了。当然，如果处理机在不同进程的线程之间切换，那么要做

的工作与以前切换进程时所做的相同。

操作系统提供创建和撤销线程的系统调用。线程也可以创建其他线程。因为线程只与处理机分配相关，所以创建线程时要设置的相关信息也少。因此，创建/撤销线程的开销比创建/撤销进程要少很多。

因为同一进程的所有线程共享地址空间和文件，所以这些线程可以通过共享内存和文件直接通信，就不用再走进程间通信那些必须由操作系统协助的烦琐办法了。不过问题是操作系统必须提供线程同步的手段，帮助程序员协调线程对共享空间的访问。

关于线程的概念有很多说法。例如，线程是进程内的一个执行单元；线程是进程内的一个可调度的实体；线程是程序中的一个相对独立的控制流序列；线程是执行的上下文，其含义是执行的现场数据和其他的调度所需的信息；线程是进程内一个相对独立的、可调度的执行单元等。把这些说法列在这里供你参考和理解用。

小提示

在使用线程的系统中，线程是分配处理机的单元实体，而进程则是处理机之外其他资源的分配单元实体；一个进程至少包含一个线程；同一进程中的线程之间的切换不用切换进程的资源信息，因而开销比切换不同进程之间的线程小。

附文：

我国研制“银河麒麟”操作系统用于国防

人民网 http：//it. people. com. cn/GB/42891/42893/5124224. html

2006 年 12 月 05 日 07：45

中新社北京十二月四日电(记者 孙自法)中国科学技术部四日下午向媒体发布信息说，中国科研人员经过四年多攻关与创新，先后突破一系列核心技术，研制成功目前中国通过认证的安全等级最高、拥有自主知识产权的操作系统——“银河麒麟”服务器操作系统。

这一中国目前安全等级最高的服务器操作系统当天在北京通过专家验收，它是国家“八六三”计划软件重大专项课题，由国防科技大学牵头承担。专家称，该系统研制成功，是中国“八六三”计划一项重大成果，标志着中国操作系统研发水平迈上一个新台阶。

“银河麒麟”操作系统由中国自主研发的基本内核层和基于 FreeBSD(一种 UNIX 操作系统)改造的系统服务层组成，是一个拥有层次式内核、安全等级达到结构化保护级、能支持多种微处理器和多种计算机体系结构，并与 Linux 目标代码兼容的国产服务器操作系统。

对该系统的严格测评结果表明，“银河麒麟”操作系统实现典型服务器操作系统的全部功能，具有高安全性、高可用性、强实时性、可扩展性和软硬件适配性等特点，系统整体性能与国际主流 UNIX 操作系统相当，部分性能指标以及实时性指标更好。该系统已通过自由标准组织 Linux 标准基认证，并在中国国内率先通过公安部等有关部门的安全认证。

目前，国防科技大学已与联想集团等签署“银河麒麟”操作系统的产业化合作协议，成功应用于金融、政府、教育、证券等领域，并已在国防领域相关信息系统中得到成功应用。

计算机操作系统是计算机的灵魂。长期以来，中国的计算机操作系统市场基本上被国外产品所占领，研制具有自主知识产权的操作系统是中国的一项战略决策。

来源：中国新闻社

4.2　内存的管理

内存管理就是要解决进程有关信息在内存中的存储问题。你可能会问：内存管理也要讲吗？当程序要运行时，如果内存中有空闲空间就装入程序，否则就拒绝运行。多简单的事情！这一节我们就来看看这件事情是不是就真的这么简单。

为了保证能够找到已经装入的程序，操作系统需要记录程序在内存中的位置。为了标识位置信息，操作系统必须把内存空间进行编号。我们已经知道内存是由一些相同的存储单元构成的。整个内存可以被看作是一个大盒子，盒子中有无数一模一样的小盒子。操作系统把这些小盒子按照顺序编号。号码从 0 开始一直到小盒子总数。每个盒子中可以放一个字节，也就是 8 位 0/1 数码。图 4-10 是内存空间的一个示意图。

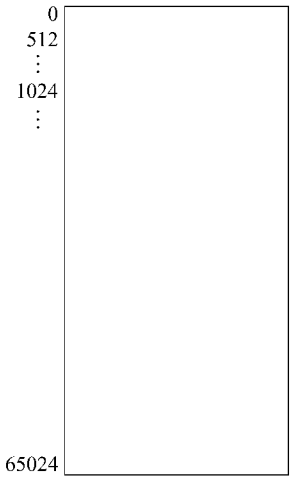


图 4-10　内存空间示意图

做好这个准备工作之后，操作系统就可以开始分配和使用内存空间了。这件事其实并不像看起来那么简单。为了让你能够理解操作系统所做事情的前因后果，我们还要从头说起。

在计算机刚刚出现时，一个系统中只能有一个程序在运行。这个时期称为单道程序时代。那时整个内存都由一个进程使用，进程的全部程序和数据都放在内存中。那时如果程序太大，超过了内存的大小，程序就不能运行。所以，程序员写程序时必须知道所用计算机内存的大小。他要在写程序时指定好程序和数据在内存中的位置。这个时候系统中也根本不存在操作系统。所以更不用说由它来分配内存了。内存的使用完全由程序员自己控制。

后来出现了监控程序(操作系统的前身)。系统启动时，监控程序首先会被装入到内存中。在系统运行的整个期间，监控程序始终都在内存中。因为监控程序很重要，所以必须把它保护起来。因此，它被装载在内存中特定的地方，留给监控程序使用的那段内存被称为系统空间。用户的程序不能使用系统空间这段内存。用户程序能够使用的内存区域被称为用户空间。操作员把要运行的程序成批交给监控程序。监控程序控制计算机按顺序读入用户程序。如果用户程序要求的空间小于计算机的用户空间，监控程序就把它读入到用户空间，并让它开始执行。用户程序执行完毕，监控程序收回内存，再读入下一个用户程序。如果用户程序要求的空间大于用户空间的大小，那么这个用户程序就被拒绝。这个阶段被称为批处理系统阶段。这时的系统中同时存在两个程序：监控程序和一个用户程序。当用户程序要使用监控程序所在的区间，监控程序就会终止该用户程序。对系统空间的保护功能从此成为操作系统内存管理的重要功能之一。

可以看到无论在单道程序阶段还是批处理系统阶段，如果用户程序需要的空间大于可以使用的内存，那么程序就无法执行。即程序要受到内存空间大小的限制，程序员写程序时

必须知道可用内存空间有多大。

在单道批处理系统之后，计算机就进入了多道程序批处理阶段以及后来的分时系统阶段。总之，从那以后计算机的内存中就不再只有一个程序，应该说不只有一个进程的信息了。进程相关的信息有哪些？我们已经知道它至少包括进程的控制块、进程要执行的程序代码以及用到的和产生的数据等。这时内存的分配和管理就变得复杂起来。为什么复杂？有多复杂？下面我们就从多道程序的出现开始介绍操作系统内存管理的发展及相关技术。从中来体会看似简单实际上未必就真的简单的问题。

4.2.1 固定分区

在多道程序出现初期，为了支持多个进程共享内存，操作系统采用了一种非常简单的办法。它把整个内存分成大小相等的若干个区域，然后以区域为单位给进程分配内存。这些区域被称为分区。操作系统用一个表格记录整个内存空间被分成了几个区、每个区的起始位置和大小、分区是否被使用以及分配给了哪个进程。当要创建新进程时，系统就查看表格，找到一个未被使用的分区分配给新进程，并更改表格中相应的记录。当然，进程要求的空间要小于等于分区的大小。当进程退出时，系统就把分配给它的分区状态更改为未使用。分区在系统启动时就分配好，在整个系统运行期间都保持不变。因此，这种内存管理办法被称为等长固定分区法。图 4-11 是用等长固定分区法分配内存的一个示意图。图中 64MB 内存一共被分为 8 个 8MB 的分区。因为是以分区为单位分配内存，所以即使一个进程只使用了分区中的很小一部分空间，分区中的剩余空间也不能被其他进程使用了。例如，图 4-11 中作业 A 只有 1MB 大小，分配给它的 8MB 分区还剩 7MB 没有使用。但是，其他进程也不能使用。因为在操作系统的表格中这个分区（整个 8MB 空间）已经被分配出去。这些已分配分区中剩余的空间被称为内部碎片。

按照等长固定分区法，图 4-11 中还剩下三个分区、共 24MB 的空间可以分配给三个进程使用。而实际上图 4-11 中的 64MB 内存只使用了 26MB。所以等长固定分区法比较浪费空间。后来人们就对其进行改进。人们发现各个进程需要的空间大小是不同的。而等长分区法却把分区分成相同的大小，所以小进程会形成很大的内部碎片，浪费空间。为此，人们又把分区分成各种不同的大小。这样创建进程时，系统就可以根据进程的需要选择一个大小合适的空闲分区分给它。这种办法被称为不等长固定分区法。图 4-12 是不等长固定分区法的内存分配示意图。图中 64MB 的内存空间被分成 8MB、2MB、4MB、8MB、16MB、26MB 几个大小不等的分区。看起来好像浪费的空间确实少了。不过如果作业需要的空间大小不像图中那样，比如，5 个作业都要求 1MB 的内存，那么空间浪费还是很严重的。

如果进程需要的内存大于所有的分区，那怎么办呢？没办法！这个程序在这台机器目前的操作系统下，就没有办法运行。这就给程序的规模带来了限制，给程序员提出了很高的要求，尤其是在早期计算机内存比较小的情况下。为此，当时的程序员想出一个办法。他们把程序分成可以不同时装入内存的几部分。程序运行时，它只向操作系统申请够装入一部分程序的内存。当需要用到其他部分代码时，进程再把这些程序读入到内存，覆盖掉自己前面所用内存的内容。这种技术被称为覆盖技术。它是在程序员的控制之下实现的一种内存扩展办法。

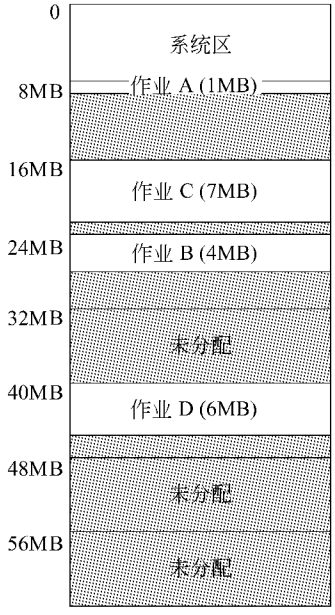


图 4-11　等长固定分区法示意图

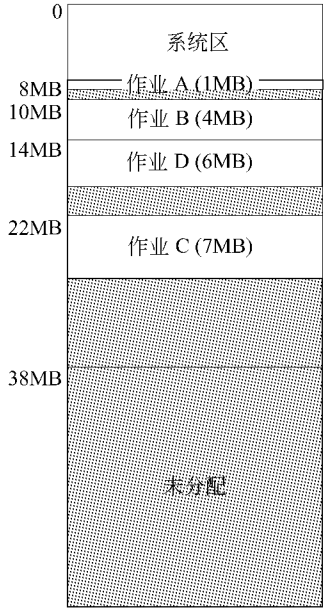


图 4-12　不等长固定分区示意图

采用固定分区的办法，系统只要维护一个固定行数的表格，里面记载分区的情况就行了。内存分配算法也很简单，有足够大的空闲分区就分配，否则就拒绝。因此，固定分区法是最简单的内存管理办法。但是，在这种策略下，无论系统中进程所占用的内存是多少，系统最多能同时容纳的进程数目是固定的，就是分区的个数。无论是等长还是不等长分区法，分区内部碎片造成的内存浪费都比较严重。因此现在这种方法已经很少使用了。

小提示

固定分区法是最简单的内存管理方法。
固定分区法分为等长和不等长分区两种。
固定分区法限制了系统中同时存在的进程数目，其产生的内部碎片会浪费内存。

4.2.2　动态分区

既然固定分区会产生内部碎片浪费内存，那么我们就不在系统启动时确定分区大小，而是在系统运行中根据进程需要的空间来确定分区大小。这样就不会产生内部碎片了。这种办法就是在固定分区法之后出现的动态分区内存管理，也叫可变分区法。即分区大小和分区数都是根据实际情况变化的。下面我们来了解实现动态分区法操作系统要做的事情。

系统启动时，除了操作系统占用的系统空间外，用户进程可用的空间整个都是空闲的。它被看作是一个大的空闲分区。当操作系统要创建新进程时，它就从这个大空闲空间的一端按照进程的需求分割一块给这个进程。此时空闲分区还是一个，只不过大小有些变化。再创建新的进程就继续这样分割。系统运行一段时间之后，有些进程执行完毕，退出了内存。它原来所占的区间就空闲出来，而这段区间前后的内存可能已经被分配出去了。这样

就增加一个空闲分区。随着进程的不断装入、退出，内存中的空闲区间就不再是一个连续的大区间，而是很多大小不等的、不连续的区间，如图 4-13所示。

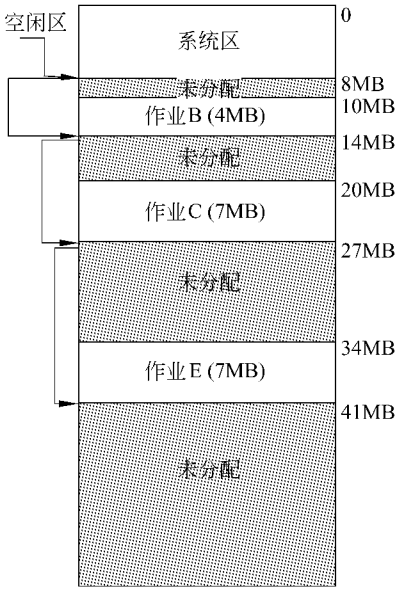


图 4-13　可变分区空闲区间示意图

为了分配空闲分区，操作系统需要掌握这些分区的情况。这包括有哪些空闲分区，它们都在什么地方、有多大。通常操作系统会把这些空闲分区组织成链表。它在一个特定的地方记录第一块空闲分区的开始地址以及大小。然后在第一块空闲区中的某个位置，如分区开始的头几个字节中或者最后几个字节，记录第二块空闲分区的起始地址和大小。这样就形成一个链表，通过它操作系统就能够找到所有的空闲区间了。图 4-13 中的空闲分区就用链表连接起来。

系统中有这么多大小不一、不连续的空闲分区，如果有新进程需要空间，应该给它哪一个空闲分区呢？这就要看操作系统采取什么放置策略了。首先能想到的分配方案就是选择与新进程需要空间大小最接近的空闲分区给它。这种算法被称为最佳适应法。最佳适应法看起来应该是最合理的。但实际情况并不是这样。为了找到大小最接近的空闲分区，系统需要把所有空闲分区都检查一遍，才能比对出到底哪个分区大小是最接近的。检查所有空闲分区需要花费时间，尤其是分区很多的时候。而且，分割大小最接近的分区，意味着分割所剩分区的空间也是最小的。这些剩余分区最终会小到不够任何进程使用。这种小分区因此被称为外部碎片。随着系统的不断运行，内存中的外部碎片会越来越多。最终会出现这种情况，即使内存中总的剩余空间可以满足新进程的需要，但是系统却找不到空间可以分配。要解决外部碎片问题，操作系统可以定期进行内存紧凑。就是移动内存中的进程分区，让剩余分区连到一起。这样就可以形成大的空闲区间。但是移动内存中的代码和数据需要很多时间，并且还需要更改每个进程的进程控制块，要保证进程能够继续正确地访问到自己的程序和数据。通常操作系统需要硬件配合才能完成这件事情。所以，移动数据和程序，将空闲空间全部集中到内存一端的代价是非常大的。

既然最佳适应法容易产生很多小得无法分配的区间，那么，如果让每次分割剩下的分区尽可能地大，小区间不就少了吗？怎样能让剩下的区间尽可能地大呢？办法很简单，选择最大的空闲区间分割就行了。即每次分配分区时，都选择最大的空闲分区分配。这种办法被称为最差适应法。最差适应法解决了小区间的问题，减少了系统紧凑的概率。但是，要找到最大的空闲区间同样也需要把所有的空闲区间都比较一遍才行。最差适应法的一个新问题是，因为总是从大的空闲区间开始分配，所以系统中很难剩下面积很大的空闲区间。如果进程需要大的空间，那么就很难满足了。

最佳适应法和最差适应法都要检查所有空闲分区，要花费很多时间。因此，我们改变一下策略：在从空闲分区链表开始查找分区时，只要发现能用的分区就把它分配给申请进程，无论这个分区比需要的空间大很多还是大一点点儿。这样就可以减少空闲分区的查找时间

了。这种办法被称为首次适应法。不过首次适应法总是从链表的开始查找，找到合适的分区就分配出去。时间长了，在空闲分区链表的前端就会剩下很多小的分区。这些分区会变得越来越难以分配出去。但是，每次还不得不从这些小分区开始检查。所以，实际上首次适应法并不能很好地减少查找时间。怎么解决这个问题呢？如果不是每次都从头开始查找就好了。那么从哪里开始呢？从上次分配的空闲分区之后开始查找应该是个解决办法。为此操作系统要记住接下来该检查的空闲分区的位置。等到要给进程分配分区时，系统就从记录的这个分区开始依次向后查找直到碰到能用的分区为止。如果到链表尾还没有找到，那就再从头开始。这种办法被称为下一个适应法。这种办法存在的问题和最差适应法类似：因为它循环分配所有的空闲区间，所以系统中很难剩下面积很大的区间。

这么多办法都各有利弊，没有一个绝对好于另外一个。从前面几章我们就注意到这个现象了：十全十美的解决办法似乎是不存在的。所以，多数时候系统的设计者都要根据具体情况权衡利弊，选取一个折中方案。在实际的工程中，工程师们通常要考虑成本、工期、质量等方方面面的原因最终给出折中方案。这也是你看到很多方案都不能尽善尽美的原因之一。

小提示

动态分区法可以解决内部碎片的问题。
动态分区法常用的放置算法有最佳适应法、最差适应法、首次适应法以及下一个适应法。
动态分区产生的外部碎片，会降低内存的使用率。

4.2.3　页式内存管理

固定分区法和可变分区法都各自存在着问题。因此，人们还需要不断地努力寻找更好的内存管理方法。尽管这两种办法已经很少使用了，但是它们为后来的解决方案奠定了很好的基础。所以把它们放在前面介绍作为内存管理的一个铺垫。

现在我们再来看看这两种办法存在的主要问题，以便能够针对问题找到解决方案。固定分区的问题是内部碎片浪费空间。因为进程需要的空间大小差异很大，并且不可预知。所以，很难确定分区的大小取多少合适。如果分区选得很大，对于小的进程，分区内就会产生很大的内部碎片。如果分区分得很小，大的进程又会因为不够用而不能运行。这样一分析，我们发现了一个问题：为什么一个进程不能使用几个分区呢？这样就可以把分区的面积缩小，不就可以减少内部碎片了吗？因为这样在分给进程的所有分区中，只有最后一个分区会存在内部碎片，而且小于分区的大小(别忘了现在分区已经变小了)。

动态分区的问题是会产生小得不够任何进程使用的外部碎片，造成空间的浪费，如果进行内存紧凑，又会带来时间上的开销。针对这个问题，你一定也想到了解决办法：把多个小分区分配给一个进程，不就行了吗？这样不就不存在外部碎片了吗！这个方案实际上改变了进程空间必须连续分配的做法。把一个进程需要的空间分散在内存中不相连的几个分区中。为实现这个方案，操作系统必须做一些额外的工作。我们知道程序计数器中存放的是下一条指令的地址。系统通过将其值自动加 1 形成下一条指令的地址。这里暗含着下一条指令紧挨着当前这条指令。也就是说计算机认为指令是连续存储的。如果程序在内存中不是连续存储的，那就会出现问题。例如，当前指令位于进程所分配

的某个分区的末尾，那么程序计数器加 1 后得到的地址就不是下一条指令的地址。为了支持不连续的内存分配，操作系统需要能够找到指令或数据在内存中实际的存放位置。这个工作被称为地址重定向。

现在整理一下解决内存分配问题的新思路。首先，内存要分成固定大小的区域；其次，这些区域要很小；第三，可以给一个进程分配多个区域；第四，分配给一个进程的多个区域可以不用挨着；第五，需要地址重定向技术的支持。好！我们已经得到了一个崭新的内存管理方案，它的官方名称是页式内存管理。下面就把它具体化。

当系统启动时，操作系统把内存分成固定大小的区域。这些区域大小相等（因为等长分区要比不等长分区好管理）。假设每个区域为 512KB。这些小区域不再叫做分区，我们给它换个名字叫做页框（或称页帧、物理块）。为什么叫页框？其实也不难理解。就像一个大仓库，如果直接往里面堆放货物，放的时候不是很方便，而且找起货物来也很麻烦。通常的做法是，在仓库中搭一些木头或者金属的货架子。货架子分成一个一个格子。这就是框的意思。有了框之后，货物就放在每个框中。给每个框编上号，就可以记录货物的位置了。查找货物就很方便了。内存就好比仓库，页框就好比仓库中货架子的格子。仓库的每个框是用来放货物的，而内存中的框是用来放置进程信息的。操作系统把进程要存放在内存中的信息按照页框的大小分成一个个小部分，就好像是把每个客户的货物打成包一样，每个小部分称为一个页（或者页面）。这就是为什么内存的框要叫页框，因为是用来放“页”的。两方面都分好之后，操作系统就可以为进程分配内存空间了。

与仓库管理一样，操作系统把每个页框都编上号。进程的每个页也被编号。每个进程的页是独立编号的，不受其他进程影响，因此不同进程使用的页号可能是一样的。这就好比把仓库中每个货主的货包单独编号一样。

为了在分配空间时能够找到空闲的页框，操作系统需要记录空闲的页框有哪些。采取的记录方式可以是空闲页框链表，也可以用位图表示每个页框的使用情况。位图是什么？即位数与内存页框数相等的一个 0/1 串，0 表示对应的页框是空闲的，1 表示正在被使用着。

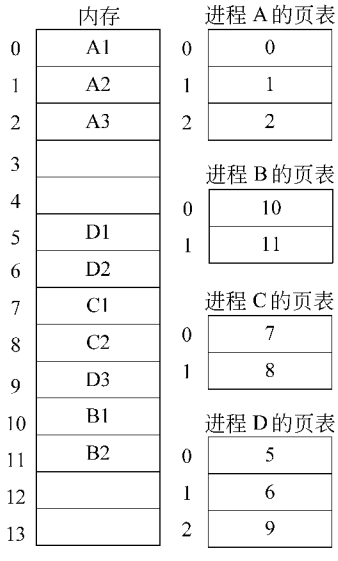


图 4-14　内存分页及进程页表示意图

进程的页被放到不同的页框中。操作系统必须记录进程的页都放到哪个页框中了，不然就找不到了。所以，操作系统为每个进程维护了一个记录这种对应关系的表。这个表被称为页表。每个进程都有一个页表。

下面通过一个例子来了解页表。假定系统中有 A，B，C，D 共 4 个进程。其中，A 进程需要的空间大小为 1069KB，B 进程需要的空间为 947KB，C 进程大小为 524KB，而 D 进程则需要 1500KB。假定系统设定的页面大小为 512KB。那么这 4 个进程需要的页框数分别为 3，2，2，3。分给每个进程的最后一个页框并没有用完，有的剩的多一些，有的剩的少一些。但是都不会超出一个页框的大小。当进程需要空间较多时，内部碎片的比例就会变得越来越小。图 4-14 说明了这 4 个进程使用内存的情况以及这 4 个进程对应的页表。从图中可以看出来，整个内存的页框是统一编号

的。而每个进程的页号都是从 0 开始。因为进程的页号是按照顺序从 0 开始的，所以如果按照页号顺序将对应的页框号放在页表中，页表中就可以省略页号。因为通过页框号在页表中的位置就可以算出其对应的页号。例如，页表中的第二个页框号对应的是进程的 1 号页(页号从 0 开始)。现在系统中不再存在任何外部碎片，因为任何一个空闲页框都可以被分配出去。

要执行程序中的指令，系统需要知道每条指令在内存中的地址。可是现在程序每次执行时被放到内存中的位置都可能不同。这与系统当时内存的使用情况相关。为此，在程序放入内存中之前，每条指令和数据也都被编了一个号。通常这个编号工作由编译程序来做。这个编号可以看作是程序在纸面上的一个地址。这个地址与程序实际放在内存中的位置无关。我们把这个地址称为逻辑地址。编译程序为每个程序单独编址。为了得到进程指令在内存中的实际位置，操作系统需要根据进程页表中的记录信息把指令的逻辑地址转换成该指令在内存中的实际物理地址。具体转换过程如图 4-15 所示。因为操作系统会按照页框大小先将进程代码和数据分页。每条指令/数据所在的页号正好应该是其逻辑地址除以页框大小的整数部分，除法的余数正好是该指令在页中距离页开始地址的偏移量。又因为页框大小通常都是 2 的整数次幂，如 $512=2^9$。这样二进制除法就变成了向左移位。例如，二进制地址除以十进制数 512，就变成小数点向左移动 9 位。二进制地址的后 9 位就是余数，即偏移量，9 位之前的数值就是商，也就是页号。所以，当要访问某个逻辑地址的内容时，操作系统首先按照页的大小把地址分成两部分。一部分作为页号，另一部分作为该指令在页中的偏移量。然后操作系统根据页号查找进程的页表，从中得到该页在内存中使用的页框号。操作系统把页框号转换为该页框在内存中的起始地址。最后，把页框的起始地址加上指令在页中的偏移量，就求出指令在内存中实际存放单元的物理地址了。

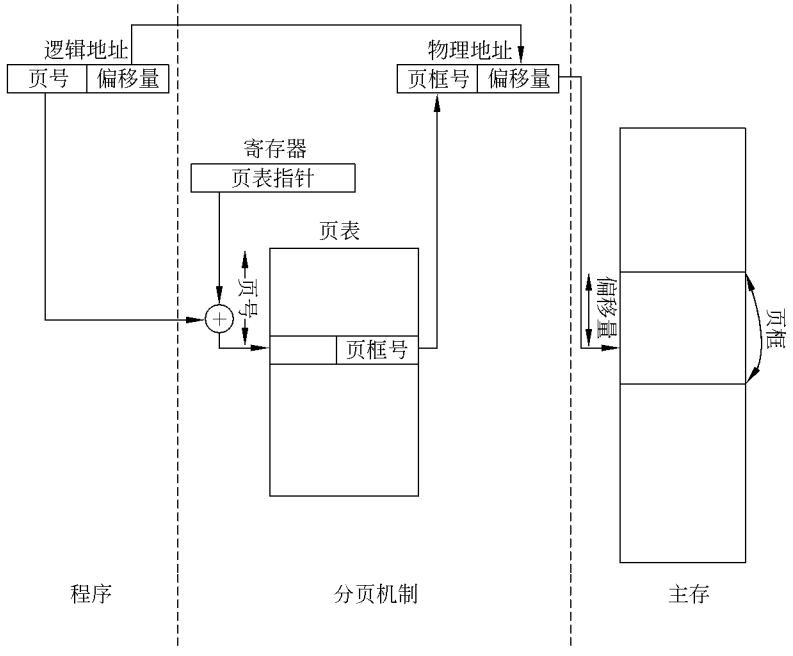


图 4-15　页式内存管理下的地址转换过程

小提示

页式内存管理的具体做法是：系统把内存分成固定大小的页框，需要存储的信息被分成大小相等的页，每个页被放到一个页框中。系统以页框为单位分配内存。一个进程可以使用多个页框，并且分配的页框在内存中的位置可以不连续。

页与页框的对应关系存放在页表中，系统为每个进程维护一个页表。

页式内存管理的实现需要硬件的支持。

4.2.4 段式内存管理

分页内存管理在对程序或者数据分页时并不考虑每个页内容的意义。有的操作系统对这种方法加以修改，按照逻辑意义把程序分成若干小部分，然后放到内存中的不同区间。这种分配内存的方法被称为段式内存管理。分段是按照程序和数据的意义进行的，因为不同的代码表示的逻辑意义不同，长度也因而不同，因此每个段的长度也不确定。由于需要知道代码的意义，所以分段必须有程序员来完成。一个进程的段可以被放到内存中的不同位置，但是每个段的空间必须是连续的。为了找到指令在内存中的位置，操作系统需要记录从逻辑段到内存物理段的映射关系。这些信息记录在段表中，如图 4-16 所示。因为每个段的长度是随着程序段的逻辑意义变化的，所以段表中要给出段的长度。

当要访问一个逻辑地址时，系统首先判断这个地址所在段的段号，然后以此查找段表，找到段在物理内存中的起始位置。接着系统再计算这个地址在该段中的偏移量，把它加上物理段的起始地址，就可以得到该逻辑地址对应的物理地址。分段方式的地址转换过程如图 4-17 所示。可以看出与页式管理方法的地址转换方式非常相像。但是，段式管理方法实际上和动态分区法非常相近。

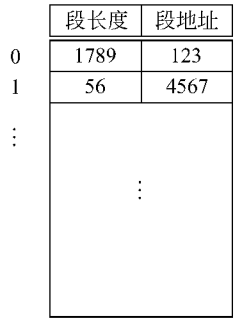


图 4-16 段表

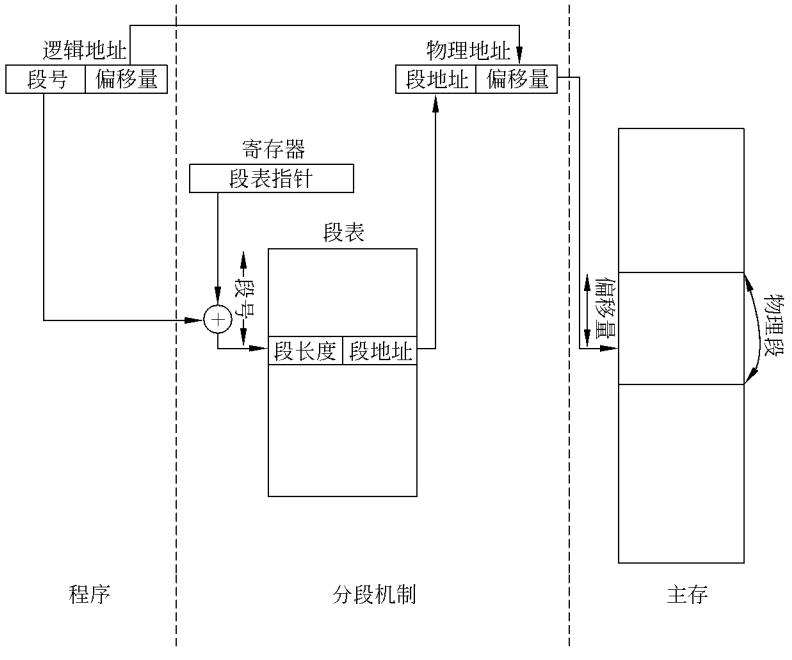


图 4-17 分段内存管理下的地址转换过程

有时候人们会把分段和分页两种方式结合起来，形成段页式内存管理。也就是先把程序按照逻辑意义分成段，然后每个段再分成固定大小的页，以此为基础进行内存分配。段页式方法下，逻辑地址要转换成物理地址需要经过两次查表，即先查段表，再查页表。

小提示

段式内存管理按照逻辑意义将要存储的信息分成段。系统以段为单位进行内存的分配。不同段的长度是不同的。

逻辑段与物理段的映射关系被存放在段表中。

段式与页式内存管理结合形成段页式内存管理。

4.2.5　虚拟内存管理

前面介绍的内存管理方法都没有解决一个问题，即每个程序运行所需空间不能超过可用内存大小，否则程序就会因不能装入内存而无法运行，除非程序员采用覆盖技术。但是发展的趋势是软件功能越来越复杂，代码随之也越来越长。让每个程序员在写程序时考虑内存的大小、考虑怎样覆盖，也越来越不现实。所以，这个问题必须解决。人们发现，如果程序可以不用全部放在内存中就能够执行，问题就解决了。

经过研究，人们确信这个想法是可行的。因为有以下的事实和技术作为基础。首先，无论多大的程序，其实并不需要所有的代码和数据都放到内存中，因为一个 CPU 在某个时刻只能访问一条语句或者一个数据。当然，为了减少 CPU 等待数据从外存转载到内存的时间，可以把一段代码或数据预先放到内存中。不过根本用不着把一个大程序的全部内容都放到内存中。覆盖技术其实就是基于这种思想。其次，我们有成熟的地址重定向技术，允许程序在内存中的位置不连续且可以变化。

既然这样，操作系统就不需要再一次把一个进程的全部信息都装入内存了。它只要装入一部分，然后就可以调度进程运行了，其他信息可以等到需要时再装入。这样无论多大的程序都可以在有限的内存中运行了。程序员写程序时再不用考虑内存的大小了。他可以编写使用任意大内存空间的程序。比如，程序员可以写一个需要 1GB 空间的程序。编译程序对这个程序进行编址，地址空间从 0～1GB。但是这个程序可以放到只有 256MB 内存的计算机上运行。因为它的内容是分几次装入内存的。在这种情况下，程序员感觉他有 1GB 大的内存空间，而不是 256MB。程序员感觉到的 1GB 空间可以理解为在纸上的，实际上可能并不存在这么大的物理内存，所以我们把程序员写程序时使用的这个地址空间叫做虚拟内存空间。因此，这种内存管理办法也有了一个名字——虚拟内存技术。具体来说，虚拟内存技术就是采用虚拟空间独立编址，操作系统负责把一个大的虚拟空间的内容分阶段装入实际内存中运行的技术。在虚拟内存技术下，每个程序员都会以为自己有一个很大的内存空间，而且是自己独享的。例如，程序员张三可以认为自己有一个无限大的内存，地址从 0 到无穷大；而李四同样可以认为自己有一个从 0 到无穷大编号的内存空间。他们想用多大就有多大，根本不用受限制。当然，实际上这个空间也是有上限的，它受制于操作系统设定的虚拟地址宽度。例如，如果用 32 位二进制数表示虚拟空间中的地址，那么虚拟空间的上限就是 $2^{32}=4G$。

只要把前面介绍内存管理方法稍加改造就可以实现虚拟内存技术。下面我们来看看

为了实现虚拟内存技术，操作系统具体需要做哪些事情。我们以改造页式内存管理为例。先来想一下大体思路。首先，为了把一部分程序放到内存中，操作系统需要先把程序分成若干部分。这个不难，前面介绍的分页就是在做这个事。分完之后，操作系统要选择一部分内容装载到内存中。这时操作系统需要记录进程的哪些部分装载到内存中，哪些没有；装载到内存中的内容放在什么地方。这个做起来也很容易。把页表稍加改造即可。每个进程的页表中增加一个标志，表示该页是否装入内存；如果已装入，就在页表中记录下装载其的页框号。

这样做完之后，当进程执行过程中需要访问某个信息时，操作系统需要先判断要访问的内容是否已装入到内存中。如果已经被装入，那么操作系统要知道放在什么位置了。这需要操作系统查找页表，把要访问信息的虚拟地址转换成对应的物理地址。如果要访问的内容还没有装入到内存，这时操作系统需要在内存中找到一块空闲空间分配给进程，然后把要访问的内容从外存读取到内存，并且修改页表。如果内存中没有空闲空间，或者即使有空间，但为了防止内存空间被用完，操作系统并不想把这些内存分给进程，那么，这时操作系统就要选择内存中一些正在被使用的单元，把里面的内容写回到外存，把这些空间释放出来。这个过程叫做换出或者淘汰。这个工作需要做得很谨慎。如果选择换出的单元不合适，刚刚换出的内容马上又被访问，那么操作系统就不得不又把它换进来。如果这种换入换出的频率太快，计算机就不用做别的事了。这种糟糕现象被称为抖动。它会使计算机系统的效率急剧下降。为了防止抖动，操作系统必须采用一些策略来选择合适的内容换出。这些策略被称为淘汰算法。

讲了这么多，你是不是还有点儿不放心？只把一部分代码放到内存中，行吗？如果要执行的一段代码分布在程序不同的页中，比如，接下来要执行的1000条指令分别在1000个不连续的页中，那么系统是不是要不停地从外存装入页面，然后再为腾出内存而换出页面？其实，你倒是可以放心，在大多数情况下，这种假设情况是不会发生的。因为人们通过实验发现，一般情况下一个进程在一段时间内要访问的指令和数据都集中在一起，不会那么分散，更不会那么均匀地分布在整个虚拟空间。这个现象被称为局部性原理。

小提示

虚拟内存技术是采用虚拟空间独立编址、操作系统负责把一个大的虚拟空间的内容分阶段装入实际内存中运行的技术。

虚拟内存技术实现的基础是局部性原理和地址重定向技术。

虚拟内存技术使程序在一定程度上不再受物理内存大小的限制。

前面说过每个进程都可以使用一个无限大的虚拟空间。实际中，每个操作系统都按照地址长度确定了虚拟空间的大小。例如，32位的Linux操作系统的虚拟空间大小就是4GB。那么如果把4GB的虚拟空间分成512字节大小的页，应该能分多少个页？那就应该是$4\times2^{30}/2^{9}=4\times2^{21}=8$M个页。假定每个页的页表项占用4个字节，那么你算算一个进程的页表应该有多大？应该是8M×4B＝32MB。也就是说一个进程光页表就要占用32MB。你一定会说现在很少有程序有4GB那么大，所以页表中很多项都是空的，没有意义。而且即使程序真的有4GB那么大，根据局部性原理，在某段时间内，CPU也不会访问

到所有程序，也就是不会用到全部的页表项，把它们都放到内存中，是不是太浪费空间了？你想得很对。实际上，操作系统通常会把页表也看作是在虚拟空间中。整个页表也被分页，每次系统只装载页表的一部分。例如，32MB 的页表被分成 32M/512K＝64K 个 512KB 大小的页。系统按照需要装入这 64K 个页表页中的一部分。当访问到在内存页表中没有记录的页面时，操作系统再为这些页面建立页表项。也就是说，现在页表不再全部放在内存中了。而且放在内存中的页表页也不再连续存放了。可是，这就需要一个类似页表的数据来描述哪些页表页已经在内存中了、哪些还不在、在内存中的页表页放在什么地方。也就是需要一个页表的页表。这个页表的页表被称为页目录表。页目录表中的项指出某一个页表页是否在内存中，以及在内存中的页框号。这种机制被称为多级页表。多级页表实际上是采用虚拟内存技术管理大页表的一种方式。

小提示

分页技术可以实现虚拟内存。
多级页表机制使内存中只存放大页表的一部分。

采用虚拟内存技术后，当要访问某个虚拟地址对应单元的内容时，系统通过页号找到页表中相应的项时，并不能直接取用该项中的页框号，而是要先检查该页表项中一个称为“存在位”的标志位。如果存在标志位被设置了，则表明该页面已经被装入到内存中，这个页表项中的页框号是有效的；否则，表明要被访问的页面还没有被装入内存，因此这个表项中的页框号是没有意义的。要提醒你注意的是，内存单元用两种状态来表示 0 和 1，它并没有第三种状态表示该单元未使用。所以，没有使用时，内存单元要么表现出是 1 要么是 0。因此，读取内存单元时一定要注意：任何时候你都能读出东西来，即使你什么都没有往里面写过。内存单元中可能是上次使用后留下的结果，或者就是一个随机的状态。

当要访问某个虚拟单元的内容但发现它所在页的页表项表明它还没有被装入内存时，就会发生一个被称为“缺页异常”的事件。异常是计算机技术中一个常用的术语。系统在做一件事情时，突然发生了某件必须处理的事情，于是要暂停当前工作，去处理这件事情，然后如果可能再回头继续做前面做的事情，这就是异常。因为要访问还没有装入内存中的内容，所以该条指令无法继续执行，因此就发生了异常。这时系统要执行处理缺页异常的代码。这段代码的作用就是把需要的内容装入内存并设置相应的页表项。等到缺页异常处理完之后，刚刚因为缺页而暂停的指令又可以重新执行了。

如果采用多级页表，当要把虚拟地址转换为物理地址时，因为要找到页表项，必须先找到这个页表项在内存中的位置。所以，要先根据虚拟地址得出页号，然后根据页号计算出该页表项位于哪个页表页中。之后再根据这个页表页号去查找页目录，从中找到页表项在内存中的位置。接着才能像前面一样，读取页表项、找到页框号、计算出物理地址、访问物理单元。如果要访问的页表页还没有在内存中，那么也要发生缺页异常，由异常处理程序来装入这个页表页。只不过这次不是从外存装入，而是系统直接创建一个新的页表页。因为页表项并不存在于外存上，如果内存中没有，实际上就是还没有创建。

小提示

访问未装入内存的虚拟单元的内容时，会发生缺页异常。
缺页异常负责申请内存，把需要的内容装入内存，并更改相关的页表项。

在实现虚拟内存的系统中，有两个时刻可以装入页面。一是在页面使用之前，这被称为预装入。另一时刻是在页面被访问时，被称为按需装入。采用预装入方式，访问页面时，页面已经在内存中，所以访问速度很快。但并不是所有预装入的页面都可能在最近一段时间内被使用，所以那些未被访问的页面所占用的内存空间可以说是一种浪费。而按需装入则不存在浪费空间的问题。但是访问页时，因为页不在内存，进程必须等待缺页异常处理程序将页装入后才能继续执行。因而会浪费时间。通常操作系统会综合利用这两种方式。在创建进程时，系统为每个进程预装入一定数量的页。当进程执行到一定阶段需要新页时，系统再按需要装入页。外存的访问速度很慢。访问硬盘所用时间大部分是在磁头寻找数据位置时发生的。一旦磁头定位之后，读取少量字节和一大块数据时间则相差不大。因此，操作系统在按需装入要访问的页时通常会捎带把后面的页也预装入一些。因为这样也多花不了多少时间，甚至时间就是一样的。按照局部性原理，这些与所需页相邻的页可能很快就会被访问。因此，这样做可以减少缺页异常的发生次数，节省时间。

小提示

预装入和按需装入是虚拟内存系统中装入页面的两种方式。
通常操作系统把两种装入方式结合起来用，减少系统时间开销。

采用了内存管理技术后，你有没有注意到我们引入了一个很大的时间开销？以分页系统为例，当要访问某个单元的内容时，必须先读取页表，才能知道这个单元在内存中的具体位置，然后才能到内存中读/写所需内容。而页表是放在内存中的，所以读取页表是一次访存动作。这样本来只需要访存一次的动作，现在因为地址转换读取页表而变成了两次访存。也就是说所有的访存动作时间都因此而加倍。而如果再采用页面目录的话，那就变成了三次访存。你还记不记得前面说过，比起 CPU 的执行速度，内存的访问速度可要慢多了。因此，这会导致系统的效率成倍降低。有什么办法？使用缓存！对。正因为这样，多数 CPU 中都设置了一个专门用来存放页表的缓存，放置最近经常用到的页表项。

我们知道页表中的页表项是按照页号的顺序组织的。因此，查找某页的页表项时，并不需要从页表头一项一项去对比页号，通过页号就能够计算出该页表项在页表中的位置，直接读取那个位置的内容就行了。因此，页号也就不必在页表项中出现了，这部分空间就省下了。但是页表缓存中只保留了一部分页的页表项，因此，这些页表项的页号不再与其位置有一一对应关系了。所以，缓存中的每个页表项都必须带有该项所对应的页号。而且在缓存中也不能再直接通过页号计算出页表项的位置了。如果要查找某页的页表项是否在缓存中，也只能逐项去比对页号了。这种查找办法可太慢了。为此，存放页表的缓存硬件本身支持一种特殊的功能——它能够同时比对缓存中的所有单元。也就是把串行的逐项比对变成并行的同时比对。这样查找整个缓存的速度就和比对一条页表项的时间相同了。这种缓存

被称为高速关联缓存。关联存储器的另外一种解释是以其中某一存储项内容作为地址来存取的存储器。在有的系统中这种页表缓存也被称为 TLB(Translation Lookaside Buffer)。

从这里可以看出,计算机系统的某项功能,如查找比对页表项,可以用软件实现也可以用硬件实现。一般硬件实现的速度较快,但是成本也高些。所以,系统的设计者通常需要考虑平衡多种因素,然后在软硬件实现之间找到一个平衡点。

有了高速页表缓存,操作系统在转换虚拟地址时,会先到缓存中查找该页的页表项。如果找到,就从里面读出对应页框的物理地址进行地址转换。如果在缓存中没有找到,那么操作系统还需要到内存的页表中去读取页表项。如果页表项中的"存在位"表示这个页已经在内存中,那么系统就通过里面的页框信息转换地址,并可能把该页表项放到页表高速缓存中。如果页表项的"存在位"没有被设置,那么就会发生缺页异常将该页装入,然后填写页表,同时也可能将该页表项放到高速关联缓存中。高速缓存中放置哪些页表项,由专门的算法来决定,这些算法的好坏会决定缓存的效率。

小提示

计算机系统使用高速关联缓存来保存页表项,以减少地址转换时读取页表增加的访存时间。

现在来考虑最后一个问题,当系统要装入一个页而内存中并没有空闲页框可用时,操作系统应该怎么办?解决办法很简单:把一些不用的页从内存中换出。有时为了防止内存用光的事情发生,操作系统也会在还有一定空闲页框时就选择一些页换出。有些操作系统还可能限制每个进程能够使用的最大内存数,当进程使用的页框数达到界限时,操作系统也会把该进程的一些页淘汰出去。

为减少缺页异常的发生,在换出页时,最好选择那些以后再也不会用到的页。如果没有这样的页,就选择那些下次使用时间距离现在最远的页。要实现这个想法,操作系统需要知道将来要使用的页顺序,然后查找目前内存中的页哪些在这个序列里没有再出现过,或者出现的时间离现在最远,把这些页替换出去。这种策略被称为最优策略(Optimal Strategy,简称 OPT)。但是,通常操作系统是不能预知将来要发生的事情的。有些进程的代码执行序列是由实际执行环境决定的,如分支跳转指令就是根据程序运行当时的状况决定下一步动作的。因此,最优策略通常只是作为一个最好的标准用在理想的实验环境下评测其他实用的淘汰策略。

既然最优策略无法实现,我们只好另辟蹊径。最简单的办法就是直接换出最早装入的页。这被称为先进先出(First In First Out,FIFO)法。先进先出策略实现起来很容易。操作系统将内存中的页按照装入顺序排队。需要换出页时,取出队列头的页换出即可。但实验表明 FIFO 的效率并不像想象的那么好。人们通过实验还发现 FIFO 策略的一个缺点。按照道理,如果内存中可以存放的页数增多,发生缺页异常的可能性应该随之减少。但是,FIFO 策略却反常:内存中可装入页数增加了,缺页异常数反而也增加了。

FIFO 策略的效果有点儿不尽如人意,OPT 策略又没办法实现。因此,我们还需要想出别的办法来解决淘汰问题。既然 OPT 策略是最好的,那么我们就想办法尽量实现它。OPT 策略不能实现的原因是不能知道将来进程要访问的页序列,那么是不是可以找一些办

法来猜测一下。根据这个想法，人们提出了一种被称为最近最少访问的策略（Least Recently Used，LRU）。这个策略的核心思想是：如果一个页很久没有被访问了，根据局部性原理，可以推测在将来它被访问的可能性也会比较小。因此 LRU 算法选择未被访问时间最长的那些页换出。为了能够比较页未被访问的时间，操作系统需要为每个内存中的页维持一个计时器。当页被访问时，计时器被清 0，否则它就随时间增长。当操作系统需要淘汰页时，就比较页的计时器，选择时间最长的页。实验表明 LRU 的效果比 FIFO 要好。

但是 LRU 策略的实现开销非常大。因为内存中可装入的页数量很大。你可以算算，如果内存大小是 512MB，每个页的大小是 512B，那么内存中可以装入的页数是多少？为每个页设置一个计时器，开销非常大。管理这些计时器要花费很多时间。对这么多计时器的时间进行比较，更是件很可怕的事情。因此，在实际中，通常使用的是 LRU 策略的一个简化形式。操作系统为每个页设置一个标志位，表示这个页最近是否被访问过。因此这个标志位被称为访问位。访问位通常也被放在每个页的页表项中。当某个页被访问时，访问位就被设置为 1。操作系统定期将所有页的访问位清 0。当操作系统需要挑选页换出时，它就选择那些访问位为 0 的页。这样可以将开销大大降低，而且能收到与 LRU 相似的效果。

选好要换出的页之后，操作系统需要把页写回到外存，然后把要装入的页读入到换出页原来在内存的页框中，并更改这两个页的页表项。我们看到这里涉及外存的读和写，因此会很耗时。如果要换出的页没有被改写，那么它和外存中原来的页是一模一样的，也就没有必要再把它重新写回到外存。因此我们得到启示：在选择换出的页时，要尽量选择那些没有被改动的页。这样换出这些页时就不用写外存了。要注意的是，在计算机中让出一段内存并不需要把内存中的内容清空，而只要操作系统标记它没有被占用就行了。这和我们生活中的实体仓库有一点儿不同。所以，换出没有被改动的页，系统只要改动该页的页表项，并在系统中标记其原来占用的页框未被使用就行了。为了找到这种没有被修改的页，操作系统在每个页的页表项中又增加了一个标志位，称为“修改位”，来表示页是否被修改过。这样当选择页换出时，除了检查页的“访问位”之外，操作系统还会检查页的“修改位”。

小提示

当系统中缺少空闲页框时，系统需要选择一些页换出。常用的淘汰算法是近似的最近最少被使用策略。

选择换出页时，选择没有被修改的页可以省去写外存的时间。

附文：

微软官员回眸开发 Windows 2000 的艰辛历程

摘自微软中国网站

http：//www.microsoft.com/china/windows2000/news/rev.asp

随着 Windows 2000 工作组结束这项微软历史上最艰巨的开发任务，公司 Windows 业务部高级副总裁 Brian Valentine 和项目经理 Iain McDonald 怀着喜悦的心情向记者畅谈了

一年来所取得的业绩及所经受的挫折。

记者：有多少人参与了 Windows 2000 的开发？

Valentine：我们核心部门的成员有 2500 人。不过，微软的每一位人员都为 Windows 2000 作出了贡献。不仅仅限于为该项目工作的全职员工。

记者：哪类人员参与其中？

Valentine：微软的每位员工都作出了巨大奉献。包括程序开发员、测试人员、管理人员，还有后勤服务人员。真是名副其实的全公司项目。在微软，我们所信奉的文化是每个人都同样重要。因此，尽管工作性质千差万别，但是都是这项事业所不可或缺的。我们还与联合开发计划(JDP)和快速部署计划(RDP)的首批采纳者密切合作，并向预览版测试人员发送了几千份 β 版软件，征求他们的反馈意见。无论在微软内部还是外部其他公司，都有许许多多的人对这个项目作出了贡献。

记者：核心开发部的成员多长时间开一次会？

McDonald：我们有一个会议室，我们称之为“作战室”。这就是项目管理中心。项目的关键人员经常在这里会面，通常是每周七天。根据项目的进展情况有时一天的会晤达三次之多。我们之所以开这么多会议，往往是为了集中兵力研究新出现的重大课题。这样，我们就能够审查我们所取得的进展及所面临的挑战，并让每位成员及时掌握项目现状。

记者：你们是否有一些特别有趣的故事愿与我们共享？

McDonald：在完成像 Windows 2000 这样的大规模项目过程中无疑会有不少逗人的事情发生。我们总是想方设法放松一下自己，特别是在气氛变得异常紧张的时刻。我记得有一次我们正在集中精力推出临时性的 β-3 版本。我们把推出日期定为今年的 4 月 1 日，而且决心予以实现。最后，我们终于如期完成任务。就在该版本最终完成的前一刻，我们部门全体人员在咖啡室聚集一堂。在会上，Brian 说：“我现在就在这里签字批准交付这个版本。”于是，他就抓起记号笔在咖啡室的墙壁上写下名字。突然，这群人发疯似的纷纷拿笔在墙上签下自己的名字以庆祝他们的胜利。几分钟内，一面墙壁就写满了上千个姓名。然后，一些人找来油漆和辊筒，把他们的名字印在另外几面墙上。一时间，咖啡室变得一塌糊涂。那天真痛快。反正咖啡室需要重新粉刷——我们只是把粉刷时间提前了些。几天后我们就把这一切都掩盖了起来。

记者：开发部是如何渡过最艰难的时刻的？

McDonald：在以前我们也完成过大型项目，因此我们对前景有所准备。从项目启动之日起，每个成员都以最佳状态全力以赴。此外，作为项目经理，我设法确定若干短期目标。他们能够以跳跃方式获得成功的回报，这一点很重要。他们必须感受到最终胜利目标在向他们招手——下一个目标就在前头。

记者：在开发 Windows 2000 过程中有哪些重要里程碑？

Valentine：在这么大规模的一个项目中，必然会有许多里程碑式的事件。最大的事件要数 β-3 版本的发布，因为这是微软迄今为止所发布的最大预览版。当时，我们感到异常兴奋，因为我们知道距离最终目标不远了。第二个里程碑就是我们开始将 Windows 2000 交付微软公司内部及一些重要客户使用。这意味着该产品已作好向广大公众发布的准备，也意味着人们信赖我们的软件，愿意用它来改善他们公司的经营。对于我们来说，这就是无上荣誉。

记者：你是怎么确定 Windows 2000 已经可以交付生产了？

Valentine：我们需要首先达到若干测试指标。我们每天都在一个一个地解决难题。在这个过程中，难题数量逐步减少。这项软件每天都要经受强化测试，而且负荷量逐步升级。此外，Windows 2000 已在微软的信息技术基础设施上运行了几个月。所以，我们知道可以放心了。

记者：在开发过程中你们面临过哪些重大挑战？

McDonald：挑战主要来自心理方面。有时当你正处于项目的开发中期，会感到永远也达不到目标似的。特别是当你错过一个预定的目标日期，这种绝望心情就会油然而生。不幸的是，在项目接近最终目标时，我们比预定日程落后了几天到一周的时间。但这是为了确保软件的最高质量。不过，每当没有遇到挫折而未能完成预定目标，有时就会出现目标遥遥无期的心态。

记者：Windows 2000 中有哪些因素确保其质量？

Valentine：我们采用几项措施来确保质量。我们观察了该系统的实际运行状态。为此，我们密切观察了公司内部 100 台服务器的工作状况。我们在公司内部及客户的系统中进行测试，以期发现任何可能出现的错误，并且仔细地对运行状态进行跟踪。我们还进行了强化试验，每个晚上在百忙之中的 2000 台计算机上运行，模拟二至三年的计算机使用量。然后，我们设法解决所出现的每个问题。我们非常重视一切质量问题——比以往任何时候都重视。

记者：与以往的 Windows 版本相对比，Windows 2000 的性能如何？

Valentine：运行要快得多。我们有一个特别工作组，专门研究如何改进性能。我注意到媒体及新闻报道都对性能的改进作了肯定性评价，许多报道都证实了这一点。人们一旦使用这个产品，就会立即发现在性能方面获益匪浅。

记者：在你对各项目的记忆中，有什么最美好的事情？

McDonald：就个人而言，我的最美好回忆是在“作战室”内的经历。参与该项目的核心活动真是妙不可言。有许多有趣的事。这里所举行的会议绝对不会令人厌倦，不会出现大家坐成一圈一言不发的情景。而是高声嚷嚷，天翻地覆。我真的喜欢与这些人共事，进行交流。

4.3 信息存储的管理

本节讲解操作系统怎样把信息存储在外存上，并在需要时帮助我们找到以前存储的信息。

我们已经知道内存是易失性存储材料制造的，计算机断电之后内存中的信息就全部消失了，所以不能够用它存放需要长期保存的信息。我们写的程序或文档，都希望能够保留一段时间。否则，每天重写一遍程序，每个人最后都会把计算机扔掉。这个任务当然就交给了断电也能保持信息的外存，如磁盘、磁带。

可是，怎样在外存上存储信息，操作系统还需要对其进行管理吗？我们通过一个例子来思考这个问题。假定现在没有计算机，也没有外存。存储信息的载体就是纸张。假设你把自己的全部作品都写在了 A4 白纸上。这些白纸被你小心地存放在书柜里。你写的东西很

多，包括一部短篇小说的初稿，为 Linux 操作系统写的补丁程序，还有下学期的学习计划，以及写给 IBM 老总的自荐信等，足有上千页。现在到了交数据结构作业的日子，你要从这堆纸张中找到写有数据结构作业的那几张纸。这时你发现一个不小的问题：数据结构作业已经淹没在数千页的纸张中了。更糟糕的是，你发现同一内容的纸张甚至都不是挨着放的，而且你还没给每类内容的纸张分别编写页码。于是你只好一页一页检查，根据每页上的内容判断它是不是数据结构作业。经过乏味、漫长的艰苦工作，你终于从几千页纸中找出了那三页数据结构作业。这件事情使你变得聪明起来。在查找数据结构作业时，你同时把其他属于同一内容的纸张分出来，并且分别编上页码，而且还用钉书器把它们装订在一起，并在外面装了个封面写上一个题目，如“Linux 操作系统内核补丁”。这样下次再找时就不用再一页页地翻了，只需要对照名字一本一本查找就行了。你把这样一本一本的内容称为文件。过了一段时间，你的文件越来越多，渐渐有上百本之多。你发现查找一份文件花费的时间越来越多。这时聪明的你又开始想办法。通过分析，你发现你是按照每个文件产生的顺序直接把它放在文件堆上的，也就是这些文件是按照产生的时间顺序叠放的。而查找文件时，往往是按照文件的内容或者说名字查找的，而这和时间一般没有直接的关系。找到问题之后，你便开始着手解决这个问题。既然通常是按照内容查找，那么你就把内容相似的文件放在一堆，并且在上面放了一张纸，写上这类文件的内容，如“Linux 相关”、“个人资料”等。后来你发现商店里面有一个叫做文件夹的东西，可以把这些相似内容的文件都放进去，并且外面还有一个插标签的地方，可以写文件夹的名字。于是你就买了足够多的文件夹，把所有文件都分门别类地放进去。事情终于变简单了。这次再要查找数据结构作业，你只要找到名字为“作业”的文件夹，然后再在里面找名字为“数据结构作业”的文件就行了。

说了这么多该回来说说计算机外存上存储的信息了。如果不考虑其他因素，在外存上存储信息并不复杂。只要在硬盘上找到可用的空间，通过磁头把要存储的 0/1 信息转换成磁信号记录在硬盘上就行了。就好像在纸上找到空白的地方在上面写字一样。不过，通常我们不是把信息存储在外存上就结束了。存储信息的目的是为了将来使用它。当要使用外存中的程序时，例如一个提醒服务软件，必须要做的事就是把它从外存中找出来然后装入内存。需要在外存上存储的信息非常多，如接收电子邮件的程序、编辑相片的程序，还有个人简历，以及数据结构作业等。如果存储信息时什么都没有考虑，只是找到空闲空间就写。那么现在要在外存中找到这个提醒软件，只能逐个字节地去判断，看看哪些 0/1 串是这个提醒软件的代码。这和前面在几千页纸中找作业是一样的。所以必须要把外存上的信息管理和组织起来，才能方便快捷地找到它们。

与前面例子类似，操作系统也把相关的信息组织成文件，并要求用户为每个文件起个名字，如“个人简历”。用户通过文件名字告诉操作系统他要访问的信息。为了提高查找文件的速度，操作系统提供手段让用户把文件分门别类组织到不同的文件夹中。每个文件夹有一个名字。同一文件夹中相同文件名的文件是没有办法区分开的，因此操作系统会要求用户必须想出不同的名字。不同文件夹中的文件可以使用相同的名字，因为通过文件夹名能够把它们区分开。

文件夹还有另外一个名字叫做目录。这个名字也很好理解。这好比在前面的例子中，你用一张纸记录下文件夹中所有文件的名字，以及这些文件在文件夹中的位置等信息。然后把这张纸放到文件夹的最前面。这张纸实际上就是一个目录。每个文件夹正好有一个目

录。如果文件夹/目录太多了，查找起来也不方便。我们还可以用类似的方式，把文件夹分类，然后放在更大的文件夹中。同样，用一张纸记录大文件夹中的小文件夹的名字，以及它们的位置，然后放在大文件夹的最前面。这样就形成上一级目录。我们可以根据需要决定创建多少级目录。操作系统中的目录也起到类似的作用，也有类似的组织形式。实际上通常在外存中文件夹/目录也被存储成一个文件，称为目录文件。它里面记录着在该文件夹下的所有文件的名字和位置等相关信息。

与现实中文件例子不同的是，计算机系统中文件的内容在外存上的存放位置可能是不连续的。实际生活中，通常把一个文件的所有内容都放在一起装订起来。这是因为我们用的白纸是散页的，每页白纸之间没有固定顺序，所以可以按照内容进行移动，而且白纸一般是不重复使用的。而外存空间则不同，它是要重复使用的。随着存储信息的增加和删减，外存上可用的空间就不再是连续的了。而外存单元在物理上是不能移动的，我们不能像拿纸张一样，把空闲的几段盘片，重新挪动位置，让它们挨着（你也打不开硬盘）。要想把一个文件在外存上连续存储，只有把硬盘单元的内容移动，让空闲空间连续。这就是我们有时候会做的磁盘碎片整理。可是移动外存上的内容涉及外存单元的读和写，很耗时。磁盘碎片整理一般要花上几个小时的时间。所以，通常操作系统会想办法支持在外存上不连续的若干个区间存储一个文件。

可以看出，对于需要长期存储的信息，操作系统要做两方面的管理工作。首先，为了便于找到存储的信息，操作系统要把信息按照某些特征以某种形式组织起来，如文件和目录。此外，操作系统还需要对外存上的空间进行管理，就像内存管理一样，以便存储信息时能够找到合适的空闲区间，同时还要提高外存空间的利用率，减少信息访问的时间。

文件的组织方式被称为文件系统。不同操作系统组织文件的方式不同。这样就出现了很多不同的文件系统，如 DOS 操作系统的 FAT16、Windows 的 NTFS 和 FAT32 以及 Linux 中的 EXT2 等。通常也把操作系统中用来管理文件系统以及对文件进行操作的机制及其实现称为文件系统。

小提示

操作系统通常把外存上的信息组织成文件的形式。
文件的组织方式被称为文件系统。
通常操作系统中用来管理文件系统以及对文件进行操作的机制及其实现也被称为文件系统。

4.3.1 外存管理和文件的物理组织

现在我们来了解操作系统怎样在外存上放置存储的信息。在外存上存储信息与使用内存有很多类似的地方，所以多数操作系统也采用了与内存管理类似的观念来管理外存。在内存管理技术上，我们经历了固定分区、动态分区、分页等技术发展过程，其中使用较多、效率较高的是页式内存管理。因此，在外存管理上，很多操作系统都使用了类似页式内存管理的技术。

我们这里只介绍最常见的外存设备——硬盘的管理。根据页式内存管理的思想，操作系统首先把硬盘分成大小相等的区域。这些区域不再叫做页框，而是被称为物理块或块。块的大小随操作系统不同而不同。前面讲过硬盘是由一个个盘片构成的，通常以扇区为单

位来读写。因此，一般常见的物理块大小为扇区的长度，如 512B 或者 1KB。有时块的长度也会大于扇区的长度，但多数时候是扇区长度的倍数，一般倍数都是 2 的整数次幂。按照这样的单位分配磁盘空间，可以减少访问磁盘的次数，从而减少从外存读取/写入数据的时间。

当用户要求在硬盘上存储信息时，操作系统首先要查找是否有足够的空闲块。为此，操作系统需要事先统计系统中的空闲块。空闲块信息的记录形式会直接影响查找空闲块的速度。常见的空闲块记录形式与管理内存空闲页框的方法类似，有位图、空闲块链表以及空闲块分组链表等。

在查找和分配空闲块的过程中，操作系统还要决定另外一件事情：是否一定要分配连续的块。通常情况是，除非用户要求连续存储(这时操作系统要找到足够多的、连续的块来存放这些信息)，否则操作系统并不坚持要把一个文件的信息放在前后相连的物理块中。

把文件放在硬盘上之后，为了下次使用时能够找到它，操作系统需要记录文件存储的位置。对于连续存储的文件，操作系统只要记录其在外存上的起始块号以及文件长度就可以了。连续分配方式和内存管理中的动态分区方式类似。因此它的缺点也是很明显的：外存上有时会很难找到足够大的连续空闲块，除非进行外存单元内容的移动，而这又太耗费时间。因此，考虑到效率问题，多数操作系统并不在硬盘上连续存储文件。这种情况下，为了找到分散在外存上不同物理块中的文件内容，操作系统需要记录每个文件在外存上的分布情况。也就是操作系统要记录每个文件所使用的全部块号及其顺序。

有些操作系统把一个文件所用物理块的号码按照其所存内容的顺序排列在一起写在另外一个文件中。这个文件被称为原始文件的索引文件。索引文件通常存放在外存上的特定物理块中。然后操作系统在另外一个表格中(例如，文件分配表)记录系统中所有文件名及其对应的索引文件所在的块号。这样操作系统就可以通过文件名找到分配给文件的实际物理块了。可以看出，文件的索引和内存管理使用的页表功能有些类似。这种在外存上组织文件的形式被称为索引结构。图 4-18 是采用索引文件记录文件位置的示意图。图中，一个文件被存储在 8，23，24，12，27 几个物理块中，而这些信息被存放在物理块 4 中。操作系统只要在文件分配表中记录该文件的名字以及索引块号 4 即可。

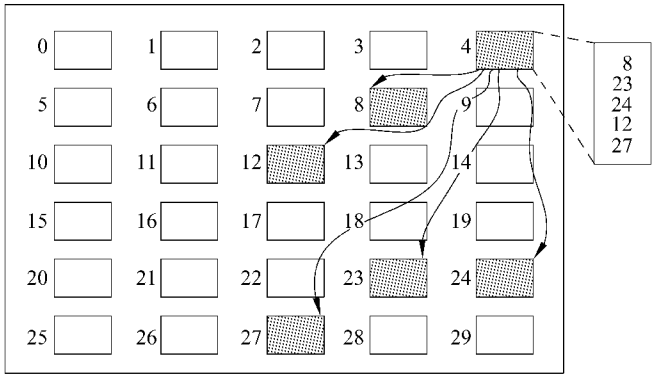


图 4-18　索引结构示意图

索引文件是用专门的地方存放文件所用物理块的块号。另外，还有些操作系统用存放文件内容的物理块来记录其所用的块号及其顺序。具体的做法是，在文件的第一个物理块

中的某个特定位置记录第二块的块号;在第二块中再记录第三块的块号;这样一直到最后一块为止。这种文件的组织形式被称为链表结构。这种情况下,操作系统的文件分配表中只要记录文件名、文件的第一个块的块号以及总块数即可。图 4-19 是与图 4-18 相同的一个文件用链表结构组织后的示意图。

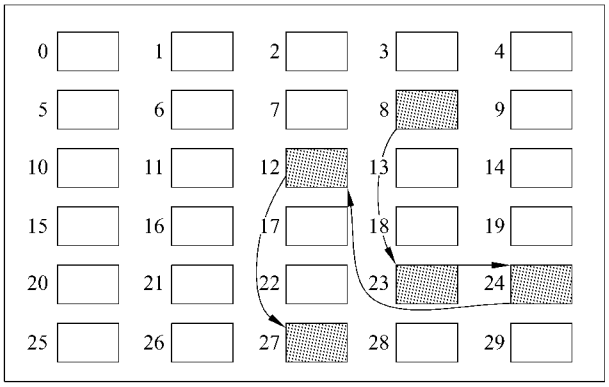


图 4-19 链表结构示意图

按照链表结构存放文件,如果要访问文件中某个位置的信息,那么必须从第一个物理块开始按照链表一直找到信息所在的块。因此这种结构比较适合存储经常被顺序访问的文件,而对于那些经常要被随机访问某部分内容的文件,则不太适合。因为访问信息会花费很多时间。所以,从这里可以得出结论:文件的访问速度,不只和硬盘的存储速度有关,还与文件在硬盘上的组织方式有关。这还是前面说过的一个道理,提高效率不能只靠提升硬件,管理也要跟得上才行。我们把上述这些文件在外存上的存储组织形式称为文件的物理组织结构。

小提示

操作系统采用类似页式内存管理的方法来管理外存。

文件在外存上的存储组织形式称为文件的物理组织结构。常见的文件物理组织结构有连续结构、索引结构以及链表结构。

连续结构指文件被存放在外存连续的物理块中。

索引结构用额外的物理块来记录文件使用的块号及其顺序。

链表结构在存放文件的每个物理块中记录该文件使用的下一个物理块的块号。

4.3.2 文件操作

用户对于其存储的信息通常有以下一些操作需求。首先,当用户要保存信息时,他要“创建”一个文件,并给它起个名字。创建好文件之后,用户要往这个文件中“写”信息。在写的过程中,用户希望能够“改变写的位置”,如回退到前面写过的内容中插入一行。写完之后,用户会“关闭”文件。一段时间以后,用户希望看看以前保存的信息。于是他“打开”文件,查看其中的信息,并作了一些“修改”之后,把文件关闭。在打开文件之前,他需要“浏览”硬盘上都有哪些文件,通过文件名找到要查看的文件。用户还会“运行”一些可执行文件。对于不需要的文件,用户会希望能够把它从硬盘上“删除”。总结一下,通常用户需要对文件

进行的操作有创建文件、打开文件、阅读文件、写文件、执行文件，还有在文件中变动读写的位置、浏览文件、删除文件等等。一般这些操作都是通过应用程序来完成的。但是，应用程序最终是依靠操作系统提供的文件操作接口完成这些任务。例如，当我们要保存文件时，通常只是按一下应用软件中的一个“保存”按钮。我们并不需要到硬盘上去查找空间，然后调用写硬盘的指令，把这个文件写到硬盘中选好的位置，然后再在某个地方记录这个文件的内容都存在硬盘上的哪些地方了。实际上应用软件也不做这些工作。这些工作都是由操作系统来完成的。

操作系统中负责管理文件的那部分模块被称为文件系统。文件系统负责定义文件在外存中的组织和保存方式，以及向用户提供访问文件的手段，即接口。不同操作系统的文件系统不同。它们提供给用户以及应用程序的接口也不同。但是，一般都提供以下几个主要功能：打开文件、关闭文件、读文件、写文件，以及定位到文件中的某个特定位置等。

支持这些操作的基础是文件系统要能够通过文件名找到外存上的文件。我们已经知道文件系统通过自己维护的记录表就能够找到文件在外存上的位置。但是，当系统中的文件越来越多时，在表格中逐个比对名字查找文件，也变得越来越慢。而且为了能区分开文件，文件名是不能重复的。随着文件的增多，这个要求也显得越来越苛刻。对于多个用户共享外存的计算机来说，不同用户的文件都混杂在一起，使用起来也非常不方便。所以，就像前面提到的，文件系统开始支持目录。用户可以把相关的文件放到一个目录中。多个目录还可以被组织到更高一级的目录中。每个目录也由一个名字来标识。同一目录下的文件和子目录不可以重名，但是不同目录下的文件和子目录则可以同名。最高一级的目录被称为根目录。图 4-20 是 Windows 操作系统的一个目录结构。Windows 操作系统把每个逻辑盘符(如 C 盘、D 盘)作为根目录。这样使用 Windows 操作系统的计算机，一般都有几个并列级别的根目录。而有些操作系统则只支持一个根目录，如 Linux。

文件系统支持目录结构后，文件被组织到目录中，文件系统只记录文件名及其在硬盘上的块号就不行了。因为不同目录下的文件名可能是重复的。而且，这样记录，文件多了，查找起来也慢。所以，文件系统要在硬盘上记录这种层次结构的目录关系。怎么记录呢？很多文件系统把同一个目录下的文件/子目录的文件名以及物理位置信息记录在一起，也以一个文件的形式存放在硬盘中。这个文件被称为目录文件，它的名字就是这个目录的名字。这个目录文件就好像前面例子中放在文件夹最前面的那张目录纸一样。而目录文件存在外存上的位置信息则记录在其所在的上一级目录的目录文件中。这样重复一直到根目录为止。在这种情况下，如果文件系统想在外存上找到一个文件，只有文件的名字就不行了。文件系统必须知道这个文件所在的各级目录的名字，才能从最高一级目录文件开始逐级查找，最终找到这个文件在硬盘上的位置。

文件所在的各级目录名写在一起称为文件的路径。为什么叫路径，后面介绍文件定位时你就会清楚了，因为正是沿着这条路径的指引才找到文件在磁盘上的位置。通常路径从根目录开始，一直写到文件名为止。每个目录名之间用一个分割符隔开。不同的文件系统可能会使用不同的分隔符，如 Windows 下用反斜杠“\”，而 Linux 则相反，用斜杠“/”来分割路径中的目录。现在，如果我们要表示图 4-20 中的文件 biologon. dll，那么就要写完整它的路径，即 C: \Program Files\IBM fingerprint software\bioglon. dll。从根目录开始的路径被称为绝对路径。多数操作系统都定义了两个特殊的目录名。一个点“.”，表示当前工作目

录。两个点“..”,表示上一级目录,或者说父目录。在有的操作系统中,如果表示文件/目录位于当前目录中,那么写文件名时可以省略路径。例如,假定我们已经进入 C:\Program Files 目录,那么当前目录就是 C:\Program Files。现在如果要访问文件 bioglon.dll,那么在指定文件名时可以只写 IBM fingerprint software\bioglon.dll,意思就是位于当前目录下。这里把 IBM fingerprint software 称为相对路径。

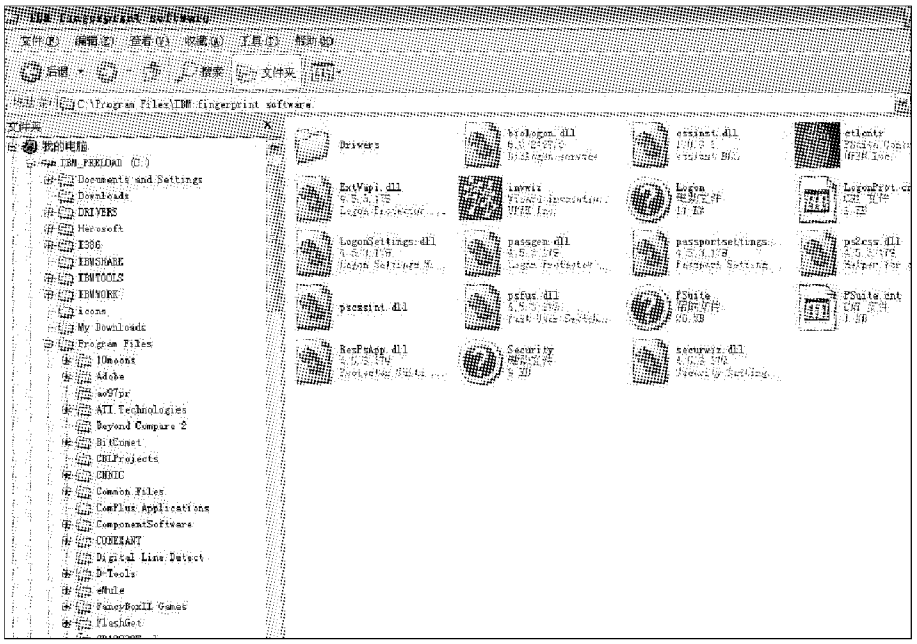


图 4-20 Windows 操作系统的目录结构

现在你对文件系统怎样管理文件已经有了一些基本的了解,但是这些信息感觉还有些零散,下面我们通过一个例子来完整体验一下文件系统怎样管理外存上的文件。我们以类 UNIX 操作系统(如 Linux 操作系统)的文件系统为例。

类 UNIX 操作系统通常采用索引结构在外存上存储文件。在系统启动时,外存被分成大小相同的物理块。每个文件使用的物理块号记录在文件的索引中。系统在外存的特定位置留出一些物理块,专门放置所有文件的索引,这些块被称为索引块。

类 UNIX 系统中既有目录又有文件。文件中存放的是需要存储的具体内容,存放在分配给文件的物理块中。目录也以文件形式存储在硬盘上。每个目录文件的内容是该目录下的子目录或文件的名字及其相关信息,其中最重要的信息是子目录或文件的索引文件在外存上的位置。

当要访问文件时,操作系统必须先找到文件的索引块,然后才能找到文件内容在硬盘上的具体位置。文件索引的位置存储在文件所在目录的目录文件中,所以要想找到文件的索引,必须要找到它所在目录的目录文件。而目录文件和其他文件一样是存储在外存上的。要从外存上读取这个目录文件,那也必须先找到它的索引块。这样操作系统就必须知道存有这个目录文件索引块位置的上一级目录的名字,然后在这个上一级目录的目录文件中根据目录名字去查找该目录的索引块地址。可是上一级目录的目录文件在哪儿呢?操作系统

需要先找到它的索引块。这个循环到什么时候为止呢？答案是一直查找到根目录。因为根目录已经没有上一级目录了。那么到哪里去找根目录文件的索引块的位置呢？为了解决这个问题，类 UNIX 操作系统把根目录文件的索引块位置记录在一个被称为“超级块”的地方。每个文件系统都有一个超级块，并且超级块在设备上的位置是固定的。对于硬盘来说，超级块是在第二个物理块上（第一个块上放的引导块，在后面我们会介绍引导块的作用。到时候你会明白为什么把它放在第一块上）。超级块中除了有根目录文件索引的位置外，还有文件系统的一些参数，如外存一共被分了多少块、每块有多大等。在类 UNIX 操作系统下，每个存储设备在使用前都需要安装。安装过程的主要工作就是读入存储设备的超级块，在内存中建立一个相应的数据结构记录超级块中的信息。这样当我们要访问某个文件时，可以很快找到根目录文件的索引，并最终找到文件的内容。

图 4-21 用一个虚拟的示例描述了类 UNIX 系统通过文件名定位文件位置的过程。假定我们要访问磁盘上的文件/root/src/hello. c。路径最前面的“/”表示根目录。类 UNIX 操作系统首先要从超级块中找到根目录文件的索引位置。在图 4-21 中，超级块中给出根目录文件的索引块号是 4。这个号码即是根目录索引文件在系统的索引块特定区域中的位置。根据这个位置信息，系统找到根目录文件的索引块并从中读出根目录的索引。然后系统再根据这个索引给出的块号，如图 4-21 中的 106，201，300 等，到相应的物理块中读取根目录文件。之后，系统在根目录文件中查找子目录名 root。找到以后（在图中的 106 号物理块中），系统就可以取得 root 子目录文件的索引块的位置信息，图 4-21 中是 19。系统再根据这个信息到外存索引块区域找到 root 子目录文件的索引。索引中给出了 root 子目录文件在外存中的块号，即 188，198，211 等。然后系统再从这些块中存储的内容里查找子目录名 src 及其索引文件所在位置，图中在数据块 198 中找到索引号 48。根据这个信息系统找到 src 子目录文件的索引文件后，见图 4-21 的索引块 48，系统再根据索引文件中的内容找到 src 子目录文件。在这个目录文件中含有文件名 hello. c 及其文件索引的位置，见图 4-21 中数据块 224。通过这个索引，系统最终得到 hello. c 文件的位置，即 245 号物理块和 267 号物理块。实际系统中文件的索引并不会占用一个物理块，通常是多个索引在同一个物理块上。但是通常也用索引号来表示索引位置。只不过在这种情况下，一个索引号对应的是

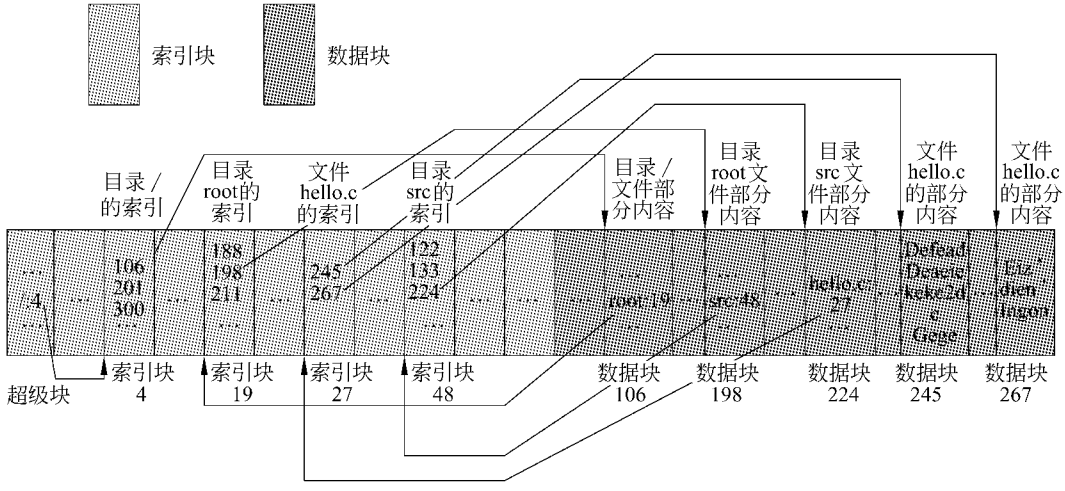


图 4-21　类 UNIX 操作系统的文件查找示意图

某个物理块中的某个位置。

小提示

文件系统要为用户和应用程序提供访问文件的手段，通常是一些接口函数，如创建、打开、读、写、删除等。

文件系统把文件组织成层次的目录结构。

文件系统把目录信息也以文件形式存放在外存上。

目录文件中记录该目录下的文件以及子目录文件的索引文件的位置。

文件系统要在外存上找到文件的内容，必须知道文件的路径。

从上述过程可以看出，为了访问一个文件，系统需要沿着文件的路径多次访问外存。这会花费非常多的时间。所以，文件系统一般都把根目录索引文件的内容读到内存中备用。同时，文件系统都支持“当前工作目录”的概念。当前工作目录的索引也被缓存在内存中。如果访问当前目录下的文件和子目录，就可以直接通过内存中的索引找到当前目录的目录文件，然后在其中查找子目录名或者文件名就行了。因为当前目录的索引已经被保留在内存中，系统就不用再从根目录开始找它了。这样可以大大减少访问文件的时间。因此，访问当前目录下的文件和子目录也不需要给出从根开始的绝对路径，只要给出从当前目录开始的相对路径即可。

通常用户要访问某个文件的内容时，操作系统都要求先打开文件。打开动作是通过操作系统提供的接口函数实现的，通常接口函数名叫做 open。当应用程序想要读取一个文件时，它也要使用操作系统提供给它的工具，比如读函数 read。在调用函数 read 之前，操作系统要求必须先调用 open。Open 函数要求用户给出文件的名字和路径。操作系统根据这些信息按照上述的办法找到文件的索引信息，然后把文件的索引放到内存中，并在调用 open 的这个进程的进程控制块中记录下索引信息在内存中的位置。这样进程接下来读写这个文件时就可以直接使用内存中文件的索引找到外存上的文件了，就不用再到外存上一步步查找了，这样可以节省很多时间。这就是访问文件之前必须先要调用 open 的原因。

下面来说说文件的读/写问题。读/写是文件的两个重要操作。操作系统提供的读/写接口函数一般有读文件 read、写文件 write、在文件末尾添加 append 等。在经常接触的操作系统中，如 Windows 和 Linux，读写文件的单位都是字节。也就是文件内容是以字节为单位组织的。在介绍管道时，我们提到过“字节流”这个词。意思是说，操作系统并不考察文件内容的逻辑。无论什么内容，在操作系统看来都是一个个互不相干的字节。因此进行读写操作时操作系统只要求用户指明从文件的什么位置开始读写多少个字节。文件中的位置也是按照字节来编号的。通常操作系统为每个文件维护一个当前访问指针，即当前的访问进行到文件中的哪个位置。当前访问指针的移动单位也是以字节为单位的，如向前或向后移动若干字节。我们把文件内容的组织形式称为文件的逻辑结构。除了按照字节流组织文件内容外，一些大型的商用操作系统会支持按照记录来读取文件中的内容。文件中相关的内容会被组织成一个记录。每个文件都是由这样的一个个记录构成的。操作系统允许用户按照记录来读写文件。这样做可以提高文件的访问速度，且便于用户使用。在这里我们不对此进行详细介绍。大家可以查阅相关资料来学习。

虽然文件是按照字节组织的，但是对外存的访问并不是以字节为单位的。我们已经知

道对外存成块访问可以提高效率。因此，尽管文件系统提供给用户的接口函数是按照字节流的形式读写文件的，但是，当实际从/往外存读写时，却是按照数据块的形式进行的。操作系统负责把字节流转换成块。即在写文件时，通常操作系统要等到数据足够写满一个块时，才写回到外存。同样当用户要读取一个字节时，操作系统会把该字节所在的整个块都读取到内存，然后把用户要求读取的字节交给用户。

操作系统提供给应用程序的接口函数还有在文件中查找 seek 以及对目录的一些操作，如改变当前工作目录、移动到上一级目录、创建目录、删除目录等。

各种应用程序，如编辑软件 Word、Gedit，在操作系统提供的这些接口基础上，把对文件的操作进行封装和加工，例如把 open 函数以及 read 函数的调用封装到一个“打开”按钮上。这样普通用户就能够很方便地使用文件，而不用了解文件管理的任何细节，更不用知道什么接口函数了。

小提示

使用当前目录可以减少定位文件时访问外存的次数。
文件内容的组织形式被称为文件的逻辑结构。常见的逻辑结构有字节流式文件和记录式文件。
文件系统负责把字节流或记录形式的用户操作转换成对外存的成块访问。

4.3.3　文件保护

除了帮助应用程序以及终端用户在外存上存储信息外，文件系统要做的另一件重要事情就是保护这些信息。保护信息要做的事情很多，尤其是在多用户的计算机系统中。例如，因为计算机是被多个用户共享的，所以外存上会同时存储着不同用户的文件。一个用户的文件，如个人信息文档，是不希望被别人看到的。这时，操作系统必须保证别的用户看不到这些文档。此外，系统中一些非常重要的文件，关乎系统的存亡，是坚决不能允许普通用户随便使用和更改的。因此，操作系统必须想办法防止用户更改重要文件。

为了实现对文件的保护，文件系统通常让用户为每个文件设置权限。每个文件的访问权限记录在系统为文件维护的相应管理文件中，如文件的索引文件。操作系统根据用户设置的访问权限对文件实施相应的保护。在用户打开文件时，除了提供文件名外，操作系统一般还要求用户给出打开文件的目的，如读、写等。操作系统要先根据管理文件中的权限信息查看用户是否有权利进行要求的操作。如果权限信息表明这个文件只允许读，而用户声明要写这个文件，那么他的打开动作就会以失败告终。

因为 Windows 操作系统是从个人计算机上起家的，而个人计算机多数情况下是一个用户独享的，所以，文件保护起初在 Windows 操作系统上体现并不明显。通常用不同用户名登录计算机，对硬盘上的所有文件也都是能够一目了然的。不过，现在 Windows 已经开始支持越来越多、越来越灵活的文件保护手段了。多用户系统出身的 Linux，或者说类 UNIX 操作系统在文件保护上则从一开始就做了很严格的工作。例如，当用户以某个普通用户账号登录后，他只能够看到自己主目录下的内容以及一些其他用户授权其看到的文件。通常他并不能够随意安装程序，更不能向别的用户目录，尤其是系统的一些目录，添加或者更改文件。如果想做这些事情，必须使用具有更高权限的账号。因为这些多用户操作系统的首要任务就是，保证在多人使用计算机的情况下，计算机是安全的、可靠的，且稳定的。

下面来看看操作系统需要为用户提供哪些种不同的文件访问权限。我们从文件主人以及系统对文件保护的要求出发来分析这个问题。

首先有些文件的主人可能不希望别人知道他的目录下有哪些文件，更别说访问这些文件了。因此，操作系统要允许用户设置一个权限，表示别人对某些目录或文件“无”任何权利。

此外，有些文件的主人并不在乎别人知道自己的目录下存储了哪些文件。但是，他仍然不希望别人对他的目录/文件进行任何动作，如读/写等。为此操作系统需要提供一种称为“可知”的文件权限类别，表明这个目录/文件可以被别人看到。

有些文件是能够执行的，有些文件是不能够执行的。但是，即使是可执行文件，有时用户也会希望能够禁止一些人运行自己的这些文件。尤其是一些重要的、与系统管理有关的应用程序，系统不希望普通用户能够使用它。因此，操作系统应该允许用户设置或者禁止文件的“执行”权限。

还有一些文件，可以让别人阅读，但是主人并不希望别人更改它。因此，对于这些文件，用户希望设置它们的权限为“只读”。

如果文件可以被更改，那么用户希望设置的权限为“可写”。

还有一些文件只能在末尾添加内容，而不能更改前面已经写入的内容，如日志文件。那么使用这种文件的用户会希望操作系统能够支持这个需求，设置一个“末尾添加”的权限。

前面提到的都是对文件内容本身的一些权限考虑。既然可以给每个文件设置权限，那么操作系统就要提供一个方式来控制谁可以更改文件的权限。这种权力被称为“改变保护”。

对于文件来说，还有一个非常大的操作权力，就是“删除”。这是一个非常危险的动作。因此，用户也希望通过操作系统提供的保护机制能够把这个权力控制在自己的手中。

每个文件的权限对于不同的人应该是不同的。为此，多数操作系统都支持针对不同的用户设置不同的文件权限。设置权限的过程其实也可以理解为对某些用户访问文件的授权。在Linux中，通常一个文件的权限被分成三个部分。例如，当我们使用命令“ls-l”时(这个命令的功能就是详细列出当前目录下文件以及子目录的信息)会看到除了文件名之外，屏幕上还会出现形如“-rw-rw-r--”的字母串，如图4-22所示。这个串一共有10个字母，后面的9个字母便是文件的权限。字母r表示文件可读，w表示文件可写，而x则表示可以执行文件。这9个字母被分成三组。最前面的三个字母表示文件主人的权限。中间的三个字母表示那些与文件主人在同一个用户组的用户的权限。而最后三个字母则表示其他人的权限。从图4-22中可以看到，test1文件允许主人读写，但是不允许执行。而与文件主同组的用户同样可以读写这个文件。除此之外的其他人则只能读这个文件。Linux还提供了一个访问列表工具，它可以增加其他权限和功能，如设置某个具体的用户对某个文件的访问权限。

```
[test1@mail fs1]# ls -l
-rw-rw-r--+ 1 test1 test1 10 Feb 16 13:52 test1.txt
```

图4-22 Linux的文件权限

Windows操作系统也有对文件权限的支持。我们可以在Windows的“资源管理器”窗

口中选定一个文件，然后单击鼠标右键，在弹出的快捷菜单中选择“属性”项，如图 4-23 所示，就可以看到一个如图 4-24 所示的属性窗口。选择“安全”标签，就会见到对这个文件进行权限设置的窗口，如图 4-24 所示。从图上可以看出用户能够按照用户名或者组名来设置对该文件的权限。

图 4-23　Windows 的资源管理器

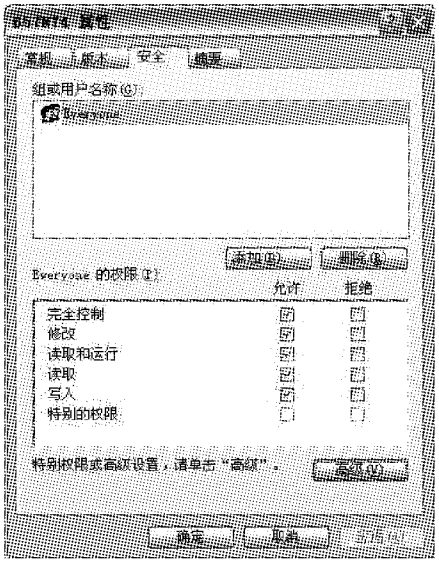


图 4-24　Windows 的文件权限

前面列出的是我们能够想到的对于文件保护的一些需求。当然，实际情况中可能还有好多别的特定要求。在某些环境下，有些需求可能并不迫切，而在另外的场合则不同。因此，由于操作系统的目标不同，每个操作系统对文件实施的保护权限也不同。并不是所有操作系统都实现前面提到的文件保护权限。不同操作系统提供给用户进行文件保护的手段和方法也各不相同。操作系统的实现者一般会仔细分析系统的应用环境，再对需求和成本进行平衡然后做出最后的决策。

小提示

保护文件是文件系统的一个重要任务。

文件系统提供手段让用户设置文件/目录的使用权限。文件系统在用户使用文件/目录时，检查用户对该文件/目录的使用权限以此来保护文件。

常见的文件使用权限有无、可知、读、写、执行等。

4.3.4 虚拟文件系统

用过 Linux，尤其是早期 Linux 系统的人都知道，要在 Linux 下使用 U 盘或者软盘，甚至光盘都必须要用类似下面的命令先安装一下：

```
mount  -t  vfat  /dev/sda1  /mnt/usb
```

为什么要这样做呢？是不是 Linux 设计的有问题？其实，这正是 Linux 的一大优势。它的优势不在于多了这个安装的步骤，而是因为它可以支持多种不同的文件系统。这也是在 Linux 上使用外存之前必须先执行安装命令的原因。通过安装（实际上还有注册），Linux 就可以访问由其他操作系统的文件系统产生的文件。虽然在 Windows 下访问 U 盘不需要安装，但是，如果 U 盘是在 Linux 下用它的 EXT2 文件系统格式化的，那么在 Windows 下就不能访问。而反过来，如果在 Windows 下用它的文件系统 NTFS 格式化的 U 盘，在 Linux 下注册和安装之后就能够访问。本节就来详细说说这件事。

从前面几节的内容我们发现文件系统做的事情可真不少：管理外存、组织外存上的文件、管理文件的目录结构、管理文件的逻辑结构、提供用户访问文件的接口，以及保护文件等。不同的操作系统会根据其目标选择不同方式来实现这些功能。不同文件系统管理外存的方式可能会不同，提供给用户的接口也可能不一样。例如，Linux 采用索引文件来记录文件在外存上的位置，而 Windows 则采用了完全不同的一种方式。实际上 Windows 的硬盘分配方式与 Linux 都是完全不同的。在接口方面，各个操作系统提供的文件操作的种类和数量也不相同。有些操作系统虽然提供同样的接口，但是接口的实现方式却不一样。正因为这样，目前市面上并存着很多不同的文件系统，如 Linux 的 EXT2 文件系统、Windows 的 NTFS 文件系统、OS/2（IBM 为 PC 开发的一个操作系统）的 HPFS 文件系统、光盘使用的 ISOFS 文件系统等。不同的设计和实现方案使各个操作系统的文件系统不能兼容。常常在某个操作系统下创建的文件无法被另外一个操作系统识别。用某一种操作系统的文件系统格式化的外部存储设备，在另外一个操作系统上也无法识别。例如，Windows 操作系统就不能识别磁盘上的 Linux 分区。这种情况给用户带来很大不便。有时一个系统中的用户会需要访问另外一个系统下的某些文件。例如，在 Linux 下可能要访问用 MSDOS 的文件

系统 FAT 格式化的软盘，而访问 ISOFS 格式的光盘则更是常事。为此，各个操作系统都在想办法解决文件系统的兼容问题。

Linux 操作系统的设计者想出了一个办法。他们设计了一种称为“虚拟文件系统”的文件管理器。这个虚拟的文件系统分成两个部分。下面部分与具体的文件系统相关；而上面部分则与具体的文件系统无关。

与具体文件系统无关的部分向用户提供了访问文件系统的统一接口。这组抽象的接口由一组标准的、抽象的文件操作构成。它们以系统调用的形式提供给应用程序，如 read()、write()、seek()等。用户对任何文件系统都使用这组统一的接口，无论底层的文件系统怎样实现这些操作，甚至不实现某些接口函数都可以。当用户调用这些操作访问具体的文件时，虚拟文件系统把这些操作映射成该文件所属文件系统对这些函数的实现。也就是说，虚拟文件系统通过提供抽象的、统一的调用接口来屏蔽底层不同的文件系统。

虚拟文件系统记录每种文件系统具体实现这些文件操作的函数的位置。在进程打开文件(调用 open())时，虚拟文件系统记录这个文件所属的文件系统。当用户要对文件进行操作时，系统根据记录的信息找到该文件所属文件系统实现这个操作的函数并调用。这样虚拟文件系统就把文件和该文件所属的具体文件系统所提供的操作函数挂上了钩。虚拟文件系统中与具体文件系统有关的部分就是各个文件系统对统一接口函数的实现。

总结起来说就是，虚拟文件系统为用户提供统一的文件访问接口，它负责把这些访问接口映射到具体文件系统的实现上。图 4-25 说明了虚拟文件系统的机制。

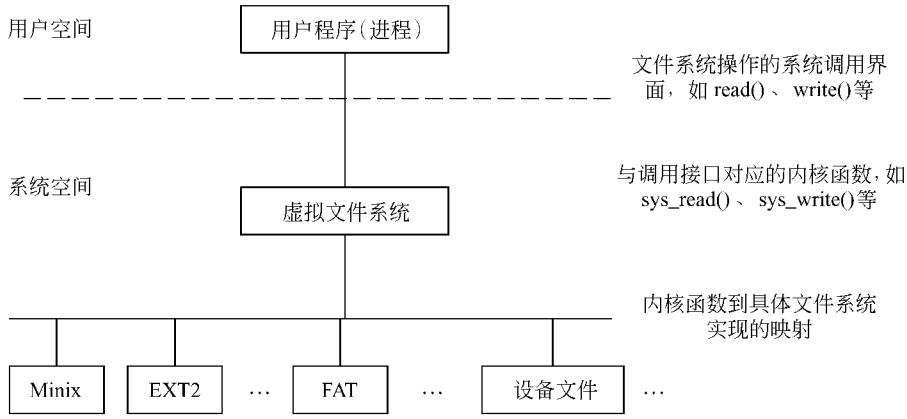


图 4-25　Linux 虚拟文件系统示意图

虚拟文件系统就好像一个系统总线，上面有若干个插槽。每个具体的文件系统就好像是插在插槽上的接口板。虚拟文件系统的插槽可以将不同接口板的访问方式转换成统一的一个面向用户的抽象接口。这种方法不但使 Linux 能够支持多种不同的文件系统，而且还能支持这些文件系统之间的相互访问[2]。

尽管 Linux 可以支持很多文件系统，但是在 Linux 启动时并不是默认支持所有可支持的文件系统。因为这意味着它要有识别所有文件系统的能力，并带上所有文件系统的对统一接口函数的具体实现。它要轻装上阵。因此，Linux 启动时只能识别几种常见的文件系统。要想让 Linux 支持某种文件系统，首先必须向系统注册这种文件系统。就好比要想把接口卡插到总线插槽上，总线需要首先有能力识别这种接口卡才行。注册一个文件系统的

过程就是教给总线识别这种接口卡的知识。因为不同的文件系统的文件组织形式不同。有关文件系统的组织信息都记录在每个文件系统的超级块中。但是不同文件系统的超级块的结构也是不一样的。因此,文件系统注册的一个重要工作就是把读取这个文件系统超级块的函数告诉虚拟文件系统。有了这个函数,Linux 就可以从超级块中获得该注册文件系统的信息,也就能够识别这个文件系统了。除了提交超级块读取函数外,注册过程还会告诉虚拟文件系统,这个文件系统对文件操作的具体实现函数的模块是什么。虚拟文件系统把这些注册信息记录在系统的一个文件系统类型表中。这个表格里面记录着系统中注册过的,即目前系统可以支持的文件系统。每种不同的文件系统在文件系统类型表中占据一项。每项中记录的内容有文件系统的名字、实现文件系统的模块名,以及读取文件系统超级块的函数指针等。除了从外存上读取超级块中的信息外,超级块读取函数还有另外的工作。通过它可以找到该文件系统对文件操作的具体实现函数指针,如读文件操作 read()的具体实现代码的位置。所以,有了文件类型注册表中的这些信息,Linux 就能够很轻松地使用这些文件系统了。你可以通过查阅 Linux 的文件/proc/filesystems 找出已注册的文件系统。

在 Linux 下有两个时机可以注册文件系统。一是在系统初始化时。这时文件系统的实现代码就在操作系统代码中。编译操作系统时,这些代码就与操作系统编译在一起。在系统启动初始化时,这些文件系统就会被加入到文件系统类型表中。

另外,用户也可以在系统启动后,把文件系统当作一个模块插入到系统中。Linux 提供了一种模块机制,允许用户在系统启动之后,向操作系统核心插入代码,而不用重新编译操作系统。这与安装设备驱动程序是一样的思想。Linux 提供了命令来插入和卸装模块。插入模块的命令是 insmod,删除模块的命令是 rmmod。你可以用这些命令来向操作系统注册或者注销一个文件系统。当你调用插入模块命令 insmod 插入一个文件系统时,系统就把这个文件系统的信息写入到文件系统类型表中了。也就完成了文件系统的注册过程。

仅仅具备识别接口卡的能力还不行,只有把接口卡插在插槽上,才能够使用它。同样,在使用一个位于 U 盘上的文件系统之前,我们还必须安装这个文件系统才行。也就是把这个文件系统插入到虚拟文件系统这个总线的插槽中。Linux 提供了一个安装命令 mount 来完成这个任务。

在安装文件系统时,必须告诉 Linux 几件必要的事情。首先,必须告诉它要使用的文件系统在哪个设备上。然后还要告诉它要安装的文件系统是什么类型。另外,在 Linux 下,文件系统被组织成一个树型目录结构。整个目录结构只有一个根。在 Linux 系统启动时,会自动把默认文件系统(一般是 EXT 系列的文件系统,如 EXT2 或 EXT3)的磁盘分区作为 Linux 文件系统的根文件系统。所有后安装的文件系统必须安装在这个根文件系统的某个目录下,即成为这个目录树的某个树枝。所以,在安装新文件系统时,最后还要告诉系统我们希望把这个文件系统安装在哪个目录下。

例如,如果我们想在 Linux 下访问在 Windows 操作系统下存放在 U 盘中的一个文件,那么就需要先安装产生 U 盘上这个文件的文件系统。具体的安装命令如下:

```
mount  -t  vfat  /dev/sda1  /mnt/usb
```

这条命令中,"-t vfat"表示文件系统类型是 vfat,就是 U 盘上的文件系统的类型。Vfat 是 Windows 使用的一种文件系统类型。接下来我们要告诉 Linux 要安装的文件系统在 U 盘

上。U 盘在 Linux 通常被仿真为 SCSI 设备。在 Linux 下用设备文件来表示设备，设备文件通常放在目录/dev 下。因此，用“/dev/sda1”表示文件系统所在的设备 U 盘。这里 sda1 表示是第一块 SCSI 设备的第一个磁盘分区。最后，要告诉 Linux 我们准备把 U 盘上的文件系统安装到/mnt/usb 目录下，所以我们给出最后一个参数“/mnt/usb”。通常都把文件系统安装在/mnt 目录下。运行完这条 mount 命令之后(必须是特权用户，才有权力运行这条命令)，我们就可以在/mnt/usb 目录下看到 U 盘上的文件了。当然前提是我们已经在系统中注册过 vfat 文件系统(通常 Linux 会默认支持 vfat)。

那么，在安装过程中，系统都做了什么呢？在我们输入 mount 命令之后，系统首先就到文件系统类型表中去查找是否有我们要用到的这个文件系统类型，即 vfat。找到之后，系统就会调用类型表中给出的超级块读取函数，到我们给出的设备上，即参数/dev/sda1 指出的 U 盘上，取出该文件系统超级块中的信息。同时，系统还利用这个超级块读取函数获得该文件系统各种文件操作(如读文件、写文件)的实现函数指针，并记录下来。

当用户程序要访问一个文件时，首先必须调用“打开”操作。在虚拟文件系统中，“打开”操作所做的另外一个工作就是把文件所属文件系统对统一接口函数的具体实现函数指针记录在这个文件相关的数据结构中。而这些指针就是系统在前面的安装过程中得到的。这样，当用户接下来对文件进行操作时，如读文件，系统就可以通过打开文件时记录的这些指针找到具体的实现函数了。

小提示

虚拟文件系统由与具体文件系统无关的统一接口以及具体文件系统对接口的实现两部分构成。
虚拟文件系统通过提供抽象的、统一的文件操作接口来屏蔽底层不同的文件系统。
虚拟文件系统负责把统一接口与具体文件系统的实现函数关联起来。

4.4　外设的管理

计算机系统中的外设非常多。这些外设的功能不同，使用方法也不同。为此，操作系统这个平台必须给用户提供帮助，让用户能够方便快捷地使用这些不同的外设。前面曾介绍过，不同设备的控制方法不同，为方便起见，应用程序并不直接和外设打交道，而是借助于操作系统提供的手段使用外设。操作系统为用户及应用程序提供访问设备的接口函数。这些函数屏蔽了不同设备的硬件区别。尽管系统中设备的种类很多，且可能会变化，设备控制方式也不同，且也可能会变化，但是操作系统会尽量保持提供给用户的接口稳定不变。这样用户就不会因为这些不同和变化而手忙脚乱。操作系统帮助用户做到以不变应万变。而操作系统则是通过设备厂商提供的驱动程序来实现“应万变”——对不同种类、变化的设备进行识别、管理和控制。

对于支持多用户、多进程并发执行的现代操作系统来说，仅仅为用户提供使用外设的帮助是不够的。操作系统要负责设备的分配，即在多个要求使用外设的进程中决定谁可以使用外设，以及使用哪一个外设。操作系统要满足用户的要求，尽量做到公平，同时要考虑设备的使用效率。因此，对于外设管理，操作系统要做的事情主要有三个：为用户提供方便快捷地访问外设的手段；分配和管理外设；以及提高外设和系统其他部件的使用率。第三个任

务的实现可能会贯穿在第一和第二个任务的实现过程中。但是也有一些措施是专门为实现第三个目标而实施的。操作系统完成上述三个任务的组成部分构成了操作系统的设备管理子系统，也称为输入输出子系统。

下面我们就来看看为了完成上述的三个任务，操作系统都想了些什么办法。

小提示

设备管理的三个任务分别是为用户提供方便快捷访问外设的手段、分配和管理外设，以及提高外设和系统其他部件的使用率。

4.4.1 外设访问接口

因为计算机系统外围设备种类繁多，且每种设备都有不同的特性，所以操作系统的一个任务就是要把各种硬件设备的复杂操作细节屏蔽起来，让用户能够以一个较简单、统一、相对固定的方式来使用这些不同的设备。经过多年的不断摸索，人们发现在逻辑上外设的特性与文件非常相似。因为使用外设的最终目的无非是从外设输入或者向外设输出一些信息。这与文件的使用目的一样。这个发现使种类繁多的外设世界与文件这一单一事物对应起来。这样提供统一的、以不变应万变的外设使用方式的任务就同支持异构文件系统类似了。

操作系统的设计者采用类似虚拟文件系统的思想完成了第一任务：屏蔽不同设备细节，为用户提供简单、统一、相对稳定的使用外设的手段。在这种思想下，设备管理子系统也被分成了上下两个部分。在与用户接口的上部分中，设备管理子系统向用户提供了一组统一的设备使用接口函数。不同设备的操作是不同的，有些设备仅能读，有的设备仅能进行写操作。操作系统设计者选择的这组函数集合，包括了对任何设备可以进行的所有可能操作。这些接口看起来像读写文件的接口，如它们的名字是“打开”、“关闭”、“读”、“写”、“移动当前访问位置”等，特殊的是多了“控制设备”接口。每个设备对接口函数都有自己的具体实现。这些实现的具体代码通常是在设备的驱动程序中。因为设备的功能不同，所以对这些函数的实现也不同。例如，同样是输出函数——“写”函数，打印机的实现与显示器的实现肯定不同，而作为输入设备的键盘则根本不实现写函数。设备管理子系统负责把这些公共接口的调用转移到特定设备的驱动程序上。设备驱动程序负责对设备进行真正的操作。驱动程序是设备管理子系统的下部分。与虚拟文件系统类似，公共设备接口是设备无关部分，而驱动程序是设备管理子系统中的设备相关部分。

在设备管理子系统中，“打开”函数完成的工作是为进程分配设备，并做好其他的准备工作以便进程能够管理设备的 I/O 操作。而“关闭”函数则释放占用的设备及其他相关软件资源。“移动当前访问位置”函数则可让进程对设备进行读写定位，使得进程能够对设备中的特定地址进行读写，如将磁带移动到一个可以读写的位置。“控制”函数是与设备相关的，例如，磁盘存储设备的“控制”函数可以完成设备断电/加电的任务，而显示器的“控制”函数则可以让屏幕反转，使黑像素变白，白像素变黑。

在早期操作系统中，设备的驱动程序不是独立于操作系统程序的，而是写在操作系统代码中的。只要有新设备添加到系统中，就要修改操作系统的代码：将新设备驱动程

序添加进去，同时修改公共接口函数与特定设备实现函数之间的对应关系。然后再对操作系统代码进行重新编译。因此安装驱动程序是一件相当复杂、技术含量非常高的工作。安装驱动程序的单位和组织必须有一套操作系统源代码的复本，以及添加设备所需要的知识。

这种方式对系统设备的增删带来的制约显然太大了。后来，人们想出了一个办法解决这个问题。这个办法说起来也并不复杂。系统为每个设备维护一个表格。表格中记录着每个公共接口函数与设备操作具体实现函数之间的对应关系，也就是从公共接口函数名到具体实现函数地址的映射。系统提供特殊的系统调用让特权用户可在系统运行之后来生成或者修改这个表格。这样每次更改或添加新的外设时，都可以通过更改或生成这个表格中的项来让系统找到新的驱动程序，也就不用重新编译系统了。只是每新添一个设备都必须进行注册和安装驱动程序。实际上就是把驱动程序放置到特定位置，然后把这个位置添加到接口函数映射表中。这种方式被称为可重新配置的驱动程序。现代操作系统都采用类似的方式支持外设。这也是为什么新添一个设备必须安装驱动程序的原因。

小提示

设备管理子系统由与设备无关部分和设备相关部分构成。
设备无关部分向用户提供简单、统一、相对稳定的外设使用接口。
设备驱动程序是设备管理子系统中的设备相关部分。
设备管理子系统负责将统一的外设使用接口映射到设备驱动程序中对操作的具体实现。

4.4.2　设备驱动程序

设备驱动程序是直接与外设打交道的软件，是系统中直接操纵硬件动作的软件。驱动程序通过接口电路中的寄存器同硬件设备的控制器打交道来使用外设。这些寄存器包括命令寄存器、状态寄存器和数据寄存器。驱动程序正是通过这些寄存器来控制外设的。把设备命令写入命令寄存器，驱动程序可以控制设备发送/接收数据、开启/关闭；通过读取状态寄存器的内容，驱动程序可以了解设备的状态；数据寄存器中则是驱动程序与外设之间传递的数据。不同设备对命令寄存器中的命令以及状态寄存器中的状态的解释都不同。因此，驱动程序的任务就是把不同设备的操作进行抽象，并尽可能地让这些抽象相同或相似。例如，打印机驱动程序封装了如何控制打印机的知识，磁盘驱动程序专用于磁盘管理，但是它们都提供了“写”接口函数。当然，并不是每个设备都能实现标准接口上的每个功能，如打印机的驱动程序就没有实现“读”函数。设备驱动程序的设计是个严格的软件设计过程，设计者必须了解设备控制的所有细节。因此，驱动程序通常都是由设备厂商来设计和实现。

在输入输出系统一节中，我们介绍了系统与外设传递信息的几种方式。其实就是驱动程序与外设控制器之间传递信息的几种方式。当然信息传递方式不同，驱动程序的实现也不一样。在这些方式中，最简单的是程序直接控制方式。如果驱动程序采用程序直接控制方式与外设传递信息，那么驱动程序实现的“读”函数的具体流程大致如下[3]：

① 接收到操作系统传递过来的、应用进程请求的读操作之后，驱动程序查询设备接口的状态寄存器，确定设备是否空闲。

② 如果设备忙，则循环等待，直到设备变为空闲为止。

③ 把输入命令放入设备接口的命令寄存器中，从而启动设备。

④ 重复读取状态寄存器中的值等待设备操作完成。

⑤ 把设备接口的数据寄存器的内容复制给用户进程。

驱动程序要通过注册和安装最后变成操作系统的一部分。因此，操作系统对设备驱动程序的结构方面都有统一要求。但是，不同操作系统对驱动程序结构的要求是不一样的。因此，同一设备在不同操作系统下会有不同的驱动程序版本，一般是不能够通用的。

小提示

驱动程序通过外设接口中的寄存器与外设打交道。

操作系统对设备驱动程序的结构方面有统一要求，不同操作系统对驱动程序结构的要求是不一样的。

4.4.3 设备分配

现在我们来讨论进程请求使用外部设备，尤其是多个进程同时要求使用外设时，操作系统应该怎么办。因为系统中的设备资源有限，所以不是每个进程在需要时都能够立即被满足。因此，进程在需要设备时必须向系统提出请求，由操作系统的设备分配程序按照一定的分配原则和策略把设备分配给进程。如果设备正在被其他进程使用，那么申请设备的进程就必须进入相应的设备等待队列中排队等待。

常见的设备分配算法有先来先服务和优先级两种。在分配设备之前，操作系统要了解系统中设备的情况，如共有多少设备、都是什么、目前的状态怎样等。为此，通常系统会为每类设备设置一个设备表，登记这类设备中每台设备的状态。包含的内容一般有设备名、占有设备的进程号、是否已经分配、状态好/坏等。系统还会维护一个设备类表。里面记录着系统中共有多少类设备，每类设备的设备类名、总台数、空闲台数，以及该类设备的设备表在什么地方等。

系统中的设备按照本身的固有特性可以分为独享型和共享型两种。独享型设备每次只能分配给一个进程使用，如打印机、绘图仪。而共享型设备则可以由若干进程同时共享，如磁盘。操作系统对不同类型设备的分配和管理策略也不同。通常对于独享型设备，进程在使用前必须有明确的申请动作，使用后也必须有释放操作。而对于共享型设备，进程在使用前后并不用做特殊的操作，直接使用就行了。

因为系统中的资源有限，进程又很多，所以设备管理子系统在分配设备时，必须考虑设备的利用率问题。独享型设备分配给一个进程后，其他的进程如果要使用必须等待占用进程释放才行。不过，通常占用独享设备的进程并不是时时刻刻都在使用设备。独享设备因而时常处于空闲状态，利用率很低。为解决这个问题，一些操作系统用共享设备，如磁盘来模拟独享设备，如打印机。当用户需要使用独享设备时，系统就把模拟的设备分配给它，而不分配给进程实际的独享设备。例如，当进程要求使用打印机时，系统并不把打印机分配给进程，而是在磁盘上申请一段空闲区，然后把需要打印的数据传送到空闲区中，再把进程的打印请求挂到打印队列上。如果打印机空闲，系统就从打印队列中取出一个请求，然后从磁盘指定区域取出数据打印。由于磁盘是共享的，因此系统可以随时响应打印请求并把数据

缓存起来，这样就把独享设备改造成了共享设备。这种技术被称为虚拟设备技术。实现这一技术的系统被称为 SPOOLing 系统。SPOOLing 是联机同时外围操作的英文(Simultaneous Peripheral Operation On Line)缩写。

小提示

常用的设备分配算法有先来先服务和优先级两种。

设备按照物理特性可以分为独享型和共享型两种。

虚拟设备技术用共享设备模拟独享型设备，把独享型设备改造成共享设备，提高了独享设备的利用率。

4.4.4　中断技术

最后我们来看看为了提高系统的效率，操作系统在设备管理这里还做了什么工作、采用了什么技术。

在早期，计算机系统 CPU 的执行与 I/O 操作是串行的。一段时间以后，计算机设计者意识到，依赖于机械运动的 I/O 设备操作与纯电子开关的 CPU 计算相比，速度差距太大了，通常是在几个数量级以上。于是硬件和软件设计师开始寻找这样一种技术：CPU 计算可以不必等待 I/O 操作而持续进行。中断技术就在这种情况下产生了。采用中断技术后，在外设处理数据期间，CPU 可以去做别的事情。外设处理完数据后，给 CPU 发送一个信号，称为中断信号。CPU 每执行完一条指令都会查看一下是否有设备发出中断信号。如果有，CPU 就停下手头工作，处理发出中断信号的这个设备相关的事情。在第 2.5.3 节中已经提到过中断，现在我们再对中断有关的内容给出进一步的说明。

在计算机系统中，设备发送中断信号、CPU 检查中断信号、CPU 自动转入中断处理程序，这几个过程都是在硬件的支持下实现的。在支持中断技术的系统中，要有一个硬件标识来表示中断请求，这被称为中断请求标志位。CPU 每执行完一条指令都自动去检查中断请求标志位看看有没有中断事件发生。如果把所有设备的状态寄存器中的完成标志位连接到 CPU 的中断请求标志位，那么当设备控制器的完成标志位被设置时，CPU 的中断标志位也就被设置了[3]。由此，当 CPU 查看中断标志位时就会知道有设备完成了操作。如果 CPU 轮流检查这些设备的状态，就可以判断出是哪一个设备完成了工作。实际上，现在的系统都使用专门的硬件芯片来支持中断。一般可以支持一组中断请求标志位。我们常听到的中断号，就是这组中断标志中某一位的序号。系统一般会为不同的设备分配不同的中断号。但是，因为芯片能支持的中断标志位不是很多，所以中断号也是系统中的紧俏资源。对应于每个中断号，系统都保存了一份处理程序。一旦 CPU 发现某个中断标志位被设置了，就会去执行相应的处理程序。为了在需要时能够找到中断处理程序，这些处理程序的开始地址都被放在系统中特定位置的一个表格中。表格里的中断处理程序地址在以前的系统中也被称为中断向量。这个表格也被称为中断向量表。在现代操作系统中，记录类似信息的表格通常被称为中断描述符表[2]。

当发现有中断事件发生时，CPU 就要处理这个中断了，即进入中断响应过程。为此，CPU 首先必须放下手头的活。但是处理完中断以后，CPU 还要回来继续完成现在的任务。为了以后能够从放手的地方继续执行，系统必须保存接下来要执行指令的地址以及当前的

环境。这与进程切换时做的保存上下文工作类似。所谓环境主要指与CPU执行有关的一些寄存器的值，如状态寄存器和程序计数器。为此，中断响应过程的第一项工作就是保护被中断进程的现场。做好这个工作后，接下来就要分析中断产生的原因。然后根据原因转入相应的中断处理程序。只要将中断处理程序的开始地址送入程序计数器，并根据要求设置状态寄存器，便转入了中断处理程序。中断处理程序执行完毕，系统会再恢复被中断进程的现场，继续执行原来被中断的进程。

如果进程不希望被中断打断，那么它可以告诉CPU不要理会中断。这样CPU就不再检查中断标志位了。怎样告诉呢？支持中断的系统通常在CPU中的状态寄存器会提供另外一个标志位，称为中断禁止标志位。如果这一标志位被设置了，那么CPU就不会检查中断标志位了。通常中断处理程序的执行不希望被其他的中断信号打断，因而在中断处理程序的开始都会设置中断禁止标志位。

小提示

中断是指处理机在执行过程中，由于某些事件的出现，中止当前进程的运行，转而去处理出现的事件，待处理完毕后返回原来被中断处继续执行或调度其他进程执行的过程。

引起中断的事件称为中断源。

设备向CPU发出请求中断处理信号称为中断请求。

当CPU发现有中断请求信号时，中止当前程序的执行，并自动进入相应的中断处理程序的过程称为中断响应。

如果外设通过中断方式与主机进行信息传递，那么设备的驱动程序就要被分成两部分。一部分用来进行操作的初始化；另一部分则是中断处理程序，负责处理操作完成后的事情。

我们还是以“读”函数的实现来说明采用中断方式通信的驱动程序的大致实现流程。当一个应用进程要从外设读取信息时，系统把这个请求传达给设备的驱动程序的第一部分。这部分的工作过程一般如下：

① 查询设备接口的状态寄存器，确定设备是否空闲。

② 如果设备忙，则等待，直到它变为空闲为止。

③ 把输入命令放入接口的命令寄存器中，从而启动设备。

④ 睡眠。

这时驱动程序的前一部分就完成任务了。这时请求读设备的进程就进入阻塞状态，因而系统会调度其他进程来执行。当设备完成操作，即已经把用户要求的数据准备好了，它发出一个中断信号。CPU发现这个中断后暂停手头进程的执行，运行中断处理程序。中断处理程序确定是由哪个设备引起的中断后执行该设备对应的设备处理程序。这个设备处理程序可以看作是驱动程序的第二部分。它接下来做的工作包括如下内容：

① 把设备接口的数据寄存器中的内容复制到用户进程空间。

② 唤醒睡眠的进程。

然后，驱动程序就可以返回到调用它的进程，该进程获得要读取的数据，就可以继续执行了[3]。

小提示

每个中断都对应一个中断处理程序。

中断处理程序的开始地址被存放在系统中特定的地方。

当中断发生时，系统根据发生的中断号调用相应的中断处理程序。

采用中断方式通信的设备驱动程序被分割成两部分。一部分负责设备操作的初始化。另一部分是中断处理程序，负责在设备完成操作后，将结果和控制权返还给执行设备操作的用户进程。

4.4.5　缓冲技术

缓冲技术是外设管理部分用来提高“进程感受到的”外设访问速度的一个手段。因为外设的速度相对 CPU 来说会慢几个数量级，如果不采取措施，CPU 启动一个 I/O 操作引起的等待时间相当于可以执行成千上万条非 I/O 操作的时间，从而造成 CPU 能力的浪费。为此，一些操作系统专门在内存中留出一些空间作为缓冲。在进程并不需要 I/O 操作时，设备管理子系统还是让外设工作，因为进程还没有请求数据，所以系统先把这些数据放到输入缓冲区中。这样可以使得设备与 CPU 操作交叠执行。当进程需要数据时，可以立即获得。同样，当进程向外设输出数据时，系统先将信息保存在主存中的输出缓冲区中，然后在进程继续执行的同时将信息写入到设备。也就是说，外部设备本身并没有变快，通过设置缓冲区，在向外设写数据的情形，进程不用等待外设完成操作就可以继续下面的计算；在从外设读数据的情形，外设将数据提前写进缓冲区，从而使进程感到它能从外设读得很快，其实是从缓冲区中读的。

下面来看一个具体的例子。假定字符设备控制器用一个字节容量的数据寄存器保存提供给驱动程序的字符。当应用程序开始一个读操作时，驱动程序把读命令传送给命令寄存器。设备控制器根据命令指示设备输入一个字节到数据寄存器中。需要这个字节的调用进程等待设备输入操作结束后从数据寄存器中取走字符。这个过程看起来顺理成章，但是设备输入字符到数据寄存器的速度比起计算机的速度会慢很多。如果增加一个字节的数据寄存器作为硬件缓冲，让控制器在进程要求取字符之前就把字符放入到数据寄存器，就可以减少进程等待字符的时间了。控制器在进程读取字符前先把一个要读取的字符放到数据寄存器中。当进程提取已经提前读入到数据寄存器中的字符时，设备填充另外一个寄存器。这样如果设备读取下一个字符的时间刚好与进程处理上一个字符到请求下一个字符的时间间隔相等，那么就可以交叠进行了。这就是缓冲的作用。如果进程的处理速度和设备的操作速度相差太多，那么还可以通过增加缓冲的办法来缓解这个问题。这种技术也同样适用于面向块的设备，如磁盘设备。这种情况下，磁盘控制器中的每一个缓冲必须足够大到存储整个块的数据，而不是单个字节。其实在讲硬盘时，我们已经介绍过缓冲了。

同样在驱动程序中也可以增加一级缓冲来存放应用程序还没有要求的数据。这样一旦应用程序要求这些数据，驱动程序就可以用最快的速度把它们提交给应用程序[3]。

但是，不是在所有情况下增加更多的缓冲都能提高性能。有时使用缓冲是能够解决问题的，但是有些情况下则不然。这取决于应用程序和设备的特性。例如，如果应用程序的代码很紧凑，驱动程序总是能够预测到它将要使用的数据，并把它们事先准备好，那么缓冲的效果就很好。反之，则不然。驱动程序每次准备的数据都不对。那么缓冲的作用就一点儿

都没起到，反而增加了很多无用功。

小提示

缓冲可以在一定程度上缓解 CPU 与外设速度不匹配的问题，节省 CPU 的时间。

附文：

专访 Linus——“Reclusive Linux founder opens up”

Friday, May 19, 2006 Posted: 0954 GMT (1754 HKT)

新闻来源：美国 CNN 网站，

http://edition.cnn.com/2006/BUSINESS/05/18/global.office.linustorvalds/

译者：不详

今年 36 岁的芬兰人 Linus Torvalds 是 Linux 操作系统的创始人。他于 1991 年创建的 Linux 操作系统，不断获得互联网上众多志愿者“程序员”的支持。美国 CNN 的记者 Kristie Lu Stout(KLS)受 LT(Linus Torvalds)邀请，采访了 Linus。以下是采访内容。

KLS 问：现在的 Linux 开发过程中，你充当着什么样的角色？

LT 答：一开始，我是作为一名主要的 Linux 开发者。但如今我似乎已经变成了一个“通信中心”，我的主要工作就是收集大量外围开发者的开发代码或意见，然后和其他一些人一起讨论，再决定是否将新的代码和意见纳入 Linux 操作系统的核心中。

KLS 问：你的工作圈子里大概有多少开发人员？

LT 答：实际上我只同少数的人在一起工作，能和我直接交换意见的人大概只有 10～20 人的样子，这 10～20 个人会分别负责和世界各地的其他的“志愿开发者”做 Linux 代码开发交流，最核心 Linux 代码的开发人数不超过 50 人，如果将所有参与核心 Linux 代码开发的代码志愿者加在一起算的话，估计是 5000 来个人。这 5000 来个志愿者的 Linux 代码交流工作是通过 E-mail 来完成的。

KLS 问：你掌握了核心的 Linux 开发者和测试者，这 5000 来个开发人员能开发出新的 Linux 产品，他们的动机是什么呢？

LT 答：这些 Linux 开发人员每当在技术上有新的发现和突破，都会感到无比激动，他们对技术本身非常喜好，并不是因为痛恨微软才来参与开发 Linux 操作系统的。他们从事 Linux 开发完全是因为他们喜欢做他们爱做的事。

KLS 问：在互联网上冲浪的很多家庭主妇，很多都不知道 Firefox 或 Linux 是什么东西，你怎么看？

LT 答：开源软件产品已明确地走进人们的日常生活中，人们已经开始注意到开源的概念，这种开源概念下的产品优点很多，性能更好，更安全以及其他一些优点。除了微软的 Windows 产品，人们还可以选择 Linux 产品来完成许多工作。

KLS 问：让我们回到 1991 年，当时你完全公布操作系统内核代码的动机是什么？

LT 答：对于公开操作系统内核代码这件事，我没有做什么故意的考虑，因为当时我才21岁，在 Helsinki 大学，一半以上的时间我都是用来进行编程的，每完成一个编程上的技术挑战，我都会非常地激动。公开这些程序的代码，让别人能看到我做的每一步具体内容，使我的自豪感得到了充分满足。

KLS 问：你认为自己的工作有没有一些夸大的吹牛成分？

LT 答：当然有。我总喜欢独自一人在楼下的地下室里坐着工作，和其他技术专家类似，我们很不爱社交的特点是有些夸张的意味。

KLS 问：用企鹅作为 Linux 的标志和吉祥物，这是如何发生的？

LT 答：我感觉 Linux 应该需要一个很友好的吉祥物来弥补技术工程师和技术给人的生硬感觉，小狗、小猫这些动物太常见了，人们需要异域情调的动物，这样我就想到了企鹅，人们都喜欢企鹅，它很可爱，我们就决定用企鹅作为 Linux 的吉祥物，希望它能被所有人喜爱并可以轻轻地抚摸。我们还竞相设计出了许多的企鹅吉祥物形象，如今你可以在网上看到这些可爱的吉祥物了。

KLS 问：你是否想到过你创立的 Linux 操作系统如今会变得这么具有商业价值，而且很多人还因为 Linux 都发家致富了呢？

LT 答：如今我已经致富了。我不认为这是坏事。整个 Linux 商业市场繁荣起来了，我也很高兴。每月我都会收到很多捐助，但这些捐赠只是被我用在 Linux 技术发展方面，我和 Linux 商业市场上的那些市场业务人员没有瓜葛。

KLS 问：数年以来，受 Linux 这个开放源代码的软件项目的启发，也出现了像维基百科等免费开放的在线百科全书项目，以及众多的开源项目，形成了开源精神，开源哲学，你是如何看待这一现象的？

LT 答：开源精神并不是源自 Linux，在 Linux 出现之前就已经存在很多的开源软件项目和自由软件，Linux 不过是其中一个相对较大的软件项目，开源的概念也不是一个新概念，开源项目和科学发现有些类似，我们的成绩是积累在很多前辈科学家的研究基础之上的。

KLS 问：你最喜欢开源哲学的哪一个部分？

LT 答：这个问题很突然，我不知道，我想最漂亮的不在于开源代码这一部分，而是在于整个 Linux 社区里的人们自愿无偿参与技术开发的这种热情。这种热心是我一直从事 Linux 工作的最大策动力。

KLS 问：在 IT 技术领域，现在你就好像是当年的摇滚巨星……

LT 答：我没有注意到日常生活中的这些变化，我很少出席那些商业 Linux 会议，每年只出席一到两次 Linux 的技术会议，我是非常平常的人，我在我的小屋里做着我自己喜欢做的工作。

KLS 问：多长时间，你才会和你的 Linux 代码自愿贡献者有见面的机会？

LT 答：几乎很少有面对面交谈的机会，在我工作的城市里有几个同事，我们倒是经常见面，我们会一起去喝啤酒或吃早饭什么的。即使是在 Linux 技术大会等社交场合，大多数的 Linux 代码工作者，也会坐在会议桌上，用手提电脑上的 E-mail 系统来和大家交流代码，因为 E-mail 系统可以让大家对所讨论的代码有清楚的体现。

KLS 问：那么你认为面对面的软件技术交流，其实是一种做作，不能起到什么效果吗？

LT答：我认为是这样的。很久以前我就决定不去参加什么软件技术研讨会，我发现参会只会浪费我们程序工作人员的大量时间，对我们的软件发展没有什么帮助。大量实际的工作是被那些坐在椅子上的，尤其是程序员，花费了大量的编程时间才能真正完成的。

KLS问：你认为Linux的未来怎么样？离开你Linux还会不会继续发展？

LT答：Linux的发展已经完全超越了我。十年前，Linux的发展少了我可能就不行了，但现在有许许多多的公司和技术人员已经了解了Linux技术，我正在试图结束自己作为Linux技术中心人物的地位，但很多人知道我，认识我，他们信任我，我是技术中立角色。我喜欢做Linux，我喜欢这种技术挑战，我喜欢和广大Linux技术爱好者交流，我想做这个工作。

KLS问：听起来，你还是将继续从事你的工作，继续作为Linux整个开发过程的中心关注点继续存在吗？

LT答：是的。同时我也会尽可能地让其他人来分担我的一部分核心Linux开发工作，但我仍然将是整个Linux开发项目的中心点，如果整个项目没有一个中心点来控制和协调，那将是非常不可行的，整个Linux项目就会瓦解。

KLS问：你将来还要实现一些什么目标吗？

LT答：没有。我不喜欢对未来作很多规划。15年前我造出了Linux操作系统，当时我完全没有想到Linux现在会这么有商业价值。我是那种“泥萝卜，擦一段，吃一段的人”，我很高兴Linux能受到人们的喜爱，确实也有很多人在使用。但我现在没有，也永远不会有什么巨大的理想，或者是目标需要去达到或实现。

复习题

1. 什么是进程？进程与程序有什么联系和区别？
2. 请画出进程的基本状态转换图。
3. 常见的进程调度策略有哪些？分别有什么优点和缺点？怎样改进？
4. 为什么进程切换会产生系统开销？
5. 操作系统用什么方法实现进程对资源的互斥访问？
6. 操作系统用什么方法实现进程间的同步？
7. 操作系统为进程间的通信提供了哪些手段？它们各有何优缺点？适合什么样的场合？
8. 线程是什么？与进程有什么区别和联系？为什么要使用线程？
9. 什么是固定分区法？它有什么优点？缺点是什么？
10. 什么是动态分区法？它有什么优点？又存在什么问题？
11. 简述页式内存管理。
12. 页表的作用是什么？页表放在什么地方？一个系统中有多少页表？
13. 分页机制下，逻辑地址是怎样转变为物理地址的？
14. 什么是虚拟内存技术？
15. 什么叫缺页异常？为什么要减少缺页异常发生的次数？
16. 实现虚拟内存技术，装入页的时机有哪两个？各自的优缺点是什么？怎样能够充分发挥这两种方式的长处？
17. 为什么要使用高速缓存存放页表项？什么叫关联存储器？
18. 有哪些常见的页淘汰算法？最常用的算法是什么？
19. 什么是文件？什么是文件系统？
20. 常见的文件物理组织结构有哪些？它们适合什么样的应用？
21. 为什么访问文件要提供文件的路径？

22. 请简述类 UNIX 操作系统在外存上定位文件的过程。
23. “当前目录”的作用是什么？
24. 操作系统通常使用什么手段保护文件？
25. 简述虚拟文件系统怎样支持异构文件系统。
26. 在 Linux 中，一个文件系统在使用前为什么必须先注册？注册过程的主要任务是什么？
27. 在 Linux 中，一个文件系统在使用前为什么必须先安装？安装过程中，系统都做了哪些工作？
28. 设备管理子系统采用什么方式来屏蔽设备的多样性？
29. 举例说明什么叫虚拟设备技术？
30. 采用“中断”方式同外设通信的驱动程序的工作流程框架是什么样的？
31. 为什么要使用缓冲？

讨论

1. 请查阅相关参考书了解什么是位图，与使用空闲链表管理页框相比，使用位图管理空闲页框有何优势和不足？

2. Windows 支持线程级的 CPU 分配，Linux 只支持进程，二者有何区别？这两种不同的任务管理方式，对于系统性能有何影响？查阅相关资料，解释实际情况是怎样的。

实验

在 Linux 操作系统中挂载 NTFS 文件系统。实验参考步骤：

(1) 多数 Linux 内核版本默认并不带有对 NTFS 的支持，所以，需要首先找到并下载对 NTFS 支持的内核模块。

(2) 安装该模块，安装过程即文件系统的注册过程。

(3) 安装成功后，确定磁盘上的 NTFS 分区。

(4) 用 mount 命令将该 NTFS 分区安装到合适的目录上。

(5) 在 Linux 下访问 NTFS 分区上的文件。

第 5 章　系统初始化及 Shell 编程

本章来了解另外一些与计算机运行相关的基础知识。其中一部分是系统的初始化过程。通过这部分内容我们将明白从按下电源开关直到能够通过操作系统使用计算机的这段时间内计算机在忙什么。在此之后本章将简介与启动过程相关的 CMOS 设置。最后还将介绍怎样使用 Shell 脚本来控制计算机。这些都是在操作系统平台这个层面理解和控制计算机的基本常识和技能。不过，这一章并不介绍怎样使用操作系统提供的命令或者图形界面来操作计算机。这些内容可以从现代操作系统中通常都提供的在线帮助中轻松得到，就留作课外作业自己去学习和练习吧。

5.1　计算机系统初始化

5.1.1　启动过程要做什么

通常使用计算机第一步要做的事情就是按下电源开关，然后等待，直到计算机出现用户登录界面。那么，在等待的这段时间里，计算机在做些什么呢？经过前面的知识积累，我们已经有可能理解计算机在这段时间里所做的工作了。因此，现在我们就来看看这些工作到底是什么。

我们已经知道以下一些事实。计算机按照指令行事；执行的指令要放在内存中，因为这样才能保证计算机的处理速度；内存是“易失性的”，关机之后里面就什么都没有了，所以，指令或者说程序要存储在能够长久保存信息的外存上；程序在运行之前必须被装入到内存里；这种装入一般是在操作系统的控制下完成的。从这些事实，可以推出一个结论：在运行任何应用程序之前，必须先运行操作系统。也就是说，计算机开机后，首先应该运行操作系统。

操作系统也是保存在硬盘上的一组软件包。因此，要运行操作系统，也需要先把它的代码（至少是一部分重要代码，如内核）装进内存。所以，计算机启动时的第一件事情就是把操作系统，至少是操作系统内核，从硬盘读入内存。

现在的操作系统一般都是一个比较庞大的软件包，里面包含很多以文件形式存在的程序。如果要从硬盘上读取以文件形式存在的操作系统软件包，就必须有文件系统以及设备子系统的支持。因为文件的存储格式以及访问方式都是由文件系统定义的。只有通过文件系统提供的访问接口才能够找到并访问文件。而对外存进行访问则必须有设备子系统的支持。只有通过设备子系统才能找到磁盘的驱动程序把信息从磁盘读出来。可是这两个子系统又都是操作系统的一部分，在操作系统装入之前，它们也都不在内存。这样看来，装入操作系统的任务是不能依靠这两个子系统的。

你可能会想，把文件系统和设备子系统事先放入内存，而且保证开机时就在内存中是不是就可以了呢？这个想法实际上是部分可行的。可行部分是在计算机中可以增加一个非易失性内存事先存放一些代码。这样，这些代码一开机时就能够在内存中。实际上，现在计算

机主板上都有这样的一个内存。以前这个内存是个只读存储器，现在通常是可以擦写的闪存(不过，我们仍然把它称为 ROM，因为一般并不更改这个存储器的内容)。但是，把操作系统的文件系统和设备子系统存放在主板上的这种非易失性内存中是行不通的。因为文件系统的种类太多了，而且又在不断地变化。而不同操作系统又可能是以不同的文件系统格式存储在硬盘上。如果在主板非易失性内存中预装文件系统，那么应该装入哪一种呢？还是哪几种？而且一般文件系统的实现代码占有空间也都很大，主板的非易失性内存也实在是不够装入多少个文件系统。

既然不能依靠文件系统和设备子系统，那么应该怎样做呢？我们从分析外存向内存装入文件的过程入手来看看是否可以绕开这两个子系统把操作系统读入内存。读入文件的过程可以分解为三部分，即定位文件在外存上的位置、从外存读信息、按照文件格式解释读取的信息。通常这三部分工作是由文件系统和设备子系统协同来完成的。现在我们就来看看在没有这两个子系统帮助的情况下，能不能完成上述工作。实际上，CPU 都支持通过物理地址从外存读取 0/1 串的 I/O 指令。所以，从外存读入信息的任务可以完成。但是，怎么获得文件在外存上的物理地址呢？有文件系统的时候，我们是通过系统的文件索引或者其他记录信息得知文件的位置。但是，现在没有文件系统。最简单的解决办法就是把要读取的信息放在外存上的固定位置。这样就不用再为找到它的位置伤脑筋了。好！还有最后一个任务，就是解释所读取信息的意义。通常信息的格式和含义是由文件系统定义的，可是现在我们还不知道使用什么文件系统。但是，如果我们把内容的格式固定下来，让它与任何文件系统无关。这样我们不就可以不通过任何文件系统而能够解释这些代码和数据了吗？到现在我们已经找到了绕过文件系统和设备管理子系统从外存读取信息的策略了。不过这种策略很原始，效率不高，因此，我们只能用它来读入少量的代码。如果计算机上运行的全部代码都用这个方法读入那就太慢、太不方便了，所以才会需要操作系统。

实际的计算机系统正是使用上述方法解决了外存上第一批代码和数据读入内存的问题。以磁盘为例，根据上述策略，操作系统内核在磁盘上的位置信息和读取代码被放到磁盘的某个固定区域，通常是第一个扇区；第一个扇区中信息的格式是固定的，与任何文件系统无关；计算机厂家在主板非易失性内存中预装通过磁盘调用读取磁盘第一个扇区的代码。这样就可以一步一步地把操作系统(包括文件系统)装入并运行起来了。

除了读取磁盘的调用外，计算机启动时需要的另外一些代码，如检测硬盘的代码，也被放在主板非易失性内存中。这些代码合在一起被称为基本输入输出系统。也就是我们前面提到过的 BIOS。因此，通常也把主板上的这个非易失性内存称为 BIOS ROM。

因为计算机启动时不能通过文件系统知道操作系统内核在磁盘上的位置以及它的大小，所以操作系统内核的位置信息被放在磁盘的第一个扇区中。不同操作系统的地理位置信息表示形式可能会不同，所以第一扇区中还装有读取这些信息的方法。这样计算机就可以绕过文件系统找到操作系统内核了。因此磁盘第一扇区也被称为引导(启动)扇区。BIOS ROM 中预装有负责读取磁盘引导扇区的引导代码。BIOS ROM 中的引导代码与具体操作系统无关，一般由计算机厂商提供。而磁盘引导扇区中的内容是与操作系统相关的，它实际上是具体操作系统的引导代码，所以就由具体的操作系统来填写了。通常引导扇区

的内容是在安装操作系统时由操作系统安装程序写入的。因为一个扇区通常只有512B,所以引导扇区中的代码不可能很大。它的功能一般是继续读取其他扇区上的内容。然后一步一步完成装入并运行操作系统的任务。

按照前面的办法,一个磁盘只有一个引导扇区,因此,也就只能安装一个操作系统。不过现在的情况并不是这样。人们通过逻辑磁盘来实现安装多个操作系统的目标。一个逻辑磁盘就是一个硬盘分区。一个物理磁盘可以被划分成若干个逻辑磁盘。逻辑磁盘的第一个扇区仍然是引导扇区,可以放一个操作系统的引导代码。这样,每个逻辑磁盘都可以安装一个操作系统。但是,逻辑磁盘的引导扇区不再是物理磁盘上的第一个扇区。BIOS还把整个硬盘的第一个扇区当作硬盘的引导扇区,加电后还是从这里引导系统。但是,整个硬盘的第一个扇区不属于任何逻辑磁盘。这个扇区被称为"主引导扇区",即MBR(Master Boot Record),安装操作系统时经常能够见到这个词。通常MBR中有关于硬盘的逻辑磁盘划分状况信息的分区表。MBR中的引导代码不直接引导操作系统,而是从分区表中找出被标记活跃的逻辑磁盘。然后读取该活跃逻辑磁盘的引导扇区来引导操作系统。在这种情况下,磁盘上可以安装多个操作系统,但是只有激活(活跃)磁盘的操作系统才能启动。这种方式对于使用多个操作系统的用户说,还是不够方便。为此,人们又编写了一种专门负责引导和装载操作系统的引导装载(加载)程序。引导装载程序与具体的操作系统无关,它们可以按照安装操作系统时配置的信息来找到具体操作系统的引导代码,可以根据用户的选择去读取相应的引导扇区。这样,磁盘上装载的多个操作系统就可以根据用户的选择被启动了。可以看出,引导装载程序的地位和作用与主板BIOS ROM中的引导代码类似,不过功能更强大而已。常见的引导装载程序有LILO、GRUB。因为引导扇区只有512B,所以引导装载程序必须很小,通常都用汇编语言编写。也有大一些的引导装载程序,如GRUB。它把自己分成几部分。第一部分放在MBR中,其他部分放在其他扇区,通过第一部分的代码来装入。

现在问题好像解决了。但是,别忘了计算机是按照指令行事的。光有启动代码还不行,计算机还必须在一启动时就能够找到BIOS ROM中的代码才行。解决办法同样很简单。计算机被设计成一启动就从一个固定的内存地址读取指令。这个固定的内存地址就是主板BIOS ROM中第一条指令所在单元的地址。这样我们就解决了指令起点的问题。这全仰仗计算机的一个天生的本领:它一出生就知道一个固定的地方有第一条指令。有了第一条指令作起点,后面的事情就好办了。

最后,我们再来啰唆一下为什么不把操作系统直接放在主板上的非易失性内存中。这是因为操作系统的功能很复杂,因而代码量一般都比较大。而主板非易失性内存的空间都比较小。尤其计算机刚刚出现时,还没有现在的闪存技术,那时主板非易失性内存的空间非常小,只能放十几条指令。即使现在主板上的闪存可以做得很大,但是,操作系统仍然不适合放在主板上。因为操作系统一直在不断地发展完善,所以我们可能会升级操作系统或者更换操作系统。如果把操作系统放在主板的非易失性内存中,由计算机厂家来负责预装操作系统,那就太不方便了。所以我们都把操作系统放在外存上,通常是硬盘上,有时也会在网络上。

小提示

计算机启动时，会从一个固定地址读取第一条指令，该地址指向 BIOS ROM 中的第一条指令；
BIOS ROM 中的指令再读取磁盘的第一个扇区；
磁盘第一扇区中的代码再去读磁盘上其他部分更多的代码；
这样一步一步把操作系统装入并启动起来。

5.1.2　计算机启动过程

了解计算机启动过程要做的事情之后，我们来看看具体的启动过程。图 5-1 是启动过程的一个概要流程图。这一小节将详细介绍最上面一个虚线框中的内容，即 BIOS ROM 中的引导代码所做的事情，包括开机自检、初始化，以及读入引导扇区内容。

我们从按下电源开关那个瞬间开始。此时计算机的电源开始向主板和其他设备供电。当 CPU 检测到电压稳定后，便开始读取第一条指令。对于 Intel 的 i386 结构 CPU，第一条指令的地址是 FFFF0H（最后一个 H 表示是十六进制数）。这个地址对应的是 BIOS ROM 中的一个单元。

BIOS 中包括计算机系统的基本输入输出程序、系统信息设置程序、开机上电自检程序和系统启动自举程序。在计算机出厂时，BIOS 程序就被固化在主板上的 BIOS ROM 中。在早期，主板上的 ROM 非常小，因此里面并没有这么多程序，只有引导系统必需的一些指令。后来技术越来越好，ROM 越做越大，设计者就往 ROM 中加了开机自检程序、系统信息设置程序。最后干脆把一些常规外设的驱动程序也放在了里面。它的名字也就变成基本输入输出系统，即 BIOS 了。

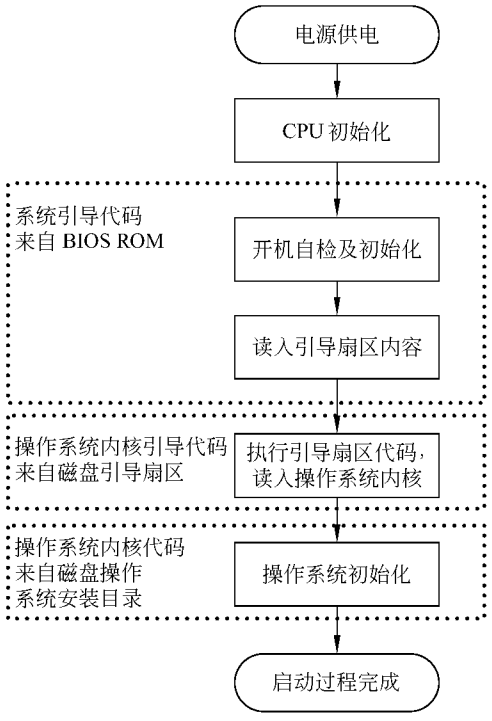


图 5-1　计算机系统启动过程

下面继续介绍启动过程。BIOS ROM 中的 FFFF0H 单元中通常是一条跳转指令，它把 CPU 指引到 BIOS 中引导代码真正开始的地方。

首先执行的是系统自检程序，也就是常说的 POST(Power On Self Test，加电自检)。POST 首先检查 BIOS 芯片，然后测试 CMOS(一小块特殊的存储器，存放系统基本的配置信息)。如果一切正常，POST 接下来就会初始化 CPU，检查系统中的关键设备，如内存、显示卡等是否正常。因为在 POST 进行检查时，显示卡还没有初始化，所以显示器还不能用。如果检测到错误，计算机只好用喇叭来告诉我们。它通过声音的长短和次数来告诉我们是什么出了问题。这有点儿像电报。不同厂家生产的不同类型 BIOS 定义的报警铃声不一样。例如，常见的 BIOS 有 AWARD、AMI 和 PHOENIX 三种，AWARD BIOS 定义一个短声表明一切正常，而 AMI BIOS 定义一个短声则为内存刷新失败。所以在仔细听铃声之前还要知道计算机使用的是哪种 BIOS[46]。

接下来 BIOS 将使用显卡上自带的一些初始化代码来初始化显卡。这时计算机会在屏幕上显示出一些初始化信息，如介绍显卡生产厂商、图形芯片类型等。不过这个画面几乎是一闪而过。初始化显卡之后，BIOS 就可以在显示器上显示自己的启动画面了。一些系统此时会在屏幕上显示 BIOS 的类型、序列号和版本号等内容。

接着 BIOS 将检测和显示 CPU 的类型和工作频率。然后开始测试内存，并同时在屏幕上显示内存测试的进度。内存测试通过之后，BIOS 将开始检测系统中安装的一些标准硬件设备，包括硬盘、光驱、串口、并口、软驱等设备。一些 BIOS 在这一过程中还要自动检测和设置内存以及硬盘的一些参数。标准设备检测完毕后，BIOS 内部支持即插即用的代码将开始检测和配置系统中安装的即插即用设备。每找到一个设备，BIOS 都会为其分配必要的资源，其中最重要的就是中断号。

接下来 BIOS 将更新扩展系统配置数据(Extended System Configuration Data，ESCD)。这些数据被存放在 CMOS 之中。通常 ESCD 数据只在系统硬件配置发生改变后才会更新。

如果我们在操作系统中选择“重新启动”来热启动计算机，那么上述的 POST 过程将被跳过去。

当确信所有部件都正常之后，BIOS 开始查找操作系统，并将其装入。BIOS 通过 CMOS 存储器中存储的“启动顺序”得知在哪些设备上可以找到操作系统。这个启动顺序由用户通过 BIOS 的系统信息设置程序来设置(后面很快就将介绍 CMOS 中参数的设置)。BIOS 会按照启动顺序依次试着在这些设备上寻找操作系统直到成功为止。然后 BIOS 会装入并启动相应设备上的操作系统。

假定 CMOS 启动顺序中设置的第一启动设备是硬盘。那么，BIOS 将读取硬盘主引导扇区 MBR 中的内容。当 MBR 中的引导程序被装入内存后，BIOS 就会将系统的控制权转交给它，也就是 CPU 开始执行该引导程序。至此，BIOS 的任务胜利完成[43]。

MBR 中的引导程序会把活跃分区(或根据用户的选择，如果 MBR 中放置的是独立的引导装载程序)的引导扇区内容从硬盘读到内存，并把系统控制权交给它。引导扇区上的操作系统内核装入程序负责装入具体操作系统的内核。内核装入成功后，系统的控制权将交给操作系统内核。内核代码再进行操作系统相应的初始化工作，直到把所有准备工作都做完，在屏幕上显示出用户登录界面，等待用户登录。到这里，启动过程就结束了。

小提示

计算机启动过程如下：
计算机启动时，首先读取 BIOS ROM 中的程序来执行。
BIOS 首先进行系统自检，成功后再根据用户在 CMOS 中的配置从相应的设备上装入操作系统。
对于硬盘作为启动设备，BIOS 读取硬盘的 MBR，然后把控制权交给 MBR 中的引导代码。
MBR 中的引导代码读取相应逻辑磁盘的引导扇区。
逻辑磁盘引导扇区中的操作系统内核装入程序接着装入具体操作系统的内核。
操作系统内核代码最后完成操作系统的初始化工作。

5.1.3　操作系统的启动过程

这一小节我们把引导扇区读入内存之后直到操作系统启动完毕这个过程中的工作再具体一下。我们以 Linux 操作系统的启动过程为例。

这个过程如图 5-2 所示。在计算机启动时，BIOS 或者引导装载程序会把 Linux 操作系统引导扇区中的内核引导代码读入内存中的一个特定位置(内核引导代码用汇编语言编写，在 Linux 操作系统的/usr/src/linux/arch/i386/boot/目录下可以找到它的源码文件 bootsect. S)。接着计算机开始执行这段内核引导代码。

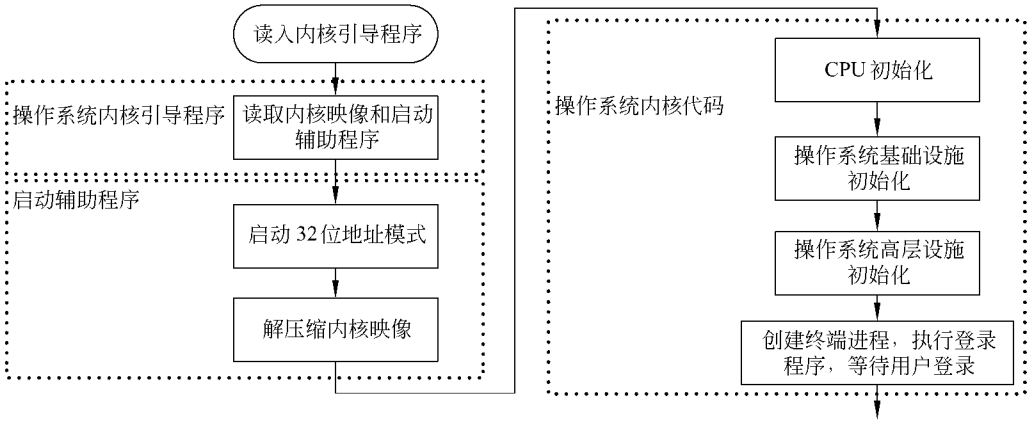


图 5-2　Linux 操作系统启动过程

这段代码先把自己在内存中的位置挪动一下，因为它觉得自己目前的位置有点儿碍事，影响后面调入的代码。然后它让 CPU 到新位置继续读取自己的代码并执行。内核引导代码通过 BIOS 提供的磁盘调用读取硬盘上的启动辅助程序和内核映像。

计算机在刚启动时，系统处于 16 位地址模式。也就是程序中使用 16 位的地址，因为支持 32 位地址的代码还没有运行。16 位地址能够访问的内存空间有多大？2^{16}B＝64KB。也就是说系统启动时只能使用最低的 64KB 内存空间。Intel 的 CPU 使用 16 位的段地址寄存器和 16 位段内偏移指针寄存器(也叫指令指针寄存器)协同可以形成 20 位的物理地址(它把段地址寄存器中的地址后面加上 4 个 0，然后同段内偏移指针寄存器中的地址相加，这样就可以得出一个 20 位的地址)。20 位地址可以访问的空间是 2^{20}B＝1MB。这样在 16 位地址模式下，Intel 系统中，启动时可以访问的地址空间是从 0～1MB－1B 这 1MB 大

小的空间。在这段空间中，从 640KB～1MB 的地址空间被分配给了显示卡以及系统 BIOS。还记得我们提到计算机执行的第一条指令的地址 FFFF0H 吗？它就是在这个区间内。所以，引导操作系统时只能使用称为系统基本内存的最低 640KB 内存。这样可以用来存放内核映像的空间非常小。因此，通常内核映像都是经过压缩的。

在成功读取启动辅助程序以及内核映像之后，系统就转而开始执行启动辅助程序。启动辅助程序通过 BIOS 中断获得系统内存的大小，并设置键盘参数、检查显卡类型和软硬盘配置，然后把这些参数保存在内存中的一个特定的安全位置（就是原来放置内核引导程序的地方，现在内核引导程序已经完成任务了，所以它的空间就被收回再利用了）。

接下来辅助程序开始做一些和 32 位地址模式有关的准备工作，如设置相关寄存器，然后系统转入 32 位地址模式。接着系统转去执行启动辅助程序中一个负责解压缩内核映像的汇编程序。现在系统已经能够使用大于 1MB 的内存空间了。因此，就有空间来放解压后的内核了。辅助程序把解压后的内核放到 1MB 地址之上被称为高端的内存中（因为在以前 1MB 的内存已经非常大了，所以才会有这样的称呼。尽管现在的内存已经可以达到 1GB 了，但是这个称呼还是被沿用下来）。之后系统就跳转到内核代码的开始处继续执行。从这时起操作系统的初始化过程就开始了[44]。

在内核开始执行之前的程序都是利用 BIOS 提供的调用来完成读磁盘等操作。在内核开始执行之后，系统就不再使用 BIOS 提供的工具了。

内核的初始化是个相当复杂的过程，可以分成三个阶段。第一个阶段主要是 CPU 本身的初始化；第二阶段主要是系统中一些基础设施的初始化，如内存管理和进程管理的建立和初始化；最后是高层设施的初始化，如根设备的安装和外部设备的初始化等。

在第一阶段，内核程序首先对段寄存器进行相应的设置（Intel 的 CPU 设计成支持段式内存管理，但实际 Linux 并未使用，不过仍然要设置段寄存器）。然后重新设置中断描述符表。设置完这些寄存器后，系统开始设置页表并启动分页机制。在此之前 CPU 都是以物理地址暂时来读取指令的。系统在这里先设置了两个 4KB 的临时页表。Linux 的内存管理使用多级页表。在 i386 计算机上除了页表外，至少还要有一层页面目录。系统接下来要做的事情就是初始化页面目录，并设置分页机制中涉及的寄存器，如设置 cr0 寄存器中相应位表明启动分页机制。至此，分页机制已经启动。

接下来 CPU 把从启动辅助程序作为参数传过来的那些从 BIOS 收集来的参数复制到内存中一个特定地方，形成参数块，以备后面的初始化过程使用。

到现在为止，系统内核已经被放到它应该所处的最终位置，与硬件平台有关的启动过程基本结束。系统内核进入与硬件平台无关的内核数据结构初始化过程。

前面的程序都是汇编代码写成的。从这时系统才开始执行第一个用 C 语言写的程序。从现在开始内核进入了一个较高层次的初始化过程。实际上可以说内核真正的初始化才刚刚开始。作为第二阶段的初始化过程，系统要为内核中各个部件或模块的运行准备环境，设置系统的总体数据结构。例如，为页管理机制准备必要的数据结构，包括扩充前面建立的页面目录和页表，建立完整的从系统空间到物理存储空间的线性映射，为每个内存管理区创建空闲页框队列等。

接下来的第三阶段要进行进程调度机制的初始化、对系统主控终端的初始化，以及根设备的初始化、应用层的初始化等工作。这阶段为应用程序的运行，首先是系统管理，做好了

准备。操作系统内核将根文件系统安装到根(“/”)目录,并初始化进程表的第 0 项,使根(“/”)成为 0 号程的当前目录,为 0 号进程的运行形成环境。然后,系统开始以 0 号进程的身份运行。0 号进程在核心态创建一个新的进程,即 1 号进程。1 号进程继续运行在核心态做一些系统初始化工作,并创建一些管理进程的守候服务进程。最后 1 号进程通过系统调用演变成为第一个用户态进程。1 号进程负责初始化所有的新进程。因此 1 号进程又称为初始化进程,它的名字就是 init。

init 进程进入用户状态后,除了产生其他进程外,一个主要任务是让用户通过终端登录,因此它创建若干个 getty 进程。getty 进程会去执行登录程序 login,让用户登录。用户输入用户名和密码登录成功后,系统就根据配置去执行相应的 Shell 程序[2]。此时用户就可以开始工作了。操作系统的启动过程宣告结束。

小提示

Linux 操作系统的引导代码被装入内存执行后,它会读取硬盘上的启动辅助程序和内核映像,并执行启动辅助程序。

启动辅助程序启动 32 位地址模式,解压缩内核映像,并将系统控制权转交给操作系统内核。

操作系统内核接下来开始操作系统的初始化过程。

内核的初始化过程可以分成三个阶段,分别是与具体 CPU 相关的初始化、系统基础设施的初始化及高层设施的初始化。

5.2　CMOS 设置

现在我们来讲一讲怎样设置 CMOS 中的参数。前面介绍过,CMOS 是主板上的一个特殊存储器,里面存放着计算机系统硬件的一些配置信息。这些信息一般在系统初始化时需要使用。对这些参数的设置要通过 BIOS 中的系统信息设置程序来完成。所以这个设置过程有人叫“BIOS 设置”,也有人叫“CMOS 设置”。其实指的都是一回事,只不过一个讲的是所设置参数影响的对象,另一个则是存储该对象的介质。

通常我们并不需要对 CMOS 中的参数进行设置,因为系统会自动检测到一部分参数的值,而另外一些参数系统会有默认的设置。那么什么时候需要运行 BIOS 中的设置程序,对这些参数进行设置呢? 一般有以下几种情况[45]:

1. 新购计算机

新买的计算机,尤其是组装的计算机,有些外设的属性计算机不能自动识别出来,如硬盘的容量。这就需要用户自己通过 BIOS 去设置。

2. 新增设备

同样道理,如果系统新添了一个外设,由于系统不一定能认识新增的设备,所以也要通过 CMOS 设置来告诉它。此外,一旦新增设备与原有设备之间发生了中断号(IRQ)或者 DMA 地址冲突,也往往需要通过 BIOS 设置来解决。

3. CMOS 数据意外丢失

在系统后备电池失效、病毒破坏了 CMOS 数据或者意外清除了 CMOS 中的参数等情况下,只能重新进入 BIOS 设置程序完成新的 CMOS 参数设置。

4. 系统优化

对于内存读写等待时间、硬盘数据传输模式、内/外 Cache 的使用、节能保护、电能/源管理、开机启动顺序等参数，BIOS 中预定的设置对系统而言并不一定是最优的，所以需要有经验的人来设置。

那么怎样进入 BIOS 设置程序呢？在我们按下计算机电源开关时，BIOS 中的自检程序就开始运行了。当屏幕下方出现 Press F2 to enter SETUP 时，按住 F2 键就可以进入 BIOS 设置程序了。进入不同 BIOS 需要按的键不一样。例如，进入 PHOENIX BIOS 的键是 F2，而进入 AWARD BIOS 的键通常是 Del 键。我们不需要记住这些键，但是需要仔细看屏幕下方的提示(很遗憾，提示是英文的)。而且留给我们按这个键的时间越来越短，因为计算机越来越快，BIOS 很快就干完活，启动操作系统了。这时就不能再进入 BIOS 设置了。所以，如果我们不是很小心，那么可能需要启动几次，才能成功进入 BIOS 设置。

下面以 PHOENIX BIOS 为例，简要介绍 BIOS 的设置，大致看看 CMOS 中有哪些参数可以设置，它们都有什么作用。不同版本的 BIOS，界面以及可以设置的项都不一样。但是，多数 BIOS 都支持常见的一些参数。

小提示

CMOS 设置也被称为 BIOS 设置。

CMOS 是主板上的一个特殊存储器，里面存放着计算机系统硬件的一些配置信息。CMOS 中的信息通过 BIOS 中的设置程序来设置。

在新购计算机、新增设备、CMOS 数据意外丢失，以及需要优化系统时需要进行 CMOS 设置。

在成功进入 BIOS 设置程序后，会看到如图 5-3 所示的界面。在屏幕上方那条主菜单中可以看到 6 个选项。这 6 项分别如下：

① Main：主要参数设置。主要进行系统日期、软盘[①]、硬盘、系统内存的设置。

② Advanced：高级设置。包括芯片组、I/O 设置、集成外设端口、即插即用功能设置等。

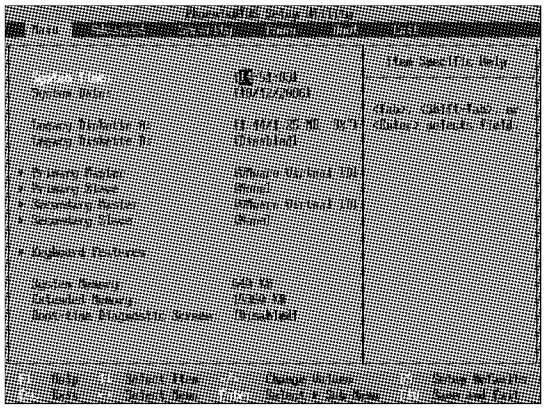


图 5-3 BIOS 设置程序主界面

① 由于闪存的出现，现在的计算机一般都没有软驱了。

③ Security：安全选项设置。

④ Power：电源/电能管理功能设置。

⑤ Boot：启动选项设置。

⑥ Exit：选择退出 BIOS 程序的方式并退出。

现在只能使用键盘移动光标进行这些参数的设置。BIOS 通常不支持鼠标。在屏幕的最下方是进行各种设置应该使用的按键。我们可以使用上、下、左、右箭头移动光标来选择各项。根据提示，可以知道如果想改变某一个选定项的值，那么要按“－”或者“＋”键。如果选定的项有子菜单，那么按 Enter 键可以进入选定项的子菜单。按 F9 键可以恢复默认的设置值。按 F10 键则可以快速保存并退出。F1 键可以提供帮助。无论在任何窗口下，要想退出当前的这个窗口，只要按键盘左上角的 Esc 键就行了。当光标移动到某个选项上时，屏幕右侧的窗口中会给出这一项的说明。不过是英文的，所以需要我们仔细、耐心地看才行。屏幕上蓝色的项是不能修改的，是 BIOS 检测出来的参数，只是显示给我们检查校对的。黑色的数值是可以根据情况设置的。

下面对一些主要项进行简要说明。先从 main 菜单开始，如图 5-3 所示。这个界面上主要有以下这些项。

① System Time：系统时间。System Date：系统日期。Diskette A 和 Diskette B 用于设置软驱类型。

② Primary Master：主 IDE 接口上的主设备。Primary Slave：主 IDE 接口上的从设备。

③ Secondary Master：第二个 IDE 接口上的主设备。Secondary Slave：第二个 IDE 接口上的从设备。这几项内容是系统自动检测到的。还记得可以接在 IDE 接口上的设备有哪些吗？（主要是硬盘和光驱）

④ Keyboard Features：设置键盘的属性，按 Enter 键进入子菜单，可以设置是否锁住键盘的数字小键盘、设置键盘的重复率以及重复延迟值。现在很少在 CMOS 这里设置这些参数了。因为操作系统也都提供了设置键盘、鼠标这些参数的工具。通过操作系统设置比在 BIOS 中设置感觉要方便一些。

System Memory：这项显示系统的基本内存。一般都是 640KB。Extended Memory：显示扩展内存的大小。这两项内存的信息都是系统自动检测到的。如果与计算机上的实际情况不符，那就是计算机的内存出问题了。或者是内存坏了，或者是和系统不兼容，它识别不出来。

Boot-time Diagnostic Screen：设置在启动时是否显示 BIOS 检测过程的一些显示信息。

接下来我们移动光标到主菜单的第二项，高级选项 Advanced。这里可以设置的项见图 5-4。其中，Multiprocessor Specification 表示多处理器规范的版本，在系统有多个处理器时使用。Installed O/S 项用来告诉 BIOS 系统上安装的、最常用的操作系统。Reset Configuration Data 项指明是否清空/重新设置扩展系统配置数据（Extended System Configuration Data，ESCD）区。Large Disk Access Mode 大盘访问模式项的值有 dos 和 other 两种。如果安装 UNIX、Novell 等操作系统则需要选择 other。I/O Device Configuration 项可以对外设的属性进行配置，其实主要是对串口和并口的特性进行设置。

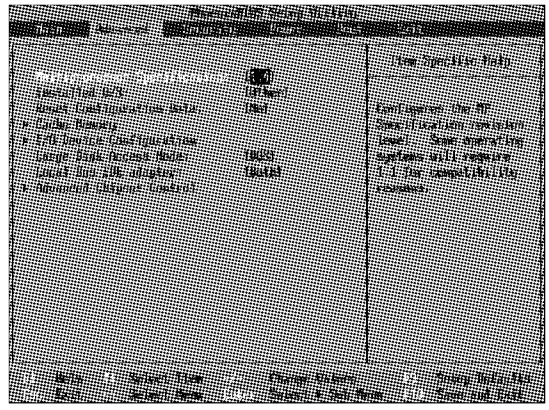


图 5-4　Advance 界面

Cache Memory 项决定内存中一些特殊块的缓存属性。进入这项的子菜单，我们会看到如图 5-5 所示的内容。Memory Cache 项指明是否缓存内存。后面的项则对不同的内存区块的缓存属性进行设置。我们可以根据情况决定是否对这些特殊的内存区块进行缓存。如果缓存，那么采取什么样的形式。例如，选择 Disabled，表示这一块内存不被缓存；选择 Write through 表示如果缓存的区域被写入，那么就马上写回到主存；选择 Write protect 表示对这个缓冲区的写会被忽略，即这段内存不允许写入；选择 Write back 表示写入的内容先放在缓存中，在需要的时候再写回到主存。

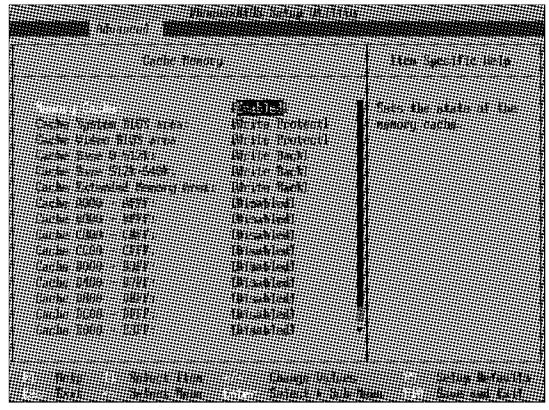


图 5-5　Cache Memory 界面

主菜单的第三项，Security 安全选项，可以对密码进行设置，如图 5-6 所示。密码包括系统启动时使用的密码以及进行 CMOS 设置所使用的密码。

主菜单的第四项，Power 电源/电能管理项，主要对系统的能源进行管理。这里可以设置节能选项，包括系统在空闲多久后进入待机状态，以及在待机多久后自动挂起，如图 5-7 所示。

主菜单的倒数第二项，Boot 启动选项，用来设置启动顺序，如图 5-8 所示。这在前面已经提到过。BIOS 在装入操作系统时，按照这个顺序依次查找设备，直到找到一个存在操作系统的设备。

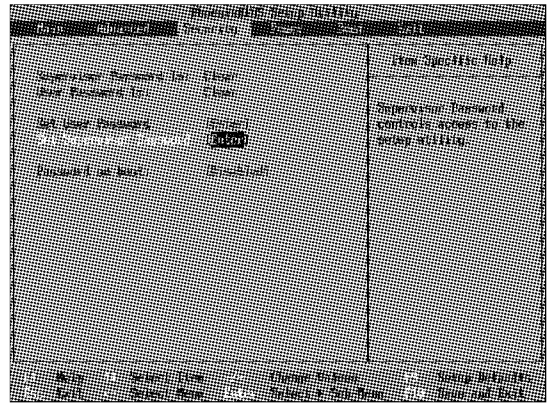


图 5-6　Security 界面

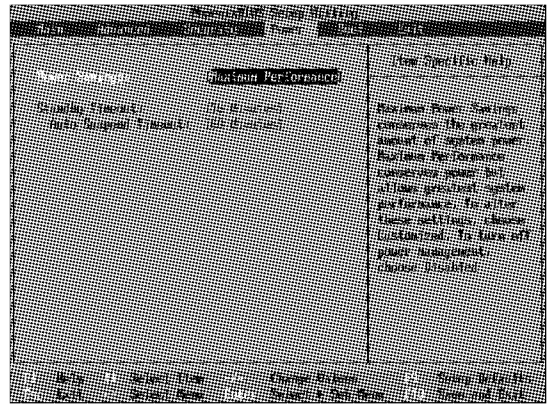


图 5-7　Power 界面

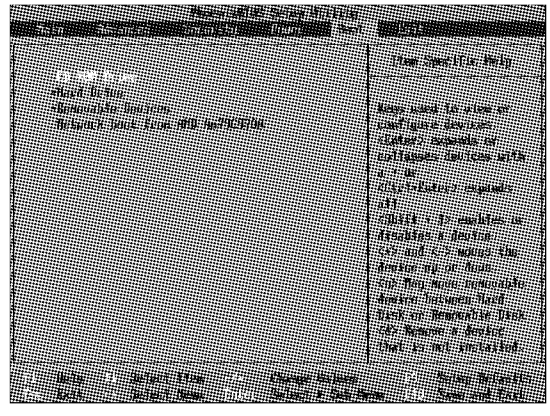


图 5-8　Boot 界面

主菜单的最后一项 Exit 可以让我们选择退出 BIOS 设置程序的方式，如图 5-9 所示。这些选项的含义从名字上就可以一目了然了。在退出 BIOS 设置时，根据情况选择合适的一项即可。例如，在更改了 CMOS 设置准备退出时，要选择保存更改并退出，即 Exit Saving Changes。

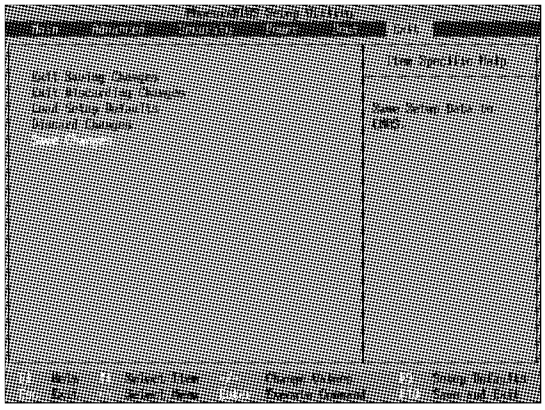


图 5-9 Exit 界面

5.3 操作系统安装

本节来了解怎样在计算机上安装操作系统。从前面的介绍我们能够想到操作系统安装程序要做的事情就是把操作系统的文件复制到硬盘的适当目录下，并填写引导扇区的内容。我们仍然以 Linux 操作系统为例来介绍操作系统的安装过程。

前面说过，在 Linus 主持下实现的是 Linux 操作系统的内核。它实现了操作系统的主要功能，如进程管理、内存管理、文件管理等。起初，Linux 是以内核源代码的形式交付给用户的。用户需要在其他操作系统下编译 Linux 的源代码，然后才能使用。这对于普通用户来说是件非常有难度的事情。后来，一些组织和厂家将 Linux 系统内核与一些外围实用软件以及文档包装起来，并且配套提供系统安装界面，以及一些系统配置和管理工具。这样用户使用起来就方便多了。这些不同的软件组合就构成了各种发行版本的 Linux。发行版使用户安装和使用 Linux 变得相对容易，对 Linux 的推广做出了很大的贡献。

这些 Linux 发行版本，实际上就是以 Linux 为核心的操作系统软件包。一个 Linux 发行版套件包含大量的软件，如软件开发工具、数据库、Web 服务器、X Window、桌面环境（如 GNOME、KDE）、办公套件（如 OpenOffice. org）等。根据 GPL 准则，这些发行版本虽然有各自的贡献，但他们只有整体集成版权。因为 Linux 的各个发行版本都使用同一个 Linux 内核，所以它们在内核层不存在兼容性问题。每个发行版本的不同只是在最外层表现上。

小提示

Linux 的发行版本是以 Linux 为核心的操作系统软件包。

一个 Linux 发行版套件包含大量的软件，如软件开发工具、数据库、Web 服务器、X Window、桌面环境、办公套件等，并且配套提供系统安装界面，以及系统配置和管理工具。

这里我们用 Linux 的发行版本之一——Fedora 的版本 9 来介绍 Linux 系统的安装过程。Fedora 9 是在 Red Hat 公司和自由开放源码软件社团 FOSS（free and open source

software community)联合承担的 Fedora 项目下产生的。Red Hat 公司以前发布了非常流行的 Linux 发行版 Red Hat Linux。

5.3.1　安装前的准备工作

1. 磁盘空间

要安装 Fedora 9,硬盘上必须有足够的空间。如果想安装桌面系统加上默认的应用程序,那么就至少需要 5GB 的空间。也可以在同一台计算机上安装多个 Fedora。如果真的需要这么做,那么空间当然也是加倍的[48]。

2. 网络配置信息

如果计算机需要连接到网络中,那么安装操作系统时还需要我们对计算机所在网络的配置信息有所了解。我们现在对网络还不了解,不过也没有关系,在后面的章节中会介绍相关知识。现在安装操作系统时可以先跳过网络配置这部分。等到学习过相关的网络知识后,再进行网络配置让计算机联网。

3. 安装盘

安装 Fedora 9 需要有它的启动安装光盘。如果没有启动安装光盘,可以到网上,如 http://fedoraproject.org/zh_CN/get-fedora,去下载它的多张 CD 或一张 DVD 映像的 ISO 文件。然后,用其他计算机上的 CD 或者 DVD 烧制程序把这些文件刻录到光盘上。很多 CD 或 DVD 烧制软件都有一个叫做类似“从映像制作光盘”的选项。例如,图 5-10 就是 Windows 操作系统下的一个烧制刻录软件的界面,上面就有一个“烧录映像”的选项。选择这个选项就可以按照提示用从网络上下载来的 ISO 文件刻录出 Fedora 9 的启动安装光盘。如果我们正在使用的操作系统是 Linux,那么在 Gnome 桌面环境中,一般用右键单击下载的 ISO 文件,然后在弹出菜单中选择“写到盘中”即可。KDE 桌面环境中可以使用应用程序 K3B,在其界面下选择“工具”→“烧制 CD 映像”,或者“工具”→“烧制 DVD 映像”。

图 5-10　Windows 下的一个光盘刻录软件

实际上，也不是一定要用光盘安装操作系统，我们还可以选择从硬盘安装，甚至网络安装。本书只介绍光盘安装，如果你想知道怎样用其他载体安装操作系统，可以到图书馆或者互联网上查找相关资料。

5.3.2 安装过程要点

准备好安装盘之后，就可以安装了。首先我们要用安装光盘启动计算机。打开计算机电源后，把第一张光盘放到光驱中。如果计算机设置的第一启动设备是硬盘，那么我们需要在启动计算机时，进入 BIOS 设置，将系统的启动顺序更改为光盘是第一个启动设备。做好这些工作后，屏幕上会显示出如图 5-11 所示的光盘启动界面。

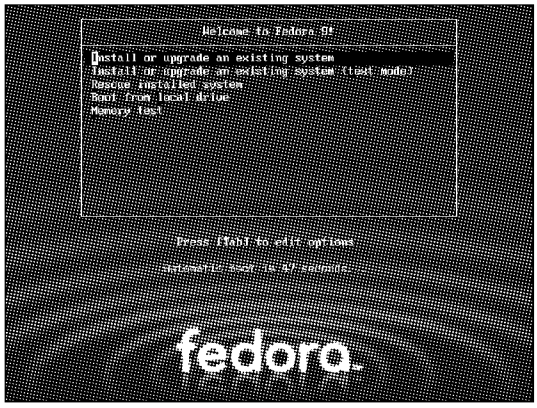


图 5-11 Fedora 9 安装启动界面

移动光标，选择第一项进入“安装或升级现有系统”。默认安装模式使用图形界面。我们也可以选择文本界面安装模式。本书只介绍默认的图形安装模式。具体的安装过程在附录 1 中有详细介绍。下面我们对安装过程中的一些要点进行说明。

1. 网络配置

如果计算机要上网，那么它首先要有硬件支持，一般是要装有网卡，并且网卡的驱动程序已经安装完毕。计算机的网卡还需要用网线或者无线连接到网络中。除此之外，还要对系统进行一些必要的配置。其中最重要的一项就是配置计算机的 IP 地址。IP 地址是计算机（实际上计算机用来通信的网络接口）在网络中的一个独一无二的号码，就像电话的号码一样。网络中的计算机通过 IP 地址互相识别。所以，要想让计算机上网，必须先给它指定一个与别的计算机不一样的号码。

Fedora 通常不需要我们手工进行网络配置。很多网络都有 DHCP（Dynamic Host Configuration Protocol，动态主机配置协议）服务。DHCP 服务是一种网络服务（后面很快也会介绍它）。DHCP 的主要任务是为计算机分配一个 IP 地址以及提供其他必要的配置信息。通常提供 DHCP 服务的网络会在网络中设置一个 DHCP 服务器。有了 DHCP 服务，用户就不用再配置 IP 地址等信息了，但是必须在系统配置时指明计算机使用 DHCP 服务。这样计算机就会在局域网中寻找 DHCP 服务器，然后得到配置信息自动进行配置了。

默认情况下，Fedora 激活计算机上的所有网络接口，并用 DHCP 对它们进行配置。既然现在对网络还不熟悉，那么安装操作系统时就选择“通过 DHCP 自动配置”。如果计算机

所在的网络提供 DHCP 服务，那么计算机就可以上网了。如果不提供，那它暂时还不能上网。等到学习第四篇“计算机网络平台”后，可以再回头配置这些网络信息。下面我们对安装过程中的网络配置内容再进行一些说明，你可以暂时跳过这些内容，等到学习网络知识后再回头来看。

安装程序会显示出在计算机上检测到的所有网络接口。每个接口在它所连接的网络中必须有一个唯一的 IP 地址。接口可以从 DHCP 服务器获得这个地址。我们也可以选择手工给某个接口分配 IP 地址。如果我们的计算机要作为服务器，那就不要使用 DHCP 服务，一定要手工配置。因为这样，即使 DHCP 服务器出现问题，我们的服务器也仍然能够接入网络。

Fedora 9 还要求我们提供其他的一些网络设置信息。这些信息是网络中其他一些系统的 IP 地址，主要包括网关的 IP 地址、DNS 服务器的 IP 地址。网关是一个让我们的计算机访问其他网络的设备，是计算机所在网络与其他网络连接的关口。网关的地址需要询问计算机所在网络的管理员来得知。DNS(Domain Name Service，域名服务)也是一种标准的网络服务。它负责把主机名转换成 IP 地址，或者反过来。很多软件依靠 DNS 服务器来定位网络中的机器和服务。Fedora 系统可以使用多个 DNS 服务器。如果主 DNS 服务器没有应答，那么计算机会向从 DNS 服务器提出查询。要指定 DNS 服务器，在安装程序给出的网络配置界面中的主、从 DNS 服务器的文本框中分别输入这些服务器的 IP 地址即可。同样这些地址需要询问网络管理员。不过，也可以查看网络中其他计算机上的这些配置信息，因为在同一网络中，这些信息都是一样的。

2. 时区配置

在安装程序的时区配置界面上我们会看到一个复选框叫“系统时钟使用 UTC”。UTC 是统一协调时间(Universal Coordinated Time)的英文缩写。这个时间也叫做格林威治时间(GMT)。说到格林威治时间你大概就清楚这个时间的意思了(如果还不知道那就自己去查资料吧)。系统时钟是计算机系统硬件的一部分。如果 Fedora 是计算机上唯一的操作系统，那么我们就选择“系统时钟使用 UTC”。Fedora 就会用时区设置判定本地时间和系统时钟 UTC 之间的时间差。这是类 UNIX 操作系统的标准做法。但是，如果计算机上也运行 Windows 操作系统，那么就不要选“系统时钟使用 UTC”。微软的操作系统会把 BIOS 时钟改变成与当地时间一致，而不是与 UTC 一致。这可能会引起 Fedora 意想不到的行为[47]。

3. 磁盘分区

在安装操作系统的过程中，一件重要的事情就是安排操作系统使用的磁盘分区。Fedora 系统至少要使用三个分区：一个挂载在目录“/boot”的数据分区，一个挂载在目录“/”的数据分区，以及一个交换分区。除这三个分区外，用户也可以根据需要定制其他分区。例如，创建“/home”分区来存储用户数据。每个分区都必须有一个挂载点。没有挂载点的分区是不能被访问。

如果你不知道怎样分区，那么接受 Fedora 9 安装程序提供的默认分区布局就行了。

Linux 系统用交换分区来扩展内存。当内存不够用时，Linux 会自动在内存和交换分区之间移动内存中的内容。此外，电源/能源管理系统也会把所有挂起的内容存在交换分区中。如果选择自己来指定交换分区，那么要创建容量比内存大的交换分区。

数据分区是用来存储文件的。在前面提到的最小分区方案下，Linux 系统/boot 目录下

的所有数据都在/boot 分区下。例如，文件/boot/grub/grub. conf 就在/boot 分区中(/boot 目录下存放的是与系统启动相关的文件)。所有/boot 分区之外的文件，如/etc/passwd 都在根/分区下。

Fedora 也支持把子目录分配给一个分区。例如，一些管理员会创建/usr 和/usr/local 分区。在这种情况下，在/usr/local 目录下的文件，如/usr/local/bin/foo 就在/usr/local 分区中。而其他在/usr 目录下的文件，如/usr/bin/foo，就在/usr 分区中。(如果我们创建了很多分区，而不是一个大的根/分区，那么升级就会容易一些。)[47]

每个分区都有一个分区类型。它指明该分区上的文件系统类型。安装 Fedora 时，如果没有其他文件系统类型的需求，那么就为数据分区选择 ext3 文件系统。

小提示

Fedora 系统至少要有三个分区：一个挂载在目录“/boot”的数据分区，一个挂载在目录“/”的数据分区，以及一个交换分区。

要创建容量比内存大的交换分区。

创建多个数据分区，而不是一个大的根/分区，升级或者重新安装 Fedora，就可以不删除用户数据文件。

Fedora 提供了一种对磁盘分区进行管理的机制称为 LVM(Logical Volume Manager，逻辑卷管理器)。LVM 是建立在硬盘和分区之上的一个逻辑层，用来提高磁盘分区管理的灵活性。

采用 LVM，分区首先被格式化成物理卷。多个物理卷被组合成一个卷组。每个卷组的全部存储空间然后被分成一个或多个逻辑卷。逻辑卷的功能和非 LVM 的标准分区很像。它们也有文件系统类型和挂载点。不过，只有处于活动状态的 Linux 系统才能读写逻辑卷。因此，初始化系统的/boot 分区必须放在 LVM 物理卷之外。

LVM 最大的优点是，管理员可以扩大或者缩小逻辑卷的大小，但是并不破坏数据。这与标准磁盘分区不一样。不过如果逻辑卷的大小缩到比数据所需的空间小，那就可能会丢失数据。我们可以创建满足当前要求的逻辑卷，留下额外的空间不分配。这样就可以很安全地按照需要用未分配空间增加逻辑卷的大小。

默认情况下，Fedora 安装程序会在 LVM 卷中创建数据和交换分区，外加一个卷外的/boot分区。如果要在系统中存储数据，那么可以在卷组中创建一个单独的/home 分区。有了单独的/home 分区，就可以升级或者重新安装 Fedora，但是并不会删除用户数据文件。

小提示

Fedora 支持逻辑卷管理器 LVM。

LVM 是建立在硬盘和分区之上的一个逻辑层。

LVM 使管理员能够改变逻辑卷的大小，但是并不破坏数据。

/boot 分区必须放在 LVM 物理卷之外。

4. 引导装载程序

Fedora 系统的启动需要有独立的系统启动盘，要么，必须通过 GRUB 等引导装载程序

引导。引导装载程序是一个小程序，它读取并启动操作系统。在系统启动一节中已经介绍过它。如果计算机上装有多个操作系统，引导装载程序通常会给出一个菜单，让用户选择启动哪一个操作系统。默认情况下，Fedora 使用 GRUB 引导装载程序。

如果计算机上已经安装了其他操作系统，Fedora 会自动检测并配置 GRUB 来启动它们。如果 GRUB 没有检测到这些操作系统，可以进行手工配置。可以用安装程序提供的选项来添加、删除、改变检测到操作系统的设置。

我们的计算机上有可能已经安装了别的引导装载程序。因为之前其他操作系统可能会安装它喜欢的引导装载程序，或者我们自己安装了第三方的引导装载程序。如果这些前面安装的引导装载程序不能识别 Linux 分区，那么我们就不能启动 Fedora。GRUB 可以启动 Linux 和绝大多数其他操作系统。但是，安装 GRUB 可能覆盖掉前面安装的引导装载程序。默认情况下，安装程序会在根文件系统所在驱动器中安装 GRUB。

GRUB 可以在没有操作系统的协助下读取很多文件系统。操作员可以中断启动过程选择不同的操作系统来启动，可以改变启动选项，或者从系统错误中恢复等。但是，这些功能在某些环境下可能会引入很严重的安全隐患。为此，安装程序允许我们给 GRUB 增加密码。这样操作员必须输入密码才能中断正常的启动过程。当然，如果系统有可信的操作员，或者物理上很安全，或者终端访问是受控制的，那么可能并不需要 GRUB 密码。但是，如果不可信的人可以接近计算机键盘和显示器，那他就可以重新启动系统并访问 GRUB。在这种情况下，设置密码是有用处的。

如果我们选择“使用引导装载程序密码”复选框设置启动密码。安装程序会弹出“改变密码”对话框。输入需要的密码，然后重新再输入一次来确认即可。

关于引导装载程序，就介绍这些内容。关于它的高级设置到 Fedora 的官方网站 http://fedora.redhat.com 上可以查询到相关的说明。

5.4　Shell 编程

Linux 的主要终端用户界面是 Shell。Shell 提供了联机命令和 Shell 脚本两种不同的计算机交互方式。这一节我们介绍怎样通过编写 Shell 脚本对计算机中的任务进行管理。从这些实例大家会发现 Shell 脚本的独特功能，找到使用 Shell 脚本的原因。Shell 脚本编写是系统管理员必须掌握的一门技能。我们以 Linux 上流行的 Bash Shell 为例介绍脚本的编写(本书参考文献[49-54])。Bash Shell 是免费软件，是 GNU 操作系统上默认的 Shell，因此经常被称为 Linux Shell。

5.4.1　Shell 的启动

在用户登录成功后，Shell 将被启动并始终作为用户与操作系统内核交互的手段直至用户退出系统。因为有很多种不同的 Shell 可供选择，所以系统上的每位用户都设定一个自己喜欢的 Shell 作为默认 Shell。每个用户的默认 Shell 在系统的/etc/passwd 文件里指定。Passwd 文件中包含有每个用户的 id 号、用户名，以及用户登录后立即执行的程序等信息。大多数情况这个执行程序都是 Shell。例如：

```
root:x:0:1:super user:/:/bin/bash
```

就是/etc/passwd 文件中的一行。这一行指明 root 用户登录成功后，系统执行/bin/bash 程序，即 Bash Shell。这个 Shell 也被称为登录 Shell。

如果 passwd 文件中指明启动的不是 bash，你又不想改变 passwd 的设置，那么可以在当前 Shell 的命令行中输入 Bash Shell 名就可以了。例如，你登录后启动的是 Bourne Shell，那么只要在命令行中输入 bash，Bash Shell 就会被启动起来。

Bash Shell 启动后，它会查找系统文件/etc/profile，并执行其中的命令。/etc/profile 是一个系统级初始化文件，由系统管理员设置。里面是一些需要在每个用户登录时执行的任务，如查找用户新邮件等。执行完/etc/profile 后，Bash Shell 接着会在用户的主目录下（系统为每个用户创建一个主目录）查找名为.bash_profile的初始化文件，并执行其中的命令。如果用户没有.bash_profile文件，那么它将查找是否有.bash_login初始化文件。如果有，则执行；如果没有就继续查找是否有.profile文件。执行完这些初始化文件之后，我们将看到命令提示符“$”出现在屏幕上。/etc/profile 文件为系统的每个用户设置环境信息。当用户第一次登录时，该文件被执行。.bash_profile里面有每个用户自己专用的 Shell 设置信息。默认情况下，它设置一些环境变量，通常它还会执行用户目录下的另外一个文件.bashrc。.bashrc文件里面是一些在 X-windows 中启动 Bash Shell 时进行的一些初始化设置。但是，这些初始化设置在用户登录时启动 Shell 也要进行，因此，通常.bash_profile中就不再填写这些内容，而是直接调用.bashrc了。你可以根据需要更改这些初始化文件。不过，通常这些文件在目录下都是隐藏保存的。

小提示

Shell 有两种启动方式：一是用户登录后，由操作系统根据配置文件直接启动；二是用户在命令行中输入 Shell 的名字来启动。

Bash Shell 启动后，会依次执行/etc/profile、.bash_profile、.bash_rc 等几个初始化文件。

5.4.2 Shell 命令

Bash Shell 有很多命令。这些命令从功能上大致可以划分为以下几类：目录操作与管理、文件操作与管理、系统管理与维护、用户管理与维护、系统状态、进程管理、通信命令及其他命令。

Bash Shell 命令从实现方式上可以分为内置命令和外部命令两种。内置命令是 Shell 内部源码的一部分。外部命令是与 Shell 身份等同的、经过编译的二进制程序。它们被存储在磁盘上。Shell 执行内置命令的开销很小。对于外部命令的执行，因为要从磁盘上把命令对应的程序读取出来，因而速度远不及内置命令。我们可以用 type 命令来查看命令是内置还是外部的。例如，在命令提示符后输入

```
type echo
```

会显示出

```
echo is a shell builtin
```

表示 echo 是一个内置命令。如果输入

```
type useradd
```

会显示

```
useradd is /usr/sbin/useradd
```

表明该命令实际上是/usr/sbin/目录下的一个程序 useradd。

Shell 命令的一般格式为：

```
命令名　选项　参数
```

命令名和命令参数之间用一个或几个空格分开。选项也有称为开关的，用来扩展命令的特性或者功能，一般以连字符“-”开始。根据需要用户可以使用两个以上的选项。一行可以写多条命令。命令之间用分号“;”隔开。

在用户输入命令之后，Bash Shell 首先检查命令是否是内置命令。如果不是，则检查是否是一个应用程序。这里的应用程序可以是 Linux 本身的实用程序，如 ls 和 rm，也可以是购买的商业程序或公用软件，如 gostview。然后 Shell 试着在搜索路径中寻找这些应用程序。搜索路径即我们第 1 章中提到 PATH 变量的内容，是一个能找到可执行程序的目录列表。如果输入的不是一个内置命令，并且在路径里没有找到这个可执行文件，那么 Shell 将会显示一条错误信息。如果命令被找到，那么 Shell 内置命令或应用程序将被分解为系统调用传给 Linux 内核。

当 Bash Shell 执行应用程序时，它将创建一个新进程。子进程和父进程拥有一样的环境，唯一的区别只有进程号。因为内置命令包含在 Shell 自身里面，所以 Shell 执行内置命令不需要创建新的进程。

Bash Shell 支持命令和文件名补全机制。在 Bash 下输入命令的一部分后，只要按下 Tab 键就可以把命令补全。例如，假定当前的工作目录包含一个名字叫做 woshiyigechangmulu 的子目录。当你想进入这个子目录时，一般要输入以下的命令：

```
cd woshiyigechangmulu
```

实际上，你可以不用输入这么长的文件/目录名，只要输入前面的一个或者几个字母即可。例如：

```
cd w
```

然后按下 Tab 键。如果当前目录下只有这一个以 w 开头的子目录。那么 Bash 就会把命令补全为前面那样的完整形式。如果目录下存在多个以 w 开头的文件或子目录，那么 Bash 不会补全命令，而是给出一个喇叭声提示。这时如果你再按一次 Tab 键，所有以 w 开头的文件名和子目录都会被它们列出来。然后，你可以继续在 w 后面添加字母并按 Tab 键，一直到 Bash 能够区分出来你到底是要输入哪个名字，它就会为你补全这个命令。不只是文件或目录名，对于其他命令名也适用上述补全规则。

在 Bash Shell 中，有一些字符具有特殊意义，它们表示字符本身之外的内容。这些字符被称为元字符，也被称为通配符。表 5-1 是一些常用的元字符。

表 5-1 Bash 常用元字符

元字符	含　　义
*	代表任意字符串，包括空字符串
?	代表任意单个字符
[… , - , !]	按照某种形式代表指定的字符，“…”给出列表、“-”给出范围、“!” 表示不匹配
.	当前目录
..	上一级目录
\	转义符，消除紧跟在后面的单个字符的特殊含义
‘’	消除被括在单引号中的所有特殊字符的含义
“”	消除被括在双引号中大部分特殊字符的含义，但不消除 $ 、'、"、\4 个字符的特殊含义
&	在后台运行
|	管道符，管道符之前的命令的标准输出作为其后命令的输入
＜＞	重定向操作符，分别表示输入和输出的转向，即输入和输出不再来自标准设备键盘或者显示器，而是将跟在“＜”后面文件的内容作为输入，输出到跟在“＞”后面文件名所指的文件中
（命令）	在子 Shell 中执行命令

例如，如果命令行中出现：

[a-d, x, y]　代表字符 a,b,c,d,x,y；

z*　　代表以字符 z 开始的任何字符串，如 zhangli；

x?y　　代表以 x 开始，以 y 结束，中间为任意单个字符的字符串，如 xay、xdy；

[!Z]　　代表不是 Z 的单个字符，如 a，或者 L 等。

如果在 Shell 命令行提示符后输入命令：

```
"ls [a-d] *.* "
```

则 Bash 会列出所有以字符 a、b、c 或 d 开头带有句点隔开的后缀的文件名，如 all.jpg、b.cfg等。

下面是一些常见命令的用法和功能。

- echo "some text"：将文字内容 some text 打印在屏幕上。
- ls：显示文件列表。
- cp sourcefile destfile：将文件 sourcefile 复制为 destfile。
- mv oldname newname ：重命名文件或移动文件，即将文件 oldname 改为 newname。
- rm file：删除文件 file。
- cat file.txt：把文件 file.txt 的内容输出到标准输出设备(屏幕)上。
- file somefile：得到文件类型。
- sort file.txt：对 file.txt 文件中的行进行排序。
- uniq：删除排好序的文件中重复出现的行，如 sort file.txt | uniq。

- expr：进行数学运算，如 expr 2＋3。
- find：搜索文件，如根据文件名搜索，find . -name filename -print。
- dirname file：返回文件所在路径，如 dirname /bin/tux 返回 /bin。
- head file：打印文件 file 的开头几行。
- tail file：打印文件 file 的末尾几行。

可以根据需要去 Linux 的帮助中去查找适合的命令。命令的功能和具体语法可以通过帮助命令 man 和 info 来查看。例如，man ls 或者 info ls 就可以查看到 ls 命令的用途和用法。

5.4.3　编写 Shell 脚本

1. 一个简单的示例

在 Linux 下，脚本的使用非常广泛。在很多目录下，都可以看到脚本文件的影子。现在我们就来看看怎样写 Shell 脚本。图 5-12 是一个简单脚本示例。它的功能是在屏幕上输出“hello world”。

我们首先要用编辑器建立这个脚本文件。假设我们用 vi 编辑器（是 Linux 一个常见的编辑器程序）将这段内容输入到一个名字为 hello 的文件中。现在我们就有了一个脚本文件，就可以在 Bash Shell 中执行它了。不过，这个文件必须具有可执行的属性才行。所以，在执行之前，我们用命令 chmod 来使其具有可执行属性。具体的命令为：

```
chmod +x hello
```

然后，我们就可以在 Shell 的命令提示符后输入：

```
./hello
```

来执行这个脚本文件了。图 5-13 显示了刚刚描述的过程。

```
#!/bin/sh
#对变量赋值：
a="hello world"
# 现在打印变量 a的内容：
echo "A is:"
echo $a
```

图 5-12　hello 脚本的内容

```
[zl@zhanglivmware book]$ vi hello
[zl@zhanglivmware book]$ chmod +x hello
[zl@zhanglivmware book]$ ./hello
A is :
hello world
[zl@zhanglivmware book]$
```

图 5-13　hello 脚本的执行

我们可以使用任意一种文字编辑器，如 gedit、kedit、emacs、vi 等，来编写 Shell 程序。

现在对这个简单的 Shell 小程序作一个解释。程序的第一行为：

```
#!/bin/sh
```

符号“#!”用来告诉系统该用哪个程序来解释脚本中的行。在这个例子中，我们使用/bin/sh来解释执行这个脚本程序。#!这一行必须是脚本顶端第一行。

程序的第二行为：

```
#对变量赋值：
```

是一个注释行。在 Shell 脚本中以符号“#”开头的句子表示注释。注释的内容直到这一行的结束。注释可以让别人很容易看懂自己的程序，也能够让自己在很长时间没有使用这个脚本之后，仍然可以很快想起当初写这个脚本的目的。

接下来的一行是：

```
a="hello world"
```

根据上一行的注释，你可以猜到这行的功能是对变量赋值。这里 a 是一个变量。等号“=”表示对变量赋值。赋值的内容是“hello world”。与其他程序设计语言一样，在 Shell 脚本中同样用变量来表示存放信息的单元。每个变量由一个名字来标识。所不同的是，在 Shell 脚本中，所有的变量都是字符串类型。因此，我们并不需要对变量的类型进行声明。变量的赋值方式为：

```
变量名=值
```

注意等号两边不能有空格。写 Shell 程序最麻烦的就是格式，尤其是空格符的使用，所以要尤其小心。取变量值时，需要在变量前加一个美元符号 $：

```
$变量名
```

我们继续来看刚才的程序，接下来的三行是：

```
#现在打印变量 a 的内容：
echo "A is:"
echo $a
```

同样符号“#”后面为注释行。从这行我们知道后面的语句是要显示变量 a 的内容。echo 是一个 Shell 命令。它的功能是显示一行文本。倒数第二行 echo 输出了字符串“A is:”。最后一行，在 a 前面多了一个符号“$”，这一次 echo 输出的是变量 a 的值。

好了，现在你可以把这个脚本写入自己的计算机并运行它。然后，再试着做些更改。你会发现编写脚本其实不太难。

2. Shell 变量

现在我们来对 Shell 脚本中的变量进行一些稍微详细的介绍。

Shell 变量的名字必须以字母或下划线开头。名字中可以包含字母、数字和下划线，除此之外的任何其他字符都会表示变量名的结束(注意)。变量名中的字母是区分大小写的。Shell 变量能够而且只能存储字符串。即 Shell 只有一种类型的变量——字符串变量。给变量赋值使用等号。例如：

```
Shuliang=10
```

等号前后不能有任何空白符。要想给变量赋空值，可以在等号后面跟一个换行符。

当引用变量的值时，需要在变量前加上一个“$”符号。例如：

```
echo $Shuliang
```

显示结果为：

```
10
```

未定义变量的隐含值为空。也就是说，如果引用一个前面没有定义和赋值的变量，那么显示的是一个空值。例如，如果我们没有在脚本的任何地方对变量 kong 进行赋值，那么语句

```
echo "变量 kong 的值是$kong!!!"
```

的显示结果为：

```
变量 kong 的值是!!!
```

现在假定我们想让一个名为 Xianshi 的变量中存放字符串 Xianzai de renshu shi 10，我们使用如下赋值和显示语句：

```
Xianshi=Xianzai de renshu shi 10\n
echo -e $Xianshi
```

Xianshi 赋值语句最后面的两个字符\n 是个转义序列，表示换行。echo 命令后面的-e 选项表示支持转义序列。即在这里 echo 显示变量 Xianshi 时，会把\n 显示为换行而不是两个字符\n。当我们运行这个脚本时，会发现脚本运行到给 Xianshi 赋值的这条语句时，Bash 会提示说它不认识命令 de。这是因为 Bash 认为赋值在 Xianzai 后面的空格处结束了，所以 Bash 认为 de 是一个命令的开始。可是它并不认识这个命令。为了解决这个问题，我们可以用单引号' '将这个字符串值括起来：

```
Xianshi='Xianzai de renshu shi 10\n'
```

这样 Bash 就能够知道要赋给 Xianshi 的字符串是到哪里结束的，而不再用空格来做标记了。用单引号' '括起来的字符串可以出现空格、制表符和换行符等特殊字符。不过，如果我们把上面的语句修改成：

```
Xianshi='Xianzai de renshu shi $Shuliang\n'
echo -e $Xianshi
```

这时显示的结果是：

```
Xianzai de renshu shi $Shuliang
```

而不是

```
Xianzai de renshu shi 10
```

这是因为单括号内不支持变量替换。如果想达到变量替换的目的，必须使用双引号" " 将值串括起来，即：

```
Xianshi="Xianzai de renshu shi $Shuliang\n"
```

用双括号括起来的字符串可以出现空格、制表符和换行符的特殊字符，而且允许有变量替换。

现在想把显示的内容再改变一点儿。变成 Xianzai de renshu shi 10ge(这里例子中想用字符的地方没有使用英文，因为拼音是我国法定的符号。)我们这样写：

```
Xianshi="Xianzai de renshu shi $Shuliangge\n"
echo -e $Xianshi
```

再来运行这个脚本，发现结果并不是我们所想象的。而是

```
Xianzai de renshu shi
```

这是因为 Bash Shell 把 Shuliangge 看作是一个变量名。显然，这个变量并没有出现过，所以它的值是空。怎么能让 Shell 把 Shuliang 看作是一个变量名，并且不需要在 Shuliang 和 ge 之间加入空格呢？如果紧跟在变量名称后面的字符是字母、数字或下划线时，可以用花括号{}将变量名称括起来。这样就可以使变量名称与它的后续字符分隔开。即：

```
Xianshi="Xianzai de renshu shi ${Shuliang}ge\n"
echo -e $Xianshi
```

这样就可以得到想要的结果了。

从作用范围来讲，Shell 变量可以被分为全局变量和局部变量两种。局部变量是在某一局部特定环境下使用的变量。其作用范围仅限于定义它的 Shell。

全局变量是一种特殊的变量，可以被任何运行的子 Shell 来引用。全局变量通过 export 命令来定义，格式如下：

```
export variables
```

其中 variables 是要变成全局变量的变量名表。一旦变量被定义为全局变量，对于以后的所有子 Shell 来说，它们都是全局变量。没有用 export 命令定义过的变量都是局部变量。

子 Shell 不能存取由父 Shell 设置的局部变量，也不能改变父 Shell 的变量值。子 Shell 中可以存取和修改父 Shell 的全局变量，但这种修改对于父 Shell 全局变量没有任何影响。也就是说，在子 Shell 中无法改变全局变量的值。在子 Shell 中改变全局变量的值，实际上只是对全局变量在子 Shell 中的副本进行更改，不影响原全局变量的值。子 Shell 中局部变量的使用优先于全局变量。在子 Shell 中用 export 命令定义的全局变量和对此变量的修改对父 Shell 的变量没有影响。全局变量保持它的全局性，不仅能直接传递给它的子 Shell，而且子 Shell 还能将它传递给子 Shell 的子 Shell。在对变量赋值之前和之后的任何时候都可以将变量转换成全局变量。

从存放内容上来看，Shell 变量可以被分成 4 种变量形式，分别是用户自定义变量、位置变量、环境变量以及预定义的特殊变量。它们的含义如下。

1）用户自定义变量

用户自己定义的变量，存放的内容由用户根据情况来指定。用户按照规定指定变量的名字和值。前面例子中出现的变量都属于这种类型。

2）位置变量

位置变量是一种特殊的变量。顾名思义它是与位置有关的变量。什么位置呢？是在命令行中输入脚本文件名执行程序时所输入参数的位置。Shell 将整个命令行的第一个字符串定义为位置变量 $0，后面的参数依次为 $1，$2，…，$9。例如，在 hello 脚本中加入如下语句：

```
echo "$0 $1 $2 $3 $4 $5..."
```

然后我们在 Shell 的命令提示符后输入：

```
./hello ni hao ma?
```

来运行这个脚本，则脚本运行的显示为：

```
./hello ni hao ma?...
```

即 $0为./hello，$1为 ni，$2为 hao，$3为 ma，$4为?，而 $5则为空。

3）环境变量

Shell 的执行环境由一系列环境变量组成。这些变量由 Shell 维护和管理。不过所有环境变量都可被用户重新设置。环境变量名由大写字母或数字组成。一些常见的环境变量如下。

CDPATH：执行 cd 命令时使用的搜索路径；

HOME：用户的主(home)目录；

IFS：内部的域分隔符，一般为空格符、制表符或换行符；

MAIL：指定特定文件(信箱)的路径，供邮件系统用；

PATH：寻找命令或可执行文件时的搜索路径；

PS1：主命令提示符，默认为“$”；

PS2：从命令提示符，默认为“>”；

TERM：使用的终端类型。

如果想设置环境变量，必须在给变量赋值之后或者设置变量时使用 export 命令。例如，在脚本中增加如下语句：

```
echo $PATH
export PATH=$PATH:/home/xiaoming:.
echo $PATH
```

则脚本的运行结果为：

```
/usr/local/bin:/usr/bin:/bin:/home/zl
/usr/local/bin:/usr/bin:/bin:/home/zl:/home/xiaoming:.
```

其中，/usr/local/bin：/usr/bin：/bin：/home/zl 为原来 PATH 变量的值，/usr/local/bin：/usr/bin：/bin：/home/zl：/home/xiaoming：. 为用 export 那条语句作用后的 PATH 的值。注意这里最后有一个句点，它可不是表示字符串的结束。它是表示当前目录。把它加入 PATH 变量之后，以后再执行当前目录下的文件时就不用在程序名前再输入“./”表示当前目录了。Shell 会根据 PATH 变量查找当前目录，就能找到我们要执行的程序了。

环境变量可以用于创建它们的 Shell 及其创建的子 Shell 或者进程。因此，它们经常被称为全局变量。一个 Shell 的环境变量会被传递给其启动的任意子进程。但是，子进程对环境变量的改变不会影响到原来父 Shell 的环境变量。按照惯例，环境变量的名字都是大写字母。

4）预定义的特殊变量

在 Shell 中有一组特殊变量，其变量名和变量值都是 Shell 预定义好的。常见的特殊变

量如表 5-2 所示。

表 5-2 常见特殊变量

变量名	含 义
$#	传递给 Shell 脚本的参数个数
$?	最近一次命令执行后的退出状态(返回码)：执行成功返回码为 0，执行失败返回码为非 0 值
$$	当前 Shell 的进程号
$!	最后一个在后台运行的(使用“&”)进程的进程号
$@	代表所有位置变量，即命令行中给出所有实际参数
$*	代表所有位置变量，即命令行中给出所有实际参数

我们在 hello 脚本中增加如下语句：

```
gftp&
echo "Canshu geshu shi $# ."
echo "Zuihou yici mingling de tuichuma wei $?."
echo "Dangqian jinchenghao wei $$."
echo "Zuihou yige houtai jinchenghao wei $!."
echo $@
echo $*
```

其中，gftp 是一个 Linux 下的 ftp 客户端图形应用程序。我们在脚本中启动这个程序，并在命令中用“&”符号让其在后台运行。我们在命令行中输入：

```
./hello Ni hao ma ?
```

启动脚本程序，运行的结果显示为：

```
Canshu geshu shi 4.
Zuihou yici mingling de tuichuma wei 0.
Dangqian jinchenghao wei 2528.
Zuihou yige houtai jinchenghao wei 2529.
Ni hao ma?
Ni hao ma?
```

同时，ftp 客户端图形应用程序 gftp 也被在后台启动起来，它的进程号是 2529。$@和 $* 的区别是，$@把参数解释成几个字符串，而 $* 则把若干参数作为一个字符串。即在上例中，$@的内容是“Ni”、“hao”、“ma”、“?”4 个字符串，而 $* 是“Ni hao ma?”一个字符串。

3. 算术运算

因为 Shell 中的变量实际上都是字符串类型，所以我们有必要在这里对 Shell 脚本中的算术运算做些说明。在 Shell 中有 4 种方式可以实现算术运算。下面分别加以说明。

1) 使用 expr 命令

具体形式是：

```
expr  数字 1  运算符 数字 2
```

例如，如果计算 4＋5，那么在脚本中可以这样写：

```
r='expr 4+5'
echo $r
```

注意 4、5、＋这三个字符之间要有空格。而且如果要把运算结果赋值给变量时，别忘了在命令前后加上反撇号。如果要计算 4 * 5，那么必须这样写：

```
r='expr 4\* 5'
echo $r
```

因为 * 在 Shell 中属于元字符，所以必须加上转义符号\来取消它的特殊含义。

2）使用 $(())

第二种方式是使用 $(())。例如，前面的表达式可以写成另外一种方式：

```
r=$((4+5))
echo $r
r=$((4 * 5))
echo $r
```

同样要注意括号和数字之间的空格。

3）使用 $[]

第三种方式是使用 $[]。例如：

```
#除法
r=$[24/5]
echo $r
#求余数
r=$[100 %5]
echo $r
```

4）使用 let 命令

Let 命令是 Shell 的内置命令，用来执行整型算术运算和数值表达式测试。具体示例如下：

```
#加法
n=10
let n=n+1
echo $n
#结果为 11
#乘法
let m=n*10
echo $m
#结果为 110
#除法
let r=m/10
```

```
echo $r
#结果为 11
#求余
let r=m%7
echo $r
#结果为 5
#求幂
let r=m**2
echo $r
#结果为 12100
```

这 4 种方法中，以 expr 命令方式的跨平台性最强，所以建议使用这种方式。

4. Shell 的控制结构

每种程序设计语言都提供一些控制结构，让用户根据情况控制程序的执行流程。Shell 也不例外。Shell 的控制结构主要有分支控制和循环控制两种。此外还有另一个重要的结构是体现模块化设计思想的函数结构。

1）分支结构

分支结构让程序执行到这里时可以根据当前的状态决定程序接下来的执行流程。Shell 支持 if 和 case 两种分支结构。因为分支结构首先要对程序执行当前的某些条件进行判断，所以我们先要介绍 Shell 用来测试条件的命令 test。

(1) test 命令

test 命令可用于对字符串、整数及文件进行各类测试。其命令格式如下：

```
test expression
```

或

```
[ expression ]
```

要注意第二种形式[和]两边的空格。这里 expression 是测试的条件。如果测试结果为真，则命令的返回码为 0(注意)；否则命令的返回码为非 0 值。

Test 命令可以进行字符串比较，这时 expression 是一个字符串测试表达式。表达式的形式和含义如表 5-3 所示。

表 5-3 test 字符串测试表达式

expression	值为真的条件
string1＝string2(等号两边有空格)	string1 与 string2 相同
string1!＝ string2	string1 与 string2 不相同
string	string 不为空串
-n string	string 不为空串
-z string	string 为空串

在脚本中加入如下语句：

```
name=Xiaofang
friend=Xiaofang
```

```
neighbor=Linlin

[ $name=$friend ]
echo $?
[ $name=$neighbor ]
echo $?
```

则脚本运行的结果为：

```
0
1
```

test 命令还可用于整数比较。我们说过，Shell 中的变量只有一种类型就是字符串类型。因此，Shell 变量中存放的都是字符串，并不存在整数值。当使用整数比较操作符时，是由 test 命令，而不是 Shell，将存放在变量中的字符串解释成整数值。test 命令支持的整数测试表达式如表 5-4 所示。

表 5-4　test 整数测试表达式

expression	值为真的条件
int1 -eq int2	两者为数值，且 int1 等于 int2
int1 -ne int2	两者为数值，且 int1 不等于 int2
int1 -ge int2	两者为数值，且 int1 大于或等于 int2
int1 -gt int2	两者为数值，且 int1 大于 int2
int1 -le int2	两者为数值，且 int1 小于或等于 int2
int1 -lt int2	两者为数值，且 int1 小于 int2

在脚本中加入如下语句：

```
count1=100
count2=203
average=199
test $count1 -eq $count2
echo $?
[ $count1 -le $average ]
echo $?
```

则脚本运行的结果为：

```
1
0
```

test 命令还支持对文件的测试。test 命令支持的文件测试表达式如表 5-5 所示。

表 5-5　test 文件测试表达式

expression	值为真的条件
-r FileName	FileName 存在，且为用户可读
-w FileName	FileName 存在，且为用户可写

续表

expression	值为真的条件
-x FileName	FileName 存在,且为用户可执行
-s FileName	FileName 存在,且其长度大于 0
-d FileName	FileName 为一个目录
-f FileName	FileName 为一个普通文件
File1 -nt File2	File1 比 File2 新
File1 -ot File2	File1 比 File2 旧
File1 -ef File2	File1 和 File2 大小相等

在脚本中加入如下语句:

```
[-r ./hello]
echo $?

[ ./hello -nt ./result.txt ]
echo $?

[ ./hello -ot ./rusult.txt ]
echo $?
```

其中,result.txt 是与 hello 脚本文件在同一个目录下,修改时间早于 hello 的一个文件的名字。则脚本运行的结果为:

```
0
0
1
```

Test 命令的这些表达式还可以通过逻辑运算符组合起来。逻辑运算符的格式和真值如表 5-6 所示。

表 5-6 逻辑运算符

运算符	含 义
!	逻辑非单目运算符,放在其他 test 表达式之前。取表达式运算结果的非值
-a	逻辑与运算符,执行两个表达式的逻辑与运算。仅当两者都为真时,才返回真值
-o	逻辑或运算符,执行两个表达式的逻辑或运算。只要两者之一为真就返回真值

逻辑运算符优先级由高到低的排列顺序为逻辑非!、逻辑与-a、逻辑或-o。逻辑运算符优先级比前面提到的表达式中的字符串操作符、数字比较操作符,以及文件操作符的优先级低。

我们在前面的脚本中增加如下语句:

```
[ $name=$friend -o $count1 -eq $count2 -a -r ./hello ]
echo $?
```

则脚本这两行运行的结果显示为:

```
0
```

用逻辑运算符连接表达式时一定要注意这些运算符的优先级。

(2) if 结构

if 结构是常见的分支结构。它的语意很清楚，如果(if)特定条件满足了，那么程序就做一些事情，否则就去做另外一些事情。最简单的 if 结构格式如下：

```
if 命令 1
then
   命令 2
   命令 3
   ⋮
fi
```

与其他程序设计语言不同的是，Shell 的 if 结构中 if 后面不是条件表达式，而是一个或者一组命令，如命令 1。这些命令执行结果如果为真，那么程序就执行 then 后面的命令 2、命令 3……直到遇到反写的 if。如果命令执行结果值为非 0，即命令因为某种原因失败，那么 then 后面的语句就将被跳过。例如，我们在脚本中加入如下语句：

```
echo "Do you like this course? (y/n)"
read answer
if [ "$answer" = y -o "$answer" = Y ]
then
    echo "Glad to hear it."
fi
```

运行脚本，屏幕上就会出现

```
Do you like this course? (y/n)
```

如果你输入 y 或者 Y，那么屏幕上就会显示出

```
Glad to hear it.
```

如果你输入其他字母，那么程序不会有任何显示。

除了最简单的结构外，if 结构还有其他几种不同形式。这其中包括 if 的完整结构。格式如下：

```
if 命令 1
then
    命令 21
    命令 22
    ⋮
else
    命令 31
    命令 32
    ⋮
fi
```

这种结构下，如果命令 1 执行成功，则程序进入 then 后面的区域，执行命令 21、命令 22 等命令。如果命令 1 执行失败，那么程序执行 else 后面的命令。

除了完整结构外，if 结构还可以嵌套在一起，形成连用结构。具体格式如下：

```
if 命令 1
then
    命令
    ⋮
    命令
elif 命令 2
then
    命令
    ⋮
    命令
else
    命令
    ⋮
    命令
fi
```

如果 if 后面的命令 1 失败了，则测试 elif 后面的命令是否返回值 0。如果这条语句成功了，那么执行 then 后面的语句。否则就检查下一条 elif 语句。如果所有的 elif 都不成功，就执行 else 后面的命令。else 后面的命令也被称为默认操作。现在我们写一个如下内容的脚本程序：

```
#!/bin/bash

echo "How old are you?"
read age

if [$age -lt 0 -o $age -gt 120]
then
    echo "Welcome to our planet!"
    exit 1
fi

if [$age -ge 0 -a $age -le 12]
then
    echo "A child is a garden of verses"
elif [$age -gt 12 -a $age -le 19]
then
    echo "Reble without a cause"
elif [$age -gt 19 -a $age -le 29]
then
    echo "You got hte world by the tail!"
elif [$age -gt 29 -a $age -le 39]
```

```
then
    echo "Thirty something..."
else
    echo "Sorry I asked"
fi
```

你能看出它的运行结果是怎样的吗？到你的计算机上去试一试运行一下，看看结果和你想的是否一致。

(3) case 结构

case 结构也就叫做分情况结构。它可以代替前面的 if/elif/else 结构。在分支情况很多时，case 结构看起来更清晰。case 结构的格式如下：

```
case variable in
value1)     命令 11
            ⋮
            命令 1n
            ;;
value2)     命令 21
            ⋮
            命令 2n
            ;;
…           ⋮
valuen)     命令 n1
            ⋮
            命令 nn
            ;;
*)          命令
            ⋮
            命令
            ;;
esac
```

程序执行到 case 语句时，将把变量 variable 的值与 value1，value2，…，value*n* 逐个比较，直到找到一个与其相同的值。如果找到匹配的值，那么程序就执行该值后面的命令，直到遇到双分号，然后跳到反写的 case 之后继续向下执行。如果没有找到匹配的值，就执行“*”后面的命令。

下面编写一个如下内容的脚本程序：

```
#!/bin/bash

ftype='file "$1"'
echo $ftype
case "$ftype" in
"$1: Zip archive"*)
    unzip "$1";;
"$1: gzip compressed"*)
```

```
        gunzip "$1";;
    "$1: bzip2 compressed" * )
        bunzip2 "$1";;
    * )
        echo "File $1 can not be uncompressed with smartzip";;
    esac
```

在 Linux 下有很多应用程序，如 zip、bzip2、gzip 等，都可以对文件进行压缩。这些程序压缩文件的方式不同，因此，用相应程序压缩的文件必须用相应的程序解压缩。这个脚本的作用就是通过判断压缩文件的类型来选择合适的解压缩程序。脚本中出现的 file 命令可以判断出文件的类型。File 命令前后反撇号表示用命令的执行结果来替换字符串。因此，变量 ftype 的值是 file "$1"的执行结果。我们把这个脚本文件的名字存为 smartzip。然后在 Shell 的命令提示符下输入如下命令执行该脚本：

```
./smartzip /usr/share/xmms/Skins/Bluecurve-xmms.zip
```

可以看到屏幕上显示的内容为：

```
/usr/share/xmms/Skins/Bluecurve-xmms.zip: Zip archive data, at least v2.0 to extract
Archive: /usr/share/xmms/Skins/Bluecurve-xmms.zip
inflating: Bluecurve/balance.bmp
inflating: Bluecurve/cbuttons.bmp
inflating: Bluecurve/eq_ex.bmp
inflating: Bluecurve/eqmain.bmp
⋮
```

这是 Zip 程序解压缩文件/usr/share/xmms/Skins/Bluecurve-xmms.zip 过程中的显示。

2）循环结构

循环结构是程序设计语言中另外一类重要的控制结构。Shell 主要提供了三种循环结构。它们分别是 for、while 以及 until 结构。下面我们逐一来说明。

(1) for 结构

For 循环结构用于把某些命令重复执行指定的次数。for 循环结构的格式如下：

```
for variable in arg1 arg2 … argn
do
    命令
    ⋮
    命令
done
```

For 命令后面跟一个用户自定义的变量，然后是关键词 in 以及一列词。执行第一次循环时，列表中的第一个词被赋给变量，并从词表中删除。然后程序进入循环体，执行 do 和 done 之间的命令。下一次循环将第二个词赋给变量，依次类推，直到词表的最后一个词。我们来看一个小例子：

```
#!/bin/bash
for name in $*
```

```
do
    echo Hi $name
done
```

假定这个脚本文件的名字叫做 forloop。我们在 Shell 的命令提示符后输入：

```
./forloop Xiaofang Lili Tom Tiantian
```

程序运行显示的结果为：

```
Hi Xiaofang
Hi Lili
Hi Tom
Hi Tiantian
```

For 结构只能执行指定次数的循环，如果我们事先不能决定循环的次数，而是要根据实际的情况来判断，那么就要使用 while 或者 until 循环结构。

(2) while 结构

While 结构用于在某种条件满足时循环执行某些命令的情况。它的格式如下：

```
while 命令
do
    命令
    ⋮
    命令
done
```

当 while 后面的命令执行成功时，即返回值是 0 时，程序进入 do 和 done 之间的区域，执行其中的命令。程序执行到关键字 done 后，就回头到 while 再次检查其后命令的返回值。这样反复直到命令的返回值非 0 为止。我们编写一个如下内容的小脚本，你来看看它的运行结果应该是什么样子的。

```
#!/bin/bash
echo "Who creats Linux?"
read answer

while ["$answer" !="Linus"]
do
    echo "wrong,try again"
    read answer
done
```

(3) until 结构

Until 结构与 while 结构类似，不同的是程序只在 until 后面的命令失败时，即命令返回码为非 0 时，才执行 do 和 done 之间的命令。程序执行遇到关键字 done 后，重新回头测试 until 后面命令的退出值，如此反复，直到该命令返回码为 0，程序结束循环，从 done 后面的命令继续执行。Until 结构的格式如下：

```
until 命令
do
⋮
done
```

Until 结构我们不给示例了。你可以自己试着写一个，或者把 while 结构的示例更改一下。

(4) 循环体中的其他命令

在循环体中，有时候我们会需要改变既定的循环流程。为此，Shell 提供了以下两个命令。

i. break 命令

break 是 Shell 的内部命令，用于在循环体中根据命令运行的返回条件，直接终止循环体内命令的执行。当执行 break 命令时，控制流从循环体中转移到 done 之后的第一条命令上。

ii. continue 命令

continue 是 Shell 的内部命令，用于在循环体中根据命令运行的返回条件，直接进入下一次循环命令的执行。当执行 continue 命令时，控制流直接转到本循环体中第一条命令上。

3) 函数

在 Shell 中也支持函数这种形式，以便使复杂问题的解决方案看起来清晰一些。如果在程序中几个地方使用了相同的代码，使用函数会方便很多。函数的格式如下：

```
function functionName( ) {
⋮
}
```

函数名后面的括号可以省略。函数体开始和结束的花括号两边必须有空格。函数在当前 Shell 中执行，即 Shell 并不为函数派生子进程。函数必须先定义后使用。我们编写了一个如下内容的脚本文件 checker：

```
#!/bin/bash
function usage { echo "error: $*"; exit 1; }
if [ $#!=2 ]
then
    usage "$0: requires two arguments"
fi
if [ !\(-r $1 -a -w $1 \) ]
then
    usage "$1: not readable and writable"
fi
echo The arguments are: $*
```

我们在 Shell 命令提示符下输入如下命令：

```
./checker ./t3 ./hello
```

t3 是当前目录中一个没有读写权限的文件，则显示的运行结果为：

```
error: ./t3: not readable and writable
```

5. Shell 脚本的适用环境

除了 Shell 之外，还有很多种程序设计语言，如 C、Java 等。那么什么时候使用 Shell 脚本呢？从前面的语法和命令中，你可以看出来，Shell 可以使大量的任务自动化，特别擅长系统管理任务。因为 Shell 脚本是由 Shell 解释执行的，执行速度较慢。所以 Shell 适合那些易用性、可维护性和便携性比效率更重要的任务。

具体来说，可以分为以下几种情况：

- 当一个问题的解决方法包含了许多 Linux 系统的标准命令操作时，可使用 Shell 程序设计语言。
- 如果一个问题能用在 Linux 系统中已建立的基本操作所表示，则使用 Shell 程序设计语言能构成更强的功能。
- 如果处理问题的基本数据是正文行或文件，则 Shell 可描述一个很好的解决方法；如果基本数据是数字或字符，那么使用 Shell 可能不是个好办法。

总之，在 Linux 系统中，虽然有各种各样的图形化接口工具，但 Shell 仍然是一个非常灵活的工具。Shell 不仅是命令的集合，而且还是一门非常好的编程语言。关键是看你能否充分发挥它的作用。

复习题

1. 计算机系统执行的第一条指令是怎样获得的？
2. 为什么要使用引导扇区？
3. 磁盘的第一个扇区为什么称为主引导扇区？
4. 引导装载程序的作用是什么？
5. BIOS ROM 中程序的作用是什么？
6. 计算机启动后，首先执行的是什么程序？
7. 什么是系统基本内存？
8. Linux 引导代码的主要任务是什么？
9. 为什么内核映像通常都是经过压缩的？
10. Linux 的启动辅助程序主要做什么工作？
11. 操作系统的内核初始化主要做哪三部分工作？
12. 为什么 CMOS 设置也被称为 BIOS 设置？
13. 怎样能够进入到 BIOS 设置？
14. 什么时候需要进行 CMOS 设置？
15. Linux 发行版和内核有什么区别？
16. Fedora 系统至少要有哪三个分区？作用是什么？
17. 安装 Fedora 时，如果创建一个/home 分区会有什么好处？
18. 交换分区的作用是什么？
19. LVM 是什么意思？使用 LVM 有什么好处？
20. Shell 有几种启动方式？
21. Bash Shell 启动时，要读取哪些文件？打开你计算机上的这些文件，看看其中的内容是什么？
22. 什么是内置命令？如何知道一个命令是内置命令还是可执行程序？
23. 怎样改变你的登录 Shell？
24. Shell 的局部变量和全局变量有什么区别？

25. Test 命令表达式中各种运算符的优先级次序是什么样的？
26. 以下 shell 脚本中，date 命令将执行多少次？
    for i in a b; do date; done
    (A) 0
    (B) 1
    (C) 2
27. 以下 shell 脚本中什么条件才会回显 hello?
    if [-d newitem]; then echo hello; fi
    (A) 如果 newitem 是一个目录。
    (B) 总是回显。
    (C) 从不，因为 netitem 是非空字串。
28. 当在 shell 脚本中使用 while/do 循环时，continue 语句的作用是什么？
    (A) 暂停一秒，然后继续执行。
    (B) 打断 do 循环，跳至 while 语句重新判断条件语句。
    (C) 跳至 done 语句后继续执行。
29. 哪些情况适合使用 Shell 脚本？

练习题

1. 查看你的计算机 CMOS 中的启动顺序，并将第一启动顺序更改为 CD-ROM。
2. 进入 BIOS 设置，为你的计算机增加启动口令。
3. 请试着练习使用 5.4.2 节中的 Shell 命令。如果有问题，请用 man 查看命令帮助。
4. 写一个命令列出所有名字中只包含两个字母的文件。
5. 写一个脚本，它能够将 1～100 之间的奇数输出到一个文件中，每个数一行。
6. 写一个脚本，它将当前目录下以数字开头的文件打包。
7. 写一个脚本，它将当前目录下名为 test.txt 文件中的内容除最后 10 行外全部清空。
8. 写一个脚本，它能够检查另外一个特定进程是否在运行，如果该进程没有运行，则启动该进程。
9. 写一个脚本，它能够执行简单计算器的功能。它提供一个菜单：

```
[a] Add
[s] Subtract
[m] Multiply
[d] Divide
[r] Remainder
```

具体功能如下：读取用户从菜单中选择的字母；提示用户输入 0～100 之间的两个整数；如果数字超出范围，输出错误信息并退出，否则按要求对两个数字进行运算；显示结果。

实验

下载最新的 Fedora 的 ISO 镜像安装文件。在你的计算机上安装一个虚拟机软件。然后在此虚拟机上用下载的镜像文件安装 Fedora。

第6章　应用软件开发平台

有了操作系统这个软件运行和用户操作平台，人们就可以使用计算机了。终端用户可以在上面运行应用程序，例如用文字处理软件编写文档、用播放器软件看电影、用电子表格软件管理自己的信息等。程序员可以利用其提供的程序调用接口编写应用程序，获得操作系统平台提供的服务。但是，对于程序员来说，光有操作系统提供的系统调用的接口还不行，他们需要有一个能够编辑程序代码的软件来组织源程序，需要有编译软件将用高级语言编写的程序转换成机器指令以便其能被计算机执行，还需要有帮助他们发现和纠正程序中逻辑错误的调试工具。除了编辑器、编译器、调试器这些基本工具之外，现代应用程序开发常常是基于某种共性软件设施进行的。那些软件设施为复杂应用系统的开发和运行提供了极大的便利，它们的表现形式一般为某种 API(Application Program Interface)，即从应用程序中可以直接调用的函数库，Java 中的各种框架(Framework)可以看成是这样的设施，整个 Websphere 也可以看成是这样的设施。这些实际上是支持代码重用的软件设施，加上各种工具就形成了应用软件开发平台。这个平台就好像是布满各种原材料和零配件的生产车间或装配场地，加上钳子、锤子等各种必需的工具。

应用软件开发平台本身也是软件，作用就是帮应用软件设计人员更有效地开发出应用软件来，它本身并没有直接的最终应用目的。有些开发平台是通用的，例如 CORBA；而有些是与特定领域相关的，例如 SAP，它能够最有效地支持企业管理类软件的开发。这一章我们就来了解程序员开发应用软件时所用的应用软件开发平台。特别值得指出的是，随着开发平台的发展，人们将相关工具和可重用的代码统一组织起来，形成所谓“集成开发环境”，程序员在其中工作能够更自如地进行软件开发，大大提高了工作效率。我们将首先从程序设计语言说起，因为这是程序开发的基础，也是应用软件开发平台得以体现的纽带。然后简要介绍上述工具和集成开发环境中的一些基本概念，旨在让读者对应用软件开发平台形成一个总体轮廓。至于各种工具和集成开发环境的具体使用，则不属于本书的内容。

6.1　高级程序设计语言

现在人们多采用高级程序设计语言来编写应用软件。不过，计算机世界中有几千种不同的高级语言。这些高级语言的语义、语法以及功能上都有着很大不同，因而它们也都有着各自不同的编译器。与之对应的就是程序员看到的不同的应用软件开发平台。因此，我们对应用软件开发平台的了解首先从高级语言入手。

计算机世界中的高级语言真的是非常多，人们很容易就可以随口讲出几个。这些语言产生的时间、背景不同，设计的目标也不同。一般程序员会根据所做项目的情况，有时还有自己的背景来选择程序设计语言。曾经出现过的程序设计语言有 Fortran、Pascal、Basic、C、Cobol、Ada、C ++、Java、Delphi 等，可以说数不胜数。在这些语言中，早期较常见的有 Fortran、Pascal、Basic 以及 C 语言。而现在最常用的是 C、C++ 以及 Java，目前使用 Visual

Basic、Delphi 的程序员也不少。Fortran 是最早产生的高级程序设计语言。它是为工程计算设计的，因而它非常适合处理公式以及进行各种数值计算。Basic 是一种非常容易学的语言。它现在仍然在被使用，只不过用的是它的高级后代，Visual Basic。而 Pascal 是一门系统地体现结构化程序设计思想的高级语言。因此，它曾经作为教学语言是各个大学计算机系的必修课程。现在的 Delphi 就是由 Pascal 语言发展而来的。C 语言是为了编写 UNIX 而设计的一种语言。因为是用来写操作系统的语言，所以它具有低级语言的一些特征，能够像汇编语言一样对硬件进行操作。但同时它也具有高级语言的特性，具有结构语句，也比较接近人的语言。因为这些特点，C 语言现在仍然是比较常用的一种程序设计语言。现在的操作系统如 UNIX、Linux 就是用 C 语言编写出来的。现在 C 语言更多的是被用在嵌入式程序设计等一类与底层硬件比较接近的应用软件的开发中。

C 语言属于结构化程序设计语言。人们把程序的任务分解为一个一个功能，用函数来实现，程序的主流程就是调用这些功能函数来完成任务。这种程序设计模式也被称为面向过程的程序设计，因为程序的主体是一个求解任务的过程。但是后来随着程序设计规模的扩大，以及软件业的发展，人们的思想发生了变化。人们发现如果把设计应用软件的事情看作是在创造一个个对象或者说物体，那就可以更好地模拟现实世界。程序设计就是设计一个个具有属性和行为的对象/物体。应用软件因此不再是函数的集合体，或者解题过程的描述，而是变成了对象/物体的集合体。程序员在程序中定义对象、对象的属性和行为，然后通过对象之间相互作用产生需要的结果。在这种思想下，产生了两个非常流行的程序设计语言——C++ 和 Java。

从名字上看就知道 C++ 和 C 是有关系的，而且好像应该是 C 语言的超级加强版，因为它有两个加号。事实上也是这样。C++ 保留了 C 语言的语法和语句。C 语言的程序在 C++ 环境中可以继续运行。因为在设计 C++ 时，C 语言已经非常流行，有不计其数的程序是用 C 语言开发的。不过，C++ 实际上比 C 语言要丰富多了。它增加了面向对象的一系列支持，如类、模板、名字空间、异常等概念，以及运算符重载、引用，还有在任何地方声明变量的能力等。因此，它的功能更强大。

Java 语言也是一个面向对象的程序设计语言。不过，另外一个追求目标使它与 C++ 出现了很大的不同。这个目标就是“跨平台”。我们知道有很多种不同的操作系统平台，不同的操作系统平台提供的终端用户接口、程序员级接口，以及底层的运行机制都是不一样的。因而在不同操作系统平台上运行的应用软件也是不同的，它们必须适应自己底层的平台环境。这些编译后的应用软件，甚至在源程序中，带有与操作系统平台相关的元素，如对特定系统接口的调用。所以，通常在某一个操作系统下设计开发的应用软件是不能在另外一个操作系统下运行的。例如，Linux 操作系统下的应用软件就不能在 Windows 操作系统平台下运行，反之亦然。而 Java 的跨平台目标就是要让在一个操作系统平台下写的应用软件可以在任意操作平台上运行。

Java 并没有采用像 C++ 那样传统程序设计的思路，即把高级语言源程序编译成可执行的机器指令代码文件，然后再执行。因为这样编译的结果就是针对编译器所在的具体操作系统平台的了，可执行文件也就被限制在特定的平台上了。为了实现跨平台的目标，Java 语言的源程序写完之后，会被转换成一种 Java 定义的中间状态的字节码形式。同时，Java 语言的设计人员还设计了一个在操作系统之上的称为 Java 虚拟机（Java Virtual

Machine,JVM)的平台层。Java 虚拟机实际上也是一个程序,它运行在操作系统平台之上,其作用就是把 Java 字节码翻译成程序当时所在运行平台的机器指令形式并执行。因此,Java 程序必须在 Java 虚拟机上才能运行。也就是说,到不同运行平台的映射是由 Java 虚拟机来完成的。Java 虚拟机屏蔽了不同运行平台的差异,让 Java 程序有一个统一的运行平台。可以看出,运行 Java 程序时,每条指令都是经过 Java 虚拟机的转换之后再执行的。这被称为解释执行。而像 C++ 语言写的应用程序被编译后生成的可执行文件的代码都是直接的机器指令,所以执行程序时,CPU 可以直接逐条完成指令。因此,Java 程序的执行要比其他编译后执行的程序要慢。这是为跨平台付出的代价。

小提示

目前常用的高级程序设计语言有 C、C++ 、Java 等。
C 语言是面向结构的程序设计语言。
C++ 和 Java 都是面向对象的程序设计语言。
Java 语言有跨平台的特性。

6.2　开发工具和开发环境

6.2.1　单独的工具

因为编写程序的需要,在程序设计的历史进程中,逐渐形成了各种供程序员使用的工具。这些工具中最主要的有三类,即用来编辑源程序的编辑软件、把高级语言编译成机器语言的编译软件,以及调试程序的调试软件。这些工具构成了应用软件开发平台的基本元素。

不过,刚开始时应用软件开发平台中的这些软件工具都是独立的,它们之间并没有必然的联系。程序员需要自己决定使用哪个工具来完成某个阶段的工作。某个阶段候选的工具也不只有一个。例如,编辑程序时,程序员可以选择 emacs,也可以选择 vi。实际上,有很多编辑软件都可以被程序员选用来编辑程序。这些编辑器中,有些是专门为编写程序而设计的。它们会对程序设计有特殊的支持,如用特殊的颜色显示程序设计语言中的特殊保留字。但有些编辑软件就是普通的编辑器。它们的目标就是帮用户写一般的文档,就像 Windows 中的记事本软件一样。但是,只要程序员愿意,他们同样可以用这些编辑器输入程序。采用单独的工具,在程序员编辑完源程序后,他们需要把代码以某种后缀名的文件形式保存起来。特殊的后缀名用来表明该源程序是用什么语言写的,如. c 后缀表示是 C 语言的源程序,这样做通常是为了满足后面要用到的编译程序的要求。接下来程序员需要用其所在平台下的编译软件来编译源程序。每种语言在支持的平台下都会有对应的编译器。例如,在 Linux 平台下,C 语言的编译器叫做 gcc。程序编译成功后,并不意味着程序的运行结果是正确的。因为程序中可能存在逻辑性的错误。为此,当程序并没有产生预期的结果时,程序员需要调试程序以确定到底哪里有问题。这时程序员会选择一些调试工具来帮他们解决问题。例如,如果在 Linux 系统平台下,程序员通常会选择一个叫做 gdb 的调试器来调试程序。

尽管程序员在开发应用软件的过程中,每个阶段都有相应的工具可以使用。但是,这些

工具并没有被关联在一起。程序员必须知道他要用什么工具做下一步的工作。这就好像生产车间里放着很多可用的工具，但是，操作员必须自己决定接下来用什么来做事。不过，缺点从另外一个角度来看就是优点。使用单独的工具，程序员有很大的自由度，如果他们不喜欢某个调试器，那么他们完全可以换一个更好的。因为出现得较早，所以这些单独的工具多是基于文本命令界面的。这里要说的一句是，尽管这些工具出现得很早，但是，它们的功能一点儿也不差，而且这些工具仍处于不断的发展进步之中，所以现在仍然有很多程序员喜欢使用这些工具。下面通过几个典型的工具来考察一下这些工具的面貌特征并体验在文本界面下开发应用软件的过程。

```
#include <stdio.h>

greeting()
{
        printf("Hello, world");
}

main()
{
        greeting();
}
```

图 6-1　Vi 的界面

首先是编辑器，以 Vi 为例。在 Linux 的 Shell 提示符后输入 Vi，就进入了 Vi 编辑器。图 6-1 是 Vi 编辑器的界面。Vi 就是指 Visual，意思是说马上就可以看到编辑的结果。这是相对于以前一些不能够马上见到编辑结果的编辑器来说的，当然现在编辑器绝大多数都支持所见即所得了。不过，从 Vi 的名字来看，在它出现时这个特性还是比较先进的。

Vi 支持用不同的颜色表示程序中不同的内容，如关键字 include 和其他语句的颜色就不同。函数 printf()的参数是一个用双引号包括起来的字符串“hello, world”，这个字符串也用另外一个颜色显示。这样程序员就可以看出双引号中间的字符串是什么，可以防止程序员忘记了写后面的双引号，造成程序的编译错误，或者运行中的逻辑错误。Vi 不支持鼠标，也没有命令菜单。它的命令和需要编辑的内容全是靠键盘输入的。为了区分键盘输入的内容是编辑命令还是被编辑的内容，Vi 设置了命令和编辑两个模式。按键盘左上角的 Esc 键即进入命令模式。在命令模式下键盘的输入被解释成命令，如输入

```
:q
```

表示退出 Vi。输入

```
:w
```

表示存盘。而按下字母键 a 表示在当前光标后输入内容，输入这个命令后，Vi 即进入编辑模式，如图 6-2 所示。这时屏幕下方的提示“——插入——”也提示这时可以往文本中添加内容。例如，从键盘输入“:q”，这两个字符就会出现在编辑内容中，如图 6-2 所示。这时按 Enter 键，光标会移到下一行。接下来如果我们按下 Esc 键再输入“:q”，就会看到这两个字符出现在屏幕的下方，如图 6-3 所示。表明现在我们是处于命令模式，输入了退出命令。如果接下来按 Enter 键，那么 Vi 就会结束退出。

Vi 设置了很多的命令。例如，在命令模式下，按下 x 键会删除光标处的字符，输入 dd 会删除光标所在的行，输入 s 会删除光标处的字符并进入输入模式，输入 S 删除光标所在的行并进入输入模式等。在命令模式下，按下 y 键(yank 复制)就可以把光标所在行复制下来，移动光标，然后再按下 p 键(put 放置)就可以把这一行粘贴到光标所在位置。按下斜杠键/，就可以在后面输入要查找的内容，如在图 6-4 中在“/”后输入 printf 然后按 Enter 键，就可以看到如图 6-5 所示的查找结果。Vi 还有很多命令，这里不一一介绍。

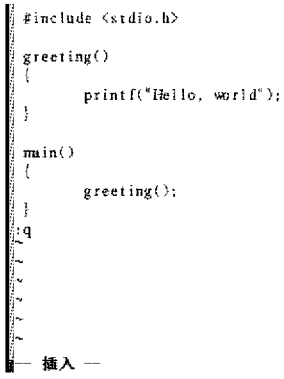


图 6-2　Vi 的输入模式

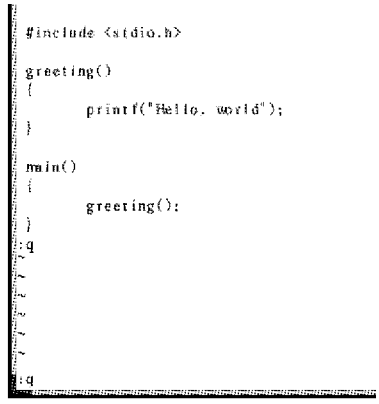


图 6-3　Vi 的命令模式

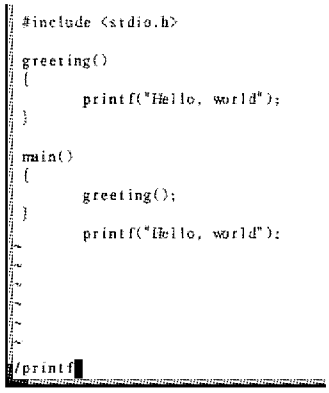


图 6-4　Vi 的查找命令

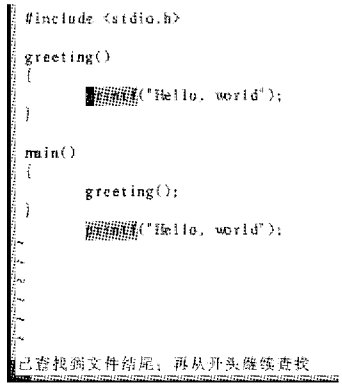


图 6-5　Vi 的查找命令结果

不支持鼠标、通过键盘输入命令，Vi 的这个特性让初次接触它的用户感觉很不方便。在某种程度上，可以说 Vi 的界面并不友好。这是因为它是特定条件下的产物，受制于当时的条件。不过，时至今日，Vi 仍然没有退出舞台，那是因为键盘输入命令实际上也有它的优势，那就是程序员的手指并不用在鼠标和键盘之间切换。对于熟悉命令的人来说，这样确实很省力、方便，速度非常快。因而，现在仍然有很多程序员喜欢 Vi，仍在使用它。

源程序编辑完毕之后，要想让它能在计算机上运行，必须要转换成机器语言。在 Linux 的命令行界面下，最常用的编译器是 gcc。gcc 是 GNU Compiler Collection 的简称。如果要编译前面编辑的文件，那么可以如图 6-6 所示输入命令：

```
gcc hello.c-o hello
```

假定前面的文件被我们保存为 hello. c。命令中的-o 是 gcc 的选项，该选项及其后的参数 hello 表示生成的可执行文件的名字为 hello，否则 gcc 会产生默认的文件 a. out。图 6-6 显示出一个错误提示，指出文件 hello. c 的 12 行有错误。这是因为前面在文件的末尾输入了“:q”。去掉这一行，重新编译，即成功地生成一个可执行文件 hello。运行 hello，屏幕上显示出“Hello, world”，如图 6-7 所示。这正是该程序的正确输出。

```
[root@zhanglivmware platform]# gcc hello.c -o hello
hello.c:12: parse error before ':' token
[root@zhanglivmware platform]#
```

图 6-6　gcc 编译文件 hello.c

```
[root@zhanglivmware platform]# gcc hello.c -o hello
hello.c:12: parse error before ':' token
[root@zhanglivmware platform]# vi hello.c
[root@zhanglivmware platform]# gcc hello.c -o hello
[root@zhanglivmware platform]# ./hello
Hello, world
[root@zhanglivmware platform]#
```

图 6-7　成功编译文件 hello.c 并运行

除了-o 之外，gcc 还有很多选项供程序员使用。在 Shell 提示符后输入

```
gcc-help
```

可以看到这些选项，如图 6-8 所示。

```
[root@zhanglivmware platform]# gcc --help
Usage: gcc [options] file...
Options:
  -pass-exit-codes         Exit with highest error code from a phase
  --help                   Display this information
  --target-help            Display target specific command line options
  (Use '-v --help' to display command line options of sub-processes)
  -dumpspecs               Display all of the built in spec strings
  -dumpversion             Display the version of the compiler
  -dumpmachine             Display the compiler's target processor
  -print-search-dirs       Display the directories in the compiler's search path
  -print-libgcc-file-name  Display the name of the compiler's companion library
  -print-file-name=<lib>   Display the full path to library <lib>
  -print-prog-name=<prog>  Display the full path to compiler component <prog>
  -print-multi-directory   Display the root directory for versions of libgcc
  -print-multi-lib         Display the mapping between command line options and
                           multiple library search directories
  -print-multi-os-directory Display the relative path to OS libraries
  -Wa,<options>            Pass comma-separated <options> on to the assembler
  -Wp,<options>            Pass comma-separated <options> on to the preprocessor
  -Wl,<options>            Pass comma-separated <options> on to the linker
  -Xlinker <arg>           Pass <arg> on to the linker
  -save-temps              Do not delete intermediate files
  -pipe                    Use pipes rather than intermediate files
  -time                    Time the execution of each subprocess
  -specs=<file>            Override built-in specs with the contents of <file>
  -std=<standard>          Assume that the input sources are for <standard>
  -B <directory>           Add <directory> to the compiler's search paths
  -b <machine>             Run gcc for target <machine>, if installed
  -V <version>             Run gcc version number <version>, if installed
  -v                       Display the programs invoked by the compiler
  -###                     Like -v but options quoted and commands not executed
  -E                       Preprocess only; do not compile, assemble or link
  -S                       Compile only; do not assemble or link
  -c                       Compile and assemble, but do not link
  -o <file>                Place the output into <file>
  -x <language>            Specify the language of the following input files
```

图 6-8　gcc 的选项

程序员可以按照自己的需要来选择这些选项安排 gcc 来完成不同的工作。例如，在前面的编译过程中，gcc 实际上顺序完成了预处理、编译、汇编、链接等几个工作。我们可以使用选项把前面 gcc 一次完成的 4 个步骤分开来做。首先进行预处理。预处理过程主要处理源文件中的 # ifdef、# include 和 # define 等命令。预处理使用的选项是-E，表示只进行预处理，不做编译、汇编和链接。预处理过程输入文件应该是 C 语言源文件。预处理过程会产生一个中间结果，我们用-o 选项指定存放预处理结果的文件名为 hello. i。这样预处理 hello. c 的具体命令即为：

```
gcc-E hello.c-o hello.i
```

接下来对预处理的结果进行编译。编译过程的输入就是前面产生的中间文件 hello. i。编译后生成汇编语言文件，所以文件名的后缀应该是. s。这时可以用 gcc 的选项-S，意思是只做编译，不做汇编和链接。命令如下：

```
gcc-S hello.i-o hello.s
```

下面对编译后的文件进行汇编将其转换成机器语言文件。在 Linux 下机器语言文件的后缀为.o。我们使用如下命令：

```
gcc-c hello.s -o hello.o
```

这时生成的文件就是机器指令了，但是仍然不能够执行。因为其中有一些对库函数的调用需要进行链接。最后，我们用下面的命令将机器代码文件与其他机器代码文件或库文件链接成一个可执行的二进制代码文件：

```
gcc hello.o -o hello
```

上述整个过程如图 6-9 所示。

```
[root@zhanglivmware platform]# gcc -E hello.c -o hello.i
[root@zhanglivmware platform]# gcc -S hello.i -o hello.s
[root@zhanglivmware platform]# gcc -c hello.s -o hello.o
[root@zhanglivmware platform]# ./hello.o
bash: ./hello.o: 权限不够
[root@zhanglivmware platform]# gcc hello.o -o hello
[root@zhanglivmware platform]# ./hello
Hello, world
[root@zhanglivmware platform]#
```

图 6-9　用 gcc 进行预处理、编译、汇编和链接

如果程序中存在问题，那么程序执行的结果就可能不像程序员期望的那样。这时程序员需要把问题找出来，调试工具会帮助程序员来完成这件事。例如，如果在前面的 hello.c 文件中增加一段代码，希望能够输出 1～5 几个数字的和，如图 6-10 所示。但是编译后程序的执行并没有输出预想的结果，如图 6-11 所示。

```
#include <stdio.h>

greeting()
{
        printf("Hello, world\n");
}

main()
{
        int sum,i;
        greeting();
        for(i=1;i<=5;i++){
                sum = sum +i;
        }
        printf("The sum of 1 to 5 is %d.\n",sum);
}
```

图 6-10　更改后的 hello.c

```
[root@zhanglivmware platform]# gcc hello.c -o hello
[root@zhanglivmware platform]# ./hello
Hello, world
The sum of 1 to 5 is 1073828719.
[root@zhanglivmware platform]#
```

图 6-11　hello.c 编译后运行的结果

检查程序好像也没有问题。这时，程序员就可以寻求调试器的帮助了。在 Linux 文本命令行界面下常用的调试器叫 gdb。要想使用 gdb，必须在编译源程序时使用-g 选项，这样编译过程中就会为调试产生必需的辅助信息。用-g 选项重新编译程序之后，就可以运行 gdb 来对生成的可执行文件进行调试了，如图 6-12 所示。在图 6-12 中，运行 gdb 之后，在 gdb 的提示符(gdb)之后，输入了 gdb 的调试命令 break，即

```
break 12
```

break 命令的作用是在程序中设置断点。上面这条命令的意思就是在 hello 程序的第 12 行设置断点。第 12 行就是 for 循环语句的那行。

接下来，用 gdb 的命令 run 来执行 hello 程序，如图 6-13 所示。程序将执行到第 12 行的断点处。这时我们输入 gdb 命令 next，让程序向下执行一条语句，即 for 语句，如图 6-14

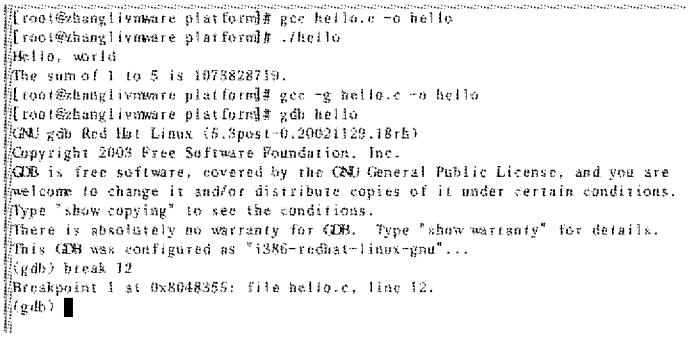


图 6-12　运行 gdb 调试 hello

所示。再用 gdb 的命令 print 查看变量 i 的值，从图 6-14 中可以看到变量 i 的值是 1。这是一个正确的值。

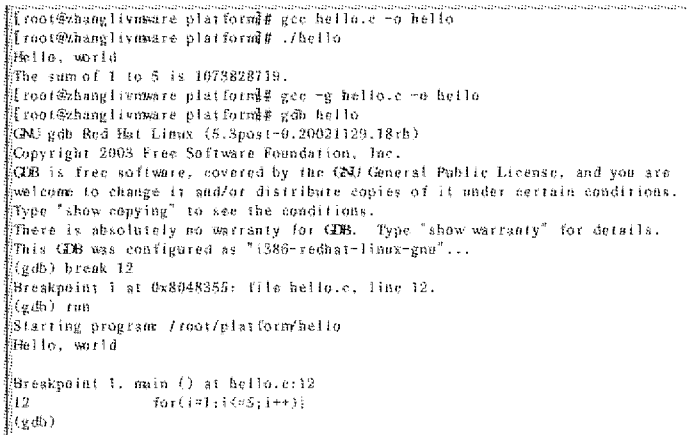


图 6-13　执行 gdb 命令 run

接着再输出变量 sum 的值，如图 6-15 所示。从图中可以看到变量 sum 的值是一个很大的数，即 1073828704。现在可以初步判断是变量 sum 的初值出了问题。

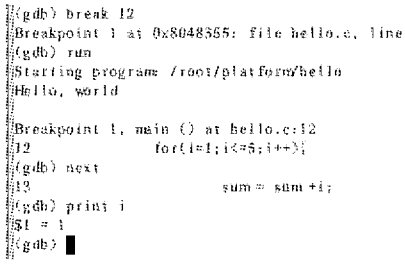


图 6-14　执行 gdb 命令 next 和 print

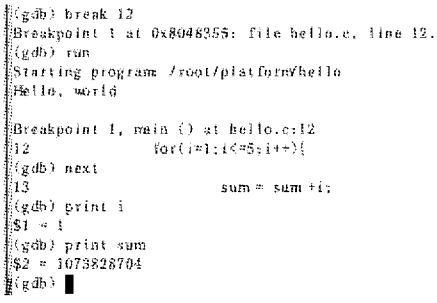


图 6-15　查看变量 sum 的值

输入调试命令 next，让程序继续执行下一条指令，即 sum＝sum＋i。然后再输出变量 i 和 sum 的值，发现 i 变为 2，sum 的值被加了 1，如图 6-16 所示。这说明程序到这里的逻辑是正确的。我们用命令 quit 结束 gdb 的调试过程。重新用 vi 打开 hello.c 源程序，发现

确实没有对变量 sum 赋初值就拿来使用了。因此，在 for 循环之前增加给 sum 赋初值 0 的语句，如图 6-17 所示。然后再编译运行，发现程序的执行结果就是正确的了，如图 6-18 所示。其实，这个例子如果不用 gdb 帮忙，程序员也可能会找出问题，这里只是用它来说明 gdb 的功能。对于复杂的程序，程序员没有 gdb 的帮助就会很难或者花很多时间才能找到问题的所在。

```
12              for(i=1;i<=5;i++){
(gdb) next
13                      sum = sum +i;
(gdb) print i
$1 = 1
(gdb) print sum
$2 = 1073828704
(gdb) next
12              for(i=1;i<=5;i++){
(gdb) print i
$3 = 1
(gdb) next
13                      sum = sum +i;
(gdb) print i
$4 = 2
(gdb) print sum
$5 = 1073828705
(gdb)
```

图 6-16　执行下一条指令

```
int sum,i;
greeting();

sum=0;
for(i=1;i<=5;i++){
        sum = sum +i;
}
printf("The sum of 1 to 5 is %d.\n",sum);
```

图 6-17　修改 hello.c 源程序

```
(gdb) next
13                      sum = sum +i;
(gdb) print i
$4 = 2
(gdb) print sum
$5 = 1073828705
(gdb) q
The program is running.  Exit anyway? (y or n) y
[root@zhanglivmware platform]# vi hello.c
[root@zhanglivmware platform]# gcc -g hello.c -o hello
[root@zhanglivmware platform]# ./hello
Hello, world
The sum of 1 to 5 is 15.
[root@zhanglivmware platform]#
```

图 6-18　重新编译执行 hello 程序

单独的开发工具赋予了程序员很大的灵活性，他们可以按照自己的喜好选择工具来完成应用软件的开发。不过，这些单独的开发工具一般都是在文本命令行界面下的。因此，一般经验比较丰富、接触计算机较早的程序员中采用这些工具的比较多，因为他们开始接触计算机时，计算机还以文本命令行界面为主，因而他们比较习惯这种界面。而对于新生代的程序员来说，他们更喜欢图形界面的操作平台，让他们自己选择工具，他们中的一些人会感觉茫然不知所从。

小提示

编辑、编译、调试工具是应用软件开发平台的基本元素。
单独的工具可以让程序员更自由地选择自己喜欢的工具。
单独的工具多是基于操作系统的文本命令界面的。
Linux 下常见的编辑、编译、调试工具有 vi、gcc 和 gdb。

6.2.2 集成的开发环境

为了克服单独工具的缺点，不再让程序员自己寻找和组合开发工具，人们把软件开发过程中必需的工具整合在一起，形成了集成的开发环境。这样，程序员只要启动一个集成环境软件就可以在其中完成全部的程序开发步骤了。这就好像把生产车间里面零乱的工具组装成了一个生产机床。在程序设计的历史中出现了很多集成的开发环境。这是因为有很多种不同的程序设计语言，而一般针对某一种语言，在每种操作系统平台上，都会有一个或者几个集成开发环境。例如，Windows 操作系统平台下，C++ 的集成开发环境就有 Visual C++、Turbo C/C++、C++ Builder 等。Java 的集成开发环境有 JDK、Eclipse、JBuilder、WebSphere Studio、NETBeans、JDeveloper 等。其中一些开发平台可以支持多种语言，如 Eclipse、KDevelop[55]等。我们将在第 6.2.3 节专门讨论最后这种类型的开发平台。

这些集成开发环境中早期的一些版本也是在文本命令行界面的操作平台下，如 Turbo C。后来随着图形界面的普及，这些开发平台就多是基于图形界面的了。这些开发平台在界面和用法上会有一些不同，但是主要界面轮廓和基本功能是一致的。它们都支持编辑、编译、调试、运行等基本功能。这些功能的用法也基本类似。这就好像各种汽车的样子和功能会有一些不同，但是，基本的组成都是要有车体、车轮，主要功能都是能在路上行驶。车的驾驶方式也不相同，不过，总的来说，都是通过控制方向盘、离合、刹车等装置来驾驶。集成开发环境与此非常类似。因此，熟悉一种开发平台，可以很容易地快速掌握另外一种。

图 6-19 是 Windows 下的集成开发环境 Visual Studio 2005 的界面。Visual Studio 2005（以下简称为 VS2005）是微软公司设计开发的一个集成开发环境。

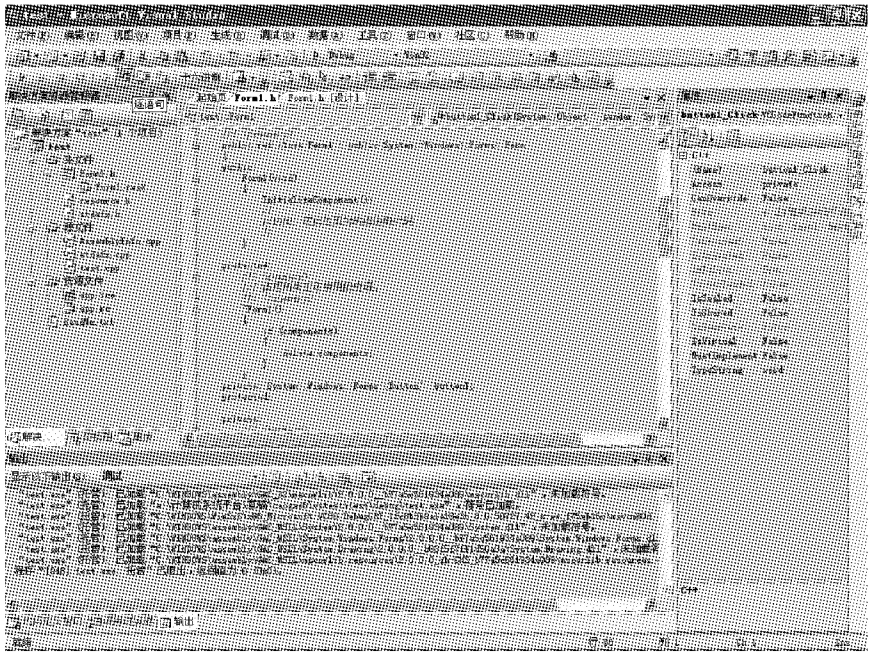


图 6-19 Visual Studio 2005 界面

图 6-19 反映出了目前多数集成开发环境的面貌。界面中间的窗口是程序的编辑窗口，程序员可以在其中输入代码。左侧的窗口会显示出当前项目（或解决方案）涉及的各种文

件。除此之外，左侧窗口中通常还有一些标签窗口来显示项目中包含的类以及类的属性等内容。屏幕下方是输出窗口，这里可以输出程序编译链接的结果、程序调试的中间结果等。屏幕的上方是菜单和工具栏。菜单和工具栏反映了开发环境的主要功能，如从菜单上可以看到编辑、调试、工具等主要菜单项；在调试工具栏中可以看到启动调试、全部中断、逐语句、逐过程调试等按钮。VS2005 支持可视化的界面设计，如图 6-20 所示，程序员可以直接摆放一些要在屏幕上显示的元素(在 VS2005 中这些元素被称为控件)。这正是 VS2005 名字中 Visual 的含义。在进行可视化界面设计时，屏幕的右侧通常会有一些属性栏或者工具箱等，供程序员创建控件类时设置或查看一些控件类的属性。目前的集成环境大都支持可视化的界面设计，即使它们的名字中没有 Visual。不同集成开发环境在细节上会有不同，但是大体上的样子和功能都与 VS2005 大同小异。

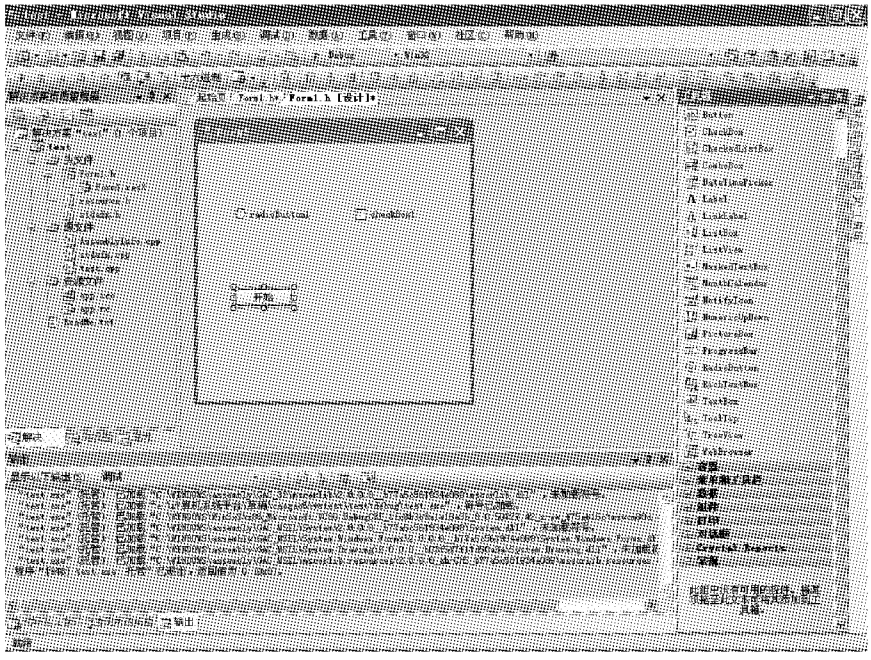


图 6-20　可视化的界面设计

集成开发环境给程序员带来了极大的便利。这是一个复杂而且功能强大的生产机床。程序员可以在上面完成所有开发应用软件所需的动作。例如，由于现在的应用软件通常都不再由一个文件构成，所以集成开发环境会帮助程序员把这些文件组织在一起，通常称为一个项目。程序员在开发应用软件时可以先创建一个项目，然后向其中添加相关的文件。图 6-21 是 VS2005 新建项目的窗口。而且，集成开发环境还为常见的项目类型设计了模板。例如，在图 6-21 中可以看到，VS2005 为 CLR(公共语言运行时态)类型的项目默认提供了 ASP. net Web 服务、CLR 控制台程序、SQL server 项目、Windows 窗体控件库、类库等应用程序模板。程序员可以根据自己的需要选择合适的模板，这样集成环境就会自动生成该类型项目所必需的一些文件和程序框架。然后程序员可以在这些项目文件中添加自己的代码，或者向项目中添加新的文件。这些文件会以项目为单位管理，它们会作为一个整体被存储到特定的地方。程序员可以对整个项目相关的文件进行编译和链接。如果在文本命令行的情况下，这些通常需要程序员自己编写一个叫做 Makefile 的文件，把这些文件的相

关性、编译时的一些选项等按照规定的格式写进 Makefile 文件，然后使用工具，如 make，来完成整个应用程序的编译和链接工作。这个过程并不是一个简单过程，而且很容易出错。而在集成环境下这些都由开发平台通过一些窗口中的选项设置自动完成了，如图 6-22 所示。用户不用再去查看和编写 Makefile 那样的文件了(关于 make 和 Makefile 请自己查阅资料)。

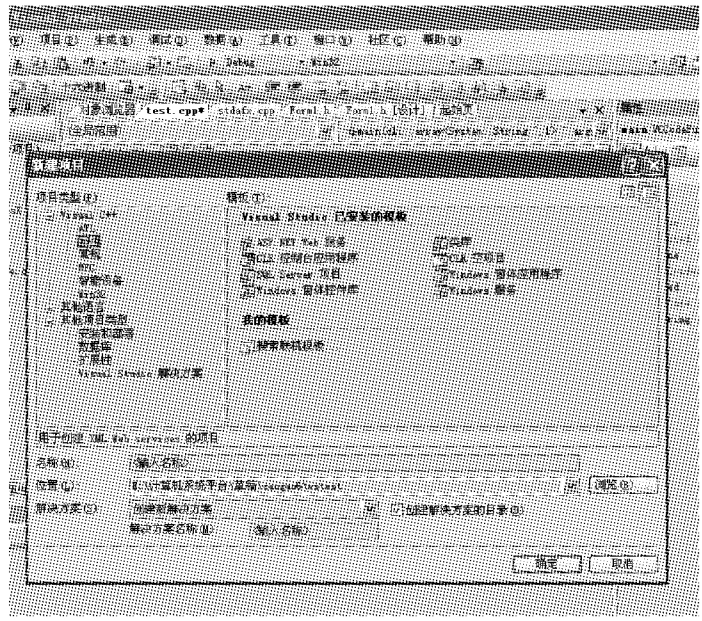


图 6-21　Visual Studio 2005 的新建项目

图 6-22　项目属性设置页

在集成环境中编辑工作也变得更加轻松。程序员可以在图 6-19 中左侧的窗体中双击选中需要编辑的源程序文件，这时屏幕中间的编辑窗口中就会显现出该文件的内容，程序员就可以在其中进行编辑。在图 6-19 中的编辑窗口中显示的是 Form1.h 的文件内容。从中可以看出程序中不同属性的文本被用不同的颜色标识出来，如关键字 public、void 等都用蓝色表示，而注释部分则用绿色表示。从图 6-19 上还可以看到，第一个绿色部分的注释提示程序员“在此处添加构造函数代码”。这个注释是集成环境自动生成的。这个 Form1.h 文件也是用户创建工程项目之后 VS2005 自动为用户生成的。VS2005 会自动为项目生成一个窗体，并生成窗体对应的源程序文件 Form1.h。Form1.h 中自动包含了必需的程序结构。需要用户添加自己内容的地方，VS2005 就生成相应的注释来提示程序员。集成环境中除了可以使用常用的快捷方式来进行文本的剪切、复制和粘贴之外，还提供了另外一些很方便的编辑功能。例如，图 6-23 是 VS2005 编辑菜单的选项，从中可以看到 VS2005 提供的编辑工具。这些功能包括撤销或者重复操作、查找替换、显示大纲、设置和使用书签，以及能够自动补全命令的 IntelliSense 等。其中，“查找”可以在全部项目的文件中查找需要的内容，“自动补齐”可以帮助程序员免于记忆一些文本内容。例如，在图 6-24 中，当我们在源程序中输入 Application::之后，VS2005 会给出一个提示框，里面显示出 Application 这个类的属性和方法，即可以放在 Application::之后的内容。程序员可以在其中选择所需内容。这样就不用必须把这些内容记得很确切了。如果在编辑窗口中把鼠标移动到某个字符单元上，单击鼠标右键，就会看到一个下拉菜单。例如，图 6-25 就是我们在函数 run()上单击右键产生的菜单。从图 6-25 中可以看到，VS2005 会通过这个菜单帮助程序员找到该内容的定义或者声明，或者对该内容进行复制、剪切等编辑动作，还可以进行断点设置或者运行到此等调试工作。从这些功能可以体会到集成开发环境为程序员提供的便利。

图 6-23　Visual Studio 2005 的编辑菜单条

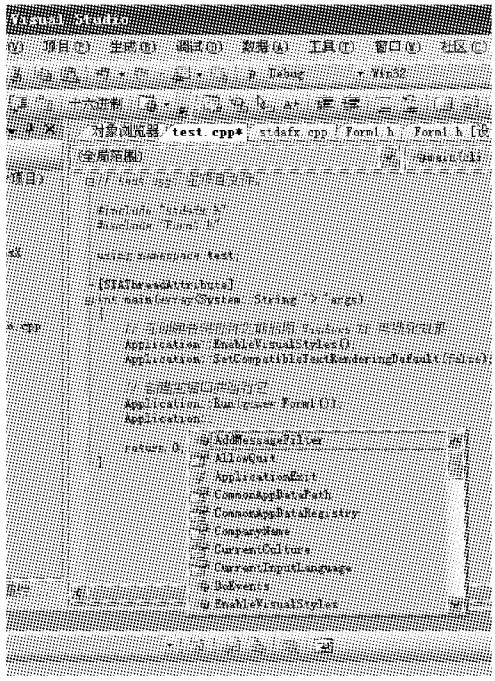


图 6-24　Visual Studio 2005 的编辑提示框

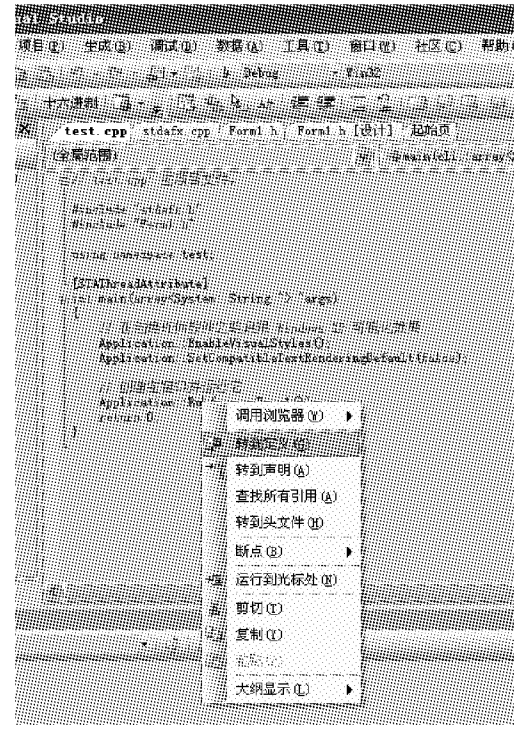


图 6-25　编辑窗口中的下拉菜单

在集成环境下，调试程序也变得非常简单。例如，从图 6-26 中可以看到 VS2005 的调试菜单。通过这些选项程序员可以在程序中设置断点，逐语句或者逐过程调试程序。程序

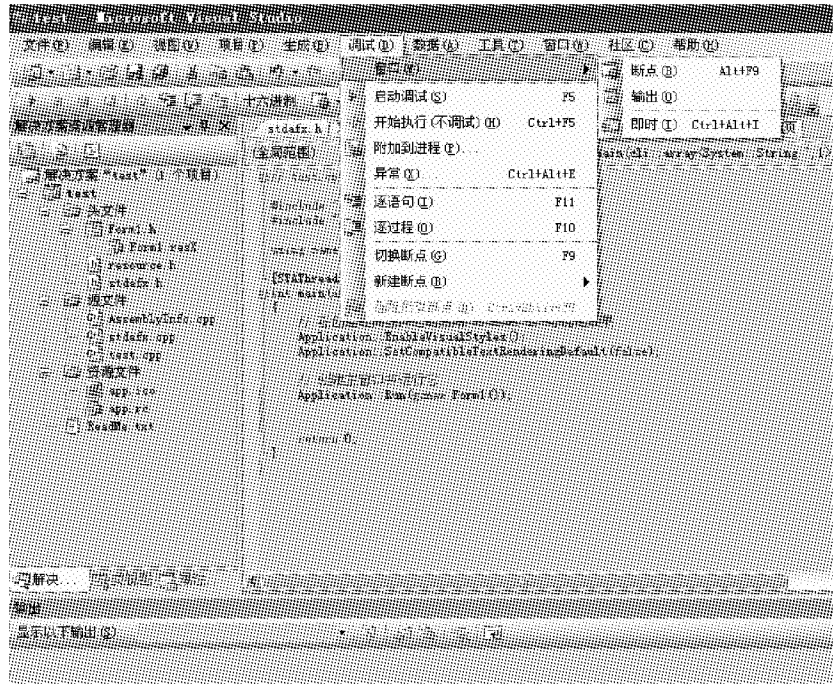


图 6-26　Visual Studio 2005 的调试菜单

员直接用鼠标在编辑窗口左侧相应的语句行旁边单击，就可以把断点设置完毕，如图 6-27 所示。这时单击工具栏上的绿箭头“启动调试”，VS2005 就会让程序运行到断点处。程序员可以按 F11 或者 F10 键逐语句或者逐过程调试程序。工具栏上要显示哪些按钮，同其他应用软件一样，由用户通过“视图”菜单项中的“工具栏”来选定。

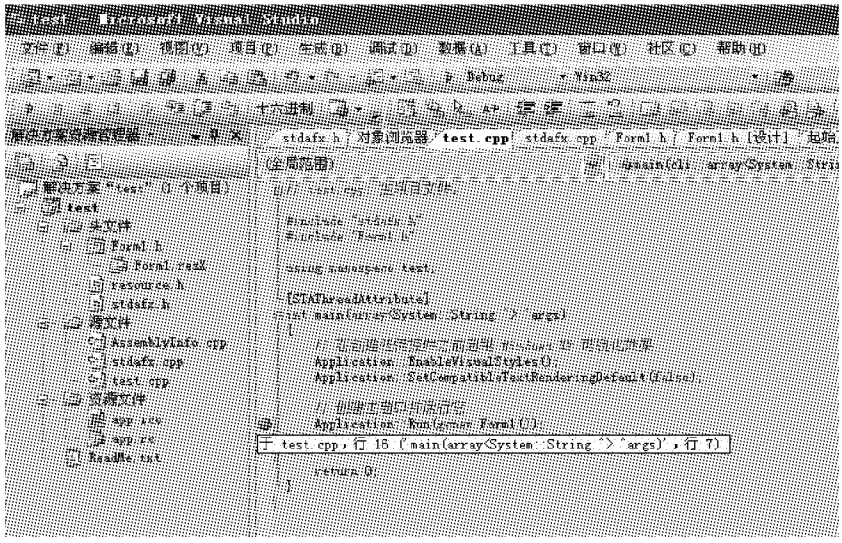


图 6-27　设置断点

集成开发环境的功能非常多。例如，VS2005 除了提供基本的编辑、编译、调试功能外，还有非常多的辅助功能。比如说，选定“工具”菜单中的“选项”，可以看到如图 6-28 所示的选项设置页。程序员可以在其中进行一系列设置，以实现方便应用程序开发的目的。例如，图 6-28 中显示的“导入和导出设置”项就可以让程序员把当前开发环境的设置保存在文件中。在另外一台计算机上可以通过“使用团队设置文件”来导入环境的配置文件。这样，两台计算机上开发环境的设置就可以完全相同了。这个选项可以帮助一个团队的开发人员实现统一的设置，如代码的缩进和格式化选项等。VS2005 的功能还有很多，这里就不再一一介绍。

随着功能的不断增强，集成开发环境有时复杂得需要程序员专门来花上一些时间来学习怎样使用它。这就像一个功能强大的机床需要操作员经过特定的使用培训一样。

其实，集成开发环境提供的很多功能，程序员都可以找到单独的工具得以实现，如用 make、gcc 实现编译，用 tag 工具可以使 Vi 能够定位到函数的定义等。不过，那样需要程序员对工具非常了解，而且这些工具多数用键盘控制，不向鼠标单击界面那么直观。

从前面介绍的内容，我们可以体会到集成开发环境，尤其是可视化集成开发环境为程序员开发应用软件提供了非常大的便利。因此，现在大多数程序员都工作在各种集成开发环境平台上。

最后，再对微软的集成开发环境多说几句。起初，微软针对于不同的语言分别推出了相应的开发平台，如 Visual C++、Visual J++ 等。不过现在的 Visual Studio 已经把 Visual C++、C#、Visual Basic 等语言的开发环境都集成在一起。用户可以选择使用其中任何一种语言来开发应用软件。但是，Visual Studio 这种多语言开发环境的集成只是一个静态的集成，

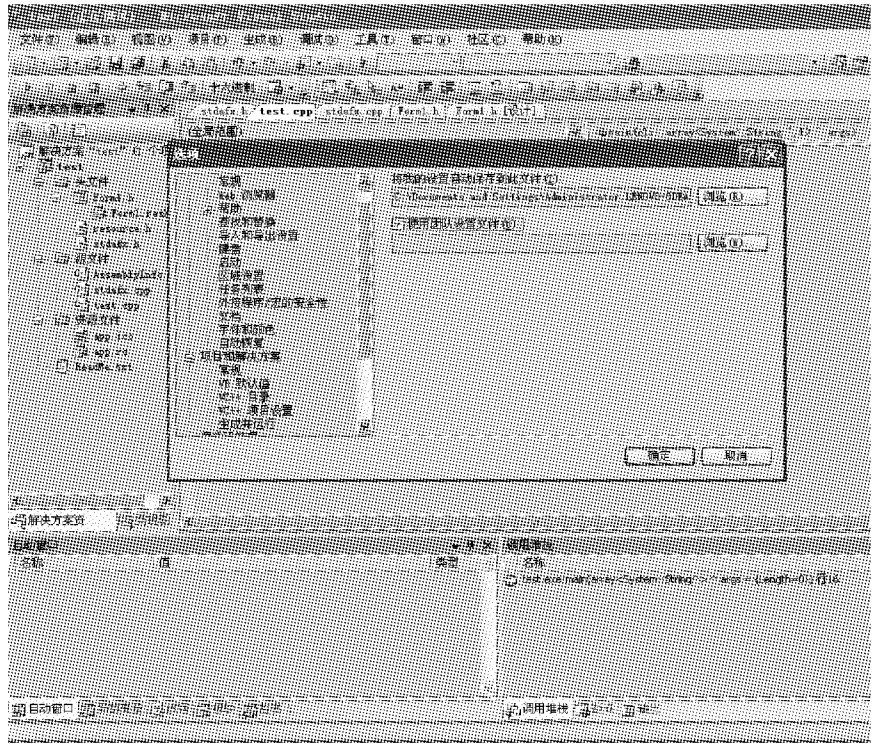


图 6-28　Visual Studio 2005 的选项页

它只能集成微软特定的开发产品，并不能集成其他的开发工具或者开发环境。也就是说，用户并不能够自己选择、添加或者去除其他集成的内容。这与下一节要介绍的可构建的集成开发环境还有很大不同。

小提示

人们把开发应用软件所需的工具集成在一起形成了应用软件集成开发环境。
集成开发环境为应用软件的开发提供了很大的便利，使程序员的开发过程变得轻松、快捷。
Visual Studio 是 Windows 操作系统中常见的集成开发环境。

6.2.3　可构建的集成开发环境

尽管 Visual Studio 可以集成多种语言的开发工具，用户也可以选择或者去除某种语言的开发环境，如选择安装 Visual C++ 环境，或者只安装 Visual Basic 环境等。但是这些能够选择的工具还是局限于微软自己开发的特定软件工具上。用户并不能选择其他工具，更不能自己设计编写开发工具，然后把它嵌入到 Visual Studio 中。如果一个开发平台可以让用户自己添加喜欢的工具，甚至能让用户设计实现自己的工具，那么这样的平台会具有更强的可扩展性和适应性，会更让用户青睐。实际上，现在的开发平台也正是在朝着这个方向发展。这种可构建的开发平台的一个典型代表就是 Eclipse。下面通过 Eclipse 来了解这种开放式开发平台的理念以及模式。

Eclipse 是一个基于 Java 的可扩展开发平台[99]。实际上，它只是一个框架和一组服务。

Eclipse 提供了一个插件机制。除了开发平台的核心基础外，Eclipse 平台上的所有其他元素都是通过插件构建的。也就是说，差不多 Eclipse 的每样东西都是插件。Eclipse 的标准扩展中带有一个插件开发环境 PDE（Plug-in Development Environment）。希望扩展 Eclipse 的软件开发人员可以在 PDE 中构建与 Eclipse 环境无缝集成的工具。由于这种开放机制，Eclipse 可以扩展支持各种语言的应用程序开发，如 C、C++、C#、PL1、COBOL 等[100]。目前支持 C/C++、Cobol 等编程语言的插件都已经存在。除了用于编辑、编译和调试应用程序的插件外，从建模、生成自动化、单元测试、性能测试、版本控制，直到配置管理的完整开发过程的插件也都已经被开发出来，如 CVS、Rational ClearCase、Rational XDE（代替 Rose）等等。Eclipse 甚至都可以被扩展成为图片绘制的工具，因为只要把工具做成符合 Eclipse 规范的插件就可以集成进来。当然，一般普通开发人员并不会去开发工具插件，通常是由工具开发商组织人来做这些事情。普通程序员可以利用开发商提供的插件来构建自己的 Eclipse 开发平台。Eclipse 是基于 Java 开发的，因此，它附带的标准插件集中包括了 Java 开发工具 JDT（Java Development Tools）。它支持 Sun 公司的 JDK 和用于生成代码文档的 JavaDoc 标准[57]。也正因此，目前使用 Eclipse 平台的开发者仍然以 Java 程序员为多，尽管 Eclipse 可以支持多种程序设计语言。

小提示

开放具有高可扩展性是应用软件开发平台的一个发展趋势。
Eclipse 是开放性应用软件开发平台的典型代表。

1. Eclipse 外观

Eclipse 是开源软件，任何人都可以从它的官方网站 www.eclipse.org 上下载 Eclipse。Eclipse 可以运行在 Windows 操作系统，也可以运行在 Linux 系统下。在 Windows 操作系统平台上，下载的 Eclipse SDK（软件开发包）可以解压到任意目录下，然后双击其中的 eclipse.exe 即可运行，因为 Eclipse 不需要安装。

图 6-29 是 Eclipse 3.2 的 Resource（资源）透视图（Eclipse SDK 是英文版的软件开发工具包，不过 Eclipse 已经能支持很多种语言，如简体中文、法语等。只要从 Eclipse 网站上下载语言包，然后解压缩到 Eclipse 目录中，重启 Eclipse 即可，我们这里以英文版为例）。Resource 透视图是很多旧版本的 Eclipse 启动时用户首先看到的一个界面（不过在 Eclipse 3.2 中，Resource 透视图并不是默认的初始界面）。透视图是 Eclipse 的一个术语。我们看到的这个界面在 Eclipse 里面被称为工作台（workbench）。工作台，顾名思义，就是用户工作的台面。工作台由几个被称为视图（view）的窗格组成，如图 6-29 左上角的 Navigator 视图。窗格的集合被称为透视图（perspective）。Resource 透视图是一个基本的通用视图集，用于管理项目以及查看和编辑项目中的文件。Navigator 视图可以让用户创建、选择和删除项目。Navigator 右侧的窗格是编辑器区域。根据在 Navigator 中选定的文档类型，编辑器区域会打开一个合适的编辑器窗口。Eclipse 提供标准的编辑器，其他的编辑器则需要通过插件来实现。每种文件有默认的编辑器。用户可以选中文件通过单击鼠标右键弹出的菜单项“打开方式”来选择其他编辑器。Navigator 下面的 Outline 视图显示当前编辑器中文档的

大纲。对于Java源文件,大纲显示所有已声明的类、属性和方法。Tasks视图收集关于正在操作的项目的信息。

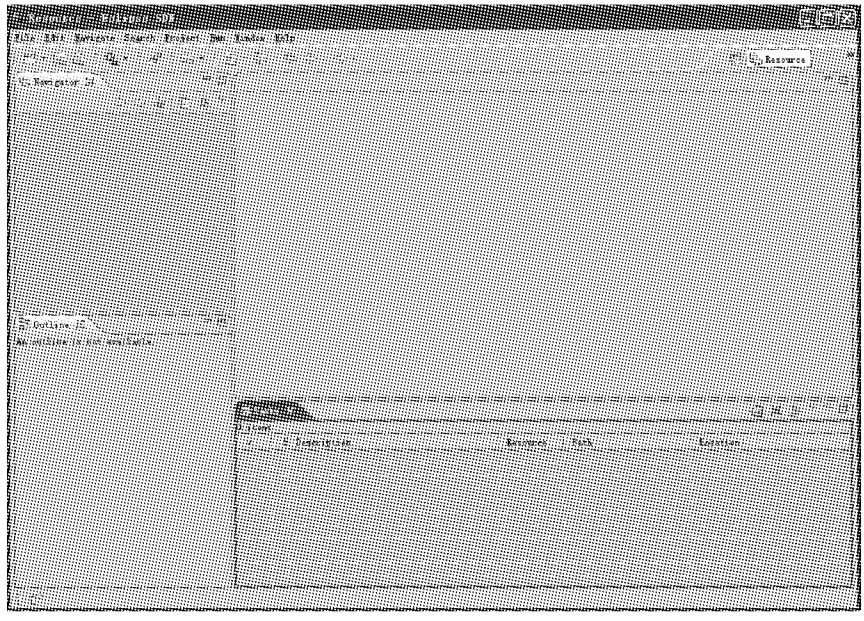


图 6-29　Eclipse 3.2的资源透视图

通过菜单条上Window菜单项下的open perpective,如图6-30所示,我们把透视图转到如图6-31所示的Java透视图(其实,它是Eclipse 3.2的默认工作台,进入Eclipse首先看到的就是这个界面),来编写Java应用程序。前面提到的Visual Studio提供的基本编辑、编译、调试等功能,在Eclipse的Java开发环境中都有。从图6-31中可以看到,Eclipse的Java程序开发界面的外观与Visual Studio非常相似。Eclipse虽然默认带了Java的开发工具包JDT,但是,它并不带有Java的运行环境JRE。所以,在使用Eclipse开发Java程序之前,需要安装JRE或者Java SDK。至于怎样在Eclipse下开发Java程序,就不详细介绍了。如果你想知道可以到网络上或者图书馆查找相关的资料。

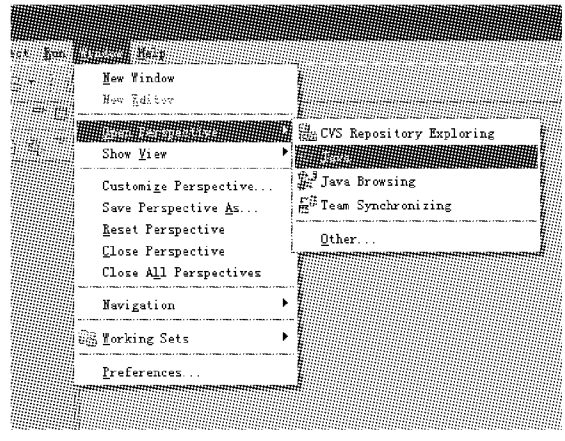


图 6-30　Eclipse 3.2透视图的转换

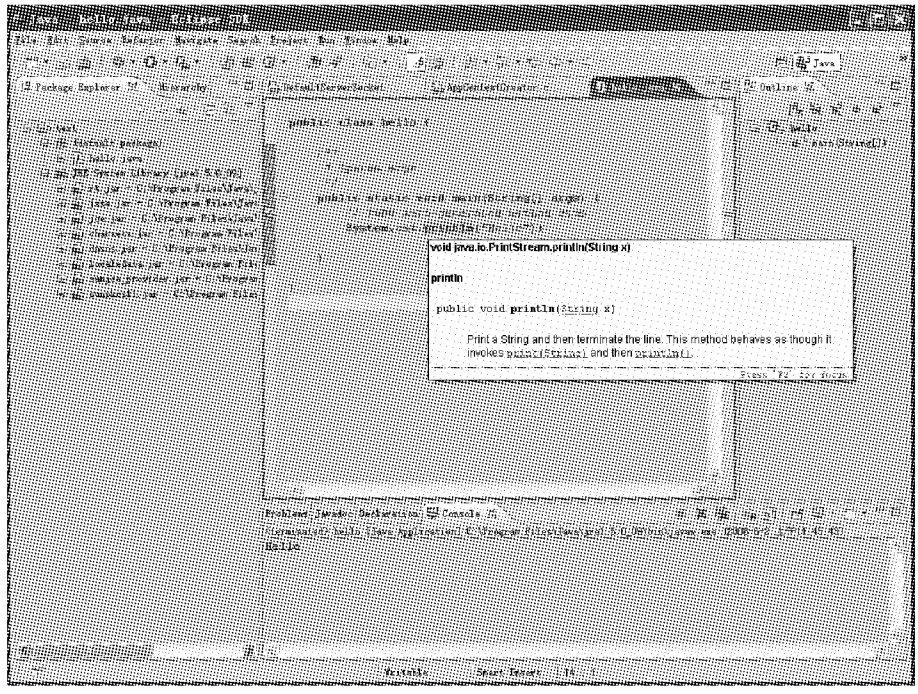


图 6-31　Eclipse 3.2 的 Java 透视图

2. Eclipse 插件的使用：构建 C/C++ 开发环境

这一部分我们通过构建 C/C++ 的开发环境来体会 Eclipse 的开放性。因为 Eclipse 平台只是一个框架，所以它不直接支持 C/C++ 程序的开发。要想构建 C/C++ 开发环境必须通过外部插件来实现。目前最常见的 C/C++ 开发环境插件叫做 CDT。CDT 是完全用 Java 实现的开放源码项目，是 Eclipse SDK 平台的一组插件。这些插件将 C/C++ 透视图添加到 Eclipse 工作台中。CDT 能为 Eclipse 平台提供功能完全的 C/C++ 集成开发环境。有了这组插件，用户就可以用视图、向导，以及高级编辑和调试功能来进行 C/C++ 程序的开发了。CDT 主要是针对 Linux 系统平台来设计的，但它在能使用 GNU 开发工具的所有环境（包括 Win95/98/Me/NT/2000/XP、QNX Neutrino 和 Solaris 平台）中都能工作。我们就以 CDT 为例来说明 C/C++ 集成开发环境的构建。

要构建 Eclipse 的 C/C++ 环境，首先要得到 CDT 插件。可以直接从网络上，如 http://download.eclipse.org/tools/cdt/releases/callisto/dist/3.1.2/下载 CDT 插件。很遗憾的是，尽管 CDT 是用 Java 编写的，但它还是不能做到与平台无关。所以，下载时需要根据 Eclipse 所在操作系统平台选择下载相应的 CDT。

安装 CDT 之前，要确保系统上有 GNU C 编译器，即 gcc，以及所有附带的工具，如 make、binutil 和 gdb 等，都是可用的。因为 CDT 最终是使用这些工具完成工作的。如果是在 Windows 系统中，那么就需要安装 Cygwin 工具箱或者别的类似工具。Cygwin 是 Windows 操作系统下的类 UNIX 环境，包括 gcc 以及所有必需的开发工具，如 automake 和 gdb 等。

下载完 CDT 的插件文件之后，将压缩的文件解压到临时目录中，然后将临时目录的

plugins 子目录中的所有内容都移到 Eclipse 的 plugins 子目录中。Features 子目录中的内容也要移到 Eclipse 的 features 子目录中。接下来，重新启动 Eclipse，选择 Open Perspective(打开透视图)菜单项，从弹出的窗口中就可以看到新出现的 C/C++ 选项，如图 6-32 所示。选择这一项，Eclipse 即显示出 C/C++ 透视图，如图 6-33 所示。在这个透视图中就可以创建 C/C++ 项目，编写和调试 C/C++ 程序了。

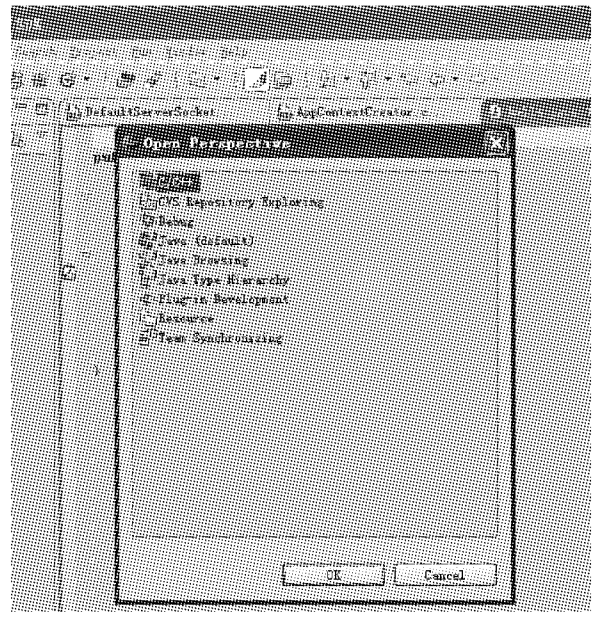


图 6-32　增加了的 C/C++ 透视图

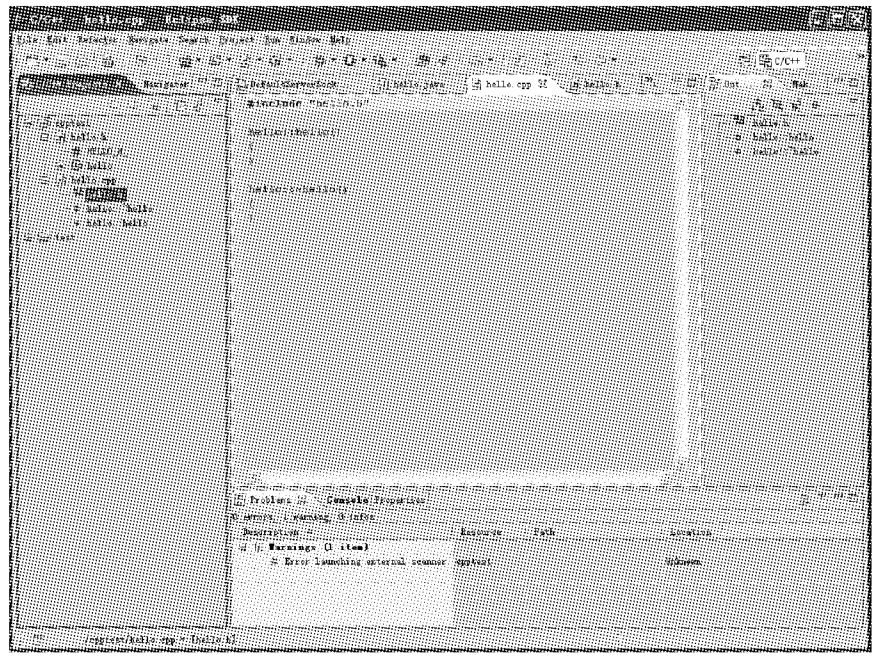


图 6-33　C/C++ 透视图

CDT 很复杂，所以被分成了几个组件，每个都以独立插件的形式出现，如提供调试功能的 CDT 调试核心(CDT Debug Core)、为编译器和调试器等外部工具提供启动机制的 CDT 启动(CDT Launch)等。所有的插件都是 CDT 正常工作所必需的。因此，我们可以从 CDT 文件解压后的 plugins 子目录中看到很多插件文件[101]。

从 C/C++ 环境的构建可以看到在 Eclipse 中加入插件是非常容易的事情。在后面的小节中我们会了解怎样开发 Eclipse 插件。

小提示

只要把插件文件复制到 Eclipse 目录下的 plugins 子目录下，重新启动 Eclipse，即可使用该插件。

3. Eclipse 体系结构

在对 Eclipse 平台有了初步的了解之后，我们来简要介绍 Eclipse 的体系结构，看看它是通过怎样的机制来实现它的可扩展性。Eclipse 的体系结构如图 6-34 所示，主要由平台运行库、工作空间、工作台、团队支持和帮助几个主要部分构成。

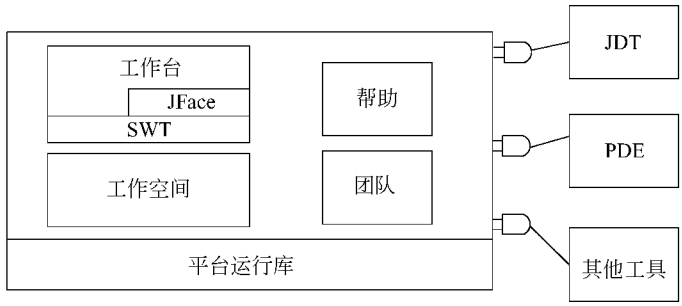


图 6-34　Eclipse 平台体系结构示意图

平台运行库(Platform run-time)是 Eclipse 的内核，负责在启动时检查已安装的插件，并创建插件的注册表信息。不过 Eclipse 只有在需要使用某个插件时才加载该插件。这样可以减少启动时间和资源占用。除了平台运行库外，Eclipse 其余所有的内容都是作为插件实现的。

工作空间(Workspace)和工作台(Workbench)是 Eclipse 中最重要的两个插件。工作空间负责管理用户资源，包括用户创建的项目、项目中的文件以及文件变更和其他资源。工作空间还负责通知其他插件关于资源变更的信息，如文件创建、删除或更改。各种工具都是在用户工作空间的资源上进行操作的。

工作台的作用是提供用户界面，它是用 Eclipse 的标准窗口工具包 SWT 和一个更高级的叫做 JFace 的 API 来构建的。JFace 建立在 SWT 基础上，提供用户界面组件，但它并不屏蔽隐藏 SWT。

团队支持(team)负责提供版本控制和配置管理支持。它会根据需要添加一些视图，来让用户与所用版本控制系统交互。除了提供版本控制服务的插件外，大多数插件都不需要与团队支持组件交互。

因为 Eclipse 是由用户来扩展的一个框架，与各种功能相关的帮助当然也是随用户的功能扩展而扩展的。因此，Eclipse 的帮助也很特别，跟平台本身也很一致。Eclipse 的帮助(Help)组件提供一个导航结构，让用户能够以 HTML 文件的形式为平台添加帮助文档。

既然 Eclipse 平台所有的组成部分都是以插件实现的，那么平台要想工作，这些插件必须结合成一个整体才行。Eclipse 采用一个称为扩展点的概念来实现插件之间的整合。扩展点实际上就是为插件留出的接口。每一个插件上都留有自己的扩展点。工作台和工作空间提供了大多数插件使用的扩展点，如工作台组件包含了一些扩展点可以让用户的插件扩展 Eclipse 用户界面，工作空间组件包含了可以让用户与资源(包括项目和文件)交互的扩展点。每个插件都是在现有一些插件的扩展点上设计开发的。扩展点就好像是可以插插件的插口。而各种作为工具的插件可以插在这些插口(扩展点)上，与 Eclipse 的其他插件联系在一起，形成一个整体平台。这也是插件名字的含义。图 6-35 是 Eclipse 插件以及扩展点机制的一个示意图。(可以在 Eclipse 平台帮助中找到基本组件提供的扩展点说明，在开发插件时会需要这些信息。在平台插件开发者指南中也会涉及这些内容。)

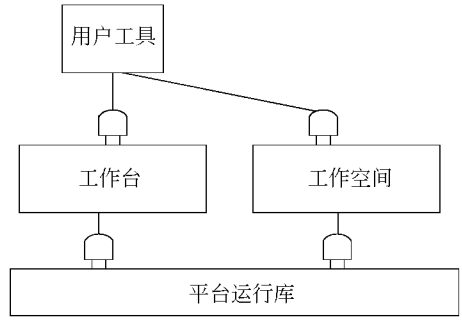


图 6-35　Eclipse 的插件和扩展点示意图

除了核心部分的平台运行库和插件形式的核心组件外，所有其他平台所需的工具都以插件形式插入到 Eclipse 系统中，如 Java 开发工具 JDT、插件开发环境 PDE 以及其他各种工具，如 C/C++ 开发环境等，图 6-34 说明了这种情况。

小提示

除了平台运行库，所有其他 Eclipse 的构成部分都以插件形式出现。
Eclipse 的插件之间通过扩展点整合在一起。
工作空间和工作台是 Eclipse 两个最重要的核心插件。

4. Eclipse 插件的开发

了解 Eclipse 的体系结构、插件机制以及使用过插件之后，下面来看看 Eclipse 的插件是怎样开发出来的。开发插件首先要确定插件要扩展哪个扩展点，也就是插件打算插在哪一个插口上。然后就可以根据自己的需要编写对扩展点的扩展代码了。此外，Eclipse 需要插件开发者提供一个名为 plugin.xml 的清单文件说明插件代码的位置，以及在哪个扩展点上进行的扩展等信息。编辑完扩展代码以及清单文件之后，就可以测试插件代码，并对插件进行封装和发布了。

在 Eclipse 下，有两种办法可以创建 Eclipse 插件。一是采用 JDT 来开发。另外一种更方便的方式是直接在插件开发环境 PDE 下开发。PDE 和 JDT 都是 Eclipse 的标准扩展。PDE 提供了帮助创建插件的向导，因而更方便一些。下面简单介绍一个在 PDE 下开发插件的小例子，以便能够对开发插件有个了解。具体怎样开发出高效便捷的插件需要做很多工作，掌握很多知识，需要去参考专门的书籍和帮助，这里就不介绍了。

现在来看看怎样在 PDE 环境下创建插件[102]。首先需要创建一个项目。在 Eclipse 上选择菜单 File→New→Other。这时 Eclipse 会弹出窗口让我们选择一个向导，如图 6-36 所示。我们选择 Plug-in Project 来创建一个插件项目，然后单击 Next 按钮。出现的下一屏会要求输入项目的名称，一般 PDE 项目的名称会直接作为插件的标识符，因此要起一个独一无二的名字。这里输入 com. example. hello。随后的一屏上会看到插件的名称就是 com. example. hello。在之后出现的“定义插件所需数据”窗口中选择默认选项。

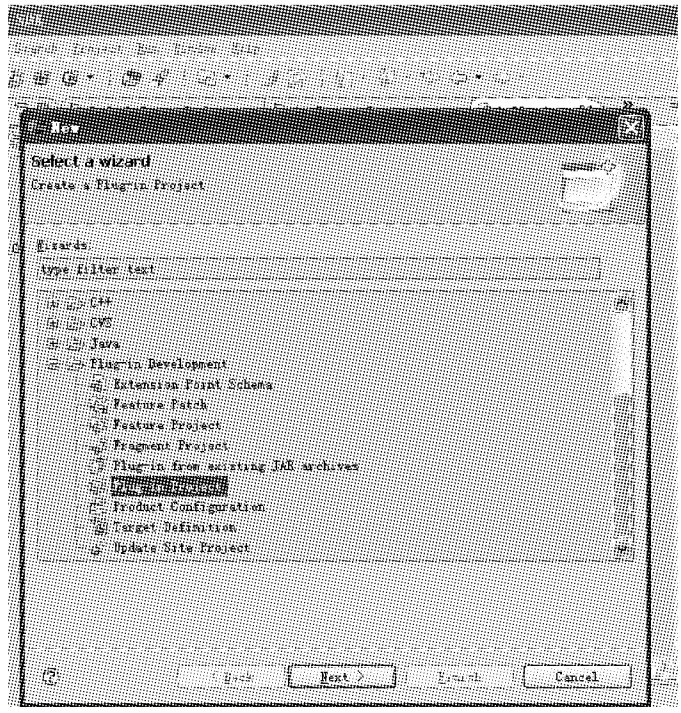


图 6-36　创建插件项目

当屏幕上出现如图 6-37 所示的界面时，选定屏幕上方的“用一个模板创建插件”复选框，Eclipse 就会用模板来创建插件，否则就会生成一个空的插件。这里选定复选框，自动生成插件。接着在界面上列出的可用模板中选择“Hello，world”模板。窗口右部说明了自动生成的插件的信息。从中可以看到，这个模板会创建一个动作集，向菜单条增加一个 Sample Menu 项和一个按钮，单击菜单项或者按钮都会调用相同的动作，即打开一个消息对话框。单击 finish 按钮，就完成了插件项目的创建。这时 Eclipse 将自动转到“插件开发”透视图，并显示出刚刚创建的插件项目的情况，如图 6-38 所示。

从图 6-38 的左侧导航窗格可以看到，平台自动为插件生成了一个 plugin. xml 文件以及相关的 Java 源文件。从图 6-38 中间编辑窗格中的扩展点描述中，可以看到这个自动生成的插件所用的扩展点是 org. eclipse. ui. actionSets。org. eclipse. ui. actionSets 是工作台的一个扩展点。它可以将菜单、菜单项和工具栏按钮添加到“工作台”窗口中的公共区域。这个自动生成的插件的作用就是扩展这个扩展点，在 Eclipse 菜单上增加一个菜单项，选择该项后可以弹出一个写有“Hello，world”的消息对话框。

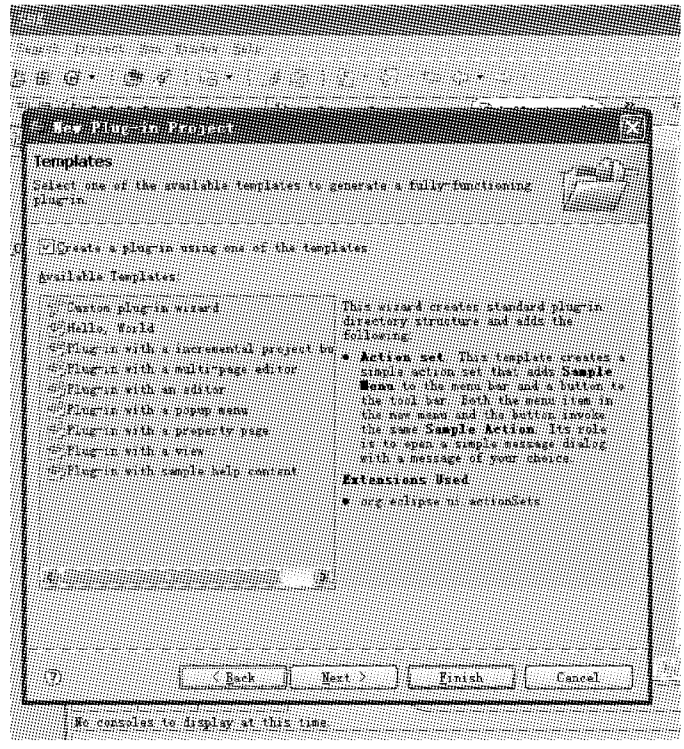


图 6-37　用模板创建插件

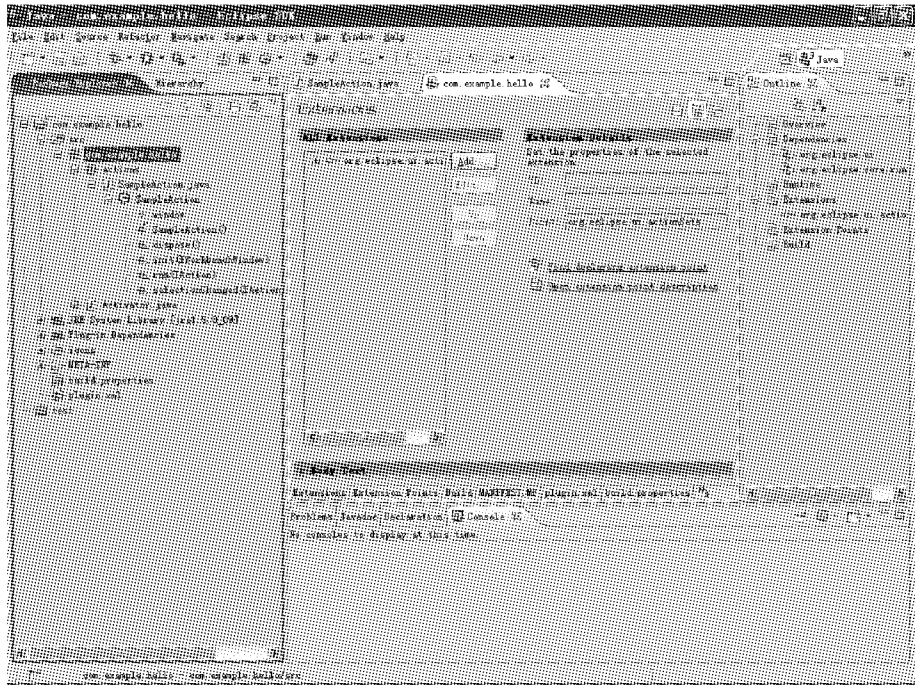


图 6-38　PDE 透视图

下面来测试一下这个插件项目是否能够工作。PDE 为插件提供了测试环境。单击工具栏上调试的箭头，打开 Run As 菜单项，可以看到 Eclipse Application 选项，如图 6-39 所示。选定这一项，Eclipse 就会启动一个新的工作台，一个一模一样的 Eclipse。但是在新的 Eclipse 平台下，新的插件就好像已经被注册了一样，可以运行它，如图 6-40 所示。

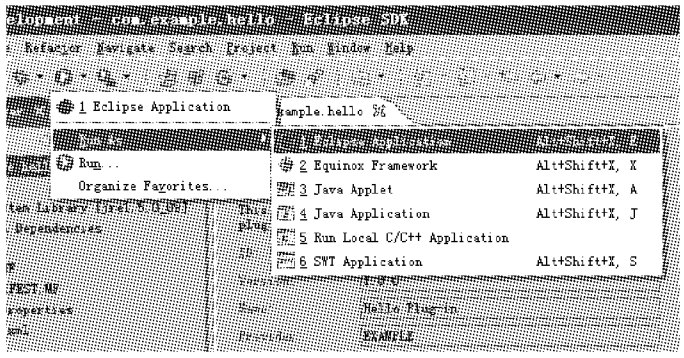


图 6-39　调试运行插件

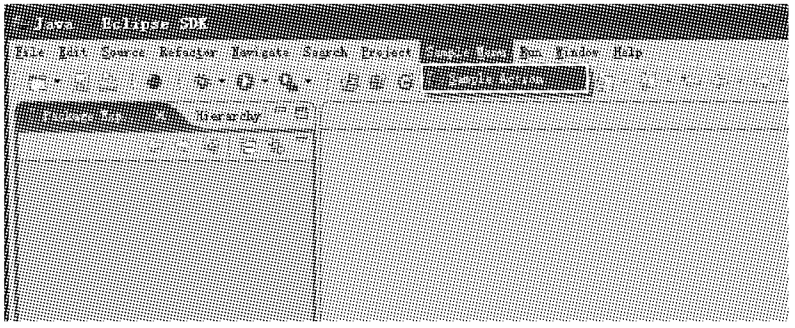


图 6-40　插件的测试

在图 6-40 中，可以看到菜单上多了一个 Sample Menu 项，工具栏上还多一个蓝色的按钮。这正是前面插件代码 Hello World 中添加的内容。单击 Sample Action，会出现如图 6-41 所示的窗口。这正是我们所做插件的功能。当然，这些代码都是通过“Hello，world”模板生成的。

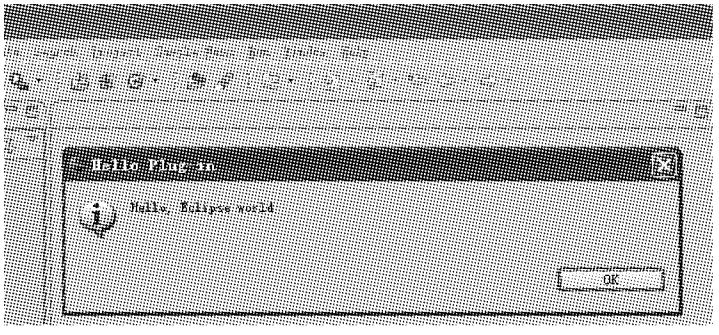


图 6-41　插件的运行结果

这里并没有编写代码来实现插件，因为这并不是本课程的内容。只是想通过这个例子让读者了解插件大致的开发过程。

小提示

开发 Eclipse 插件首先要确定扩展点，然后编写相关扩展代码，并提供 plugin.xml 说明文件，测试通过后，即可发布插件。

可以使用 Eclipse 的插件开发环境 PDE 开发 Eclipse 插件，也可以用其他 Java 开发环境。

5. 其他

最后再花一点儿时间简单介绍一些关于 Eclipse 的事情。不过，这并不是本节的重点，本节重点是希望读者能够通过 Eclipse 来了解开放平台的概念和机制。

Eclipse 是开放源代码的，它遵循 CPL 版权，就是和 Linux 一样。不过要说明的是，人们开发出来的工具插件并不都是使用 CPL 版权，所以使用插件时要注意。

Eclipse 最初是由 IBM 设计开发的，据说花费了 4000 万美元。后来 IBM 把它贡献给开源社区。现在 Eclipse 由一个叫做 Eclipse.org 的协会管理和指导正在进行中的开发工作。Eclipse.org 协会吸收了许多软件工具提供商，包括 Borland、Merant、Rational、RedHat、SuSE、TogetherSoft 和 QNX、Hewlett Packard、Fujitsu、Sybase 等。

其实，可移植、可扩展、开放源代码的框架并不是个新思想。但是，Eclipse 是一个成熟、成功的项目，它很好地实现和诠释了这种开放框架的思想。

最后我们用一个类比来结束本节。6.2.2 节中介绍的 VS2005 就好像是一个生产车间，开发软件需要的工具被制造成流水线机床，人们可以很轻松地在上面工作，不需要再自己在车间里找可用的工具。而且，人们可以根据需要，在预先设计好的几条流水线之间，选择适合自己的生产平台。而 Eclipse 则是一个更加开放的车间。在这个车间中有基本的流水线框架，用户可以根据自己的需要把各种插件插在流水线框架的插口上来搭建自己的流水线平台。用户还可以在这个框架上开发自己的工具插件。

6.2.4 更广泛的、更易使用的开发平台

在以前，应用软件都很复杂，需要很多专门的程序员花很多时间才能编写出来，然后这些应用软件会被使用很长时间，产生很大的经济效益或者社会效益。但是，随着互联网应用的不断发展，这种情况发生了一些变化。人们常常会有一些简单的应用需求，甚至这些应用很可能只用一次，以后就再不使用了。但是，会有很多人对这些简单的应用感兴趣。这些应用程序的开发也不是很复杂。在此背景下，应用软件开发平台出现了另外一个新趋势。那就是专业的程序员构造更通用、更简单易用的非专业开发平台，让那些非专业的计算机用户可以自己构造这些简单的应用程序。这些新平台下构造应用程序的过程很简单。有时只要用户做一些简单的配置，有时可能需要他们写一些简单的脚本。这样的应用及平台现在已经发展出很多，例如网上的个人空间、博客以及 RSS feeds 等。

6.3 开发平台中的可重用代码

应用软件开发平台实际上不光只有开发工具。它们通常还配备了丰富的可重用代码供程序员在写程序时使用。这样可以节省程序员的大量时间，而且能够增强程序的质量。这

就好像生产车间中除了为程序员准备了必要的工具、机床外，还提供了一些可以直接使用的零件以及可以被组合的独立部件。这样当程序中需要某种功能时，程序员就不用逐条语句地写代码了，直接引用相应的代码段，或者组装成形的部件就行了。也就是不用再从原材料开始加工了。这些可重用的代码或者软件构件有多种不同的形式，对应着不同的目的和效率。常见的几种方式是函数、类、组件以及框架。

6.3.1　函数

在使用面向过程的语言编写程序时，人们把常用的一些功能代码段写成函数形式，供程序员在写程序时重用。例如，把求余弦三角函数值的过程写成一个函数，命名为 cos()。当程序中需要求一个角度的余弦三角函数值时，就可以直接调用这个函数，而不用再写求解余弦三角函数值的过程了。人们把功能相似的函数放到一个文件。例如，把求正弦三角函数值的函数 sin()同函数 cos()放在一个文件中，而把负责输出的函数 printf()和输入的函数 scanf()放在一个文件中。这些文件被人们称为函数库。通常每个语言都有相应的函数库。例如，Turbo C 的数学库中放有与数学运算相关的函数，而它的标准输入输出库中放有向屏幕显示以及从键盘读入的函数。

函数库中的函数已经被编译成可执行的机器指令。程序员在程序中通过函数接口调用库函数，提供需要函数处理的数据。程序员写完的源程序编译后通常还必须有一个链接的过程才能最终变成可执行的程序。这个链接过程，就负责把程序中对库函数的调用与函数库中函数的具体实现挂上钩。

因为函数库是以编译好的二进制代码方式提供给程序员的，因此程序员不能对其做任何修改。函数库提供的接口一般是语言相关的，因此必须在所要求语言的源程序中才能重用。

6.3.2　类

在采用面向对象的程序设计模式后，人们把程序设计过程变成了定义对象类别，即定义一些物体种类，以及生产这些类别的对象/物体实例的过程。不过，重用代码的思想仍然被延续。这时人们重用的不再是过程，而是定义好的对象/物体。在面向对象的程序设计过程中人们把对象/物体的类别定义称为“类”。这些类的定义就好像是物体的设计说明。比如，名字为“门”的类就定义了一类物体“门”。定义中包括门的属性和行为，如形状是矩形的，材质可以是木头的、铁的，它可以绕着一个轴转动等。又比如，人们在程序中会通过文件来获取或者保存信息。因此，人们就定义一个“文件”类。这个类的属性有文件内容、是否可读、可执行等，它的行为有打开、读取、写入等。当人们在程序中需要这些对象/物体时，就会实例化这些类。也就是让系统按照这些类的定义，即物体的说明书，生产一个物体，如按照“门”的定义生产一扇门。很多程序会有相同的一些类的需求，如很多应用软件都会需要“文件”这样的物体来获取或保存信息。因此，人们把一些可能重用的类的定义放在一些文件中，这样程序员再需要类似的物体时，就不需要重新定义了。这些文件被称为类库。

类定义的是一类事物，而不是一个解题过程，这是与函数的区别。此外，类可以被继承、派生。也就是人们可以在原来类的基础上进行新的加工。就好像是，当某个人建一座房子

需要一扇门时，他从类库中取出“门”类的定义。但是，他对这个门的设计并不满意。因此，他在这个类的基础上重新设计了一个新的门，如把门的形状从矩形改称圆形，也就是更改了原来类的属性。

同样，针对每种面向对象的程序设计语言，人们都设计和开发了很多的类库。例如，Java 的类库就有实用工具类库、语言类库、输入输出类库、网络类库、数学类库、文本类库、安全类库等涉及各个方面的支持。而且类库的数量还在不断地增加。每个类库中都定义了很多的类。例如，实用工具类库中定义的类有日期类、日历类、产生各种类型随机数的随机数类，以及堆栈、向量、哈希表等。其他语言的类库也与此类似。

小提示

应用软件开发平台中包含很多形式的可重用代码。常见的形式有函数、类、组件以及框架。

函数描述了一个个解题过程，以可执行代码形式存在。应用程序中可以直接调用函数，编译或链接程序将调用链接到可执行代码。

面向对象的程序中以类的形式重用代码。应用程序可以直接生成现有类的实例，也可以对类进行扩充和修改，创建新的类。

6.3.3 组件

除了函数库和类库之外，人们还设计了另外一种软件重用的方式，叫做组件。人们希望能够用一些小程序搭建成更大的应用软件。这些可以被搭建组装在一起的程序就被称为组件。一个组件通常能够完成某个特定的功能。组件可以在称为容器的环境中被组装起来。这样，构建应用软件的过程就和搭积木差不多了。不同于函数，组件在特定的环境下能够运行，也就是说，它是在某种环境下可以运行的独立程序。但组件不能是普通的程序，它们必须具有能够被装配到一起的一些特性。比如说，它们至少要具有能够和容器通信的机制[56]。

实现组件的技术有很多，因而组件的表现形式也有很多种。它可能是一个特殊的类、一些类的组合、一个包或者一个可执行文件甚至是一个图像文件。常见的支持组件的技术有微软的 COM 和 ActiveX，以及 Sun 公司的 JavaBean，另外 Delphi 也支持组件。

人们采用这些组件技术，编写各种组件，然后把它发布出去，供其他程序员使用。集成开发环境加入这些制作组件的技术，并提供在程序中装配组件的手段。这样程序员就可以直接使用这些组件来组建应用程序了。

小提示

组件是在某种环境下运行的独立程序，它可以在容器中组装来构建应用软件。

支持组件的技术有很多，如微软的 COM 和 ActiveX，以及 Sun 公司的 JavaBean。

6.3.4 框架

从功能重用的需求出发，人们发明了函数、类以及组件等代码重用方式。慢慢人们发现

在程序设计开发中其实还有一类内容可以重用。那就是应用程序的架构。很多应用程序在架构上是相似的或者就是一样的。例如，各种 Web 程序，除了基本业务逻辑上的区别外，它们其实在架构上非常相似。为此，人们对应用程序的架构进行研究，抽象出一些高效的架构提供给程序员在设计应用程序时使用。这些架构也被称为框架。这些框架定义了某一类应用程序应该是怎样的层次构成、大概有哪些组件，以及组件之间应该怎样协同等。这些可重用的框架也通过开发环境提供给程序员。这就好像应用软件开发平台这个生产车间，不但有工具、机床、零部件以及组件，又增加了一些产品的框架。车间里的生产者选择合适的框架，然后在里面添加上需要的组件，另外完成自己特定的业务逻辑，可能还需要增加一些语句作为粘合剂，就可以快速地搭建/生产出需要的产品了。

目前在构建 Web 应用程序时常用的一个应用框架叫做 Struts。Struts 框架认为一个 Web 程序应该分为模型(model)、视图(view)、控制器(controller)三大部分。这种三大部分的构成是目前 Web 应用框架的一种流行模式，被称为 MVC 模式。其中，模型层负责业务流程的处理，即改变业务状态。视图层负责向用户展现处理的结果，并接收用户的输入，把它交给控制器。控制器负责处理用户的请求，根据请求调派模型层和视图层进行相应的处理和显示。Struts 只是采用 MVC 模式的架构之一。在 Struts 框架下，模型层的功能由 JavaBean 或者 EJB(即企业级 JavaBean)技术实现的组件完成，控制器则通过 Servlet 技术实现，而视图部分则采用 JSP 技术实现。图 6-42 描述了 Struts 框架实现的 MVC 模式[58]。

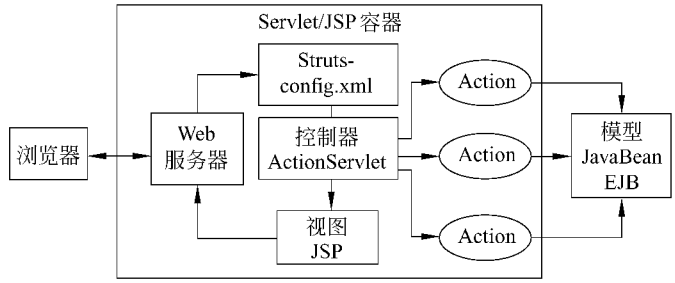


图 6-42　Struts 实现的 MVC 架构

为了让程序员能够使用这个框架，人们开发了 Struts 工具集。里面包含了按照 Struts 架构搭建程序所需的相互协作的类(组件)、Serlvet 以及 JSP 标签库。因此，应用程序开发环境中如果包含这个工具集，那么，程序员就可以采用这个框架，使用相应的组件来快速构造 Web 应用程序了。例如，在集成开发环境 Eclipse 中开发 Java 项目时，如果安装了 Struts 插件，那么程序员就可以使用 Struts 架构来开发应用程序了。

除了 Struts 外，还有很多的应用程序框架，如 Hibernate、Spring 等。前面提到的这些框架都是开源的、免费的。人们可以免费得到和使用这些框架的工具集，把它加入到自己的应用软件开发环境中，用它们来构造自己的应用软件。

小提示

框架是应用程序的软件架构。
Struts、Hibernate、Spring 是目前常见的一些开源框架。

复习题

1. 程序员对于应用软件开发平台有哪些需求？
2. C、C++、Java 语言各有什么特点？分别适用于开发什么样的应用软件？
3. 试着找出 Linux 操作系统下，适合编辑 C 语言程序和 Java 程序的编辑器，以及编译这两类源程序的编译器的名字。
4. C++ 常见的集成开发环境有哪些？Java 呢？
5. Eclipse 采用怎样的机制实现可扩展性？
6. 什么是函数？
7. 什么是类？重用类比重用函数有什么好处？
8. 什么是组件？常见的组件技术有哪些？
9. 什么是框架？目前常见的框架有哪些？

讨论

1. 请查阅资料，考察 Visual Studio 是采用怎样的机制支持多种不同的程序设计语言的？
2. 单独的开发工具和集成开发环境各有什么优缺点？

实验

1. 试着用 Eclipse 编写一个简单的 C++ 程序和一个 Java 程序。
2. 询问有经验的程序员或者上网查询，考察程序员目前正在使用的应用软件开发平台。调查这些平台各有什么特点？程序员为什么选择这样的开发平台？

第四篇　计算机网络平台

网络使计算机系统之间能够交换信息，使每个系统的能力得以增强。现在的计算机几乎都连接到网络上，很少能看到游离在网络之外的系统了，除非是特殊的安全要求。就像当今的世界，各个国家都通过政治、经济等方式联系在一起，很少再有与外界没有任何关联的国家了。因此，联网已经作为计算机的基本功能被内置在系统中。对用户来说，只要打开计算机，就可以轻松进入网络世界。网络也是一个平台，每台计算机都是网络的一部分。这一篇我们就来看看在网络上的计算机系统可以为我们做什么，我们要怎样才能使用它。关于计算机网络，还有很多很多知识，如网络是怎样搭建的、信息是怎样在网络中传递的等。这些内容会在另外一门单独的课程中学习。计算机网络已经是一门涉及面很广、内容很多的学问。这一篇只从用户的平台角度来了解计算机网络。

第 7 章　网络平台的服务

计算机具有联网功能，人们因此可以多做很多事情。加入了网络平台的计算机系统为人们开辟了通向世界的道路。例如，可以给朋友或者老师发送电子邮件通知事情或者请教问题。电子邮件比起传统的邮件真是又快又省钱。人们可以通过像 QQ 这样的软件和朋友聊天，甚至可以与不认识的远在千里万里之外的陌生人讨论早上吃点儿什么的问题。可以上万维网浏览看看今天全世界都发生了什么重大事情，也可以了解像一户人家的小狗生了几个小狗宝宝这样的小事情。还可以通过像电驴这样的软件找到并下载喜欢的歌曲或者电影看看。当然也可以在线看电影或者电视。还可以在网上发表自己关于某件事情的看法。事情可能是社区该不该在大门口设置一个垃圾箱，也可能是为什么现在的原油价格涨得如此厉害。人们还可以在网上买书或者预定今天的午餐。总之，网络极大地拓宽了人们的视野，使人们真正地做到足不出户，尽知天下事；网络也极大地方便了人们的生活；并为人们的生活增添了无数的色彩和内容。

其实网络带给人们的还不止这些。网络可以让管理机构共享信息。这给管理与服务提供了深刻变革的支持。例如，全国的公安局可以共享身份证信息，这就为我国的户籍改革提供了很大的便利条件，而且需要验证身份的组织和个人都可以通过网络到公安局的系统上核实某个人的身份。网络还可以让工厂、公司迅速地交流供求信息。这大大地提高了生产率。网络还可以让程序员们及时沟通、交流开发中的问题和进度。因此，印度很多人可以不用离开国家而和美国人合作开发项目，赚得了大笔的美元。网络使技术支持人员不用再到现场，通过远程登录就能把客户的问题解决了。网络带给我们的变化可以说数不胜数。网络带给我们的变革就像电和计算机的出现曾经和正在带给我们的一样那么深刻和深远。

在计算机通过网络互联起来之后，计算机也发生了一些变化。用户的计算机上都增加了获取前面提到的这些网络功能的能力(软件)。另外，一些计算机分化出来，它们被放置在网络中，专门为网络中的用户计算机提供服务。有一些计算机专门负责为其他的计算机周转数据，如将北京小明计算机发出的问候转交给上海小芳的计算机。这些计算机被称为网络设备(往往不称为计算机)。它们常见的名字有路由器、交换机，以及网桥、网关等(这些设备的有关知识将在后面章节以及计算机网络课程中介绍)。另外，还有一些计算机专门负责为用户计算机提供信息服务，如提供天气预报、新闻等信息让人们浏览，或者提供歌曲、电子书籍让人们下载。这些计算机被称为服务器。其实服务器原是指这些计算机上运行的提供服务的软件。但是，装有这些软件的计算机通常也不做别的用途，而且硬件配置也比较高，要能同时应付很多人的访问。因此，后来这类计算机也常被人们称为服务器。

要想从服务器上获得信息，用户计算机必须与提供信息的服务器软件交流。因此，用户计算机必须有专门能和服务器交流的软件，通常被称为客户端软件。例如，浏览器 IE 就是这类软件中的一个。客户端和服务器要想能够交流，必须事先有些约定。例如，客户端给服务器发送字符 hello 表示问候，服务器必须明白这件事，不能认为客户端是在说它“该死”。因此，客户端和服务器软件都必须由同一个人或组织开发。如果不是这样，至少要合作开发

才行。客户端和服务器交流的有关约定被称为协议。为了解除客户端和服务器必须捆绑设计开发的限制，人们把一些常见服务的协议标准化。这样客户端和服务器都可以单独开发了，只要它们遵循相同的协议就能够交流。例如，Web 服务（马上就介绍什么是 Web 服务）的客户端和服务器之间使用的协议就被标准化了，称为 HTTP 协议。Web 服务器的客户端软件就是常听说的浏览器。目前世界上就有很多种不同的浏览器软件，如 IE、Mozilia、Netscape 等。而 Web 服务器也有很多种，比如 Windows 下的 IIS 和 Linux 系统中常用的 Apache。这些浏览器和 Web 服务器相互都能交流，因为它们都使用 HTTP 协议。

实际上服务器这个词的确切定义并不是指提供信息的软件，而是另外一个概念。在网络中，如果一个进程想与另外一台计算机上的某个进程通信，那么对方进程必须存在才行，即对方的程序必须在另外一台计算机上运行起来。可是两台计算机上的程序怎么知道应该什么时候运行呢？这就像一个约会。小明到小芳家，发现她没在家，小明就离开了。当小芳到小明家去找他时，发现小明也没有在家，然后小芳也就回来了。如果两个人总是这样，那么他们就很难碰到面。要想解决这个问题，最简单的办法就是一个人不动，始终在家等着。这样，第二个人来的时候就能够见面了。网络上需要通信的程序也一样。如果一个程序启动起来就一直保持运行、等待其他进程与其联系，而不是看看与其通信的进程不存在就退出，那么通信就没有问题了。因此，在进行网络程序设计时通常采用这样的模式：通信双方程序总是有一方先运行起来，等待另外一方的联系，通常是无限期地等待。而另外一方的程序则可以在需要通信或服务时才启动起来。等待的一方就被称为服务器，而需要通信时才被启动起来的一端则被称为客户端。这种应用软件的体系结构被称为客户/服务器结构，也就是人们常说的 C/S 结构，是网络应用最基本的概念。通常等待的一方都是提供信息的软件，所以，我们才会有服务器就是提供服务的软件的印象。但是，有时候等待的一方也可能是获取信息的软件，在软件结构的概念中，它仍然是服务器。

在涉及网络应用程序时，我们还经常听到 B/S 结构和 P2P 结构。这是两种新的软件体系结构。不过它们可以看成是 C/S 结构的变形。

所谓 B/S 结构，即浏览器/服务器（Browser/Server）结构。这种结构的应用软件通常不用再编写客户端。它的客户端软件就是浏览器。浏览器只能读懂 HTTP 协议，因此 B/S 结构应用软件的服务器端必须通过 HTTP 协议与客户端的浏览器通信。所以，B/S 结构实际上还是 C/S 结构，只不过客户端以及通信协议被固定下来了。

在 P2P 结构中，参与应用活动的计算机本身没有服务器和客户端之分，因此大家看起来都是平等的。在实现上，每台计算机实际上都运行有相同的两部分软件，一部分就相当于是服务器软件，另一部分就是客户端软件。当两台计算机之间发生具体的活动（例如下载文件）时，也是一台扮演服务器，另一台扮演客户端的角色。只不过当计算机 A 在为计算机 B 提供文件服务的同时，它也可能通过客户端软件正从计算机 C 获得文件服务。这样，从宏观上看，大家都是“对等的”（peer-to-peer）。

小提示

网络应用通过在多台计算机上运行的一组程序来提供服务。

客户/服务器结构是网络应用常见的体系结构。一直运行、被动等待的一方被称为服务器，在需要时才主动发起通信的一方被称为客户端。

另外常见的软件结构还有浏览器/服务器和 P2P 模式。严格地说，它们都是客户/服务器模式的特殊形式。

下面具体来介绍一些常见的网络服务，来了解一下它们到底是怎么回事、提供什么服务、是怎样提供的。这些服务包括 Web 服务、电子邮件服务以及 FTP 服务。

7.1　Web 服务

7.1.1　万维网概述

1. 万维网

Web 是 Internet 上最常用的一种服务。可以说它对 Internet 的普及起了非常大的推动作用。正是 Web 提供的丰富多彩的内容，让越来越多的人成为 Internet 的用户。

Web 是万维网(World Wide Web，WWW)的简称。万维网这个网络和我们平时说的 Internet 这个网络的概念是不同的。Internet 是一个由主机、传输线路、网络设备构成的网络。Internet 在物理上为通信铺平了道路，同时在软件上也达成了信息传递的通路。在 Internet 这个平台上，各种各样的应用都可以运行起来。Internet 这个网络在一套协议的控制下运行。而万维网是在 Internet 这个物理网络上由一组分布式软件提供的关于内容的服务。这些位于不同计算机上的内容之间相互链接，构成了一个内容的网络。万维网提供的内容包括文字、图像、音频、视频等各种形式的信息。

说了这么多，我们还是来看看 Web 服务到底是什么吧。要得到 Web 服务，我们必须运行一个称为浏览器的软件。目前在 Windows 操作系统上最常用的浏览器是 IE，而 Linux 下常用的是 Mozilia、Netscape 等。浏览器运行起来之后，我们就可以在其界面上一个称为地址栏的文本框中输入一串称为网页地址的字符，如 http://www.bjut.edu.cn。回车之后，等上若干时间(可能很短，也可能很长)，我们就能够在浏览器的窗口中看到一些信息。它们可能是图片、文字，也可能是视频，或者别的什么。我们看到的这些内容并不是在我们自己的计算机上，而是浏览器从称为 Web 服务器的计算机上取回来的。每个想把信息发布给别人看的用户，都可以在自己的计算机上运行一个 Web 服务器软件，然后把自己要发布的内容通过 Web 服务器发布出来。浏览器就是和这些 Web 服务器联系获得网页的。而我们在浏览器的地址栏中输入的字符串就指向这些 Web 服务器所在的主机以及这些内容在该主机上的位置和名称。

当用户在浏览器显示的内容上移动鼠标时，鼠标的形状在某些位置会发生变化。在这些位置按下鼠标后，浏览器就会为用户显示出一些新的内容。这被称为超链接。实际上新链接到的内容可能位于另外一台计算机上，可能距离我们千山万水。全世界的内容就通过这样的相互链接而联系在一起，如图 7-1 所示。Web 就这样在用户面前展现出一个由各种内容构成的、联系错综复杂的网络。这就是万维网的意思。

万维网并不是研究计算机的人想出来的。它是一个叫做 Tim Berners-Lee 的物理学家的想法。这件事情还要从欧洲原子能研究中心说起。这个中心有几台加速器——很贵的仪器。很多国家的科学研究组利用这些加速器进行粒子物理的研究。这些来自不同国家和不同研究组的科学家非常希望能够通过共享彼此不断更新的报告、计划等文档来互相合作。于是 Tim Berners-Lee 借用了以前的一个“超文本”的概念，提出了链接文档网的想法。他建议每个人负责维护自己的文件，然后专门设置一台服务器，服务器上维护一个目录，目录

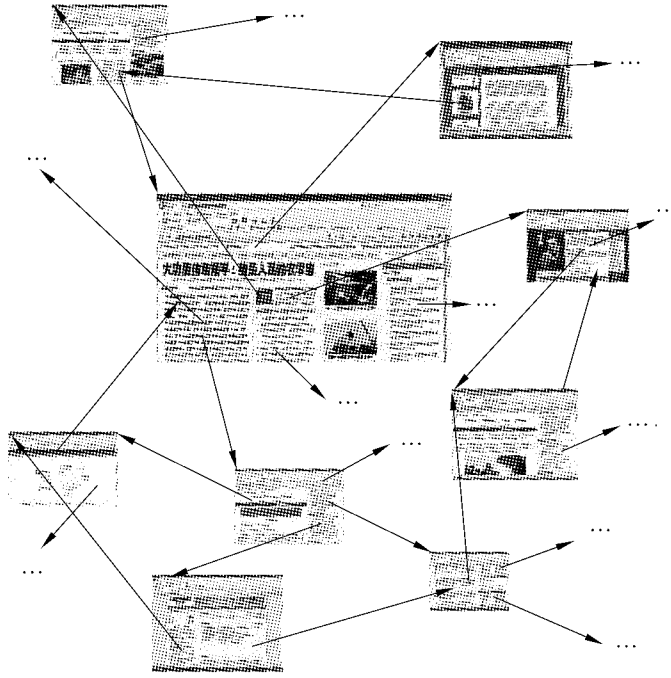


图 7-1　万维网示意图

通过链接指向每个人的文件。这样大家就可以连接服务器，然后通过服务器的目录链接到每个人的文档上了。而每个文档的主人负责保证给别人呈现的都是最新的文档。后来链接文档的能力范围被进一步扩大。不但服务器上的目录能够指向其他计算机上的文档，而且其他文档中的某些文字也可以含有指向其他文档的链接。这样，位于不同计算机上的、各种各样的文档就被关联起来，构成了一个错综复杂的、相互关联的文档(内容)的网络。

在 Web 服务中，浏览器从 Web 服务器获得并显示出来的内容被称为 web page，即网页。这些网页中能够链接到其他网页的文字被称为超链接。这些文字在网页上的显示有与众不同的标记，如有下划线或者不同的颜色。通常，当我们把鼠标移动到这些超链接上时，屏幕上鼠标的形状就会发生改变。这些超链接可以将用户引入到另外一个网页上。

2. URL 统一资源定位符

这里我们发现第一个问题。既然是引向另外一个网页，万维网上有那么多的网页，怎么来标识和区分它们呢？只有把它们区分开，我们才能有办法指出需要的是此网页而非彼网页。也就是说必须有个统一的命名规则。这样才能在成千上万数不清的网页中定位某个网页。除了要考虑区分功能外，命名规则还要能够对定位网页有所帮助。不然，在浩如烟海的网页中按照某个名字一个一个去查找也太麻烦了。虽然说起来有点儿复杂，但实际做起来并不复杂。人们用了一个很简单的办法就把这个问题解决了。因为每个网页是作为文件存放在 Web 服务器主机上的；主机上的文件系统通过目录和文件名机制保证能够在这台计算机上唯一定位到某个 Web 页；而在 Internet 网络中，每台主机都有一个独一无二的主机名。把这两项结合起来实际上就可以在网络中唯一标识一个网页了，并且还能给出网页的位置。因此，Web 中使用如下格式的字符串来唯一地标识一个网页：

```
http://www.pku.edu.cn/index.html
```

这种格式的字符串被称为统一资源定位符(Uniform Resource Locator,URL)。在这个字符串中,除了含有前面提到的主机名和目录/文件名外,还多了一部分内容,即开头部分 http://。它表示浏览器从服务器上获得网页时要使用的协议。实际上,我们可能都见过 URL 了,只不过我们常把它称为网址。每个 URL 由三部分构成。第一部分,如 http://,表示使用的协议。即获得本 URL 指向的资源需要采用的协议。浏览器和 Web 服务器通信时使用 HTTP 协议(后面还会介绍)。除了 http://外,有时候我们还会看到 file://、ftp://等。file://表示浏览器要通过文件系统访问本机的文件。而 ftp://则表示浏览器需要使用 ftp 协议和服务器进行联系。当然,这时连接的服务器也不再叫做 Web 服务器,而叫做 ftp 服务器了。相应的服务也就不是 Web 服务了。因此,可以看出浏览器不但可以作为 Web 服务的客户端,也可以作为其他像 ftp 服务的客户端。我们会在后面介绍 ftp 服务。URL 的中间部分,如 www. pku. edu. cn,表示资源/网页所在主机的名字,也可以是 IP 地址(每个主机在 Internet 网络中一个独一无二的标号)。最后一部分,如 index. html,表示资源在主机上的位置和名字。即 Web 网页文件在 Web 服务器根目录下的路径和文件名。有了 URL,浏览器就可以很容易地找到网页了。它通过 URL 中的主机名先找到服务器主机,然后再通过 URL 中指定的协议与服务器通信。联系上服务器之后,浏览器就把 URL 中的最后一部分交给它,指出需要的网页在服务器上的路径和名称。

小提示

万维网 Web 是一个由各种网页相互链接而成的内容网络。
网页中能够链接到其他网页的文字称为超链接。
Web 用统一资源定位符 URL 标识和定位网页。

3. HTML 语言

解决了网页的标识和定位问题,接下来我们还有一个困惑。既然网页中的某些文本是可以链接到其他网页的,那么怎样表示某些文本是超链接,而别的不是呢? 而且怎样把这些文本和一个 URL 关联起来呢? 这个任务就交给了超文本标记语言 HTML。

HTML 是一种由各种标记符号构成的语言。它是一种人类可阅读的语言。人们把要在网页中显示的内容存放 HTML 文件中。在这个文件中,除了正常要显示的文字内容之外,还在文字的周围增加了一些 HTML 定义的标记来表示它们应该显示的格式或者其他要求等。就像以前人们在要出版的手稿上用一些特殊的符号标记某部分应该用粗体、某部分应该用 5 号字一样。

图 7-2 是一个简单的 HTML 文件的内容。图 7-3 是这个 HTML 文件所表示的网页在浏览器中显示的结果。图 7-2 中的行,如:

```
<p><font face="隶书">你好!</font></p>
```

定义了“你好!”这句话的显示格式。符号 和 规定中间的内容,即“你好!”,应该是一个段落。符号和则定义这段内容显示时使用的字体是隶书。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>示例 </title>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>

<body>
<h1><strong>这是一个新网页。 </strong></h1>
<p><font face="隶书 ">你好！ </font></p>
<p><font face="楷体 _GB2312">点击 <a href="http://www.bjut.edu.cn">这里 </a>可以看到北京
工业大学的主页。 </font></p>
</body>
</html>
```

图 7-2　简单的 HTML 文件

图 7-3　图 7-2 文件的显示效果

除了格式外，HTML 语言中含有用来标记超链接的符号。例如：

```
<a href="http://www.bjut.edu.cn">这里</a>
```

里面的符号标记＜a＞和＜/a＞表示中间的文本，即“这里”，是一个超链接。href＝"http://www.bjut.edu.cn"表示这个超链接所链接网页的 URL。这个 URL 中主机域名后面没有具体的文件名，通常表示默认的文件 index.html。这一行会使网页上显示带有下划线的“这里”这两个字。当把鼠标移动到这两个字上并单击时，如果可能，浏览器上就会显示出 http://www.bjut.edu.cn/index.html(实际上是 index.jsp)网页的内容。

图 7-2 中的标记＜html＞和＜/html＞之间定义了这是一个 HTTP(HyperText Transfer Protocol，超文本传输协议)文档。网页中所有用 HTML 标记的内容都在这两个标记之间。

从这里我们看出，在 HTML 中，每个规定都有一个开始的标记和一个结束的标记。通常它们是用同一个字符串来表示，并且用尖括号括起来，只是结束标记中字符串前面多一个

斜杠“/”。不同的字符串表示不同的规则，如 font 表示对字体的规定，a 表示对超链接的规定。在 HTML 语言中并不区分大小写，因此，font 和 FONT 是一样的。这样一对一对的标记，在 HTML 语言中被称为标签，如＜a＞＜/a＞，也叫做格式化命令。有些标签带有参数，这些参数被称为属性，如＜FONT＞中的 face。这些标签都被标准化，因此，任何人/浏览器看到这些标签都会给出相同的解释。关于 HTML 的内容还有很多。如果你感兴趣可以去找相关的书来看，市面上这类的书非常多。

4. 浏览器

Web 网页以 HTML 文件的形式存储在网络中的各个 Web 服务器上。当用户想要看某个网页时，他需要借助 Web 服务的客户端——浏览器的力量。浏览器和 Web 服务器联系，取回用户需要的网页并把它显示出来。HTTP 定义了浏览器和 Web 服务器之间的通信规则，如浏览器应该怎样与 Web 服务器建立联系、浏览器需要某个网页时应该发送什么样的消息、当 Web 服务器收到某种格式的消息时应该做什么样的应答等。

HTTP 协议定义了两种格式的报文，即浏览器发送给 Web 服务器的请求报文和 Web 服务器发回来的响应报文。HTTP 报文都以 ASCII 码字符的形式表现，因此我们都可以直接看懂 HTTP 报文中的内容。图 7-4 是一个 HTTP 的请求报文。通常浏览器发送给 Web 服务器的报文都是类似形式。这个报文的第一行称为请求行。请求行中按照要求应该按顺序给出 HTTP 命令、URL 以及 HTTP 协议的版本号。图 7-4 中的 GET 就是命令，它表示向 Web 服务器请求获取资源。HTTP 协议还定义了一些其他命令，如 POST。HTTP 协议最初的版本是 1.0，后来又出现了 1.1。这两个版本有什么区别，你可以自己去查资料。在请求行后，是一些被称为头部的字段。每个头部字段有自己的名字以及值。头部字段的名称和值由 HTTP 协议定义。字段的名字和值由冒号分开。这些头部字段用来在浏览器和 Web 服务器之间交换一些信息。例如，图 7-4 中的 Accept-language 即是一个头部字段名，在该行冒号后面的 fr 是该头部字段的值。浏览器通过这个头部字段告诉服务器自己可以接受的语言是法语。这样如果所请求的网页有多种语言版本，那么服务器可以选择把法语版的网页发送给浏览器。请求报文通常只有请求行和头部字段两部分内容，不过有时也会带有少量的数据，如用户在网页上输入的要求查询的关键字[59]。

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr

(更多的回车，换行)
```

图 7-4　HTTP 请求报文

响应报文和请求报文类似。同样含有各种头部字段。所不同的是，响应报文的第一行称为状态行。它表明 Web 服务器对浏览器发来的请求的处理结果，如没找到网页或者网页已经移动等。响应报文中多会带上浏览器索要的数据，如网页的 HTML 文件等。

除了能够按照 HTTP 协议与 Web 服务器联系之外，浏览器另外一个重要的功能就是要能够看懂 HTML 语言。当它从 Web 服务器取得一个网页的 HTML 文件后，它就要在自己的窗口上显示这个网页。例如，当看到文件中的

```
<a href="http://www.bjut.edu.cn">这里</a>
```

时，浏览器要在自己的窗口显示带有下划线的“这里”两个字，并且当用户的鼠标单击这两个字时，浏览器要使用 HTTP 协议联系主机 www.bjut.edu.cn 上的 Web 服务器，并请求取

回 index.html 文件，取回文件后再按照文件中的内容和 HTML 格式化命令显示出来。所以说，浏览器是一个 HTML 解释器。

小提示

HTML 是一种标记语言，描述如何格式化文档。
Web 页用 HTML 语言描写，浏览器是一个 HTML 解释器。
浏览器和 Web 服务器使用 HTTP 协议进行通信。

实际上，浏览器的功能远不止这些。每个网页上除了文本内容之外，还会有图片、视频以及各种不同格式的文档，如 PDF、Word 格式的文档等。浏览器能够根据这些文档格式调用相应的外部辅助程序，如 Adobe Reader、Word，来显示这些文档。对于某些格式，浏览器还会使用一些专门为它们设置的插件来实现显示。

插件与辅助程序的区别是，插件是个代码模块，在使用之前必须安装。安装之后，它成为浏览器的扩展模块。当浏览器需要用到某个插件时，它从磁盘的特定目录读出模块并执行。插件的模块是在浏览器内部运行。执行完毕，当浏览器从当前页面转入其他页面后，插件模块从内存中删除。如果下次再用到，就再重新加载。

而辅助程序则是独立于浏览器的程序。它作为独立进程来运行。有时候这些程序会为浏览器提供调用它们的插件。一般浏览器只是启动这些外部程序，并把需要它们处理的临时文件名交给它们。辅助程序通常要做的事情是按照它们的格式显示这些临时文件的内容。

可是，当浏览器遇到网页中一个不是 HTML 文件的对象时（网页上的内容被称为对象），怎么知道应该去调用哪个插件模块，或者启动哪个辅助程序呢？浏览器能够做好这件事情，主要依赖两方面的帮助。首先，在浏览器通过 HTTP 协议从 Web 服务器获得要显示的对象时，HTTP 协议规定在消息头中可以给出消息体中所带对象的类型。第二，浏览器手头有一张表格，表格中记录着各种类型对象及其相应的查看器。这张表格在插件以及辅助程序安装时填写。这样，当浏览器获得一个自己并不支持的类型对象时，它便去查看这个表格，然后去调用相应的代码模块或者启动辅助程序。通过这种方式，浏览器可以很容易地被扩展支持越来越多类型的对象。

除了根据对象的类型启动相应的辅助程序或者插件外，浏览器还可以运行网页中的一些代码来完成一些操作。这需要在 HTML 文件中嵌入一些脚本。最常见的脚本是用 JavaScript 脚本语言编写的。JavaScript 是 Netscape 公司引进了 Sun 公司有关 Java 的程序概念，改造了自己原来的一种称为 LiveScript 的脚本语言形成的。JavaScript 脚本可以嵌入在 HTML 文件中。它主要用来实现与用户的交互。例如，显示一个计算年龄的网页。用 JavaScript 可以在网页上生成一个要求用户输入出生日期的文本框。当用户输入生日之后，JavaScript 会计算出用户的年龄并显示出来。这里要强调的是，JavaScript 是写在网页的 HTML 文件中，是浏览器从 Web 服务器获得的。但是，这些脚本是由浏览器在用户的计算机上执行的。

小提示

浏览器通过调用外部辅助程序或者执行插件来支持各种不同格式的对象。

插件是代码模块，使用之前必须安装，安装后成为浏览器的扩展模块，在浏览器内部运行。

辅助程序独立于浏览器，作为独立进程运行。浏览器在需要时启动辅助程序，并把要处理的临时文件名交给它们。

浏览器通过注册表格了解文件格式及其处理程序的对应关系。

JavaScript 脚本嵌入在 HTML 文件中，在用户端由浏览器执行。

5. 动态网页

开始时，Web 的主要功能只是信息共享。因此，共享的信息以文件的形式存放在 Web 服务器上。当用户需要时，浏览器就把它取来并显示。Web 的普及使人们对它的期望值越来越高，人们希望它能够完成更多的任务。因而 Web 再只提供给浏览器那些事先准备好、存放在磁盘上的文件，已经不能满足要求了。比如说，现在人们使用最多的搜索引擎，就是以 Web 页面的形式提供给用户的。用户在搜索引擎，如 e. pku. edu. cn 的 Web 页上的文本框中输入要查询的信息，那么浏览器从服务器所获得网页的内容应该是搜索引擎根据用户输入的关键词找到的相关网页的信息。关键词不同，显示的内容也不同。也就是说，网页的内容不再是静止不变的，而是根据具体情况动态生成的。因此，这种网页被称为动态网页。相对于此，那些以文件形式存放在 Web 服务器上的网页则被称为静态网页。

对于动态网页，当 Web 服务器收到浏览器请求网页的报文时，就不能像静态网页一样到磁盘上把文件取出，发送给浏览器就算完成任务。Web 服务器必须想办法根据浏览器传送过来的条件生成相应的网页。通常 Web 服务器要运行一些小程序来完成这个任务。这些小程序包括 CGI 脚本、内嵌在 HTML 中的 PHP 脚本、ASP 脚本以及 JSP 脚本等。这些名词一定把你搞晕了吧？别着急，我们一个一个解释。

CGI(Common Gateway Interface，公共网关接口)是一个标准化的接口，它定义了 Web 服务器与后端程序或者脚本通信的规则。这些后端程序/脚本从 Web 服务器获得浏览器传送过来的信息(通常是用户输入的信息)，然后根据这些消息生成 HTML 网页返回给 Web 服务器。一般这些后端程序/脚本会访问一个后台数据库，来形成网页中的信息。这种通过 CGI 接口运行的脚本可以是任何程序设计语言写的，常见的一种是 Perl，它的特长是对字符串有很强的处理能力。这些脚本存放在 Web 服务器主机的一个特殊的目录下，通常是 cgi-bin。调用脚本生成网页的 URL 格式通常如下：

```
<A HREF="http://www.xyz.com/cgi-bin/sum.perl"></A>
```

这种 URL 中含有 cgi-bin 目录。这样的 URL 被浏览器提交给 Web 服务器后，服务器就会到 cgi-bin 目录下寻找并执行脚本 sum. perl。sum. perl 再根据传给它的参数形成 HTML 网页返回给 Web 服务器。这个脚本的作用可能是根据用户选择的商品生成用户的账单或者别的什么事儿。CGI 脚本是在 Web 服务器之外运行的，它通常是 Web 服务器与后台数据库之间的桥梁。

PHP(Hypertext Preprocessor，超文本预处理器)也是一种脚本语言，与通过 CGI 访问的脚本不同的是，这种脚本写的代码内嵌在 HTML 文件中，由 Web 服务器来执行。嵌入

PHP 脚本的 HTML 文件的扩展名变为.php。PHP 实际上是面向 Web 和服务器数据库交互过程的一种程序设计语言。显然,Web 服务器必须能够理解 PHP 语言,就像浏览器要认识 HTML 一样。

JSP 代表 Java 服务器页面(JavaServer Pages)的意思。它是由 Sun Microsystems 公司倡导、许多公司参与一起建立的一种动态网页技术标准。它与 PHP 类似。只不过网页中的动态部分是用 Java 语言而不是 PHP 编写的。含有这种脚本的网页文件的扩展名为.jsp。

ASP(Active Server Page,活动服务器页面)是一种包含了使用微软私有的 VB Script 或 JScript 脚本程序代码的网页。ASP 网页文件名的后缀是.asp。

小提示

由服务器根据具体情况动态生成内容的网页被称为动态网页。
生成动态网页的方法有使用 CGI 脚本、PHP 脚本、JSP 脚本以及 ASP 脚本。

介绍了这么多内容,现在你知道为什么访问网页时,有时在浏览器的地址栏中,或者超链接的属性中看到 URL 的末尾不是 html,而是 asp、jsp 或者 php 了。

7.1.2 Web 服务器的性能提升

Web 服务器现在绝对是最忙的服务器之一了。它要时刻准备为来自四面八方的网络客户服务。每台 Web 服务器必须有足够强的能力才能应付得过来。随着用户的增多,人们发现访问一些网页的速度已经慢得超出了耐心的极限。因此,Web 服务的提供者不得不想一些办法来解决或者缓解这个问题。不过,单靠一台服务器硬件性能的提升是怎样也不能赶上万维网用户的增长速度的。为此,人们必须想出一些能在目前的硬件条件下缓解问题的办法。下面就来介绍一下目前常见的一些提升 Web 服务器能力的手段。

1. 并发处理

要提升 Web 服务器的性能,首先想到的就是能不能提高每台 Web 服务器的工作效率,也就是减少它应答每个请求的时间。在某个时刻,可能会有很多用户同时访问同一个 Web 服务器。如果服务器每次只能处理一个用户请求,而让其他的请求等待,显然是不合适的。人们对这种服务器的响应速度不可能满意。所以,服务器必须能够同时为多个用户服务,也就是提供并发服务。每当服务器收到一个客户请求时,就创建一个新的进程/线程来完成为该客户服务的工作。这样,服务器就可以同时响应很多客户的请求了。在只有一个 CPU 的情况下,这些进程/线程分时使用 CPU。在有多个 CPU 时,这些线程/进程就可以在不同的 CPU 上真正的同时执行了。使用并发技术可以提高服务器机 CPU 的利用率。CPU 可以在为某个客户服务的空当再为其他客户服务,而客户则感觉服务器好像只在为他自己服务。现在所有的服务器都是并发处理用户请求的。

2. 缓存技术

要想提高整个系统的服务效率,还必须减少服务器处理每个用户请求的时间。怎样加快处理速度呢?还要从服务器处理请求的过程入手。服务器所做的主要事情是按照用户的

请求到磁盘上把文件取出来通过网络交给用户。对于动态网页则需要执行/启动相应的脚本生成网页。访问磁盘的速度是比较慢的，所以如果服务器把从磁盘取出的网页文件发给用户之后，并不把它从内存删除，等再有新的请求也需要这个网页时，服务器就不用访问硬盘了，这样响应的速度就加快了，效率也就提高了。可能你已经看出来了。这实际上就是缓存技术。当然，缓存的网页只能是静态网页，动态生成的网页是不能采用这种技术的。

3. 服务器园

依靠缓存可以缓解一下服务器的压力。不过，能作为缓存的内存大小毕竟有限，单台计算机的处理能力也有限。随着访问量的持续增长，一台服务器的处理能力已经远远不能满足要求了。所以还必须想出新的办法。既然一台服务器忙不过来，那就多增加一些服务器。让这些服务器提供相同的服务。不过客户的请求应该由哪个服务器来响应呢？这就必须有另外一个服务器来解决这个问题。为此，需要另外增加一台计算机作为前端服务器，由它来接收并把用户的请求分发给这些 Web 服务器。这些相同的一组服务器被称为服务器园(farm)。只增加服务器数量还不行，要发挥好这些服务器的作用，一些细节问题必须考虑到。例如，前端服务器在分发任务时，如果每次都把对相同页面的请求分发给同一服务器，那么这个服务器就可以使用缓存来提高响应速度了。此外，每个用户的请求通过前端服务器到达最终提供服务的服务器，按照网络中的正常规则，服务主机返回的网页文件也要通过前端服务器传给用户。但是，这样做不但加大了前端服务器的负担，而且也让回复的数据包多走了一段路。因此，需要想办法让回复的数据不再经过前端服务器直接按照合理的路径到达客户主机。

4. 镜像服务器

我们已经想了这么多的办法。可是，比起 Web 服务器的用户量来说还是不够多。采用服务器园的办法使总体服务能力大大提高，但是，所有的请求和应答仍然都要通过服务器园周围的网络运输。持续增长的用户访问流量很快就使这些网络不堪重负。怎么解决这个问题呢？讲出来也很容易。我们把这些提供相同服务的服务器不放在一起，而是放在网络中的各个不同地点，这样流量不就不那么集中了吗？这些分散在各地的服务器被称为镜像服务器。不过，现在就不能再使用前端服务器机来分配任务了。通常人们是在网站添加一些链接指向这些镜像服务器。用户通过这些链接到达镜像服务器。以后的访问就都针对这些镜像服务器了。这样就把流量分散开了。通常镜像服务器上都是一些静态信息，因为如果信息经常更新，那么就必须想办法保持这些镜像服务器之间的信息同步。如果信息时刻在更新，那么同步信息将是一件很不容易，也很麻烦的事情。

5. 代理服务器

除了镜像服务器，人们还使用了另外一种办法来提升 Web 服务器的能力。这就是前面提到的缓存。不过这次不是在服务器的内存中，而是在网络中的某台计算机上缓存网页。在这台计算机上有一个进程，它会缓存经它传递的网页。这台计算机(其实是计算机上的进程)被称为代理服务器。一般代理服务器被放在公司或者运营商网络的出口位置上。这些代理服务器会像 Web 服务器一样响应用户的网页获取请求。如果代理服务器上没有用户

需要的网页，那么它会像客户端一样到真正的 Web 服务器上取回网页，然后交给向其请求的客户，并同时把网页缓存下来。当下次再有客户需要这个网页时，它就可以直接从自己的缓存中取出应答客户了。为了使用代理服务器，用户需要在自己的浏览器中进行相应的设置，让所有的网页请求发送给代理服务器，而不是真正的 Web 服务器。

因为代理服务器离用户比较近，所以它不但可以减轻 Web 服务器的工作负担，而且能够加快响应的速度。代理服务器能够减少 Web 服务器周围网络的流量，尤其是减少用户网络的出入流量。这对于按照流量向上一级网络商付费的运营商来说，可是一笔额外的收获。

不过，你现在一定在想一个问题。代理服务器要把一个网页缓存多长时间呢？如果 Web 服务器上的网页变化了，代理服务器仍然用旧的缓存网页就不好了。为此，代理服务器需要确定网页的缓存时间。每个网页更新的速度是不一样的。例如，一个介绍经典名著的网页可能很久也不会更新；但是一个播放天气预报的网页则会每小时或者每天更新；如果是股票的网站，那么则要时刻更新，但是等到当天股市收盘后，网页的信息就会保持不变直到第二天开盘。因此，如果只泛泛地为所有网页设置相同的过期时间显然不合适。代理要想工作得好，它必须有一个很好的技巧来处理缓存网页的过期时间。例如，它可以根据网页上次修改时间距现在的时间差来推断网页可能再次被修改的时间。

当代理觉得一个网页已经过期了，它应该怎么办呢？最简单的办法就是直接从缓存中清除掉，等到有用户需要时，再到 Web 服务器上取回并缓存。可是，这时如果网页没有更新过，那么就白白多了一次对 Web 服务器的访问。为此，人们就在 HTTP 协议中增加了一个头部字段，If-modified-since。这个头部字段向 Web 服务器询问网页在某个时间之后是否被修改了。如果修改了，那么 Web 服务器就会返回修改后的网页。如果没有改动，那么它就在应答报文中说明网页没有改动，也不再附上网页的内容了。尽管这样代理服务器还是和 Web 服务器之间收发了两条报文，但是流量还是减少了很多，服务器的处理时间也会减少很多。

现在可以得出一个代理服务器的工作思路了。它接受用户的网页请求，如果自己缓存有用户需要的网页，那么它用缓存的网页应答。如果没有，它就到 Web 服务器上去取，取回来之后缓存，同时应答用户。代理服务器缓存网页时要计算每个网页的缓存有效时间，并与网页一同保存。当网页的缓存时间超过有效期时，代理服务器不再直接用缓存网页应答用户，而是向 Web 服务器发送对该网页的请求，里面带有一个询问网页在其缓存时间之后是否修改过的头部字段。Web 服务器收到请求后，对于已经修改的网页发回带有网页的应答。代理服务器缓存新网页并把它交给用户。如果网页没有被修改，则 Web 服务器直接发送一个“未修改”的应答。代理服务器收到应答后，更改过期网页的缓存时间，然后用缓存的网页应答用户。

显然，使用代理服务器是有一些风险的，如果网页过期时间设置不当，是会使用户得到过期的网页。同样，对于动态生成网页，是没有缓存的必要的。

小提示

提升 Web 服务器性能的方法有进行并发处理、采用缓存技术、使用服务器园、使用镜像服务器、设置代理服务器等。

7.2　电子邮件服务

电子邮件服务是互联网上最早提供的服务之一。它对人们接受互联网起到了重要的作用。电子邮件的实现，说起来比较容易。寄件人把要讲的事情写出来，然后通过网络发送给收件人就行了。这个简单的操作中存在几个必须解决的小问题。首先，收发双方必须有一定的收发规则，也就是协议。这包括用什么方式建立联系、收发邮件之前应该发送什么样的信息、邮件的格式应该是什么样的。收发双方还要对邮件地址的表示和使用达成一致。其次，如果发件人的主机直接把邮件发送给收件人的主机，那么在发件人发送邮件时，收件人的主机必须开着，而且准备好接收邮件。通常终端用户的主机都不是 24 小时运行的。这就给邮件的发送带来了困难。而电子邮件和电话相比，一个优势就是收件人可以在他有时间的时候处理邮件。即接收者具有主动权。而电话的接收者则比较被动，呼叫方拨打电话时，接收方并不一定很愿意处理这件事情。为了保持电子邮件的这个特点，同时又保证邮件的发送成功，人们在网络中设置了 24 小时不间断运行的服务器来代替用户接收/发送邮件。这就是电子邮件服务器，就好像日常生活中的邮局。正是由于上述原因，邮件系统呈现出如图 7-5 所示的结构。

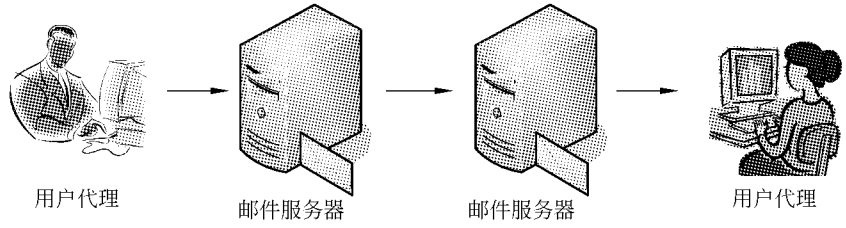


图 7-5　电子邮件系统结构

1. 邮件地址

为了发送电子邮件，用户首先需要在网络中的某个邮件服务器上注册，获得一个邮箱。邮箱由邮件地址来标识，如 xiaofang@youju.com。邮件地址由两部分构成，在符号@后面的是邮件服务器所在域的名字，而前面则是用户在该域邮件服务器上的账号名。每个注册的账号名，服务器都会为其开辟一个邮箱用来存放发给该账号用户的邮件。邮箱实际上就是文件系统中的一个目录。

2. 用户代理

要发送邮件，寄件人必须先把邮件写好。这时用户需要借助用户代理程序，也称为电子邮件客户端。用户代理程序有很多，如常见的 Foxmail、Outlook 等，以及现在不常见但是以前很流行或者在国外使用比较多的 elm、Netscape Messenger、Eudora 等。人们完全可以按照自己的喜好选择某个用户代理程序。这些用户代理程序的主要功能都是一样的。即为用户提供编撰邮件的界面，代替用户发送邮件，以及从邮件服务器上取回并显示接收的邮件。不同的代理程序只是在具体实现细节上有些不同，如不同的界面风格，像邮件列表、自动回复、信纸模板等不同的个性化服务等。

用户写好邮件之后，就可以告诉用户代理程序发送邮件了。一般这只需要按下用户代

理界面上提供的一个“发送”按钮。接到发送命令之后，用户代理就要和一个邮件服务器建立联系，把邮件交给它，让它帮忙发送给收件人的邮件服务器。这时，用户代理和邮件服务器之间的通信模式是典型的客户/服务器模式。用户代理是邮件服务器的客户端。它主动和服务器联系，建立连接。而邮件服务器则始终在运行，等待用户代理的连接。

与哪一个邮件服务器联系，一般用户代理程序是依据用户设置的“发送邮件服务器”这个选项中的内容来决定的。在用户初次使用用户代理程序，或者用户在代理程序中建立一个新账号时，用户代理程序都会要求用户设置“发送邮件服务器”或者“SMTP 服务器”。这个服务器通常是用户自己邮箱所在的服务器，例如 smtp. bjut. edu. cn。

小提示

邮件地址由用户在邮件服务器上的账号以及邮件服务器所在域的域名构成。

用户代理程序为用户提供编撰邮件的界面、代替用户发送邮件、从邮件服务器上取回并显示接收的邮件。

3. SMTP 协议

为什么发送邮件服务器，有时候也叫做 SMTP 服务器呢？那是因为用户代理程序和邮件服务器之间使用 SMTP 协议进行通信。SMTP(Simple Mail Transfer Protocol，简单邮件传输协议)协议是专门为发送电子邮件设计的。它是一个标准协议。因此，我们才能有各种风格不同、不同厂家/组织实现的，但是都能够与邮件服务器建立联系发送邮件的用户代理。这些用户代理在与邮件服务器通信时都使用标准的 SMTP 协议。所以，理解 SMTP 协议的邮件服务器都能够与它们通信，从其接收邮件。这就是定义标准协议的作用。实际上，邮件服务器也有很多厂家/组织实现的各种不同软件和版本，如 Linux 下的 Sendmail、Qmail、Postfix、exim、Zmailer，以及 Windows 下的 Microsoft Exchange Server 等。在 Internet 网络中 SMTP 协议应该算是一个古老的协议(Internet 的历史实在太短了，只有几十年)，在它被标准化的 20 年后人们仍然在使用它(在标准化之后也在不断地进行一些小的改进)，并且使用的人非常多。

SMTP 是一个非常简单的协议。但是，对于传送邮件，已经足够了。也许正是因为这个原因使它战胜了 ISO 定义的、复杂的邮件标准 X. 400。SMTP 协议定义了邮件传输的过程，以及在此过程中使用的命令和状态码、传送的邮件报文的格式。与 HTTP 协议一样，SMTP 定义的交互命令和状态码都是 ASCII 码字符[59]。因此，我们都可以直接读懂这些消息。例如，命令

```
mail from
```

指明邮件的发送人，后面要跟发件人的邮件地址。

SMTP 协议规定收发邮件的双方必须先打招呼。为此，它定义了 Hello 命令。该命令后面带一个域名，指明打招呼的客户端的名字。互相打完招呼之后，双方就可以进行邮件报文的传输了。最后互相说再见，结束通信。当收到发送方传递过来的命令时，邮件服务器要发送相应的状态码以及解释作为应答。例如，当客户端发出 HELO mydomain 打招呼时，邮件服务器发回

```
250  Hello mydomain, pleased to meet you
```

作为应答。这里 250 就是 SMTP 定义的状态码。它表示请求的动作没有问题，已经完成。这里表示它接受客户端的问候。

邮件报文中包括一些定义好的头部字段以及用户的邮件内容。头部字段中记录着邮件的发送者、接收者以及邮件经由的路径等。如果邮件发送者希望邮件接收者回复的邮件寄往另外一个信箱，那么也在头部中用一个特定的字段来表示（该头部字段名为 reply-to）。SMTP 采用 7 位的 ASCII 码来传送全部信息，包括邮件的内容。SMTP 协议的细节你可以去查看相关的书籍或者资料。

下面继续介绍邮件的发送过程。当用户主机上的用户代理使用 SMTP 协议联系上邮件服务器，并把邮件发送给它之后，用户代理的发送任务就结束了。通常每个邮件服务器都是 24 小时不间断运行的。所以在用户代理与发送邮件服务器建立联系时，一般都会成功。如果很不幸，发送邮件服务器出问题了，那么用户代理程序会向用户显示一个发送失败的通知。

邮件服务器收到要发送的邮件之后，把它放到自己的输出队列中。邮件服务器接下来的任务就是把这个邮件投递给收件人所在的邮件服务器。因此，发送邮件服务器需要和收件人的邮件服务器建立联系。收件人的电子邮件地址中符号@后面是邮件服务器所在域的名字。为此，发送邮件服务器需要先找到收件人邮件服务器的名字。这要借助网络中另外一个称为域名服务器 DNS 的帮助。在 DNS 的数据库中，存有各个域的邮件服务器的名字。发送邮件服务器通过 DNS 系统和收件人地址中的域名找到邮件服务器的名字，然后再通过 DNS 找到这个邮件服务器名字对应的 IP 地址，就可以和收件人的邮件服务器联系了。有时候一个域中会设有多个邮件服务器，那么在 DNS 记录中会给出这些邮件服务器的优先级。发送邮件服务器会根据优先级顺序试着向这些服务器投递邮件，直到邮件投递成功。发送邮件服务器和收件服务器之间仍然采用 SMTP 协议进行通信。这时发送邮件服务器变成了客户端，而收件人的服务器是服务器端。

收件人的邮件服务器收到邮件后，就会把邮件放入收件人的信箱，等待收件人阅读。虽然邮件服务器都是始终运行的，但是如果碰巧收件人的邮件服务器出了问题，那么发送邮件服务器会把邮件保持在自己的发送队列中，过一段时间再试着发送，直到成功或者试验了规定的时间和次数。如果最终没有成功，服务器会给发送者发送一个邮件告知邮件发送失败以及原因。

4. 邮件访问协议

到这里，邮件已经到达了收件人的邮件服务器，任务的大部分已经完成了。只剩下收件人的用户代理把邮件取出并显示给用户看了。可是，这时候出现了一个问题。用户代理程序不能用 SMTP 协议来从服务器上取回邮件。因为 SMTP 协议只规定了发送邮件的命令。而没有接收的命令。按照 SMTP 的机制，如果用户代理想要获得邮件，那么它必须作为服务器端，等待邮件服务器与其建立联系，并把邮件投递给它。但是，用户代理是在用户主机上运行的程序，通常是不能 24 小时运行的。在网络刚刚组建时普通用户非常少，网络上的主机都是研究网络的那些研究院的主机，这些计算机都是 24 小时开机的。所以当时设计出来的 SMTP 协议就成了这个样子。可是我们也说过就是因为现在用户的主机不是 24 小时运行的，我们才采用的邮件服务器，而不是把邮件直接递到收件人的主机。所以，要想把邮

件取到用户的主机，必须定义新的协议。这个新协议要提供“拉取”的方式获得邮件。也就是说，用户代理可以作为客户端通过协议与邮件服务器建立联系。它可以主动向服务器查询是否有邮件并请求从服务器获得邮件。邮件服务器能通过这个协议把邮件传送给用户代理。这样的协议被称为邮件访问协议。

邮件访问协议不只有一种。邮局协议(Post Office Protocol，POP)就是其中一种。另外常用的还有因特网邮件访问协议(Internet Message Access Protocol，IMAP)。在我们使用用户代理，配置“发送服务器”的同时，通常都要配置“接收服务器”的地址。也就是用户代理接收邮件时要联系的邮件服务器。有时候这一项写的是配置“POP3 服务器”，就是因为收取邮件时使用的是 POP3 协议。

有了这些邮件访问协议，收件人就可以命令用户代理程序(如单击“接收”按钮)从邮箱中取回并显示邮件了。这时用户代理程序就会与用户指定的接收服务器建立联系，然后通过邮件访问协议把邮件取到本地然后显示给用户看。

现在从发件人开始撰写邮件到收件人看到邮件的整个过程终于就完成了。我们用图 7-6 来对电子邮件系统做个总结。

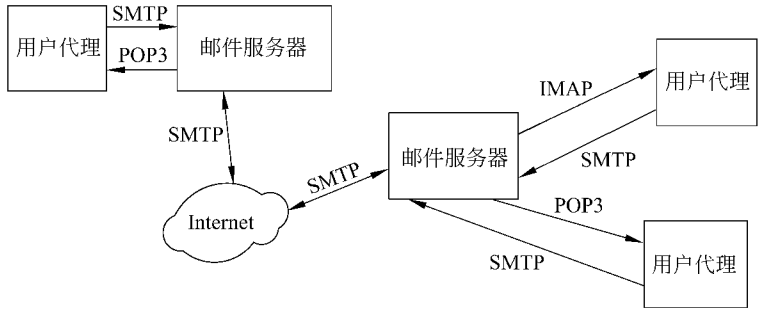


图 7-6　电子邮件系统

不过仔细回想上述这个过程，可以发现还是存在一个问题。前面说过 SMTP 是按照 7 位的 ASCII 码来传递邮件的。这意味着邮件中只能是 ASCII 码支持的那 128 个字符的 7 位编码。这个限制可太大了！可是，我们在使用电子邮件时并没有感觉到这个限制。我们几乎可以传送任意格式的文件，如视频、带有各种字符的 Word 文档，以及一些可执行文件等。这些内容中每个字节的最高位不可能都是 0，都是有意义的。这是怎么回事呢？是怎样实现的呢？

小提示

用户代理和邮件服务器之间、发送邮件服务器和接收邮件服务器之间都使用 SMTP 协议进行通信。

发送邮件服务器通过 DNS 系统获得接收邮件地址中域名所示域的邮件服务器地址。

用户代理通过邮件访问协议，如 POP、IMAP 从邮件服务器取回邮件。

5. MIME 及 BASE64 编码

在早期因为邮件的作用主要是用来互相通信——传递一些话语，而且当时的用户都使用英文。所以，SMTP 传递 ASCII 码已经足够了。可是后来人们希望能够在邮件中传递更

多的信息，如歌曲、可执行文件等。而这些文件中的内容是不能用 ASCII 码表示的。就是简单的邮件内容，因为不是用英文写的，比如汉语，ASCII 码也表示不了。为此，人们提出了一种称为多用途 Internet 邮件扩展的解决方案，简称为 MIME(Multipurpose Internet Mail Extensions)。这种方案不更改 SMTP 传输 7 位 ASCII 码的规则。但是，可以在邮件中传送音频、视频、图像等多种不同类型的邮件内容。这些不同类型文件中每个字节的 8 位都是有效数位。为了能够用 7 位 ASCII 码来传送这些文件，人们需要把这些文件中的二进制串进行重新编码，以使每个字节看起来都是 ASCII 码，来满足 SMTP 的要求。我们打个比方来说明这个解决方案。我们有一个运输车，车上每只箱子只能装 7 个物品(就是 7 位 ASCII 码)。但是我们的物品是按照 8 个一组打包的。因此我们必须重新把这些物品按照 7 个一组打包，以适应运输车每个箱子的大小。这被称为编码。在把物品从运输车中取出后，我们需要再把它们重新按 8 个一组打包。这被称为解码。MIME 就是采用这种编码和解码的办法，把各种文件的内容通过 SMTP 协议发送出去。

MIME 支持两种编码方式。我们这里只简单介绍其中一种——base64 编码。另外一种称为 QP，你可以自己去查资料。Base64 的编码表把 0～63 这 64 个数对应于 64 个字符的 ASCII 码作为编码表。这些字符是 A～Z、a～z、0～9、+、/。Base64 编码每次从要编码的文件中取出 3 个字节共 24 位数据。这 24 位数据然后被分成 6 位一组，这样一共是 4 组。每组被看作是一个二进制数。6 位二进制数对应的十进制数范围正好是 0～63。根据每组 6 位二进制数对应的十进制数，base64 按照编码表把它编码为对应字符的 ASCII 码。例如：

10101010　00110101　10001011

这三个字节按照 6 位一组重新分组后为：

101010　100011　010110　001011

它们对应的十进制数为 42，35，22，11。因此，它们的 base64 编码为这些数字对应的字符的 ASCII 码，即字母 q，j，W 和 L 的对应 ASCII 码：

01110001　01101010　01010111　01001100

这种方式就是把原来 3 包中的东西，现在装在 4 个包中。如果最后剩余的数据不足 4 个字节，即剩下一个字节(8 位)或者两个字节(16 位)，它们按照编码规则生成两个或者 3 个 ASCII 码，那么 Base64 会在后面再加两个或者一个等号字符“=”的 ASCII 码凑够 4 个字节。

现在邮件的内容已经被改造成 SMTP 协议能够接受的形式了。但是，发送方必须通知接收方，让其按照同样的规则来解释这些内容，否则接收方仍然按照 ASCII 码去解释，那就是乱码了。为此，MIME 在邮件报文头中增加了一些头部字段来向接收方描述这些信息。这些头部字段告诉接收方邮件的内容是使用哪个版本的 MIME、采用的是什么编码方式。头部字段还告诉接收方邮件内容是什么类型的文件，如 video/mpeg。这样接收方在解码之后，就知道用什么软件来解释邮件的内容了。通过这种方式，SMTP 就可以发送各种类型的文件了。

6. 邮件转发

按照上述过程，因为用户代理并不是始终运行，所以我们需要借助一台发送邮件服务器和一台接收邮件服务器的帮助。除此之外，邮件不需要经过其他任何邮件服务器。但是，由于最初的一些技术原因，某些邮件服务器之间并不能直接通信，所以邮件要通过中间若干邮

件服务器的中继才能到达收件人的信箱。所以，SMTP 协议允许邮件服务器中继转发与自己无关的邮件[61,62]。随着环境的变化，技术的提高，这个功能现在已经没有原来的用武之地了。但是，它却被发送垃圾邮件的人看中。这些人利用这个功能来掩饰垃圾邮件的来源。为此，较新版本的邮件服务器都自动关闭了中继功能。但是，一些老版本的邮件服务器默认是支持中继的。这就需要管理员自己来检查一下，把中继功能关闭，或者有选择地中继转发某些域的邮件。否则白白被人利用做了坏事还不知道。

7. WebMail

在电子邮件这部分内容的最后，还要讲另外一件事情。那就是 WebMail。除了使用 Foxmail、Outlook 等用户代理收发邮件之外，很多人也使用过浏览器收发过邮件。我们可以在任意地方通过浏览器访问邮件服务器，而不用进行任何设置。这对于在外出差，没有固定机器的用户来说非常方便。在我们看来，使用浏览器和用户代理程序没有什么太大的区别。不过，在技术上它们是有很大不同的。你一定记得，浏览器是使用 HTTP 协议与服务器打交道的。既然我们用浏览器来访问邮件，那么这个客户端就不会按照 SMTP 或者 POP3 协议来发送信息，而只能发送 HTTP 消息包。为此，在实现 WebMail 时，一般在邮件服务器端会增加一些特殊的程序来接收和应答用户通过浏览器发送过来的邮件请求。有时候，这些程序需要再使用 SMTP 或者 POP3 协议同 SMTP 邮件服务器或者 POP3 服务器进行交互。目前 WebMail 的实现并没有统一的标准，人们可以使用各种策略支持通过浏览器收发邮件的功能。也就是因为这种还在探索的现状，WebMail 系统，特别是发送邮件的功能，并不是总能如预期那样正常工作。

小提示

人们通过编码来让只能发送 7 位 ASCII 码的 SMTP 协议传递各种类型的文件。
早期邮件系统的中继转发功能默认情况下是打开的。
WebMail 通过浏览器来收发邮件。客户端和服务器端使用 HTTP 协议通信。

7.3 FTP 服务

FTP 服务，确切地说是文件传输服务。它可以让用户把文件放到服务器上，同时可以从服务器上下载所需要的文件。这种服务由用户机器上的客户端软件和服务器机上的服务器端软件协同提供。服务器端软件和客户端软件通信时遵循 FTP(File Transfer Protocol)协议。

FTP 协议是网络中出现最早的应用协议之一。最早的电子邮件就是采用 FTP 协议实现的，直到后来才逐步分离出来，形成独立的协议。

FTP 服务各相关部件的关系如图 7-7 所示[59]。在用户机器上运行的客户端程序主要有三个功能。首先为用户提供一个访问 FTP 服务器的界面，如显示本地和远端文件及目录、提供按钮让用户能够发布文件上载或者下传的命令。此外，客户端软件会与本地文件系统打交道，以便把本地文件上传给服务器，或者把从服务器获取的文件存放在本地。第三，就是与 FTP 服务器建立联系，进行文件传输。因为 FTP 协议已经被标准化，所以只要依据这个协议开发的客户端和服务器端程序就能够顺利通信。因此，现在存在很多不同的 FTP

服务器的客户端软件，如 CuteFtp、LeapFtp，以及服务器端软件，如 Linux 下的 VSFtp、Windows 下的微软服务器套件中带有的 FTP 服务器以及 Serv-U 等。

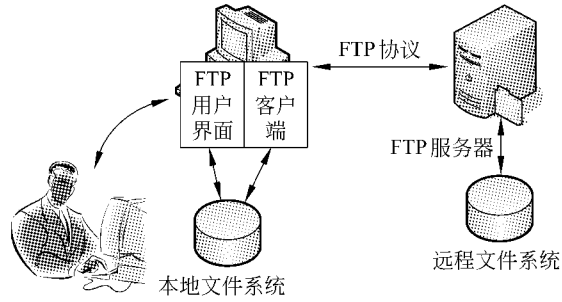


图 7-7　FTP 系统结构

默认情况下，FTP 服务器在 21 号端口等待客户端的连接。客户在与服务器连接成功后，服务器会对客户进行鉴别。这主要通过要求用户输入用户名和密码来完成。通过鉴别之后，用户就可以使用 FTP 定义的命令来查看服务器上的文件、下载或者上传文件了。

FTP 协议定义了用户可以使用的命令以及服务器返回的状态码。与 SMTP 协议一样，这些命令都是以 ASCII 码方式传输的，因此我们都是能够看得懂的。现在的 FTP 客户端软件多数是图形界面。它们以各种按钮/鼠标按键响应的形式封装了 FTP 的文本命令。我们可以使用文本行访问 FTP 服务器，这样就可以看到这些 FTP 命令了。例如在图 7-8 中，我们用命令 ftp 启动了 FTP 客户端。这时使用帮助命令“?”显示出全部可以使用的命令。其中，open 命令用来连接一个 FTP 服务器。从图中可以看到当用 open 命令成功与 FTP 服务器 ftp. pku. edu. cn 建立联系之后，屏幕上显示出以数字 220 开始的一行文字。这里 220 是 FTP 协议定义的状态码之一，表示成功。然后屏幕上显示出 User 这一行要求输入用户名来让服务器鉴别。这些命令的功能你可以自己去查阅手册来了解。

图 7-8　文本界面的 FTP 客户端

有一点特别的地方是，当用户发布上传或者下载某个文件的命令后，客户端和服务器并不通过刚刚创建的这个连接来传递数据，而是另外再创建一个新的数据连接来传输数据。而原来的那个连接则继续用来传递其他控制命令。因此，它被称为控制连接。当创建的数据连接完成数据传输任务后，就会被关闭。如果需要传送新的文件，就要再创建新的数据连接，即每个数据连接只用来传递一个文件。

因为服务器总是运行的，而客户端只在需要时才运行起来。所以，在创建控制连接时，必须是客户端主动与服务器建立连接。可是，创建数据连接时，一定是在控制连接建立之后，这意味着通信双方的软件已经都在运行。那么，无论谁主动发起建立数据连接的请求，都可以准确地找到对方。也就是说，在传输文件时，通信双方谁作为客户端进程、谁作为服务器端进程都是可行的。因此，FTP 的数据连接有两种创建方式。一种是由 FTP 服务器端主动发起连接请求创建的。另外一种方式则仍然是由客户端软件发起建立。前者被称为主动模式，后者称为被动模式。可以看出，这种模式命名法是从服务器的角度出发的（通常用户所在网络防火墙会禁止 FTP 的主动模式，因此只能选择被动模式）。

小提示

文件传输服务的服务器和客户端通过 FTP 协议进行通信。
FTP 使用控制连接传送命令、用独立的数据连接传送文件。
数据连接的建立分为主动和被动两种模式，这种命名法是从服务器的角度出发的。

复习题

1. 什么是客户/服务器模式？为什么说它是网络应用程序结构的基础？
2. 万维网指的是什么？
3. 什么是 URL？
4. Web 服务器的客户端软件叫什么？它们之间通过什么协议通信？
5. HTML 语言是做什么用的？为什么说浏览器是 HTML 解释器？
6. 什么是动态网页？什么是静态网页？
7. 实现动态网页的技术有哪些？
8. 有哪些技术可以提升 Web 服务器的性能？
9. 邮件地址由什么构成？
10. 简述邮件从编写到被收件人看到所经历的过程。
11. 为什么要使用邮件访问协议？
12. 怎样能让只会传送 ASCII 码的 SMTP 协议可以支持多种类型的文件传输？
13. FTP 服务中，主动模式指的是什么？

讨论

1. 你认为计算机网络为人们的生活带来了什么变革？
2. 提升 Web 服务器性能的技术是否可以应用到其他服务器上，如 FTP 服务器，邮件服务器，为什么？
3. 你认为设计并提供一个网络服务应该注意哪些问题？
4. 如果让你来为现有的邮件服务器增加通过 Web 访问的支持，你能想出应该怎样实现吗？

实验

查找相关资料，配置一个 Web 服务器来发布你的个人主页。

第 8 章　网络平台基础知识

带有网络功能的计算机能够为人们做更多的事情。不过，计算机要想连接到网络中，是需要用户做一些工作的。尽管计算机的基本联网功能已经内置在操作系统中，但是这些功能程序的运行还需要用户提供一些基本的信息。通常是做一些必要的设置。尽管这些工作越来越少，越来越容易做，但是，还是需要用户参与一下。这一章我们先来说说为什么要做这些事，也就是计算机网络涉及的一些基本知识和技术。然后再介绍要做哪些事情以及怎样做。最后，还将介绍 DHCP 和 DNS 两个服务。这两个服务与第 7 章所介绍的服务有所不同。因为它们所提供的服务并不是满足用户使用网络共享资源的需求，而是为方便用户使用网络而提供的支持性服务，所以，把它们放在这一章中介绍。

8.1　计算机网络的层次结构

前面讲过，计算机系统因为太复杂而被分解成不同的层次。这样可以使问题的解决变得简单一些。同样道理，要想实现两台计算机的通信也有很多事情要做。为此，计算机网络的通信任务也被分解成不同的小问题，然后由不同的专家有针对性地解决。一部分通信软件的功能要依赖于其他软件的服务来完成。这样整个网络系统也形成一种层次结构。这就是我们经常听到的计算机网络层次结构。

8.1.1　分层动机

计算机网络平台要完成通信到底要做多少事情？真的需要分层吗？为了回答这个问题，现在我们就来想一想，如果位于不同计算机上的两个进程要相互通信大致上必须做的事情。我们还是从小明说起吧。假定现在没有 QQ、MSN 等任何聊天软件工具。小明拥有的只是一台计算机，它被用双绞线连接到小芳的机器上。为了能够通过计算机与小芳联系，小明要自己写一个聊天程序。现在我们就和小明一起想想他的程序代码要完成哪些任务。

首先，小明向小芳表达的意思必须传达给计算机，因此，程序需要提供一个人机界面。例如，程序可以在开始运行时出现一个“语音”按钮，单击它之后，小明就可以对着麦克风讲话，然后这些语音以数字形式存储下来。为了实现这个功能，小明的程序需要产生一个按钮；在判断按钮被单击之后，要通过麦克风的接口去采集声音；然后把声音以某种形式记录在计算机里面。处理声音还是比较麻烦。所以，小明还是先考虑最简单的方式，通过键盘把要说的话用文字表达出来。那么，程序中需要编写一个界面，上面有一个窗口，能够让小明输入文字，窗口下面有一个“发送”按钮。小明输入文字之后，程序需要把这些文字以计算机中的某种二进制形式暂存下来。等单击“发送”按钮时，程序就把暂存的信息转换成电信号放到铜线上发送出去。因为是双向通信，小明的程序不能只有发送信息的功能，还要有接收信息的功能。当小明这台计算机上的信息发送出去之后。同样的程序也要在小芳计算机上运行来接收信息。它首先需要从铜线上接收电信号，然后把电信号转换成计算机中信息表示格式，最后把信息翻译成字符，在屏幕上显示出来。整个事情也就做完了。当小芳回复

时，程序所作的事情也就是反过来重复这个过程。事情现在看起来好像还挺简单的！好像还用不着大动干戈——把它层次化。

事情确实是这样吗？在开始动手写程序之前，我们需要再把事情想得周全一些。首先，字符在计算机中用什么表示比较好呢？是用 ASCII 码还是 Unicode 呢？选择什么编码有很大影响吗？另外，信号发送出去后，如果在线路上受到干扰，接收方怎么判断出信号是否完好无缺呢？例如，如果发送的“我今天没有课”到了对方变成了“我今天有课”，接收方程序怎么能判断出来对错呢？如果知道发送的信号出问题了，那么应该怎么改正或者补救呢？如果两台计算机分别放在北京和上海，那应该怎么办呢？是不可能用一根线直接把这么远的两台计算机连接起来的。是不是这么远的计算机就不能通信了？是不是可以通过其他计算机或者网络设备转接？可是小明发送的数据包怎样才能够越过这些台中间的计算机或网络设备找到小芳的计算机呢？怎么才能让中间的计算机或网络设备为小明转发数据包呢？这些中间的计算机或网络设备怎么知道数据包是要发送给小芳，而不是别人呢？如果有很多网络设备可以转发数据包，有很多条路可以到达小芳的计算机，那么走哪条路更好呢？当从铜线上接收电信号时，怎样判断哪些信号是有效信号？即有用信息从哪里开始，到哪里又是结束呢？在铜线上用什么样的信号来表示信息呢？即用什么表示二进制数码 1？什么表示 0？如果发送方发送的速度太快，接收方的计算机处理不过来怎么办？如果途中的链路有些不是铜线，而是光纤，那怎么办呢？电信号是不能通过光纤的。我们就先想到这里吧。可以看出，如果只有物理上的网络连接，没有任何其他软件上的服务支持，要想写一个能够运转的网络程序要做的事情非常多，尤其是要进行远距离的通信。实际上，我们前面虽然考虑了很多，但是仍然有很多细节没有涉及。因此得出的结论是，一个人或者几个人是不可能完成网络通信这个任务的。为此，网络系统中也借鉴了社会分工的思想，将这个庞大的任务分割成不同的小任务。每个小任务由不同的人或组织来完成。最终一个网络通信程序依靠其他程序提供的服务完成通信目标。在网络系统中这种分工具有服务与被服务的关系，因此形成的是一种层次结构，所以被称为分层。

8.1.2 分层与参考模型

既然我们已经同意小明要写的这个网络聊天程序要分工来完成，现在就来分一下。首先，可以把程序中与硬件打交道的部分分离出来，由熟悉通信电路和硬件的人来写。因为小明对硬件实在是了解得太少了。这部分程序完成低级通信，把交给它的 0/1 串传递给网络上相连的计算机，同时能够把链路上相邻计算机传递过来的信号转换成 0/1 串。当然，这段程序必须在通信双方的计算机上都运行，两方都运行收发功能，才能实现一个合理的双向通信。假定这段程序在两台通信主机上表现为两个进程，分别称为低级通信进程 A 和 B，如图 8-1 所示。从实现上来说，这段程序要判断链路是铜线，还是光纤或者别的什么，然后根据情况将 0/1 串转换成相应的电信号或者光信号发送出去[①]。接收信号时，要能够判断出链路上何处是有效信号，要能够根据发送时采用的信号转换规则把信号翻译出来。这两个通信进程需要事先进行一些约定，如使用什么频段的信号，用什么样的信号来表示有效信息的开始以及结束等。只有这样两个进程才能正确地把对方发过来的信号解释成原来的 0/1 串。

① 这些功能在实际的网络中大部分是由硬件直接完成的，而且不同传输介质的实现硬件也不同。这里把它说成一段程序，只是为了理解上的简单。

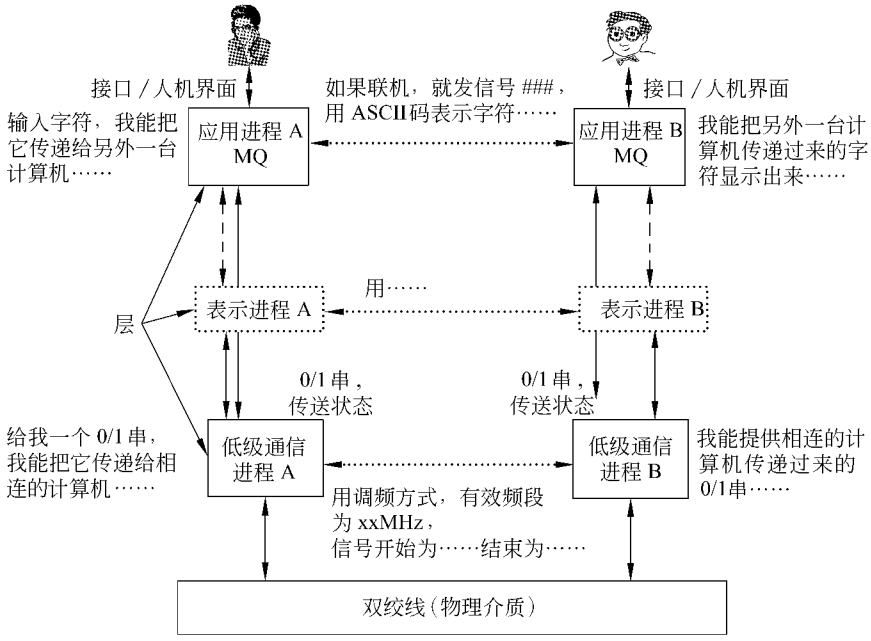


图 8-1　计算机网络的分层示意图

有了低级通信程序，如果小明再想写一个聊天程序，现在他可以把注意力集中在界面设计以及字符的表示上了。假定小明写的聊天程序名为 MQ。MQ 的功能是把用户输入的字符传递给另外一台计算机，还能够把另外一台计算机传递过来的字符显示出来。当然，MQ 程序应该是收发方同体，需要同时在两台计算机上运行。我们把 MQ 在两台计算机上的运行分别称为应用进程 A 和 B，如图 8-1 所示。同样这两个应用进程之间也需要有一些约定，例如，对方计算机闲置就发送“＃＃＃”信号通知对方，用 ASCII 码表示字符等。这样接收方才能正确解释发送方的意思。现在 MQ 把收发信息的事情完全交给低级通信程序来处理。MQ 只要把输入的字符转换成 0/1 串，然后交给低级通信进程。在接收方 MQ 再把低级通信进程收到的 0/1 串转换成字符显示出来。这样小明就可以不用了解硬件知识了。他再也不用去了解什么是频率、相变、折射率、光电转换了。他甚至都不用知道两台计算机是通过什么物理线路连接在一起的。

我们还可以把任务进一步细分。例如，把数据表示这一块也从小明的程序中提取出来，由熟悉各种数据格式以及编码的人去做。图 8-1 中的表示进程就是完成这部分功能的程序的运行体。这样，小明连编码都不用考虑了。他只要专心设计一个用户喜欢的界面。多出来的时间他就可以想想怎样提供语音聊天了。当然，如果距离很远的两台计算机想要通信，那么还必须有人写程序来解决数据包怎样跨越千万台网络设备到达目标主机的问题。

这样一个大的任务就被分解成若干个不同的子任务。每个子任务可以分别由相应的专家来完成。这些子任务之间形成一种服务与被服务的层次关系。因此，划分网络通信任务的这个过程被称为分层。每层的功能称之为服务[60]。例如，在图 8-1 中，低级通信进程提供的服务就是“把 0/1 串通过某种形式传递给相邻计算机，并且把相邻计算机传递过来信号转换成 0/1 串”。低级通信进程是为表示进程提供服务的。而表示进程是为应用进程提供

服务的。我们把每个子任务称之为一个层。例如，在图 8-1 中我们可以看到三个层。

现在我们已经达成共识，网络通信的功能很复杂，必须要分层实现它。但是，整个网络系统应该分成几层？每层的功能应该是什么？人们对此形成了不同的看法。下面介绍的两个参考模型就是两种不同的分法。

在刚刚开始建设网络时，人们就意识到分层的问题，并开始讨论到底应该怎样分层。最终人们设计出了一个分层的参考模型。这就是 1978 年国际标准化组织定义的“开放系统互联参考模型(Open System Interconnection Reference Model)”，简称 OSI 参考模型。OSI 是个参考模型。顾名思义就是让实现网络的人来参考。

OSI 模型把网络分成了 7 层，从上到下依次分别是应用层、表示层、会话层、传输层、网络层、链路层以及物理层。图 8-2 是 OSI 参考模型的示意图。不过，OSI 参考模型只给出了每层应该完成的任务，对于实现细节没有更多的描述。它也没有定义每层上具体的服务和相关协议。因为这是在网络实施之初或之前制定的计划，所以要想有很多周密的实现细节很难。就好像我们在第一次开始做某个项目时制定的详细计划一样，其实总是很难有丰富的细节。

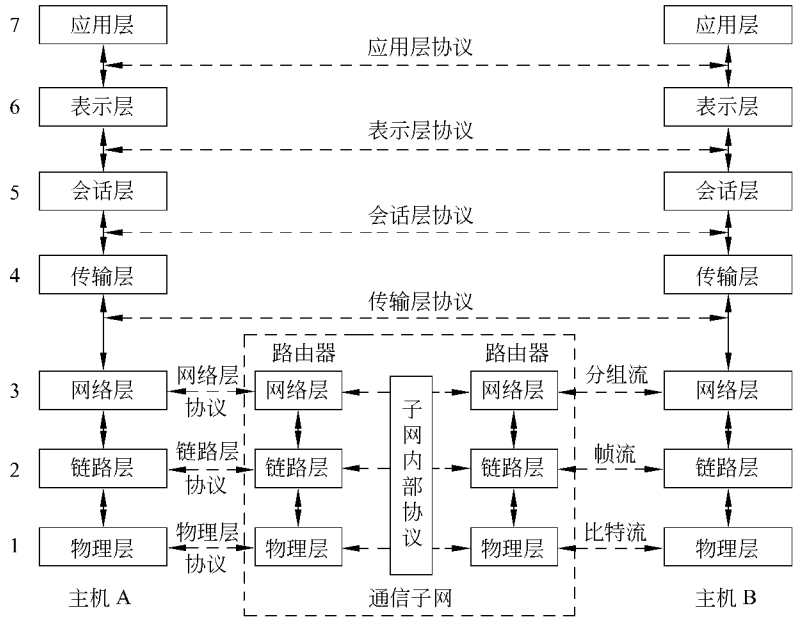


图 8-2 OSI 参考模型

OSI 七层网络参考模型并没有被实现。它只是一个设计方案，而实际中运行的网络系统却是另外一个样子。为此，我们不得不提及另外一个参考模型——TCP/IP 参考模型。这个模型与 OSI 参考模型的出身正好相反。它不是设计网络时的产物。而是在网络构建起来之后，根据实际的网络状况抽象出来的一个结果。这个实际的网络就是今天被广泛使用的 Internet。因为 Internet 使用的两个最重要的协议是 TCP 协议和 IP 协议，所以 Internet 使用的协议栈被称为 TCP/IP 协议栈。它的网络模型也被称为 TCP/IP 参考模型。图 8-3 是 TCP/IP 参考模型的示意图。因为是一个实际网络的抽象，所以 TCP/IP 参考模型的内容非常丰富。每一层有具体的服务，相应的协议也非常多而且具体。但是，因为是先

有网络后有模型，所以 TCP/IP 模型各层之间的界限有些地方不是很清晰，存在上层使用下层数据，甚至下层服务依赖于上层服务的情况。

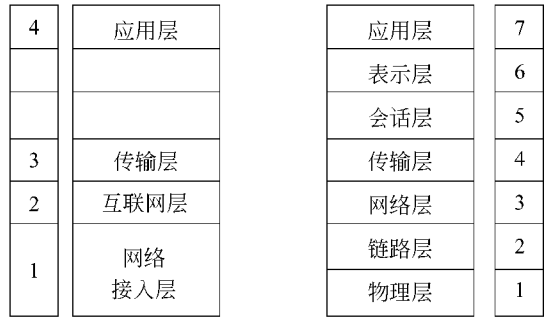


图 8-3　TCP/IP 参考模型与 OSI 参考模型的对应关系

在 TCP/IP 模型中，Internet 网络被划分成 4 层，即应用层、传输层、互联网层以及网络接入层。它们分别对应于 OSI 参考模型的应用层、传输层、网络层以及链路层和物理层，如图 8-3 所示。

在 TCP/IP 模型中并没有表示层和会话层，只有一个应用层。在网络建设的初期，表示层和会话层的任务显得不是非常重要，因而实际的 TCP/IP 网络中这部分功能很弱，因此并没有把它们抽象成为一层。随着计算机网络的发展，表示层和会话层的一些任务正变得越来越重要，如数据格式统一，只不过它们目前都作为应用层的一部分正在被实现。或许有一天，TCP/IP 参考模型会与 OSI 参考模型变得一致起来。

小提示

常见的网络参考模型有 OSI 参考模型和 TCP/IP 参考模型两种。

OSI 参考模型把网络分成了 7 层，从上到下依次分别是应用层、表示层、会话层、传输层、网络层、链路层以及物理层。

在 TCP/IP 模型中，Internet 网络被划分成 4 层，即应用层、传输层、互联网层以及网络接入层。

8.2　将计算机接入网络

8.2.1　实现物理连接

要把计算机连接到网络中，至少需要三样硬件，即两个接口和一根连接线。第一个接口是网络的接口或者说端口。一般办公室或者房间中会预装有这样的网络端口。它们看起来有点儿像电话的接入插口。这些端口的另一端连接着网络中的集线器、交换机或路由器的一个端口。第二个接口在计算机上，由网络接口卡提供。网络接口卡，也叫网络适配器，即常说的网卡。接入不同的网络，需要不同的网络适配器。网卡负责将信息转换成相应的电/光信号发送出去并翻译接收到的电/光信号。用连接线把这两个接口连接起来，计算机就接入网络了。

现在使用最多的局域网是以太网，所以就以接入以太网为例来具体说明怎样把计算机接入到网络中。首先，必须找一个空闲的网络端口，比如所在房间墙上预装的网络插

口，或者集线器、交换机、路由器的一个空闲端口。这些以太网的插口形状如图 8-4 所示。它与电话机上接电话线的插口有些像，不过看起来更宽一些。这种插口叫做 RJ45 端口，图 8-5 是它的一个示意图。这种插口实际上是个 8 芯接口。要想把网线插入这个插口中，必须使用 RJ45 连接器，就是常说的水晶头，如图 8-6 所示。图 8-7 是连接在网线上的水晶头。

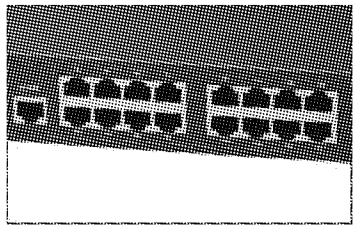


图 8-4　路由器的端口

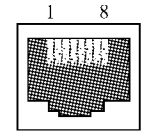


图 8-5　RJ45 端口

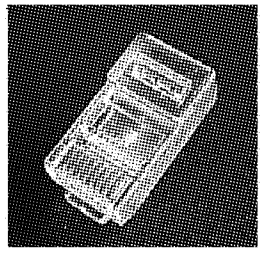


图 8-6　RJ45 连接器

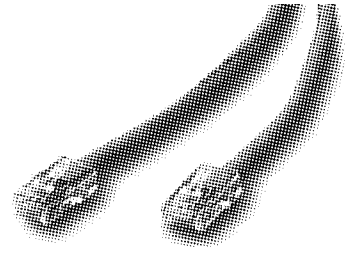


图 8-7　网线上的水晶头

找到网络端口之后，计算机上还需要一个以太网卡。目前网卡的速度一般在 100Mb/s、1000Mb/s 不等。我们可以根据需要选择。当然光有一个高速的网卡是不行的。还要考虑网络接入端口的速率，而且网线的传输能力也要与之配套。这就好像买了一个很大的水桶，但是如果只有一根很细的水管接到一个很小的水龙头上，是不能在很短时间内接到很多水的。网卡需要插入到计算机主板相应的接口上。现在网卡多数用的是 PCI 接口。还有很多计算机主板已经把以太网功能集成在其中了。这样就不用考虑再另外添置网卡了。图 8-8 是一块以太网卡。从图上可以看到网卡留有一个 RJ45 插口。用网线把这个插口和网络端口连接起来，从物理上我们的计算机就连接到网络上了。

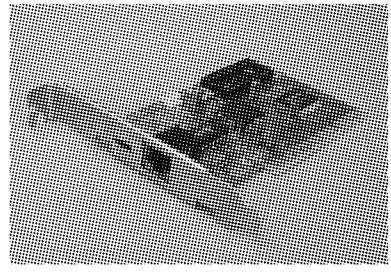


图 8-8　网卡

网线虽然只是一根简单的线，但是里面也有很多说道。目前以太网主要使用双绞线接入，可以直接去买做好的网线，也可以自己动手来做一根网线。如果自己动手做，那么需要有一根/捆双绞线、两个水晶头，以及一些必要的工具，如压线钳，如图 8-9 所示，电缆测试仪，如图 8-10 所示。

每根双绞线中有 4 对不同颜色的线。这 8 根线的颜色为白绿、绿、白橙、橙、白蓝、蓝、白棕、棕。这 4 对线的两端分别要插入两个水晶头中，这样网线才能够插入到端口中。要想网线能够畅通且干扰小，两端水晶头中4对线的排列顺序是要有讲究的。目前有两种排序的

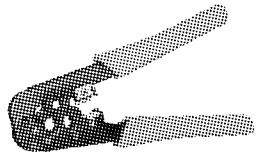


图 8-9　压线钳

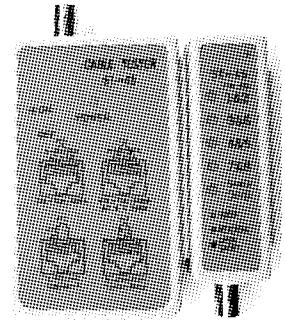
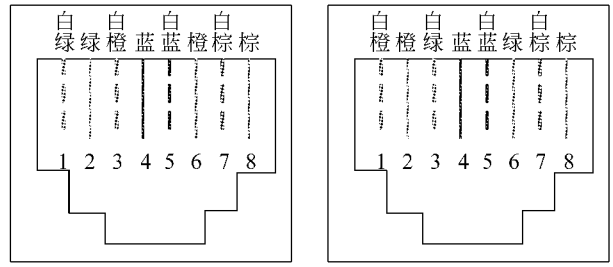


图 8-10　电缆测试仪

标准，分别称为 EIA/TIA568A 和 EIA/TIA568B。这两个标准规定了这 4 对线的排列以及每根线的用处。图 8-11 是这两个标准规定的线序。实际上这两个标准并没有本质区别，只是颜色不一样而已。在这 4 对线中，4/5、7/8 这两对没有定义。而 1/2 和 3/6 这两对线则分别用于发送和接收信号。仔细看这两个标准，发现它们的区别是把 1/2 和 3/6 这两根线的颜色对调了一下。图 8-10 中的电缆测试仪的背面印着的图形就是这两个标准。



(a) T568A　　(b) T568B

图 8-11　T568A 和 T568B 标准

小提示

接入不同的网络需要使用不同类型的网卡。
以太网使用 RJ45 接口，双绞线有 T568A 和 T568B 两种线序标准。

如果网线两端的线序都是按照 T568A 或者 T568B 的顺序，这样的网线被称为直通线。如果要连接网卡与集线器或交换机的普通端口那么就需要使用直通线。如果网线一端的线序是 568A，而另外一端是 568B，则被称为交叉线。交叉线适合连接两个计算机的网卡，或者两个交换机的普通端口（你能想明白是为什么吗？想一下每根线的用处）。通常我们使用的以太网端口的另外一端连接的都是集线器或交换机的普通端口，所以需要用直通线把计算机网卡连接到墙上预留出的以太网端口或者直接连接到集线器或交换机的普通端口上。

小提示

连接计算机到交换机/路由器的端口要用直通线。
直接连接两个计算机要用交叉线。

制作一根网线一般经过以下基本步骤[63]：

① 从双绞线缆上剪下所需要长度的双绞线(至少 0.6m,最多不超过 100m)。

② 将双绞线一端的外皮除去 2～3cm。

③ 按照 568B 标准或者 568A 标准把线排好序。

④ 将裸露出的双绞线剪下只剩约 14mm 的长度。

⑤ 将双绞线的每一根线依序插入水晶头的线槽内,一直插到线槽的顶端(将水晶头有塑造料弹簧片的一面向下,有针脚的一方向上,使有针脚的一端指向远离自己的方向,有方型孔的一端对着自己。此时,最左边的是第 1 脚,最右边的是第 8 脚,其余依次顺序排列)。

⑥ 用 RJ45 压线钳用力压水晶头,这样水晶头中的金属针脚就会切入线的塑料皮与其中的线连接上了。

按照这些步骤把双绞线两端都接好水晶头之后,一根网线就做好了。在使用之前,我们要用图 8-10 中的电缆测试仪先测试一下网线是否连通。把网线接在测试仪的两端,打开测试仪的开关。如果指示灯依次为绿色闪过,那么我们就成功了。在自己动手做网线之前,建议你到互联网上去看一看,你会发现更详细的图示以及注意事项。

小提示

如果你选择去买做好的、带有水晶头的网线,那么要记得讲清楚需要的是直通线还是交叉线。

8.2.2 进行网络信息配置

安装完网卡、连接好网线之后,我们还需要做一些工作,计算机才能与网络上的其他计算机通信。首先,必须安装网卡的驱动程序。以前我们需要手工来做这件事情,不过现在的操作系统已经能够支持很多常见类型的网卡了。它可以自动检测到网卡的类型和型号,并自动安装相应的驱动程序。对于操作系统不能识别的网卡,就需要你自己到该网卡生产厂家的网站上或者跟经销商联系获得网卡在我们所用操作系统中的驱动程序。然后,按照配套提供的通常叫做 Readme 或者 Install 等名字的指导文档,一步一步安装即可。

驱动程序安装完毕之后,还需要进行一些必要的网络参数设置。主要是设置一系列相关的 IP 地址。这就好像电话已经安装完毕,但是还需要配置电话号码一样。在介绍这些设置之前,我们先来了解一些关于 IP 地址的知识。

小提示

计算机要想上网需要做的工作包括进行物理连线、安装网卡驱动、配置网络参数。

1. IP 地址

IP 地址说起来很简单。它就是主机在网络中的一个标识号,是由 IP 协议定义在 Internet 的 IP 层(互联网层)中使用的,所以叫 IP 地址。IP 地址的作用是在网络中标识主机或网络设备,所以是不能重复的,就像电话机的电话号码。每个需要在网络中通信的设备至少要有一个与众不同的 IP 地址。

现在 Internet 使用的 IP 协议版本号是 4,即常说的 IPv4。IPv4 中定义的 IP 地址是

32 位的二进制数。不过 32 位的二进制数写起来太麻烦了，而且很容易出错。因此，IPv4 规定了 IP 地址的点十进制表示法。这种表示法把 32 位的二进制 IP 地址分成 4 段，每段 8 位。这 4 个 8 位二进制数分别被转换成十进制数，按顺序排列，中间用句点隔开，就形成了 IP 地址的点十进制表示。例如，我们前面提到的 IP 地址 162.105.203.25，它实际上表示的标识号是 32 位二进制数：

10100010　01101001　11001011　00011001

如果书写起这个数，真的很难保证不出错。162.105.203.25 就是这个 32 位 IP 地址的点十进制表示。

小提示

IP 地址是 IP 协议定义的、在网络层用来标示主机或设备的地址。
IPv4 定义的 IP 地址是 32 位二进制数。通常用点十进制表示法来写 IP 地址。

人们日常用的通信地址都是有层次结构的。例如，“北京市朝阳区平乐园 100 号”。按照这个地址，邮局在寄信件时首先会把信件寄到北京市，北京市的邮局再分发到朝阳区的支局，然后朝阳区的支局再找到平乐园 100 号。IP 地址也被定义成这样。32 位的 IP 地址被分成了两个部分。前面一部分称为网络号，后面部分称为主机号。每个网络号标识一个网络。路由器设法得到每个网络的位置，并记录下来。当路由器拿到一个数据报的目标 IP 地址后，它从中取出网络号，以此查找自己记录的位置信息，找到数据报应该转发的方向。等数据包到达目标网络后，目标网络中的路由器再根据主机号把数据报发送给目标主机。

实际上网络号部分还会被细分为更小的子网号。就好像“北京市朝阳区”这个地址实际还可以分为两部分一样。那么 32 位 IP 地址中到底有多少位是网络号呢？为了说明这件事情，每个 IP 地址在使用时必须配套使用一个掩码来指明它所在的网络号，即 IP 地址中前多少位是网络号。掩码有两种表示方式。一种是在 IP 地址后面加上一个“/”，然后加上网络号的位数，如 162.105.203.25/24 表示地址的前 24 位为网络号。另外一种方式是用点十进制表示法，形成一个类似 IP 地址的表示。即用 32 二进制数表示掩码，掩码的网络号部分全为 1，其余为 0，如前面 24 位的掩码的点十进制表示即为 255.255.255.0。

这里还要说明一点。IP 地址实际上是主机上每个网络接口的标识。每个 IP 地址中的网络号部分必须与其所在网络相同。就像你家在北京市朝阳区平乐园 100 号。你不能使用北京市海淀区平乐园 100 号作为你家的地址，因为邮局如果按照这个地址会把邮寄给你家的信件投递到海淀区。结果在海淀区找不到平乐园，那么邮寄给你的这封信就会被当作无主信件丢掉了。路由器在路由数据报时也是这样做的。如果一个计算机有两个不同网络接口连接到两个不同的网络中，那么它必须有两个不同的、分属两个网络的 IP 地址。就好像你家有前后两个门分别属于两条不同的街道，而这两条街道又分属不同的区，那么要想从前后两个门都能收到邮件，就要使用两个不同的地址。

每台计算机在网络中必须使用与众不同的 IP 地址，即地址不能重复。所以必须有人来管理 IP 地址分配的事情。这个工作现在由一个非营利性的机构——互联网名称与地址分配机构 ICANN(Internet Corporation for Assigned Names and Numbers)来负责。分配 IP

地址是ICANN的工作之一，实际工作是由其下设的一个地址支持组织来完成。起初IP地址是由美国政府授权另外一个机构分配的，即互联网编号分配机构IANA(Internet Assigned Numbers Authority)。在ICANN成立以后，IP地址的分配就由ICANN统一负责管理了。但是具体执行还是由IANA来负责[64]。

ICANN并不直接面向终端用户。它先是把地址分配给地域性的IP地址管理机构RIR(Regional Internet Registry)。目前有5个地域性的IP地址管理机构，即ARIN(北美地区)、LACNIC(拉丁美洲)、RIPE NCC(欧洲地区)、APNIC(亚太地区)、AFRINIC(非洲地区)。在RIR之下是国家级注册机构(NIR)和本地区注册机构(LIR)。我国的国家级注册机构是中国互联网络信息中心(CNNIC)。RIR负责将地址空间分配给下一级注册组织，如NIR或者ISP，同时授权他们进行地址空间的指定和分配的权利。而用户再从NIR或者本地区注册机构LIR，甚至更下一级的ISP处申请到IP地址。

小提示

掩码表明IP地址中有多少位是网络号。
IP地址由非营利性机构ICANN负责管理。

因为IPv4的地址分配紧张以及其他一些问题，人们提出了IP协议高级版本，即IPv6。在IPv6中，地址长度被定义为128位二进制数。这样可以有2^{128}个不同的IP地址。从理论上这足够覆盖连接到网络上的物体。不过128位二进制数的写法是个不小的问题。IPv6规定了一种称为“冒号十六进制”的写法。首先，把128位的IP地址分为8组，每组16位，然后每组写成4个十六进制数，每组之间用冒号分隔。例如，二进制IPv6地址：

```
0010000111011010000000001101001100000000000000000010111100111011
0000001010101010000000001111111111111111000101000100111000101101O
```

按照16位一组转换成十六进制，以冒号分隔之后，形成的IPv6地址表示为：

```
21DA:00D3:0000:2F3B:02AA:00FF:FE28:9C5A
```

每个十六进制组中的前导0可以省略。全为0的组则至少保留一个0。例如：

```
FE80:0000:0000:0000:02AA:00FF:FE9A:4CA2
```

可以表示成：

```
FE80:0:0:0:2AA:FF:FE9A:4CA2
```

而

```
0000:0000:0000:0000:0000:0000:0000:0000
```

则可以写成：

```
0:0:0:0:0:0:0:0
```

冒号十六进制格式中相邻的连续零位可以被压缩，用双冒号“::”表示。不过，每个地址中只能出现一个双冒号“::”，否则就无法判断出双冒号之间省略多少位了。例如：

```
FE80:0:0:0:2AA:FF:FE9A:4CA2
```

可以表示成：

```
FE80::2AA:FF:FE9A:4CA2
```

而

```
0:0:0:0:0:0:0:0
```

则可以表示成：

```
::
```

IPv6 的地址空间也像 IPv4 地址一样被划分成不同的用途。一部分地址已经被分配出去，还有很大一部分没有定义。这里我们就不详细介绍了，你可以自己去查相关的资料了解详细信息。目前 Internet 使用的还是 IPv4 协议。

IP 地址与电话号码的使用有一些不同。电话机是哑终端，电话号码在由电信管理局在他们的交换设备中设置。而 IP 地址则需要用户在所用的计算机上设置。也就是说需要计算机来宣称自己所用的号码。

2. 网络配置

做好一切硬件准备以及 IP 地址的知识储备之后，我们就可以通过操作系统提供的网络设置工具来进行网络配置工作了。我们仍然以 Fedora 操作系统为例来说明配置过程。在 Fedora 图形窗口的菜单栏下，选择“桌面”→“管理”→“网络”，或者在终端文本窗口中输入 system-config-network 就可以看到如图 8-12 所示的网络配置窗口（这需要使用管理员账号）。在这个窗口中可以看到 5 个标签，分别是设备、硬件、IPSec、DNS 和主机。

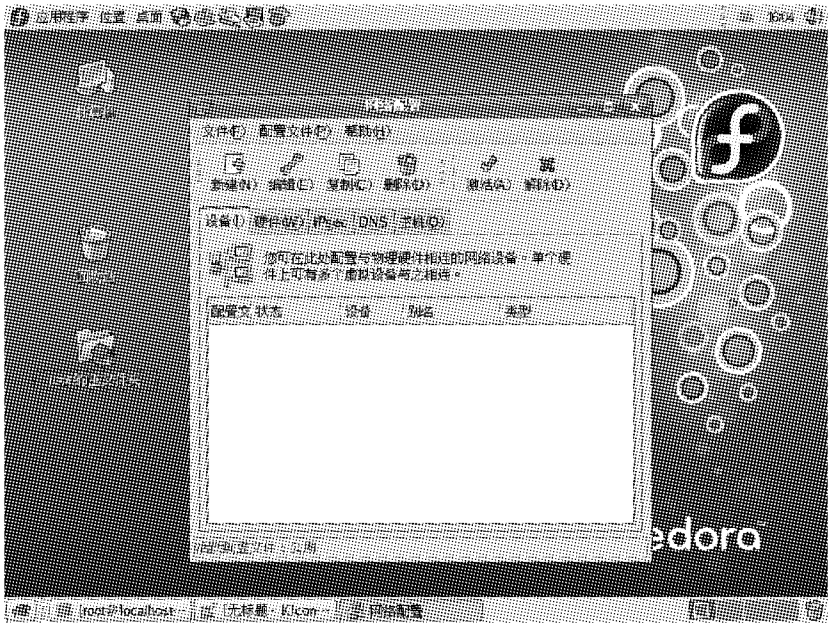


图 8-12　网络配置主界面

从标签中间的提示可以看出，“设备”标签这一项让我们配置与物理硬件相连的网络设

备。这里配置的网络设备是虚拟设备。多个虚拟设备可以与一个硬件(即物理网卡)相连。下面来创建一个网络设备。选中“设备”标签,单击“新建”按钮,会看到如图 8-13 所示的设备类型选择界面。在设备类型中选定“以太网连接”,然后单击“前进”按钮进入如图 8-14 所示的设备选择界面。设备选择界面中列出了本机已经安装的以太网卡。从图上可以看到计算机上一共安装了两块以太网卡。选定其中的一块以太网卡后,单击“前进”按钮进入网络设置界面,如图 8-15 所示。

图 8-13 设备类型选择

图 8-14 设备选择界面

图 8-15　网络配置界面

每台计算机必须有一个 IP 地址。这样网络上的其他计算机才能够把它和别的计算机区分开，才能把数据报发送给它。因此，要为计算机的网络接口配置 IP 地址。一般一个网络接口至少要有一个 IP 地址，而且该 IP 地址不能与别人重复。图 8-15 的网络配置就是让我们为网卡的网络接口配置 IP 地址。从图中可以看出有两种配置 IP 地址的方式。一种是手工配置，另一种是自动获取 IP 地址。

在图 8-15 所示的配置界面中选定“静态设置的 IP 地址”项，手工设置 IP 地址相关的栏目就由灰转亮。我们就进入 IP 地址的手工配置过程。在配置之前，需要先向所在网络的管理员申请这个网段内一个没有使用过的 IP 地址。然后把这个地址在地址一栏中填入，并填上本网络的掩码，如图 8-16 所示。

手工设置 IP 地址的第三项要求给出默认网关的 IP 地址。什么是默认网关呢？网关是 gateway 的中译名。通常网关是一个网络路由设备，是一个网络的入口，或者说是出口。它知道怎样把数据包传送到其他网络。通常我们的计算机也会知道一些路由，但是内容非常少。它并不知道到达 Internet 中每一个接口地址的全部路由。默认网关是与我们计算机在同一子网的网关，也就是本网络的出入口。当我们的计算机不知道应该用什么路由来发送数据包时，就可以把数据包交给默认网关，由它来转发数据包。因为在同一个子网，所以默认网关地址通常与我们计算机的 IP 地址非常像，多数的位是相同的。但是，默认网关的地址不会与计算机的 IP 地址相同。同样默认网关的地址要从网络管理员那里获得。

配置完这些内容之后，单击“前进”按钮出现“创建以太网设备”窗口，如图 8-17 所示。这个窗口中列出了我们前面进行的一些设置。如果没有问题，那么就可以单击“应用”按钮完成网络设备的创建过程。

单击“应用”按钮之后，网络设备就创建成功了，从网络配置主界面的“设备”标签中能够

图 8-16　配置地址及默认网关

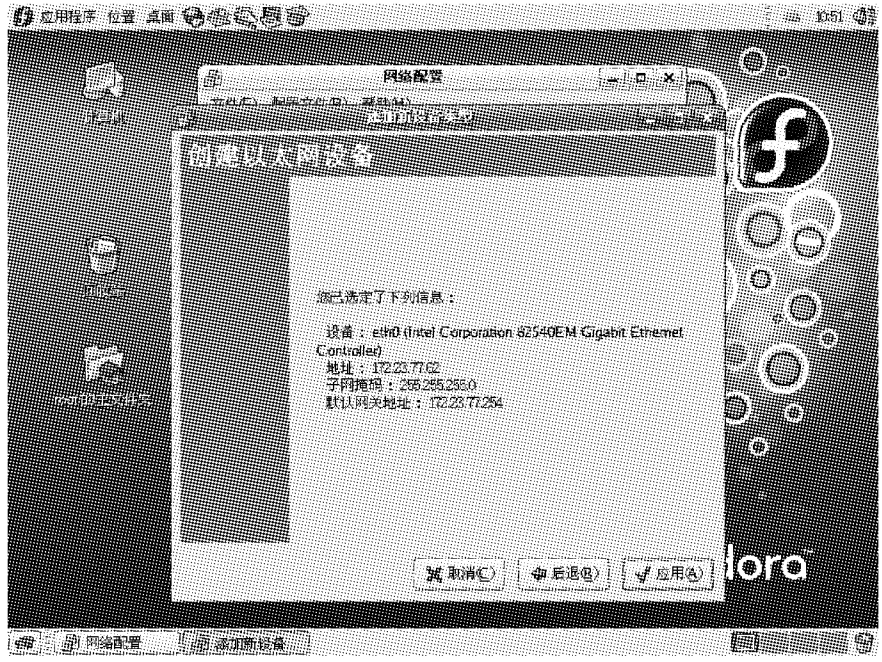


图 8-17　配置完成界面

看到出现一个叫做 eth0 的设备，如图 8-18 所示。不过，这个设备的状态是“不活跃”，还需要选中它，然后单击“激活”按钮来激活这个网络设备。设备的状态转为活跃，如图 8-19 所示，这时计算机就可以通过这个网络设备上网了。

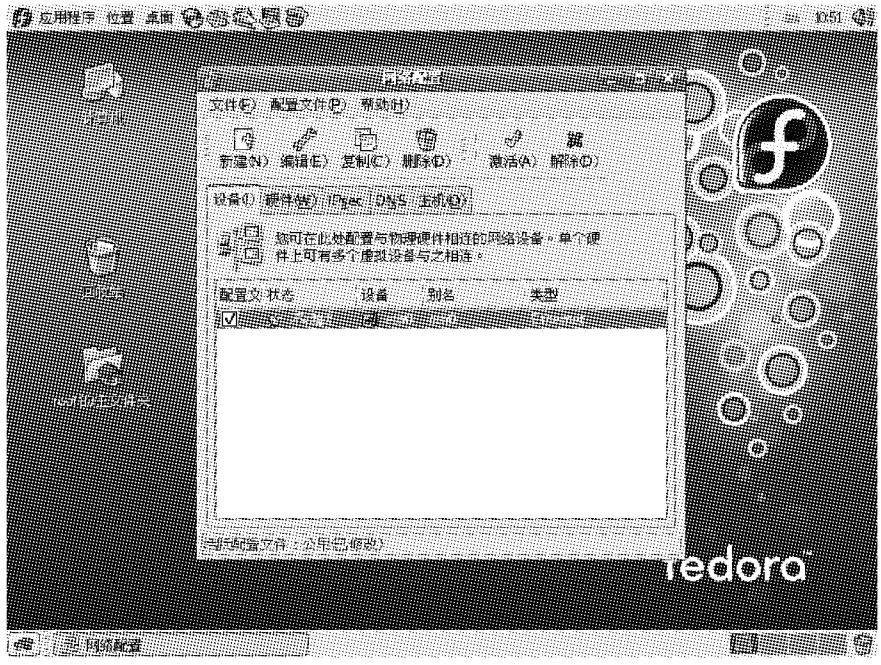


图 8-18　网络设备添加成功

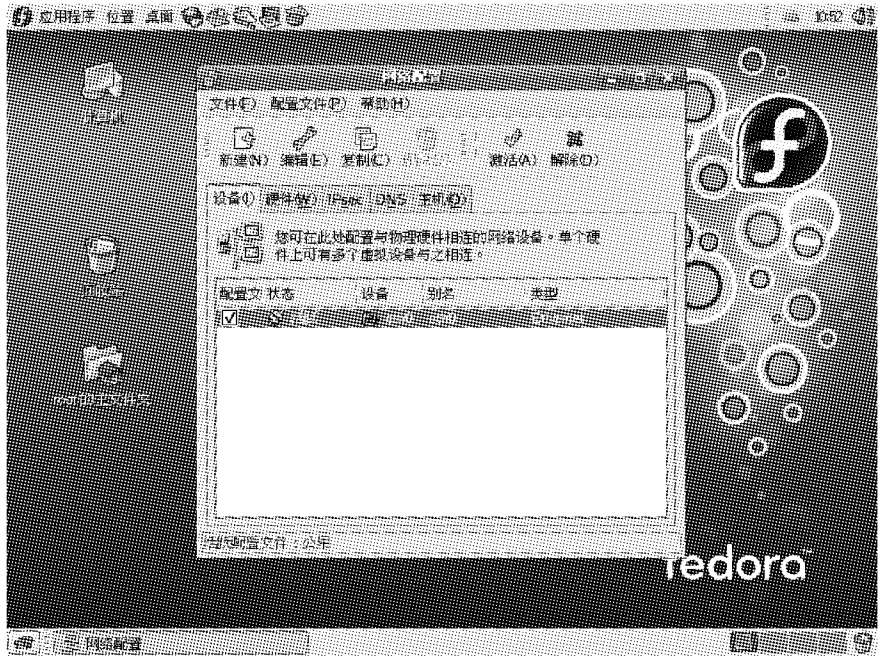


图 8-19　网络设备激活

我们可以在终端命令窗口用命令 ifconfig 来查看网络接口的配置情况，如图 8-20 所示。在图上可以看到以太网接口 eth0 的情况。它的 IP 地址正是我们配置的 172.23.77.62。Hwaddr 表示硬件地址，即为这个接口选择的以太网卡的 MAC 地址(每个以太网卡都有一

个号码，称为 MAC 地址）。MAC 地址的长度为 48 位二进制数（6 个字节）。通常表示为 12 个十六进制数，每 2 个十六进制数之间用冒号隔开。从图 8-20 可以看到这块网卡的硬件地址是 08:00:74:19:CF:BE。其中前 6 位十六进制数 08:00:74 代表网络硬件制造商的编号，是由 IEEE（电气与电子工程师协会）分配的。而后 3 位十六进制数 19:CF:BE 代表该制造商所制造的某个网络产品（如网卡）的系列号。从图上我们还可以看到这个网络接口一些统计信息数，如已经接收和发送的数据包数、字节数、错误的数据包数等。其实，ifconfig 这个命令不只是能够显示网络接口的信息。它能做的事情还有好多，如配置接口的 IP 地址、掩码等。怎样使用这个命令你可以用 man 命令去查它的手册，或者到网上去找一些例子。

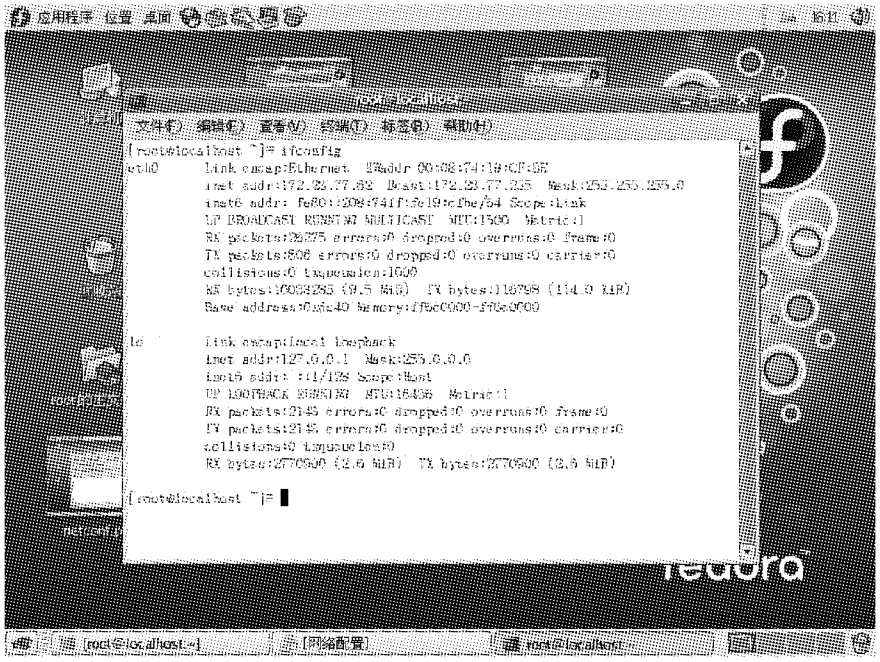


图 8-20　使用 ifconfig 命令

可以用 Ping 命令来测试网络的连通性。Ping 是一条常用命令，它可以测试网络的联通性。它通过向计算机发送某种特殊的报文并且监听报文的返回，来测试与其他计算机的网络连接。现在我们用命令 ping 来测试计算机与另外一台地址为 172.23.77.71 的计算机的网络联通性。结果如图 8-21 所示。

图 8-21 显示收到了从 172.23.77.71 发回的应答。这表明我们的网络从物理到逻辑上确实都连通了。可是，这时如果我们运行浏览器，在地址栏中输入 www.bjut.edu.cn 并不能够访问这个网页。这是为什么呢？这是因为我们的计算机并不知道 www.bjut.edu.cn 在哪里？在网络中是根据数据包中的目的主机 IP 地址来传递数据包的。而我们给出的 www.bjut.edu.cn 并不是 IP 地址，而是被称为主机名。用户的计算机需要把这个名字对应主机的 IP 地址写到数据包中。为了知道这个主机的 IP 地址，需要先告诉计算机一个运行着称为 DNS 服务器的主机的地址。

图 8-21　ping 命令的使用

DNS 是“域名系统”英文的缩写。域名是什么呢？先来看个生活中的例子。北京工业大学位于朝阳区平乐园 100 号。因此，当我们给北京工业大学的同学写信时，信封上的通信地址应该是“朝阳区平乐园 100 号”。可是通常直接写“北京工业大学”也可以。而且人们更愿意这样写，因为这样更好记一些。如果北京工业大学搬迁到海淀区，那么我们仍然可以使用“北京工业大学”这个地址。邮局会知道北京工业大学不在“朝阳区平乐园 100 号”而在一个新的地址。也就是说，使用名字比地址更方便一些。因此，在计算机系统中，也采用了这种方式。除了 IP 地址外，一些计算机还会使用一个名字，特别是一些服务器。这个名字被称为主机名。例如，北京工业大学的主页服务器所在计算机的名字就是 www. bjut. edu. cn。如果要访问北京工业大学主页可以直接用它的名字，而不是 IP 地址 220. 192. 0. 130。

但是，网络设备投递数据包时使用的是主机的 IP 地址，而不是名字。所以，计算机必须把用户给出的目标主机的名字转换成 IP 地址。这种名字与 IP 地址的对应关系记录在网络中一群称为 DNS 服务器的计算机上。所以，当使用名字访问计算机时，系统必须先找到 DNS 服务器把名字转换成 IP 地址。就好像邮局的邮递员看到寄往“北京工业大学”的信件时，必须先找到记录“北京工业大学”与具体通信地址对应的本子，然后在其中查找到具体地址后，才能投递信件一样。邮递员的本子就是 DNS 服务器。不同的是，网络设备并不去查这个本子，而是要求用户计算机来查。为了让计算机能够找到 DNS 服务器，我们需要把 DNS 服务器的地址告诉计算机。也就是配置 DNS 服务器。通常每个局域网中都会有这种服务器，可以向网络管理员询问这些服务器的地址。在图 8-12 中我们看到就有一个 DNS 的标签。选择这个标签，进入如图 8-22 所示的 DNS 配置界面。从图上可以看出系统允许我们设置三个不同 DNS 服务器的地址。系统在解析名字时会依次试着联系这三个服务器，

直到成功获得 IP 地址为止。关于域名系统后面还会详细介绍。配置好 DNS 服务器之后，退出配置程序，然后再用浏览器访问 www. bjut. edu. cn。这一次就会成功看到北京工业大学的主页了。本章后面很快就会详细介绍 DNS 是怎么一回事。

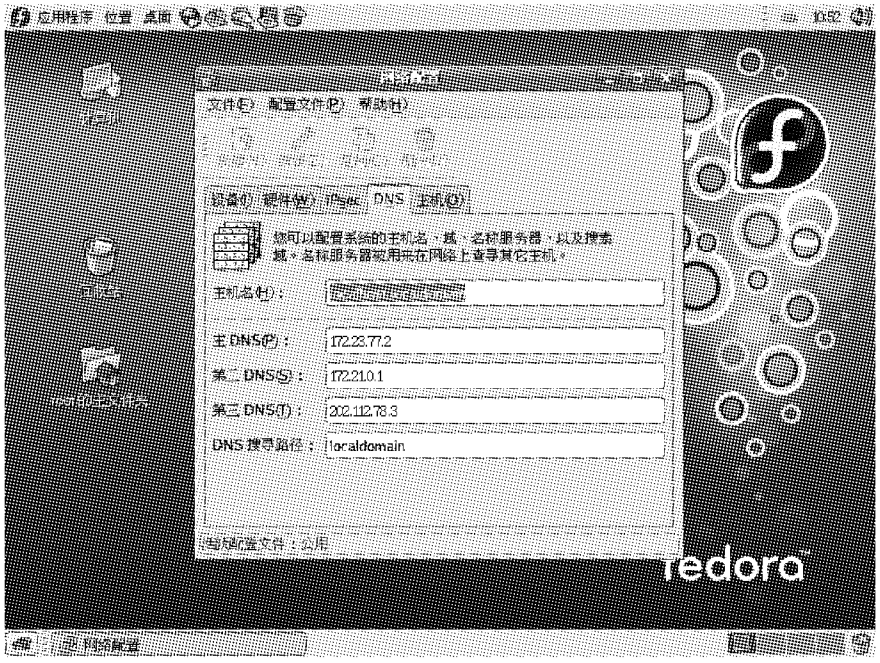


图 8-22　DNS 配置界面

除了手工配置之外，还有另外一种更简单的网络接口 IP 地址配置方式。那就是“自动获取 IP 地址”。采用这种方式不需要向管理员申请 IP 地址，而是由计算机通过一些协议自己到网络上的一些服务器上申请 IP 地址来使用。例如，我们可以在图 8-19 所示的界面中，选中已经创建好的这个 eth0 设备。然后，单击“编辑”按钮。进入与图 8-23 所示的网络配置界面。这一次我们选中“自动获取 IP 地址设置使用”项。选中之后，在后面的下拉框中可以看到三个选项：dhcp、bootp、拨号。这是三种可以获得 IP 地址的方式。选择哪一项需要根据网络中提供的服务来决定，所以要询问网络的管理员。通常使用 dhcp 的网络比较多。我们在后面会介绍 DHCP 到底是什么。至于 Bootp 和拨号这两个选项意味着什么，需要自己去查资料了解。在图上还可以看到一个“自动从提供商处获取 DNS”复选框。选上这一项之后，我们连 DNS 也不用设置了，计算机会自动从网络中获取这些信息。

自动获取 IP 地址很简单。选好之后，就不用管其他的事情了。每次网络启动时，计算机就会自己向相应的服务器申请一个可用的 IP 地址。不过，这种功能是需要有 DHCP 服务支持才能得以生效的，服务器每次分配的 IP 地址可能不同，所以如果你的计算机是用作某种服务的服务器，那么这样经常改变地址，就会导致用户找不到它。因此，对于这种情况最好申请一个固定的 IP 地址，然后用手工配置。

图 8-23　自动配置 IP 地址

8.3　网络基础支持服务

在网络中，除了 Web、电子邮件这样为满足用户的信息获取或传递需求而提供的服务之外，还有另外一类服务。它们是为了方便用户使用网络而设置的一些基础服务。这些服务可以看作是网络基础设施的一部分。这些服务包括动态地址配置服务(DHCP 服务)、名字服务(DNS 系统)、NAT 服务(网络地址转换)等。这一节我们就来介绍这些服务中的 DHCP 服务和名字服务，来看看它们是怎样方便用户使用网络的。计算机网络平台实际上就是在这些基础服务的支持下帮助用户访问网络的。另外，我们把网络管理也放在这一节中介绍，因为它也是网络平台正常运行的基础。

8.3.1　DHCP 服务

DHCP(Dynamic Host Configuration Protocol，动态主机配置协议)我们已经不陌生了。在前面章节曾经多次提到过，只不过都是一带而过。这一节就来详细介绍 DHCP 到底是个什么样的服务。

从字面上看，DHCP 的意思是动态主机配置协议。它可以为用户主机提供动态的 IP 地址、掩码、默认网关地址以及 DNS 服务器的 IP 地址等配置信息。有了 DHCP，用户就可以不用申请固定的 IP 地址，也不用向管理员询问那些必需的网络配置信息了。

DHCP 服务由 DHCP 服务器和客户端协同提供。通常拥有局域网的组织或者网络运行提供商会在自己的网络中设置一个或多个 DHCP 服务器。这些组织或者运营商先申请到一些 IP 地址，然后交给 DHCP 服务器，让它分配给网络中的计算机(手工配置时，这个活

由管理员来完成)。每个 DHCP 服务器被设置可以分配一定范围内的 IP 地址。当网络中的主机需要上网时,它们的 DHCP 客户端会向服务器申请租用并获得一个 IP 地址。这些主机关机之后,它们的 IP 地址就被收回,可以重新分配给别的计算机使用。除了 IP 地址,用户还可以从 DHCP 服务器获得其他必需的网络配置信息。

有了 DHCP,用户配置计算机就方便多了,尤其是不熟悉计算机的用户,再也不用考虑什么 IP 地址、掩码、DNS 服务器,只要选择"通过 DHCP 自动获得"就全解决了。DHCP 还可以提高 IP 地址的利用率。因为关机的用户不再占有 IP 地址了。因此,对于 IP 地址有限的网络,或者规模比较大、用户又不固定的网络来说,DHCP 服务是一个非常好的解决方案。

那么 DHCP 服务是怎样工作的呢? 首先,要使用 DHCP 服务,主机必须被配置为通过 DHCP 获得 IP 地址。这样,当这台计算机启动时,它会检查自己有没有得到一个 IP 地址。如果没有,它就在所属网络中广播一条 DHCP 发现消息来寻找 DHCP 服务器(使用本地网络广播地址 255.255.255.255),并在消息中带有自己的 MAC 地址。网络中的 DHCP 服务器收到这个消息后,在自己管辖的地址范围内找一个还没有分给其他主机使用的 IP 地址,然后把这个地址,以及能够使用的时间长度、掩码放在一个应答消息中广播出去(因为请求的主机还没有 IP 地址,所以只能广播)。应答消息中包含请求主机的 MAC 地址。请求主机收到应答消息后,广播一条 DHCP 请求消息告诉 DHCP 服务器它接受这台 DHCP 服务器分配给它的 IP 地址。DHCP 服务器收到广播的 DHCP 请求消息后,会再发回一个确认,并在确认中捎带其他必需的配置信息,如 DNS 服务器地址等。

如果网络中不只有一个 DHCP 服务器,那么这些服务器都会收到主机广播的 DHCP 发现消息。这些服务器同样会在自己的地址范围内选择一个 IP 地址应答主机。主机通常会选择使用第一个应答提供的 IP 地址。DHCP 服务器都会收到主机广播的接受 IP 地址的 DHCP 请求消息。如果 DHCP 服务器发现主机接受的不是自己分配的 IP 地址,那么它就把刚才准备分给该机的地址重新拿出来分配给其他需要的主机。

小提示

DHCP 服务可以为用户主机提供动态的 IP 地址、掩码、默认网关地址以及 DNS 服务器的 IP 地址等网络配置信息。

DHCP 服务可以提高 IP 地址的利用率,方便用户配置计算机。

DHCP 服务器分配的 IP 地址有一个租用期限。到期之后,主机必须停止使用租用的 IP 地址。一般在到达租用期的一半时,主机就会向其租用 IP 地址的 DHCP 服务器发送请求,请求更新并延长租用期限。

DHCP 也可以把 IP 地址固定分配给特定的主机。这就如同在主机上进行配置一样,只不过这些配置信息是在 DHCP 服务器上,而不是在用户主机上。通常 DHCP 在配置中把一个 IP 地址和一个 MAC 地址固定映射在一起。当使用该 MAC 地址的主机请求 IP 地址时,DHCP 服务器就把相应的 IP 地址分配给它。这种方式被称为手工指定 IP 地址。这种方式适合为一些服务器分配 IP 地址。另外,DHCP 服务器也支持把一群 IP 地址一对一、固定地分配给一群主机。这种方式与手工方式一样:每台主机一旦成功从 DHCP 服务器获

得 IP 地址后，就会永久使用。不同的是，这种 IP 地址和主机的映射关系不是由管理员手工配置的，而是服务器在第一次分配时随机确定下来的，在以后的分配中就保持不变了。这种方式称为自动指定 IP 地址。而前面提到的 IP 地址可以重新回收利用的方式，称为动态指定 IP 地址。

小提示

DHCP 可以支持手工指定 IP 地址、自动指定 IP 地址以及动态指定 IP 地址三种方式。

8.3.2　域名系统

现在多数网络用户上网的目的是要获取某种服务，如浏览信息、下载文件等。他们通常不关心这些服务是由哪台主机提供的，只要能够得到服务就行。因此用户访问服务主机时，更愿意说“我要联系提供某某服务的那台计算机”。如果让他们说“连接 IP 地址为某串数字的那台计算机”，他们会感觉很不方便而且很奇怪。因为 IP 地址标识的是一台物理计算机（确切地说，是主机的一个网络接口），它不会对主机的服务功能给出任何暗示。而且 IP 地址也不是很好记，也不太好写。当服务从一台主机转移到另外一台主机，或者主机的 IP 地址发生变化时，用户必须使用新的 IP 地址才能得到服务。因此，人们就想如果能够用更好记的名字，而不是 IP 地址，来指明要访问的服务主机就好了。这样主机的名字就可以与主机所在的物理网络之间没有关系，而只与提供的服务相对应。用户在访问服务时可以很容易地写出目标主机的名字。如果一台主机提供多种服务，那么这台主机可以取多个名字。用户访问不同的服务时可以使用不同的名字来连接这台主机。当服务从一台主机转移到另外一台上时，也只要把主机的名字更改一下，而用户完全可以不知道这件事情，仍然能够使用原来的计算机名找到该服务。这就好像我们使用“北京工业大学”而不是“朝阳区平乐园 100 号”作为寄信地址一样。因为“北京工业大学”比“朝阳区平乐园 100 号”记起来要容易得多，而且意思也更明显。邮局会把寄往“北京工业大学”的信件投递到“朝阳区平乐园 100 号”。当北京工业大学搬到别的地方时，我们仍然可以使用“北京工业大学”作为地址，而邮局会把信件投递到北京工业大学新址。

小提示

主机名可以与主机提供的服务相对应，它比 IP 地址更容易记，因此，用户用主机名访问服务更方便。

服务器替换主机时，可以保持原主机名，这样用户仍然可以使用原来的主机名访问服务。

既然用名字而不是 IP 地址来指明要访问的主机比较好，那么我们就决定采用这个方案。不过，要先想想实现这件事情，必须做哪些工作。首先，主机的名字是用来在网络中标识主机的，因此必须保证不同主机的名字是不重复的。其次，网络中的设备是使用 IP 地址，而不是主机名，来决定数据包在网络中的投递路线。这样数据包必须带有 IP 地址。因此，必须提供一种办法把用户提供的主机名转换成该主机对应的 IP 地址。我们把对使用主机名的支持，称为名字服务。为了支持名字服务，网络中必须有相应的系统来完成上述两项

工作。

为了保证主机名不重复，可以要求每个主机名在使用之前必须先注册。注册的主要任务是查询这个主机名是否已被使用。这需要把全世界所有主机名都放在一个数据库中，然后在这个库中查找是否已经存在要启用的名字。如果没有重名，就可以使用这个新主机名，并把它加入到数据库中，这样其他的主机就不能再使用这个名字了；否则，注册失败，用户必须为其主机重新选一个名字。

至于主机名到 IP 地址的转换，实现起来似乎也很简单。只要在注册成功、把名字记入数据库时，把该主机的 IP 地址也记下来即可。当用户通过名字访问主机时，系统只要以主机名为关键字到数据库中查询，就可以找到其对应的 IP 地址了。

不过，全世界有那么多的计算机，而且一个主机可能会使用多个主机名，这样就会产生成千上万个主机名。如果只用一个中心数据库，这个数据库必然很大。在注册主机名时，查询这个巨大的数据库判别是否重名很费时间。而且，用户在为主机起名字时，要想到一个别人没有用过的名字也是件非常不容易的事。更严重的问题是，当用户每次用名字访问主机时，计算机都要来查询这个数据库以找到对应的 IP 地址。全世界每时每刻有数不清的用户/进程在网络上发送数据包，所以中心数据库的负担之大是可以想象的。

针对这个规模/可扩展性的问题，科学家们沿用以前的策略——分而治之。人们把中心数据库分开，放在不同的计算机上。不过，这里的分开可不是简单地把数据分布到几个小的数据库中就行了。因为那样判别是否重名以及映射 IP 地址时，仍然要查找所有数据库，每个数据库的负担不但没有减少，反而还延长了注册和映射查询时间。因此，要想真正解决问题，数据的分布必须有个讲究，要让用户能够一下就定位到映射记录所在的数据库主机。要想做到这件事，必须从名字的起法上做文章。为此，人们把主机名划分到不同的区域。不同区域的信息记录在不同的数据库中，存放在不同的主机上。主机名所在的区域在其名字上就有体现。这样，计算机看到主机名就可以知道主机名所属的域，然后直接到该域数据库所在的主机上去查询就行，而不用到无关的数据库中去浪费时间了。

人们把可用的名字空间看作一个大的域，用一个句点“.”表示，称为根域。然后把这个域再划分成子域。这些域称为顶级域。顶级域的名字由其独有的字符串加上表示其上级域——根域的句点构成，如 cn. 和 edu. 等。顶级域再被划分为子域，子域还可以再被划分。每个子域的完整名字由其独有的一个字符串加上其上级域的名字构成。两部分之间用句点分开。如 edu. cn. 和 bjut. edu. cn. 。这种域名被称为绝对域名。而类似 edu 的名字被称为相对域名。每个主机使用的名字必须属于某个域。我们写域名时常常把域名最后的根域名的“.”省略掉。这种命名方案被称为分层次的、基于域的命名方案。这种方案下的整个域名空间可以用一棵如图 8-24 所示的树表示出来。

这种分层次的命名方案带来的好处是，人们可以很容易地想到一个没有使用过的名字，因为不同域中的域名后缀是不同的，所以只需要在自己的域中查看是否有人用过这个名字就行了。除此之外，分层次的命名也为域名到 IP 地址的映射带来了方便。采用这种方案，每个域可以单独维护自己域内名字的数据库，由相应的组织来负责。而且每个子域也可以由不同的组织来管理，它们向自己的上级域负责。这样当要把某个域名映射成 IP 地址时，只需要根据名字中的域名后缀到相关的数据库中去查找就行了，而不用漫无目的地去找遍所有的数据库了。

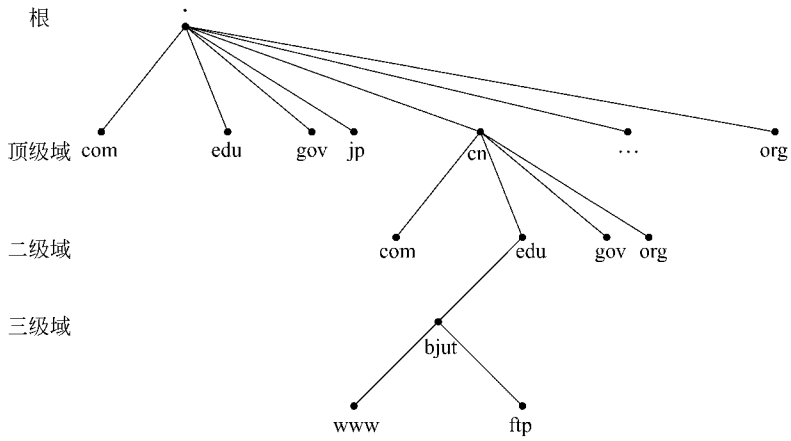


图 8-24　域名空间的树形结构

小提示

要想支持用主机名访问服务，网络中必须提供机制保证主机名的唯一性、实现从主机名到 IP 地址的映射。

为了实现域名信息的分布式管理，人们采用了分层次的、基于域的命名方案。整个域名空间形成一个树形结构。

与 IP 地址一样，全世界的域名由 ICANN 来管理。ICANN 批准了 200 多个顶级域。这些顶级域分为两种类型：通用域和国家域。常见的通用域有 com(商业)、edu(教育)、gov(美国政府)、org(非盈利性组织)、net(网络供应商)等。每个国家有一个国家域，如中国的国家域名为 cn、日本的为 jp。

当然，ICANN 也不直接管理全部域名。它只负责批准顶级域，以及授权顶级域名注册服务商。而申请二级域名的人只要到管理顶级域的组织去申请注册即可。例如，一个公司想为自己注册一个域名 haogongsi. com. ，那么只需到管理 com 域名的服务商或管理机构去注册。如果登记员在 com 域内的数据库中没有查到这个名字，那么这个名字就会被登记在数据库中，这个公司就可以使用这个域名了。当然，这个公司还要支付一些费用。接着，这个公司就可以用一台主机建立一个数据库。从那以后，想让自己的计算机使用类似 xxx. haogongsi. com. 域名的组织或个人就可以到这个公司来登记了。当网络中有用户使用 xxx. haogongsi. com. 访问主机时，系统就到这个公司的这台数据库机来查找名字对应主机的 IP 地址。

采用上述命名方案和分布式数据库实现域名到 IP 地址映射的系统被称为域名系统，即常说的 DNS(Domain Name System)。

现在我们来整理一下域名系统的思路。我们的目的是让用户能够通过名字指明要访问的服务主机。为此，服务主机应该先有一个与众不同的名字。为了达到名字唯一的目的，需要知道所有其他已经在使用的名字，这需要在某个管理机构登记注册所有正在使用的名字。管理机构需要维护一个数据库，里面放着所有正在使用的名字。为了减小名字的冲突域，名字空间被分割成域，每个域中的名字以自己所在域的域名为后缀。这样，只要保证名字在本域中不重复，即可保证在全世界都不重复。因此，就可以不用维护一个完整的中心域名数据

库了。每个域维护自己域的域名数据库，完成域名注册登记的任务。

主机名注册登记之后，用户就可以使用它们来访问其对应的主机了。但是用户以主机名为目标的数据，在进入网络之前必须改用 IP 地址作为目标主机的标识。为此，在名字登记时，数据库中应该同时记录每个名字对应主机的 IP 地址。那些用主机名指明发送目标的数据，在进入网络前，系统需要到主机名注册登记的数据库中去查找该名字对应主机的 IP 地址，然后把得到的 IP 地址作为目标主机地址写入到数据包中。这样数据才能被网络设备投递到目标主机。通过名字查找对应主机 IP 地址的过程称为名字/域名解析。

从上面的描述可以看出，因为主机名和 IP 地址的映射信息存放在网络中某台计算机的数据库中，所以为了实现使用主机名的目标，在用户的计算机上必须有能够发送名字解析请求和接收名字解析结果的程序，而负责登记名字的系统还要能够响应这些用户程序的名字解析请求，提供名字到 IP 地址的映射服务。我们把用户计算机上帮助解析名字的程序称为 DNS 解析器。而把负责登记名字并提供名字到 IP 映射的程序称为 DNS 服务器。DNS 解析器实际上是通过与 DNS 服务器建立联系来获得名字解析结果的，即它是 DNS 服务的客户端。所以，DNS 系统是客户/服务器模式的一个分布式系统。

小提示

通过名字查找到对应主机 IP 地址的过程称为名字解析。
在用户计算机上帮助解析名字的程序称为 DNS 解析器。
负责登记名字并提供名字到 IP 映射的程序称为 DNS 服务器。
DNS 系统是客户/服务器模式的一个分布式系统。

DNS 解析器在解析某个名字时，必须找到存有该名字的 DNS 服务器。按照前面的介绍，每个域都有自己的 DNS 服务器，那么解析器怎么能找到要解析名字所在域的 DNS 服务器呢？它是不是要求管理员把所有的域名服务器的地址都配置在计算机中，然后它在其中找到对应域的 DNS 服务器，然后去联系呢？实际上不是这样的。我们在配置计算机时，确实配置了一个或者几个 DNS 服务器的地址，但绝对不是所有域的 DNS 服务器地址(那数量也太大啦，需要准备个数据库了)。实际上，我们只告诉了计算机一个离它比较近的 DNS 服务器的地址，这个 DNS 服务器通常称为本地 DNS 服务器。

当用户要通过名字联系某台主机时，计算机上的名字解析器便开始工作。名字解析的过程如图 8-25 所示。假定计算机 pc1. sina. com 上的浏览器想访问 www. bjut. edu. cn 主机上的网页。pc1. sina. com 主机需要在发送请求之前，用名字解析器把 www. bjut. edu. cn 这台主机的 IP 地址找到。名字解析器接到任务之后，从本机的配置中找到本地 DNS 服务器的地址，然后向其发送一个消息，如图 8-25 中的消息 1，问它是否知道 www. bjut. edu. cn 这台主机的 IP 地址。本地 DNS 服务器查找自己的数据库，发现并没有这个名字的记录。这时它并不直接告诉询问的计算机它没有映射记录，而是向根 DNS 服务器发送一条查询消息，如图 8-25 中的消息 2，问它是否知道 www. bjut. edu. cn 这台主机的 IP 地址。每个 DNS 服务器都预先配置有根 DNS 服务器的地址。全世界一共 13 台根 DNS 服务器。它们多数分布在北美洲。唯一在亚洲的根 DNS 服务器放在了日本。

按照前面介绍的命名和注册机制：每个域 DNS 服务器只登记自己域内的域名记录，因

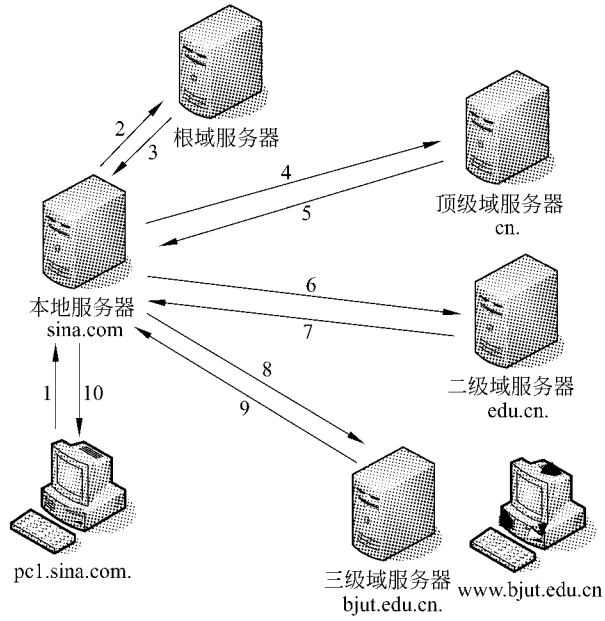


图 8-25　DNS 解析过程

此，这些根 DNS 服务器上面只记录着其下的各个顶级域的记录，其中包含有各个顶级域 DNS 服务器的地址。因此，根 DNS 服务器同样也没有 www. bjut. edu. cn 这个名字的记录。但是，通过查看名字的后缀，根 DNS 服务器发现要查找的名字属于顶级域 cn。根 DNS 服务器存有这个顶级域 DNS 服务器地址，因此它把这个地址发回给询问的本地 DNS 服务器，如图 8-25 中的消息 3。意思是说，我不知道你要找的主机的 IP 地址，你到这个服务器那里问问吧，它或许知道。本地 DNS 服务器得到这个地址后，它便向 cn 域的 DNS 服务器发出询问，即图 8-25 中的消息 4。

顶级域 cn 的 DNS 服务器同样也没有 www. bjut. edu. cn 这个名字的记录。通过后缀，它发现要查找的名字属于自己的子域 edu。因为每个域的 DNS 服务器都存有自己子域的 DNS 服务器的地址。所以，顶级域 cn 的 DNS 服务器把 edu. cn 这个子域的 DNS 服务器的地址发回给询问的本地 DNS 服务器，如图 8-25 中的消息 5，让它到那里去查找。

按照这样的过程，本地 DNS 服务器将从 edu. cn 这个域的 DNS 服务器获得 bjut. edu. cn 域的 DNS 服务器地址，图 8-25 中的消息 6 和消息 7。当本地 DNS 服务器向 bjut. edu. cn 域的 DNS 服务器发出查询时，即图 8-25 中的消息 8，因为这台服务器便是 www. bjut. edu. cn 主机注册的域名服务器，所以服务器在自己的记录中找到主机的 IP 地址交给查询的本地 DNS 服务器，如图 8-25 中的消息 9。本地 DNS 服务器最后将结果交给主机 pc1. sina. com 上的名字解析器，即图 8-25 中的消息 10。至此，名字解析器的工作完成。pc1. sina. com 这台计算机接下来把从名字解析器获得的 IP 地址填写到数据包中，把数据发送到网络中。

从上述查询过程可以看出，名字解析会涉及很多 DNS 服务器，因此会花费很多时间；而且位于树形结构顶端的根域名服务器的负担非常大，因为几乎所有的域名解析都要经过它。为此，DNS 系统采取了一些措施。其中最主要的措施就是缓存机制。DNS 服务器会把解析过程中获得的名字和 IP 地址的映射记录缓存下来。这样，当再有用户需要解析这些名字

时，就不用再去查询了，直接用缓存的记录应答就可以了。这样既提高了名字解析的速度，又减轻了其他 DNS 服务器的负担。因此，对于某个名字来说，它的记录就不再只存在于注册的原始 DNS 服务器上了。网络中的各个 DNS 服务器都可能缓存有它的记录。存有名字最原始记录的那台 DNS 服务器，被称为该名字的权威 DNS 服务器。此外，多数 DNS 服务器都缓存有顶级域 DNS 服务器的地址，这样就可以减少对根域服务器的打扰了。当然，这些缓存的内容要设置一定的生存期，以便在原始名字和 IP 地址信息发生变化时，能够及时更新缓存内容。

小提示

名字解析过程要访问若干 DNS 服务器，所以要花费很多时间。
DNS 系统采用缓存技术减少查询时间以及服务器负担。

8.3.3 网络管理

随着网络规模的扩大，网络故障定位、网络计费、网络安全、网络效率优化等一系列问题都变得复杂起来。再用管理几台主机规模的小型网络的方法，如用 ping 来探测网络故障点，已经行不通了。人们需要专门开发出一套能够监控网络运行状态、提高网络运行效率的管理软件。

要想把网络管理起来，首先必须知道各个设备的状态。因此，在每个设备点，需要有一个监控程序。这些监控程序就像一个摄像头，可以实时获得设备的运行状态信息。获得这些信息之后，网络管理软件需要把它们以合适的方式显示给管理员看。如果管理员要到每个设备点去看监控程序取出的状态信息，对于大一点儿的网络显然很不方便。因此，需要把各个网络监控程序获得的状态信息汇总到一个管理员方便访问的地方。另外，要想把网络管理好，必须把这些原始数据进行加工和分析，得出有意义的数据和决策信息。显然在汇聚信息的地方而不是各个网络设备上来做这些统计和分析工作比较合适。一来可以得到更全面的信息，二来可以减少对被管理设备的干扰。由此，我们形成了一个网络管理系统的模型，如图 8-26 所示。

在所有被管理的设备上，如路由器、服务器，都运行一个很小的软件。这个软件负责获得被管设备的状态。这些软件被称为代理。在网络中的某个位置运行着一台计算机。这台计算机从各个代理获得网络设备的状态，并进行相应的统计、分析，以及显示，或者进行故障告警等事情，则这台计算机被称为管理站。这样，网络管理员就可以通过管理站获得整个网络的运行状态，及时地发现问题，并通过统计结果对网络进行优化。

在图 8-26 的模型中，我们看到最关键的地方是设备上要有代理。只有这样，对于管理站来说，这个设备才是可管理的。网络中的设备类型和厂家各有不同，怎样为这些不同设备编写代理，而这些代理又都能够与管理站通信呢？按照前面的经验，只要我们把管理站和代理通信的协议标准化，那么谁来编写代理和管理站端软件就都没有问题了。因此，人们定义了代理和管理站之间的通信协议——简单网络管理协议（Simple Network Management Protocol，SNMP）。这样每个厂家都可以针对自己的设备编写代理软件并预装在自己的设备中，而另外一些组织则可以专注编写管理站软件。因为都遵循 SNMP 协议，所以这些代

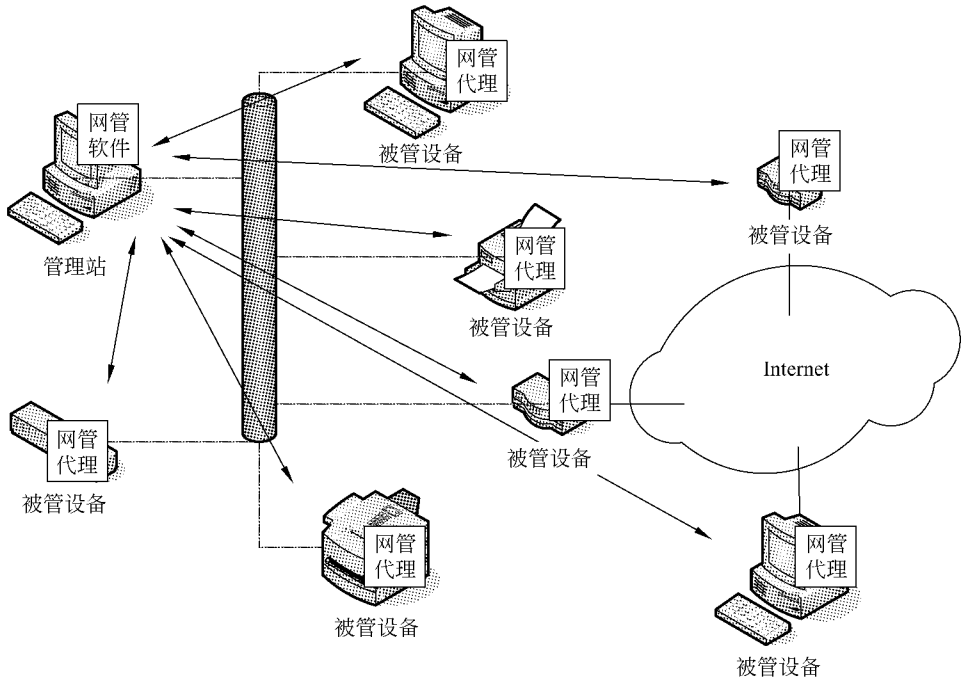


图 8-26　网络管理系统模型

理和管理站软件之间可以互通。

SNMP 协议称为简单网络管理协议，是因为它确实简单。它只定义了代理和管理站通信时要发送的、很少的几种消息类型及其格式，如请求消息、应答消息。SNMP 协议规定管理站可以向代理发送请求消息向其索要信息或者命令代理修改某些信息。代理在一些特殊的情况下，也可以主动向管理站报告信息。

现在我们已经解决了管理站和代理的通信问题。可是，还有另外一个问题。网络中设备的种类非常繁多。它们的状态信息也各不相同。例如，对于路由器来说，它的状态信息可能包括它有多少个网络接口、每个接口的目前状态怎样、收到了多少个数据包、发送了多少字节等。而对于服务器主机，它的状态则可能包括 CPU 的使用状况、目前连接的用户数等。因此，管理站必须能够识别出各种代理发送过来的各种状态信息。那么，管理站依据什么向代理索要信息呢？代理给管理站发来的各种状态信息，管理站又怎样才能都看懂呢？全世界有这么多设备，到底要定义多少种状态和状态值呢？针对这个问题，人们想出了一个很聪明的解决办法。设计者并没有一个一个地去定义被管理的对象及其各种不同的状态。而是定义了一个语法和结构。让用户自己去定义要管理的对象及其状态。定义对象的文件被称为管理信息库(Management Information Base，MIB)。每个 MIB 文件都有固定格式、遵循相同语法。而管理软件和代理都能够识别这种格式和语法，也就是它们都能读懂 MIB 文件。MIB 库中定义的对象都有一个对应的对象标识符。每个对象的标识符在整个网络管理系统中是唯一的。这种对象标识符采用了一种树形的编号规则，如图 8-27 所示。图 8-27 左方窗格中显示出了一个名为 RFC1213-MIB 文件中定义的对象在对象命名树的位置及其名称。图中选定的对象 ifInErrors，在右侧上方对象 ID 文本框 object ID 中显示

的是. iso. org. dod. internet. mgmt. mib-2. interfaces. ifTable. ifEntry. ifInErrors。这与它在树形结构中的位置相对应。在右下方的说明中可以看到这个对象的实际标识符 ID 是 .1.3.6.1.2.1.2.2.1.14。这个 ID 是管理站和代理在通信时真正使用的 ID。而前面用字符串表示的 ID 是为了看起来更方便。从说明中可以看到 ifInErrors 这个对象表示网络接口收到的错误数据包的数目。从图 8-27 中间的结果窗口中可以看到 172.23.77.215 这台设备一共有三个接口，它们收到的数据包错误数都为 0。

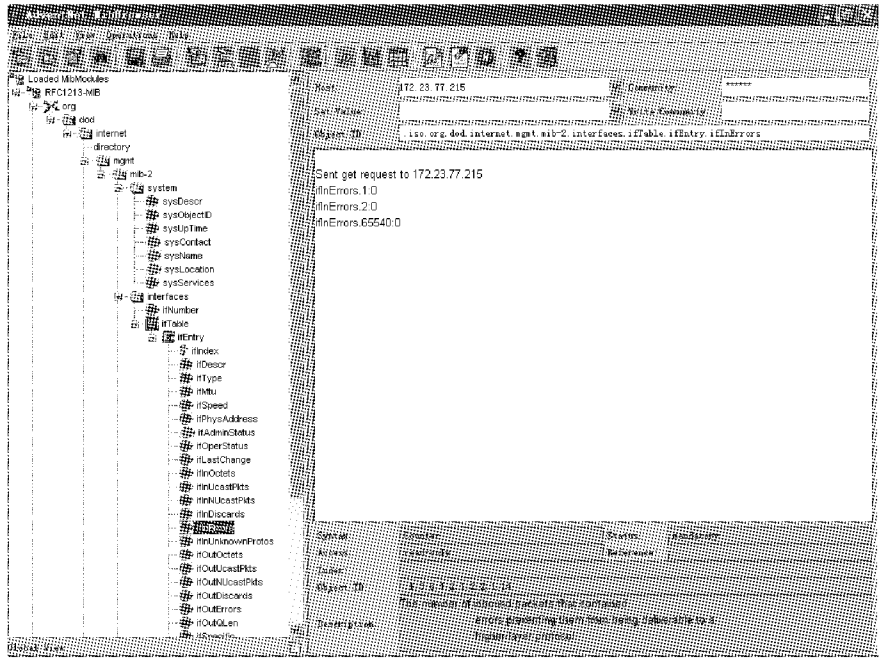


图 8-27　MIB 库浏览器

每种设备的厂家都可以把自己的设备对象定义在 MIB 对象树形结构的适当位置上，形成对象的唯一标识符。然后按照规定的语法和格式来定义自己的 MIB 库。定义好 MIB 库之后，厂家实现自己的设备管理代理。代理中含有获得厂家定义对象的状态值的代码。要想获得这些厂家定义的被管设备的状态，管理站需要加载相应的 MIB 文件，获得这些对象的标识符，然后向设备代理发送请求，索取这些对象的状态值。代理收到管理站索取状态值的请求后，会检查设备该状态的值并应答管理站。这样加载不同的 MIB 库，管理站就可以从不同的代理获得各种不同对象的状态值了。

小提示

网络管理多采用管理站、管理代理模型方式实现。

代理负责监测被管设备的状态，管理站从代理获取被管设备的状态，进行分析和显示等进一步的处理工作。

管理站和管理代理通过 SNMP 协议通信，它们之间通过加载相同的 MIB 库来获知管理对象的标识符。

复习题

1. 网络分层是什么含义？
2. 常见的网络参考模型有哪两个？它们分别把网络分为哪几层？
3. 什么是直通线？什么是交叉线？它们各用在什么场合中？
4. IP 地址是什么？有什么作用？
5. 掩码是做什么用的？怎样表示？
6. 为什么要配置默认网关、DNS 服务器地址？
7. DHCP 指的是什么？
8. 使用 DHCP 服务能带来什么好处？
9. 简述主机通过 DHCP 获得地址的过程。
10. DHCP 支持哪几种地址分配方式？
11. 使用域名有什么好处？
12. 简述 DNS 的命名方案和数据的分布式存储。
13. 什么叫绝对域名？什么叫相对域名？
14. 简述域名在 DNS 系统中的解析过程。
15. 网络管理通常使用什么协议？

讨论

1. 请分析分层的好处。
2. TCP/IP 参考模型为什么能成功？
3. 请到网上查一查，目前有哪些流行的网络管理软件，它们可以提供哪些功能？各有什么优缺点？

实验

将一台 PC 接入到以太网中。步骤如下：

(1) 找到一个网络接入端口。
(2) 安装网卡。
(3) 做一根双绞线，把网卡的端口连接到网络接入端口。
(4) 安装网卡驱动程序。
(5) 配置网络。
(6) 测试网络的联通性。

第五篇　设施与环境平台

第9章　计算机系统基础设施

前面几章的内容告诉我们一个事实，现在的计算机系统已经不再是以孤军奋战的形式为我们工作了。若干年前，用户各自有一台PC，每天看到的、用到的信息和数据都是该机器产生的，或者是通过软盘等介质复制进来的，一切都是在一种对结果相当可预知的感受下进行着。现在不同了，各种计算机通过互联网联系起来，实现了最广泛的信息交流和资源共享。如今当人们面对计算机的时候，从中能得到的信息，来自地球的每一个角落，日新月异。遍布全球的各种计算机及其相关设备和资源已经形成了互联网这样一个最大的服务平台。人们每天正是在这个大平台上进行着各种工作、学习和娱乐活动。构成这个大平台的设备及其相关资源被称之为计算机系统的基础设施。它们就像电力系统那些电厂、变电站，以及电线杆、电线一样，是提供系统(电力)服务的基础。这一章我们就来简要介绍这些基础设施。这其中包括网络中两大类主要的设备——服务器和网络设备，以及放置它们的计算机机房。

9.1　主要设备

前面讲过，随着网络的出现，计算机服务的形式发生了变化，一批计算机的功能也随之产生了变化。购置这些计算机的目的不再是个别用户需要它们，而是要为网络中的其他用户提供服务。这些特殊的计算机可以划分为两种类型。一种以提供信息服务为主要工作，被称为服务器，如Web服务器、DNS服务器。另外一类则专门负责为网络中的其他计算机转发传递数据包。这类计算机通常被称为网络设备。路由器、交换机等都属于这类设备。这些设备虽然本质上还都是计算机，但是因为有特定的使命，因而在设计和性能上都与普通计算机不同。下面分别介绍这两类特殊的计算机。

9.1.1　服务器

服务器这个词，可以指网络中等待其他程序与其联系的应用软件，也可以指在网络环境下运行应用软件、为网上用户提供某种服务的高性能计算机。也就是说它有软、硬件两个不同含义。这一节我们主要关注硬件概念上的服务器。

人们按照计算能力把计算机分为巨型机、中型机、小型机、和微型机(PC)。不过什么是微型机，什么样的计算机算得上是巨型机，具体的标准在不断变化。一个时期的巨型机过了若干年就变成一般的计算机了。例如，20世纪80年代巨型机的标准是运算速度要达到每秒1亿次以上、字长为64位、主存储器的容量要是4～16MB的数字式电子计算机。这个标准在今天看来简直太低了，普通的微机都不止这个水平了。而前些年那些被称为中型机的系统，到现在都被称作小型机了。但不论怎样，无论在何时，小型机、中型机、巨型机总是对应着一个性能递增的次序。

服务器通常要为网络中大量的用户提供服务，所以对它的性能要求也比较高。因此，它都是由性能较强的计算机来担任。小型机、大型机，甚至巨型机都会被用来充当服务器。不

过，现在由于PC性能逐步提高，在要求不很高的场合用PC充当服务器也常见。这里需要说明的是，PC这个概念也分为两种，一是“桌面机”，二是“服务器”，核心都是采用微处理器，但在外部设备配置上（尤其是磁盘访问的能力），以及可管理性方面，后者比较强调；而前者更强调图形显示等用户体验。所谓用PC充当服务器，主要还是指“PC服务器”，尽管用“PC桌面机”有时也能凑合。为了满足性能上的高要求，PC还可以通过集团优势来弥补自己的差距。也就是常说的“三个臭皮匠顶上一个诸葛亮”的道理。常见的做法是，几十台甚至几百台PC通过高速网络联合在一起，以一个服务器名义向外提供服务。目前常用的技术被称为集群技术。通过集群技术产生的系统也就是价廉且灵活的集群系统。

下面从性能划分的角度，按照巨型机、大型机、小型机、集群系统的顺序来简要介绍服务器。然后再从服务的角度概要分析各种服务器的不同。

1. 巨型机

巨型机是最昂贵的计算机，也被称为超级计算机。巨型机的功能是如此强大，价格也是如此昂贵，以至于我们在普通机房中很难见到这种计算机。

巨型机的技术门槛极高，它体现了一个国家的科研和经济发展实力。因而它的研制水平甚至被用来标志一个国家的科学技术和工业发展的程度。

巨型机主要被用来承担重大的科学研究、国防尖端技术和国民经济领域的大型计算课题及数据处理任务，如大范围天气预报、整理卫星照片、原子核的探索、洲际导弹、宇宙飞船研究等。

Blue Gene/L，如图9-1所示，是2007年世界上最快的超级计算机[65]。这是2007年由美国劳伦斯·伯克利国家实验室、田纳西大学和德国曼海姆大学组织的全球超级计算机500强评选的结果。Blue Gene/L由IBM公司、利弗莫尔实验室和美国能源部联合研制。这台超级计算机的峰值速度能够达到360万亿次浮点计算。它的内部有65 536颗处理器，运行Linux操作系统。

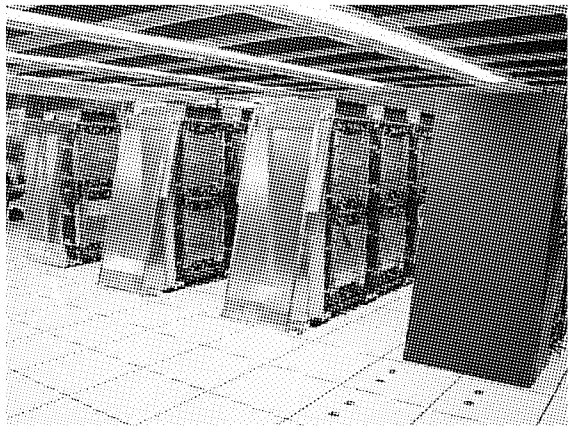


图9-1 超级计算机Blue Gene/L

Blue Gene/L被用来模拟包括蛋白质在内的生物化学过程。人们曾经用Blue Gene/L执行过一项与半个老鼠大脑一样复杂的脑皮层模拟实验。要知道半个老鼠大脑就有800万神经元细胞，每一个细胞与其他神经纤维之间会有8000个链接。

我国也预计很快推出中国百万亿次计算机。中国科学院计算技术研究所和曙光公司研制的十万亿次计算机“曙光 4000A”于 2004 年问世，如图 9-2 所示。曙光 4000A 落户在上海超级计算中心，承担包括国家网格、上海基础科研平台和华东地区信息服务三方面的重任，为华东地区各行各业提供海量信息处理、信息开发服务，以及科学研究高性能计算服务。

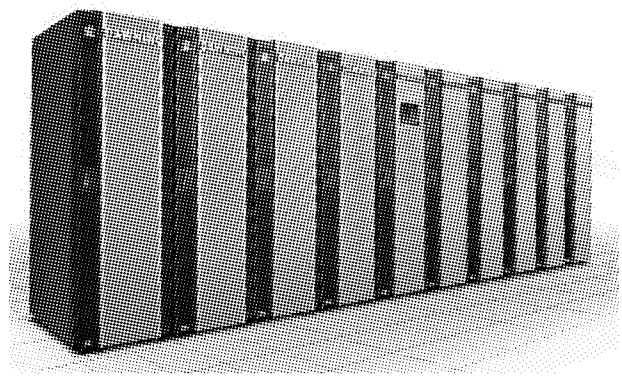


图 9-2　曙光 4000A

现在技术的发展速度非常快。超级计算机的评选机构在 2007 年的评选中就总结到，近半年(2007 年)全球超级计算机整体运算速度的提升，“是近 15 年评选史上最快的一次”。在上一届评比中，超级计算机的浮点运算速度只需达到每秒 2.737 万亿次，就可入围 500 强。仅仅半年后，这一“门槛”就已经提高到了每秒 4.005 万亿次。该次评比中位列第 500 名的超级计算机，在半年前可以排到第 216 名。可以看出，超级计算机技术也正以超级计算机的计算速度在发展。

小提示

巨型机也被称为超级计算机。

巨型机的研制水平标志着一个国家的科学技术和工业发展的程度，体现着国家科研和经济发展的实力。

巨型机主要用来承担重大的科学研究、国防尖端技术和国民经济领域的大型计算课题及数据处理任务。

2. 大型机

按照分级，大型机应该是比巨型机能力稍差一些的一类计算机。不过，IBM 公司在这个档次的计算机市场和技术上占据了绝对优势。因此，很多时候，甚至提起大(型)机就是指 IBM system/360 开始的一系列计算机[66]。实际上，一些公司，如 Amdahl、Hitachi Data Systems (HDS)、EMC、HP、Sun，也都制造大型机，只是市场份额不大而已。

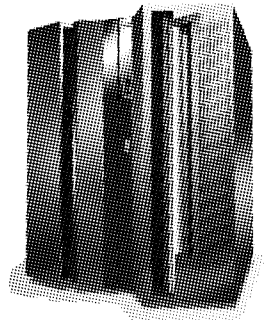


图 9-3　IBM z9

图 9-3 是 IBM 公司的大型机 z9。它是 5 种型号 IBM zSeries 机器之一。这个系列的大型机型号还有 800，890，900 和 990。z9 是 IBM 公司投入 12 亿美元，经过三年研发，在 2005 年夏季推出的。该机可以虚拟化数百种软件应用程序，每天可以处理

10亿笔交易，还可以按需扩容。它使用16芯片的IBM Multichip Module作为处理器。z9可以运行多个操作系统，包括z/OS、z/OS. e、OS/390、Linux on zSeries、z/VM、VSE/ESA、zVSE和TPF。所有的主流数据库和企业事务处理环境均可在该大型机上运行，它们包括CICS、IMS、WebSphere Application Server、DB2和Oracle。从上面的描述可以看出大型机的功能确实比较强。

大型机与平常见到的计算机(微机)有很大不同。大型机通常使用专门为其量身定做的专用软件，包括操作系统、编译系统以及其他系统软件和上层应用软件。例如，IBM公司专门为大型机设计开发了操作系统z/OS。前面提到的z9上可以运行的操作系统，多数也是人们不熟悉的。大型机上的程序设计也与人们常接触的有很大不同。例如，在程序设计语言不断更新的现在，IBM的大型机仍然使用一个名为COBOL，在计算机界应该算很古老的程序设计语言来开发程序。这种封闭的系统模式为大型机系统带来了高可靠性和安全性[67]。但是同时也让人们对大型机的世界很陌生，因为很少接触到大型机，而平常接触到的软、硬件与大型机又不同。

由于成本巨大，大型机系统的用户一般是一些对信息的安全性和稳定性要求很高的企业和单位，通常以政府、银行、保险公司和大型制造企业为主。大型机在这些大公司和企业中占据着重要地位。例如，美国的银行中2/3的事务处理运行在大型机上。

小提示

大型机通常使用专门为其开发的软件，包括操作系统、编译系统，以及其他系统软件和上层应用软件。

大型机系统的用户一般以政府、银行、保险公司和大型制造企业为主。

3. 小型机

现在很多机房中运行的是被称为小型机的系统。具体什么是小型机也没有明确的说法和分类标准。以前小型机常常指采用RISC结构专用处理器、支持UNIX操作系统的、封闭专用的计算机系统。因而以前小型机也被称RISC服务器或UNIX服务器[68]。

RISC(Reduced Instruction Set Computer，精简指令集计算机)是一种CPU的设计模式。与其相对的是CISC(Complex Instruction Set Computer，复杂指令集计算机)。那么，RISC是什么意思呢？我们知道每个CPU都有一个固定的指令集。这些指令通过CPU硬件直接完成。如果想让计算机完成一个复杂的任务，通常有两种实现方式。一是设计一条机器指令，用CPU硬件直接完成。二是通过软件编程用CPU现有的若干条指令完成。通常采用硬件指令要快于软件编程的方式。为此，在计算机的设计中，人们不断地加入新的、复杂的指令来加强计算机的功能。为了支持这些新增复杂指令，CPU的设计也变得越来越复杂。可是，后来人们发现，在CPU庞大的指令集中，只有约20%的指令会被经常使用。也就是说，计算机系统复杂的结构实际上效率并不是很高。人们费劲气力增加的、完成特定复杂指令的处理器硬件实际上很少被用到。为此，人们提出了减少指令集的想法，即RISC。这样与其相对的，具有复杂长指令的CPU就被称为CISC处理器。RISC通过减少指令条数来简化CPU设计、提高CPU效率。在IBM公司成功开发出RISC系统之后，RISC技术就成为设计超级计算机普遍采用的模式。常见的RISC处理器有AVR、PIC、ARM、DEC

Alpha、PA-RISC、SPARC、MIPS、Power 架构等[72]。这些 CPU 被用在大到各种超级计算机、工作站、高阶服务器，小到各类嵌入式设备、家用游戏机、消费电子产品、工业控制计算机上。不过这些都离普通计算机用户比较远，所以知道这些处理器的人也比较少。而 CISC 结构的典型代表就是 Intel 的 CPU，它被广泛地用在 PC 领域。由于 PC 的普及，所以人们对这种 CPU 非常熟悉。

实际上，RISC 和 CISC 各有所长。CISC 结构 CPU 有着丰富的指令集合，使计算机利用一两条指令就能执行非常复杂的操作，而 RISC 机器则可能用很长的程序才能完成。但是，这也使 CISC 结构 CPU 设计复杂、成本高。相反 RISC 结构的 CPU 执行效率很高、芯片功耗也低，进而形成较低的制造成本。

随着技术的进步，RISC 与 CISC 机型的差距也在不断减小。很多小型机开始采用 CISC 结构的 CPU 来实现。从全球高性能计算机 500 强排行榜来看，使用 AMD、Intel 的 CISC 处理器的计算机从 2001 年 11 月份的 4%上升到了 2005 年 11 月的 77.6%，而使用 RISC 处理器的计算机则从 96%下降到了 22.4%[70]。

因此，现在人们在讲小型机时，不再专指 RISC 结构的机器了。现在人们认为那些采用 8～32 颗处理器，性能和价格介于 PC 服务器和大型主机之间的一种高性能 64 位计算机就是小型机[69]。

图 9-4 是我国曙光公司生产的第二代小型机——天演 EP850。这台计算机采用了 8 个双核 CPU。这些 CPU 就是 CISC 结构的。这台小型机有 128GB 的海量内存，还有专用高速的 I/O 通道。它支持 Linux、Solaris 等多种操作系统[71]。

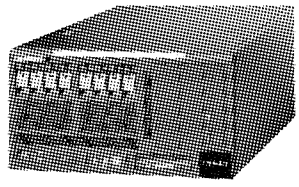


图 9-4　天演 EP850

小提示

RISC 和 CISC 是 CPU 设计的两种模式。
早期小型机以 RISC 结构居多，现在 CISC 结构机型在增多。

4. 集群系统

网络的发展使各种服务器的用户量猛增。这要求人们必须提升服务器的性能，用能力更强的计算机。但是，小型机、大型机的价格非常高，巨型机则更是昂贵。在这种情况下，将多台计算机组织起来模拟一台功能强大的计算机的集群技术应运而生，并且迅速地被接受和推广。

集群其实并不是一个新概念。在 20 世纪 70 年代计算机厂商和研究机构就开始了对集群系统的研究和开发。不过，那时候的集群系统主要是用在科学工程计算上，离普通用户比较远，因而大家也不知道这件事。这几年，网络服务器性能需求的提高以及 Linux 集群技术的出现让集群系统成为网络服务广泛使用的硬件基础，使集群这个词越来越多地出现在大众面前[73]。

集群系统是一组计算机。不过，它们是作为一个整体向用户提供服务。这里的服务可以是网络共享资料，也可以是计算能力。在集群系统中单个的计算机被称为集群的节点。集群在用户看来就是一个系统，而不是多个计算机。用户不会意识到集群系统底层的节点，而且集群系统的节点可以由管理员随时增加和删改。其实在介绍 Web 服务时，已经提到过

集群这种形式了，只不过没讲集群这个术语。我们那时说，多个计算机一起工作，为用户提供 Web 服务；有一个前端机负责将用户的请求分配给各个工作的计算机；在用户看来，只有一台服务器在为它服务。

总的来说，集群技术可以采用相对较低的成本来提高系统的性能。它可以提高系统的计算能力、可靠性以及可扩展性。集群技术实际上有很多种。它们针对不同的应用，有各自侧重的设计目标。通常人们按照计算能力、可靠性以及可扩展性三个目标把集群技术分为三类。专注于提高系统计算能力的集群技术被称为高性能计算集群技术。它是出现较早的一种集群技术，用在被称为并行计算的领域里。而以提高系统的可用性或者说容错性为目标的集群技术被称为高可用性集群。它应该算是容错技术研究的产物。而使集群这个词为人们所熟悉的是以提高系统可扩展性为目标的集群技术。这种集群技术也被称为负载均衡集群技术。它被广泛地使用在网络服务器的架设中。

高性能计算集群技术的英文是 High Performance Computing Cluster，简称 HPC Cluster。这种集群系统通常要将一个应用程序分割成多块可以并行执行的部分并指定到多个处理器上去执行。这个过程也被称为并行处理。因而 HPC Cluster 也被称为并行计算集群。HPC Cluster 向用户提供一个单一计算机的界面。前置计算机负责与用户交互，并在接受用户提交的计算任务后通过调度程序将任务分配给各个计算节点执行，运行结束后通过前置计算机将结果返回给用户。程序运行过程中的进程间通信通过专用网络进行。这种集群可以获得很高的计算性能。但是，通常构成这样集群的成本要比一台同等计算能力的大型机或巨型机价格要低很多。

高可用性集群的英文为 High Availability Cluster，简称 HA Cluster。在这种集群系统中，当一个节点不可用或者不能处理客户的请求时，请求会被转到另外的可用节点来处理。这个过程对于客户来说是透明的，完全由集群系统自动完成。因此，客户会认为系统一直在正常运行。这样系统的可用性就提高了。这种集群技术对需要 24 小时不间断运行的系统来说非常适用。

负载均衡集群系统会把负载尽可能平均分摊给集群节点。它非常适合于那些拥有大量用户的服务，如 Web 服务。集群的每个节点运行同样的服务程序，每个节点都可以处理一部分负载，并且负载可以在节点之间动态分配。在这种集群系统中，需要有一个负载均衡算法，而且它的优劣直接影响到系统的效率。Web 服务是使用负载均衡集群技术的典型代表。因为 Web 服务的访问量太大了，所以必须选用高性能的系统才能满足要求。之所以选择集群系统而不用小型机或者大型机，除了小型机和大型机价格太贵以外，还因为集群系统其实比小型机和大型机更适合处理 Web 业务。这是因为 Web 用户请求的处理实际上都是一些简单的任务。大用户量带来的工作强度主要是在用户请求的并发处理上。而小型机或大型机的能力通常是体现在复杂单一任务的处理上。高强度的并发处理其实不是它们的优势。而负载均衡集群技术却正好擅长于并发处理，而每个节点的处理能力却不一定很强。

Web 服务器集群的概念最早由伊利诺斯州大学的国家超级计算应用中心提出。他们还实现了一个原型系统。后来 Berkeley 的 NOW 小组、NSC 和科罗拉多大学的 Harvest 小

组、Cisco 及 IBM 公司等也都加入 Web 集群的研究行列，并很快推出产品。这些系统的成功证明了 Web 服务器集群是改善 Web 服务的一种有效解决方案。Yahoo、Net Center 和 MSN 都是用几百到几千台计算机组成 Web 服务器集群来对外提供服务的。采用集群技术，要扩大系统规模非常容易，只要在集群中增加新的 Web 服务器节点即可。

图 9-5 是北京大学计算机网络与分布式系统实验室的天网搜索引擎所用的集群系统。系统中一共有 53 台机器，主要是 DELL 的机架式服务器。

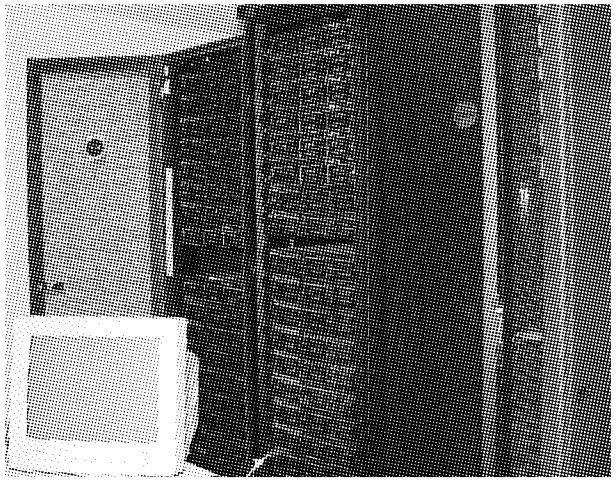


图 9-5　天网搜索引擎的集群系统

要实现集群系统，首先需要有多台计算机。然后需要把这些计算机连接起来。最重要的是要有相应的软件来协调集群系统中的节点。集群的节点可以选用一般的 PC，也可以选择性能较高的计算机。计算机之间可以采用普通的网络连接，使用普通的网卡。不过要提高集群系统的效率，最好采用速度较快的、专门的网络互联设备。有许多专为集群系统设计的网络接口卡、交换机，甚至还有带专门通信协议的网络。有些并行集群系统就绕过像 TCP/IP 这样的网络协议进行通信。这样可以得到高带宽和低延迟。因为网络协议包含非常大的开销。这些开销对于广域网很重要，但是在节点相互已知的封闭网络集群系统中就是不必要的、多余的了。现在有很多管理集群系统的软件，如 RedHat 的 Piranha、Turbo Linux 公司的 Turbo Cluster 等。它们通常都会支持几种芯片，并且支持常见的操作系统平台。这些软件支持的集群系统的节点数甚至可以达到无限台。有些还能够支持异构的网络环境[74]。如果读者对集群系统感兴趣，可以自己去查阅相关资料，进一步了解集群系统的概念，学习怎样架设集群系统。

小提示

集群技术将多台计算机组织起来协同工作模拟一台功能强大的计算机。
集群技术分为三大类：高可用性集群技术、高性能计算集群技术和高可扩展性集群技术。
集群系统具有良好的性价比，因此，很多服务都采用集群系统实现。

5. 服务器的选型

除了统称为服务器外，其实我们日常更习惯根据服务器所提供的服务来称呼它们，如文件服务器、Web 服务器、电子邮件服务器、数据库服务器等。这些服务器提供的服务不同，服务器软件实现的机制也不同，而且用户群也不同，因而服务器硬件的负载也就不同。这使服务器的硬件配置也就不相同。比如说，因为 Web 服务器要用缓存来提高响应速度，因而服务器的内存配置会比较高，而邮件服务器因为要存储大量的信件，所以它的硬盘无论在容量和速度上都必须达到一定的水平。因此，人们在设置服务器时，都要从服务器的用途出发来确定具体的配置。

对于 Web 服务，在设置服务器时，人们要考虑可能的用户访问频度、支持服务的复杂程度以及服务器上运行的服务器软件。再具体点儿来说，要考虑的因素包括可能的用户访问数、每个访问要求的数据量、服务器使用的软件、CPU 处理能力、内存访问速度、硬盘访问速度等。对于用户访问量，既要考虑每天访问服务器的平均用户数，也要考虑单位时间内访问服务器的最大用户数。此外，不同软件厂商的 Web 服务器软件以及同一 Web 服务器软件的不同版本对资源的耗费和本身的性能也不一样。所以这些必须都列入考虑范围。在所有机器硬件指标中，Web 服务器对内存的要求最高。因为 Web 服务器要用内存作缓存来提高服务的响应速度，内存的大小对服务效率的影响很大。

如果要配置邮件服务器，那么需要考虑的因素有总用户数、常用用户数、总存储量、单位时间平均的信件收发量等。电子邮件服务对实时性要求不高，但是存取邮件的数量比较大，因而对硬盘空间的要求比较大。磁盘的存储速度也直接影响系统的性能。在收发邮件时，系统需要对用户身份先进行认证。这个过程需要比较快才行，因而需要服务器 CPU 的处理速度也要比较快。所以，邮件服务器通常会需要非常大的硬盘空间，硬盘的访问速度也应该比较快，并且拥有一个较强的 CPU。

此外，很多服务都会涉及数据库的访问，如邮件系统用户身份的认证、Web 服务 cookie 的使用等。通常为了提高系统的性能，数据库都放在单独的计算机上。此外，因为数据库系统会被频繁地访问，所以数据库服务器的性能和内存配置都要比较高。而且一般要采用双机热备份的方式保证系统的可用性[75,76]。

总之，设置某种服务的服务器时，必须针对服务本身，考虑多方面因素来确定服务器硬件的具体配置。

小提示

在所有服务器硬件指标中，Web 服务器对内存的要求最高。

邮件服务器通常会带有非常大的硬盘空间，而且硬盘的访问速度也应该比较快，并且拥有一个较强的 CPU。

数据库服务器的 CPU 性能和内存配置都比较高，一般会采用双机热备份。

9.1.2 网络设备

除服务器外，计算机系统中另外一类常见基础设备就是网络设备了。这些设备为计算机之间的通信日夜奋战。但是，它们在用户眼中近乎是透明的，用户几乎感觉不到它们的存

在。这一节我们就来简单了解这些设备。

1. 路由器

路由器是网络中最重要的设备。图 9-6 是一些常见的路由器。路由器最主要的功能就是为网络中的数据包选择传递的路线，让这些数据包能够又快又好地到达目的地。这个工作可以分为两部分。首先是选路，然后是转发。

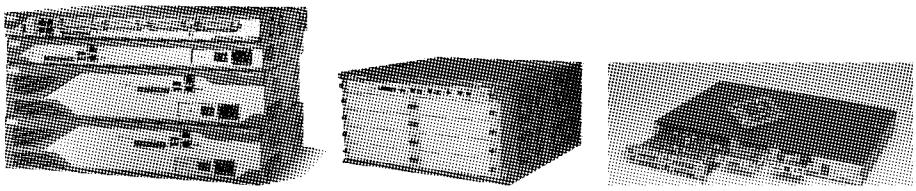


图 9-6　常见路由器

选路即常听到的路由，这也是路由器名字的由来。选路的过程需要网络中所有的路由器共同参与才能完成。这些路由器通过路由协议相互联络，获得网络的拓扑结构情况，然后计算出网络中两个设备之间的最优路线。路线形成的时机通常有两个。一是在用户将数据包交给网络要求送达目的地时。路由器们在接到投递任务后，开始协商并找到一条路线。这被称为按需路由。因为是在数据包到来时，才寻找路线，所以按需路由比较慢。但是，找到的路径总是最能反映当前网络的状况。按需路由通常用在无线网络中，因为这种网络变化比较快。另一个路由形成的时机是在数据包投递任务发生之前。路由器启动之后，就与其他路由器沟通讨论路由的事情，然后每个路由器都形成从自己到网络中全部节点的最佳路线。这样，等到有数据包需要传递时，就可以直接按照事先找到的路线投递了。这被称为预先路由。路由器会根据路由协议的规定按照某种方式在需要时更新这些路线，以便能够及时反映网络的情况。一般有线网络中的路由器多采用这种方式。

无论按照哪种方式确定路径，当数据包到达路由器时，路由器都要按照确定的路线将其传递给下一个路由器。这个过程被称为转发。这是路由器第二个重要的工作。为了减少数据包在网络中的停留时间，转发过程需要尽可能地快。

此外，路由器还可以把不同网络，如以太网和令牌环网，之间的不同格式的数据包互相翻译。这样就把网络互联的范围扩大了，实现了不同网络的互联。路由器还有很多其他重要的功能，如网络计费、网络管理等。

路由器上有很多端口。有些可以连接广域网，有些可以连接局域网。路由器需要管理员配置必需的管理信息，如使用的路由协议、每个端口的地址等。路由器有个控制台(console)端口专门提供给管理员让其配置路由器时使用。图 9-7 是路由器配置的一个界面。这里“?”是个命令，它的作用是列出当前配置模式下可以使用的全部命令。在学习计算机网络原理或者网络管理与配置相关的课程中，你会了解这些命令的作用和用法。

按照路由器的能力，路由器可以分为接入路由器、企业级路由器以及骨干级路由器。那些能够把家庭或小型企业连接到网络服务提供商的路由器被称为接入路由器。企业级路由器则负责连接一个校园或企业内的计算机。骨干级路由器把各个企业网络连接起来构成互联网。通常骨干路由器并不直接连接终端计算机。

现在生产路由器的国外厂商主要有思科公司、北电网络等，在国内则主要是华为公司。

```
Router1>?
Exec commands:
  access-enable     Create a temporary Access-List entry
  atmsig            Execute Atm Signalling Commands
  cd                Change current device
  clear             Reset functions
  connect           Open a terminal connection
  dir               List files on given device
  disable           Turn off privileged commands
  disconnect        Disconnect an existing network connection
  enable            Turn on privileged commands
  exit              Exit from the EXEC
  help              Description of the interactive help system
  lat               Open a lat connection
  lock              Lock the terminal
  login             Login as a particular user
  logout            Exit from the EXEC
-- More --
```

图 9-7　路由器配置界面

小提示

路由器最重要的两个功能就是路由和转发。

按照路由器的能力，路由器可以分为接入路由器、企业级路由器、骨干级路由器。

2. 交换机

交换机是网络中心里最常见的设备之一。在一些计算中心，甚至普通用户的办公室也经常能够看到交换机。图 9-8 是一些交换机的外观。

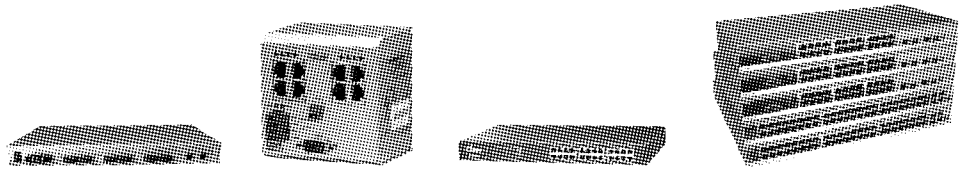


图 9-8　常见交换机

交换机最主要的功能就是交换。“交换”是来自电话网络中的一个术语，在早期就是“转接”的意思，即由接线员负责把一个用户的电话转接到其呼叫的用户电话上。而现在则多指“转发”。交换机负责把从一个端口接收来的数据转发到合适的输出端口上，从而让数据能够通过交换到达或者接近目的地。

与路由器不同，交换机没有那么宽的视野和能力，它不能知道数据包从发出点到目的地的整个路线。交换机只能把从自己一个端口进来的数据包从另外一个合适的端口转发出去。什么是合适的端口，交换机有自己的一套确定办法。

比如说，以太网中的每个数据包除了带有发送该数据包的主机以及数据包目的地主机的 IP 地址外，还都带有两个 MAC 地址。这两个 MAC 地址不是源主机和目的主机的 MAC 地址。它们分别是网络中正在发送该数据包的网络端口（或主机端口）的 MAC 地址，以及下一站要接收这个数据包的网络端口的 MAC 地址。也就是说，当数据包在网络中流动时，这两个 MAC 地址是随路线一站一站改变的。以太网交换机通过数据包的 MAC 地址来确定数据包的去向。交换机通过记录经手数据包的源 MAC 地址来学习网络中各个设备的位置。例如，假设它的端口 1 收到源 MAC 地址为 A1 的数据包，那么它就推断 MAC 地址为 A1 的设备可以通过端口 1 到达。因此，如果接收到需要发送给 A1 的数据包，它就将其转发到端口 1，而不再向其他端口转发。就这样，交换机通过记录数据包的源 MAC 地址及其来自的端口号形成自己的转发表。如果交换机收到数据包，但是在转发表中没有找到该数据包目的 MAC 地址的记录，那么交换机就会向数据包到来端口之外的所有端口转发该数据包。

除了交换外，现在的交换机还有很多其他功能，如支持虚拟局域网 VLAN。有的还具有防火墙的功能。一般来说，交换机的每个端口都用来连接一个独立的网段。通常属于该网段的多台计算机连接到集线器，集线器再连接到交换机的某个端口。当然我们也可以把一些重要的计算机直接连接到交换机的端口上。在共享的以太网中使用交换机可以将网络“分段”。这可以减小数据包冲突的可能。

按照传输介质和传输速度，交换机可分为以太网交换机、快速以太网交换机、千兆以太网交换机、FDDI 交换机、ATM 交换机和令牌环交换机等。在应用规模上，交换机又可划分为企业级交换机、校园网交换机、部门级交换机、工作组交换机以及桌机型交换机 5 种。按照交换机的端口结构，交换机可分为固定端口和模块化交换机两种。

通常我们会听到交换机有存储转发（Store & Forward）、直通（Cut Through）、碎片避免（Fragment Free）三种不同的交换方式。这都是什么意思呢？下面来简单解释一下。

存储转发，就是先存储，再转发。交换机把输入端口接收到的数据包先存储起来；接着检查数据包是否有错误；如果没错就取出数据包的目的 MAC 地址，查表然后把数据包送往输出端口。存储转发方式可以对进入交换机的数据包进行错误检测，因而可以改善网络性能。还可以支持不同速度端口之间的转发，让高速端口与低速端口能够协同工作。它的缺点就是数据包在交换机中停留时间太长。

为提高转发速度，交换机在接收到数据包的目的地址，而其他内容还没有全部接收完时，就开始查表，把数据包直通到相应的输出端口。这种方式就是直通式交换。由于不需要存储，直通式交换延迟非常小。不过，因为数据包内容并没有被交换机保存下来，所以也无法检查是否有误。由于没有缓存，也不能将不同速率的输入/输出端口直接接通。

我们常用的局域网是以太网。在以太网中，计算机共用同一通信线路。因此，不同计算机发送的数据包就可能发生冲突。按照以太网的协议，计算机发送数据包后会检测是否发生冲突，如果有，就停下来过一段时间再重新发送。可是在冲突过程中，已经在网络中留下一些破碎的数据包。由于以太网协议的关系，这些碎片包的长度通常会小于 64B，而以太网正常的数据包长度都大于 64B。但是 64B 已经足够包含数据包的目的 MAC 地址，所以直通式交换会把这些无用的碎片数据包又转发到网络中。这会浪费网络带宽，增加交换设备的开销。为此，人们把直通式交换进行了一些改良。让它检查数据包的长度是否够 64B。如果大于 64B，则转发该包，否则就丢弃该包。这种交换方式被称为碎片避免式交换。由于

碎片避免式交换也不进行数据校验，所以，它的数据处理速度比存储转发方式快，但是，因为要检查数据包的长度，因此比直通式交换要慢。

通常交换机只需要很少的配置，比起路由器来，交换机的配置要简单多了。不过现在很多交换机的功能被加强，有些路由器的功能它们都有了，当然，配置也就变得复杂一些。

小提示

交换机最重要的工作就是交换。
交换机有存储转发、直通、碎片避免三种交换方式。

3. 网桥

网桥可以把两个局域网连接起来。网桥通常用于连接数量不多的、同一类型的网络。这就是它的名字表现出来的意思。网桥也根据数据包的 MAC 地址来确定数据包要转发的方向。网桥的转发表也是自学得来的。其实，网桥的功能看起来和交换机非常相像。只不过交换机的端口通常比较多。而网桥常常只有两个端口。这从它名字上的“桥”字也可以看出来。而且，一般交换机采用硬件进行转发，所以速度要比网桥快。图 9-9 是一些网桥的外观。图中可以看到带天线的无线网桥。

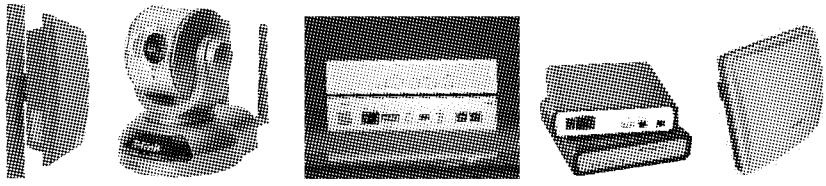


图 9-9 常见网桥

与交换机一样，网桥可以将一个较大的局域网分成段。这样可以减少共享域中主机的数目，因而减小数据包冲突的可能性。

网桥通常有透明网桥和源路由选择网桥两大类。

小提示

网桥通常用于连接数量不多的、同一类型的网络。
网桥可以将较大的局域网分成段，减少数据包冲突的可能性。

4. 集线器

集线器(hub)负责放大网络中的信号，延长网络的距离。集线器一般有很多端口。每个端口可以连接一个设备。但集线器从一个端口上收到数据时，会把它复制到所有其他端口转发出去。也就是说，集线器是共享设备，不过现在也有交换式集线器。

通常人们喜欢按照端口数量来区分集线器，如 8 口、16 口或者 24 口集线器。如果按照带宽来划分，常见的集线器有 10Mb/s、100Mb/s、10/100Mb/s 集线器等。图 9-10 是一些常见的集线器。

5. 中继器

因为存在损耗，所以线路上传输的信号功率会逐渐衰减，衰减到一定程度时就会造成信

图 9-10　常见集线器

号失真。要想延长信号的传输距离，就必须在网络中对信号进行重生、放大。中继器就负责这件事情。中继器是最简单的网络互联设备。它负责在两个节点间完成信号的复制、调整和放大功能，以此来延长网络的长度。通常，中继器的两端连接的是相同的媒体。图 9-11 是一些光中继器。

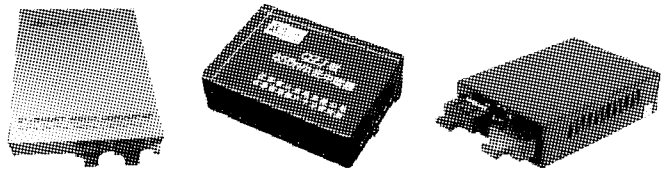


图 9-11　光中继器

集线器与中继器的功能和特性非常相似，因而集线器也被称为多端口中继器。通常中继器只有两个端口，而集线器有很多端口。因为有多个端口，所以集线器比中继器多出一个功能，它可以作为网络汇集点连接多个设备。但是与交换机不同，通过集线器连接的设备都属于同一个局域网段。

小提示

集线器和中继器的主要作用是对信号进行放大和重生。
集线器还可以用来互联同一局域网段的多个设备。

9.2　计算机机房

最后我们来看看放置服务器等计算机设备的机房。它可以算是基础设施的基础设施了。可别小看了这个放计算机的房子，里面的学问也不少。计算机价格很贵，并且它们做的事情又很重要，所以它们的房间必须绝对安全可靠。因此，机房必须能够防盗、防火。计算机属于精密仪器，不合适的温度、湿度、空气质量、供电系统都会对计算机造成损害。所以，机房里的温度、湿度、洁净度和室内空气流通度都必须严格控制。此外，雷电、振动、电磁干扰、水甚至老鼠和虫子都会对计算机的正常工作产生不良影响[77,78]。这一系列的保卫防护工作就都交给了机房。由此可以看出，机房的工作也是举足轻重的。为此，我国还专门制定了机房建设的一系列标准，如《电子计算机机房设计规范》(GB 50174—93)。

小提示

机房建设要考虑温度、湿度、洁净度和室内空气流通度，要注意防水、防火、防电磁干扰、防振、防雷、防鼠、防虫以及防盗。

9.2.1 机房的位置

要想胜利完成安全存放计算机的任务，首先机房的位置就很有讲究。机房一般会建在电力、水源充足，自然环境清洁，通信与交通运输方便的地方。通常会尽量避免设在那些会产生粉尘、油烟、有害气体，以及生产或储存具有腐蚀性、易燃、易爆物品的工厂、仓库、堆场的旁边。低洼、潮湿、落雷区和地震活动频繁的地方当然也很少会作为设置机房的选择地。强振动、强噪声以及强电磁干扰会影响计算机的运行，所以机房也会尽量远离强振动源、强噪声源以及强电磁场。如果机房必须要设置在那些不适宜的地方，那么就要采取一些相应的技术措施克服困难，如设置电磁屏蔽避免强电磁场的干扰。

如果机房设在多层楼内，通常是放在第二或第三层，而不是顶层、地下室。这样做是为了防水、防潮，并把太阳辐射热的影响减到最小。同样原因，机房也很少会设在用水设备的下层。机房中放置的一些设备的体积会比较大，所以机房都会有一个比较宽敞的设备进出口场地，而且会为大型设备留有足够大的出入口。机房通常会留有设备扩充空间，因此一般机房设计时的使用面积都是设备占地面积和的 5～7 倍[80-83,88]。

小提示

机房选址：

应尽量在电力、水源充足，自然环境清洁，通信与交通运输方便的地方；

应该远离产生粉尘、油烟、有害气体以及生产或储存具有腐蚀性、易燃、易爆物品的工厂、仓库、堆场等；

尽量避免建在低洼、潮湿、落雷区和地震活动频繁的地方；

应尽量远离强振动源和强噪声源；

尽量远离强电磁场的干扰。

不能满足上述条件，则应采取相应的技术措施。

9.2.2 机房的环境条件

1. 温度和湿度

计算机属于精密仪器。过高、过低或陡然变化的温度对计算机设备运行的稳定性、可靠性以及寿命的影响都非常大。过高的温度会使集成电路内离子的扩散与漂移加快。电子的运动速度加快，穿透电流即增大，容易造成集成电路的损坏。计算机内电阻元件的阻值在温度过高时也会发生变化，也会使计算机出现故障。统计表明，当器件周围的温度超过 60℃时，就会引起计算机设备发生故障。温度每升高 10℃，计算机设备的可靠性就会下降 25%。而温度过低则会使设备的绝缘性能下降，并且容易产生结露，导致接触性故障。过高或过低的温度都会影响计算机晶体振荡器电路时钟的主振频率。这直接影响到计算机的工作效率。温度的剧烈变化会使磁盘等精密机械由于热胀冷缩影响而出现读写错误。而且热胀冷缩也容易使插头、插座和开关等部件接触不良。因此，机房的温度必须控制在合适的范围内[77,89]。

国家专门对机房的温度制定了标准。例如，A 级标准要求开机时机房夏季的温度应该控制在 22℃左右，温度变化小于 5℃/小时，并且保证不要凝露；停机时温度要控制在 5～35℃之间，同样温度变化要小于 5℃/小时，并且保证不要凝露（其他具体的规定可以查看标准《计算站场地技术条件》(GB 2887—89 代替 GB 2887—82)）。

为了控制温度，机房都会安装专用的空调机，有些还采用温度自动报警装置来监测温度

及其变化。

小提示

温度对计算机系统的性能影响非常大。
机房要使用专用空调机来控制温度。

除了温度外，湿度对计算机的运行也有很大影响。湿度过高时，电子元件表面就会产生水膜，形成“导电通路”，影响集成电路的电气性能，导致错误的逻辑判断。湿度过高也会使接插件和集成电路的引脚容易氧化和生锈霉烂，造成接触不良或断路。湿度过低也不行。太干燥时，机房中转动的机器、活动地板等这些有磨擦的部件就容易产生静电。静电电荷积聚到一定程度就会引起磁盘、磁带等设备读写错误，甚至烧坏半导体器件。磁盘带静电还容易吸附灰尘，造成磁头损坏，划坏盘片。因此，机房的湿度也必须进行严格控制。国家标准规定开机时湿度要控制在 45％～65％，停机时也要在 40％～70％。这是对湿度较高的要求。对于一般机房，国家标准也给出了两个略低的要求（详细内容见《计算站场地技术条件（GB 2887—89 代替 GB 2887—82）》）。

要想把机房湿度控制在合适的范围内，通常要使用恒温恒湿装置。一般机房内会安装除湿机和加湿机，或者使用具有加湿和去湿功能的空调系统。

小提示

为控制湿度，机房内要安装除湿机和加湿机，或者使用具有加湿和去湿功能的空调系统。

2. 灰尘和有害气体

干灰尘进入设备会使接插件触点间接触不良，还会使机械设备的摩擦阻力增加，加快设备的磨损。灰尘覆盖在电子元件上，会影响电子元件散热能力。长期积聚大量热量就会使设备工作不稳定。如果灰尘中含有水分和腐蚀物质，那么就会影响电路的正常工作，严重时会烧坏电源、主板或其他设备部件。现在的设备在运行过程中一般都会产生很多热量。通常设备都采用风冷方式散热。散热孔与对流的空气配合，很容易将灰尘带入设备内部。因此，机房中都要尽量减少灰尘。

通常灰尘的来源有以下几种[79]：

① 由进出机房的人员带入。

② 由空调系统在送风时带入。

③ 通过机房的门窗（特别是未经防尘处理的普通房间）流入。

④ 由机房老化表面（如墙壁、地面、顶棚等）产生的表皮脱落形成。

⑤ 机房设备，如打印机等，在运转过程中产生，如纸屑或墨粉颗粒。

⑥ 由气压差导致灰尘从外界流入。因为大多数机房在运行时对机房外部都是负压，即外界气压高于机房内气压，这使灰尘容易流入机房。

因此，为了减少机房中的灰尘，机房对出入人员都有严格的控制，无关的人员会被禁止进入机房。大的机房会被分成区域，如分为服务器主机区、控制区、数据处理终端区等，以减少设备与无关人员的接触机会。经常有人参观的机房应设置专门的参观通道，通道与主机

区用玻璃幕墙隔开。工作人员要配备专用的工作服和拖鞋，并且经常清洗。进入机房的人员，要更换专用拖鞋或使用鞋套。

为了防止灰尘从空调系统进入，机房的空调系统中会安装空气过滤器，收集进入机房和回风中的尘埃。空调的风网也会被按时清理。机房四壁及地面通常会选用不起尘、不易积灰、易于清洁的饰面材料。机房的室内装饰也会选用气密性好、不起尘、易清洁的材料。而且机房室内的门、窗一般采用双层密封式。一些机房门前还会设置一个前置门，减少灰尘进入机房的可能。那些容易产生灰尘的设备，如打印机，通常会与其他设备分区放置。一些机房还会使用设备加大机房内部的气压，使机房内的空气通过密闭不严的窗户、门等的缝隙向外泄气，达到防尘的效果。

机房要定期进行除尘。除尘周期一般根据机房的具体情况而定，如每三天清理一次机房内部卫生；每天清洁机房外部卫生；每周对设备吸附尘土情况进行检查；每月清洁一次需要清洁的设备等。

如果机房所在地的空气洁净度不够好，含有腐蚀性、导电性气体，如汽车或工厂排出的二氧化硫(SO_2)、硫化氢(H_2S)、氮氧化物(NO_x)、一氧化碳(CO)、氯气(Cl_2)、臭氧(O_3)等气体，也会对设备造成危害。这些有害气体与空气中的水作用后生成酸、碱、盐等腐蚀性物质对集成电路等精密设备有很大的腐蚀作用。这些气体中，SO_2 和 H_2S 对计算机系统的影响最大。因为这两种气体的腐蚀性相对较强，而且在大气中的含量也较高。

为了保证机房空气的洁净度，控制有害气体含量，一些机房在空调入风口、通风管道中会设置化学过滤器，并且保证机房的密封性，让室内有足够的正压，防止有害气体侵入。有害气体对设备的腐蚀作用大小与机房温度、相对湿度以及灰尘浓度相关。所以，对有害气体的防护通常与控温、控湿、控尘措施综合考虑(关于空气洁净度和机房气压的具体要求可以参看国家标准《计算站场地技术要求》和《电子计算机机房设计规范 GB 50174—93》)。

小提示

防尘对机房设备正常运转很重要。

机房要避免灰尘进入，并定期除尘。

可以在空调入风口、通风管道中设化学过滤器，同时保证机房的密封性，保持机房空气的洁净度，防止有害气体腐蚀计算机。

3. 机房中的空调机

前面几节介绍机房的环境控制时都提到了空调机。因此，这里对机房内的空调机专门进行一些讨论。

在机房里设置空调系统时都要考虑系统的负荷。机房的热负荷包括计算机及其他设备的散热、建筑围护结构的传热、太阳辐射热、人体散热、散湿、照明装置散热以及新风负荷等。机房的热负荷可以采用简单的估算方法计算。通常可以按照 $0.3kW/m^2$ 来计算。这其中包括计算机设备、人工照明、工作人员、围护结构等产生的热量。当然，也可以按照每一项产生热量的热源来计算整个机房的热负荷。但是，计算起来有点儿复杂，每一项计算的方式都不一样。计算机设备、外围设备及机房设备的发热量大约占总热量的 80%以上。余下的热量来自于照明热、传导热、辐射热等。一般计算机的制造商会提供设备发热量的具体数值。如果没有提供，那就只能根据计算机的耗电量计算其发热量。计算机和各种设备以及

照明、围护结构等都有相应具体的计算公式。这里就不做详细介绍了。一般空调的负荷能力会保留有 15%～20%的余量。中高档的机房通常采用模块化的机房专用空调。这样易于机房扩容和改造[97]。

机房空调在使用时，需要综合考虑设备对空调的要求、设备本身的冷却方式、设备布置密度、设备发热量以及房间温湿度、室内风速、防尘、消声等各项指标，还有建筑条件等因素，来设计空调的气流组织形式。表 9-1 是气流组织的一些参考建议，常用在计算机系统对气流组织没有明确要求的情况下[81]。

表 9-1　气流组织、风口及送风温差

气流组织	下 送 上 回	上送上回(或侧回)	侧送侧回
送风口	1. 带可调多叶阀的格栅风口 2. 条形风口(带有条形风口的活动地板) 3. 孔板	1. 散流器 2. 带扩散板风口 3. 孔板 4. 百叶风口 5. 格栅风口	1. 百叶风口 2. 格栅风口
回风口	1. 格栅风口，2. 百叶风口，3. 网板风口，4. 其他风口		
送风温差	4～6℃ 送风温度应高于室内空气露点温度	4～6℃	6～8℃

设备布置的密度较大且设备发热量也较大的机房，如网络中心的主机房，通常采用活动地板下送上回方式。即把活动地板下的空间，用作空调送风的通道，空气通过在活动地板上装设的送风口进入机房，而回风口则设在机房顶棚上，让回风从这里回至空调装置。采用活动地板下送风时，考虑到工作人员和设备运转需要，出口风速不大于 3m/s，而且送风气流不直对工作人员。

在一些非专业机房安排的场合，例如大学的研究小组，通常也可能将许多服务器集中放在一个房间，门窗关严实，安装一些空调来进行散热。一个常见的误区是：将那些服务器摆放得很分散，好像能够更容易散热的样子。如果是自然通风，这样是有道理的；但在依赖空调散热的情况下，如此安排热交换效率往往会很低。较好的安排是将服务器适当集中起来，与空调的风口之间形成流畅的冷热交换气流，从而让空调能够迅速将热空气带出去。

机房里一般使用专用空调，而不是普通的民用空调。因为这两个空调设计的目标不同，工作重点也不同。机房专用空调的作用是为机房设备提供恒温恒湿的运行环境，因而要力保恒温恒湿。民用空调则是直接服务于人的。为了让人感觉舒适，民用空调要保证低噪音、低风量，通常它还要进行除湿。因为人通过呼吸和经由皮肤的汗液挥发都会产生水汽。而机房却没有湿气来源，一方面机房密封性好，而且机房设备也不会像人一样发出水蒸气。但是，机房的设备会散发出很高的热量，所以机房专用空调需要大风量循环，风量会很大。而且为了避免凝露，机房专用空调的出风温度通常要比民用空调高。此外，民用空调一般没有加湿功能只能除湿，但是机房专用空调通常都会有加湿功能。否则机房还要另外设置湿度控制设备。还有，因为机房的要求较高，所以机房专用空调的温湿度控制都可以达到很高的精度，民用空调则不需要按照这么高的标准去实现[98]。

总之，机房空调系统和设备的选择是根据计算机类型、机房面积、发热量，以及对温、湿度和空气含尘浓度的要求综合考虑的结果。此外，一般机房都会选用风冷冷凝器的空调机。

因为如果采用水冷机组，在冬季还要对冷却水系统采取防冻措施。为减少机房里的噪声和振动，机房空调和制冷设备都会尽量选择使用低噪声、低振动、高效率的设备。对于需要长期运行的系统机房，空调系统一般会设有备用装置。

小提示

机房要用专用空调来保持恒温恒湿以及空气洁净度。

空调的负荷计算要考虑计算机和其他设备的散热、建筑围护结构的传热、太阳辐射热、人体散热、散湿、照明装置散热以及新风负荷等。

9.2.3 机房的电气

在我国的标准中把电力系统的负荷按照重要性分为了三个等级。其中，一级负荷指特别重要的用电负荷，中断供电会造成重大损失，如造成人身伤亡、爆炸、火灾或者重大设备损坏、国民经济中重点企业的连续生产过程被打乱需要长时间才能恢复等。二级负荷就是指那些相对重要的用电负荷。如果中断供电将造成较大损失，如主要设备损坏、大量产品报废、造成大型影剧院、大型商场等较多人员集中的重要公共场所秩序混乱等。最后，那些不属于一级和二级负荷的供电被列为三级负荷。按照这个标准，国防、交通运输、金融财政、航空等部门的机房都属于一级电力负荷。一般科研、计算机控制系统的生产单位的机房供电属于二级负荷。计算机应用领域，如数据统计、查询、情报检索等工作的机房供电属于三级负荷。

按照要求一级负荷必须有两个电源供电。一个电源发生故障时，另一个电源不应该同时受到损坏。也就是说一级负荷要保证不间断供电。在一级负荷中特别重要的负荷，除了由两个电源供电外，还要增设应急电源，并且其他负荷不能接入应急供电系统。应急电源可以是独立于正常电源的发电机组、供电网络中独立于正常电源的专用馈电线路、蓄电池、干电池等。二级负荷要采用带备用的供电系统。而三级负荷采用一般用户的供电系统即可[84-87]。

小提示

按照标准电力负荷分为三级。

一级负荷采用不间断电源供电，二级负荷采用带备用的供电系统，三级负荷采用一般市电。

通常机房都会采用 UPS 供电来满足不间断供电的要求。采用 UPS(Uninterruptible Power System，不间断电源系统)供电，除保证供电的可靠性外，还可以将供电各项参数较精确地稳定在正常值之内。这是因为机房中的设备对供电质量的要求很高，供电指标不正常的波动对设备的工作和寿命都会有很大的影响。

UPS 是一种含有储能装置，以逆变器为主要组成部分，恒压恒频的不间断电源。按照工作方式 UPS 可分为后备式、在线互动式以及在线式三种类型。

从名字上可以看出后备式 UPS 只是作为后备。在市电输入正常时，后备式 UPS 直接将市电交给设备使用，它不做任何处理，只是同时通过充电电路给蓄电池充电。当市电电网

供电发生异常时，如断电或者电压变化超过额定范围，后备式 UPS 就会切换到电池逆变状态，由蓄电池对负载进行供电。后备式 UPS 对市电没有净化功能，因此它的稳压特性不好。在由市电转为电池逆变工作时会有一段约为几毫秒的转换时间。因此，后备式 UPS 不适用于那些不允许有切换时间的场合，仅适合比较简单、不很重要的环境，如办公或家用 PC、不重要的网上终端等。这种 UPS 的优点是价格便宜、结构简单。通常后备式 UPS 容量都比较小。

在线互动式 UPS 是后备式的一个变种。在市电正常时，还是由市电直接供电；当市电电压偏低或偏高时，在线互动式 UPS 内部稳压线路会将电流稳压后输出；当市电异常或停电时，转为电池逆变供电。这种 UPS 有较宽的输入电压范围、噪音低、体积小，但同样存在切换时间。相对来说，它的成本也比较低，能够适应大多数对断电要求不是很严格的场合，但不适合大型数据网络中心和关键用电领域。它的容量通常在 5kVA 以下。

而在线式 UPS，不论电网供电是否正常，都由 UPS 内的电池对负载进行供电。当市电供电正常时，它首先将市电交流电变成直流电，然后逆变器再将直流电重新变成需要的交流电向负载供电。这样，原来市电电网上电压幅度不稳、频率漂移、波形畸变及噪音干扰等电源问题都得到了解决。当市电中断时，蓄电池提供后备能源继续供电。由于不存在市电供电到逆变器供电的转换操作，因此也就不存在转换时间问题了。因为在线式 UPS 可以支持不间断供电、恒稳电压、备份电，所以它适合对供电质量要求较高的场合，如大型数据网络中心、重要的仪器设备、控制系统等。因此，在线式 UPS 是较为主流的一种 UPS。不过，价格也就比较高了[95]。

机房里的 UPS 要保证其容量足够支撑要其供电的设备。UPS 容量的单位为伏安(VA)。UPS 要支持设备的总耗电量决定了所需 UPS 的容量。通常人们得到设备的参数是以瓦特(W)为单位标示的额定功率。这是设备电源的额定输出功率。它与 UPS 直接供给设备的电能伏安是有区别的。对于某些设备来说，有一部分负载电流没有做功。这是由负载特性引起的。因此，从 UPS 输出的到设备的电流总功率被称为虚功。而设备的功率是其获得电流做的有用功。设备有用功与其获得的电流总功率之间存在一个比值，被称为功率因子。即：

$$\text{功率(W)} = \text{电压(V)} \times \text{电流(A)} \times \text{功率因子}$$

可以看出，设备的功率因子越高，设备本身的效能越好。反之功率因子越低，表示设备本身所消耗的能源越大，也就越费电。通常设备的功率因子都小于 1，只有电加热器和灯泡等的功率因子为 1。目前多数计算机类设备的功率因子大概在 0.6～0.7 之间。因此，UPS 要支持的所有设备的功率总和应该在 UPS 容量伏安(VA)数的 60%～70%。

供电时间也是 UPS 另一个重要的技术指标。它一般是指在满负载的情况下，电池可以供电的时间。毫无疑问，这个时间自然是越长越好。但是更长的时间意味着更高的成本，也就意味着更高的价格。市电电压输入范围、输出电压频率范围、波形畸变率、电压稳定度是 UPS 与供电质量相关的一些技术指标。电压输入范围宽表明 UPS 对市电的利用能力强。输出电压频率范围越小表明 UPS 对市电调整能力越强、输出越稳定。波形畸变率用来衡量输出电压波形的稳定性。而电压稳定度则说明当 UPS 突然由零负载加到满负载时，输出电压的稳定性[96]。

小提示

UPS是一种含有储能装置，以逆变器为主要组成部分，恒压恒频的不间断电源。

UPS主要参数有容量、供电时间、市电电压输入范围、输出电压频率范围、波形畸变率、电压稳定度等。

UPS可分为后备式、在线互动式、在线式几种类型。

空调设备、通风设备以及其他动力设备、照明设备、测试设备等机房辅助设备，一般单机功率较大，启动电流也很大，对供电质量要求相对要低，其工作时负荷变化频繁，特别是大功率单机启动时对局部电网电流、电压波动影响很大。所以，这些部分设备通常会与计算机类设备分开，采用另外的电源供电，以避免影响。这些设备负荷不能接入主机用的不间断电源系统。如果这类设备需要不间断电源，那么通常会采用专门为这些被称为“感性负载”设计的UPS。

除了采用UPS系统外，机房的供配电中还有很多特别的细节来保证供电安全和设备的正常运行。例如，一般配电箱柜都有短路过流保护、紧急断电按钮与火灾报警联锁、机房内分别设置维修和测试用电源插座等。还有，服务器的启动电流也比稳定运行时大很多，因此在有大量设备的机房中，设备的启动都考虑分批进行，以避免不必要的跳闸。

小提示

感性负载应该与计算机类设备采用不同的电源供电，更不应该接入同一不间断电源。

9.2.4 机房的干扰及防护

雷电、振动、噪声、静电、电磁场、水、火、鼠虫以及窃贼，都会对机房的正常运行带来影响。所以，机房在建设和管理过程中应考虑这些因素，采取一些相应的防范措施。可以看出，机房确实不是一个普通的房子。

1. 防雷、防振和防噪

雷电的厉害和破坏性是显而易见的。为了防止雷击，机房通常会采用接闪器、均压环、引下线、接地体等组成防直击雷系统。接闪器指直接截受雷击，用作接闪的器具、金属构件以及金属屋面等。它的功能就是把接引来的雷电流通过引下线和接地装置引入大地中泄放，保护建筑物免受雷害。通常接闪器可以分为避雷针、避雷线、避雷带、避雷网几种。

除了直击雷之外，雷电还有一种称为雷电感应的破坏方式。如果电器设备的电源线、信号线、天馈线等遭受雷击电磁脉冲入侵，但事先并未采取相应的等电位、屏蔽或者电涌保护等措施，就会造成电子元器件损坏，进而破坏整个电器设备、网络设备、信息系统，带来重大损失。这就是雷电感应。雷电击中建筑物时，强大的雷电在通过接闪器导入大地的同时，也会以感应雷的形式对建筑物内的电器设备造成危害。连接计算机设备的供电线路、通信线路、接地线路甚至建筑物内部的金属物体等会形成感应雷的通道，以瞬时浪涌形式对设备造成危害。所以，为了解决这个问题，机房的供电电源和通信连接线路上一般都会安装浪涌防护器。一些重要的计算机机房还设计为全屏蔽保护形式。

为了防止雷电沿着建筑物各种金属导线传输到计算机设备。接闪器的引下线及接地系统都是独立的。另外，所有进出建筑物的金属物，包括各种金属管道、各种电缆的金属外皮、

建筑物本身的基础钢筋网以及建筑物内的各种大型金属构件，如配电柜、设备金属外壳、防静电地板等通常都要连接成一个统一的电气整体，与专门的统一地网相连。所有进出建筑物的金属传输线不能直接接地的部分，如电源相线、计算机通信电源的芯线、电话线、电视传输线、各种报警通信电缆的芯线等都会被接上合适的避雷器，并将其接地端连接到统一地网。机房的电源一般也采用有金属屏蔽层的电缆全线直接埋地进线或无金属屏蔽层的电缆穿金属管进线[91,92]。

小提示

机房应该采取防雷措施，如安装接闪器、浪涌防护器、采用独立的、统一的接地网等。

振动对计算机的主要部件以及一些精度高的外设影响非常大。例如，硬盘就是怕振动的。硬盘工作时，磁头会高速运转，振动会使磁头撞击盘片，使盘片或磁头机械损坏。有些盘片还是玻璃材料制成的，这些硬盘即使在未工作时，也是怕剧烈振动的。

机房的振动可能来自外部，如公路上的中型汽车、附近的火车或者飞机，以及像发电厂发电机组这样大型机组设备等。因此，机房一般都尽量建在远离振动源的地方。实在没办法时，如发电厂的机房，则会采取防振和隔振措施。此外，机房中的设备也可能产生振动，如电源、空调。一般在选购这类设备时就会考虑这些因素。如果设备会产生较大的振动，通常要采取建筑结构隔离，防止振动扩大和传播。对于那些防振要求高的设备还会采取单独的防振措施，如安装减振器等。

机房工作人员长期在较大的噪声环境下工作会出现头昏、心烦、耳鸣、失眠、植物神经系统与内分泌系统障碍、情绪低落和工作效率低等现象。因此，我国标准规定，在计算机系统停机条件下，机房内的噪声在主操作员位置测量应小于 68 分贝。所以机房选址时都会考虑远离强噪声源。如果不能避开噪声源，那就要采取消声和隔声措施。

小提示

机房应设置在远离强振源和强噪声源的地方。
对于不可避免的振源和噪声源，要采取相应措施予以隔离。

2. 防静电和电磁干扰

机房中转动的机器、机房人员走动时与地板磨擦都很容易产生静电。可不要小瞧这小小的静电，静电电荷的积聚可以使静电电压高达上千伏。这么高的电压当然对计算机会有很大影响了，它可能击穿计算机中的芯片电路。尽管现在大多数芯片电路在设计上提高了抗静电的能力，但是静电电压太高时仍然难逃厄运。统计表明，很大一部分芯片损坏都是由于静电放电造成的。

除了破坏芯片外，静电的另外一种干扰更让人头疼。当静电带电体触及计算机时，就会对计算机放电。这会给计算机的逻辑元件输入错误的干扰信号，引起程序出错。带阴极射线管的显示设备，受到静电干扰时，就会图像紊乱、模糊不清。静电还可能让调制解调器、网卡、传真机等工作失常，磁盘、磁带读写错误。最麻烦的是，这些静电引起的错误随机性很

强，非常难检测出来。因此，机房中应采取措施来减少静电[90]。

摩擦会产生静电。因此机房的地面多采用特殊的活动地板，而不是地毯。这些地板表面能够导静电。一些机房的工作台面及坐椅垫套材料也都是导静电的。为减少静电，机房工作人员的工作服也都是不易产生静电的。他们在安装接插件或更换电子元器元件时，都要使用手腕带及脚环导走身上的静电。

对于没办法避免产生的静电，机房大多采用接地措施将其导走。机房内的导体通过接地与大地连接，没有对地绝缘的孤立导体。导静电地面、活动地板、工作台面和坐椅垫套都进行静电接地[81,88]。

此外，湿度对于静电的产生和导走也有很大影响。这也是机房内控制相对湿度的原因之一。（在国家标准《电子计算机机房设计规范》和《计算站场地安全要求 GB 9361—88》中有针对防止静电做的相应规定。）

小提示

静电会造成芯片损坏、计算逻辑错误。

为防止静电，机房使用不易产生静电的材料，采用设备接地将已产生静电的导走，并保证合适的湿度。

有时候设备出现问题，怎么也找不到原因，可是最后只是把引线重新整理一下，甚至只是把它挪动个地方，问题就解决了。这种让人感觉莫名其妙的事情就是电磁干扰（Electro Magnetic Interference，EMI）惹的祸。

电磁干扰是指电磁信号通过传导、感应、辐射等方式对设备形成干扰的现象。机房中的电源、元件、导线、接头、散热风扇、日光灯，以及雷电和静电放电都可能产生电磁干扰。电磁干扰可能会使正在执行的程序中断、磁盘出现数据存取错误、显示器的显示内容混乱、打印机夹纸、存储器数据丢失，还可能导致主板上的芯片被烧毁。所以，《计算站场地技术条件》规定主机房内无线电干扰场强，在频率为 0.15～1000MHz 时，不应大于 126dB。主机房内磁场干扰环境场强不应大于 800A/m。因此，人们在建设机房时都采取了一系列措施尽量避免或减弱电磁干扰。

人们在设计机房设备安装布局时，除了美观、操作方便之外，还考虑了电磁兼容性。容易产生干扰信号的干扰源与易受感应的敏感设备会被分开并尽可能离得远。信号线与电源线、低频线与高频信号线布线都会被分开，距离都在 20cm 以上。一般机房里布线时都会避免不同信号线并行或可能有干扰信号的引线并行。平行布线长度被尽量缩短。易受感应的引线、部位会使用接地良好的屏蔽线或屏蔽盒进行静电及电磁屏蔽。一般增加新设备时也都优先选用抗干扰能力强的设备。通常集成度高的设备要比集成度低的设备抗干扰能力强。维修人员在维修设备时，会尽量避免改变原电路和引线的排列结构[93,94]。

小提示

电磁干扰引起的错误比较难查找。

在设计机房设备安装布局时需要考虑电磁兼容性。

布线时要避免不同信号线并行或可能有干扰信号的引线并行。要尽量缩短平行布线的长度。

维修设备时应尽量避免改变原电路和引线的排列结构。

3. 防火和防水

火灾的危害自然不用讲了，尤其是对于计算机机房这个重要的地方。为了防火，机房内使用的各种材料都尽可能地选用不易燃烧的材料。例如，机房装饰材料选用非燃烧材料或难燃烧材料。围护结构的构造和材料也应满足防火要求。就是存放废弃物也应使用有防火盖的金属容器。存放记录介质的容器也应是金属柜或其他能防火的容器。

因为机房中分布的供电电缆是引起火灾的一个隐患，所以机房一般都优先选用铜芯电缆，而尽量避免铜铝导线混接。这是因为铜的导电率是铝的三倍，铜导线允许的载流量也是同等规格铝线的近三倍(导线在某种条件下的允许载流量是指保证导线在该种条件下长期运行而不至过热烧坏绝缘层所能承受的最大工作电流值)。因为铜铝的电化学性质不同，两者存在电极电位差异，如果混用连接，在其接头接触界面就会产生原电池反应，形成电化腐蚀而产生一种电阻率很高的金属氧化物，造成导体局部发热甚至烧断线路，引起火灾。机房的输入、输出用电电缆一般应选择实际载流能力比各端口实际最大工作电流值大一倍的规格品种，使其留有一半的余量。这样可以确保不因导线过载发热造成供电故障，或者发生火灾。

为了及时发现火情，机房应设置火灾自动报警系统和灭火系统。机房里采用的是二氧化碳或卤代烷灭火系统，因为水、干粉或泡沫等灭火剂会对设备产生二次破坏。机房吊顶的上下及活动地板下，都应设置探测器和喷嘴。机房内的电源切断开关应靠近工作人员的操作位置或主要出入口。空调系统所采用的电加热器应设置无风断电保护。

为保证发生火灾时，人员能够安全撤离，机房的安全出口一般不应少于两个，通常设在机房的两端，且机房出口的门都是向疏散方向开启的。

除了火之外，水也是计算机设备的一个克星。因为水是良导体，所以如果供电系统进水，就会使线路短路甚至引起火灾。计算机系统进水会使线路短路、设备损坏。因此，机房在建设和日常管理中都要非常注意防水。

机房中水的主要来源有三个：建筑物渗水、排水管道漏水、工作人员日常用水。如果机房设置在用水设备下层，那么吊顶上一定要设防水层，并设置漏水检查装置。如果机房内设有用水设备，那就要采取防止给排水漫溢和渗漏的措施。一般应暗敷给排水干管，引入支管也采用暗装形式。而且暗敷的给水管道一般应用无缝钢管，管道连接用焊接。与机房无关的给排水管道不穿过机房。管道穿过机房墙壁和楼板处，要设有套管。管道与套管之间应采取可靠的密封措施。如果机房内设有地漏，地漏下都要加设水封装置，并有防止水封破坏的措施。机房内的给排水管道应采用难燃烧材料保温。这样可以防止冻裂水管，同时也能避免引起火灾。机房一般禁止工作人员将生活用水带入机房，如必须带入，也要远离设备和电源装置。

小提示

机房要采用防火材料，要设置火灾自动报警系统和二氧化碳或卤代烷固定灭火系统。
机房要防止建筑物渗水、排水管道漏水，避免工作人员日常用水损坏设备。

4. 防鼠虫害和防盗

虫、鼠可能会啃咬设备、电缆，也可能会进入设备内部。它们的粪便和尸体会污染机房和设备内部，造成断路、断电、设备损坏等事故。所以机房应想办法防止虫鼠进入，并采取措施清除进入机房的虫鼠。

如果机房设置在老鼠比较猖獗的区域，一般在铺设线路时，要采用防鼠性能好的材料。机房内的电缆和电线上也要涂敷驱鼠药剂。为了防止虫、鼠进入，机房要密封，并且把各种洞、孔封堵掉。一般机房的管理规章会禁止在机房内用餐，以及在机房内放食品、饮料。如果机房中没有食物，虫、鼠实际上对硬邦邦的计算机设备也没有兴趣，它们就会离开机房了。有些机房内还可以设置捕鼠或驱鼠装置，如超声波驱鼠器，来对付闯入的老鼠。

说完老鼠和虫，最后来说说窃贼。因为机房中的设备都很昂贵，所以，很容易引起窃贼的兴趣。当然，计算机中存放的数据可能价值更大。不过仅仅机器硬件本身的价值对窃贼来说已经足够了。因此，机房应安装防盗设施，如安装防盗门防盗锁、使用门禁系统、自动报警系统、设置警卫等。禁止无关人员进入机房也是防盗的措施之一。

小提示

机房要有防鼠、捕鼠设施。
不要在机房用餐或者存放食物。
防盗对机房也很重要。

复习题

1. 巨型机、大型机一般用在什么地方？
2. RISC 和 CISC 是什么含义？
3. 集群技术分为哪三类？
4. 路由器是做什么用的？
5. 交换机的作用是什么？
6. 集线器、中继器、网桥有什么用？
7. 配置 Web 服务器硬件时要考虑哪些因素？
8. 为什么要设置单独的数据库服务器？
9. 邮件服务器对硬件有哪些要求？
10. 机房建在什么样的地方比较合适？什么样的地方不适宜作机房？
11. 温度对机房中的设备有什么影响？怎样控制机房的温度？
12. 为什么要控制机房的湿度？应该怎样控制？
13. 怎样做才能减少机房中的灰尘？
14. 为什么要考虑机房空气的洁净度？
15. 为机房选用空调要注意哪些事项？为什么不能用民用空调？
16. 计算空调的热负荷时，通常要计算哪些内容？
17. 电力负荷分为哪三级？三级不同电力负荷对供电系统分别有什么要求？
18. 什么是 UPS？主要有哪几种 UPS？它们分别是怎样工作的？
19. 什么是功率因子，为什么计算 UPS 负载时，要考虑设备的功率因子？
20. 空调为什么不能接入为主机供电的 UPS？
21. 机房防雷可以采取哪些措施？
22. 静电会带来什么危害？怎样可以减小静电的危害？
23. 怎样能够减小电磁干扰？
24. 为什么不要使用铜铝混接的电线？
25. 机房要使用什么样的灭火器？为什么不能使用干粉或者泡沫灭火剂？
26. 机房防火主要采取哪些措施？
27. 机房防水主要采取哪些措施？
28. 为什么不能在机房中吃食物？

讨　论

请到网络上查阅资料，讲讲怎样构建一个集群系统，目前常用的集群系统有哪些？

实　验

1. 参看有关机房标准，检查你所在单位的机房是否有不符合规范的地方，应该怎样改正。
2. 请说出你所在单位机房中所有设备的名字，并讲出它们的主要功能。

附录 A　Fedora 9 安装过程

首先我们要用安装光盘启动计算机。打开计算机电源后，把 DVD 光盘放到光驱中。如果计算机设置的第一启动设备是硬盘，那么需要在启动计算机时，进入 BIOS 设置，将系统的启动顺序更改为光盘是第一个启动设备。做好这些工作后，屏幕上会显示出如图 A-1 所示的光盘启动界面。

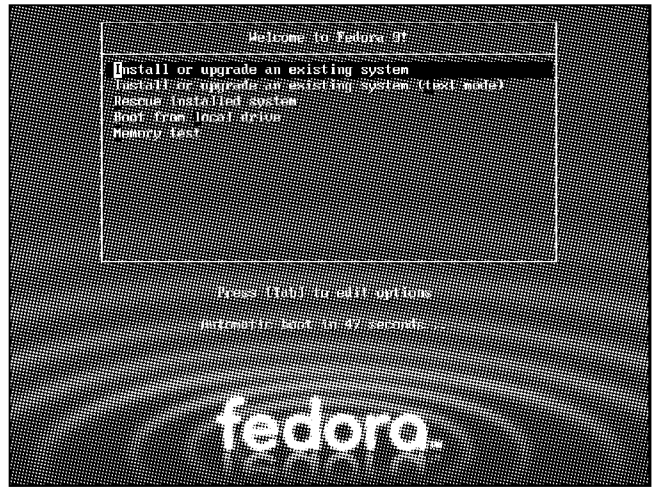


图 A-1　Fedora 9 安装启动界面

选择第一项进入安装或升级现有系统。从图 A-1 上可以看到还有其他的一些选项，这里我们只介绍怎样进行首次安装 Fedora 9 系统。

A.1　检测光盘

选择第一项进入安装或升级现有系统后，屏幕会显示出一个界面询问是否要对安装光盘进行检测。因为如果光盘有问题，安装过程就要重新开始，所以系统在安装前提供一个检测。这样如果光盘有问题就可以不开始安装过程，先解决有问题的光盘。如果决定检测，那么就选择 OK，否则就选择 Skip 跳过检测继续后面的安装进程。

如果选择 OK 进行光盘检测，那就需要等待计算机对光盘进行检测。检测完毕，屏幕上会给出结果。

A.2　识别环境

盘片检测完毕，屏幕上就会出现系统安装的图形主界面。如果选择 Skip 跳过检测，那么同样会看这个主界面。不过，如果安装程序不认识计算机的显卡或者显示器，或者计算机

的内存太小，那么它就只能给出字符界面而不是图形界面了。字符界面的功能和标准图形屏幕的功能相同，同样可以完成安装。只是在安装完成后，需要自己动手安装显卡驱动程序来启动图形界面。

进入图形主界面后，安装程序会显示 Fedora 支持的语言，如图 A-2 所示。一般选择中文。选好之后单击 Next 按钮，安装程序会显示 Fedora 支持的键盘格局。这时界面就是中文的了。我们在其中选择出自己计算机的键盘类型。选好之后，单击“下一步”按钮即可。

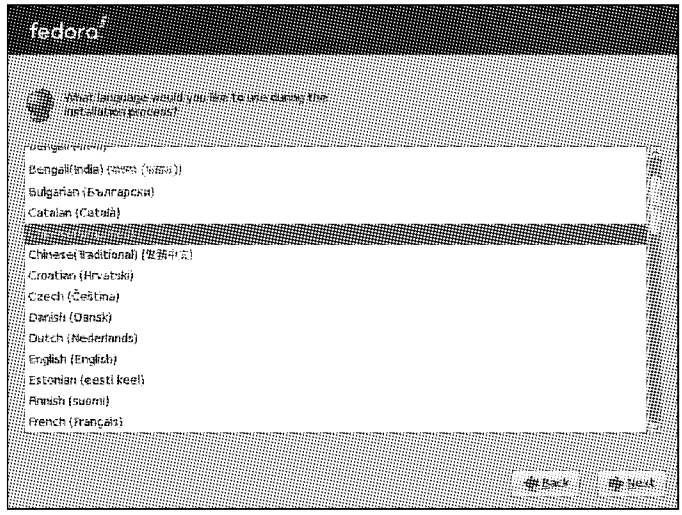


图 A-2　安装语言选择界面

A.3　网络配置

选择好键盘类型之后，Fedora 9 会给出界面要求进行网络配置，如图 A-3 所示。

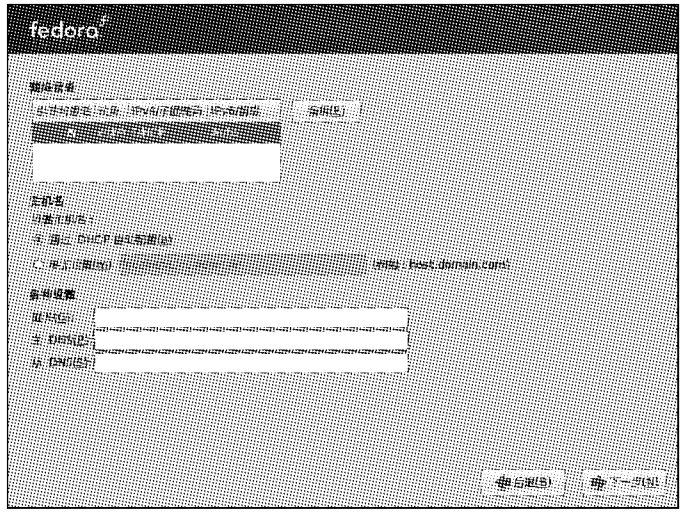


图 A-3　网络配置界面

Fedora 通常不需要手工进行网络配置。很多网络都有 DHCP 服务。DHCP 服务是一种网络服务。DHCP 的主要任务是为计算机分配一个 IP 地址以及提供其他必要的配置信息。通常提供 DHCP 服务的网络会在网络中设置一个 DHCP 服务器。有了 DHCP 服务，用户就不用再配置 IP 地址等信息了，但是必须在系统配置时指明计算机使用 DHCP 服务。这样计算机就会在局域网中寻找 DHCP 服务器，然后得到配置信息自动进行配置了。

默认情况下，Fedora 激活计算机上的所有网络接口，并用 DHCP 对它们进行配置。如果计算机所在的网络提供 DHCP 服务，那么计算机就可以上网了。

安装程序会显示出在计算机上检测到的所有网络接口。每个接口在它所连接的网络中必须有一个唯一的 IP 地址。接口可以从 DHCP 服务器获得这个地址。如果想手工给某个接口分配 IP 地址，可以把光标焦点移动到网络设备列表中的该接口上，然后单击“编辑”按钮。Fedora 9 安装程序就会显示网络配置对话框，如图 A-4 所示。选择“手动配置”，地址项就由灰变亮，即可输入接口的 IP 地址和网络掩码了。然后单击“确定”按钮即可。

图 A-4　网络地址配置界面

如果计算机要作为服务器，那就不能使用 DHCP 服务，一定要手工配置网络。因为这样，即使 DHCP 服务器出现问题，服务器也仍然能够接入网络。

在某些网络中，DHCP 也为计算机提供主机名。如果想自己指定主机名，在图 A-3 所示界面中选择“手工设置”，然后在文本框中输入主机名即可。完整的主机名包括机器的名字以及机器所在域的名字。

除了接口的 IP 地址外，Fedora 9 还需要我们提供其他一些网络设置信息，见图 A-3 的“各种设置”。网关是一个让我们的计算机访问其他网络的设备，是计算机所在网络与其他网络连接的关口。如果我们的计算机通过网关连接到别的网络，那么就要在网关的文本框中输入它的地址。这个信息需要询问计算机所在网络的管理员。

DNS 也是一种标准的网络服务。它负责把主机名转换成 IP 地址，或者反过来。很多软件依靠 DNS 服务器来定位网络中的机器和服务。Fedora 系统可以使用多个 DNS 服务器。如果主 DNS 服务器没有应答，那么计算机会向从 DNS 服务器提出查询。要指定 DNS 服务器，在主、从 DNS 服务器的文本框中分别输入这些服务器的 IP 地址。同样这些信息需

要询问网络管理员，或者查看网络中其他计算机上的配置。因为在同一网络中，这些信息都是一样的。

在图 A-3 中把这些信息都配置好之后，单击“下一步”按钮即完成网络配置。

A.4　时区配置

配置好网络之后，接下来安装程序就会让我们指定计算机所在位置的正确时区，如图 A-5 所示。可以在地图上指定时区，也可以通过在列表中选择离我们最近的城市来指定时区。

图 A-5　时区配置界面

在这一屏上可以看到有一个复选框叫“系统时钟使用 UTC”。UTC 是统一协调时间的英文缩写(Universal Coordinated Time)。这个时间也叫做格林威治时间(GMT)。系统时钟是计算机系统硬件的一部分。如果 Fedora 是计算机上唯一的操作系统，那么就选择“系统时钟使用 UTC”。Fedora 就会用时区设置判定本地时间和系统时钟 UTC 之间的时间差。这是类 UNIX 操作系统的标准做法。但是，如果计算机上也运行 Windows 操作系统，那么就不要选“系统时钟使用 UTC”。微软的操作系统会把 BIOS 时钟改变成与当地时间一致，而不是与 UTC 一致。这可能会引起 Fedora 意想不到的行为[47]。

设置好时区之后，我们单击“下一步”按钮继续下面的安装。

A.5　root 密码设置

接下来安装程序要求我们设置根账号 root 的密码，如图 A-6 所示。Fedora 用根账号 root 进行系统管理。在 Linux 系统中多数普通账号都必须服从一些特定的约定。它们的权力也都有一定的范围。root 账号是一个特权账号，它不受约定的限制。作为系统的主人或者管理员，有时候需要特权来配置或者更改系统。在这些情况下，需要使用 root 账号。不过一般应该尽量避免以 root 身份登录 Fedora。因为它的权力实在太大了，一些误操作会产

生很大的杀伤力。所以，通常都以普通账号登录系统。而不是用 root 账号登录，每时每刻都以一个特权身份做事。需要 root 权限的管理工具都会提示用户输入 root 密码。在正确输入 root 密码后就可以使用这些管理工具了。

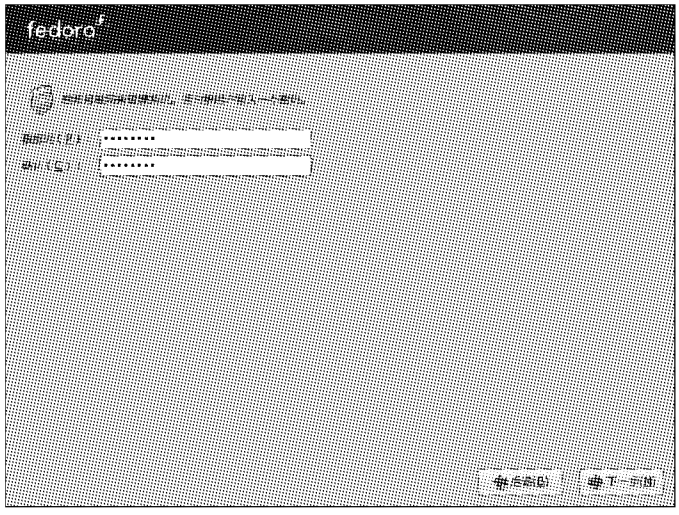


图 A-6　root 密码配置界面

Fedora 9 安装程序要求 root 密码至少为 6 个字符。密码设置一些基本建议如下：

- 使用大小写字母、数字、标点符号以及其他字符的组合。
- 不要使用单词或者名字。用替代符来弄乱名字或者单词一般没有作用。
- 不要在多个系统上使用相同的密码。

设置好密码后，单击“下一步”按钮继续安装。

A.6　磁盘分区

接下来，安装程序会对硬盘进行分区。屏幕上会显示出如图 A-7 所示界面。Fedora 9 通常会创建和使用若干个分区。屏幕中间的方框内列出了所有可用的硬盘驱动器。默认情况下，安装会影响计算机上所有的硬盘。为了确保无关的硬盘不受影响，要把无关的那些硬盘前面的对勾去掉（图 A-7 中只有一个硬盘驱动器）。

安装过程会删除选定硬盘（驱动器）上的所有现有 Linux 分区，然后用 Fedora 的默认分区取代。其他类型的分区，如 Windows 使用的分区或者计算机厂商创建的系统恢复分区等，不会被改动。安装程序还提供了其他的分区方式。把图 A-7 中的下拉菜单打开就可以看到这些选项。它们是：

（1）在选定驱动上删除所有分区并创建默认分区结构

除非希望删除掉选定驱动器上全部现有的操作系统和数据，否则千万不要选择这一项。

（2）使用选定驱动器上的空闲空间并创建默认的分区布局

如果选定驱动器上有空闲空间，这个选项会把 Fedora 安装空闲空间中。这能保证现存空间不会被安装程序修改。

图 A-7　分区界面

(3) 创建自定义分区结构

这一项要求自己动手制定选定驱动器上的分区。如果选择这一项，Fedora 不会创建任何分区，接下来的屏幕会让用户来配置驱动器和分区。

选择好驱动器和适当的分区选项后，单击“下一步”按钮，继续安装过程。

在继续安装过程之前，要对有关 Fedora 分区的情况做些说明。Fedora 系统至少要有三个分区：一个挂载在目录“/boot”的数据分区、一个挂载在目录“/”的数据分区以及一个交换分区。除这三个分区外，还可以根据需要定制分区。例如，在存储用户数据的系统中创建独立的“/home”分区。每个分区都有一个挂载点。没有挂载点的分区是不能被用户访问的。不在其他分区中的数据都处于根分区“/”下。

如果不知道怎样分区，那么接受默认的分区结构就行了。

Linux 系统用交换分区来扩展内存。当内存不够用时，Linux 会自动在内存和交换分区之间移动内存中的内容。此外，电源/能源管理系统也会把所有挂起的内容存在交换分区中。如果选择自己来指定交换分区，那么要创建容量比内存大的交换分区。

数据分区为文件的存储提供空间。在前面提到的最小分区方案下，在 Linux 系统/boot 目录下的所有数据都在/boot 分区下。例如，文件/boot/grub/grub. conf 就在/boot 分区中。/boot 目录下存放与系统启动相关的文件。所有/boot 分区之外的文件，如/etc/passwd 都在根分区下。

Fedora 也支持把子目录分配给一个分区。例如，一些管理员会创建/usr 和/usr/local 分区。在这种情况下，在/usr/local 目录下的文件，如/usr/local/bin/foo 就在/usr/local 分区中。而其他在/usr 目录下的文件，如/usr/bin/foo，就在/usr 分区中。(如果我们创建了很多分区，而不是一个大的根分区，那么升级就会容易一些。)

每个分区都有一个分区类型。它指明该分区上的文件系统类型。如果没有其他文件系统类型的需求，那么就为数据分区选择 ext3 文件系统。

Fedora 提供了一种对磁盘分区进行管理的机制称为 LVM(Logical Volume Manager，

逻辑卷管理器)。LVM 是建立在硬盘和分区之上的一个逻辑层,用来提高磁盘分区管理的灵活性。

采用 LVM,分区首先被格式化成物理卷。多个物理卷被组合成一个卷组。每个卷组的全部存储空间然后被分成一个或多个逻辑卷。逻辑卷的功能和非 LVM 的标准分区很像。它们也有文件系统类型和挂载点。只有处于活动状态的 Linux 系统才能读写逻辑卷。因此,初始化系统的/boot 分区必须放在 LVM 物理卷之外。

LVM 最大的优点是,管理员可以扩大或者缩小逻辑卷的大小,但是并不破坏数据。这与标准磁盘分区不一样。不过如果逻辑卷的大小缩到比数据所需的空间小,那就可能会丢失数据。可以创建满足当前要求的逻辑卷,留下额外的空间不分配。这样就可以很安全地按照需要用未分配空间增加逻辑卷的大小[47]。

默认情况下,安装过程会在 LVM 卷中创建数据和交换分区,外加一个卷外的/boot 分区。如果希望自己或者其他用户在系统中存储数据,那么就在卷组中创建一个单独的/home 分区。有了单独的/home 分区,就可以升级或者重新安装 Fedora,但是并不会删除用户数据文件。

如果在图 A-7 所示安装界面的下拉菜单中选择“创建自定义分区结构”来创建自定义分区,就会看到如图 A-8 所示的界面。这个界面可以创建和编辑磁盘分区。

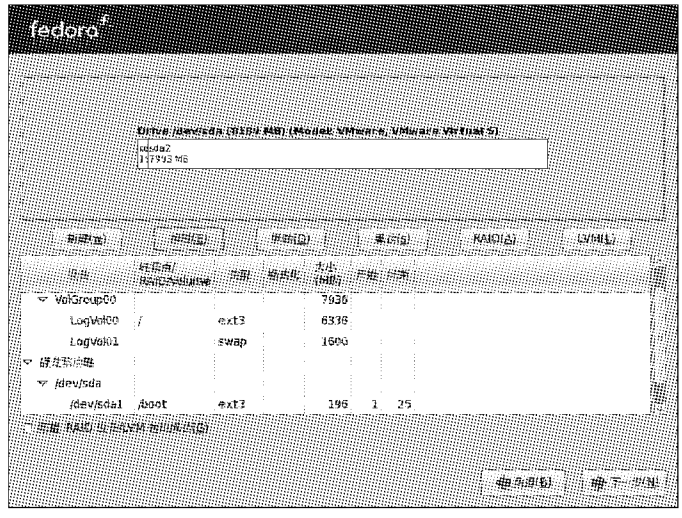


图 A-8 自定义分区结构界面

图 A-8 中可以看到关于分区定制 Fedora 安装程序支持的动作如下(图 A-8 中间的一排按钮):

(1) 新建

让我们向硬盘中增加分区。单击图 A-8 中间的“新建”按钮,就会看到如图 A-9 所示的添加分区的对话框。在添加分区对话框中,需要选择一个加载点和一个分区类型。如果系统中有多个盘,就还要选择放置这个分区的盘。然后以兆字节(MB)为单位指明分区的大小。有三种方式可以用来设置分区的大小:固定大小、指定大小、使用全部可用空间。盘上分区的实际大小可能会比指定的稍稍大或者小一点儿。这是由于盘的表面形状造成的,不是错误或者 bug。

图 A-9　添加分区界面

在这里，目录/bin、/dev、/etc、/lib、/proc、/root 和/sbin 不允许作为单独的分区。这些目录必须在根/分区中。/boot 分区不能存在于 LVM 卷组中。要在配置卷组之前创建/boot 分区。

(2) 编辑

对于已经存在的分区、LVM 卷组或者还不属于某个卷组的 LVM 物理卷，可以使用编辑选项进行修改。要改变 LVM 物理卷分区的大小，要先把它从卷组中删除。如果把一个 LVM 物理卷从卷组中删除，那么它含有的全部逻辑卷也被删除了。

编辑分区可以改变分区大小、挂载点以及文件系统类型。我们可以用这些功能来：

- 修正建立分区时的错误。
- 移动 Linux 分区，如果正在升级或者重新安装 Fedora。
- 为非 Linux 分区，如某些在 Windows 操作系统中使用的分区，提供一个挂载点。

在 Fedora 的安装程序中，不能用挂载点来标示使用 NTFS 文件系统的 Windows 分区。不过可以用挂载点来标示 vfat(FAT16 或者 FAT32)分区。如果想对分区进行非常大的改动，通常可能会要把分区删除然后重新开始。这时，如果盘上有要保留的数据，那么就要在编辑分区之前把它们备份出来。

(3) 删除

删除现存的分区或者 LVM 物理卷。要删除 LVM 物理卷，首先删除它所在的卷组。如果不小心选错了，可以使用“重设”选项来放弃所做的全部改变。

(4) 重设

让安装程序放弃对盘分区所做的一切改变。

(5) LVM

在 Fedora 系统上建立 LVM。首先，要用“新建”对话框创建至少一个分区或者软件 RAID 设备作为 LVM 物理卷。

要把物理卷分配到卷组中，首先要给卷组起个名字。然后选择卷组中使用的物理卷。最后，用“添加”、“编辑”和“删除”选项来配置卷组中的逻辑卷。

如果从卷组中删除某个物理卷会使卷组中逻辑卷的空间不够用，那么就不能删除这个物理卷。例如，一个卷组由两个 5GB 的 LVM 物理卷分区构成，它含有一个 8GB 的逻辑卷。安装程序就不会让我们删除任何一个物理卷，因为那样只能为 8GB 的逻辑卷留下 5GB 的空间。这时，只有把逻辑卷的空间减小，才可以从卷组中删除掉物理卷。例如，把逻辑卷的大小减到 4GB，那么就可以删除一个 5GB 的物理卷。

默认的分区布局把所有可用的空间都分到一个 LVM 物理卷中，并且整个系统只有一个 LVM 逻辑卷。

安装程序还提供一些设置软件 RAID 的选项，这里我们就不介绍了。

完成分区配置之后，就可以单击“下一步”按钮继续安装过程。这里我们创建了一个 /boot 分区、一个根/分区和一个交换分区。

A.7 配置引导装载程序

分区工作结束后，安装程序进入引导装载程序窗口，如图 A-10 所示。引导装载程序是一个小程序，它读取并启动操作系统。我们在系统启动一节中已经提到过。如果计算机上装有多个操作系统，引导装载程序通常会给出一个菜单，让用户选择启动哪一个操作系统。默认情况下，Fedora 使用名为 GRUB 的引导装载程序。

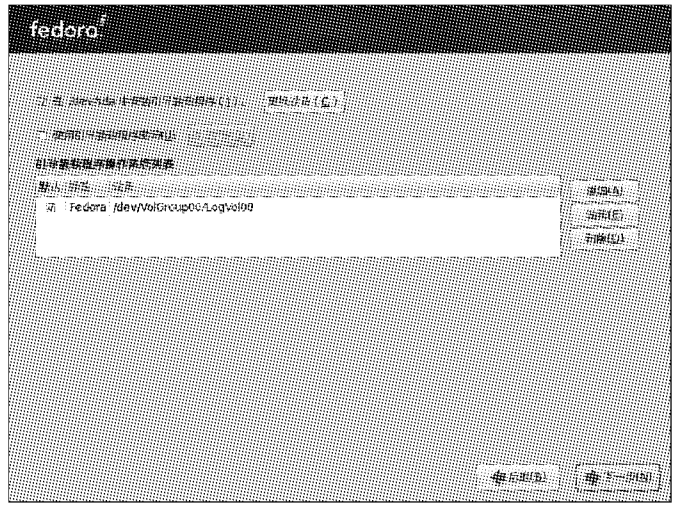


图 A-10 引导装载程序配置界面

除非创建独立的启动盘来引导系统，否则就必须安装 GRUB 或者别的引导装载程序，这样系统才能启动。不过，系统上有可能已经安装了别的引导装载程序。因为之前其他操作系统可能会安装它喜欢的引导装载程序，或者自己安装了第三方的引导装载程序。如果这些前面安装的引导装载程序不能识别 Linux 分区，那么就不能启动 Fedora。而 GRUB 可以启动 Linux 和绝大多数其他操作系统。但是，安装 GRUB 可能覆盖掉前面安装的引导装载程序。

图 A-10 显示引导装载程序的配置选项。默认情况下，安装程序会在根文件系统所在驱动器中安装 GRUB。

如果计算机上已经安装了其他操作系统，安装程序会自动检测并配置 GRUB 来启动它

们。如果没有检测到这些操作系统，可以进行手工配置。可以用安装程序提供的选项来添加、删除、改变检测到操作系统的设置。

GRUB 可以在没有操作系统的协助下读取很多文件系统。操作员可以中断启动过程选择不同的操作系统来启动，可以改变启动选项，或者从系统错误中恢复等。但是，这些功能在某些环境下可能会引入很严重的安全隐患。为此，安装程序允许我们给 GRUB 增加密码。这样操作员必须输入密码才能中断正常的启动过程。当然，如果系统有可信的操作员，或者物理上很安全，或者终端访问是受控制的，那么可能并不需要 GRUB 密码。但是，如果不可信的人可以接近计算机键盘和显示器，那他就可以重新启动系统并访问 GRUB。在这种情况下，设置密码是有用处的。

如果选择“使用引导装载程序密码”复选框设置启动密码。安装程序会弹出“改变密码”对话框。输入需要的密码，然后重新再输入一次来确认即可。

关于 GRUB 的高级设置可以到 Fedora 的官方网站 http://fedora.redhat.com 上去查询相关的说明。

A.8　选择要安装的软件

接下来的一屏安装程序要求选择想要安装的软件，如图 A-11 所示。默认情况下，Fedora 会加载被其选定的桌面系统需要的软件。可以通过在列表中选择相关的项来选上或者去掉这些常规软件。这些软件被分成三大部分。其中，办公和实用性部分是一些与办公相关的软件工具，如 OpenOffice.org 公司的产品套件，Planner 项目管理应用程序，图形工具如 GIMP，以及多媒体应用程序等。而软件开发部分则是一些在 Fedora 系统上编译软件必需的工具。网络服务器主要指 Apache Web 等服务器。

图 A-11　软件选择界面

可以选择“现在定制”来指定最终安装软件包的详细内容。选中这个选项单击“下一步”按钮时，就会看到软件定制界面，如图 A-12 所示。

Fedora 把软件分到不同的组中。为了方便使用，软件包选择界面把这些软件组分成

图 A-12　软件选择定制界面

6 个类目来显示。选定类目名，右侧框中就会列出这个类目中的软件包组名。屏幕下方的文本框中有当前焦点所在包组的详细信息。

可以选中软件包组名前面的复选框来指定要安装的软件包组。选中一个软件包组，Fedora 会自动安装这个组的基本和必要软件包。可以单击组描述框下方的“任选软件包”按钮，改变选定组中的选定软件包。在启动系统后，也可以用 Fedora 提供的工具安装新软件或者删除已经安装的软件包。具体操作过程是：从主菜单中，选择“系统”→“管理”→“增加/删除软件”。这时 Fedora 软件管理系统将从网络服务器上下载最新的软件包，而不是使用安装盘上的那些软件包。

选好所有软件包之后，单击“下一步”按钮继续安装。在进行软件依赖性检查之后，屏幕上出现如图 A-13 所示界面，Fedora 9 进入文件复制安装过程。这个过程一般要持续比较长的时间。这个过程结束后，会看到安装完毕提示界面，单击该界面上的“重新引导系统”按钮即可启动系统。安装程序会在计算机重新启动之前弹出光驱中的光盘。

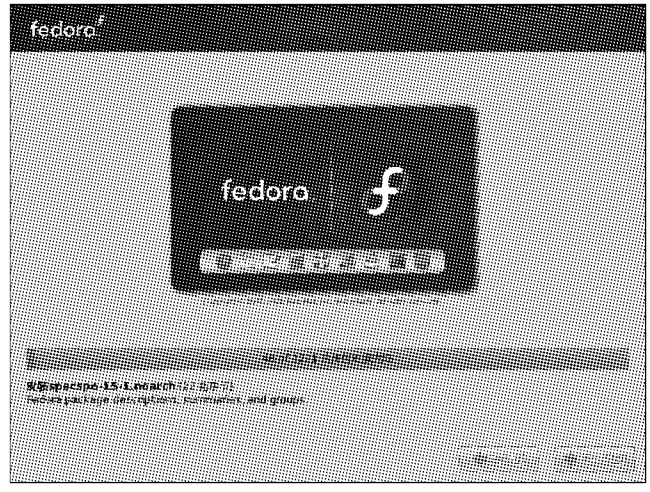


图 A-13　安装界面

重新启动计算机后，会首先看到 GRUB 的界面，在里面选择 Fedora 就行了。

A.9　安装完毕

Fedora 第一次启动后，需要进行一些设置，如设置系统时间、增加普通用户等，只要跟随配置启动代理一步一步配置就行了。配置完毕，用新增加的普通用户账号登录 Fedora，就可以使用 Fedora 了。

Fedora 在不断地推出新版本。不过各个版本具体安装过程大致差不多。可以到 http://fedoraproject.org/上去查询并下载新的版本。同时，在该网站上也有安装过程的详细说明文档供用户参考。

参 考 文 献

[1] 唐朔飞. 计算机组成原理. 北京：高等教育出版社，2004.

[2] 毛德操，胡希明. Linux 内核情景分析. 杭州：浙江大学出版社，2001.

[3] Gray Nutt 著. 罗宇，吕硕等译. 操作系统. 北京：机械工业出版社，2005.

[4] William Stallings 著. 魏迎梅等译. 操作系统——内核与设计原理(第 4 版). 北京：电子工业出版社，2002.

[5] 王红，梁国俊，沈祥玫. 操作系统原理及应用(Windows Server 2003). 北京：中国水利水电出版社，2005.

[6] 王红. 操作系统原理及应用(Linux). 北京：中国水利水电出版社，2005.

[7] 徐德民编著. 操作系统原理 Linux 篇. 北京：国防工业出版社，2004.

[8] 蒋静，徐志伟. 操作系统——原理·技术与编程. 北京：机械工业出版社，2004.

[9] 陈向群，向勇等. Windows 操作系统原理. 北京：机械工业出版社，2004.

[10] 孙钟秀. 操作系统原理. 北京：高等教育出版社，2003.

[11] 微软 Windows 操作系统发展简介，www. softhouse. com. cn/news/show/4320. html.

[12] 朱居正，高冰. RedHat Linux Fedora Core 5 系统管理. 北京：清华大学出版社，2006.

[13] 李善平等. Linux 内核 2.4 版源代码分析大全. 北京：机械工业出版社，2002.

[14] 刘卫东等. 计算机组成与结构. 北京：机械工业出版社，2003.

[15] 袁开榜，王纪成编. 计算机原理. 北京：清华大学出版社，1993.

[16] 蒋本珊. 计算机组织与结构. 北京：清华大学出版社，2002.

[17] 李文兵. 计算机组成原理. 北京：清华大学出版社，2006.

[18] 殷肖川，秦莲等. 汇编语言程序设计. 北京：清华大学出版社，2005.

[19] 胡越明. 计算机组成与系统结构. 上海：上海交通大学出版社，2002.

[20] 王诚. 计算机组成原理. 北京：清华大学出版社，2004.

[21] 硬盘知识接口类型. http://detail. zol. com. cn/product_param/index567. html.

[22] 跟小蔡学数据恢复(二). http://www. fix. com. cn/news/ReadNews. asp? NewsID=81&BigClassID=5&SmallClassID=9&SpecialID=6.

[23] 顾学文. 光盘存储发展历史及原理概述. http://202. 113. 227. 136/grzy/fsong/laser/finals/gxw. pdf.

[24] 刘为，李光平. CRT 显示器的工作原理. http://arch. pconline. com. cn/pchardware/salon/monitor/10303/138780. html.

[25] 系统总线的历史. http://pcpro. com. cn/topic. php? id=7237.

[26] 详解 USB2. 0 规格. http://tech. tom. com/hardware/usb20. html.

[27] IT168 术语解释-内存术语-接口类型. http://publish. it168. com/cword/1120. shtml.

[28] IT168 术语解释-主板术语-支持内存类型. http://publish. it168. com/cword/978. shtml.

[29] 光盘存储原理及相关标准. http://www. 1sao. cn.

[30] 浅谈闪盘的选购与维护. http://www. hackhome. com/InfoView/Article_87451. html.

[31] IT168 术语解释-CRT 显示器术语-接口类型. http://publish. it168. com/cword/1420. shtml.

[32] 激光打印机工作原理. HP 中国打印与成像专业技术中心. htm.

[33] 激光打印机通用参数详解. http://bbs. oczone. cn/thread-4904-1-1. html.

[34] 理性成本分析 原装硒鼓选购指南篇. http://www. 21tx. com/oa/2004/09/29/11482. html.

[35] 光电鼠标基础知识浅解. http://www. highdiy. com/html/misc/product/304_1. shtml.

[36] 用这么高的DPI你需要吗——鼠标光学引擎详解. http://diy.yesky.com/input/114/2511614_1.shtml.
[37] IT168术语解释-鼠标术语详解-接口类型. http://publish.it168.com/cword/2196.shtml.
[38] 轻舞飞扬的金手指——鼠标与键盘的世界（五）键盘篇. http://www.peripc.com/news/000000043U1.asp.
[39] 张雪峰,李荣源,袁海文. 一种用N＋1个I/O口实现的N×N矩阵式键盘. http://www.laogu.com/wz_4808.htm.
[40] GB2312—80信息交换用汉字编码字符集 基本集.
[41] 统一码 Unicode. http://www.unicode.org.
[42] 程序员趣味读物：谈谈Unicode编码. http://tech.163.com/05/0516/10/1JS9KEGA00091589_2.html.
[43] 计算机启动过程详解. hi.baidu.com/skan/blog/item/60d82f2ea0df85504fc22694.html, www.poptool.net,电脑报,2005专题热报.
[44] 胡小龙. Linux操作系统启动过程分析. 长沙铁道学院学报,1999年12月,第17卷第4期.
[45] BIOS与CMOS区别. http://club.beareyes.com.cn/blog/?action-viewthread-tid-7650.
[46] 朱跃明,何水来,唐曦. BIOS与Windows注册表实战秘籍. 北京：科学出版社,2005.
[47] Fedora Core安装. http://fedora.redhat.com/docs/fedora-install-guide-en/fc5/sn-hwarchitecture.html.
[48] Fedora Core安装. http://www.cse.unsw.edu.au/～hfeng/.
[49] bash编程. http://thns.tsinghua.edu.cn/jsj00002/unix2.htm.
[50] Ellie Quigley著. 刘洪涛译. UNIX Shell范例精解. 北京：清华大学出版社,2004.
[51] Bash和Bash脚本. http://xiaowang.net/bgb-cn/ch01s03.html.
[52] Shell编程基础. http://www.unix-center.net/bbs/viewthread.php?tid=2165&extra=page%3D1.
[53] bash入门基础. http://fanqiang.chinaunix.net/a4/b1/20010416/113830.html.
[54] shell的算术运算. http://linux.tnc.edu.tw/techdoc/shell/c860.html.
[55] Kdevelop介绍. http://www.kdevelop.org/, http://docs.kde.org/development/en/kdevelop/kdevelop/.
[56] Peter Maurer著. 组件级编程. 施诺译. 北京：清华大学出版社,2003.
[57] 张桂元,贾燕枫编著. Eclipse开发入门与项目实践. 北京：人民邮电出版社,2006.
[58] 贺武,汪厚祥,濮心洗. 基于Struts框架MVC模式的Web应用的实现. 舰船电子工程,V0I_27 No.6,142.
[59] Jame F. Kurose, Keith W. Ross. Computer Networking—a top-down approach featuring the Internet (Third edition)(影印版). 北京：高等教育出版社,2005.
[60] Andrew S. Tanenbaum. 计算机网络. 潘爱民译(第4版). 北京：清华大学出版社,2004.
[61] SMTP协议. http://www.vanemery.com/Protocols/SMTP/smtp.html.
[62] SMTP协议. http://www.trilug.org/～jonc/mailserver/PartI.html.
[63] 网线制作. http://www.cfan.com.cn/school/net/lan/2006-11-21/1164079246d24370_2.shtml.
[64] 互联网地址. http://edu.cn/20051121/3161915.shtml.
[65] 科学社区：世界上七台最强大超级计算机(组图). http://it.sohu.com/20070629/n250825360_6.shtml.
[66] 大型机学习之初步了解——什么是大型机. tag.csdn.net/Article/5fea4f47-3e34-40e7-96db-5d6913403432.html.
[67] 朱颖. 廉颇虽老尚能饭：大型机仍有生命力. http://www.pconline.com.cn/servers/news/0602/761712.html.
[68] 什么是小型机？概念与特点及其技术发展趋势. www.cbismb.com/articlehtml/52748.htm.

[69] 曙光在国内小型机市场取得突破. http://www.21tx.com/server/2005/12/21/10696.html.
[70] CPU 体系结构 CISC 与 RISC 之争. zih.it.com.cn/articles/83563.htm.
[71] 曙光第二代小型机. http://www.dawning.com.cn/prolist.asp?class=46&iclass=46.
[72] 精简指令集. http://wiki.ccw.com.cn/精简指令集.
[73] 计算机集群技术概述. http://www-900.ibm.com/cn/support/nav/200201/p56.shtml.
[74] Linux 集群系统大比拼. http://docs.huihoo.com/linux/linux_cluster_system.html.
[75] 宋小光. 教育城域网数据中心的硬件环境建设. 教育信息化,2004,9.
[76] 李蔚泽. Linux 架站与网管. 北京:机械工业出版社,2006.
[77] 唐红亮,陶秀,彭育斌. 计算机机房管理教程. 北京: 电子工业出版社,2006.
[78] 方刚,于晓宝. 计算机机房管理. 北京: 清华大学出版社,2005.
[79] 网管员世界. 扫除机房灰尘劲敌 延长服务器使用寿命. http://www.ccw.com.cn/server/yyjq/htm2007/20070620_274754.shtml.
[80] 机房系统. http://www.tycomputer.com/jszc.htm.
[81] 中华人民共和国国家标准电子计算机机房设计规范 GB 50174—93.
[82] 中华人民共和国国家标准计算站场地技术条件 GB 2887—89 代替 GB 2887—82.
[83] 什么是全面的机房建设解决方案(2). 网管员世界. http://www.365master.com.
[84] 供配电系统设计规范. GB 50052/95. 中华人民共和国国家标准.
[85] 潘京. 企业信息中心机房供配电系统. 工程建设与设计,2004,3.
[86] 周健. 浅析机房供配电系统各项规范要求. 华南金融电脑,2006,10.
[87] 李晋汾. 关于计算机房工程电气设计的探讨. 山西农机,[学术版],2002,3.
[88] 计算站场地安全要求 GB 9361—88.
[89] 王森,高洪军. 浅析计算机房的环境. 计算机与农业,2000,11.
[90] 刘万松. 有线电视中心机房的静电防护. 中国有线电视,2003,9.
[91] 赵旭东. 雷电对计算机网络的危害及其预防. http://www.cra-ccua.com.cn/article/articledetail.asp?id=79.
[92] 鲍岳建. 浅谈防直击雷接闪器与防雷电感应的关系. http://www.cma-lpinfo.org:81/servlet/Node?Node=5857.
[93] 罗兵. 机房的电磁干扰及屏蔽. 科技信息,2006,7: 30.
[94] 梁春廷. 浅析机房中的电磁干扰问题. 导弹试验技术,2003: 14.
[95] UPS 技术综述. http://publish.it168.com/2002/0409/20020409004101.shtml.
[96] 前端 UPS 电源设计与维护. http://publish.it168.com/2004/0413/20040413007501.shtml.
[97] 冯志强. 计算机机房热负荷计算. www.5k5k.net/air/air1/air_13458.shtml.
[98] 机房专用空调与民用空调技术差异的对决. www.ktcn.cn.
[99] David Gallardo. Eclipse 平台入门. http://www.ibm.com/developerworks/cn/linux/opensource/os-ecov/.
[100] 张云涛,龚玲. Eclipse 精要与高级开发技术. 北京: 电子工业出版社,2005.
[101] Pawel Leszek. 用 Eclipse 平台进行 C/C++ 开发. http://www.ibm.com/developerworks/cn/linux/opensource/os-ecc.
[102] David Gallardo. 开发 Eclipse 插件. http://www.ibm.com/developerworks/cn/java/os-ecplug/.

读者意见反馈

亲爱的读者：

感谢您一直以来对清华版计算机教材的支持和爱护。为了今后为您提供更优秀的教材，请您抽出宝贵的时间来填写下面的意见反馈表，以便我们更好地对本教材做进一步改进。同时如果您在使用本教材的过程中遇到了什么问题，或者有什么好的建议，也请您来信告诉我们。

地址：北京市海淀区双清路学研大厦 A 座 602　　计算机与信息分社营销室 收
邮编：100084　　电子邮件：jsjjc@tup.tsinghua.edu.cn
电话：010-62770175-4608/4409　　邮购电话：010-62786544

教材名称：计算机系统平台

ISBN：978-7-302-19722-5

个人资料

姓名：_____ 年龄：_____ 所在院校/专业：_____

文化程度：_____ 通信地址：_____

联系电话：_____ 电子信箱：_____

您使用本书是作为：☐指定教材 ☐选用教材 ☐辅导教材 ☐自学教材

您对本书封面设计的满意度：

☐很满意 ☐满意 ☐一般 ☐不满意　改进建议_____

您对本书印刷质量的满意度：

☐很满意 ☐满意 ☐一般 ☐不满意　改进建议_____

您对本书的总体满意度：

从语言质量角度看 ☐很满意 ☐满意 ☐一般 ☐不满意

从科技含量角度看 ☐很满意 ☐满意 ☐一般 ☐不满意

本书最令您满意的是：

☐指导明确 ☐内容充实 ☐讲解详尽 ☐实例丰富

您认为本书在哪些地方应进行修改？（可附页）

_____

_____

您希望本书在哪些方面进行改进？（可附页）

_____

_____

电子教案支持

敬爱的教师：

为了配合本课程的教学需要，本教材配有配套的电子教案（素材），有需求的教师可以与我们联系，我们将向使用本教材进行教学的教师免费赠送电子教案（素材），希望有助于教学活动的开展。相关信息请拨打电话 010-62776969 或发送电子邮件至 jsjjc@tup.tsinghua.edu.cn 咨询，也可以到清华大学出版社主页（http://www.tup.com.cn 或 http://www.tup.tsinghua.edu.cn）上查询。