

计算机科学与技术规划教材

Java程序设计

基础教程

王 巍 编著



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

内 容 简 介

本书系统地介绍了Java语言的基本概念、基本语法、基础类库、图形用户界面、Applet小应用程序、Java多线程、网络通信编程、基础应用编程及虚拟现实技术。

本书内容丰富,结构合理,通俗易懂。理论基础与程序实例相结合。本书可作为高等院校计算机专业“Java语言程序设计”课程的教材,也可作为初学者的自学参考书。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有,翻版必究。

图书在版编目(CIP)数据

Java程序设计基础教程 / 王巍编著. —北京:电子工业出版社 1998.6

计算机科学与技术规划教材

ISBN 7-5053-5227-1

I. J… II. 王… III. 计… IV. TP.238

中国版本图书馆CIP数据核字(1998)第023123号

丛 书 名: 计算机科学与技术规划教材

书 名: Java程序设计基础教程

主 编: 王 巍

主 审: 高丙云

责任编辑: 徐晓光

排版制作: 电子工业出版社计算机排版室

印 刷 者:

装 订 者:

出版发行: 电子工业出版社

北京市海淀区万寿路173 信箱 邮编100036

经 销: 各地新华书店

开 本: 787×1 092 1/16 印张: 11.5 字数: 294 千字

版 次: 1998年6月第1版 1998年6月第1次印刷

书 号: 7-5053-5227-1/TP.238

印 数: 6 000 册 定价: 30.00 元

凡购买电子工业出版社的图书,如有缺页、倒页、脱页、所附磁盘或光盘有问题者,请向购买书店调换;
若书店售缺,请与本社发行部联系调换。电话 68279077

目 录

第一章 Java 概述	(1)
1.1 Java 语言的起源	(1)
1.2 Java 和国际互连网	(1)
1.3 Java 语言的结构	(2)
1.3.1 Java 的层次结构	(2)
1.3.2 Java 语言的开发流程	(2)
1.4 Java 语言的优点	(3)
1.5 本书的结构和安排	(4)
1.6 小结	(5)
第二章 Java 开发环境及简单示例	(6)
2.1 硬件要求	(6)
2.2 软件安装	(6)
2.2.1 基于 Intel x86 Windows NT/Windows 95	(6)
2.2.2 基于 SunSparc/Solaris 2.x	(8)
2.3 JDK 工具的使用	(8)
2.3.1 Java 编译器	(8)
2.3.2 Java 解释器	(9)
2.3.3 Appletviewer	(10)
2.4 两类简单的 Java 程序	(10)
2.4.1 Applet 应用	(11)
2.4.2 Stand-alone 应用	(14)
2.4.3 错误诊断	(16)
2.5 Java 程序调试	(17)
2.6 几种支持 Java 的 WWW 浏览器	(19)
2.6.1 Netscape Navigator	(19)
2.6.2 HotJava	(20)
2.6.3 Microsoft Internet Explorer	(21)
2.7 小结	(21)
第三章 Java 语法要点	(22)
3.1 面向对象的概念初步	(22)
3.1.1 什么是对象 (Object) 和类 (Class)?	(22)
3.1.2 什么是消息 (Message)?	(23)
3.1.3 什么是继承 (Inheritance)?	(24)
3.2 Java 程序的类定义	(24)

3.2.1	Java 程序的一般结构.....	(25)
3.2.2	Java 的类定义语法.....	(26)
3.2.3	类的使用.....	(27)
3.3	变量和数据类型.....	(28)
3.3.1	变量类型.....	(28)
3.3.2	变量的命名规则.....	(29)
3.3.3	变量的作用范围.....	(29)
3.3.4	变量初始化.....	(30)
3.4	数组和字符串.....	(30)
3.4.1	字符串.....	(30)
3.4.2	数组.....	(30)
3.5	操作符.....	(31)
3.5.1	算术运算操作符.....	(31)
3.5.2	关系和条件操作符.....	(32)
3.5.3	位操作符、逻辑操作符.....	(32)
3.5.4	赋值操作符.....	(33)
3.6	表达式.....	(33)
3.7	流控制语句.....	(34)
3.7.1	分支语句.....	(34)
3.7.2	循环控制语句.....	(35)
3.7.3	异常处理语句.....	(35)
3.7.4	其它.....	(37)
3.8	小结.....	(37)
第四章	Java 基础类库.....	(38)
4.1	Java 类库结构.....	(38)
4.1.1	基础类库.....	(38)
4.1.2	应用类库.....	(39)
4.2	语言类库.....	(40)
4.2.1	Object 类.....	(40)
4.2.2	类型包容器.....	(41)
4.2.3	数学函数类.....	(43)
4.2.4	字符串类.....	(44)
4.2.5	System 与 Runtime 类.....	(47)
4.3	输入/输出类库.....	(48)
4.3.1	输入流类.....	(48)
4.3.2	输出流类.....	(54)
4.3.3	文件类.....	(58)
4.4	实用程序类库.....	(63)
4.4.1	接口 (Interfaces).....	(63)

4.4.2	Date 类.....	(64)
4.4.3	Random 类.....	(65)
4.4.4	Stack 类	(67)
4.4.5	Hashtable 类	(68)
4.5	小结.....	(70)
第五章	编写 Java 图形用户界面	(71)
5.1	概述.....	(71)
5.2	菜单设计.....	(72)
5.3	其它界面元素设计.....	(75)
5.4	两个例子.....	(82)
5.4.1	作图程序.....	(82)
5.4.2	文本编辑器.....	(89)
5.5	字体和颜色.....	(96)
5.6	界面元素包容器.....	(98)
5.7	小结.....	(103)
第六章	编写 Applet 应用.....	(104)
6.1	概述.....	(104)
6.2	Applet 在 HTML 文本中的嵌入方式.....	(105)
6.3	Applet 编程基础.....	(107)
6.3.1	Applet 的生命周期.....	(107)
6.3.2	图象处理.....	(108)
6.3.3	声音播放.....	(110)
6.3.4	Applet 参数的定义和使用.....	(111)
6.3.5	一个 Applet 样本程序: SlideShow	(112)
6.4	Applet 程序设计.....	(118)
6.4.1	用户界面设计.....	(119)
6.4.2	类设计.....	(119)
6.4.3	Applet 的完成.....	(124)
6.5	Applet 之间的通讯.....	(128)
6.6	Applet 的局限性.....	(133)
6.6.1	Applet 本身的功能限制.....	(134)
6.6.2	浏览器对 Applet 的限制	(134)
6.7	小结.....	(135)
第七章	多线程程序设计.....	(136)
7.1	概述.....	(136)
7.1.1	什么是线程?	(136)
7.1.2	Java 的线程类.....	(136)
7.1.3	线程的状态迁移.....	(137)
7.1.4	“精灵”线程 (Deamon Thread).....	(139)

7.2	两类编程方法	(139)
7.3	线程的优先级调度	(142)
7.4	线程组	(149)
7.5	多进程同步控制	(152)
7.6	小结	(157)
第八章	网络通讯和安全	(158)
8.1	URL 编程	(158)
8.1.1	什么是 URL ?	(158)
8.1.2	创建 URL 对象	(159)
8.1.3	URL 应用编程	(159)
8.2	套接字编程	(163)
8.2.1	面向连接的套接字编程	(164)
8.2.2	数据报套接字编程	(167)
8.3	协议和内容处理机	(172)
8.3.1	FTP 协议	(172)
8.3.2	NNTP 协议	(174)
8.3.3	WWW 协议	(174)
8.4	安全机制	(175)
8.4.1	SecurityManager 类	(175)
8.4.2	编程实例	(177)
8.5	小结	(179)
第九章	Java 综合运用举例	(181)
9.1	概述	(181)
9.2	程序结构	(182)
9.3	源程序	(182)
第十章	Java 原生函数的应用	(202)
10.1	概述	(202)
10.2	Java 原生函数的实现过程	(202)
10.2.1	编写 Java 类	(203)
10.2.2	生成 C 头文件和存根文件	(204)
10.2.3	编写 C 函数	(206)
10.2.4	生成 C 动态链接库	(208)
10.3	在数据库开发中的应用	(208)
10.4	小结	(218)
第十一章	Java 与虚拟现实	(219)
11.1	概述	(219)
11.2	虚拟现实世界的构造	(220)
11.2.1	构造虚拟现实世界的流程	(220)
11.2.2	构造一个简单的虚拟现实世界	(221)

11.3	Java 在虚拟现实中的应用	(224)
11.3.1	Java 与 VRML 的接口	(224)
11.3.2	应用举例	(228)
11.4	小结	(233)
附录 A	Java 类库层次关系图	(234)
附录 B	HTML 简介	(245)
附录 C	相关信息的 Internet 地址	(253)

第一章 Java 概述

1.1 Java 语言的起源

Java 语言最初是 SUN 公司为了电视机业的顶置盒 (Set top) 而设计的。当顶置盒行业不能起飞时, SUN 公司便把眼光转向了生机勃勃的国际互连网, 将 Java 语言加以改造并应用到国际互连网的开发编程中。随着国际互连网络的迅猛发展特别是网上应用和信息量的“爆炸”, Java 也开始名闻天下, 成为网络应用研究开发的热点之一。

Java 这个名称并不代表什么缩写符号。Java 开发组在研究了许多建议以后, 决定采用这个名字, 期望提起它的时候能让程序员感到一种轻松、愉快的感觉。Java 的特征图标是一杯热气腾腾的咖啡, 包含了同样的寓意。

它是一种完全的面向对象的语言, 风格近似于 C 语言。其编译的结果是称为 Byte Code 的一种具有高度可移植性的中间代码, 因而突破了不同的软硬件平台的限制, 在各类具有 Java 解释环境的设备上可直接运行。

由于 Java 语言逐步兴起, SUN 公司专门成立了 JavaSoft 分部, 负责 Java 技术和产品的开发销售和推广应用。其目标是使 Java 成为适用于国际互连网和企业内部网的全面的开放的软件平台, 并使 Java 作为一种应用开发语言广为接受。目前, 他们推出了如下平台上的开发工具 JDK (Java Development Kits): SPARC Solaris (2.3 or later)、Intel x86 Solaris、Windows NT/95 (Intel x86)、Macintosh 7.5 等。

除此以外, IBM 和其他的一些公司、开放软件基金会 (OSF——Open Software Foundation) 直接把 SUN 的 JDK 源代码移植到各自的平台上。所以, 从这些单位, 还可以找到如下平台的 Java 开发工具和运行环境: AIX、HP-UX、Digital UNIX、NCR SysV、Sony NEWS 等。

1.2 Java 和国际互连网

国际互连网最初由美国国防部在七十年代发起建立的, 最初称为 Arpanet。它以 TCP/IP (Transfer Control Protocol/Internet Protocol) 协议为标准, 实现了异构计算机平台的连接。经过二十多年的发展, 已有几百万的计算机连接到了网上, 并在网上已经积累了丰富的信息资源, 成为科研的最重要的资料库之一。WWW 平台的问世, 给终端用户提供了丰富的多媒体功能。国际互连网正走入寻常百姓家, 成为人们生活娱乐的好助手。

把国际互连网的规模缩小到企业的范围内, 加上必要的安全防护措施, 如防火墙等, 就成为了企业内部网 (Intranet)。它采用了国际互连网的成熟技术和产品, 是国际互连网的“微缩景园”。由于它的费用低廉, 利于提高办公效率, 加速信息的传播, 很快得到了推广应用。

Java 虽然不等同于国际互连网，但 Java 改造设计之初就是为了国际互连网的应用开发，并充分考虑了国际互连网的特点，这两者之间有密切的关系。Java 的高度可移植性就是因为考虑到了国际互连网的复杂的异构环境。Java 的独立应用程序可以直接在不同的平台上传送并执行。而 Java 的嵌入小程序也可以在多种国际互连网浏览器中解释执行。目前支持 Java 的浏览器有：SUN 公司的 HotJava，Netscape 通讯公司的 Navigator2.0 及以上版本，Microsoft 公司的 IE3.0 浏览器，Spyglass 的 Mosaic 浏览器，Oracle 公司的 PowerBrowser 浏览器等等。嵌入的小程序极大地增强了 WWW 的多媒体交互功能和数据的存取功能，使得 WWW 成为国际互连网上标准的图形化“操作系统”。

Java 正成为计算机领域的“世界语”，它为软件业未来的发展铺平了道路。“WinTel”的模式，即 Microsoft 的 Windows 系列操作系统平台加上 Intel 的 86 系列微处理器构成的对个人机的垄断，将受到严重的挑战。

1.3 Java 语言的结构

1.3.1 Java 的层次结构

Java 是一种解释性的语言，但不是直接对源代码进行解释，它还需要进行初步的编译。它的编译结果是字节码，不能在操作系统之上直接运行，两者之间有一个解释器层面。通过这一层，可以把 Java 的字节码翻成本机代码，然后再执行。图 1.1 为这种层次的示意图。这种层次结构保证了 Java 程序的可移植性。值得指出的是，Java 的解释器是依赖于不同的平台而变化的，用于 Windows 平台的 Java 解释器不能用于 UNIX 平台。但是由于它们有一个共同的 Java 编程接口，这就保证了程序的可移植性。

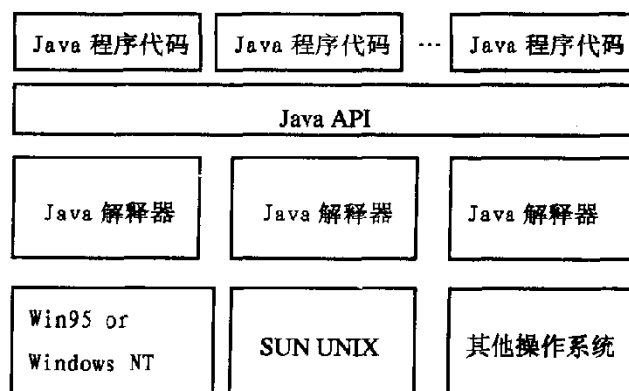


图 1.1 Java 平台的结构

1.3.2 Java 语言的开发流程

Java 语言的源代码以字符方式存放在*.java 格式文件中，源代码不能执行，必须经过 Java 编译器进行编译，其编译结果称为字节码（Byte Code），存放在*.class 的文件中。这些字节码可以在带有 Java 编译器的平台上运行。如果是为 WWW 平台设计的嵌入件，则可

以加入到 HTML 语言中，并在 WWW 浏览器中执行。图 1.2 为 Java 语言的编译流程示意图。

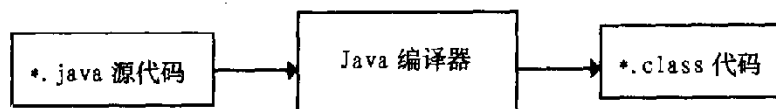


图 1.2 Java 语言的编译流程

目前，已有 Sun Solaris UNIX 和 Windows NT/95 平台可运行 Java 独立程序，Netscape2.0 以上、HotJava、MS Internet Explorer 等浏览器支持 Java 嵌入程序。Java 程序可以连接在普通的 HTML 文档中，如下面所示的例子，可在支持 Java 的浏览器（如，Netscape Navigator Gold 2.0，HotJava 等）中运行，也可以在单机的环境下运行。

```
<html>
<head>
  <title>A test for Java</title>
  <meta name="GENERATOR" content="Mozilla/2.01Gold (Win32)">
</head>
<body>
<hr><applet code=Test.class width=300 height=300></applet>
</body>
</html>
```

1.4 Java 语言的优点

Java 可以动态地下载到各种异构平台上并执行相关的操作，它改变了国际互连网和 WWW 的被动性。该语言以下的特点使它将成为国际互连网协议的最有前途的竞争者。

■ 可移植性

它可以在任何有 Java 解释器的设备上运行。这对于连接有各种各样的软硬件平台的国际互连网来说具有重要的意义。它的编译结果是称为 Byte Code 的一种中间代码。

Java 语言的下列两点特性使得 Java 语言具有很好的可移植性：

- 1) 该语言是解释型的，也就是说，任何想运行 Java 的计算机都必须有一个程序把 Java 代码转化为机器的本身代码。
- 2) Java 语言对所有的数据类型都有严格的规定，并且不会因为不同的机器而改变。如，整数类型规定严格地为 32 比特长。

■ 健壮性

该语言由运行环境系统来管理用户的内存，语言本身没有指针及指针的相关算法，这样，用户程序不会有指针悬空或内存溢出等现象发生。Java 的内存管理器自动地跟踪所有的对象，当对象结束时，系统将其所占的内存释放掉。Java 的内存管理器以独立线程的形

式，按较低的优先级运行于后台。而且，采用指针的程序结构虽然大大方便了本机编程的灵活性，但它不能分布到网络上，因为一台机器上的指针对另一台机器上就没有了意义。出于这样的考虑和分布的要求，Java 语言屏弃了指针功能。因此说，Java 的运行环境保证了代码的良好运行。

■ 安全性

除了要保护客户端免受无意的侵扰，Java 也能对付有意的侵扰。在它的运行系统中内置了防病毒和损坏文件系统的保护机制，在每条指令执行前都要进行相应的安全性检查。

■ 面向对象

Java 是彻底的面向对象的语言，它可以允许动态和静态的继承和复用。因此，它可以得到如下的优点：

- 1) 代码复用 (Reusability of Code)
- 2) 可扩展性 (Extensibility)

■ 高性能

Java 支持几个高性能的特点，如：

- 1) 内置的多线程功能
- 2) 高效的字节码 (Byte codes)
- 3) 及时编译
- 4) 内连 C 语言接口

■ 方便性

该语言可以被看作是 C 和 C++ 的延伸，对于具有一定编程经验的人员来说很熟悉，因此学习的负担较轻。同时，Java 语言的运行环境本身替程序员管理指针和内存，分担了编程人员的负担。再者，Java 语言本身提供了丰富的类库支持，程序员通过调用已有的类库构件，减小了编程的工作量；而且对于大型的软件项目，可以反复使用可复用的软件类，提高效率，缩短开发的周期。

1.5 本书的结构和安排

本书旨在为初步了解 Java 语言的编程人员提供一个深入学习和探索的机会。同时，为了本书的完整性，我们也安排了少部分初步的 Java 知识介绍。

从结构上看，我们基本上是按照由简单到复杂、由基本到高级的认知顺序排列的。

前面的第一章是概述，简单介绍 Java 语言的来历、层次结构和特点，及与国际互连网的关系。

接下来的第二章介绍了 Java 的 JDK 开发环境和其他的开发工具。在这一章结束的时候，列举了两个简单的例子，既可以让读者加强对 Java 应用的认识，又可以作为练习的例子熟悉开发流程。

第三章，讲解 Java 的语法要点。我们在介绍了 Java 的语法特点之后，对照 C++ 语言的语法进行了对比，从而帮助熟悉 C++ 的程序员避免混淆，尽快地转换到 Java。

第四章，介绍了 Java 的类库结构和各个部分的功能。由于 Java 是彻底的面向对象的语言，因此对其类库的了解显得尤其重要。本章的要求不在于读者在读完后能学会使用每

一个类，而在于对 Java 类库结构的了解，知道 Java 的类库中有什么样的支持，以便在真正需要时知道去那一个库中去找。

熟悉 Java 初步编程的读者可以跳过以上的四章，直接从第五章开始。

第五章，介绍编写独立应用程序的技巧。其中包括，菜单和界面设计等。该类应用与传统的程序设计很相似。

第六章，我们介绍 WWW 中的 Java 嵌入小程序。这是目前广泛应用的一种形式。通过举例，介绍了在 HTML 中嵌入的方式和相应的参数。然后，重点介绍界面的设计、多个嵌入小程序间的通讯、高级动画和声音播放的编程，最后介绍 Java 嵌入小程序在系统安全性等方面的局限性。

第七章，介绍多线程设计。多线程是 Java 语言的一个重要特征，也是难点之一。本章充分介绍了其特点和实际用法，并在多线程同步、例外出错处理等方面进行了深入探讨。

第八章，介绍 Java 的网络编程。包括网络套接字的编程、协议与内容处理机的使用、安全防护手段等。本章体现了 Java 在网络方面的优势，是本书的重点之一。

第九章，在前几章的基础之上，介绍了一个网络交互软件，综合应用了图形化用户界面、套接字编程和多线程等技术。

第十章，介绍了 Java 与 C 语言的接口问题，及综合应用 Java 和 C 函数开发数据库应用。

第十一章，介绍了 Java 在国际互连网多媒体交互程序设计方面的应用。对有兴趣开发网络游戏和虚拟现实应用的读者有很大参考作用。

1.6 小 结

在本章中我们介绍了如下的内容：

■ Java 是一种彻底的面向对象的编程语言，它与国际互连网的应用开发密不可分，具有高度的可移植性，是国际互连网应用开发的首选工具。

■ Java 是一种解释性的语言，通过 Java 解释器层面，把 Java 的字节码翻译成本机代码，然后再执行。

■ Java 摒弃了指针的概念，由 Java 的运行环境来管理内存，既适应了程序分布的需要，又保证了程序的安全，减少了程序员的工作量。

在下一章中，我们将介绍 Java 的开发环境。

第二章 Java 开发环境及简单示例

学习 Java 编程的最好方法就是从简单的 Java 程序入手,而建立 Java 开发环境则是首先需要做的。SUN 公司为我们提供了一套 Java 开发工具 (JDK), 包括 Java 编译器、解释器和调试器等, 使用它可以开发各种 Java 应用。

本章将首先介绍 Java 开发环境的硬件要求, 软件安装及 JDK 工具的使用。在建立 Java 开发环境后, 我们将用一个简单的例子来说明 Java 编程的基本知识, 以此作为学习 Java 编程的开始。在此基础上, 读者可以进一步学习各种复杂的 Java 编程技术。最后本章将介绍几种支持 Java 的 WWW 浏览器。

2.1 硬件要求

Java 编程的硬件要求取决于你使用的操作系统平台, 目前主要有三种系统支持 SUN 公司的 Java 开发工具: 运行于 SPARC 处理器的 Solaris 2.3 或以上系统, 及运行于 Intel x86 系列的 Windows NT 和 Windows 95。

表 2.1 Java 编程的硬件要求

SUN 工作站	Windows NT 工作站	Windows 95
SPARC 处理器	Intel x86 处理器 (486 以上)	Intel x86 处理器 (486 以上)
Sun Solaris 2.3 或以上	Windows NT3.5 或以上	Windows 95
32 MB 内存	16 MB 内存	8 MB 内存
9 MB 以上硬盘空间	6 MB 以上硬盘空间	6 MB 以上硬盘空间
声卡设备	声卡设备	声卡设备

除了上述三种平台, SUN 公司已在为 MacOS 7.5 研制 Java 开发工具。此外其它操作系统, 如 OS/2、Linux、Amiga 和 NextStep 也将提供相应的 Java 开发工具。本书我们主要探讨 Solaris 和 Windows NT/Windows 95 平台上的 Java 编程。

2.2 软件安装

当你的计算机能满足运行 Java 的硬件要求时, 你就可以着手进行软件的安装了。Java 开发工具软件可以从 Internet 上获取, 它的当前版本是 JDK 1.0.2。

2.2.1 基于 Intel x86 —— Windows NT/Windows 95

运行于 Windows NT/95 上的 JDK 是一个近 4.0 MB 的自解压缩文件。当你完成安装后, JDK 的文件和目录将占用大约 5.5 MB 的硬盘空间。本书附录列出了 Internet 上提供 JDK

的一些节点，你可以通过 FTP 或 WWW 浏览器下载文件，我们以从节点 ftp.javasoft.com (198.70.96.253) 下载文件为例来说明：

```
c:\ > ftp ftp.javasoft.com
Name (ftp.javasoft.com): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password: username@domain-name
...
ftp> binary
200 Type set to I.
ftp> cd pub
...
CWD command successful.
ftp> get JDK-1_0_2 - win32 - x86.exe
200 Port command successful.
150 Opening BINARY mode data connection for JDK-1_0_2-win32- x86.exe(3841189bytes)
226 Transfer complete.
local: Jdk-1_0_2 - win32 - x86.exe remote: JDK-1_0_2 - win32 - x86.exe
3841189 bytes received in 1.4e+03 seconds (3.0 Kbytes/s)
```

当然，如果采用 WWW 浏览器下载文件将更加方便。SUN 公司在其 Web 节点上提供了 JDK 安装软件，你可以使用如下格式的 URL：

```
ftp://ftp.javasoft.com/pub/JDK-1_0_2 - win32 - x86.exe
```

在 JDK 软件下载完毕后，需要运行该自解压缩文件来进行软件安装。一般是在 C 盘根目录下执行如下命令：

```
c:\ > JDK-1_0_2 - win32 - x86
```

它将生成目录 C:\ java 及其相应的子目录。如果想将 JDK 安装在其它目录，只需在该目录下运行下载的自解压缩文件。系统安装后会产生两个文件：src.zip 和 lib/classes.zip，如果你想查看 JDK 类库的原码，可以解压缩文件 src.zip，但一定不要解压缩文件 classes.zip，因为它将作为 Java 源程序编译时的输入类库。

JDK 安装后，还要设置环境变量，你需要在文件 AUTOEXEC.BAT 中加入以下两行：

```
SET PATH=c:\ java\ bin
```

```
SET CLASSPATH=c:\ java\ lib\ classes.zip; c:\ java\ MyProgram
```

其中 c:\ java\ MyProgram 是你编写 Java 程序的工作目录。

2.2.2 基于 SunSparc/Solaris 2.x

运行于 Solaris 系统的 JDK 是一个不超过 5 MB 的压缩文件，在解压缩并安装后将占用大约 9 MB 的硬盘空间。Solaris 系统上的软件安装过程和 Windows NT/Windows 95 基本上类似，你可以通过 FTP 或 WWW 浏览器，从节点 `ftp.javasoft.com` 上将文件 `JDK-1.0.2-solaris2-sparc.tar.z` 下载到本地。然后执行如下命令：

```
$ zcat JDK-1.0.2-solaris2-sparc.tar.z | tar xf -
```

该命令将生成目录 `java/` 及包含所有可执行程序 and 文档的子目录。在安装完毕后，不要忘记删除 JDK 压缩文件。

2.3 JDK 工具的使用

2.3.1 Java 编译器

Java 编译器是用来将 Java 源代码文件编译成可执行的 Java 字节码 (bytecode) 的。Java 源代码文件的扩展名为 `.java`，Java 编译器把这种扩展名的文件编译成扩展名为 `.class` 的文件。源文件中的每个类在编译后都将产生一个 `class` 文件，这就意味着很多情况下，一个 Java 源代码文件可能编译生成多个 `class` 文件。

JDK 提供的 Java 编译器是一个命令行程序，其使用语法如下：

```
javac Options Filename
```

参数 `Filename` 指定你想要编译的源代码文件名；参数 `Options` 指定编译器如何生成 Java 可执行类的选项，主要的编译选项如下：

- `-classpath Path`
- `-d Dir`
- `-g`
- `-nowarn`
- `-verbose`
- `-o`

选项 `-classpath` 告诉编译器用 `Path` 指定的路径替代 `CLASSPATH` 环境变量，这将使得编译器在 `Path` 指定的路径中寻找用户定义的类。选项 `-d` 确定了编译后的类存放的根目录，使用该选项，类目录结构将在 `Dir` 指定的目录下生成，下面用一个例子来说明：

```
javac -d ../Flower
```

在该例子中，输出文件 `Flower.class` 将存放于当前目录的上一级目录中。

选项 `-g` 使编译器为 Java 类生成调试表，调试表是给 Java 调试器使用的，它包括诸如局部变量和行号等信息，缺省时编译器只生成行号。

选项 `-nowarn` 用来关闭编译警告，编译时出现的警告表示源代码中潜在的问题，使用 `-nowarn` 选项将不输出警告信息。选项 `-verbose` 的作用正好与 `-nowarn` 相反，它将打印出编译过程的详细信息。

选项 `-o` 使编译器对编译后的代码进行优化, 在这种情况下, 优化只是意味着在线编译 `static`、`final` 和 `private` 方法 (`method`)。这将加快运行速度, 因为它减少了方法的过多调用。优化类通常较大, 以容纳复用的代码。

2.3.2 Java 解释器

Java 解释器对编译生成的字节码格式的可执行程序的运行提供支持, 它是运行非图形 Java 程序的命令行工具, 而图形程序的运行需要浏览器的支持。使用 Java 解释器的语法如下:

```
java Options Classname Arguments
```

参数 `Classname` 指定你想运行的类名, 当解释器运行一个类时, 它实际上是在执行该类的 `main` 函数, `main` 函数及它产生的所有线程运行完毕时解释器即退出。 `main` 函数可接收一些参数来控制程序的运行, 参数 `Arguments` 即是用来指定传递给 `main` 函数的参数。例如, 如果你有一个 Java 类 `TextFilter`, 它对一个文本文件执行一些过滤处理, 你就可以把文件名作为参数, 如下:

```
java TextFilter SomeFile.txt
```

参数 `Options` 指定与解释器运行 Java 程序相关的选项, 以下是解释器一些最重要的参数:

- `- debug`
- `- checksource, - cs`
- `- classpath Path`
- `- verbose, - v`
- `- verbosegc`
- `- verify`
- `- verifyremote`
- `- noverify`
- `- D PropertyName=NewValue`

选项 `- debug` 以调试方式启动解释器, 使你能把 Java 解释器和调试器结合起来使用。选项 `- checksource` 将使解释器比较源文件和类文件的修改日期, 如果源文件最近修改过, 将自动重新编译。

Java 解释器使用环境变量 `CLASSPATH` 来确定用户定义的类所在的目录, `CLASSPATH` 包含一些以分号隔开的用户定义的 Java 类的系统路径。通常, Java 工具使用 `CLASSPATH` 来找到用户定义的类, 而选项 `- classpath` 则是用来通知解释器以 `Path` 指定的路径替代环境变量 `CLASSPATH`。

选项 `- verbose` 控制解释器在每次装载 Java 类时显示一条信息, 类似地, 选项 `- verbosegc` 控制解释器在每次收集废物, 如清除不需要的对象或释放内存时显示一条信息。

选项 `- verify` 使解释器在所有代码装入运行环境时进行字节码校验, 它的缺省功能只是只对使用类装载器启动的代码进行校验。缺省功能也可以用选项 `- verifyremote` 明确指定。选项 `- noverify` 则将关闭所有代码校验功能。

选项 -D 使你能重新定义属性值, `PropertyName` 指定了你想改变的属性名, `NewValue` 指定了你想设定的属性值。

2.3.3 Appletviewer

Appletviewer 是一个 Java Applet 的简单测试工具, 你可以使用它来测试 Java Applet 程序, 而不需要 WWW 浏览器的支持。使用 Appletviewer 的命令如下:

```
appletviewer Options URL
```

参数 URL 用来指定一个嵌入了 Java Applet 的 HTML 页的 URL 文本。参数 Options 指定如何运行 Java Applet。Appletviewer 只支持一个选项: `-debug`, 它能够通过 Appletviewer 启动 Java 调试器, 这就使得你能调试 Applet 程序。图 2.1 即是 Appletviewer 运行的一个范例。

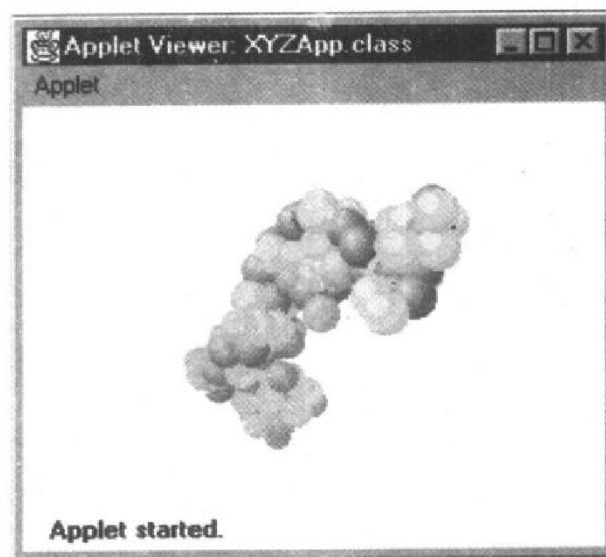


图 2.1 由 Java AppletViewer 运行的 MoleculeViewer Applet

图 2.1 是在 Appletviewer 中运行的一个 JDK 的示例, 你需要指定包含 MoleculeViewer 的 HTML 文件的目录, 并在命令行下执行如下命令:

```
appletviewer example1.html
```

`example1.html` 是包含了嵌入的 Java Applet 的 HTML 文件。你可以看到, 使用 Appletviewer 运行 Java Applet 没有什么复杂的, 它是在简单环境下测试 Java Applet 的一个有用工具。

2.4 两类简单的 Java 程序

现在你已建立了 Java 开发环境, 下一步要做的就是开始 Java 编程。我们要实现的第一个 Java 程序就是著名的 “Hello World!” 程序, 它只是在屏幕上显示 “Hello, World!”。

虽然这个程序比较简单，但它却能较好地说明 Java 编程语言的风格特点。

Java 程序分为两类：Applet 和 Stand - alone 程序。简单地说，Applet 是在编译后通过 Applet View 或支持 Java 的 WWW 浏览器来运行的 Java 程序；而 Stand - alone 程序是在操作系统环境下可直接运行的 Java 程序，它与传统的 C 或 C++ 程序非常类似。下面将分别说明“Hello World”程序作为 Applet 和 Stand - alone 程序的实现方法。

2.4.1 Applet 应用

对于多数读者来说，学习 Java 编程的目的就是要在 Internet 上为支持 Java 的浏览器提供 Applet 应用，因此，这里我们首先来介绍“Hello World”程序的第一种实现方法——Applet。适合浏览器环境的 Applet 应用需要用到一些有关窗口界面的类库，尤其是要用到 AWT (Abstract Window Toolkit) 类来控制 Applet 在屏幕上的外观。通常，为了将字符串“Hello World”显示在窗口中，需要重写 java.awt.Graphics 类中的成员函数 paint()。

所有的 Applet 都是基类 java.applet.Applet 的继承类，如果你想改变窗口大小，可以重写该类的成员函数 init()。Applet 应用是依靠基类来接收窗口调用，处理各种事务，如当窗口改变大小时重绘窗口等。

■ 设置工作环境

编写“Hello World”程序的第一步是建立该程序存放的目录。2.2.1 中已说过，为了让 Java 编译器和 AppletViewer 能工作，需要在 AUTOEXEC.BAT 中定义环境变量 PATH，包含目录 java\bin 和你存放工作文件的目录。这里我们生成一个目录 java\MyProgram 来存放 Java 文件。

```
c:\> md java\MyProgram
```

本例中的目录结构只是为本机使用的，如果需要将你的 Applet 应用提供给 Internet，你最好请系统管理员，把相关文件放到通过 Internet 能访问的目录结构中。

■ Hello World 程序源代码

建立工作目录后，就可以开始 Hello World 程序的编码了，首先，需要在该目录中生成一个名为 HelloWorld.java 的文件，扩展名 .java 是用来表示 Java 源代码文件的标准方法。你可以使用任意的一个文本编辑器来编辑源代码文件，在 Windows NT/95 中，我们可以使用目前较流行的工具 JavaMaker，它提供了 Java 程序的编辑、编译及 Applet 的运行环境。关于 JavaMaker 的详细信息，可以浏览 JavaMaker 的 Web 节点：

<http://net.info.samsung.co.kr/~hcchoi/javamaker.html>

Hello World 程序的源代码如下：

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

public class HelloWorld extends Applet {
    public void init() {
```



```

    resize(200, 100);
}
public void paint(Graphics g) {
    g.drawString("Hello, world!", 50, 50);
}
}

```

下面我们来逐行研究程序源代码。

```

import java.awt.Graphics;
import java.applet.Applet;

```

上述两行告诉编译器你将使用 Graphics 类和 Applet 类，编译器在你的源代码中将包括这两个类的定义。

```

public class HelloWorld extends Applet {
    该行定义了一个新的类 HelloWorld，并告诉编译器它是 Applet 类的子类。
    public void init() {

```

该行重写了 Applet 类的 init() 函数，public 表示其它的对象可以调用它，void 表示该函数无返回值，空括弧表明该函数不需要参数。当生成 Applet 时，父类 Applet 中的成员函数将调用它，并执行下一行：

```

        resize(200, 100);
        该行告诉浏览器将窗口大小设置为 200•100。
        public void paint(Graphics g) {

```

该行重写了 Applet 类的 paint() 函数，当 Applet 在窗口中显示时调用该函数。它也是一个公用函数，并且无返回值，但它有一个参数，当调用 paint() 时，需要一个 Graphics 类对象作为参数。

```

        g.drawString("Hello, World!", 50, 50);

```

在这行中，Graphics 对象使用参数 Hello, World!、50 和 50 调用它自己的成员函数 drawString()，结果在 (50, 50) 的位置打印字符串 Hello, World!

当 Applet 运行完毕后，Java 环境中的“垃圾” (garbage) 函数将自动地释放分配的内存，确保程序干净的退出。

■ 编译

生成源代码文件后，就可以进行编译了。在 Solaris 和 Windows NT 中，都可以使用如下命令编译：

```

c:\ > javac HelloWorld.java

```

如果编译成功，将在与文件 HelloWorld.java 相同的目录生成一个名为 HelloWorld.class 的文件。如果编译有问题，可以检查一下输入的源代码是否正确。如果编译还是通不过，请参考本节最后一部分的错误诊断。

如果你使用 JavaMaker 来编译该程序，将会弹出一个编译输出窗口，并显示编译状态信息。如图 2.2 所示。

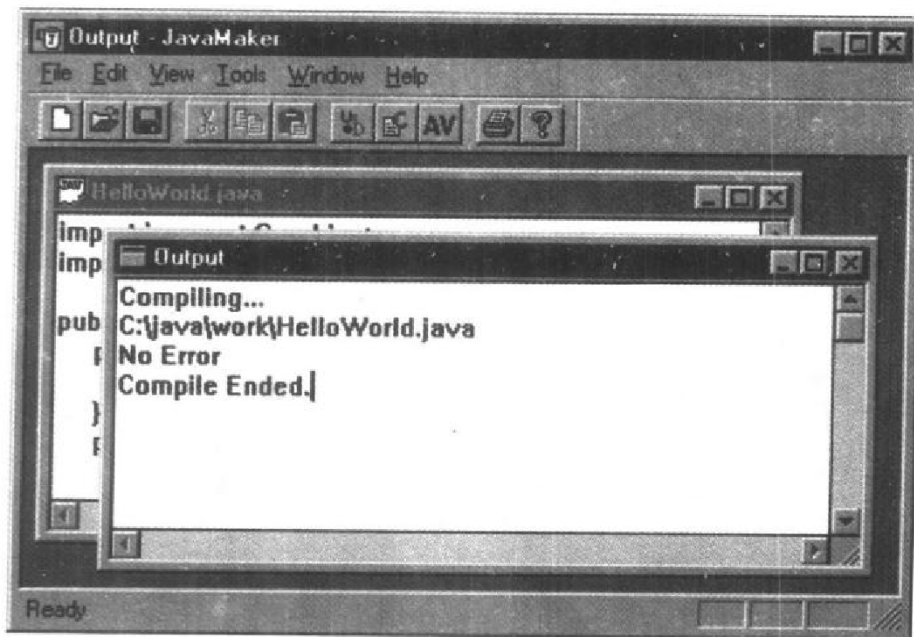


图 2.2 JavaMaker 编译窗口

■ 生成 HTML 文件

完成编译并生成 HelloWorld.class 文件后，还需要把它放到 HTML 文件中。在同一目录下编辑一个文本文件 HelloWorld.html，这是包含<applet>指令的标准 HTML 文件格式。关于 HTML 语言，请参阅附录 B；关于 Applet 的嵌入方式和参数，请参阅 5.1。这里我们仅简单地将 HelloWorld.html 的内容列出如下：

```
<HTML>
<HEAD>
<TITLE>Hello, World!</TITLE>
</HEAD>
<BODY>
Say Hi to Everyone:
<applet align=center code="HelloWorld.class" width=300 height=200></applet>
</BODY>
</HTML>
```

■ 载入 Applet

完成上述工作后，就可以使用 AppletViewer 来观看 Applet，调用 AppletViewer 的命令如下：

```
c:\>appletviewer HelloWorld.html
```

你也可以在 JavaMaker 环境中，选择菜单项 Tools 下的命令 Appletviewer 来运行该 Applet，但你首先需要在选项 Environment 中设置 JDK 文件的路径和嵌入 Applet 的 HTML

文件名。无论你是在命令行还是在 JavaMaker 环境中运行 AppletViewer，都应该看到如图 2.3 所示的窗口。

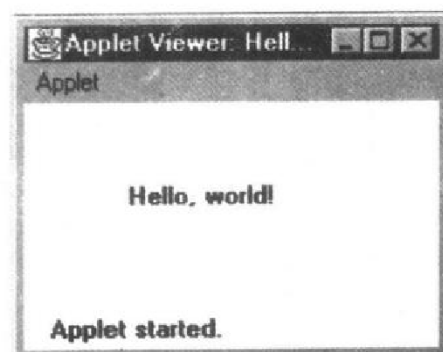


图 2.3 由 AppletViewer 运行的 HelloWorld Applet

到目前为止，你已通过了生成你自己的第一个 Internet 上的 Java Applet 的所有步骤。显然，要编写复杂的 Applet 应用，还需要学习更多的相关知识，但这个过程是基本一样的，我们归纳如下：

- (1) 编写 Java 程序代码，然后编译并检查编译错误；
- (2) 编写包含该 Applet 的 HTML 文件；
- (3) 使用 AppletViewer 等工具测试该 Applet。

2.4.2 Stand - alone 应用

Hello World 程序也可以作为 Stand - alone 应用来运行，当然，它不是真正意义上的独立应用，因为它需要 Java 解释器来运行。在后面的程序源代码中，你将看到它没有使用 Applet 中的那些继承类，而是生成了一个新类 HelloWorldApp。实际上，它还是有类继承的，当一个类在定义时，如果没有指定所继承的类，就将继承类 Object。除了类 Object，Java 中的每个类都有超类 (superclass)。

■ 程序源代码

HelloWorldApp 的源代码比较简单，只有如下 5 行：

```
class HelloWorldApp {  
    public static void main (String args[]) {  
        System.out.println("Hello, World!");  
    }  
}
```

我们还是来逐行分析源代码。

```
class HelloWorldApp {  
    该行定义了类 HelloWorldApp，但并没有指定所继承的类，因此就将继承类 Object。
```

其实该行也可以写作:

```
class HelloWorldApp extends Object {
```

显然, 第一种写法更简洁!

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

该行有几项 Applet 程序中未出现的新特点。首先, 它使用了修饰语 `static`, 这个关键词告诉编译器该函数将引用类自身, 而不是引用该类生成的一个特定实例。这就使得不用先生成 `HelloWorldApp` 类的对象, 就可以调用该函数。其次, 该函数使用了一个参数 `args[]`, 它是类 `String` 的一个对象。实际上, 这个对象包含了运行命令后面敲入的参数。最后, 是函数本身的名字——`main()`, 与 C 和 C++ 语言类似, 它告诉解释器程序运行的起点。解释器调用 `main()` 函数, 并将命令行参数传递给它。

```
System.out.println("Hello, World!");
```

该行在屏幕上输出字符串 `Hello World!`

■ 编译

现在你已完成源程序文件 `HelloWorldApp.java`, 并将它存到目录 `MyProgram` 中。下一步就是要使用 `javac` 编译器来编译。

```
c:\> javac HelloWorldApp.java
```

编译后将在目录 `MyProgram` 中生成文件 `HelloWorldApp.class`。此外, 你也可以使用 `JavaMaker` 来编译该文件。如果有问题, 可以参考本节最后一部分的内容。

■ 运行

在完成上述工作后, 你就可以使用 Java 解释器来运行该程序, `JavaMaker` 不提供调用 Java 解释器的手段, 所以你能在命令行下来运行。运行 `HelloWorldApp` 的命令如下:

```
c:\> java HelloWorldApp
```

解释器实际上引用的是 `.class` 文件, 而不是源文件。如果运行正常, 你将看到如图 2.4 所示的输出。

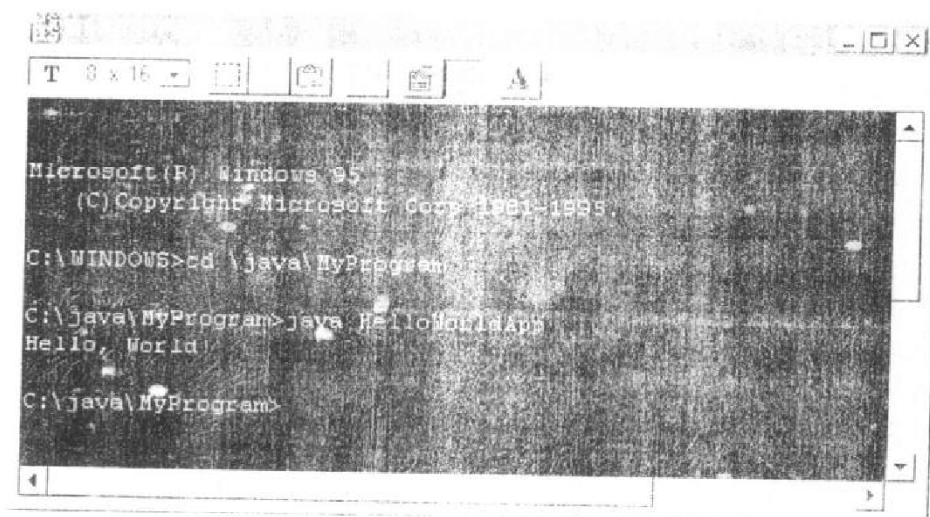


图 2.4 由 Java 解释器运行 HelloWorldApp 的输出

这样，你就完成了第一个 Stand - alone 的 Java 应用。与 Applet 应用相比较，开发 Stand - alone 应用的过程要简单一些，你不必使用<HEAD>、</TITLE>、<APPLET>等说明符来编写 HTML 文件，也不必使用浏览器来运行 Stand - alone 应用。Java 程序具有在许多不同系统上运行的能力，虽然程序 HelloWorldApp 并不如何复杂，但说明了 Java 是真正可靠的、高性能的编程语言。

2.4.3 错误诊断

当你使用 JDK 工具编译和运行 Java 程序时，可能会遇到一些错误信息。我们这里针对某些错误类型，提出了相应的处理方法。虽然不很完备，但对初学者还是有一定用处的。

■ 编译错误

下面列出的编译错误可能是在程序编译时最常出现的问题。每当编译器输出错误信息时，它就不会产生 .class 文件，所以如果你在编译程序后没有得到相应的 .class 文件，编译器很有可能在程序代码中遇到了错误。

→ javac: Command not found

如果环境变量 PATH 设置不正确，就会出现该错误。你需要在文件 AUTOEXEC. BAT 的 PATH 语句中包含 java/bin 目录。

→ filename: lineNumber: ';' expected
some line of code
^

n error(s)

当你在某个语句或说明的末尾漏了分号(;)时，就会遇到该错误信息。出现该错误表明编译器能理解你的程序。很多情况下，编译器会不知所云，并在后面的代码行中输出一些奇怪的错误信息，如：Invalid type expression 或 Invalid declaration。如果编译器发现错误的那行程序没有什么问题，就应该检查该行前面的程序行是否少了一个分号。

→ filename: lineNumber: Variable variable-name may not have been initialized.
Variable
^

n error(s)

每当程序中的变量没有定义时，编译器就输出一个错误信息而不生成 .class 文件，因此要确保所有变量在使用时都要预先定义。

■ 解释运行错误

许多情况下，程序编译时没有发现什么错误，但在解释运行时却出现了问题。下面是经常遇到的两种错误类型：

→ Can't find class classname.class

当用户解释运行类文件名而不是类名时将出现该错误，通过解释器运行 Java 程序时，一定不要加上文件扩展名.class。例如运行 HelloWorldApp，你应该输入 java HelloWorldApp，而不是 java HelloWorldApp.class。

→ In class classname: void main(String argv[]) is not defined

当一个类没有定义 `main()` 方法时将出现该错误，为了让解释器知道运行开始处，所有的应用都应该定义 `main()` 函数。

2.5 Java 程序调试

Java 开发环境为程序调试提供了一个调试工具 JDB，JDB 是一个调试 Java 程序的命令行工具，它通过 Java Debugger API 对 Java 解释器提供调试支持。为了使用 Java 调试器，在编译程序时一定要使用 `-g` 选项。下面我们以 HelloWorld 程序的调试来进行说明，首先按如下命令方式编译：

```
c:\> javac -g HelloWorld.java
```

然后以调试方式执行 AppletViewer 命令：

```
c:\> appletviewer - debug HelloWorld.html
```

```
Initializing jdb ...
```

```
0x13a4f38: class (sun.applet.AppletViewer)
```

```
>
```

你可以通过 `help` 命令来查询 JDB 中的调试命令。

```
> help
```

```
** command list **
```

```
threads [threadgroup]  ——列出所有线程（组）
```

```
thread <thread id>  ——设定默认线程
```

```
suspend [thread id(s)]  ——暂停线程运行
```

```
resume [thread id(s)]  ——恢复线程运行
```

```
where [thread id] | all  ——列出线程运行栈
```

```
threadgroups  ——列出线程组
```

```
threadgroup <name>  ——设定当前线程组
```

```
print <id> [id(s)]  ——打印对象或对象某个域
```

```
dump <id> [id(s)]  ——打印所有对象信息
```

```
locals  ——打印在当前栈中的所有局部变量
```

```
classes  ——列出目前使用的类
```

```
methods <class id> ——列出某个类的方法
```

```
stop in <class id>.<method> ——在某个方法中设置断点
```

```
stop at <class id>:<line> ——在某行中设置断点
```

```
up [n frames] ——向上移动到某个线程的栈
```

```
down [n frames] 向下移动到某个线程的栈
```

```
clear <class id>:<line>清除某个断点
```

```
step  执行到当前行
```

cont 从断点继续运行

catch <class id> 出现某个例外时中断运行

ignore <class id>忽略某个例外

list [line number|method] 打印源代码

use [source file path] 显示或改变源文件路径

memory 报告使用的内存

gc 释放未使用的对象

load classname 载入要调试的 Java 类

run <args>开始运行载入的类

!! 重复执行上一命令

help (or ?) 列出所有调试命令

exit (or quit) 退出调试器

>

输入命令 threadgroups 可以显示当前运行的线程组，线程组 “system” 和 “main” 正在运行。

> threadgroups

1. (java.lang.ThreadGroup) 0x13930b8 system

2. (java.lang.ThreadGroup) 0x13a37b0 main

使用 stop 命令在程序中设置一个断点。

> stop in HelloWorld.paint

Breakpoint set in HelloWorld.paint

输入命令 run 将启动 AppletViewer。

> run

run sun.applet.AppletViewer HelloWorld.html

runnig ...

main[1]

Breakpoint hit: HelloWorld.paint (HelloWorld:9)

AWT-callback - Win32[1]

list 命令将显示当前断点处的源代码。

AWT-callback -Win32[1] list

6}


```

7public void paint(Graphics g) {
8=>g.drawString("Hello, world!", 50, 50);
9}
10}
AWT-callback - Win32[1]

```

使用 step 命令将单步运行当前行，可以很方便地调试程序。虽然 JDB 可能不如你熟悉的某些开发环境的可视化调试工具，但它毕竟使你调试 Java 程序容易了许多。

2.6 几种支持 Java 的 WWW 浏览器

影响 Java 程序开发的一类重要 Java 应用是 WWW 浏览器，没有支持 Java 的浏览器，Java Applet 就没有什么用处，Java 浏览器实际上充当了 Java 程序的操作系统。由于这个原因，Java 很大程度上取决于 Java 浏览器的成功，值得庆幸的是，WWW 浏览器的主要厂商，如 Netscape、Sun、Microsoft 等都已声称将支持 Java。下面简单介绍几种支持 Java 的浏览器。

2.6.1 Netscape Navigator

作为 WWW 浏览器的最大供应商，Netscape 比较早将 Java 技术集成到了浏览器中。随着 Navigator 2.0 的出现，Netscape 提供了对 Java 语言的全面支持。

在 Netscape Navigator 中，支持 Java Applet 的运行已成为一项缺省功能，当你安装好 Navigator 后，即可观看 Applet 并与具有 Java 功能的 Web 节点交互信息。你可以通过访问节点 <http://java.sun.com/> 来测试，在显示页面（Homepage）的右上角将有一个热汽腾腾的转动的咖啡杯，如图 2.5 所示。实际上，你可以通过选项来设定 Navigator 是否支持 Java。



图 2.5 Java Applet 在 Netscape Navigator 中的运行

如果你在浏览某个包含 Java 的 Web 节点时遇到了问题,可以将 Navigator 设置为不支持 Java。在 Navigator 菜单选项 Options 下,选择 Security Preferences 命令,将显示一个对话框,由它可设置是否支持 Java。

Netscape 不仅提供支持 Java 语言的浏览器,而且帮助开发了一种起源于 Java 的基于对象的语言—JavaScript, JavaScript 的目的是要快速开发分布式的 Client - Server 应用。限于篇幅,本书不详细讨论 JavaScript。

2.6.2 HotJava

HotJava 是 SUN 公司在 WWW 浏览器市场上的竞争产品,它最初是作为 Java 浏览器开发的一项试验,但却成了 WWW 浏览器未来发展的一种模型。目前还不清楚 HotJava 是否会成为浏览器市场的有力竞争者,但显然它对 Java 应用开发者是有吸引力的。HotJava 本身是完全用 Java 实现的,无疑是最适合 Java 的浏览器,不考虑 HotJava 是否能在专业 WWW 浏览器市场上占有一席之地,它还是可以作为 Java 编程者的一个非常有用的测试工具。

HotJava 区别于其它 WWW 浏览器的主要特点是它的扩展性,不需要修改 HotJava 应用本身,即可与新的对象类型和 Internet 协议动态交互和处理,而且它能自动扩展提供新的支持。

安装配置 HotJava 非常简单,在 Internet 节点 <http://www.javasoft.com/> 上有 HotJava 浏览器的最新版本,它是一个自解压缩文件,下载该文件后直接运行即可安装。图 2.6 为 HotJava 浏览器。

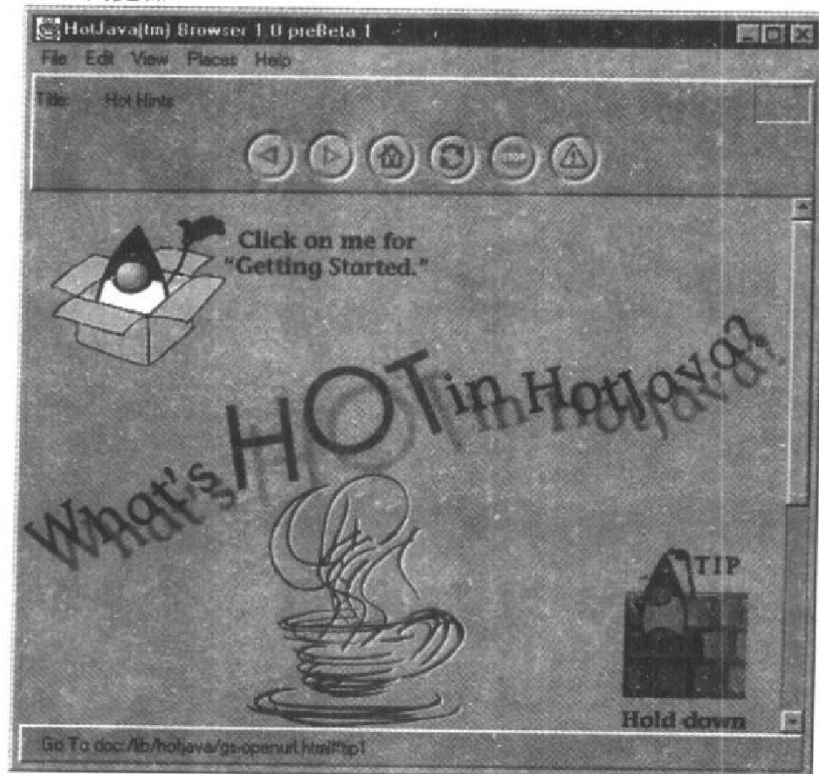


图 2.6 HotJava 浏览器

2.6.3 Microsoft Internet Explorer

当 Java 引起广泛注意后, 微软公司当然不会袖手旁观, 它已购买了 Java 技术的授权, 并在最新推出的 Internet Explorer 3.0 中提供了对 Java 的支持。考虑到 Internet Explorer 与 Windows 95 联系紧密, 人们有理由认为 Internet Explorer 将在 WWW 浏览器市场上占有较大份额。

2.7 小 结

本章主要讲述了 Windows 95/NT、Sun Sparc/Solaris 2.x 等几种操作系统平台 JDK 的安装与使用, 并在此基础上, 阐述了简单 Java 程序的编程及调试, 最后介绍了几种支持 Java 的 WWW 浏览器。通过本章的学习, 你应该能熟悉 Java 开发环境各种工具, 如 Java 编译器、解释器和调试器等; 对两种类型的 Java 应用 — Applet 和 Stand - alone 应用的编程有一个初步的了解。你需要多花些时间研究 Hello World 程序, 试着改变程序中某些行, 检查编译器或解释器会输出什么错误信息, 看一看是否有错误诊断部分列出的错误。学习编程语言的诀窍就是多做试验, 发现错误并解决错误。

第三章 Java 的语法要点

语法是一个编程语言的基本要素。理解语法规则是编程的第一步。在本章中，我们将学习面向对象的概念，分析类、对象的区别，理解消息机制、封装等概念，还将介绍 Java 中的类定义语法和类的使用方法。最后，详细地介绍 Java 程序中的语法细节，包括变量类型和定义，数组、字符串，操作符和表达式，和流控制语句等。

3.1 面向对象的概念初步

随着信息业的发展，软件工程遇到了许多的问题，出现了所谓的“软件危机”，其表现为：软件系统越来越复杂，经常处于变化之中，可靠性难以保证，测试的难度加大，等等。为了对付这种软件的危机，面向对象概念应运而生，并迅速在软件业流行起来。面向对象的设计和编程可以帮助编程人员编写更可靠、安全和更易于维护的代码。

3.1.1 什么是对象（Object）和类（Class）？

对象是包含了数据（Data）[或称为属性（Property）]和操作（Method）的软件模块结合体。如图 3.1 所示，属性是对象的核心所在，它记录了对对象的特征和状态。外界一般通过操作函数来存取内部的数据。当然，对于数据变量中的公共部分则可以在外部直接存取，但面向对象的方法本身不鼓励这样的方式，因为这样可能会威胁到数据的安全。这样，通过定义对象，把软件的操作和数据封装在一起。

对象是对客观的抽象映射，不仅包括实际存在的东西，如，房子、汽车等，还可以是虚拟的东西，如软件中的进程、通讯中的套接字等。不是所有的东西都是对象，如房子的颜色就是房子的属性，它代表了房子这个对象的属性。但这也不是绝对的，在不同的背景之下可能有不同的划分。如我们对颜色有深入的探讨时，可以把颜色当成一个对象，把构成颜色的三个基本色的大小当成属性。如下所示：

```
class Color {
    int R; // 红色属性
    int G; // 黑色属性
    int B; // 兰色属性
public:
    GetR(); // 读取红色属性
    SetR(); // 设置红色属性
    TurnRed(int increment); // 增加或减少红色比例
}
```

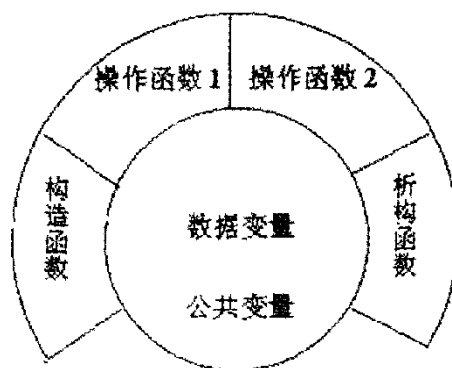


图 3.1 对象的结构示意

把相关的变量和操作封装到一个“对象”软件块中的好处是：

- 模块化：一个对象的源代码可以独立地编写和维护，而且，一个对象可以方便地为其他项目复用。
- 信息隐蔽：一个对象有一个公用的接口，其他的对象可以通过该接口与它进行通讯，而不必关心其具体实现的细节。只要该接口不变，该对象又可以维护私有的信息和操作，而不影响其他相关的对象。

类是一类对象的抽象和总结，是实际对象的“模板”。对象在生成的时候，要依据类的定义来确定数据和操作，对象是类的一个实例。从一个类可以“实例化”多个对象，而这多个对象可以有不同的属性。如，从颜色类中实例化出颜色 1 和颜色 2，颜色 1 为红色，它的 RGB 值为 255, 0, 0，颜色 2 为黑色，它的 RGB 值为 0, 0, 0。

3.1.2 什么是消息（ Message ）？

消息是对象间交互和传送信息的载体。对象是面向对象的软件的基本模块，但一个对象的能力往往有限，需要多个对象协同合作，来提高解决问题的能力，如图 3.2 所示。往往对象间的消息的传送是双向的，而且，在每个方向上可能有多个不同的消息定义。在 Java 中，消息的传送是以操作调用的形式出现的，对象 A 传递了一个消息给对象 B 实际上是对象 A 调用了对象 B 的操作。

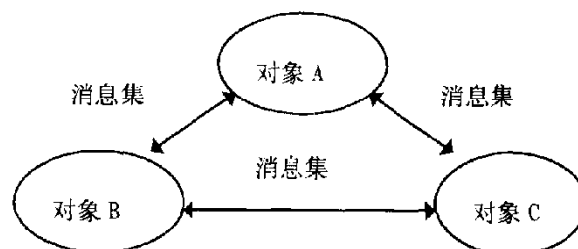


图 3.2 消息传递示意

组成消息的三个要素是：

- 消息要作用的对象

■ 消息要完成的操作名

■ 操作所需的参数

有了以上的元素，对象就可以进行消息所设定的任务。这样的设计的好处是：

■ 一个对象所能完成的工作通过消息和对象的操作来表达出来，所以消息的传递反映了对象间所有可能的交互和信息的传送。

■ 为了相互传送消息，对象无须在同一个进程或机器中。

3.1.3 什么是继承 (Inheritance) ?

继承是面向对象技术的一个重要的特征。它允许编程人员在已有的对象类定义的基础上作进一步的引伸。如下图所示，有一个父类是交通工具，从它引伸出机动类和非机动类两个子类，从这两个子类做进一步的引伸，就到了飞机、汽车、自行车等子类。这反映了一个从一般到具体的不断衍生的过程。子类和父类的关系在逻辑上可以这样验证：子类是不是父类中的一种。在我们这个例子中，子类 1：机动类交通工具是交通工具的一种，因而逻辑上的继承是可能的。图 3.3 是类继承的递阶结构示意图。

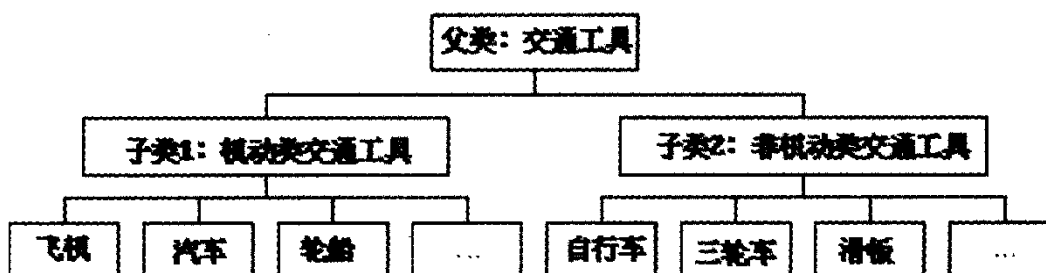


图 3.3 类继承的递阶结构

继承的好处是：子类能够从父类中继承共同的特性。这样，程序员可以多次使用父类代码。如，在父类交通工具中我们定义了大小、重量等属性是不需要在子类 1 中再重复定义一次的。在子类 1 中我们需要增加发动机功率等新属性，而该属性则不需要在子类飞机等中重复定义。只要声明是继承关系，这些属性和操作都会缺省地加入到子类中。程序员不必重写这些代码。

3.2 Java 程序的类定义

在介绍语法之前，我们说明以下在程序中夹杂程序注释的方法，在后续的程序中将要大量地用到。在 Java 中有三种注释的格式，它们分别为：

■ “// ” 用来隐蔽后面的一行文字内容。

■ “/*” “*/” 用来隐蔽所有夹在两个符号中间的内容，可以有許多行。

这两个与 C++ 的完全一样。

第三种是：

- “/** */”与上一个用法几乎同样，但夹在它中间的注释将来可以包括在自动产生的文档中。

一般地，这个注释紧靠着变量声明和操作函数声明之前。通过自动文档生成程序产生相应的代码的文档。

3.2.1 Java 程序的一般结构

Java 的程序在结构上看，有两个部分：一个是引入库声明区；另一部分是类的定义区。

```
// 引入库声明区
// 引入已有的类 OldClass
import OldClass;
// 类定义区
public class MyClass extends OldClass {
    private int var1;
    public int var2;
    protected int var3;
    friendly int var4;
    MyClass() {
        ...;
    }
    MyClass(int arg1) {
        ...;
    }
    protected void finalize () throws throwable{
        ...;
    }
    private void MyFun1(int i){
        this.var1= 10; // 也可以写成 var1=10;
        super.var0=100;
        .....;
    }
    public int MyFun2(...) {
        ...;
    }
}
```

引入库声明区的格式如下：

```
import 类库名;
```


其中的类库名可以是具体的一个类名,如, `java.applet.Applet`; 或者是一类相关的类库名,如, `java.awt.*`; 其中的*号代表了 `awt` (`Another Window Toolkit`) 库中的所有类。这样可以省掉许多重复的声明。有关类库的结构和功能将在后续的章节中要详细地介绍。

除了引入库声明以外,所有的变量、语句、操作都在类中。这与 C++ 有较大的不同。可能在一个文件中有多类的定义。在类定义中,有变量和操作函数两个部分,一般变量在前,便于阅读。它们的具体定义方法和用法在后面介绍。

值得注意的是有两个与类同名的操作函数,它们是构造函数 (`Constructor method`), 主要是为了完成初始化的工作,如把变量赋值等。有一个不带参数,另一个带参数,可以适应不同场合的需要,编译器会根据程序中用法自动地选择加载适当的构造函数。另外,还有析构函数 (`Deconstructor method`) `finalize()`, 是在对象被删除时调用的。所有的类定义的方法都是同样的。它的任务是完成一些收尾的工作,如检查有没有文件还处于被打开的状态,如果是打开的话,把它关上,等等。析构函数可以没有,但为了养成良好的编程习惯,我们应该加上。

3.2.2 Java 的类定义语法

Java 类定义的方法如下例所示:

```
[类类型关键字] class 新类名 [ extends 父类名 ] {  
    [存取权限关键字] [静态标志] 变量声明;  
    [存取权限关键字] [静态标志] 操作定义;  
}
```

在上述的定义中, [] 当中的元素表示可以缺省。

类类型关键字有:

■ `abstract` (抽象类)

抽象类至少要有有一个抽象操作函数。这样的类不能实例化,必须进一步地衍生子类,把抽象操作函数改写。

■ `final` (终结类)

该类型是类继承链的末端,不能作进一步的继承。如实现数学运算的类就是这样的类。

■ `public` (公共类)

该类可以作进一步的继承,或在其他的类中存取。这是最常见的一种方式。

■ `synchronizable` (同步类)

所有的操作函数都是同步的。

存取权限关键字有:

■ `public` (公共方式)

它表示后续的变量或操作函数可以在其他任何类的任何地方存取。

■ `protected` (保护方式)

它表示后续的变量或操作函数只能在本身和引伸的子类中存取。

■ `private` (私有方式)

它表示后续的变量或操作函数只能在本身存取。

■ friendly (友好方式)

它表示后续的变量或操作函数能且只能被同一个“包”(Package)的所有对象存取。这是一种缺省的模式。也就是说,如果程序员不加以说明的话,那么编译器就认为是友好方式。

静态标志关键字为 static。如果加在变量之前,表示所有从该类引伸出的实例都分享同一个变量,也就是说,在内存中只有一个区域,而不象非静态变量那样,每个实例都有独立的一个内存区域。一旦某一个实例修改了该变量,则所有的实例中该变量的值都发生相应的改变。如果静态标志加在操作函数之前,则该函数只能存取静态变量。

在 Java 中还有一类很特殊的结构,即,“接口”(interface)。它包含了一些未实现的操作函数集。接口中的操作函数可以是公共的(public)或抽象的(abstract),变量可以是公共的(public)、静态的(static)和终结的(final)。它与抽象类的差别在于:接口不强制用户去引伸子类。接口的好处是它能保证每个类实现具有同样的操作函数,具有规范化操作函数接口的作用。

在定义接口时,按如下的格式声明:

```
public interface 接口名 {  
    // 描述  
    void fun1();  
    // 描述  
    void fun2();  
} // 接口定义结束
```

按如下的格式声明引伸的类:

```
class 类名 implements 接口名 {  
    void fun1() {  
        <实现部分>  
    }  
    void fun2() {  
        <实现部分>  
    }  
} // 类定义结束
```

在 Java 中,有两个关键词: this 和 super。this 是指向本类的指针,往往可以省掉,而 super 是指向父类的指针。其用法可在前面的例子中看到。

3.2.3 类的使用

要使用 Java 的类,首先要实例化,也即使用类模板生成对象,并在内存中分配相应的单元。如下所示:

```
MyClass testClass = new MyClass();
```

new 后面是 MyClass 的构造函数，是选用了不带参数的一个。定义完成后，就可以引用该类的变量和操作函数，如

```
testClass.var2=12;
testClass.MyFunc(10);
```

引用时，应注意该变量和操作函数的存取权限是公用的还是私有的，如果是私有的，则不能像上面那样引用。具体可参考上面的介绍。

3.3 变量和数据类型

变量是程序中的重点，它存储了数据，所有的操作符都与之有关联。离开了变量，操作也就失去了作用的目标。

```
class Count {
public static void main(String args[ ])
throws java.io.IOException
{
    int count = 0;

    while (System.in.read( ) != -1)
        count++;

        System.out.println("Input has " + count + " chars.");
    }
}
```

一个数据声明包括两部分，变量的类型和变量名。同时，变量的作用范围，也即变量在什么范围内有效。

3.3.1 变量类型

Java 所有的变量都必须有一个数据类型，该数据类型决定了变量的性质及能对该变量所作的操作。例如，我们不能把浮点数赋值给声明为整型的变量，反过来也一样。在 Java 中，有两类数据类型，即基本类型和参考类型。基本类型仅包含一个值，如整数、浮点数、字符、布尔数。表 3.1 列出了基本数据类型的字长和关键字及注释。

参考类型是一组基本类型的组合，如数组。数组型变量本身不存储实际的值，而是代表了指向内存中的存放参考类型数据的位置。这是与基本类型有很大的差别。

除了数组变量以外，类（Classes）和接口（Interfaces）也是参考类型。所以当定义一个类或接口时，实际上也是定义一个新的数据类型。

注意与 C 和 C++相比，有三类 C 语言的数据类型不支持指针（pointer）、结构（struct）和联合（union）。指针在 C 语言中起了重要的作用，极大地增强了程序的灵

活性。但同时带来一个缺点，即指针的越界往往会引起程序混乱，甚至系统崩溃。

表 3.1

类 型	关键字	大小/格式	注 释
整型数	byte	8 比特长	字节长整数
	short	16 比特长	整数短
	int	32 比特长	标准整数
	long	64 比特长	长整数
实数	float	32 比特长	单精度浮点数
	double	64 比特长	双精度浮点数
其它	char	16 比特长	Unicode 标准代码
	boolean	N/A	布尔型变量（真或假）

3.3.2 变量的命名规则

习惯上，变量名以小写字母开头，类名以大写字母开头。为了表达清楚，可以用长字符串代表变量的意义，如，为表示员工的月工资，可如下声明：

```
int monthlyBaseSalary; // 员工月工资
或者
int 员工月工资;
```

Java 的变量命名要遵从如下的三条规则：

1. 必须由 Unicode 字符集中的字符组成。Unicode 字符集是一套字符编码系统，可以支持各类文字的字符，总数达 34168 个字符。这个特点可以允许编程人员使用各类字符集来命名变量和加入程序的注释，如在程序中加中文的注解，声明中文的变量名。这样方便了各种文化背景的编程人员使用 Java 语言。

2. 不能与 Java 语言的关键字相重，或取成布尔值（true 或 false）。

3. 在同一个作用范围内，不能有相同名字的两个变量。这条规则暗示我们，在不同的作用范围中可以有同名的变量存在。

3.3.3 变量的作用范围

所谓变量的作用范围是指可以存取变量的代码模块。变量的作用范围同时也决定了变量何时产生，何时消灭。在声明变量的同时，实际上也定义了它的范围。变量的作用范围可以分成如下的四类：

- 成员变量
- 局部变量
- 操作（method）参数
- 异常处理函数参数

成员变量是一个类或一个对象的成员，在类中声明。局部变量在一个操作内或在一段

代码中声明。一般地，局部变量的作用范围为从它被声明的点到这段代码的结束。操作变量是用来向操作或构造函数传递数据的，它的作用范围是引用该变量的整个操作或构造函数。异常处理函数参数与操作变量相同。

3.3.4 变量初始化

局部变量和成员变量可以在它们被定义时候进行初始化。所赋的值必须与变量的类型相一致，如

```
int count = 0;
```

后两类操作（method）参数和异常处理函数参数则不能这样处理。参数的值由调用者设定。

3.4 数组和字符串

3.4.1 字符串

字符串是有序的字符的集合，在 Java 中有相应的类支持（`java.lang.String`）。字符串可以有如下的两类的写法：

一种是直接的，如“`“Hello! My name is Jin Zunhe ”`”。这样能够在程序中引用。另一种是按类声明一样处理，如

```
String string1 = new String("Hello! My name is Jin Zunhe");
```

字符串之间用操作符+能够实现合并，如

```
"Hello! " + "My name is Jin Zunhe"
```

等价于“`“Hello! My name is Jin Zunhe ”`”。

值得指出的是，在 Java 中，`String` 类型中存储的值是不能改变的，如果要变化的话就要用到另外一个类 `StringBuffer`，这是 Java 的一个特点。它的用法与 `String` 很相似，这里不再赘述。参考相应的手册了解细节。

3.4.2 数组

和其他的变量一样，在使用数组之前，必须声明定义它，格式为：

变量类型 数组名[]；

如

```
int list[ ];
```

“[]”部分表示它是个数组，变量类型说明数组中的每个元素的数据类型。在进行定义后，程序实际上还没有真正地给每个元素分配内存。如果试图给数组中的任一元素赋值，编译器将要打印如下的错误：

```
testing.java:64: Variable list may not have been initialized.
```

为了给数组分配内存，需要用 Java 中的 `new` 语句，这与产生一个新对象一样。例如：

```
list = new int[20];
```

就可以在内存中分配 20 个整数元素区域。如果想简化，可以把定义语句和初始化语句合在一起，如下所示

```
int list[] = new int[20];
```

在分配区域后，就能存取数组的任何元素。如下面的这段程序，就是给每个元素赋值，并把其值打印到标准输出。程序中的 `list.length` 是指数组的长度，由程序自动计算，任何其他的数组都有同样的特性。与 C 语言一样，数组的第一个元素是从 0 开始编号的，最后的一个是数组长度减去 1。

```
for (int j = 0; j < list.length; j++) {  
    list[j] = j;  
    System.out.println("[j] = " + list[j]);  
}
```

数组的元素还可以是任何合法的参考类型的数据，甚至是对象。需要指出的是，此时数组元素的初始化，往往容易忽略。

3.5 操作符

操作符对两个或一个变量或常数进行运算或变换。只需一个操作数的操作符称为单操作符 (unary operator)，如 “++”、“--”。而需要两个操作数的称为双操作符 (binary operator)，如 “+”、“-”。

对单操作符，可以放在操作数的前面，也可以放在操作数的后面，如：

```
count++;
```

或者

```
++count;
```

对于双操作符，必须放在操作数之间，如：

```
A + B;
```

除了执行操作之外，操作符还返回一个结果，其类型取决于两边的操作数和操作符。比如，如果上边的语句中 A 和 B 都是浮点数，则所加的结果也是浮点数。

在 Java 中，操作符可以分成如下的类别：算术运算操作符 (arithmetic)、关系和条件操作符 (relational and conditional)、位操作符 (bitwise)、逻辑操作符 (logical) 和赋值操作符 (assignment)。

3.5.1 算术运算操作符

Java 中的算术操作符如表 3.2 所示。

注意：

1. 与 C++ 一样，Java 扩展了操作符 “+” 的功能，在字符串操作中可以当连接符使用。如下面的例子，当变量 `count` 等于 5 时，系统输出一个字符串 “Input has 5 chars”。

```
System.out.println("Input has " + count + " chars.");
```

2. 操作符“-”还可以用来表示把操作数正负性改变。如， $A = -B$ ；表示把 B 的相反数赋值给变量 A。

表 3.2

操作符	用 法	描 述
+	$op1 + op2$	op1 加上 op2
-	$op1 - op2$	op1 减去 op2
*	$op1 * op2$	op1 和 op2 相乘
/	$op1 / op2$	op1 除以 op2
%	$op1 \% op2$	op1 除以 op2 的余数
++	op++或++op	op 加上 1
--	op--或--op	op 减去 1

3.5.2 关系和条件操作符

关系操作符比较两个操作数并决定两者的关系。例如， $!=$ 将返回真（true）如果两个操作数不相等。表 3.3 给出了 Java 中的关系操作符。

表 3.3

操作符	用 法	返回真的条件
>	$op1 > op2$	op1 大于 op2
>=	$op1 \geq op2$	op1 大于、等于 op2
<	$op1 < op2$	op1 小于 op2
<=	$op1 \leq op2$	op1 小于、等于 op2
==	$op1 == op2$	op1 等于 op2
!=	$op1 != op2$	op1 不等于 op2
&&	$op1 \&\& op2$	op1 和 op2 同时为真
	$op1 op2$	op1 和 op2 有一个为真
!	! op	op 为假

条件操作符常常与关系操作符结合在一起使用，组成复杂的条件判断语句。如下边的语句是用来判断变量 numberOfItem 是否在 0 和常量 MAX_ITEM 之间：

```
0 < numberOfItem && numberOfItem < MAX_ITEM
```

注意：在 op1 和 op2 为布尔型数时，操作符 &、| 可以替换 && 和 ||。但为了程序的可读性，我们不鼓励这样编程。

3.5.3 位操作符、逻辑操作符

位操作符允许编程人员对数据的位进行操作。表 3.4 总结了 Java 语言中位操作和逻辑操作符的用法。

表 3.4

操作符	用 法	描 述
>>	op1 >> op2	把 op1 的每一位向右移 op2 位
<<	op1 << op2	把 op1 的每一位向左移 op2 位
>>>	op1 >>> op2	把 op1 的每一位向右移 op2 (不包括符号位)
&	op1 & op2	把 op1 和 op2 的每一个位进行“与”操作
	op1 op2	把 op1 和 op2 的每一个位进行“或”操作
^	op1 ^ op2	把 op1 和 op2 的每一个位进行“异或”操作
~	~ op	每一个位进行“取补”操作

3.5.4 赋值操作符

赋值操作符是把一个值传送到一个变量中。除了正常的赋值形式外，Java 还提供了一些组合的快速方法，如表 3.5 所示。但我们建议一般读者不要使用。

表 3.5

组合操作符	用 法	等价的一般用法
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2
&=	op1 &= op2	op1 = op1 & op2
=	op1 = op2	op1 = op1 op2
^=	op1 ^= op2	op1 = op1 ^ op2
<<=	op1 <<= op2	op1 = op1 << op2
>>=	op1 >>= op2	op1 = op1 >> op2
>>>=	op1 >>>= op2	op1 = op1 >>> op2

3.6 表达式

表达式是执行 Java 程序的基本单元之一，它是按照语言的语法规则结合在一起的一系列变量、操作符和函数调用的组合。它可以用来计算、给变量赋值、帮助控制程序执行流程。

简单的表达式组合在一起可以完成复杂的工作。这时往往有许多的操作符，而且操作符的顺序对结果有很大的影响。一般操作符的优先级规则是：位操作符大于乘除大于加减大于逻辑运算符。

3.7 流控制语句

Java 语言中的流控制语句包括:

- 分支语句, if-else, switch
- 循环控制语句, for, while, do-while
- 异常处理语句, try-catch-finally, throw
- 其他, break, continue, label:, return

3.7.1 分支语句

1. if-else 语句

该语句的一般结构为:

```
if (表达式
    语句 1;
}
else {
    语句 2;
}
```

在执行时,系统首先计算表达式,根据得出的布尔值,当为真时,执行语句 1,当为假时执行语句 2。每个语句都可以嵌套,从而变化为更复杂的形式。下例中嵌套了多重的 if-else 语句。编程时还可以嵌套其他形式的语句。

```
int score;
char grade;
if (testscore >= 90) {
    grade = 'A';
} else if (testscore >= 80) {
    grade = 'B';
} else if (testscore >= 70) {
    grade = 'C';
} else if (testscore >= 60) {
    grade = 'D';
} else {
    grade = 'F';
}
```

2. switch 语句

该语句的一般形式为:

```
switch (变量) {
```

```

        case 值 1:      语句 1;
                        break;
        case 值 2:      语句 2;
                        break;
        ...
        default:      缺省处理语句;
                        break;
    }

```

通过 if-else 语句可以实现它所有的功能，但它比 if-else 语句在某些时候显得清楚明确。

3.7.2 循环控制语句

1. while 语句和 do-while

它的一般格式是：

```

while(表达式) {
    语句;
}
do {
    语句;
} while (expression);

```

2. for 语句

它的一般格式是

```

for(语句 1; 表达式; 语句 2) {
    语句 3;
}

```

3.7.3 异常处理语句

Java 会监测错误，并诊断发生了什么类型错误。一般地，当有错误发生时，程序会停止运行，并把错误信息打印出来。程序本身也可以截获错误，作出相应的处理。下面的几条语句就是为了这个目的。

1. try-catch-finally

它的一般格式为：

```

try {
// 一些可能产生异常处理的语句 1;
// 一些可能产生异常处理的语句 2;

```

```

    } catch(某种错误类型1 错误变量名) {
    // 针对错误类型1, 执行异常处理的操作;
    catch(某种错误类型2 错误变量名) {
    // 针对错误类型2, 执行异常处理的操作;
    } finally {
    // 执行异常处理的操作
    }

```

catch 语句可以有多个, 用来处理不同的错误。如以下的代码, 主要处理了被 0 除的错误, 对其他错误统而笼之地进行处理。当语句 1 发生错误时, 第二条指令将不被执行。不管有没有捕获到错误, finally 后面的操作总是要执行的。catch 和 finally 不一定要同时出现, 可以单独出现。

```

try {
    int var=0;
    var = 1/var;
} catch ( ArithmeticException e) {
    System.out.println("Error---You tried to divide by 0");
}
    catch (Exception e) {
    System.out.println("Some other error was thrown");
}

```

2. throw

它的格式为:

throw 例外对象;

其中的例外对象可以是自己创建的新对象, 如

```

class newException extends Exception {
    .....
    void MyMethod( ) {
        try {
            if(no error) {
                // 正常流程
            }
            else throw new newException();
        } catch ( newException e) {
            // 处理该新型例外情况
        }
    }
}

```

3.7.4 其他

1. break

我们在 switch 语句中已经看到了 break 的用法,它引起程序立即跳到当前语句后的那条语句。另外, break 语句还能跳到一个标注的语句。标注的语法是:

标注名: Java 语句;

跳到标注的语法是:

break 标注名;

这条语句是 goto 语句的代用品,虽然 Java 保留了 goto 作为关键字,但实际上并没有使用。

2. continue

在循环中使用 continue 语句,可以把控制转移到当前循环的开始部分,跳过 continue 后面的部分。

3. return

return 语句可以从当前的函数调用中返回到调用的地方。return 后面可带参数,也可以不带参数。要注意的是返回的数值类型必须与函数声明的类型一样,否则会在编译时会得到警告或错误信息。如,声明为 void 的函数,就不应该在 return 后面带参数。

3.8 小 结

在本章中,我们学习了如下的内容:

- 面向对象的概念。分析了类、对象的区别,理解消息机制、封装等概念,介绍了面向对象的优越性。

- Java 中的类定义语法和类的使用方法。介绍了 java 程序的一般结构和类定义中的组成部分,具体地讲解了类的定义语法和关键词的语法和寓意,示例了类的使用方法和注意点。

- java 程序中的语法细节。包括,变量类型和定义,数组、字符串,操作符和表达式,和流控制语句等。

第四章 Java 基础类库

代码重用是面向对象程序设计的一个最显著优势，生成可重用、可继承的类可以节省大量的时间和精力，大大提高编程的效率。Java 把代码重用作为实现的核心，提供了许多可为 Java 编程者使用的标准对象，这些标准的 Java 对象统一称作 Java 类库。

Java 类库是以软件包（Packages）的形式实现的，每个软件包由一组相关的类组成。它们包括文件输入/输出，系统调用，图形用户接口和网络协议类等，使用这些类，你可以迅速地开发出 Java 应用。本章首先将简要描述 Java 类库结构，然后着重分析 Java 基础类库。

4.1 Java 类库结构

Java 类库从功能上可划分为：语言类库、输入/输出类库、实用程序类库和 applet 类库、图形用户接口（awt）类库、网络类库，我们把前三种类库称为基础类库，把后三类类库称为应用类库。从使用上可分为两大类——适用于所有 Java 解释器的类库和仅适用于浏览器环境的类库，其中 Applet 类库是只适用于浏览器环境的类库。

4.1.1 基础类库

Java 基础类库提供了 Java 编程所需的基本系统功能软件包，它们是 `java.lang`，`java.io` 和 `java.util`。你可以使用标准的 `import` 语句输入这些类库，如下所示：

```
import java.lang.*;
import java.io.*;
import java.util.*;
```

这将在你的程序中包含所有的 Java 基础类库，当然，你也可以仅输入你所需的类库：

```
import java.util.Date;
```

1. 语言类库

Java 语言类库即 `java.lang` 软件包，它封装了各种基本编程功能类方法。对 Java 语言类库一点不了解，是不可能编写出 Java 程序的。它主要包括以下一些重要的类：

- 基本对象（Object）类
- 布尔数、字符和数字类型包容器
- 基本数学函数类
- 字符串和字符串缓冲器类
- 标准输入/输出等系统类
- 线程控制和例外处理类

■ 其它抽象类

这些类是 Java 编程的基础，它提供了基本对象处理方法，本章后面部分将进一步说明。

2. 输入/输出类库

Java 输入/输出类库即 `java.io` 软件包，提供了对不同的输入和输出设备读写数据的支持，这些输入和输出设备包括键盘、显示器、打印机、磁盘文件和网络等。它主要包括如下类：

- 输入流类
- 输出流类
- 文件访问类
- 流标记类

如果你打算在你的程序中使用输入/输出功能，就需要熟悉 `java.io` 软件包，它提供了所有与 I/O 相关的类。

3. 实用程序类库

Java 实用程序类库即 `java.util` 软件包，它提供了执行各种辅助功能的类，包括日期处理、随机数生成等。主要的类如下：

- 日期类
- 向量和栈处理等类
- 随机数类

4.1.2 应用类库

Java 应用类库支持 Stand-alone 和 Applet 应用的高级功能。其中 `applet` 类库是在浏览器环境中实现 Applet 的标准，`awt` 类库提供了构成用户界面的按钮、菜单和滚动条等功能，而网络类库则为用户实现了文件传送、套接字和远程登录等网络协议。使用这些类，程序员可以迅速开发出符合用户要求的高级应用。

1. applet 类库

`applet` 类库提供了在 WWW 浏览器环境中实现 Java Applet 的类，通过它你可以控制 HTML 文本的格式，在 Applet 中实现音频播放等功能。

2. awt 类库

AWT 类库提供生成图形用户界面的类，使用该类库你可以控制 Applet 在浏览器中的外观。在 `awt` 类库中包括的类方法有：窗口、按钮、菜单、字体、图象、滚动条等。

3. 网络类库

网络类库提供了对网络协议的接口功能，在该类库中处理的网络协议包括：`Sockets`、`Telnet`、`FTP`、`NATP`、`WWW` 等。采用网络协议类，可以非常容易地开发出利

用 Internet 上资源的应用，体现了 Java 语言分布式的特点。

4.2 语言类库

Java 语言类库是 Java 类库的核心部分，从附录 A 我们可以看到，Java 类层次关系图的顶部是 Object 类，由它派生出其它的类，而 Object 类就包含在语言类库中。

java.lang 软件包在编译时自动加载，因此不需要明确地通过 import 语句输入。该类库中有许多类函数是作为静态 (static) 成员实现的，由于静态成员不需指定类的一个实例即可以访问，因此为了使用其功能，就不必生成类的一个实例。例如，为了确定一个整数的绝对值，你可以按照如下方法使用 Math 类成员函数 abs()：

```
int a = -1  
a = Math.abs(a); // 现在 a = 1
```

这里，没有生成 Math 类的一个实例就使用了其静态成员函数 abs()。在前面的例子中，System.out.println() 和 System.in.read() 也是以类似的方式使用的。

4.2.1 Object 类

Object 类可能是 Java 类库中最重要类，因为它是所有 Java 类的超类。Object 类实现的成员函数将被所有派生的类继承，这就意味着你可以在自己定义的类中使用其成员函数。以下是 Object 类中一些非常有用的成员函数：

■ protected Object clone()

调用该成员函数的对象将生成一个新对象，并将其内容复制到新对象。使用该成员函数的一个例子如下：

```
Rectangle rect = new Rectangle(10,20);  
Rectangle rectCopy = rect.clone;
```

在该例子中，生成了一个对象 rect，但只是定义了对象 rectCopy，并没有使用操作符 new 生成它，对象 rectCopy 是由对象 rect 调用成员函数 clone 生成的。

■ public int hashCode()

该成员函数返回一个对象的散列码 (hashcode) 值，散列码是 Java 系统中表示对象的唯一整数。

■ public final Class getClass()

该成员函数以 Class 类的形式返回一个对象的类信息，对象的类信息包括类名、父类名、实现的接口和其它属性。例如，Class 类有一个成员函数 getName 可返回包含该类名的字符串。

```
Rectangle rect = new Rectangle(10,10);  
System.out.println("class name: " + rect.getClass().getName());
```

■ public String toString()

该成员函数返回一个表示对象值的字符串。由于对象值取决于某个类的类型，因此派生类需要编写重载函数以获取指定类的信息。在调试时，由函数 toString 返回的信息可用

来确定该对象的内部状态。

4.2.2 类型容器

类型容器是对各种数据类型——布尔数、字符、整数和浮点数等——类实现的总称。类型容器和数据类型本身是有区别的，前者是类，而后者不是。这点很容易搞混，因此从类型容器中获取数据类型一定要使用成员函数 `typeValue()`。类型容器实现的类有：`Boolean`、`Character`、`Double`、`Float`、`Integer` 和 `Long`，其中的每个类都有其特定的成员函数，但以下几个成员函数是共同的：

■ `public ClassType`

所有类型容器的构造函数都需要一个参数作为其封装的数据类型，下面的例子生成了整数、字符和浮点数的容器。

```
Integer i= new Integer(1);
Character ch = new Character('a');
Float flt = new Float(1.234);
```

■ `public type typeValue()`

该成员函数将返回生成的类型容器的原始数据类型值，类型容器并不等价于原始数据类型，因此为了处理类型容器的值必须使用该函数。

```
int x = i.intValue();
char y = ch.charValue();
float z = flt.floatValue();
```

■ `public String toString()`

该成员函数将返回容器中类型的字符串表示，举例说明如下：

```
System.out.println(i.toString());
System.out.println(ch.toString());
System.out.println(flt.toString());
```

■ `public boolean equals(Object obj)`

除了 `Character`，其它类型容器都实现了该成员函数。类型容器不能使用等号（`=`）直接进行比较，因此必须通过该成员函数来比较两个数据类型对象是否相等，如果相等将返回真，否则返回假。

```
Integer j = new Integer(2);
boolean bool = j.equals(i); // bool 为假
```

1. Boolean 类

`Boolean` 类为布尔数据类型提供了一个对象容器及面向对象的操作方法。除了前面介绍的类型容器共同的成员函数，`Boolean` 类还有如下一个成员函数：

■ `public static boolean getBoolean(String name)`

它返回参数 `name` 的布尔属性值。

`Boolean` 类有两种状态：`TRUE`（真）或 `FALSE`（假），你可以通过下面的方法改变一

个对象的状态:

```
bool = Boolean.TRUE; 或 bool = Boolean.FALSE;
```

2. Character 类

除了类型容器共同的成员函数, Character 类还实现了以下的成员函数:

- `public static boolean isLowerCase(char ch)`

如果字符 `ch` 是小写的则返回真, 否则返回假。

- `public static boolean isUpperCase(char ch)`

如果字符 `ch` 是大写的则返回真, 否则返回假。

- `public static int digit(char ch, int radix)`

该函数返回字符 `ch` 的整数值, 其中 `radix` 作为基数 (2、8、10、16 分别表示二进制、八进制、十进制和十六进制), 例如:

```
int x = Character.digit('a', 16); // x 等于 10
```

- `public static boolean isDigit(char ch)`

如果字符 `ch` 是数字 (0 ~ 9) 则返回真, 否则返回假。

- `public static char forDigit(int digit, int radix)`

该函数根据指定的基数 `radix` 返回数字 `digit` 的字符值。

- `public static char toLowerCase(char ch)`

该函数将字符 `ch` 转换为小写字符。

- `public static char toUpperCase(char ch)`

该函数将字符 `ch` 转换为大写字符。

不知你有没有注意到, Character 类的成员函数都是静态的, 你可以直接引用 Character 类本身来使用其成员函数。如果成员函数不是静态的, 使用成员函数的唯一方法就是生成该类的一个实例, 举例说明如下:

```
Character ch = new Character('a');
```

```
boolean LowerCase = ch.isLowerCase('b'); // LowerCase 为真
```

在上面的例子中, 为了使用 Character 类的成员函数 `isLowerCase`, 生成了一个实例 `ch`。其实, 由于 Character 类的成员函数是静态的, 可以用下面的一行代码来替换。

```
boolean LowerCase = Character.isLowerCase('b');
```

3. Number 类

在 Java 语言类库中, Number 类是作为抽象类实现的, 它是整数、长整数、浮点数和双精度数的类型容器。该类提供了以下几个成员函数:

- `public int intValue()`

- `public long longValue()`

- `public float floatValue()`

- `public double doubleValue()`

上述成员函数主要进行数据类型的转换。例如, 如果你有一个双精度数, 想得到它对应的整数值, 可使用如下代码:

```
Double d = new Double(1.234);  
int x = d.intValue();
```

当然，从双精度数转换为整数，精度将受到损失。

4. Integer 类

Integer 类是整数类型包容器，它主要增加了以下几个成员函数：

- `public static int parseInt(String s, int radix)`
- `public static int parseInt(String s)`

`parseInt` 将一个字符串解析为整数值，参数 `radix` 作为基数，无该参数则以 10 为基数。

- `public static Integer getInteger(String name)`
- `public static Integer getInteger(String name, int val)`
- `public static Integer getInteger(String name, Integer val)`

`getInteger` 返回由字符串 `name` 指定的整数属性值。上述三个成员函数都是静态的，因此不需生成 Integer 对象就可以使用它。其不同就在于如果不存在该属性时，第一个函数将返回 0，第二个函数将返回参数 `val` 的整数值，而第三个函数将返回 Integer 类参数 `val`。

Integer 类包括两个静态的数据成员：`MINVALUE` 和 `MAXVALUE`，它们用来指定 Integer 对象可表示的最小和最大数。

5. Float 类

Float 类是浮点数类型包容器，它主要增加了以下几个成员函数：

- `public boolean isInfinite()`
- `public static boolean isInfinite(float v)`

`isInfinite` 判断一个浮点数是否为无穷数，这包括由成员变量 `NEGATIV_INFINITY` 和 `POSITIVE_INFINITY` 表示的负无穷数和正无穷数。

- `public static int floatToIntBits(float value)`
- `public static float intBitsToFloat(int bits)`

上面两个成员函数将一个浮点数转换为二进制位表示或者反之。

4.2.3 数学函数类

Math 类包含许多非常有用的数学函数和两个成员常量：`e`(2.7182...) 和 π (3.1415...)。该类中的成员函数都是静态的，因此不必生成 Math 类的实例即可使用，只需按如下方式调用：

```
Math.method(variable);
```

该类提供的主要成员函数如下：

- `public static double sin(double a)`
- `public static double cos(double a)`

- public static double tan(double a)
- public static double asin(double a)
- public static double acos(double a)
- public static double atan(double a)
- public static double exp(double a)
- public static double log(double a)
- public static double sqrt(double a)
- public static double ceil(double a)
- public static double floor(double a)
- public static double rint(double a)
- public static double atan2(double a, double b)
- public static double annuity(double a, double)
- public static double pow(double a, double b)
- public static int round(float a)
- public static int round(double a)
- public static void srandom(double a)
- public static double random()
- public static int abs(int a)
- public static long abs(long a)
- public static double abs(double a)
- public static int max(int a, int b)
- public static long max(long a, long b)
- public static float max(float a, float b)
- public static double max(double a, double b)
- public static int min(int a, int b)
- public static long min(long a, long b)
- public static float min(float a, float b)
- public static double min(double a, double b)

关于上述函数的说明，读者可参阅 Java 开发工具相关文档，这里不再详述。

4.2.4 字符串类

Java 类库提供了两个字符串类：String 和 StringBuffer，其中 String 类用于不变的常量字符串，而 StringBuffer 用于可变的字符串。C 和 C++ 不提供字符串类机制，它只是用字符数组来表示字符串。象 Java 这种安全性和可移植性要求非常高的环境，使用数组表示字符串是不能令人满意的，它经常可能出现数组越界的危险。而将字符串封装为类，编程者就不必为这些问题担心了。

1. String 类的使用

String 类有好几个构造函数，最基本的是使用带双引号的字符串，当 Java 编译器遇到带双引号的字符串时，即生成 String 类的一个实例，因此，String 类对象初始化的最简单方式可表示如下：

```
String s = "Hello, World!";
```

当然，也可以按如下的传统方式来初始化 String 类：

```
String s = new String(someString);
```

其参数 someString 可为字符串或 String 对象。此外 String 类还有如下几个构造函数：

- `public String(char value[])`

- `public String(char value[], int offset, int count)`

使用字符数组的子串作为初始化参数，其中 offset 是字符数组的起始偏移量，count 表示字符串的长度。

- `public String(byte ascii[], int hibyte)`

- `public String(byte ascii[], int hibyte, int offset, int count)`

使用字节数组作为初始化参数。在这种情况下，由于 Java 采用了 UNICODE 字符集，需要说明 16 位字符的高 8 位，因此使用了参数 hibyte。对于 ASCII 码字符，hibyte 设置为 0。

String 类有许多成员函数，下面列出了一些最重要的成员函数：

- `public int length()`

返回字符串的长度。

- `public char charAt(int index)`

返回指定位置 index 的字符。

- `public boolean equals(Object o)`

比较两个 String 对象，如果内容相同则返回真，否则返回假。

- `public int compareTo(String s)`

比较两个字符串的词法关系（按字典顺序）。如果 s 比该字符串大则返回一个小于 0 的数，如果 s 比该字符串小则返回一个大于 0 的数，如果两个字符串相等则返回 0。

- `public boolean regionMatches(int toffset, String other, int ooffset, int len)`

判断两个字符串是否有部分相同。其中 toffset 表示该字符的起始位置，ooffset 表示另一字符串的起始位置，len 表示比较的字符串长度。如果匹配成功则返回真，否则返回假。

- `public boolean startsWith(String prefix)`

判断该字符串是否以字符串 prefix 作为开始，如果是则返回真，否则返回假。

- `public boolean endsWith(String suffix)`

判断该字符串是否以 suffix 作为结束，如果是则返回真，否则返回假。

- `public int indexOf(String str)`

返回该字符串第一次出现子串 str 的位置，如果未找到则返回 -1。

- `public String substring(int beginIndex, int endIndex)`

返回该字符串从 beginIndex 开始, 到 endIndex 结束的子串。

■ `public String concat(String str)`

将指定的字符串 str 连接到该字符串的末尾。

■ `public String toLowerCase()`

将该字符串中所有字符变为小写字符。

■ `public String toUpperCase()`

将该字符串中所有字符变为大写字符。

■ `public char toCharArray()`

将该字符串转换为字符数组。

■ `public static String valueOf(type variable)`

该成员函数是一个多态函数, 可由各种数据类型, 包括 `boolean`, `int`, `long`, `float`, `double`, `char[]` 和 `Object` 等生成字符串。

要在本书中包括 Java 类库中的每个类的每个成员函数是不太可能的, 而且, Java 类库在不断更新, 因此你应该熟悉如何使用 JDK 提供的 Java 类库指南, 或许你会找到一个你所需的成员函数。

2. StringBuffer 类的使用

`String` 类主要处理不变的字符串, 而 `StringBuffer` 类则主要处理可变的字符串, 它提供了对字符串插入和增加数据的方法。当你生成一个 `StringBuffer` 对象后, 可以通过成员函数 `toString()` 将它转换为 `String` 对象, 然后就可以使用 `String` 类成员函数来对它进行操作。

`StringBuffer` 有如下三个构造函数:

■ `public StringBuffer()`

生成一个空的字符串缓冲器。

■ `public StringBuffer(int length)`

生成一个指定长度的空字符串缓冲器。

■ `public StringBuffer(String str)`

由字符串 str 生成一个字符串缓冲器。

`StringBuffer` 类提供了两个多态的成员函数 `append()` 和 `insert()`, 实现了对多种数据类型, 如字符、整数、浮点数和其它对象的操作。下面的例子说明了使用 `StringBuffer` 类, 根据键盘输入来建立字符串:

```
StringBuffer str = new StringBuffer();
System.out.println("Enter Width:");
while((ch = System.in.read()) != '\n') {
    str.append(ch);
}
```

在该例中, `StringBuffer` 类对象初始化为空字符串, 它将为在字符串末尾添加的新字符自动分配空间。下面是 `StringBuffer` 类一些重要的成员函数。

■ `public int length()`

返回缓冲器中的字符串长度（不是缓冲器空间大小）。

- `public int capacity()`

返回缓冲器中用于新增字符的空间大小（不包括新分配的空间）

- `public synchronized StringBuffer append(type variable)`

将 `variable` 类型的数据，如 `Object`、`int`、`long`、`float`、`double`、`char[]` 和 `boolean` 等，添加到字符串缓冲器的末尾。

- `public synchronized StringBuffer insert(int offset, type variable)`

将 `variable` 类型的数据插入到有参数 `offset` 指定的位置。

- `public String toString()`

将 `StringBuffer` 转换为 `String`。

4.2.5 System 与 Runtime 类

`System` 与 `Runtime` 类提供了对系统和运行环境资源进行访问的功能。`System` 类最有用的一个功能就是标准输入和输出流，标准输入流用于读取数据，标准输出流则用于打印数据。`Runtime` 类则主要用于执行系统命令和确定系统资源，如系统可用内存大小等。

1. System 类

`System` 类的所有变量和成员函数都是静态的，这意味着不必生成 `System` 类的一个实例，就可以调用其成员函数。

`System` 类包括三个变量：

- `public static InputStream in`

- `public static PrintStream out`

- `public static PrintStream err`

这些变量是 `InputStream` 和 `PrintStream` 类的实例，它与 `stdin`、`stdout` 和 `stderr` 的交互提供了 `read()`、`print()` 和 `println()` 等成员函数。通常，`stdin` 是指键盘，`stdout` 是指终端，而 `stderr` 在缺省时是指屏幕。

下面是 `System` 类的一些重要成员函数：

- `public static void arraycopy(type src[], int srcpos, type dest[], int destpos, int length)`

将源数组 `src[]` 中从指定位置 `srcpos` 开始、长度为 `length` 的部分，拷贝到目的数组 `dest[]` 的指定位置 `destpos`。这里 `type` 可以是 `boolean`、`byte`、`char`、`short`、`int`、`long`、`float`、`double` 或 `object`。

- `public static long currentTimeMillis()`

返回当前的格林威治时间（从 1970 年开始，以毫秒数计算）

- `public static Properties getProperties()`

- `public static String getProperty(String key)`

- `public static String getProperty(String key, String def)`

`getProperties` 获取当前系统属性并通过 `Properties` 类对象返回结果。而后两个函数

是获取系统某个属性，其中前者返回参数 key 指定的属性，后者当未找到参数 key 指定的属性时，返回参数 def 的缺省值。

- `public void setProperties(Properties props)`

用 Properties 类对象为参数来设定系统属性。

- `public void gc()`

使 Java 运行系统清理内存“垃圾”，当系统内存缺少时，可用它来释放无用内存。

- `public void loadLibrary(String libname)`

Java 系统支持动态链接库中的可执行代码，该成员函数用于载入由参数 libname 指定的动态链接库。

2. Runtime 类

Runtime 类是用于访问 Java 系统资源的另一个非常有用的类，下面是该类的一些成员函数。

- `public Process exec(String command)`

执行由参数指定的系统命令，返回一个子进程。它是一个多态成员函数，有多种参数格式，读者可参考 JDK 中的 Java 类库指南。

- `public void exit(int status)`

根据 status 指定的退出状态，终止解释器的运行。

- `public long freeMemory()`

返回系统剩余内存字节数，它只是系统可用内存的一个估计值。

- `public long totalMemory()`

返回系统总的内存字节数。

4.3 输入/输出类库

数据输入和输出在程序中是至关重要的一环，许多程序都要求有用户输入，并将信息输出到屏幕、打印机或文件。Java 输入/输出类库为各种设备数据输入和输出的处理提供了相当多的类，这里我们不可能对每个类作介绍，只对其中一些常用的类作些描述。

4.3.1 输入流类

Java 输入模型是基于输入流的概念，Java 使用输入流的方式从输入源——如键盘——读取数据。Java 支持的基本输入流类有：

- `InputStream`

- `BufferedInputStream`

- `DataInputStream`

- `FileInputStream`

- `StringBufferInputStream`

1. InputStream 类

InputStream 类是一个抽象类，它作为其它输入流类的基类，定义了读取字节流信息的基本接口。由于每个 InputStream 的派生类都有类似的功能，在你对 InputStream 类的成员函数熟悉后，就很容易掌握其它的输入流类。

使用输入流类的典型方法是生成一个 InputStream 的对象，调用相应的成员函数来获取输入信息。如果当前没有输入信息，它将使用“阻塞（blocking）”技术来等待输入数据。通过输入流从键盘读取信息正是发生阻塞的一个例子，直到用户敲入信息并回车，InputStream 对象才得到输入数据。

InputStream 类定义了以下成员函数：

- public abstract int read()
- public int read(byte b[])
- public int read(byte b[], int off, int len)

InputStream 定义了三个不同的 read 函数来以不同的方式读取输入数据。第一个函数没有参数，只是简单地从输入流读取一个字节的的数据，并把它作为整数返回，如果到了输入流的末尾则返回-1。由于该函数将输入字节作为整数返回，如果正在读取字符，则需要把它变换为 char。第二个函数以一个字节数组作为参数，使你能一次能读取多个字节数据，你应该保证该字节数组足够容纳读取的信息，否则将会出现 IOException 类型的例外。该函数返回读取的实际字节数，如果到了输入流的末尾则返回-1。第三个函数有三个参数，参数 off 指定了在字节数组中存放新数据的起始位置，参数 len 指定了读取的最大字节数。

- public long skip(long n)

该函数用于跳过输入流中的 n 个字节数据，它返回实际跳过的字节数，如果到了输入流的末尾则返回-1。

- public int available()

该函数用于确定在没有阻塞的情况下可以读取的输入数据字节数。使用它可在读数据时避免阻塞机制。

- public synchronized void mark(int readlimit)
- public synchronized void reset()

mark 函数用于标记流的当前位置；该位置可在以后由 reset 函数返回。上述两个函数常用于当你想读取流后面某个位置的数据而又不丢失原来的位置的情况。这种情况的一个例子是验证文件类型，在你读取文件头并在文件头的末尾标记后，你可能想读一些数据来检验它是否符合文件类型，然后再回到先前的位置。

- public boolean markSupported()

检验一个输入流是否支持 mark/reset 机制并返回一个布尔类型值。

- public void close()

关闭一个输入流并释放与该流相关的资源。当 InputStream 流对象“破坏”时，输入流将自动关闭，因此不必明确地调用 close 函数。但在使用完一个流以后最好还是立即调用 close 函数，因为这将刷新流缓冲器，避免文件遭到破坏。

2. BufferedInputStream 类

BufferedInputStream 类为输入提供了缓冲流，它在每次从输入设备读数据时，将读取较多的数据并放入缓冲流，以后就可以直接从流缓冲区而不是输入设备读数据。这样可以大大加快读操作，因为从缓冲区读数据实际上是从内存读数据。BufferedInputStream 类的成员函数与 InputStream 相同，它没有提供新的成员函数。但 BufferedInputStream 有如下两个不同的构造函数：

- BufferedInputStream(InputStream in)
- BufferedInputStream(InputStream in, int size)

这两个构造函数都以 InputStream 对象作为第一个参数，它们的不同是内部缓冲区的大小。第一个构造函数使用了一个缺省的缓冲区大小，而第二个构造函数可以使用参数 size 指定缓冲区的大小。为了支持缓冲输入，BufferedInputStream 类定义了如下的一些成员变量：

- protected byte buf[]
- protected int count
- protected int pos
- protected int markpos
- protected int marklimit

字节数组 buf 是输入数据存储的缓冲区；count 表示缓冲区中实际存储的字节数；pos 指示在缓冲区中当前读数据的位置；markpos 表明使用成员函数 mark() 后标记在缓冲区中的当前位置；marklimit 表示在标记位置无效前可以读取的最大字节数，它是由成员函数 mark() 的参数 readlimit 来设定的。所有这些成员变量都指定为 protected，所以你实际上可能从来不会去使用它们，但这可以让你多少了解一些 BufferedInputStream 类是如何实现 InputStream 的成员函数的。

下面是程序 ReadKey.java 的源代码，它使用 BufferedInputStream 对象代替 System.in 来从键盘读取输入。

```
import java.io.* ;
class ReadKey {
    public static void main(String args[ ]) {
        BufferedInputStream in = new BufferedInputStream(System.in);
        byte buf[ ] = new byte[10];
        try {
            in.read(buf, 0, 10);
        }
        catch (Exception e) {
            System.out.println("Error: " + e.toString());
        }
        String s = new String(buf, 0);
```

```

    System.out.println(s);
    }
}

```

3. DataInputStream 类

DataInputStream 用来从一个输入流读取基本 Java 数据类型。它只有一个以 InputStream 对象为参数的构造函数，如下：

- DataInputStream(InputStream in)

除了 InputStream 中定义的成员函数，DataInputStream 还实现了以下成员函数：

- public final void readFully(byte b[])

- public final void readFully(byte b[], int off, int len)

功能与成员函数 read() 类似，不同的是它将阻塞直到读完所有数据，而函数 read() 只是在读到部分数据前阻塞。

- public final String readLine()

读以换行符 (\n)、回车符 (\r)、回车/换行符 (\r\n) 或文件结束符 (EOF) 终止的一行文本字符。

- public final boolean readBoolean()

- public final byte readByte()

- public final int readUnsignedByte()

- public final short readShort()

- public final char readChar()

- public final int readInt()

- public final long readLong()

- public final float readFloat()

- public final double readDouble()

上述成员函数用于读取不同的基本数据类型，由成员函数的名字可以很容易地识别所读的数据类型。

下面是程序 ReadFloat.java 的源代码，它使用 DataInputStream 类的成员函数 readLine() 让用户输入一个浮点数并显示。

```

import java.io.*;
class ReadFloat {
    public static void main(String args[ ]) {
        DataInputStream in = new DataInputStream(System.in);
        String s = new String( );
        try {
            s = in.readLine( );
            float f = Float.valueOf(s).floatValue( );

```

```

        System.out.println(f);
    }
    catch (Exception e) {
        System.out.println("Error: " + e.toString());
    }
}
}

```

4. FileInputStream 类

FileInputStream 类用于执行简单的文件输入操作，对于复杂的文件输入则需要使用 RandomAccessFile 类，这将在本章 4.3.3 介绍。FileInputStream 类可以使用以下三个构造函数初始化：

- FileInputStream(String name)
- FileInputStream(File file)
- FileInputStream(FileDescriptor fdObj)

第一个构造函数以一个 String 对象为参数指定用于输入的文件名；第二个构造函数用一个 File 对象参数指定用于输入的文件名，本章 4.3.3 将介绍 File 类；第三个构造函数以一个 FileDescriptor 为参数。

FileInputStream 类与 InputStream 很类似，只是它是针对文件处理的。下面是 ReadFile.java 的源代码，它使用 FileInputStream 类从一个文本文件读数据。

```

import java.io.*;
class ReadFile {
    public static void main(String args[ ]) {
        byte buf[ ] = new byte[64];
        try {
            FileInputStream in = new FileInputStream("Test.txt");
            in.read(buf, 0, 64);
        }
        catch (Exception e) {
            System.out.println("Error: " + e.toString());
        }
        String s = new String(buf, 0);
        System.out.println(s);
    }
}

```

5. StringBufferInputStream 类

StringBufferInputStream 类使用一个字符串作为输入缓冲源，它实现了与 InputStream 相同的成员函数。StringBufferInputStream 类只有如下一个构造函数：

- StringBufferInputStream(String s)

虽然 StringBufferInputStream 类没有定义其它的成员函数，但它提供了一些新的成员变量：

- protected String buffer
- protected int count
- protected int pos

buffer 是字符串数据存储的缓冲区；count 指定了在缓冲区中使用的字符数；pos 用来表示数据在缓冲区中的当前位置。

下面是 ReadString.java 的源代码，它使用 StringBufferInputStream 从一个字符串读取数据。

```
import java.io.*;
class ReadString {
    public static void main(String args[ ]) {
        // Get a string of input from the user
        byte buf1[ ] = new byte[64];
        try {
            System.in.read(buf1, 0, 64);
        }
        catch (Exception e) {
            System.out.println("Error: " + e.toString());
        }
        String s1 = new String(buf1, 0);

        // Read the string as a string buffer and output it
        StringBufferInputStream in = new StringBufferInputStream(s1);
        byte buf2[ ] = new byte[64];
        try {
            in.read(buf2, 0, 64);
        }
        catch (Exception e) {
            System.out.println("Error: " + e.toString());
        }
        String s2 = new String(buf2, 0);
        System.out.println(s2);
    }
}
```

```
}  
}
```

4.3.2 输出流

与输入流相对应，输出流是处理对输出设备写数据的，你可以使用输出流将数据输出到各种不同的设备，如屏幕等。Java 主要的输出流类如下：

- OutputStream
- PrintStream
- BufferedOutputStream
- DataOutputStream
- FileOutputStream

Java 输出流提供了输出数据的各种方法。OutputStream 类定义了输出流的核心行为；PrintStream 类适合于输出文本数据，如送到标准输出流（stdout）的数据；BufferedOutputStream 类是对 OutputStream 类的扩充，它对缓冲输出提供支持；DataOutputStream 类用于输出原始数据类型，如整数或浮点数；FileOutputStream 类对数据输出到文件提供支持。

1. OutputStream 类

OutputStream 类是其它输出流类的基类，它定义了把数据流写到输出设备的基本协议。使用时，通常生成一个 OutputStream 对象并调用相应的成员函数来告诉它你想输出信息。OutputStream 类使用了一个与 InputStream 类似的技术，在等待当前输出被处理时将阻塞，直到数据写入输出设备。阻塞时，OutputStream 类将不允许其它数据输出。

OutputStream 类实现了以下成员函数：

- public abstract void write(int b)
- public void write(byte b[])
- public void write(byte b[], int off, int len)

OutputStream 为以不同方式写数据定义了三个 write 函数。第一个函数把由参数 b 指定的单个字节写到输出流；第二个函数把一个字节数组的数据写到输出流；第三个函数与第二个函数类似，只是它有三个参数，其中参数 off 指定从字节数组输出数据的起始位置，参数 len 指定输出数据的字节数。

- public void flush()

该函数用于刷新输出流，调用它将使 OutputStream 对象输出所有“挂起（pending）”的数据。

- public void close()

关闭输出流并释放所有与它相关的资源。与 InputStream 对象类似，通常不必在 OutputStream 对象中调用 close 函数，当 OutputStream 对象“破坏”时，流将自动关闭。

2. PrintStream 类

PrintStream 类是 OutputStream 的派生类，它主要为打印输出文本数据设计的。PrintStream 有如下两个不同的构造函数：

- PrintStream(OutputStream out)
- PrintStream(OutputStream out, boolean autoflush)

这两个构造函数都以 OutputStream 对象作为第一个参数，不同的是如何处理换行字符。第一个构造函数流输出是由对象内部决定的；第二个构造函数可以由参数 autoflush 来指定是否每当遇到换行字符即输出流。

PrintStream 类实现了许多成员函数，如下：

- public boolean checkError()
- 该函数检查流输出时是否会发生错误，如果出现错误将返回真，否则返回假。
- public void print(Object obj)
- public synchronized void print(String s)
- public synchronized void print(char s[])
- public void print(char c)
- public void print(int i)
- public void print(long l)
- public void print(float f)
- public void print(double d)
- public void print(boolean b)

PrintStream 提供了处理各种打印需求的 print 函数，以 Object 为参数的 print 函数输出对象调用其成员函数 toString 的结果，其它的 print 函数以不同数据类型为参数来指定所打印的数据。

- public void println()
- public synchronized void println(Object obj)
- public synchronized void println(String s)
- public synchronized void println(char s[])
- public synchronized void println(char c)
- public synchronized void println(int i)
- public synchronized void println(float f)
- public synchronized void println(double d)
- public synchronized void println(boolean b)

由 PrintStream 实现的 println 函数与 print 非常类似，唯一的不同是 println 函数在打印的数据后接着打印一个换行字符。无参数的 println 函数只是打印一个换行字符。

System 类有一个成员变量 out，是 PrintStream 类的一个实例，用来表示标准输出流，它在将文本输出到监视器屏幕时非常有用，这一点你可能从前面的示例中已注意到。

3. BufferedOutputStream 类

BufferedOutputStream 类与 BufferedInputStream 类非常相似，它为输出流提供了缓冲，这就使得你在每次写数据时不必写到输出设备。BufferedOutputStream 类提供了一个写数据的缓冲区，当缓冲区满或刷新时才将数据写到输出设备。相对来说，这种输出方式更有效，因为大多数的数据传送发生在内存中，而且当将数据输出到设备时可以一次完成。

BufferedOutputStream 实现的成员函数与 OutputStream 相同，但它提供了如下两个构造函数：

- BufferedOutputStream(OutputStream out)
- BufferedOutputStream(OutputStream out, int size)

两个构造函数都以一个 OutputStream 对象作为第一个参数，不同的是两者用于存储输出数据的内部缓冲区的大小，前者使用了一个缺省的缓冲区大小，后者则可以由参数 size 来指定缓冲区大小。BufferedOutputStream 类中的缓冲区是由以下两个成员变量来管理的：

- protected byte buf[]
- protected int count

字节数组 buf 是输出数据存放的实际缓冲区；count 标记缓冲区中的字节数。这两个成员变量已足够表示输出流缓冲区的状态。

下面是 WriteStuff.java 的源代码，它使用 BufferedOutputStream 对象来输出一个字节数组的文本数据。程序首先生成一个文本字符串对象和一个字节数组，并通过 String 的成员函数 getBytes() 将字符串中的数据拷贝到字节数组。然后以 System.out 作为参数生成一个 BufferedOutputStream 对象，并使用其成员函数 write() 将字节数组写到输出缓冲区。由于输出流是缓冲的，需要调用成员函数 flush() 才会实际输出数据。

```
import java.io.*;
class WriteStuff {
    public static void main(String args[ ]) {
        // Copy the string into a byte array
        String s = new String("Dance, spider!\n");
        byte[ ] buf = new byte[64];
        s.getBytes(0, s.length(), buf, 0);

        // Output the byte array (buffered)
        BufferedOutputStream out = new BufferedOutputStream(System.out);
        try {
            out.write(buf, 0, 64);
            out.flush();
        }
        catch (Exception e) {
```

```

        System.out.println("Error: " + e.toString());
    }
}

```

4. **FileOutputStream** 类

FileOutputStream 类用于执行简单的文件输出操作，对于复杂的文件输出可以使用 **RandomAccessFile** 类，这将在 4.3.3 中介绍。 **FileOutputStream** 类提供了以下三个构造函数：

- **FileOutputStream**(String name)
- **FileOutputStream**(File file)
- **FileOutputStream**(FileDescriptor fdObj)

第一个构造函数以一个字符串参数来指定用于输入的文件名；第二个构造函数以文件对象为参数来指定输入文件；第三个构造函数以 **FileDescriptor** 对象为参数。

下面是程序 **WriteFile.java** 的源代码，它使用 **FileOutputStream** 类将用户输入写到一个文本文件中。

```

import java.io.*;
class WriteFile {
    public static void main(String args[] ) {
        // Read the user input
        byte buf[] = new byte[64];
        try {
            System.in.read(buf, 0, 64);
        }
        catch (Exception e) {
            System.out.println("Error: " + e.toString());
        }

        //Output the data to a file
        try {
            FileOutputStream out = new FileOutputStream("Output.txt");
            out.write(buf);
        }
        catch (Exception e) {
            System.out.println("Error: " + e.toString());
        }
    }
}

```


}

4.3.3 文件类

如果 `FileInputStream` 和 `FileOutputStream` 两个类不能满足你对文件处理的需要,那么不要太失望, Java 提供了另外两个对文件进行操作的类,相信一定能满足你的要求,这两个类是 `File` 和 `RandomAccessFile`。`File` 类以操作系统目录结构为模型,使你能访问一个文件的信息,包括文件属性、文件所在路径等。另一方面, `RandomAccessFile` 类则提供了从一个文件读写数据的各种方法。

1. `File` 类

`File` 类可以使用以下三个构造函数初始化:

- `File(String path)`
- `File(String path, String name)`
- `File(File dir, String name)`

第一个构造函数以文件的完全路径名为参数;第二个构造函数有两个参数,参数 `path` 指定了文件所在路径,参数 `name` 指定了文件名。第三个构造函数与第二个类似,只是它用一个 `File` 对象来指定文件所在路径。

`File` 类中最重要的成员函数如下:

- `public String getName()`
获取文件名并以字符串返回。
- `public String getPath()`
返回文件路径(可能为相对路径)。
- `public String getAbsolutePath()`
返回文件的绝对路径。
- `public String getParent()`
返回文件的上一级目录,如果未找到则返回 `null`。
- `public boolean exists()`
如果文件存在则返回真,否则返回假。
- `public boolean canWrite()`
如果文件可写则返回真,否则返回假。
- `public boolean canRead()`
如果文件可读则返回真,否则返回假。
- `public boolean isFile()`
如果文件有效则返回真,否则返回假。
- `public boolean isDirectory()`
如果目录有效则返回真,否则返回假。
- `public boolean isAbsolute()`
如果文件名是绝对路径则返回真,否则返回假。

■ `public long lastModified()`

返回一个长整数值, 表示文件上次修改的时间。

■ `public long length()`

返回文件字节数长度。

■ `public boolean mkdir()`

根据当前的路径信息生成一个目录。

■ `public boolean mkdirs()`

根据当前的路径信息生成整个目录结构。

■ `public boolean renameTo(File dest)`

将文件名改为 `File` 对象 `dest` 指定的名字。

■ `public boolean delete()`

删除一个文件。

■ `public String[] list()`

列出一个目录中的所有文件。

■ `public String[] list(FilenameFilter filter)`

列出一个目录中符合文件过滤器的所有文件。

下面是程序 `FileInfo.java` 的源代码, 它使用 `File` 对象来获取当前目录下一个文件的信息。首先程序提示用户输入一个文件名, 输入结果存放在一个 `String` 对象中, 该 `String` 对象然后被用来作为 `File` 对象构造函数的参数。调用 `exists` 函数确定文件是否存在, 如果存在, 就通过 `File` 对象的各种成员函数来获得文件的信息, 并在屏幕上输出结果。

```
import java.io.*;

class FileInfo {
    public static void main(String args[]) {
        System.out.println("Enter file name: ");
        char c;
        StringBuffer buf = new StringBuffer();
        try {
            while ((c = (char)System.in.read()) != '\n')
                buf.append(c);
        }
        catch (Exception e) {
            System.out.println("Error: +e.toString());
        }
        File file = new File(buf.toString());
        if (file.exists()) {
            System.out.println("File Name: +file.getName());
            System.out.println("Path: +file.getPath());
            System.out.println("Abs. Path: +file.getAbsolutePath());
```

```

        System.out.println("Writable: +file.canWrite( ));
        System.out.println("Readable: +file.canRead( ));
        System.out.println("Length: +(file.length( ) / 1024)+"KB");
    }
    else
        System.out.println("Sorry, file not found.");
    }
}

```

2. RandomAccessFile 类

RandomAccessFile 类实现了读写文件的多种方法，虽然使用 FileInputStream 和 FileOutputStream 类也可以访问文件，但 RandomAccessFile 类提供了更多的功能和选项。FileInputStream 和 FileOutputStream 只能顺序地从头到尾访问文件，而 RandomAccessFile 类则可以明确指定从文件的某个位置获取数据。

RandomAccessFile 类有以下两个构造函数：

- public RandomAccessFile(String name, String mode)
- public RandomAccessFile(File file, String mode)

第一个构造函数由字符串参数 name 指定访问的文件名，由字符串参数 mode 指定访问方式，访问方式可以是“r”或“rw”，分别表示读和读/写。第二个构造函数以 File 对象作为第一个参数，用来指定访问的文件，第二个参数同前面一样，也是用来指定访问方式。

当你生成 RandomAccessFile 的一个实例后，就可以调用其成员函数来访问文件。RandomAccessFile 允许从文件任何位置读写数据，而不必按照流的顺序来处理。这是通过文件指针来实现的，当第一次打开文件时，文件指针指向文件头。在移动文件指针前，你需要知道文件有多大。下面的成员函数能返回文件的长度：

- public long length()

当你知道文件指针可移动的范围后，就可以使用成员函数 seek() 来移动文件指针。如果你想知道指针的当前位置，可以使用成员函数 getFilePointer()。

- public void seek(long pos)

将文件指针移动到指定的位置。

- public long getFilePointer()

返回文件指针的位置。

重要的是，当你处理完一个文件后，需要使用成员函数 close() 来关闭它。

- public void close()

打开文件并确定了文件指针的位置后，就可以从文件读取数据。RandomAccessFile 提供了从文件读取不同类型数据的多个成员函数。

- public int read()

读一个字节的数据，指针将向前移 8 位。

■ `public int read(byte b[])`

读文件数据并放到字节数组 `b` 中, 指针将向前移实际所读的字节数。

■ `public int read(byte b[], int off, int len)`

从文件读指定长度的数据, 并放到字节数组 `b` 中以 `off` 开始的位置, 指针将向前移 `len` 个字节。

■ `public final String readLine()`

读以 `'\n'` 或 EOF 结束的一行字符。

■ `public final String readUTF()`

从 UTF 格式的文件中读一个字符串。

下面的成员函数读取指定类型的数据, 并将指针向前移相应的字节数。

■ `public final boolean readBoolean()`

■ `public final byte readByte()`

■ `public final short readShort()`

■ `public final char readChar()`

■ `public final int readInt()`

■ `public final long readLong()`

■ `public final void readFloat()`

■ `public final void readDouble()`

下面是程序 `FilePrint.java` 的源代码, 它将一个文件的内容显示在屏幕上。程序首先根据用户输入的文件名生成一个 `File` 对象, 如果文件存在, 则将它作为 `RandomAccessFile` 类构造函数的参数, 然后每次从该文件读取一行并在屏幕上显示。

```
import java.io.*;

class FilePrint {
    public static void main(String args[ ]) {
        System.out.println("Enter file name: ");
        char c;
        StringBuffer buf = new StringBuffer();

        try {
            while ((c = (char)System.in.read()) != '\n')
                buf.append(c);
            File input = new File(buf.toString());
            if(input.exists()) {
                RandomAccessFile file = new RandomAccessFile(input, "rw");
                while (file.getFilePointer() < file.length())
                    System.out.println(file.readLine());
                file.close();
            }
        }
    }
}
```

```

        else {
            System.out.println("Can't read file");
        }
    }
    catch (Exception e) {
        System.out.println("Error: " + e.toString());
    }
}
}
}

```

除了读文件,也可以写文件,基本上每个 read 成员函数都对应着一个 write 成员函数。应该注意的是当你写文件时,并不是插入数据,而是用新数据覆盖文件中已有的信息,因此要清楚写到文件中的信息。

下面是一些对文件进行写操作的成员函数:

- public void write()
- public void write(byte b[])
- public void write(byte b[], int off, int len)
- public final void writeLine(String line)
- public final void writeUTF(String utf)
- public final void writeBoolean(boolean b)
- public final void writeByte(byte b)
- public final void writeShort(short s)
- public final void writeChar(char c)
- public final void writeInt(int i)
- public final void writeLong(long l)
- public final void writeFloat(float f)
- public final void writeDouble(double d)

可以说, RandomAccessFile 基本上提供了在 Java 环境中读写文件的所有功能。但有一点应该注意的是, UNIX 和 DOS 系统在文本文件格式方面有很大差别,通常, UNIX 系统文本文件每行的末尾只有一个换行字符,而 DOS 文本文件有会车和换行两个字符。

下面是程序 FileWrite.java 的源代码,它将键盘输入的多行字符写入 DOS 类型文件。程序首先根据用户输入的文件名生成一个 RandomAccessFile 类对象,这时文件长度为 0。然后等待用户输入,用户每输入一行字符则在该文件中写入一行字符,直到用户敲入“Ctrl+Z”。

```

import java.io.*;
class FileWrite {
    public static void main(String args[ ]){
        try {

```

```

        System.out.println("Enter FileName: ");
        char c;
        int in = 0;
        StringBuffer fileBuf = new StringBuffer();
        while((c=(char)System.in.read()) != '\n')
            fileBuf.append(c);

        RandomAccessFile file = new RandomAccessFile(fileBuf.toString(), "rw");
        System.out.println("File length: " + file.length());
        System.out.println("Pointer pos: " + file.getFilePointer());
        StringBuffer line = new StringBuffer();
        while(in != -1) {
            line.setLength(0);
            while((in=System.in.read()) != '\n')
                line.append((char)in);
            line.append("\r\n");
            file.writeBytes(line.toString());
        }
        file.close();
    }
    catch (Exception e) {
        System.out.println("Error: " + e.toString());
    }
}
}

```

4.4 实用程序类库

实用程序类库提供了一组实现各种标准的编程数据结构的类，这些类是其它的 Java 软件包和应用构造更复杂的数据结构的基础。除非特别声明，本章讨论的接口和类都是由 `java.lang.Object` 派生的。

4.4.1 接口 (Interfaces)

实用程序类库包括两个接口即 `Enumeration` 和 `Observer`，你可在自己设计的类当中使用它们。所谓接口是指声明实现它的类中必须提供的一组函数，这就为实现该接口的所有类的使用提供了一致的方法。

1. Enumeration 接口

Enumeration 接口指定了用于表枚举的一组函数，实现该接口的对象只能对一个表反复枚举一次，因为 Enumeration 对象用完一次以后就不存在了。例如，一个 Enumeration 对象可用来打印一个 Vector 对象的所有元素，如下：

```
for (Enumeration e=v.elements(); e.hasMoreElement(); )  
    System.out.print(e.nextElement() + " ");
```

Enumeration 接口只包括两个函数：hasMoreElement() 和 nextElement()。如果表中还有元素可枚举，hasMoreElement() 则返回真；而 nextElement() 将返回一个对象，表示所枚举对象的下一个元素。Enumeration 接口实现的细节以及数据如何表示将在指定类的实现中讨论。

2. Observer 接口

实现 Observer 接口的类允许其对象监视该类可观察的 (Observable) 其它对象，当可观察的对象发生变化时将通知该接口。Observer 接口只包括一个函数：update(Observable, Object)，被观察的对象发生变化时将调用该函数来通知观察者。该函数的第一个参数使观察者可对被观察的对象操作，第二个参数用来在两者间传递信息。

4.4.2 Date 类

Date 类使用与系统无关的方式表示日期和时间，它可以由如下几个构造函数初始化：

- public Date()

使用当前的日期和时间构造一个 Date 实例。

- public Date(int year, int month, int date)

- public Date(int year, int month, int date, int hrs, int min)

- public Date(int year, int month, int date, int hrs, int min, int sec)

以指定的年（公元年 ~ 1900）、月（0 ~ 11）、日（1 ~ 31）、时（0 ~ 23）、分（0 ~ 59）、秒（0 ~ 59）构造一个 Date 实例。

- public Date(String s)

以一定格式的字符串为参数构造一个 Date 实例。

例如，下面的程序可打印出当前或某个指定日期的时间表示。

```
import java.util.Date;  
public class Date1 {  
    public static void main(String args[]) {  
        Date today = new Date();  
        System.out.println("Today is " + today.toLocaleString() +  
            " (" + today.toGMTString() + ")");  
  
        Date birthday = new Date(89, 10, 14, 8, 30, 00);
```

```

System.out.println("My birthday is " + birthday.toString() +
    " ( " + birthday.toGMTString() + " ) " );

Date anniversary = new Date("Jun 21, 1986");
System.out.println("My anniversary is " + anniversary +
    " ( " + anniversary.toGMTString() + " ) " );
}
}

```

该程序的输出如下:

```

Today is 11/13/96 20:26:59 ( 13 Nov 1996 12:26:59 GMT )
My birthday is Tue Nov 14 08:30:00 1989 ( 14 Nov 1989 00:30:00 GMT )
My anniversary is Sat Jun 21 00:00:00 1986 ( 20 Jun 1986 15:00:00 GMT )

```

有关 Date 类的成员函数可参考 Java API 文档。

4.4.3 Random 类

Random 类用于产生一个伪随机数序列，它有如下两个构造函数:

■ public Random()

以当前的时间作为种子值，生成一个新的随机数生成器。

■ public Random(long seed)

以参数 seed 作为种子值，生成一个新的随机数生成器。这将使应用程序每次运行时都产生一个相同的伪随机数序列。当然，也可以在以后使用成员函数 setseed() 来重新设置种子值。

为了产生不同类型的伪随机数序列，可使用相应的成员函数： nextInt()、nextLong()、nextFloat()、nextDouble() 或 nextGaussian()。例如，下面的程序将打印出 5 个不同类型的伪随机数系列。

```

import java.lang.Math;
import java.util.Date;
import java.util.Random;

class RandomTest {
    public static void main(String args[])
        throws java.io.IOException
    {
        int count = 6;
        Random ranGen = new Random();
    }
}

```



```

System.out.println("Uniform Random Integers");
for (int i=0; i<count; i++)
System.out.print(ranGen.nextInt( ) + " ");
System.out.println("\n");

System.out.println("Uniform Random Floats");
for (int i=0; i<count; i++)
System.out.print(ranGen.nextFloat( ) + " ");
System.out.println("\n");

System.out.println("Gaussian Random Floats");
for (int i=0; i<count; i++)
System.out.print(ranGen.nextGaussian( ) + " ");
System.out.println("\n");

System.out.println("Uniform Random Integers [1,6]");
for (int i=0; i<count; i++)
System.out.print((Math.abs(ranGen.nextInt( ))%6+1) + " ");
System.out.println("\n");
}
}

```

该程序运行后可产生类似如下的输出:

```

Uniform Random Integers
1375730445 -509513177 1945482460 -43086621 -1001122136 1384423342

```

```

Uniform Random Floats
0.393701 0.637507 0.985329 0.00759692 0.147329 0.720654

```

```

Gaussian Random Floats
1.29508 0.965581 0.780468 -0.903144 -0.429269 0.13693

```

```

Uniform Random Integers [1,6]
3 6 3 1 4 6

```

4.4.4 Stack 类

Stack 类实现了一个后进先出 (LIFO) 的栈。栈的原理大家都非常熟悉, 它只允许数据的操作在栈顶进行。要生成 Stack 类的一个实例, 只有如下一个构造函数:

- `public Stack()`

Stack 类提供了如下的成员函数:

- `public Object push(Object item)`

- `public Object pop()`

- `public Object peek()`

- `public boolean empty()`

- `public int search(Object o)`

函数 `push()` 和 `pop()` 实现了对象入栈和出栈操作; 如果只是想查看栈顶元素, 可以使用函数 `peek()`; 而函数 `empty` 则用于检查栈是否空, 如果栈空则返回真, 否则返回假; 最后, 你也可以在栈中查找某个对象, 函数 `search()` 返回栈中第一个与所查找对象匹配的元素与栈顶的距离, 如果没有找到则返回-1。

下面是程序 `StackTest.java` 的源代码, 它将用户输入的一行字符串存入一个栈中, 并以相反的顺序打印出来。其中用到了 `StringTokenizer` 类, 它可以根据指定的分隔符 (这里是空格符) 把一个字符串分成多个标记。

```
import java.io.DataInputStream;
import java.util.Stack;
import java.util.StringTokenizer;

class StackTest {
    public static void main(String args[ ])
        throws java.io.IOException
    {
        DataInputStream dis = new DataInputStream(System.in);
        System.out.println("Enter a sentence: ");
        String s = dis.readLine();
        StringTokenizer st = new StringTokenizer(s);
        Stack stack = new Stack();
        while (st.hasMoreTokens())
            stack.push(st.nextToken());
        while (!stack.empty())
            System.out.print((String)stack.pop() + " ");
        System.out.println();
    }
}
```

我们运行该程序并输入一行字符，可得到如下的输出：

Enter a sentence:

The quick brown fox jumps over the lazy dog
dog lazy the over jumps fox brown quick The

4.4.5 Hashtable 类

Hashtable 类实现了哈希表存储机制，哈希表采用（关键字，数值）对作为存储方式，以关键字为索引可迅速查找到信息，这里关键字和数值可以是任何对象类型。

Hashtable 类提供了以下的构造函数和成员函数：

■ `public Hashtable()`

使用缺省的容量构造一个空的哈希表。

■ `public Hashtable(int initialCapacity)`

以指定的容量构造一个空的哈希表。

■ `public Hashtable(int initialCapacity, float loadFactor)`

使用指定的容量和加载因子构造一个空的哈希表。其中加载因子是一个 0 - 1 的数，当哈希表容量占用百分比达到该值时，将重新生成一个更大的哈希表。在前两个构造函数中，由于未指定加载因子，当哈希表满后才重新生成一个更大的哈希表。

■ `public int size()`

返回哈希表中的元素个数。

■ `public boolean isEmpty()`

如果哈希表空则返回真，否则返回假。

■ `public synchronized Enumeration keys()`

返回哈希表中关键字的枚举。

■ `public synchronized Enumeration elements()`

返回哈希表中元素值的枚举。

■ `public synchronized boolean contains(Object value)`

判断某对象是否是哈希表中的元素。

■ `public synchronized boolean containsKey(Object key)`

判断某对象是否是哈希表的关键字。

■ `public synchronized Object get(Object key)` 在哈希表中获取与指定关键字相对应的元素。

■ `protected void rehash()`

重新生成一个更大的哈希表。当哈希表的大小达到临界值时将自动调用该成员函数。

■ `public synchronized Object put(Object key, Object value)`

将一对关键字和元素值存入哈希表。

■ `public synchronized Object remove(Object key)`

删除与关键字相对应的元素。

■ `public synchronized void clear()`

删除哈希表中的所有元素。

下面是程序 `HashTest.java` 的源代码，它生成了一个哈希表并使用成员函数 `put()` 在表中存放了 10 对关键字和数值，然后使用成员函数 `get()` 获取与用户输入的关键字相对应的数值。

```
import java.io.DataInputStream;
import java.lang.Integer;
import java.lang.Math;
import java.util.Random;
import java.util.Hashtable;

class HashTest {
    public static void main(String args[ ])
        throws java.io.IOException
    {
        DataInputStream dis = new DataInputStream(System.in);
        int numElements = 10;
        String keys[ ] = {"Red", "Green", "Blue", "Cyan", "Magenta",
            "Yellow", "Black", "Orange", "Purple", "White"};
        Hashtable ht;
        Random randGen = new Random( );

        ht = new Hashtable(numElements*2);
        for (int i=0; i<numElements; i++)
            ht.put(keys[i], new Integer(Math.abs(randGen.nextInt( )) % numElements));
        System.out.println(ht.toString( ));

        String KeyValue;
        System.out.println("Which key to find? ");
        KeyValue = dis.readLine( );
        Integer value = (Integer)ht.get(KeyValue);
        if (value != null) System.out.println(KeyValue + " = " + value);
    }
}
```

该程序运行后的输出类似如下：

{Cyan=5, White=5, Magenta=4, Red=7, Black=3, Green=6, Purple=4, Orange=4,

```
Yellow=6,  
    Blue=2}  
Which key to find?  
Red  
Red = 7
```

4.5 小 结

作为面向对象的编程语言，Java 提供了大量的、可重用的标准类。本章描述了 Java 类库的总体结构，并较为详细地阐述了 Java 基础类库，包括 Java 语言类库、Java 输入/输出类库和 Java 实用类库的使用。我们的目标不是为读者提供 Java 类库的全面参考，而是有重点地分析一些常用类，使读者能做到举一反三。本章后面几章将着重阐述 Java 应用类库的使用。

第五章 编写图形用户界面

图形化用户界面 (Graphic User Interface ---- GUI) 是目前流行的人机交互手段之一。Java 具有丰富的用户界面元素类支持, 可以方便地实现用户友好的界面风格。下面, 我们将介绍 Java 用户界面的基本概念和各种界面元素的编程, 包括菜单、按钮、滚动条、文本域、文本输入框、列表等。在介绍完所有的界面元素后, 我们还列举了两个典型应用: 作图程序和记事本程序。最后介绍界面元素包容器、字体和颜色类的应用编程。

5.1 概 述

Java 的 GUI 编程是在抽象窗口工具箱 (Abstract Window Toolkit——AWT) 的基础上实现的。java.awt 程序包是 AWT 的工具类库, 其中包括了丰富的图形、用户界面元素构件和布局管理的支持。

其中可以简单地分为下列的几类:

■ 用户界面元素构件 (GUI Components)。如, 菜单 (Menu)、按钮 (Button)、列表 (List)、文本输入框 (Text field) 等。图 5.1 示例了所有的界面元素。

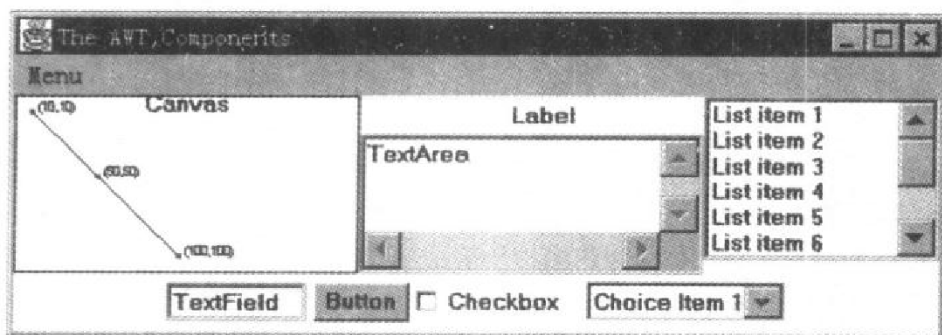
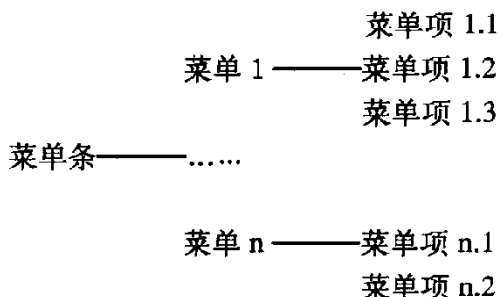


图 5.1 Java 中的 GUI 构件示例

- 布局管理。主要的任务是控制以上界面元素的位置、排列方式等。
- 基本图形。这些类包括颜色、字体、图形、图象等。
- 事件类。描述了用户交互时的事件, 在编程时要对其中的一些事件 (如, 鼠标点击, 键盘输入等) 作出反应。与其他的图形化用户界面一样, Java 的界面元素是基于事件触发的。每一个界面元素类对多数事件有一些缺省的处理, 但引伸类要处理感兴趣的事件。

5.2 菜单设计

菜单是图形化用户界面的重要部分。当你设计独立应用程序时，Java 提供了一种直接的建造和使用的方法。一个完整的菜单界面需要有菜单条（Menu Bar），菜单（Menu），菜单项（Menu Item）三个部分。这三个部分有包含关系，即，菜单条包含若干个菜单，每个菜单又包含若干个菜单项，如下所示：



与其他的构件一样，Java 的菜单界面的三个部分都分别是一个对象，以类的方式提供出来。在使用时，用 new 来生成对象，新对象用 add() 操作函数加入到上一级的对象中。如，在生成菜单对象和菜单项对象之后，就可以用菜单对象的 add() 操作函数把菜单项加入到菜单中。

在菜单类中有一个操作函数是 addSeparator()，它产生的效果是加入分割线，它把菜单项分成若干个子部分，起到分类的作用。

有一类特殊的菜单项类是 CheckboxMenuItem，它与普通的菜单条对象不同的地方是，它在点中后，会有记忆，在菜单条的一侧显示标记。

SetMenuBar() 语句是把菜单条安装到 frame 对象中。如果要安装不同的两套菜单的话，则设计不同的菜单条，在适当的时候，可以调用 SetMenuBar() 转换菜单条。

当用户点中菜单项后，系统会产生一个相应的事件。事件对象会传送到 action() 操作函数中进行处理，以确定用户点中了哪一个菜单项。下面的例子简单地显示了 action 函数的用法。它首先判断该事件是不是菜单类的事件，如果是则检查参数 arg 中的字符串，如果是等于 "About"，就执行相应的命令，直到把所有的菜单项检查一遍。

```
import java.awt.*;

public class MenuWindow extends Frame {

    public MenuWindow() {

        // 创建菜单条对象
        MenuBar mb = new MenuBar();
```

```

Menu m = new Menu("Game");
m.add(new MenuItem("Start a connection..."));
m.add(new MenuItem("Chang the listening port..."));
m.add(new CheckboxMenuItem("Leave"));
m.addSeparator();

m.add(new MenuItem("Option..."));
m.addSeparator();
m.add(new MenuItem("Exit"));
Menu Help = new Menu("Help");
Help.add(new MenuItem("About"));
Help.add(new MenuItem("Rule"));
mb.add(m);
mb.add(Help);
//安装菜单条
setMenuBar(mb);
}

public boolean action(Event event, Object arg) {
    if (event.target instanceof MenuItem) {
        if (((String)arg).equals("About")) {
            //当选中 "About" 菜单项时, 执行适当的操作
        }
        if (((String)arg).equals("Exit")) {
            System.exit(0);
        }
        if ( ( (String)arg).equals("Start a connection...")) {
            // 当选中 " Start a connection..." 菜单项时, 执行适当的操作
        }
        if ( ( (String)arg).equals("")) {
            // 当选中 " Chang the listening port..." 菜单项时, 执行适当的操作
        }
    }
    return true;
}

public boolean handleEvent(Event event) {
    if (event.id == Event.WINDOW_DESTROY) {
        System.exit(0);
    }
}

```



```

    }
    return super.handleEvent(event);
}

public Dimension minimumSize() {
    return new Dimension(500, 400);
}

public Dimension preferredSize() {
    return minimumSize();
}

public Dimension maximumSize() {
    return minimumSize();
}

public static void main(String args[]) {
    MenuWindow window = new MenuWindow();
    window.setTitle("Menu Window "); //设置窗口的标题
window.resize(400, 300);
    window.setResizable(true); //设置窗口为大小可调的
    window.show(); //显示窗口
}
}

```

以上的程序在编译后将产生 MenuWindow.class 文件，用 Java 命令行运行该代码后，会产生如图 5.2 所示的用户界面：

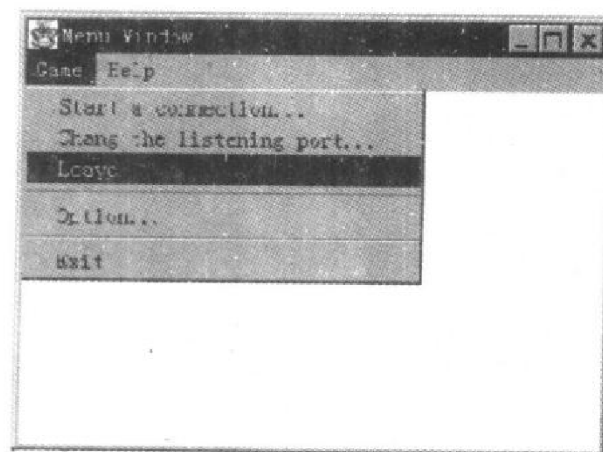


图 5.2 菜单界面设计例程的屏幕显示

5.3 其他界面元素设计

由于界面元素很多，我们在这里不再把其中的每一个元素都分开讲解，而是通过下面的一个例子来示例这些元素的用法。对每一个界面元素，我们都作了进一步的继承，引伸出自己的新类，如，类 `MyTextField` 就是 `TextField` 类的引伸类。在每一个引伸出的新类中，都有自己的事件处理函数。一般的做法是把响应在文本显示区（`TextArea textArea`）中显示出来。下面是源程序代码。图 5.3 为本例程的屏幕输出。

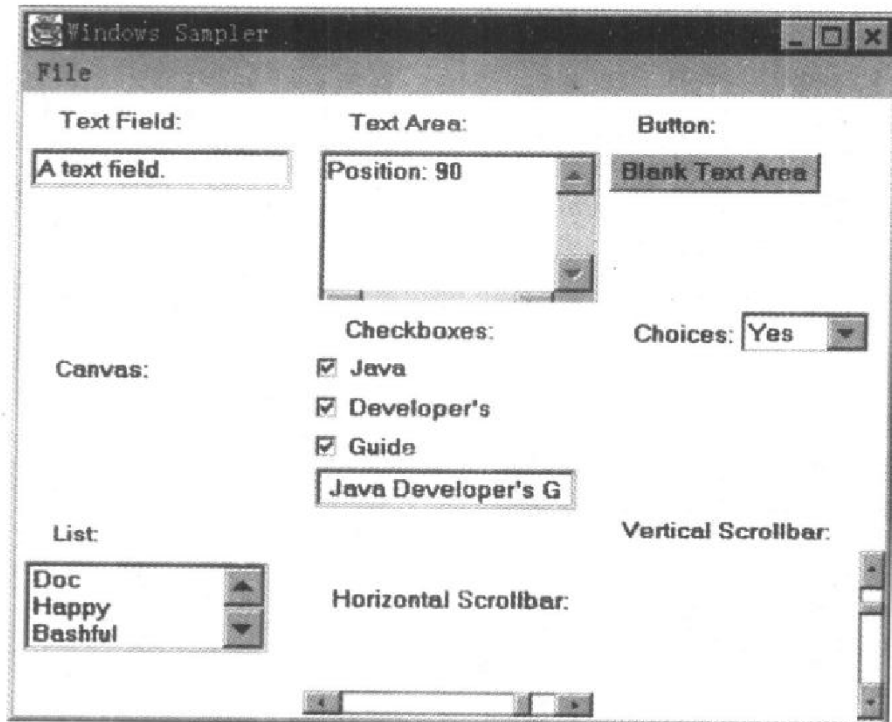


图 5.3 例程屏幕输出

```
import java.awt.*;
import java.lang.System;

public class SamplerApp extends Frame {
    TextArea textArea;
    public static void main(String args[]){
        SamplerApp app = new SamplerApp();
    }
    public SamplerApp() {
        super("Windows Sampler");
        setup();
        pack();
    }
}
```

```

        resize(400,400); // 调整窗口大小
        show();
    }
    // 初始化系统中元素，如菜单和面板
    void setup() {
        setupMenuBar();
        setupPanels();
    }
    // 设置菜单条
    void setupMenuBar() {
        MenuBar menuBar = new MenuBar();
        Menu fileMenu = new Menu("File");
        fileMenu.add(new MenuItem("Exit"));
        menuBar.add(fileMenu);
        setMenuBar(menuBar);
    }
    // 设置面板中各元素的位置，参照程序清单后的图形以帮助理解
    void setupPanels() {
        Panel mainPanel = new Panel();
        mainPanel.setLayout(new GridLayout(3,3));
        Panel panels[ ][ ] = new Panel[3][3];
        for(int i=0; i<3; ++i) {
            for(int j=0; j<3; ++j) {
                panels[j][i] = new Panel();
                panels[j][i].setLayout(new FlowLayout(FlowLayout.LEFT));
            }
        }
        panels[0][0].add(new Label("Text Field:"));
        panels[0][0].add(new MyTextField("A text field.",15));
        panels[1][0].add(new Label("Text Area:"));
        textArea = new TextArea("A text area.", 5,15);
        panels[1][0].add(textArea);
        panels[2][0].add(new Label("Button:"));
        panels[2][0].add(new MyButton("Blank Text Area", textArea));
        panels[0][1].add(new Label("Canvas:"));
        panels[0][1].add(new MyCanvas());
        String checkboxStrings[ ] = {"Checkboxes: ", "Java", "Developer's", "Guide"};
        panels[1][1].add(new MyCheckboxGroup(checkboxStrings));
        panels[2][1].add(new Label("Choices:"));
    }

```

```

String choiceStrings[ ] = {"Yes", "No", "Maybe"};
panels[2][1].add(new MyChoice(choiceStrings, textArea));
panels[0][2].add(new Label("List: "));
String listStrings[ ] = {"Sleepy", "Sneezy", "Grumpy", "Dopey", "Doc",
    "Happy", "Bashful"};
panels[0][2].add(new MyList(listStrings, textArea));
panels[1][2].setLayout(new BorderLayout( ));
panels[1][2].add("Center", new Label("Horizontal Scrollbar: "));
panels[1][2].add("South", new MyScrollbar(Scrollbar.HORIZONTAL, 50, 10, 0,
    100, textArea));
panels[2][2].setLayout(new BorderLayout( ));
panels[2][2].add("North", new Label("Vertical Scrollbar: "));
panels[2][2].add("East", new MyScrollbar(Scrollbar.VERTICAL, 50, 10, 0,
    1000, textArea));
for(int i=0; i<3; ++i)
    for(int j=0; j<3; ++j)
        mainPanel.add(panels[j][i]);
add("Center", mainPanel);
}
// 事件处理函数
public boolean handleEvent(Event event) {
    if(event.id==Event.WINDOW_DESTROY) {
        System.exit(0);
        return true;
    } else if(event.id==Event.ACTION_EVENT) {
        if(event.target instanceof MenuItem) {
            if("Exit".equals(event.arg)) {
                System.exit(0);
                return true;
            }
        }
    }
    return false;
}
// 从 TextField 引申出的新类 MyTextField 类
class MyTextField extends TextField {
    public MyTextField(String text, int columns) {
        super(text, columns);
    }
}

```

```

    }
    public boolean action(Event event, Object arg) {
        String text = getText( );
        setText(text.toUpperCase( ));
        return true;
    }
}
// 从 Button 引申出的新类 MyButton
class MyButton extends Button {
    TextArea textArea;
    public MyButton(String text, TextArea newTextArea) {
        super(text);
        textArea = newTextArea;
    }
    public boolean action(Event event, Object arg) {
        textArea.setText("");
        return true;
    }
}
// 从 Canvas 引申出的新类 MyCanvas 类
class MyCanvas extends Canvas {
    int x = -1;
    int y = -1;
    int boxSize = 10;
    public MyCanvas( ) {
        super( );
        resize(75, 75);
        setBackground(Color.white);
        setForeground(Color.red);
        show( );
    }
    public boolean mouseDown(Event event, int xClick, int yClick) {
        x = xClick;
        y = yClick;
        repaint( );
        return true;
    }
    public void paint(Graphics g) {
        setBackground(Color.white);

```

```

        setForeground(Color. red);
        if(x>=0 && y>=0) g.fillRect(x,y, boxSize, boxSize);
    }
}
// 从 Panel 引申出的 MyCheckboxGroup 类
class MyCheckboxGroup extends Panel {
    String labelString;
    String checkboxLabels[ ];
    Checkbox checkboxes[ ];
    int numBoxes;
    TextField results;
    public MyCheckboxGroup(String strings[ ]) {
        super( );
        labelString = strings[0];
        numBoxes = strings.length-1;
        checkboxLabels = new String[numBoxes];
        for(int i=0; i<numBoxes; ++i)
            checkboxLabels[i] = strings[i+1];
        results = new TextField("", 15);
        setupPanel( );
        show( );
    }
    void setupPanel( ) {
        setLayout(new GridLayout(numBoxes+2, 1));
        add(new Label(labelString));
        checkboxes = new Checkbox[numBoxes];
        for(int i=0; i<numBoxes; ++i) {
            checkboxes[i] = new Checkbox(checkboxLabels[i]);
            add(checkboxes[i]);
        }
        add(results);
    }
    public boolean handleEvent(Event event) {
        if(event.id==Event.ACTION_EVENT) {
            if(event.target instanceof Checkbox) {
                String newResults = "";
                for(int i=0; i<numBoxes; ++i)
                    if(checkboxes[i].getState( ))
                        newResults = newResults + " " +checkboxes[i].getLabel( );
            }
        }
    }
}

```

```

        results.setText(newResults);
    }
}
return false;
}
}
// 从 Choice 类引申出的 MyChoice 类
class MyChoice extends Choice {
    TextArea text;
    public MyChoice(String strings[], TextArea textArea) {
        super();
        try {
            for(int i=0; i<strings.length; ++i)
                addItem(strings[i]);
            text = textArea;
        } catch (NullPointerException ex) {
            System.exit(0);
        }
    }
    public boolean action(Event event, Object arg) {
        text.setText((String) arg);
        return true;
    }
}
// 从 List 类引申出的 MyList 类
class MyList extends List {
    TextArea text;
    public MyList(String strings[], TextArea textArea) {
        super(3, false);
        for(int i=0; i<strings.length; ++i)
            addItem(strings[i]);
        text = textArea;
    }
    public boolean handleEvent(Event event) {
        if (event.id==Event.ACTION_EVENT) {
            text.setText("Double-clicked: \n "+event.arg.toString());
            return true;
        } else if (event.id==Event.LIST_SELECT) {
            text.setText("Selected: \n "+

```

```

        getItem((new Integer(event.arg.toString( )).intValue( ));
    return true;
} else if (event.id==Event.LIST_DESELECT) {
    text.setText("Deselected: \n "+
        getItem((new Integer(event.arg.toString( )).intValue( ));
    return true;
}
return false;
}
}
// 从 Scrollbra 类引申出的 MyScrollbar 类
class MyScrollbar extends Scrollbar {
    TextArea text;
    public MyScrollbar(int orientation, int value, int visible, int min, int max,
        TextArea textArea) {
        super(orientation, value, visible, min, max);
        text=textArea;
    }
    public boolean handleEvent(Event event) {
        if (event.id==Event.SCROLL_LINE_UP) {
            text.setText("Position: "+getValue( ));
            return true;
        } else if (event.id==Event.SCROLL_LINE_DOWN) {
            text.setText("Position: "+getValue( ));
            return true;
        } else if (event.id==Event.SCROLL_PAGE_UP) {
            text.setText("Position: "+getValue( ));
            return true;
        } else if (event.id==Event.SCROLL_PAGE_DOWN) {
            text.setText("Position: "+getValue( ));
            return true;
        } else if (event.id==Event.SCROLL_ABSOLUTE) {
            text.setText("Position: "+getValue( ));
            return true;
        }
        return false;
    }
}
}

```


5.4 两个例子

有了界面元素的基础，我们就可以编写出许多应用。下面的这两个例子分别是利用画布的作图程序和利用滚动条的文本编辑器。在源程序之后，有相应的运行结果。

5.4.1 作图程序

在这个程序中，用户可以在画布上画出不同的几何形状，如直线、矩形、圆和椭圆等。图 5.4 为作图例程的界面示意。

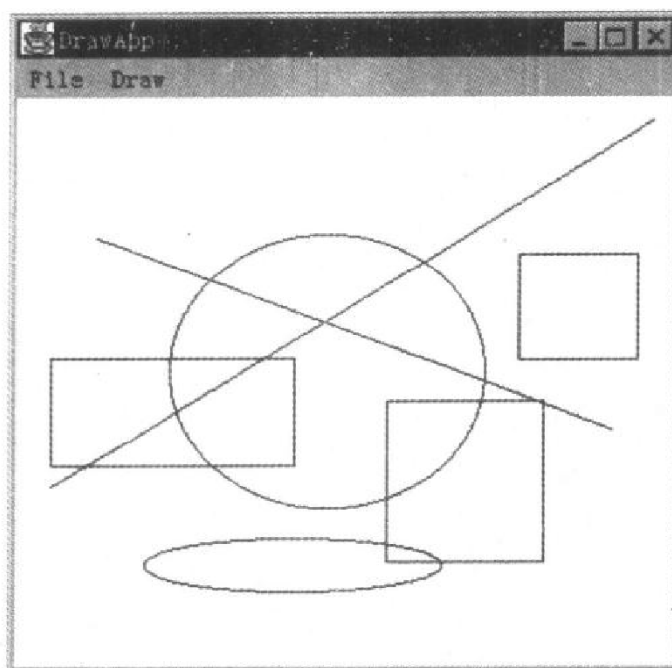


图 5.4 作图例程的界面示意

```
import java.awt.*;
import java.lang.Math;
import java.util.Vector;

public class DrawApp extends Frame {
    Object menuItems[ ][ ] = {
        {"File", "New", "-", "Exit"},
        {"Draw", "+Line", "-Oval", "-Rectangle"}
    };
    MyMenuBar menuBar = new MyMenuBar(menuItems); // 创建菜单条和菜单项
    MyCanvas canvas = new MyCanvas(TwoPointObject.LINE); // 创建画布对象
    int screenWidth = 400;
```

```

int screenHeight = 400;
public static void main(String args[ ]){
    DrawApp app = new DrawApp( );
}    // 构造函数, 初始化系统

public DrawApp( ) {
    super("DrawApp");
    setup( );
    pack( );
    resize(screenWidth, screenHeight);
    show( );
}

void setup( ) {
    setBackground(Color.white);
    setMenuBar(menuBar);
    setCursor(CROSSHAIR-CURSOR); // 设定光标形状
    add("Center", canvas);
}    // 事件处理函数

public boolean handleEvent(Event event) {
    if (event.id==Event.WINDOW_DESTROY) {
        System.exit(0);
        return true;
    } else if (event.id==Event.GOT_FOCUS && !(event.target instanceof MyCanvas)) {
        setCursor(DEFAULT-CURSOR);
        return true;
    } else if (event.id==Event.LOST_FOCUS) {
        setCursor(CROSSHAIR-CURSOR);
        return true;
    } else if (event.id==Event.ACTION_EVENT) {
        if (event.target instanceof MenuItem) {
            String arg = (String) event.arg;
            if (processFileMenu(arg)) return true;
            if (processDrawMenu(arg)) return true;
        }
    }
    return false;
}    // 处理 Draw 菜单中的各个菜单项

```

```

public boolean processFileMenu(String s) {
    if ("New".equals(s)) {
        canvas.clear();
        return true;
    } else if ("Exit".equals(s)) {
        System.exit(0);
        return true;
    }
    return false;
}

public boolean processDrawMenu(String s) {
    MyMenu menu = menuBar.getMenu("Draw");
    CheckboxMenuItem lineItem = (CheckboxMenuItem) menu.getItem("Line");
    CheckboxMenuItem ovalItem = (CheckboxMenuItem) menu.getItem("Oval");
    CheckboxMenuItem rectangleItem =
        (CheckboxMenuItem) menu.getItem("Rectangle");
    if ("Line".equals(s)) {
        canvas.setTool(TwoPointObject.LINE);
        lineItem.setState(true);
        ovalItem.setState(false);
        rectangleItem.setState(false);
        return true;
    } else if ("Oval".equals(s)) {
        canvas.setTool(TwoPointObject.OVAL);
        lineItem.setState(false);
        ovalItem.setState(true);
        rectangleItem.setState(false);
        return true;
    } else if ("Rectangle".equals(s)) {
        canvas.setTool(TwoPointObject.RECTANGLE);
        lineItem.setState(false);
        ovalItem.setState(false);
        rectangleItem.setState(true);
        return true;
    }
    return false;
}
}

// 从 Canvas 引申出的 MyCanvas 对象, 它是作图的画布。它规定了各种操作, 如清空、

```

```

// 鼠标按下、鼠标拖动事件等的处理。
class MyCanvas extends Canvas {
    int tool = TwoPointObject.LINE;
    Vector objects = new Vector();
    TwoPointObject current;
    boolean newObject = false;
    public MyCanvas(int toolType) {
        super();
        tool = toolType;
    }
    public void setTool(int toolType) {
        tool = toolType;
    }
    public void clear() {
        objects.removeAllElements();
        repaint();
    }
    public boolean mouseDown(Event event, int x, int y) {
        current = new TwoPointObject(tool, x, y);
        newObject = true;
        return true;
    }
    public boolean mouseUp(Event event, int x, int y) {
        if(newObject) {
            objects.addElement(current);
            newObject = false;
        }
        return true;
    }
    public boolean mouseDrag(Event event, int x, int y) {
        if(newObject) {
            int oldX = current.endX;
            int oldY = current.endY;
            if(tool != TwoPointObject.LINE) {
                if(x > current.startX) current.endX = x;
                if(y > current.startY) current.endY = y;
                int width = Math.max(oldX, current.endX) - current.startX + 1;
                int height = Math.max(oldY, current.endY) - current.startY + 1;
                repaint(current.startX, current.startY, width, height);
            }
        }
    }
}

```

```

    }else{
        current.endX = x;
        current.endY = y;
        int startX = Math.min(Math.min(current.startX, current.endX), oldX);
        int startY = Math.min(Math.min(current.startY, current.endY), oldY);
        int endX = Math.max(Math.max(current.startX, current.endX), oldX);
        int endY = Math.max(Math.max(current.startY, current.endY), oldY);
        repaint(startX, startY, endX-startX+1, endY-startY+1);
    }
}

return true;
} // MyCanvas 中的作图程序

public void paint(Graphics g) {
    int numObjects = objects.size();
    for(int i=0; i<numObjects; ++i) {
        TwoPointObject obj = (TwoPointObject) objects.elementAt(i);
        obj.draw(g);
    }
    if(newObject) current.draw(g);
}

// TwoPointObject 类包括了线、圆、矩形和椭圆这些用两个点就能定位的对象
// 和操作的特性。
class TwoPointObject {
    public static int LINE = 0;
    public static int OVAL = 1;
    public static int RECTANGLE = 2;
    public int type, startX, startY, endX, endY;
    public TwoPointObject(int objectType, int x1, int y1, int x2, int y2) {
        type = objectType;
        startX = x1;
        startY = y1;
        endX = x2;
        endY = y2;
    }
    public TwoPointObject(int objectType, int x, int y) {
        this(objectType, x, y, x, y);
    }
    public TwoPointObject() {

```

```

        this(LINE, 0, 0, 0, 0);
    }    // 作图函数，在 MyCanvas 的作图程序中被引用。

    public void draw(Graphics g) {
        if (type == LINE) g.drawLine(startX, startY, endX, endY);
        else {
            int w = Math.abs(endX - startX);
            int l = Math.abs(endY - startY);
            if (type == OVAL) g.drawOval(startX, startY, w, l);
            else g.drawRect(startX, startY, w, l);
        }
    }
}

// 从 MenuBar 引申出的 MyMenuBar 类
class MyMenuBar extends MenuBar {
    public MyMenuBar(Object labels[ ][ ]) {
        super();
        for (int i=0; i<labels.length; ++i)
            add(new MyMenu(labels[i]));
    }

    public MyMenu getMenu(String menuName) {
        int numMenus = countMenus();
        for (int i=0; i<numMenus; ++i)
            if (menuName.equals(getMenu(i).getLabel( ))) return ((MyMenu) getMenu(i));
        return null;
    }
}

// 从 Menu 引申出的 MyMenu 类
class MyMenu extends Menu {
    public MyMenu(Object labels[ ]) {
        super((String) labels[0]);
        String menuName = (String) labels[0];
        char firstMenuChar = menuName.charAt(0);
        if (firstMenuChar == '^' || firstMenuChar == '!') {
            setLabel(menuName.substring(1));
            if (firstMenuChar == '^') disable();
        }
        for (int i=1; i<labels.length; ++i) {
            if (labels[i] instanceof String) {

```

```

        if("-".equals(labels[i])) addSeparator( );
    else{
        String label = (String)labels[i];
        char firstChar = label.charAt(0);
        switch(firstChar) {
            case '+':
                CheckboxMenuItem checkboxItem = new CheckboxMenuItem(label.substring(1));
                checkboxItem.setState(true);
                add(checkboxItem);
                break;
            case '#':
                checkboxItem = new CheckboxMenuItem(label.substring(1));
                checkboxItem.setState(true);
                checkboxItem.disable( );
                add(checkboxItem);
                break;
            case '-':
                checkboxItem = new CheckboxMenuItem(label.substring(1));
                checkboxItem.setState(false);
                add(checkboxItem);
                break;
            case '=':
                checkboxItem = new CheckboxMenuItem(label.substring(1));
                checkboxItem.setState(false);
                checkboxItem.disable( );
                add(checkboxItem);
                break;
            case '^':
                MenuItem menuItem = new MenuItem(label.substring(1));
                menuItem.disable( );
                add(menuItem);
                break;
            case '!':
                add(label.substring(1));
                break;
            default:
                add(label);
        }
    }
}

```

```

    }else{
        add(new MyMenu((Object[ ])labels[i]));
    }
}
}
}

public MenuItem getItem(String menuItem) {
    int numItems = countItems();
    for(int i=0;i<numItems;++i)
        if(menuItem.equals(getItem(i).getLabel())) return getItem(i);
    return null;
}
}
}

```

5.4.2 文本编辑器

这个文本编辑器带有水平和垂直的滚动条，可以处理大文本。菜单的处理方式与前面的例子一样。图 5.5 带有滚动条的文本编辑器示意图。

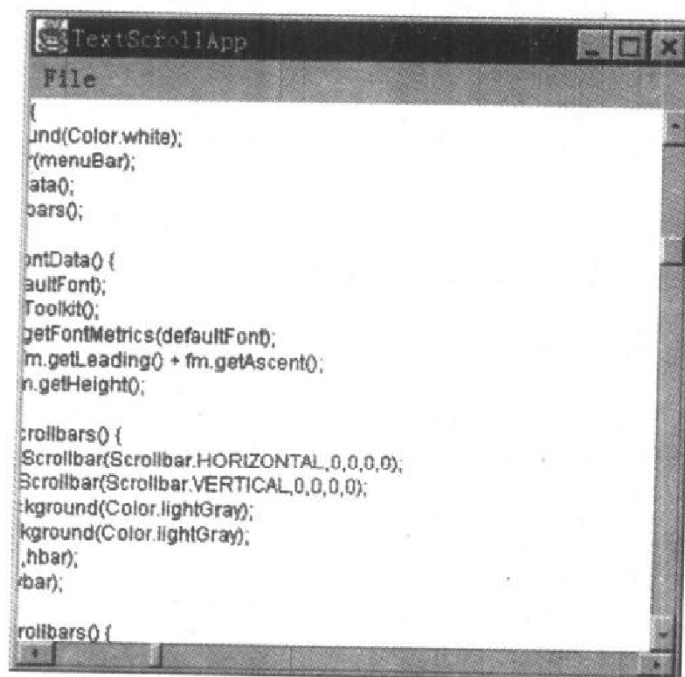


图 5.5 带有滚动条的文本编辑器

```

import java.awt.*;
import java.io.*;
import java.util.Vector;

```



```

public class TextScrollApp extends Frame {
    Object menuItems[ ][ ] = {{"File", "Open", "-", "Exit"}};
    MyMenuBar menuBar = new MyMenuBar(menuItems);
    FileDialog openFile = new FileDialog(this, "Open File", FileDialog.LOAD);
    Font defaultFont = new Font("default", Font.PLAIN, 12);
    int screenWidth = 400;
    int screenHeight = 400;
    Vector text = new Vector( );
    int topLine;
    Toolkit toolkit;
    FontMetrics fm;
    int baseline;
    int lineSize;
    int maxWidth;
    Scrollbar hbar, vbar;
    public static void main(String args[ ]){
        TextScrollApp app = new TextScrollApp( );
    }        // 初始化

    public TextScrollApp( ) {
        super("TextScrollApp");
        setup( );
        pack( );
        resize(screenWidth, screenHeight);
        show( );
    }
    void setup( ) {
        setBackground(Color.white);
        setMenuBar(menuBar);
        setupFontData( );
        setupScrollbars( );
    }
    // 初始化字体数据
    void setupFontData( ) {
        setFont(defaultFont);
        toolkit = getToolkit( );
        fm = toolkit.getFontMetrics(defaultFont);
        baseline = fm.getLeading( ) + fm.getAscent( );
        lineSize = fm.getHeight( );
    }

```

```

    }
    // 初始化滚动条
void setupScrollbars() {
    hbar = new Scrollbar(Scrollbar.HORIZONTAL, 0, 0, 0, 0);
    vbar = new Scrollbar(Scrollbar.VERTICAL, 0, 0, 0, 0);
    hbar.setBackground(Color.lightGray);
    vbar.setBackground(Color.lightGray);
    add("South", hbar);
    add("East", vbar);
}
// 重置滚动条
void resetScrollbars() {
    hbar.setValues(0, 10, 0, maxWidth+5);
    vbar.setValues(0, 10, 0, text.size()+5);
}
// 读文件
public void readFile(String file) {
    DataInputStream inStream;
    try{
        inStream = new DataInputStream(new FileInputStream(file));
    }catch (IOException ex) {
        notifyUser("Error opening file");
        return;
    }
    try{
        Vector newText = new Vector();
        String line;
        maxWidth = 0;
        while((line=inStream.readLine())!=null) {
            int lineWidth = fm.stringWidth(line);
            if(lineWidth > maxWidth) maxWidth = lineWidth;
            newText.addElement(line);
        }
        text = newText;
        topLine = 0;
        inStream.close();
        resetScrollbars();
        repaint();
    }catch (IOException ex) {

```

```

        notifyUser("Error reading file");
    }
}

public void notifyUser(String s) {
    String text[ ] = {s};
    String buttons[ ] = {"OK"};
    new Notification(this, "Error", true, text, buttons);
}

// 显示文本内容
public void paint(Graphics g) {
    topLine = vbar.getValue( );
    int xOffset = hbar.getValue( );
    int numLines = text.size( );
    screenHeight = size( ).height;
    int y = baseline;
    for(int i = topLine; (i < numLines) && (y < screenHeight + lineSize); ++i) {
        g.drawString((String) text.elementAt(i), -xOffset, y);
        y += lineSize;
    }
}

// 事件处理
public boolean handleEvent(Event event) {
    if(event.id==Event.WINDOW_DESTROY) {
        System.exit(0);
        return true;
    } else if(event.id==Event.ACTION_EVENT) {
        String s = (String) event.arg;
        if(event.target instanceof MenuItem) {
            if("Exit".equals(s)) {
                System.exit(0);
                return true;
            } else if("Open".equals(s)) {
                openFile.show( );
                if(openFile.getFile( ) != null) {
                    String inFile = openFile.getFile( );
                    readFile(inFile);
                }
            }
        }
        return true;
    }
}

```

```

    )
} else if (event.id == Event.SCROLL_LINE_UP ||
    event.id == Event.SCROLL_LINE_DOWN ||
    event.id == Event.SCROLL_PAGE_UP ||
    event.id == Event.SCROLL_PAGE_DOWN ||
    event.id == Event.SCROLL_ABSOLUTE) repaint();
return false;
}
}

// 信息对话框, 给用户以提示信息
class Notification extends MessageDialog {
    public Notification(Frame parent, String title, boolean modal,
        String text[], String buttons[]) {
        super(parent, title, modal, text, buttons);
    }

    public boolean handleEvent(Event event) {
        if (event.id == Event.WINDOW_DESTROY) {
            dispose();
            return true;
        } else if (event.id == Event.ACTION_EVENT && event.target instanceof Button) {
            dispose();
            return true;
        }
        return false;
    }
}

// 从 Menu 引申出的 MyMenu 类
class MyMenu extends Menu {
    public MyMenu(Object labels[]) {
        super((String) labels[0]);
        String menuName = (String) labels[0];
        char firstMenuChar = menuName.charAt(0);
        if (firstMenuChar == '-' || firstMenuChar == '!') {
            setLabel(menuName.substring(1));
            if (firstMenuChar == '-') disable();
        }
        for (int i = 1; i < labels.length; ++i) {
            if (labels[i] instanceof String) {
                if ("-".equals(labels[i])) addSeparator();
            }
        }
    }
}

```

```

else{
    String label = (String)labels[i];
    char firstChar = label.charAt(0);
    switch(firstChar) {
        case '+':
CheckboxMenuItem checkboxItem = new CheckboxMenuItem(label.substring(1));
        checkboxItem.setState(true);
        add(checkboxItem);
        break;
        case '#':
        checkboxItem = new CheckboxMenuItem(label.substring(1));
        checkboxItem.setState(true);
        checkboxItem.disable( );
        add(checkboxItem);
        break;
        case '-':
        checkboxItem = new CheckboxMenuItem(label.substring(1));
        checkboxItem.setState(false);
        add(checkboxItem);
        break;
        case '=':
        checkboxItem = new CheckboxMenuItem(label.substring(1));
        checkboxItem.setState(false);
        checkboxItem.disable( );
        add(checkboxItem);
        break;
        case '^':
        MenuItem menuItem = new MenuItem(label.substring(1));
        menuItem.disable( );
        add(menuItem);
        break;
        case '!':
        add(label.substring(1));
        break;
        default:
        add(label);
    }
}
} else{

```

```

        add(new MyMenu((Object[ ]) labels[i]));
    }
}

public MenuItem getItem(String menuItem) {
    int numItems = countItems();
    for(int i=0; i<numItems; ++i)
        if(menuItem.equals(getItem(i).getLabel())) return getItem(i);
    return null;
}

// 从 MenuBar 引申出的 MyMenuBar 类
class MyMenuBar extends MenuBar {
    public MyMenuBar(Object labels[ ][ ]) {
        super();
        for(int i=0; i<labels.length; ++i)
            add(new MyMenu(labels[i]));
    }

    public MyMenu getMenu(String menuName) {
        int numMenus = countMenus();
        for(int i=0; i<numMenus; ++i)
            if(menuName.equals(getMenu(i).getLabel())) return((MyMenu) getMenu(i));
        return null;
    }
}

// 从 Dialog 引申出的信息对话框类 MessageDialog 类
class MessageDialog extends Dialog {
    public MessageDialog(Frame parent, String title, boolean modal, String text[ ],
        String buttons[ ]) {
        super(parent, title, modal);
        int textLines = text.length;
        int numButtons = buttons.length;
        Panel textPanel = new Panel();
        Panel buttonPanel = new Panel();
        textPanel.setLayout(new GridLayout(textLines, 1));
        for(int i=0; i<textLines; ++i) textPanel.add(new Label(text[i]));
        for(int i=0; i<numButtons; ++i) buttonPanel.add(new Button(buttons[i]));
        add("North", textPanel);
        add("South", buttonPanel);
    }
}

```

```

setBackground(Color.lightGray);
setForeground(Color.white);
pack();
resize(minimumSize());
}
public boolean handleEvent(Event event) {
    if(event.id==Event.WINDOW_DESTROY) show(false);
    else if(event.id==Event.ACTION_EVENT && event.target instanceof Button)
        show(false);
    return false;
}
}

```

5.5 字体和颜色

字体和颜色在 Java 中分别由 Font 和 Color 支持。与字体相关的还有一个类 FontMetrics，它是用来确定字体的大小，以定位下一行字符的位置。下面的例子介绍了这三个类的使用方法。它的输出结果是在屏幕上以不同的颜色输出各种字体，字符串本身是字体的名字和风格及颜色的名字。在例子中，最后一行的输出是很杂乱的字符，可能是因为在中文 Win95 上运行的结果。图 5.6 为本例的屏幕输出结果。



图 5.6 字体和颜色例程的屏幕输出

```

import java.awt.*;

public class FontColorApp extends Frame {
    Toolkit toolkit;
    Font defaultFont;
    String fontNames[ ];
    int screenWidth = 400;
    int screenHeight = 400;
    public static void main(String args[ ]){
        FontApp app = new FontApp( );
    }    // 初始化
    public FontApp( ) {
        super("FontApp");
        setup( );
        pack( );
        resize(screenWidth, screenHeight);
        show( );
    }
    void setup( ) {
        setBackground(Color.white);
        setupFonts( );
    }    // 初始化字体
    void setupFonts( ) {
        toolkit = getToolkit( );
        defaultFont = getFont( );
        fontNames = toolkit.getFontList( );
    }    // 在屏幕上作图
    public void paint(Graphics g) {
        int styles[ ] = {Font.PLAIN, Font.BOLD, Font.ITALIC, Font.BOLD+Font.ITALIC};
        String styleNames[ ] = {"Plain", "Bold", "Italic", "Bold and Italic"};
        Color colors[ ] = {Color.black, Color.blue, Color.cyan, Color.darkGray, Color.gray,
            Color.green, Color.lightGray, Color.magenta, Color.orange, Color.pink, Color.red,
            Color.white, Color.yellow};
        String colorNames[ ] = {"black", "blue", "cyan", "darkGray", "gray", "green",
            "lightGray", "magenta", "orange", "pink", "red", "white", "yellow"};

        int y = 0;
        int size = 14;
        for(int i=0; i<fontNames.length; ++i) {

```



```

        for(int j=0; j<styles.length; ++j) {
            Font newFont = new Font(fontNames[i], styles[j], size);
            FontMetrics fm = g.getFontMetrics(newFont);
            g.setFont(newFont);
            g.setColor(colors[(i*fontNames.length+j)%colors.length]);
            String text = fontNames[i]+"-"+styleNames[j]+"- "
                +colorNames[(i*fontNames.length+j)%colors.length];
            int x = (screenWidth - fm.stringWidth(text))/2;
            g.drawString(text, x, y+fm.getLeading()+fm.getAscent());
            y += fm.getHeight();
        }
    }
}

// 事件处理
public boolean handleEvent(Event event) {
    if(event.id==Event.WINDOW_DESTROY) {
        System.exit(0);
        return true;
    }
    return false;
}
}

```

5.6 界面元素包容器

我们已经介绍了许多界面元素的应用。当有多个界面元素时，就需要解决一个界面元素之间的关系问题。由于 Java 要求的高度可移植性，没有采用原来的直接定坐标的方法，而是用了界面元素包容器。目前，已有的界面元素包容器有： BorderLayout、CardLayout、FlowLayout、GridLayout、GridBagLayout。它们安排出的效果如图 5.7 所示，在窗口的标题中有显示说明。其中 CardLayout 是由许多页组成的，需要一定的消息触发机制来翻页，在我们的例子中是借助于菜单中增加的一项来翻页的。

如果以上已有的界面元素包容器不能满足要求，可以设计自用的界面包容器类。该类要实现 LayoutManager 接口（Interface）。但通常情况下，我们鼓励使用已有的界面元素包容器。

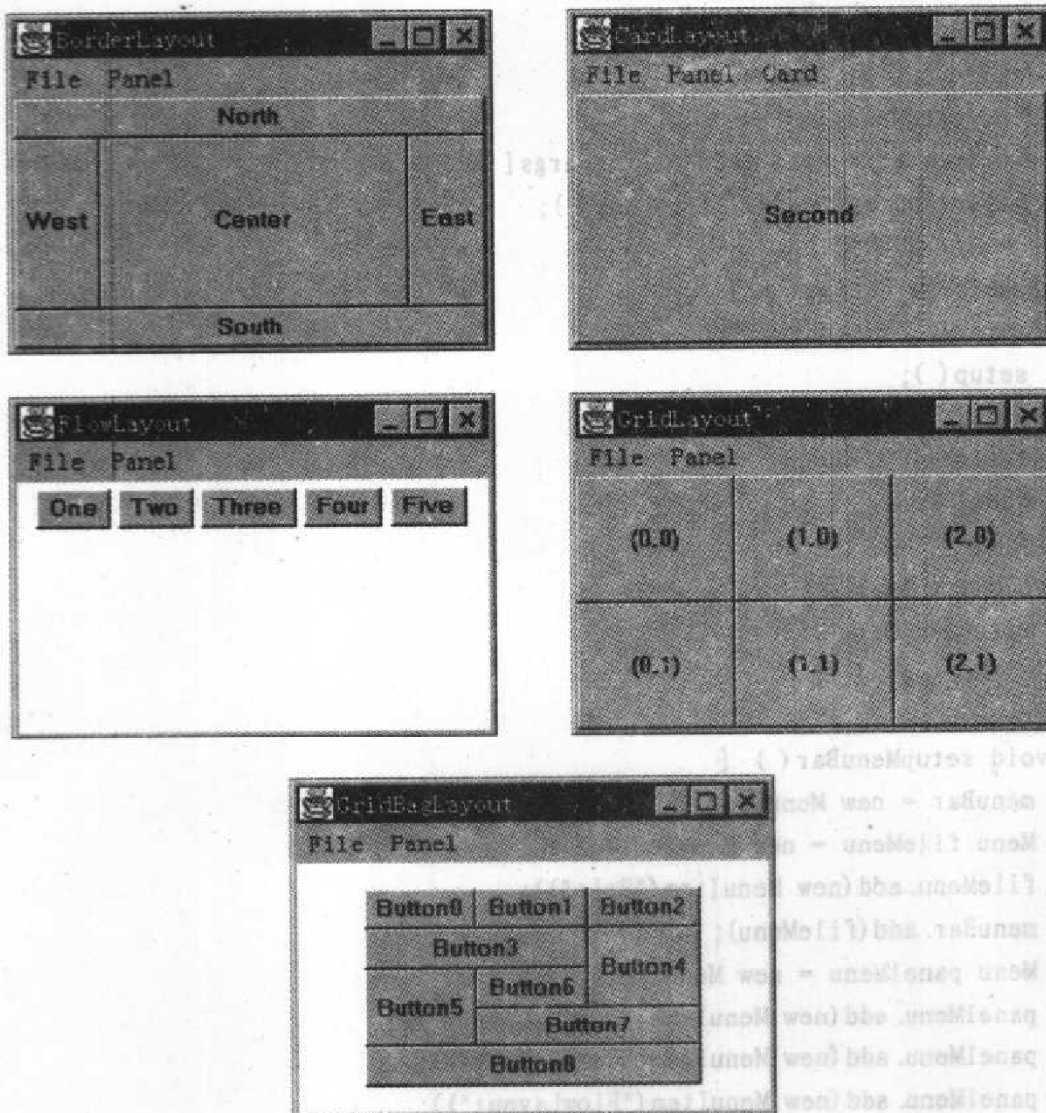


图 5.7 各种界面元素包容器的显示效果

这里是上面例子的程序源代码:

```
import java.awt.*;

public class LayoutApp extends Frame {
    MenuBar menuBar;
    Panel panels[];
    Panel currentPanel;
    static int border=0;
    static int card=1;
    static int flow=2;
```

```

static int grid=3;
static int gridBag=4;
Menu cardMenu;
public static void main(String args[ ]){
    LayoutApp app = new LayoutApp( );
}    // 初始化
public LayoutApp( ) {
    super("BorderLayout");
    setup( );
    pack( );
    resize(400,400);
    show( );
}
void setup( ) {
    setupMenuBar( );
    setupPanels( );
}
void setupMenuBar( ) {
    menuBar = new MenuBar( );
    Menu fileMenu = new Menu("File");
    fileMenu.add(new MenuItem("Exit"));
    menuBar.add(fileMenu);
    Menu panelMenu = new Menu("Panel");
    panelMenu.add(new MenuItem("BorderLayout"));
    panelMenu.add(new MenuItem("CardLayout"));
    panelMenu.add(new MenuItem("FlowLayout"));
    panelMenu.add(new MenuItem("GridLayout"));
    panelMenu.add(new MenuItem("GridBagLayout"));
    menuBar.add(panelMenu);
    cardMenu = new Menu("Card");
    cardMenu.add(new MenuItem("First"));
    cardMenu.add(new MenuItem("Last"));
    cardMenu.add(new MenuItem("Next"));
    cardMenu.add(new MenuItem("Previous"));
    setMenuBar(menuBar);
}    // 设置各种模式的 Panel 形式
void setupPanels( ) {
    panels = new Panel[5];
    for(int i=0;i<5;++i) panels[i]=new Panel( );
}

```

```

panels[border].setLayout(new BorderLayout( ));
panels[card].setLayout(new CardLayout( ));
panels[flow].setLayout(new FlowLayout( ));
panels[grid].setLayout(new GridLayout(2,3));
GridBagLayout gridBagLayout = new GridBagLayout( );
panels[gridBag].setLayout(gridBagLayout);
panels[border].add("North",new Button("North"));
panels[border].add("South",new Button("South"));
panels[border].add("East",new Button("East"));
panels[border].add("West",new Button("West"));
panels[border].add("Center",new Button("Center"));
String cardButtons[ ] = {"First","Second","Third","Fourth","Last"};
String flowButtons[ ] = {"One","Two","Three","Four","Five"};
String gridButtons[ ] = {"(0,0)","(1,0)","(2,0)","(0,1)","(1,1)","(2,1)"};
for(int i=0; i<cardButtons.length; ++i)
    panels[card].add(new Button(cardButtons[i]));
for(int i=0; i<flowButtons.length; ++i)
    panels[flow].add(new Button(flowButtons[i]));
for(int i=0; i<gridButtons.length; ++i)
    panels[grid].add(new Button(gridButtons[i]));
Button gridBagButtons[ ] = new Button[9];
for(int i=0; i<9; ++i) gridBagButtons[i] = new Button("Button"+i);
int gridx[ ] = {0,1,2,0,2,0,1,1,0};
int gridy[ ] = {0,0,0,1,1,2,2,3,4};
int gridwidth[ ] = {1,1,1,2,1,1,1,2,3};
int gridheight[ ] = {1,1,1,1,2,2,1,1,1};
GridBagConstraints gridBagConstraints[ ] = new GridBagConstraints[9];
for(int i=0; i<9; ++i) {
    gridBagConstraints[i] = new GridBagConstraints( );
    gridBagConstraints[i].fill=GridBagConstraints.BOTH;
    gridBagConstraints[i].gridx=gridx[i];
    gridBagConstraints[i].gridy=gridy[i];
    gridBagConstraints[i].gridwidth=gridwidth[i];
    gridBagConstraints[i].gridheight=gridheight[i];
    gridBagLayout.setConstraints(gridBagButtons[i],gridBagConstraints[i]);
    panels[gridBag].add(gridBagButtons[i]);
}
add("Center",panels[border]);
currentPanel=panels[border];

```

```

}
// 事件处理
public boolean handleEvent(Event event) {
    if (event.id==Event.WINDOW_DESTROY) {
        System.exit(0);
        return true;
    } else if (event.id==Event.ACTION_EVENT) {
        if (event.target instanceof MenuItem) {
            if ("Exit".equals(event.arg)) {
                System.exit(0);
                return true;
            } else if ("BorderLayout".equals(event.arg)) {
                switchPanels(panels[border], "BorderLayout", false);
                return true;
            } else if ("CardLayout".equals(event.arg)) {
                switchPanels(panels[card], "CardLayout", true);
                return true;
            } else if ("FlowLayout".equals(event.arg)) {
                switchPanels(panels[flow], "FlowLayout", false);
                return true;
            } else if ("GridLayout".equals(event.arg)) {
                switchPanels(panels[grid], "GridLayout", false);
                return true;
            } else if ("GridBagLayout".equals(event.arg)) {
                switchPanels(panels[gridBag], "GridBagLayout", false);
                return true;
            } else if ("First".equals(event.arg)) {
                CardLayout currentLayout=(CardLayout)currentPanel.getLayout();
                currentLayout.first(currentPanel);
                return true;
            } else if ("Last".equals(event.arg)) {
                CardLayout currentLayout=(CardLayout)currentPanel.getLayout();
                currentLayout.last(currentPanel);
                return true;
            } else if ("Next".equals(event.arg)) {
                CardLayout currentLayout=(CardLayout)currentPanel.getLayout();
                currentLayout.next(currentPanel);
                return true;
            } else if ("Previous".equals(event.arg)) {

```

```

        CardLayout currentLayout=(CardLayout)currentPanel.getLayout();
        currentLayout.previous(currentPanel);
        return true;
    }
}
return false;
}
// 切换各种模式的 Panel
void switchPanels (Panel newPanel,String newTitle,boolean setCardMenu) {
    remove(currentPanel);
    currentPanel=newPanel;
    add("Center",currentPanel);
    setTitle(newTitle);
    if(setCardMenu) menuBar.add(cardMenu);
    else menuBar.remove(cardMenu);
    show();
}
}

```

5.7 小 结

在这一章中，我们介绍了 Java 的用户界面编程，其中包括：

- Java 用户界面的基本概念。包括界面元素、界面元素容器、事件驱动、图形处理等。
- 各种界面元素的编程，包括菜单、按钮、滚动条、文本域、文本输入框、列表等，其中重点介绍了菜单的应用编程。在介绍完所有的界面元素后，我们还列举了两个典型应用：作图程序和记事本程序。
- 字体和颜色类的应用编程。
- 界面元素容器的几种常见形式和应用编程。通过一个例子示例了这几种常见的界面元素容器的使用。

第六章 编写 Applet 应用

Java 的一项主要应用就是生成 Applet。简单地说, Applet 是使用 AppletViewer 或支持 Java 的 WWW 浏览器来运行的编译后的 Java 程序。Applet 可为网络提供丰富的功能,如在 WWW 页面显示动画、播放音乐、接受用户输入以及操作数据等,目前,它已成为 Internet 应用不可缺少的一部分。

本书第二章已对 Applet 作了一些描述,这一章我们将深入全面地分析 Applet,从简单的 Applet 编程到复杂的 Internet 应用,为你揭示 Applet 编程的奥秘。

6.1 概 述

Applet 并不是专门为 Internet 设计的,实际上它们最初是用于手持的计算设备如 PDA (Personal Digital Assistants) 的。然而,由于 Applet 程序比较小,可从网上很快下载并启动运行,这使得它们非常适合在 Internet 和 WWW 上使用。Applet 给 WWW 页面增加了新的功能和交互性,但却没有大型应用的过高开销。随着支持 Java 的浏览器,如 Netscape 等的出现,Applet 必将对 WWW 产生重要的影响。

目前,Internet 上已经有很多 Web 页面使用了 Applet 技术,Applet 在 WWW 上的使用主要有以下几种方式:

- 动画
- 图象显示并伴有声音
- 各种图形效果,如滚动的文本
- 交互程序,如游戏

Java Applet 利用了 WWW 浏览器本身已有的特点,以很少的代码即可获得丰富的性能。例如,在 Applet 中使用 GIF 或 JPEG 文件,就不需要为这些图象文件编写解码程序,而可以直接使用浏览器固有的解码程序来显示图象。利用浏览器的功能使得 Applet 很容易扩展,如果浏览器引入一种新的流行的图象格式,那么 Applet 也可以自动处理这种格式。Java Applet 正是由于利用了浏览器的特点,减少了大小,加快了下载速度并简化了 Applet 编程,从而非常适合于 WWW。

下面我们用一个图来说明 Java Applet 在浏览器中的运行过程。如图 6.1 所示,当浏览器在 HTML 文本中遇到 applet 标签时,即开始收集启动该 Applet 所需的信息,在 HTML 文本全部解释并显示后,就由 Java 解释器来运行 Applet。

Java 解释器收到执行 Applet 的请求后,即通过加载器来获取相应的二进制文件.class。在文件成功传送到本机后,还要经过一些测试来检验它的安全性和稳定性,如果没有问题,解释器就开始运行该 Applet。除非用户连接到另一 URL 或退出浏览器环境,Applet 将一直运行到结束。

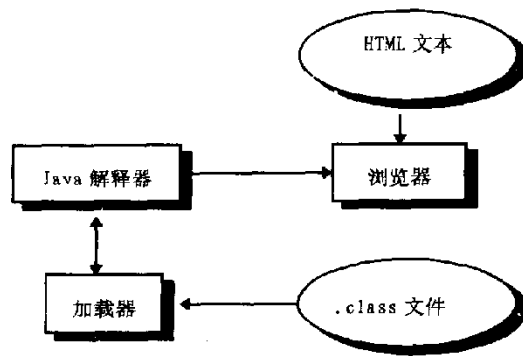


图 6.1 JAVA Applet 的运行过程

6.2 Applet 在 HTML 文本中的嵌入方式

在你编写完了一个 Applet 后，如何来运行测试它呢？通常在浏览器或 AppletViewer 中运行 Applet，需要使用<APPLET>标签将它加入到 HTML 文本中，然后，在浏览器或 AppletViewer 中指定该 HTML 文本的 URL 即可运行 Applet。

Applet 在 HTML 文本中嵌入的一个最简单例子如下：

```
<APPLET CODE=AppletTest.class WIDTH=200 HEIGHT=100></APPLET>
```

其中 AppletTest.class 是 Applet 程序编译后生成的字节码，WIDTH 和 HEIGHT 指定了 Applet 在 Web 页面中所占的宽度和高度。当浏览器遇到<APPLET>标签时，就为 Applet 保留一块指定宽度和高度的显示区域，随后载入该 Applet 的可执行字节码。

Applet 在 HTML 文本中的嵌入方式是非常灵活的，其一般格式如下：

```
<APPLET CODEBASE=class URL
  CODE=class filename
  ALT=alternate text
  NAME=applet name
  WIDTH=width
  HEIGHT=height
  ALIGN=alignment
  VSPACE=vertical spacing
  HSPACE=horizontal spacing>
<PARAM NAME=parameter1Name VALUE=aValue>
<PARAM NAME=parameter2Name VALUE=anotherValue>
</APPLET>
```

我们在表 6.1 说明了 Applet 标签中各选项的作用。这里值得一提的是，Applet 允许用户使用<PARAM>标签来设置参数，它使得 Applet 的应用极为灵活，例如下面的 HTML 语句即为 Applet 指定了多个参数


```

<applet codebase="c:\java\demo\Animator"
code=Animator.class width=200 height=200>
<param name=imagesource value="frames">
<param name=endimage value=10>
<param name=soundsources value="audio">
<param name=soundtrack value=sound.au>
<param name=sounds value="1.au|2.au|3.au|4.au|5.au|6.au|7.au|8.au|9.au|10.au">
<param name=pause value=200>
</applet>

```

表 6.1 <APPLET>标签选项说明

选 项	作 用
CODEBASE	这是一个可选的属性，用来说明.class 文件的路径，它可以是一个绝对的 URL，也可以是 HTML 文件的相对路径。
CODE	这是一个必要的属性，用来说明 Applet 加载的.class 文件名。
ALT	这是一个可选的属性，当浏览器能识别<APPLET>标签，但不能解释 Applet 程序时，可显示由它指定的文本字符。
NAME	这是一个可选的属性，表示该 Applet 的名字，可用来与同一 Web 页面的其它 Applet 通讯。
WIDTH	这是一个必要的属性，用来指定 Applet 在浏览器中以象素表示的宽度。
HEIGHT	这是一个必要的属性，用来指定 Applet 在浏览器中以象素表示的高度。
ALIGN	这是一个可选的属性，用来说明 Applet 如何在 Web 页面中对齐，可选值为： left、right、top、texttop、middle、absmiddle、baseline、bottom、absbottom。
VSPACE	这是一个可选的属性，用来说明 Applet 与 Web 页面中其它元素以象素表示的垂直间距。
HSPACE	这是一个可选的属性，用来说明 Applet 与 Web 页面中其它元素以象素表示的水平间距。

此外，当浏览器不支持 Java Applet 时，可使用<blockquote>标签来说明。例如在下面的例子中，如果浏览器不能解释<APPLET>，将忽略<blockquote>标签前面的部分；而能解释<APPLET>的浏览器将忽略<blockquote>和</blockquote>之间的部分。

```

<applet code=AppletButton.class codebase=example width=350 height=60>
<param name=windowType value=BorderWindow>
<param name=windowText value="BorderLayout">
<param name=buttonText value="Click here to see a BorderLayout in action">
<blockquote>
<hr>
<em>
Your browser can't run 1.0 Java applets,
so here's a picture of the window the program brings up:</em>

```

```
<p>  
<img src=images/BorderEx1.gif width=302 height=138>  
<hr>  
</blockquote>  
</applet>
```

6.3 Applet 编程基础

Java 面向对象的特性有助于从技术角度来理解 Applet 的定义，根据 Java 类的层次关系，Applet 是由 AWT 类库中的 Panel 派生的，而 Panel 是 Container 类的子类，如图 6.2 所示。Container 类可包容各种 Java 图形构件，如按钮、菜单、滚动条等。Applet 正是由于继承了上述类的特点，使得它具有非常丰富的功能。

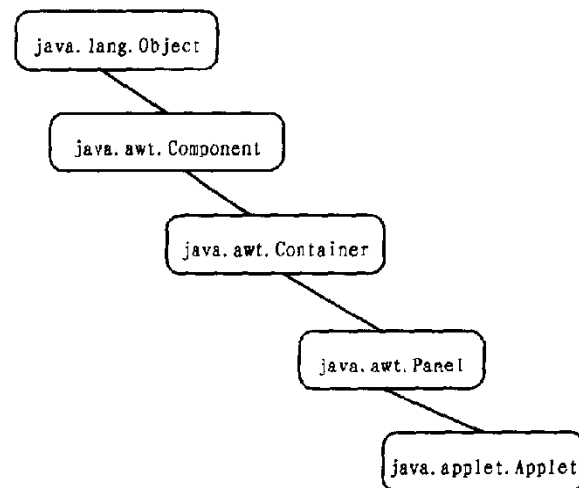


图 6.2 Applet 类的继承关系

6.3.1 Applet 的生命周期

Applet 的生命周期分为四个阶段：init()、start()、stop()、destroy()，这几个阶段的成员函数将被自动调用，并可用来控制一个 Applet 在各阶段的运行。你可以在应用中重写这些成员函数以执行特定的任务，但不需要再声明这些成员函数。

1. init()

Applet 启动后调用的第一个成员函数是 init()，由于它在 Applet 开始运行时即被调用，因此可重写该函数来执行一些初始化操作，如加载图象、建立页面布局或分析参数信息等。

例如下面的几行程序将载入指定的音频文件，并在页面布局中生成一个 Play 按钮。在 Applet 开始运行前载入需要的文件，可以避免出现诸如 Applet 试图显示一个未载入的图象文件的冲突。

```

public void init() {
    clip = getAudioClip(getDocumentBase(), soundfile.au);
    setLayout(new FlowLayout());
    play = new Button("Play Clip");
    add(play);
}

```

2. start()

当浏览器载入 Applet 后，将自动调用成员函数 start() 来开始 Applet 的运行。通常，在成员函数 start() 中包含了一个 Applet 的核心代码。例如，下面的代码将播放音频文件。

```

public void start() {
    clip.play();
}

```

3. stop()

当 Applet 停止运行时将自动调用成员函数 stop()，例如，当你离开某个运行 Applet 的 Web 页面时，就可以使用该函数来停止播放音频文件，或终止任何正在执行的线程。

```

public void stop() {
    clip.stop();
}

```

4. destroy()

当 Applet 完成运行后将调用成员函数 destroy()，它将释放由系统分配的资源。由于 Java 本身考虑了“垃圾”处理和内存管理，通常不需要重写函数 destroy()。

6.3.2 图象处理

前面已提到，在浏览器环境运行 Applet 的一个优点，就是可以使用浏览器的图象解码程序来显示图象，这就意味着你不需编写任何特别的代码，即可在 Applet 中使用 GIF 和 JPEG 格式的图象文件。

Java 中用来处理图象的两个主要函数是 getImage() 和 drawImage()。getImage(URL, string) 是 Applet 类的成员函数，它可从指定的 URL 获取一个图象文件，例如：

```

image = getImage(getDocumentBase(), "me.gif");

```

将从函数 getDocumentBase() 指定的主机载入文件 me.gif。其中 getDocumentBase() 是 Applet 的成员函数，它可获得嵌入 Applet 的 HTML 文本的 URL。

为了在屏幕上显示图象，还需要使用函数 drawImage()，它是 Graphics 类的成员函数。

```

// Draw an image on screen
import java.awt.*;
import java.applet.*;
import java.net.URL;

public class Pict extends java.applet.Applet {
    Image image;

    public void init() {
        image = getImage(getDocumentBase(), "me.gif");
    }

    public void paint(Graphics g) {
        g.drawImage(image, 0, 0, this);
    }

    public void start() {
        repaint();
    }
}

```

上面的例子体现了 Applet 中图象处理的一般特点:

- 图象文件在成员函数 `init()` 中使用 `getImage()` 载入, 这可确保图象显示前已开始加载。
- 重写成员函数 `paint()` 以调用 `drawImage()`。当在 `start()` 中调用 `repaint()` 时, 成员函数 `update()` 将被自动调用, 它接着再调用函数 `paint()`。

如果在你的 Applet 中需要使用多个图象, 如何确保执行其它代码前, 所有的图象已正常装载呢? 在 `init()` 中载入图象只是一个好的开端, 但还不够充分。幸运的是 Java 提供了一个 `MediaTracker` 类来完成这种任务。

`MediaTracker` 类使你能建立一个媒体跟踪对象, 用来监视图象加载的状态并当完成任务时通知 Applet。下面让我们来看一下使用 `MediaTracker` 类的必要代码。

```

MediaTracker LoadImage;
Image slides[ ];
int images;

public void init() {
    for (int i=0; i<images; i++) {

```

```

slides[i] = getImage(getDocumentBase(), imagefile + i + "." + type);
LoadImages.addImage(slides[i], 1);
}
try {
LoadImages.waitForID(0);
}
catch (InterruptedException e) {
System.out.println("Image Loading Failed!");
}
showStatus(imagefile + "Images Loaded");
index = 0;
repaint();
LoadImages.checkID(1, true);
}

```

在上面的程序中，生成了一个名为 `LoadImages` 的媒体跟踪对象，它将负责跟踪图象的加载，而数组 `slides[]` 则用来存放载入的图象，并以整型变量 `images` 表示图象的总数。下一步要做的就是使用媒体跟踪对象及函数 `getImage()` 来加载图象，`MediaTracker` 的成员函数 `addImage()` 可把每个图象加到所跟踪的图象表中，然后使用函数 `checkID()` 来检查图象是否已载入。

使用 `MediaTracker` 来监视图象加载状态等于为 Applet 加了一层保护，如果你要设计一个基于图象的动画，或与图形相关的 Applet，最好使用 `MediaTracker`。顾名思义，`MediaTracker` 是为各种媒体，如图形、音频等设计的，但当前实现的 `MediaTracker` 还只支持图象。

6.3.3 声音播放

在 `java.applet` 软件包中，`Applet` 类和 `AudioClip` 接口为声音播放提供了支持。当前，Java API 仅支持一种声音格式：8 位、16 位、8000Hz、单声道的“.au”文件格式，SUN 公司提供了一个制作工具，可将其它格式的声音文件转换为这种格式。

Applet 中主要有以下两类与声音播放有关的成员函数：

- `getAudioClip(URL)` 或 `getAudioClip(URL, String)`
返回一个实现 `AudioClip` 接口的对象。
- `play(URL)` 或 `play(URL, String)`
播放指定 URL 的 `AudioClip`。

`AudioClip` 接口则为声音的播放提供了以下三个函数：

- `play()`
播放一个音频文件直达结束。
- `loop()`

循环播放一个音频文件。

■ stop()

停止播放一个音频文件。

下面我们用一个例子来说明 Applet 中声音播放程序的一般结构。

```
import java.applet.*;
import java.net.URL;

public class PlaySound extends Applet {
    AudioClip sound;
    public void init() {
        sound = getAudioClip(getDocumentBase(), "hi.au");
    }
    public void start() {
        sound.loop();
    }
    public void stop() {
        sound.stop();
    }
}
```

6.3.4 Applet 参数的定义和使用

本章 6.2 中已说明，Applet 嵌入 HTML 文本的一个重要标记符是<param>，它可让用户指定 Applet 中使用的变量的值。当 Applet 需要加载图象或声音文件时，通过<param>标签把文件名传递给 Applet，无疑将使 Applet 更加灵活。

Applet 使用成员函数 getParameter() 来获取用户指定的参数值，它是如下定义的：

■ public String getParameter(String name)

该函数以字符串形式返回参数值，但你可以根据需要将它转换为其它的数据类型。例如，下面的程序行可将获取的参数值转换为整数：

```
int requestedWidth = 0;
...
String windowWidthString = getParameter("WINDOWWIDTH");
if (windowWidthString != null) {
    try {
        requestedWidth = Integer.parseInt(windowWidthString);
    } catch (NumberFormatException e) {
        // Use default width.
    }
}
```

```
}  
}
```

如果用户没有指定参数 WINDOWWIDTH 的值, 就使用缺省值 0, 窗口将以预先设定的大小显示, 因此, 应尽可能为参数提供一个缺省值, 以免程序运行出错。

此外, 还需要在成员函数 `getParameterInfo()` 中说明 Applet 参数的信息, 浏览器可使用这些信息来帮助用户设定 Applet 的参数值。例如, 我们在后面介绍的示例 `SlideShow` 中, 就使用了下面一段程序代码:

```
public class SlideShow extends Applet implements Runnable {  
    String imagefile, soundfile, type, caption;  
    int images, delay;  
  
    public String[ ][ ] getParameterInfo( ) {  
String[ ][ ] info = {  
// Parameter Name    Kind of ValueDescription  
    {"caption", "string", "Title of the Slide Viewer"},  
    {"imagefile",    "string", "Base Image File Name (ex. picture)"},  
    {"soundfile",    "string", "Sound File Name (ex. audio.au)"},  
    {"images", "int",  "Total number of image files"},  
    {"delay",  "int",  "Delay in between images (in ms.)"},  
    {"type",    "string", "Image File Type (gif, jpg, etc)"},  
    };  
    return info;  
    }  
  
    public void init( ) {  
caption = getParameter("caption");  
imagefile = getParameter("imagefile");  
soundfile = getParameter("sounfile");  
type = getParameter("type");  
images = Integer.valueOf(getParameter("images")).intValue( );  
delay = Integer.valueOf(getParameter("delay")).intValue( );  
    }  
}
```

6.3.5 一个 Applet 样本程序: SlideShow

为了更好地掌握前面的内容, 我们以一个比较完整的 Applet 样本程序来说明。`SlideShow` 是一个模仿幻灯片放映的 Java Applet 程序, 它有以下几个用户可设置的参数:

■ caption

caption 是在控制键上方显示的标题，可由用户根据应用设定。

■ imagefile

用于指定所放映的图象文件名。这里多个图象文件的命名应按照一定的规律和顺序，以基本的文件名加一个序号构成，如 image0.gif，image1.gif 等等，但不必给出图象文件个数以及扩展名，它们将由后面的参数说明。

■ soundfile

用于指定所播放的声音文件名，这里要使用文件的全名，如 sound.au。

■ images

用于指定显示的图象总数。

■ delay

自动放映图象的时间间隔。

■ type

用于说明图象文件的类型，如 GIF 或 JPEG 等。

下面是 SlideShow 在 HTML 文件中的嵌入方式，使用 AppletViewer 运行可得到如图 6.3 所示的结果。该 Applet 的界面结构相当简洁明了，它由所显示的图象以及一些控制组成。其中，对号框 “AutoCycle Images” 可让用户选择是否自动放映图象，对号框 “Sound On” 用来控制声音的开关，而 “Previous” 和 “Next” 按钮则可让用户向前或向后切换图象。



图 6.3 SlideShow：一个样本 Applet

<HTML>

```
<applet code="SlideShow.class" width=400 height=250>
```

```
<param name="caption" value="A Sample Photo Album">
```



```

<param name="imagefile" value="image">
<param name="soundfile" value="gan.au">
<param name="images" value="5">
<param name="delay" value="5000">
<param name="type" value="gif">
</applet>
</HTML>

```

SlideShow.java 的全部程序代码如下:

```

/* A Simple Slide Viewer */
import java.awt.*;
import java.lang.*;
import java.applet.*;
import java.net.URL;

public class SlideShow extends Applet implements Runnable {
    MediaTracker LoadImages;
    Image slides[ ];
    String imagefile, soundfile, type, caption;
    int images, delay;
    Thread AutoShow;
    int index = 0;
    Button forward, backward;
    Checkbox auto, sound;
    Label title;
    AudioClip clip;
    Panel marquee, control;

    public String[ ][ ] getParameterInfo( ) {
String[ ][ ] info = {
    {"caption", "string", "Title of the Slide Viewer"},
    {"imagefile", "string", "Base Image File Name (ex. picture)"},
    {"soundfile", "string", "Sound File Name (ex. audio.au)"},
    {"images", "int", "Total number of image files"},
    {"delay", "int", "Delay in between images (in ms.)"},
    {"type", "string", "Image File Type (gif, jpg, etc)"},
};
return info;
}

```

```

    }

    public void init() {
        // Parse the parameters from the HTML file
        LoadImages = new MediaTracker(this);

        caption = getParameter("caption");
        imagefile = getParameter("imagefile");
        soundfile = getParameter("sounfile");
        type = getParameter("type");
        images = Integer.valueOf(getParameter("images")).intValue();
        delay = Integer.valueOf(getParameter("delay")).intValue();

        // Use MediaTracker to load the images

        for(int i=0; i<images; i++) {
            slides[i] = getImage(getDocumentBase(), imagefile + i + "." + type);
            if (i > 0)
                LoadImages.addImage(slides[i], 1);
            else
                LoadImages.addImage(slides[i], 0);
        }
        try {
            LoadImages.waitForID(0);
        } catch (InterruptedException e) {
            System.out.println("Image Loading Failed!");
        }

        showStatus(imagefile + "Images Loaded");
        index = 0;
        repaint();
        LoadImages.checkID(1, true);

        clip = getAudioClip(getDocumentBase(), soundfile);

        // Create the SlideViewer layout
        setLayout(new BorderLayout());

        forward = new Button("Next");
    }

```

```

backward = new Button("Previous");
auto = new Checkbox("AutoCycle Images");
auto.setState(true);
sound = new Checkbox("Sound On");
sound.setState(true);
title = new Label(caption);

Panel marquee = new Panel( );
marquee.setLayout(new BorderLayout( ));
marquee.add("North", title);

Panel control = new Panel( );
control.setLayout(new FlowLayout( ));

control.add(auto);
control.add(sound);
control.add(backward);
control.add(forward);

setFont(new Font("Helvetica", Font.BOLD, 18));
add("South", marquee);
setFont(new Font("Helvetica", Font.PLAIN, 14));
marquee.add("South", control);
}

// Monitor Checkboxes and buttons for actions
public boolean action(Event evt, Object what) {
    if(evt.target == sound) {
        if(sound.getState( ) == true)
            clip.loop( );
        else
            clip.stop( );
        return true;
    }
    else if(evt.target == backward) {
        if(index != 0) {
            index --;
            repaint( );
        }
    }
}

```

```

return true;
}
else if(evt.target == forward) {
index ++;
repaint( );
return true;
} else return false;
}

// Start the animation thread
public void start() {
images = 0;
repaint( );

clip.loop( );
AutoShow = new Thread(this);
AutoShow.start(); // Start this thread will cause the run() method to be called
}

// Stop the animation thread
public void stop() {
index = 0;
repaint( );
if (AutoShow!=null) AutoShow.stop( );
AutoShow = null;
}

public void run() {
Thread running = Thread.currentThread( );
while (AutoShow==running) {
try {
Thread.sleep(delay); // Cause the currently executing thread to sleep some time
} catch (InterruptedException e) {
break;
}
}

// Automatically display the next photo
if(auto.getState( ) == true) {
if(LoadImages.checkID(1, true))

```

```

synchronized(this) {
    if(index == (slides.length - 1))
        index = 0;
    else index++;
}
repaint();
}
}

// Update is called by repaint()
public void update(Graphics g) {
try {
    paint(g);
} catch (ArrayIndexOutOfBoundsException e) {
    if(index < 0)
        index = 0;
    else if(index > images)
        index = images;
    System.out.println("No more images!");
}
}

// Paint the slide image on the screen and account for missing images
public void paint(Graphics g) {
if(LoadImages.isErrorAny()) {
    g.setColor(Color.black);
    g.fillRect(0, 0, size().width, size().height);
    return;
}
g.drawImage(slides[index], 0, 0, this);
}
}

```

6.4 Applet 程序设计方法

对 Java 程序员来说, 在 Applet 编程中难免会遇到这样或那样的错误, 然而, 如果在程序设计上多花一些精力无疑将大大提高编程效率。Applet 程序设计涉及面非常广, 但最主要的还是用户界面和类设计, 这一节我们将以调色板的实现为例来说明 Applet 程序设计

的一些方法。

6.4.1 用户界面设计

用户界面是 Applet 程序设计中非常重要的一环,没有友好的人机界面, Applet 将很难为用户所接受。由于在程序完成后,用户界面的修改比较困难,因而,从程序设计一开始就要构思好用户界面。

调色板的界面由两部分组成:颜色选择区和颜色合成区,如图 6.4 所示。颜色选择区可进行红、绿、蓝 (RGB) 三色比例的调整,而且,在底部有一个标识来显示当前颜色的十六进制表示。 RGB 的值可通过滚动条或文本字符来调整,它们是相互作用的,这样等于为上述三种颜色的调整设计了两个改变当前值的接口:字符输入和滚动条的滑动。

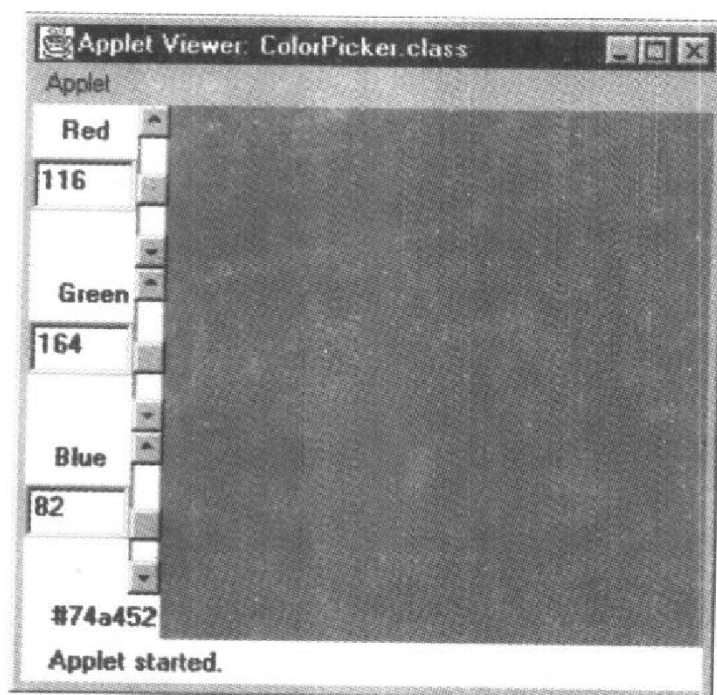


图 6.4 调色板 Applet 用户界面构成

6.4.2 类设计

由于 Java 是面向对象的编程语言,将一个应用程序分成多个类来实现是自然的,然而这将增加 Applet 编程的复制性,因此在 Applet 程序中是否使用多个类要视情况而定。通常,类设计有以下两种趋向:

■ 最小化

采用这种方法总是尽可能减少类的数量,因为每生成一个新类,就要定义该类的实例并与其它类对象交互,这无疑将增加编程的复杂性。但是最小化并不意味着要将所有的功能放在一个类中实现,而是以合适为目标。

■ 完全化

这种方法试图使用多个类来实现一个 Applet 程序,通过类对象的交互来完成需要的功能。虽然从某种意义上来说是增加了复杂性,但类的重用将加快程序开发。

当然,实际设计时可能会采用折衷的方法。如果一个项目比较大,并且其中部分功能可在以后的项目中重用,那就将它分成自然的类结构;如果只是使用一次的小 Applet,最好还是将它简单化。

我们在调色板程序中使用了多个类的实现方法,其主要的类包括: ColorPicker、ColorPanel、ColorSelector、RGBChooser,下面分别予以介绍。

1. ColorPicker 类

ColorPicker 是实现 Applet 本身的类,它将启动构成该 Applet 的其它类对象,其代码如下:

```
public class ColorPicker extends Applet {
    ColorPanel out;
    ColorSelector select;

    public void init() {
        setLayout(new BorderLayout());

        out = new ColorPanel();
        select = new ColorSelector(out, 255, 204, 102);
        add("West", select);
        add("Center", out);
    }
}
```

ColorPicker 类所做的只是定义界面布局,并生成 ColorPanel 和 ColorSelector 类的实例。值得注意的是, ColorPanel 对象的句柄作为构造函数的参数传递给了 ColorSelector 对象,它允许两个对象之间直接通讯。

2. ColorPanel 类

ColorPanel 用来表示调色板当前的颜色,刚好 AWT 提供了一个能很好完成这个任务的类: Panel。唯一要做就是设计一个成员函数来改变当前颜色,如下:

```
class ColorPanel extends Panel {
    void change(Color new_c) {
        setBackground(new_c);
        repaint();
    }
}
```

3. ColorSelector 类

ColorSelector 完成 Applet 的输入功能，它包括生成三个 RGBChooser 类对象和一个以十六进制表示当前颜色的文字标号，并使用 ColorPanel 类对象来改变颜色：

```
class ColorSelector extends Panel {
    ColorPanel controller;
    Label html;
    RGBChooser red, green, blue;

    ColorSelector(ColorPanel myController, int r, int g, int b) {
        super();

        controller = myController;
        setLayout(new BorderLayout());

        Panel controls = new Panel();
        controls.setLayout(new GridLayout(3, 1));
        red = new RGBChooser(this, r, "Red");
        controls.add(red);
        green = new RGBChooser(this, g, "Green");
        controls.add(green);
        blue = new RGBChooser(this, b, "Blue");
        controls.add(blue);
        add("Center", controls);
        html = new Label("#000000");
        html.setBackground(Color.gray);
        add("South", html);

        colorChange();
    }
}
```

上面的程序代码相当简明，应该注意的是在生成 RGBChooser 对象时引用了 ColorSelector 类本身，而且字符标记 HTML 使用了一个假字符串进行构造，以确保 Applet 界面布局时不会把它缩得太小，以致不能完全显示颜色信息。

ColorSelector 类还有一个成员函数 colorChange()，设计它是因为当每种颜色调整时都要作出反应。如果红、绿或蓝色改变，合成色也要改变，这必须马上反映在右边的颜

色合成区中。Applet 可以设计成让用户按下 “update” 键来刷新合成的颜色，但把它设计成每当 RGB 改变时就自动更新合成颜色，显然更能让用户满意。下面的程序代码正是这样做的：

```
void colorChange() {
    Color new_c = getColor();
    int col[] = new int[3];
    StringBuffer text;

    text = new StringBuffer("");
    col[0] = new_c.getRed();
    col[1] = new_c.getGreen();
    col[2] = new_c.getBlue();

    for(int i=0; i<3; i++) {
        if(col[i] < 16)    text.append('0');
        text.append(Integer.toString(col[i], 16));
    }

    controller.change(new_c);
    html.setText("#" + text.toString());
}
```

在上面的程序中首先调用了函数 getColor()，该函数只有如下一行代码。

```
Color getColor() {
    return new Color(red.value(), green.value(), blue.value());
}
```

当然也可以把该行代码直接放到 colorChange() 中，之所以把它提取出来作为一个函数，是因为它表示了该类的一个重要状态，有可能在其它程序中使用。

colorChange() 的其它代码主要完成颜色的十六进制表示，它通过 Color 类相应的成员函数把当前的颜色分成红、绿和蓝，然后，分别转换为十六进字符。最后刷新 ColorPanel 和文字标号以反映颜色的改变。

4. RGBChooser 类

程序中三个 RGBChooser 对象是用户与调色板 Applet 交互的唯一方式，因此应把它设计得尽可能灵活而简明。在该类中有一个表示颜色状态的整数，其值为 0 ~ 255。通过输入整数值来设置颜色值是最自然的，因此设计了一个文本输入域；另外，由移动鼠标来改

变颜色值则可以非常快地浏览各种颜色，而滚动条正好实现这一功能。还是让我们先来看一看该类的定义：

```
class RGBChooser extends Panel {
    ColorSelector controller;
    Scrollbar colorScroll;
    TextField colorField;
    int c_value;

    RGBChooser(ColorSelector myController, int initial, String caption) {
        super();
        controller = myController;

        setLayout(new BorderLayout());

        colorField = new TextField(String.valueOf(initial));
        colorScroll = new Scrollbar(Scrollbar.VERTICAL, 255 - initial, 0, 0, 255);
        set_value(initial);
        add("East", colorScroll);

        Panel temp_panel = new Panel();
        temp_panel.setLayout(new GridLayout(3, 1));
        Label label1 = new Label(caption);
        temp_panel.add(label1);
        temp_panel.add(colorField);
        add("Center", temp_panel);
    }
}
```

在该类中，我们设置了一个包含 RGBChooser 的 ColorSelector 句柄，当某颜色值变化时，RGBChooser 将通知 ColorSelector 对象。此外，RGBChooser 还提供了如下两个接口函数，来获取或修改当前的颜色值：

```
int value() {
    return c_value;
}

void set_value(int initial) {
    c_value = initial;
}
```

```
}
```

RGBChooser 类中事件处理是比较重要的部分, 这里我们使用了一种比较原始的事件处理方法, 其代码如下:

```
public boolean handleEvent(Event evt) {
    Integer in;

    if(evt.target == colorScroll) {
        in = (Integer)evt.arg;
        colorField.setText(String.valueOf(255 - in.intValue()));
        set_value(255 - in.intValue());
        controller.colorChange();
        return true;
    }
    else if(evt.target == colorField) {
        String tmp;
        tmp = (String)evt.arg;
        in = Integer.valueOf(tmp);
        set_value(in.intValue());
        colorScroll.setValue(-1 * (in.intValue() - 255));
        controller.colorChange();
        return true;
    }
    else {
        return super.handleEvent(evt);
    }
}
```

首先, 如果移动了滚动条, 就由事件参数 evt 获取当前值, 并修改 RGBChooser 对象的颜色值; 其次, 如果用户改变了文本输入域的值, 就将该值转换为整数并在 RGBChooser 对象中设置。随后, 通知 ColorSelector 刷新合成的颜色。

上述两种情况是相互作用的, 如果用户移动了滚动条, 相应的文本值也将改变, 反之亦然。这就避免了两种输入方法的差别, 维护了数据的一致性。

6.4.3 Applet 的完成

到目前为止, 已基本完成了调色板 Applet 程序。它采用了多个类的 Applet 实现方法,

充分反映了面向对象编程的特点：如果在项目开始阶段即进行了合理的设计，正确地实现了每部分的功能，整个 Applet 程序也将非常完善。

下面我们列出了调色板 Applet 程序的全部代码：

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

public class ColorPicker extends Applet {
    ColorPanel out;
    ColorSelector select;

    public void init() {
        setLayout(new BorderLayout()); // define layout

        // Create an instance of a ColorPanel and ColorSelector
        out = new ColorPanel();
        select = new ColorSelector(out, 255, 204, 102);
        add("West", select);
        add("Center", out);
    }
}

class ColorPanel extends Panel {
    void change(Color new_c) {
        setBackground(new_c);
        repaint();
    }
}

// Complete the input portion of the applet
class ColorSelector extends Panel {
    ColorPanel controller;
    Label html; // representation of the current color
    RGBChooser red, green, blue;

    ColorSelector(ColorPanel myController, int r, int g, int b) {
        super();
    }
}
```

```

controller = myController;
setLayout(new BorderLayout ( ));

// Construct three instance of RGBChooser class
Panel controls = new Panel ( );
controls.setLayout(new GridLayout(3, 1));
red = new RGBChooser(this, r, "Red");
controls.add(red);
green = new RGBChooser(this, g, "Green");
controls.add(green);
blue = new RGBChooser(this, b, "Blue");
controls.add(blue);
add("Center", controls);
html = new Label("#000000"); // constructed using a dummy string
html.setBackground(Color.gray);
add("South", html);

colorChange ( );
}

void colorChange ( ) {
Color new_c = getColor ( );
int col[ ] = new int[3];
StringBuffer text;

// The current color is broken down into red, green and blue
text = new StringBuffer("");
col[0] = new_c.getRed ( );
col[1] = new_c.getGreen ( );
col[2] = new_c.getBlue ( );

for(int i=0; i<3; i++) {
    if(col[i] < 16)    text.append('0');
    text.append(Integer.toString(col[i], 16));
                        // convert to a hex string representation
}

controller.change(new_c);
html.setText("#" + text.toString ( ));

```

```

    }

    Color getColor() {
return new Color(red.value(), green.value(), blue.value());
    }
}

class RGBChooser extends Panel {
    ColorSelector controller; // set up a handle to contain the RGBChooser
    Scrollbar colorScroll;
    TextField colorField;
    int c_value;

    RGBChooser(ColorSelector myController, int initial, String caption) {
super();
controller = myController;

setLayout(new BorderLayout());

// Set up the TextField and Scrollbar
colorField = new TextField(String.valueOf(initial));
colorScroll = new Scrollbar(Scrollbar.VERTICAL, 255 - initial, 0, 0, 255);
set_value(initial);
add("East", colorScroll);

Panel temp_panel = new Panel();
temp_panel.setLayout(new GridLayout(3, 1));
Label label1 = new Label(caption);
temp_panel.add(label1);
temp_panel.add(colorField);
add("Center", temp_panel);
    }

    public boolean handleEvent(Event evt) {
Integer in;

if(evt.target == colorScroll) { // the scrollbar has been moved
    in = (Integer)evt.arg;
    colorField.setText(String.valueOf(255 - in.intValue()));
}
}

```

```

        set_value(255 - in.intValue());
        controller.colorChange();
        return true;
    }
    else if(evt.target == colorField) {
        // the value of the ColorField has been changed
        String tmp;
        tmp = (String)evt.arg;
        in = Integer.valueOf(tmp);
        set_value(in.intValue());
        // Change the positon of slide on the scrollbar
        colorScroll.setValue(-1 * (in.intValue() - 255));
        controller.colorChange();
        return true;
    }
    else {
        return super.handleEvent(evt); // default event handler
    }
}

int value() {
    return c_value;
}

void set_value(int initial) {
    c_value = initial;
}
}

```

调色板 Applet 在 HTML 文件中的嵌入格式如下, 使用 AppletViewer 运行将在屏幕上显示如图 6.4 所示的 Applet, 你可以通过移动滚动条或输入数值改变右边合成的颜色。

```

<applet code= "ColorPicker" width=400 height=300 alt="Color Picker">
</applet>

```

6.5 Applet 之间的通讯

目前, Applet 之间的通讯是有一定的安全性限制的。首先, 它们必须是运行在浏览器同一页面的 Applet, 其次, 它们应该是来自同一服务器的 Applet。

AppletContext 接口为 Applet 通讯提供了两个成员函数 getApplet(String) 和 getApplets(), 其中, 前者返回指定名字的 Applet 对象, 后者返回当前运行的所有 Applet 对象。Applet 使用这两个函数, 可以找到在同一 Web 页面上的其它 Applet 对象, 并调用其成员函数来交互信息。

下面我们用一个例子来说明 Applet 之间是如何交互信息的, 在这个例子中, 有两个 Applet 程序, 一个发送者, 一个接收者。首先, 发送者查找接收者, 找到后就调用接收者的成员函数, 把发送者的名字作为参数传给接收者, 而接收者则显示一条信息 “Received message from Sender name!”, 如图 6.5 所示。

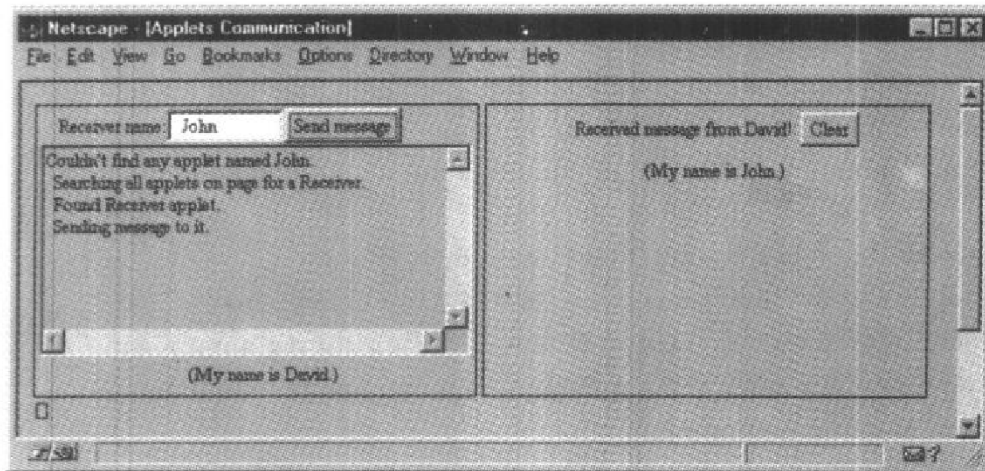


图 6.5 Netscape 中两个 Applet 之间的通讯

这里需要指出的是, Netscape Navigator 3.0 以下的浏览器不支持查找指定名字的 Applet, 因而会显示一条信息: “Couldn't find any applet named John”。但在发送者程序中可以处理这种情况, 它将调用 getApplets() 函数查找所有的 Applet 对象, 并向接收者发送信息。

下面我们分别列出发送者和接收者的程序, 发送者的程序代码如下:

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

public class Sender extends Applet {
    private String myName;
    private TextField nameField;
    private TextArea status;

    public void init() {
        GridBagLayout gridBag = new GridBagLayout();
```



```

GridBagConstraints c = new GridBagConstraints();

setLayout(gridBag);    // set the layout manager

Label receiverLabel = new Label("Receiver name: ", Label.RIGHT);
gridBag.setConstraints(receiverLabel, c);
    // set the constraints for the Label
add(receiverLabel);

nameField = new TextField(getParameter("RECEIVERNAME"), 10);
c.fill = GridBagConstraints.HORIZONTAL;
gridBag.setConstraints(nameField, c);
add(nameField);

Button button = new Button("Send message");
c.gridwidth = GridBagConstraints.REMAINDER;    //end row
c.anchor = GridBagConstraints.WEST;    //stick to the text field
c.fill = GridBagConstraints.NONE;    //keep the button small
gridBag.setConstraints(button, c);
add(button);

status = new TextArea(5, 60);
status.setEditable(false);
c.anchor = GridBagConstraints.CENTER;    //reset to the default
c.fill = GridBagConstraints.BOTH;    //make this big
c.weightx = 1.0;
c.weighty = 1.0;
gridBag.setConstraints(status, c);
add(status);

myName = getParameter("NAME");
Label senderLabel = new Label(" (My name is " + myName + ".) ",
Label.CENTER);
c.weightx = 0.0;
c.weighty = 0.0;
gridBag.setConstraints(senderLabel, c);
add(senderLabel);

validate();

```

```

}

    public boolean action(Event event, Object o) {
Applet receiver = null;
String receiverName = nameField.getText();
receiver = getAppletContext().getApplet(receiverName);
        // search the specified receiver
if (receiver != null) {
if (!(receiver instanceof Receiver)) {
status.appendText("Found applet named "
    + receiverName + ", "
    + "but it's not a Receiver object.\n");
} else {
status.appendText("Found applet named "
    + receiverName + ".\n"
    + " Sending message to it.\n");
((Receiver)receiver).processRequestFrom(myName);
    // call the method of Receiver
}
} else {
status.appendText("Couldn't find any applet named "
    + receiverName + ".\n"
    + " Searching all applets on page "
    + "for a Receiver.\n");
searchAllApplets(); // search all applets at the same web page
}
status.selectAll();
int n = status.getSelectionEnd();
status.select(n,n);
return false;
}

    public Insets insets() {
return new Insets(3, 3, 3, 3);
}

    public void paint(Graphics g) {

```

```

g.drawRect(0, 0, size().width - 1, size().height - 1);
}
public String getAppletInfo() {
return "Sender by David Gan";
}

    public void searchAllApplets() {
Enumeration appletList = getAppletContext().getApplets();
boolean foundReceiver = false;

// Search the receiver from the applet list
while (appletList.hasMoreElements() && !foundReceiver) {
Applet applet = (Applet)appletList.nextElement();
if (applet instanceof Receiver) {
status.appendText(" Found Receiver applet.\n"
+ " Sending message to it.\n");

((Receiver)applet).processRequestFrom(myName);
}
}
}
}

```

接收者的程序代码如下:

```

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

public class Receiver extends Applet {
    private final String waitingMessage = "Waiting for a message... ";
    private Label label = new Label(waitingMessage, Label.RIGHT);
    public void init() {
add(label);
add(new Button("Clear"));
add(new Label(" (My name is " + getParameter("name") + ".)",
    Label.LEFT));
validate();
}
}

```

```

    public boolean action(Event event, Object o) {
        label.setText(waitingMessage);
        repaint();
        return false;
    }

    // This method will called by the Sender applet
    public void processRequestFrom(String senderName) {
        label.setText("Received message from " + senderName + "!");
        repaint();
    }

    public void paint(Graphics g) {
        g.drawRect(0, 0, size().width - 1, size().height - 1);
    }

    public String getAppletInfo() {
        return "Receiver (named " + getParameter("name") + ") by David Gae";
    }
}

```

这个示例说明了 Applet 从发送者到接收者的单向通讯。如果接收者也要向发送者传递信息，可以在接收者程序中增加一个成员函数 `startCommunicating()`，由发送者按如下的语句调用该函数，将自身的引用传递给接收者，从而接收者也可以调用发送者的成员函数。

```
((Receiver) receiver).startCommunicating(this);
```

6.6 Applet 的局限性

本章 6.1 节已论述了 Applet 的执行过程，它是事先编译成字节码，然后通过网络下载到本地机器上运行的。从本质上来说，它是存在安全性问题的，因而 SUN 公司及一些 WWW 浏览器制造商对 Applet 的功能作了一些限制。这些限制虽然在某种程度上影响了 Applet 程序的开发，但它也确保了运行 Applet 的机器的安全性，有利于 Applet 在 Internet 上被

广泛地接受。

6.6.1 Applet 本身的功能限制

Applet 是从服务器上按字节码下载的，由于字节码与平台无关，并且它是通过 AppletViewer 或支持 Java 的 WWW 浏览器来运行的，这就使得 Java 程序很容易在各种平台上移植。考虑到 Applet 将在用户本机上运行，因而需要一定的安全机制来防止机器受到损害。

Java 实现安全的方法之一，是采用检验进程来确保字节码没有违反 Java 语言规范。为了进行检验，所有的函数和变量都要以名字调用，通过对字节码中的函数和变量的检查，可避免 Applet 访问操作系统或其它应用程序的内存区。这种限制也取消了内存指针的处理，而指针正是 C 或 C++ 等语言中最令人头痛的部分。当然，取消指针也不是没有代价的，与其它编程语言相比，Java 这种按名字实现函数和变量的方式降低了 Applet 的运行速度。

Java 实现安全的方法之二，是不允许 Applet 对本机的文件系统进行完全访问。目前，Applet 既不能对本机写文件，也不能从本机读文件。虽然这明显地限制了 Applet 的功能，但允许 Applet 访问本机文件系统，无疑将会成为一个严重的安全漏洞。因为，如果 Applet 可以读本机的文件系统，它就可以把这些文件的信息发回给 Applet 的作者；如果 Applet 可以在本机上写文件，它就有可能成为病毒的携带者。毫无疑问，文件访问将增加 Applet 的功能，并且 Java 的最终目标也是允许 Applet 读写本机文件系统，但在实现更完善的安全方法以前，还是要限制 Applet 访问文件系统。

Java 实现安全的方法之三，是限制 Applet 载入其它编程语言的动态或共享库。在第十章中我们将介绍，Java 可以将函数声明为原生的（native），让 Java 虚拟机使用其它语言如 C 或 C++ 的库。然而，由于无法对这些库进行检验，而它们有可能要求访问文件系统，因而在 Applet 中不允许使用动态链接库。Java 的目标是尽可能完全用 Java 语言来编写这些动态库，这样就可以不依赖于其它语言，随着 Java 语言的发展，许多限制都将会不存在。

6.6.2 浏览器对 Applet 的限制

除了 Java 语言自身的限制，执行 Applet 字节码的浏览器对 Applet 也有一些限制。Web 浏览器对用户来说是可信赖的应用，它不会破坏用户的文件和操作系统，而 Applet 是不可信的应用，它是在用户毫无意识的情况下下载到本机上运行的。为了维护系统的一致性，支持 Java 的浏览器对 Applet 作了一些限制。

浏览器对 Applet 最重要的限制是网络联接，目前，Applet 可以和它下载的 Web 服务器直接通讯，但不能和网上的其它服务器联接。网络互联最令人头痛的是安全问题，限制 Applet 仅可访问它的主机，将使安全风险最小，因为如果允许 Applet 访问网上的其它服务器，它就有可能执行一些破坏性的任务。本书第八章将进一步讨论网络安全问题。

6.7 小 结

本章比较全面地阐述了 Applet 的编程技术，从简单编程到复杂应用，由浅入深。并以调色板 Applet 程序为例，系统地介绍了 Applet 程序的设计方法。考虑到 Applet 应用的安全性问题，本章最后还讨论了目前对 Applet 的一些功能限制。

Applet 是非常重要的一类 Java 程序，在 Internet 上应用越来越广泛。学习 Applet 编程可以从范例入手，从而掌握 Applet 程序的一般结构，为此，本章列举了几个典型的 Applet 应用程序，希望读者能仔细揣摩，获得一些有益的经验。

第七章 多线程程序设计

支持多线程编程是 Java 的一个重要特征。在本章中，我们要介绍线程的概念、特点和 Java 线程类的属性和可用的操作函数。在此基础上，介绍两类线程编程应用方法。最后，介绍线程的优先级、线程组和多线程的同步控制和防止“死锁”（Deadlock）等问题。

7.1 概述

7.1.1 什么是线程？

线程是一个程序中的一个顺序控制流，有时也称“轻量级的进程”（Light-weight process）或“执行上下文”（Execution context）。每个进程都有一个起点和终点，在执行过程中的每个时刻仅有一个代码指令执行。但是，它不能以独立程序的形式存在，而只能在一个程序中运行。一个程序中可能有两个或多个线程在运行，达到多任务并发的效果。

HotJava 浏览器就是一个很好的多线程应用的例子。在 HotJava 浏览器中你可以在下载 Java 小程序或图象的时候滚动页面，在访问新页面时，播放动画和声音，打印文件。

7.1.2 Java 的线程类

Java 的 Thread 类是从 java.lang.Object 直接继承的，同时按 java.lang.Runnable 界面（interface）规定的接口实现的。

```
java.lang.Object
    |----- java.lang.Thread
```

它有三个成员变量：

- public final static int MAX_PRIORITY;
- public final static int MIN_PRIORITY;
- public final static int NORM_PRIORITY;

它们分别代表了该进程的最大、最小和正常优先级。优先级的范围是从 1 到 10，值越小表明优先级越低。正常优先级是 5，它是大多数线程的缺省值。

Thread 类有 7 个构造函数，分别适用于不同的情况。还有三十个左右的操作函数，分别用来设置、读取线程的状态信息，启动、暂停、停止线程等，其中较重要的有如下的几个：

- public void run()

这是 Thread 类的主体，就好象是独立应用中的 main()。如果我们从 Thread 类衍

生新类的话，必须要把它覆盖。改写的内容是新线程的程序控制流。如果它退出，则该新线程也就结束了。因此，一般设计时作成循环的形式。

■ `public void start()`

它是在构造了新线程对象以后，用它来激活该线程。它在调用 `run()` 之后马上返回，否则我们可以直接调用 `run()`。

例如：

```
myThread = new MyThread();  
myThread.start();
```

■ `public static void sleep(long millis)`

在程序中，如果我们想让某个线程的执行暂停若干毫秒，就可以调用这个操作函数。在暂停期间，其他程序正常执行。

■ `public void suspend()` 和 `resume()`

如果我们不知道要让线程停止执行多长时间，就应调用 `suspend()`，让线程挂起。在某一个条件满足时，调用 `resume()` 让线程重新执行。

■ `public final void stop()`

它的作用与 `suspend()` 相当，但一旦调用了就不能象 `suspend()` 那样能重新恢复执行。`stop()` 调用后产生一个 `ThreadDeath` 对象传给线程来结束它。所以，调用 `stop()` 和线程真正被结束之间还有很短的时间差。

还有其他一些操作函数请读者参考类库手册，具体地了解其功能和参数集。

7.1.3 线程的状态迁移

图 7.1 列出了 Java 的线程生存期中在每个时刻可能的状态和导致状态迁移的操作函数。它是一个概要图，没有列出所有的状态，但给我们一个线程生命期的总体概念。

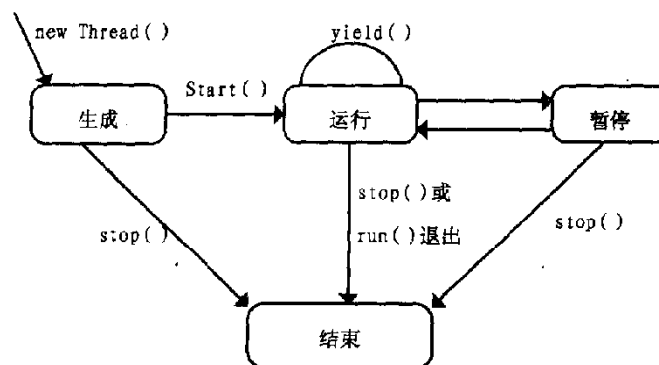


图 7.1 线程的状态转换图

线程在生命期中有四个重要的状态，分别介绍如下：

- 生成 (New thread) 状态。它是调用了 new 生成一个新的对象后的状态。


```
myThread = new MyThread();
```

当线程在这个状态时，它仅仅是一个空的线程对象，系统并没有为它分配资源。所以，系统在这个状态时，只能调用 `start()` 或 `stop()` 来启动或停止它。任何其他的操作都没有意义，并可能会产生一个 `IllegalThreadStateException` 出错对象。

■ 运行 (Runnable) 状态。如果在第一个状态时调用了 `start()`，就进入了运行期。

```
myThread.start();
```

在运行期中，线程按照 `run()` 中的指令顺序执行。不同的线程按优先级大小抢占 CPU 资源。

■ 结束 (Dead) 状态。要结束线程，有两种可能性：“自然死亡”，或者是“强制结束”。前一种情况是当线程中的 `run()` 操作函数执行结束，自然退出。例如，下面的例子中 `run()` 在打印从 0 到 999 之后退出，线程也自然结束。

```
public void run() {  
    int i;  
    for(i =0; i <1000; i ++)  
        System.out.println("\ni = " + i);  
}
```

如果我们要强制结束线程，随时可以调用 `stop()`。下面的例子中的线程在睡眠 10 秒钟后，被强制结束。

```
MyThread myThread = new MyThread();  
myThread.start();  
try {  
    Thread.currentThread().sleep(10000);  
} catch (InterruptedException e) {  
}  
myThread.stop();
```

在支持 Java 小程序的浏览器中，当用户换页时，Java 小程序经常采用这种方法来结束它所有的线程。

■ 暂停 (Not runnable) 状态。当如下的情形之一发生时，线程进入此状态。

- 1) 调用了 `suspend()`
- 2) 调用了 `sleep()` 使线程睡眠一段定长的时间。
- 3) 调用了 `wait()` 使线程等待某一个条件变量。
- 4) 因为系统输入输出而使线程执行被堵塞。

退出暂停状态的方法必须与进入的方法相对应。例如，我们调用了 `sleep(10000)` 让线程睡眠 10 秒钟，当时间未到，即使调用 `resume()` 也不能激活线程。针对上面的入口，相应的出口条件是：

- 1) 调用 `resume()` 恢复线程执行。
- 2) 等待设定的时间到。
- 3) 拥有条件变量的对象调用 `notify()` 或 `notifyAll()`。
- 4) 使得线程堵塞的系统输入输出结束。

7.1.4 “精灵”线程 (Daemon Thread)

Java 中的线程可以设计成“精灵线程”。“精灵”线程为同一个程序中的其他线程提供服务。比如，在 HotJava 浏览器中就有一个称为后台图象再现器 (Background Image Reader) 的“精灵”线程，它可以为任何线程从文件系统或网络上读取图象。

“精灵”线程一般是独立的线程，它的 `run()` 操作函数往往是一个无限循环。当应用中仅仅剩下“精灵”线程时，Java 解释器会自动退出，因为已经没有什么服务对象。编程时用 `setDaemon(boolean)` 来设定一个线程是“精灵”线程，用 `isDaemon()` 来判断一个线程是否“精灵”线程。例如，

```
...
setDaemon(true);
if(isDaemon()) System.out.println("This is a daemon thread");
```

7.2 两类编程方法

根据线程继承方法的不同，可以把实现方法分成两类。但两类的主要编程工作都是改写 `run()` 操作。

- 方法之一：从 `Thread` 类直接引伸出新的线程类。如下例中的 `FruitThread` 类，用它五次打印本线程的名字。

```
class FruitThread extends Thread {
    public FruitThread(String str) {
        //调用父类的构造函数，给线程命名
        super(str);
    }

    public void run() {
        for (int i = 0; i<=5; i++) {
            //打印本线程的名字
            System.out.println(i + " " + getName());
            try {
```

```

        sleep((int) (Math.random() * 1000));
    } catch (InterruptedException e) {}
}
}

class TwoFruits {
    public static void main (String args[]) {
        //生成两个线程，并把它们运行起来
        new FruitThread("苹果").start();
        new FruitThread("香蕉").start();
    }
}

```

下面是运行的结果:

```

0 苹果
0 香蕉
1 苹果
2 苹果
3 苹果
1 香蕉
4 苹果
5 苹果
2 香蕉
3 香蕉
4 香蕉
5 香蕉

```

■ 方法之二: 实现接口。下面的时钟类就是一个这样的例子。它是一个 Java 嵌入小程序，在浏览器的屏幕上每隔一秒钟，刷新一次时间显示。

```

//引入图形、颜色和日期库
import java.awt.Graphics;
import java.awt.Color;
import java.util.Date;

public class Clock extends java.applet.Applet implements Runnable{
    Thread clockThread;

    public void start() {

```

```

        if (clockThread == null) {
            clockThread = new Thread(this, "DigitalClock");
            clockThread.start();
        }
    }

    public void run() {
        while (clockThread != null) {
            //刷新屏幕显示, repaint()会自动调用 paint()
            repaint();
            try {
                //时钟线程睡眠 1 秒钟
                clockThread.sleep(1000);
            } catch (InterruptedException e) {
            }
        }
    }

    public void paint(Graphics g) {
        //创建时间对象和红色对象
        Date now = new Date();
        Color red = new Color(255, 0, 0);
        //安装红色对象
        g.setColor(red);
        //在屏幕上打印当前的时分秒, 并用冒号隔开
        g.drawString(now.getHours()+"-"+now.getMinutes()+"-"+now.getSeconds(),5,10);
    }

    public void stop() {
        //停止时钟线程并把对象变量复位
        clockThread.stop();
        clockThread = null;
    }
}

```

编译上述的程序后, 再编写如下的 HTML 页面, 将会有图 7.2 所示的图形。

```

<html>
<head>
    <title>数字时钟</title>
    <meta name="Author" content="Zunhe JIN">
    <meta name="GENERATOR" content="Mozilla/2.01Gold (Win32)">

```

```

</head>
<body>
<h1>数字时钟显示</h1>
<p>
<hr width="100%" ></p>
<h2><applet code=Clock.class width=200 height=100></applet></h2>
</body>
</html>

```

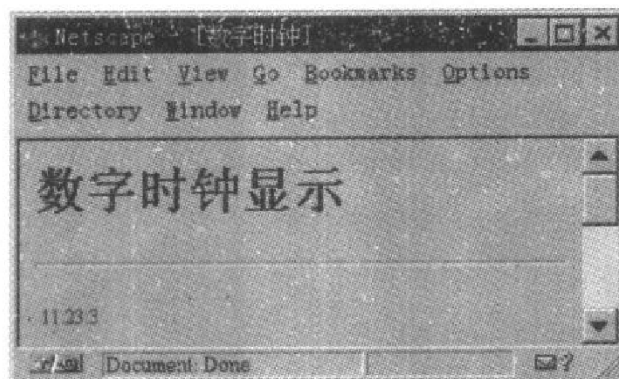


图 7.2 时钟测试页面显示

7.3 线程的优先级调度

线程从概念上讲，是并发地执行的。但在实际上，很多的计算机只有一个 CPU，在某一个时刻只能有一个线程在运行。为了使得多个线程看起来好象是在同时运行，操作系统要将 CPU 的时间分成若干个时间片，并进行合理的分配调度。Java 程序在调度时采用的是很简单的确定性优先级调度（fixed priority scheduling）方法。它的规则是：在任何时刻，优先级高的处于运行状态的线程先执行。

下面的例子是一个 Java 嵌入小程序，它生成相同优先级或不同优先级的两个线程 Runner，这两个线程分别进行累加计算，并把累加计算的结果及时地显示在页面上。其中的 Runner 特别简单，它把一个公共变量反复累加 400,000 次。显示线程则把两个 Runner 中的公共变量动态地显示在屏幕上。

```

//引入所有的图形库类
import java.awt.*;

public class RaceApplet extends java.applet.Applet implements Runnable {
    final static int NUMRUNNERS = 2; //线程数量
    final static int SPACING = 20; //显示刷新线间隔
    //创建两个线程变量数组

```

```

    Runner runners[] = new Runner[NUMRUNNERS];
    //申明显示更新线程
    Thread updateThread;

    public void init() {
        //读入 HTML 文件种的 type 参数
        String raceType = getParameter("type");
        for (int i = 0; i < NUMRUNNERS; i++) {
            //创建线程，并赋给相应的线程变量
            runners[i] = new Runner();
            //如果 type 参数为"unfair", 则设置不同样的优先级，否则同样优先级
            if (raceType.compareTo("unfair") == 0)
                runners[i].setPriority(i+1);
            else
                runners[i].setPriority(2);
        }
        //如果还没有创建显示更新线程，就创建一个，并设置较高的优先级
        if (updateThread == null) {
            updateThread = new Thread(this, "Thread Race");
            updateThread.setPriority(NUMRUNNERS+1);
        }
    }

    //处理鼠标事件
    public boolean mouseDown(java.awt.Event evt, int x, int y) {
        //如果还没有启动三个线程，则启动它们
        if (!updateThread.isAlive())
            updateThread.start();
        for (int i = 0; i < NUMRUNNERS; i++) {
            if (!runners[i].isAlive())
                runners[i].start();
        }
        return true;
    }

    public void paint(Graphics g) {
        //设置颜色为浅灰色，并用它填充一个矩形
        g.setColor(Color.lightGray);
        g.fillRect(0, 0, size().width, size().height);
        //设置颜色为黑色
    }

```

```

        g.setColor(Color.black);
        //读取线程的优先级，并把它显示在最左侧作为提示
        for (int i = 0; i < NUMRUNNERS; i++) {
            int pri = runners[i].getPriority();
            g.drawString(new Integer(pri).toString(), 0, (i+1)*SPACING);
        }
        update(g);
    }

    public void update(Graphics g) {
        //显示两个 runner 线程的运行进度
        for (int i = 0; i < NUMRUNNERS; i++) {
            g.drawLine(SPACING, (i+1)*SPACING, SPACING +
                (runners[i].tick)/1000, (i+1)*SPACING);
        }
    }

    public void run() {
        while (updateThread != null) {
            //repaint() 自动调用 paint() 刷新屏幕
            repaint();
            try {
                //刷新显示线程睡眠 0.01 秒，参数以千分之一秒为单位
                updateThread.sleep(10);
            } catch (InterruptedException e) {
            }
        }
    }

    public void stop() {
        //如果三个线程还“活着”，就分别关闭它们
        for (int i = 0; i < NUMRUNNERS; i++) {
            if (runners[i].isAlive()) {
                runners[i].stop();
                runners[i] = null;
            }
        }
        if (updateThread.isAlive()) {
            updateThread.stop();
        }
    }

```

```

        updateThread = null;
    }
}

class Runner extends Thread {
    public int tick = 1;
    public void run() {
        while (tick < 400000) {
            tick++;
        }
    }
}

```

上面的程序在编译后，与下面的 HTML 文件放在一个目录中。

```

<html>
<head>
<title>MultithreadTest</title>
<meta name="Author" content="Zunhe JIN">
<meta name="GENERATOR" content="Mozilla/2.01Gold (Win32)">
<meta name="Description" content="Multithreaded test of different priority">
</head>
<body>
<h1>线程优先级试验程序</h1>
<p><hr size=4 width="100%" ></p>
<h2>请点击下面的 Java 嵌入小程序</h2>
<p>(不同优先级的线程) </p>
<p><applet code=RaceApplet.class width=500 height=50>
<param name="type" value="unfair"></applet></p>
<p>(相同优先级的线程) </p>
<p><applet code=RaceApplet.class width=500 height=50>
<param name="type" value="fair"></applet></p>
</body>
</html>

```

下面是调用了上面的 HTML 文件后，分别迅速地在下图画线的两个区域点击鼠标，则有如下的屏幕显示。经过一段时间后，所有的线都会平齐。我们图 7.3 是程序运行过程中截获下来的。因为上下的区域激活的时间不同，在此我们不作比较。在每个区域中可以清楚地看到，不同优先级的两个线程和相同优先级的两个线程的运行情况。

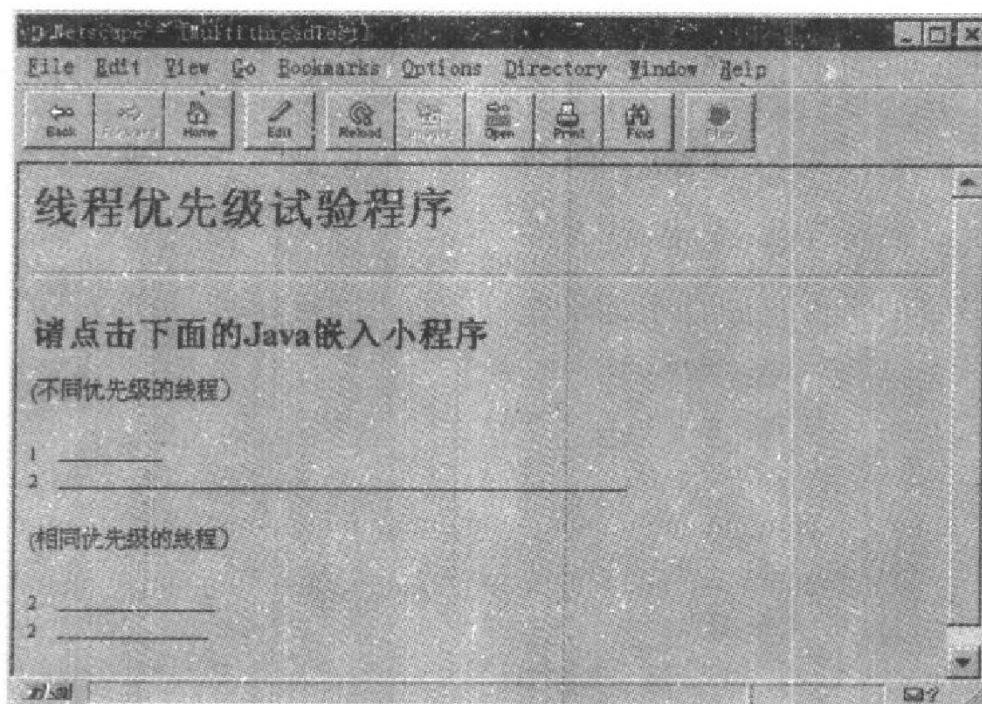


图7.3 线程优先级试验程序的屏幕输出

对于相同优先级的线程，在不同的操作系统上可能有不同的结果。有的可能是一个线程在抢占到 CPU 资源后就一直运行，不会“自愿”放弃，其他的线程只能等它运行完成后在抢占 CPU 资源。还有一种可能是系统采用“时间片”的办法在相同优先级的可运行线程轮转，这样相同优先级的线程都有机会运行。Runner 对象中的 run() 中是一个很长的循环，如果在非时间片的系统上运行，可能会是另外一种结果。

下面的例子中可以帮助我们更清楚地看这个问题。在这个例子中，我们直接在线程中打印两个 **SelfishRunner** 对象的运行信息。

```
class RaceTest {

    final static int NUMRUNNERS = 2;

    public static void main(String args[]) {
        SelfishRunner runners[] = new SelfishRunner[NUMRUNNERS];

        for (int i = 0; i < NUMRUNNERS; i++) {
            runners[i] = new SelfishRunner(i);
            runners[i].setPriority(2);
        }
        for (int i = 0; i < NUMRUNNERS; i++) {
            runners[i].start();
        }
    }
}
```

```

    }
}

class SelfishRunner extends Thread {

    public int tick = 1;
    public int num;

    SelfishRunner(int num) {
        this.num = num;
    }

    public void run() {
        while (tick < 400000) {
            tick++;
            if ((tick % 100000) == 0) {
                System.out.println("Thread #" + num + ", tick = " + tick);
            }
        }
    }
}

```

编译后在命令行键入如下的命令, 屏幕结果如下:

```

C:\java\MyProgram>java RaceTest
Thread #0, tick = 100000
Thread #1, tick = 100000
Thread #1, tick = 200000
Thread #0, tick = 200000
Thread #1, tick = 300000
Thread #0, tick = 300000
Thread #1, tick = 400000
Thread #0, tick = 400000

```

如果在一个非“时间片”轮回的系统上, 结果可能是:

```

Thread #0, tick = 100000
Thread #0, tick = 200000

```

```
Thread #0, tick = 300000
Thread #0, tick = 400000
Thread #1, tick = 100000
Thread #1, tick = 200000
Thread #1, tick = 300000
Thread #1, tick = 400000
```

针对这种情形，我们可以在编程上作一些改进。在线程的循环中，用线程的 `yield()` 来主动放弃 CPU 资源，该操作函数仅仅使相同优先级的线程有机会抢占 CPU 资源，优先级低的则没有机会。我们将上面的例子修改如下：

```
class RaceTest {

    final static int NUMRUNNERS = 2;

    public static void main(String args[]) {
        //产生 NUMRUNNERS 维 GoodRunner 对象数组
        GoodRunner runners[] = new GoodRunner[NUMRUNNERS];
        for (int i = 0; i < NUMRUNNERS; i++) {
            runners[i] = new GoodRunner(i); //逐个生成 GoodRunner 对象
            runners[i].setPriority(2); //赋予每个对象以相同的优先级 2
        }
        for (int i = 0; i < NUMRUNNERS; i++) {
            runners[i].start();
        }
    }
}

class GoodRunner extends Thread {
    public int tick = 1; //时间片累计计数器
    public int num;      //线程标志号

    GoodRunner(int num) {
        this.num = num;
    }

    public void run() {
        while (tick < 400000) {
            tick++;
        }
    }
}
```

```

        if ((tick % 100000) == 0) {
            System.out.println("Thread #" + num + ", tick = " + tick);
            yield();    // 主动放弃CPU资源
        }
    }
}
}

```

为了保证 Java 应用程序有良好的运行机制，我们建议读者在编程时采用 `yield()` 来主动放弃对 CPU 的独占，这样才能真正保证应用在不同的平台上具有相同的运行效果。

7.4 线程组

`ThreadGroup` 类可以管理 Java 应用中的一组线程。一个线程组可以容纳任意数量的相关线程，比如，用同一个类生成的线程，同一个时间起停的线程等等。更进一步，线程组还可以包容线程组。在 Java 应用中，最顶层的线程组是“main”程序组。我们可以在“main”线程组和“main”的子线程组中建立线程或线程组。结果可能形成如图 7.4 所示的一种树状的层次结构。

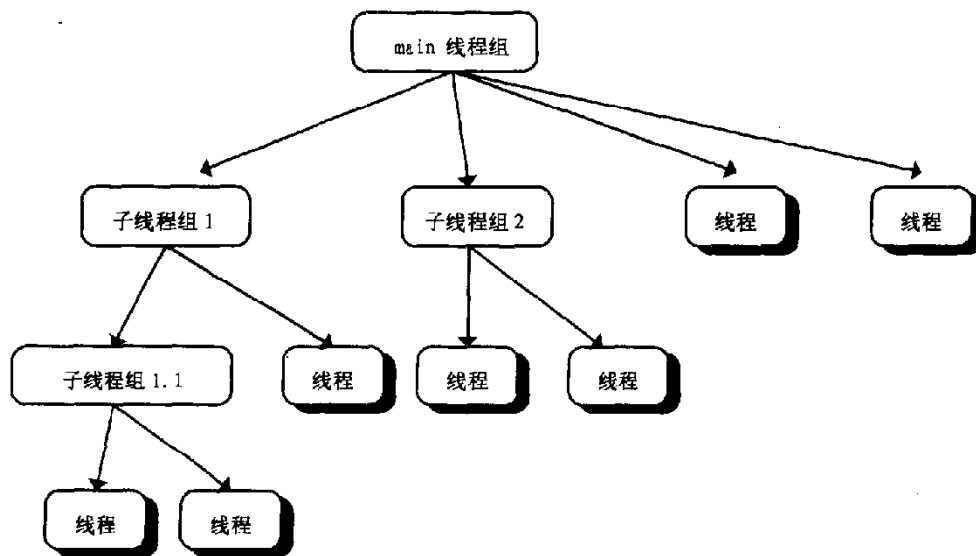


图 7.4 Java 线程组的树状结构

`ThreadGroup` 类的操作函数可以分为如下的几类：

- 集管理性操作函数（Collection Management Method）。它可以用来管理线程组内的线程或线程组，为其他对象提供该线程组的内容信息。比如，我们调用 `activeCount()` 操作函数就能够返回该线程组中的正在运行的线程数目。下面的例子中，`listCurrentThreads()` 操作函数将把本线程组的所有活动线程填入线程组，并把它们的名字打印出来。其他的操作函数还有 `activeGroupCount()` 和 `list()`，使用时请参考有关手册。

```

class EnumerateTest {
    void listCurrentThreads() { //得到本线程所属的线程组
        ThreadGroup currentGroup = Thread.currentThread().getThreadGroup();

        int numThreads;
        Thread listOfThreads[];
        // 计算当前线程组中活动的线程数
        numThreads = currentGroup.activeCount();
        listOfThreads = new Thread[numThreads];
        currentGroup.enumerate(listOfThreads);
        for (int i = 0; i <= numThreads; i++) { // 打印所有线程的名字
            System.out.println("Thread #" + i + " = " + listOfThreads[i].getName());
        }
    }
}

```

■ 针对线程组本身的操作函数。ThreadGroup 类支持的一些属性，可以制约所有组中的所有线程的属性。这些属性包括线程组中最大的优先级，线程是否是"daemon"组，线程组的名字，线程组的父类等。这些操作函数只对线程组层次的属性进行设置或查询，并不影响其中任何线程的属性。比如，当你调用 setMaxPriority() 来改变一个线程组的最大优先级时，我们仅仅改变了线程组的属性，并未改变线程组中每个线程的优先级。下面是 ThreadGroup 类中的操作函数列表。

- 1) getMaxPriority(), setMaxPriority()
- 2) getDaemon(), setDaemon()
- 3) getName()
- 4) getParent(), getParentOf()
- 5) toString()

412

下面的一个例子说明了线程组的最大优先级和其中的线程的优先级的关系。

```

class MaxPriorityTest {
    public static void main(String args[]) {

        ThreadGroup groupNORM = new ThreadGroup("A group with normal priority");
        Thread priorityMAX = new Thread(groupNORM, "A thread with maximum
            priority");
    }
}

```

```

// 设置线程优先级为最大(10)
priorityMAX.setPriority(Thread.MAX_PRIORITY);

// 设置线程组的最大优先级为一般线程的正常优先级(5)
groupNORM.setMaxPriority(Thread.NORM_PRIORITY);

//打印线程组的最大优先级和其中线程的优先级
System.out.println("Group's maximum priority =" + groupNORM.getMaxPriority());

System.out.println("Thread's priority = " + priorityMAX.getPriority());

// 再次设置线程优先级为最大(10)
priorityMAX.setPriority(Thread.MAX_PRIORITY);

// 再次打印线程组的最大优先级和其中线程的优先级
System.out.println("Group's maximum priority =" + groupNORM.getMaxPriority());
System.out.println("Thread's priority = " + priorityMAX.getPriority());
}
}

```

编译运行后的系统输出是:

```

Group's maximum priority = 5
Thread's priority = 10
Group's maximum priority = 5
Thread's priority = 5

```

从输出的前两行说明在设置线程组的最大优先级为一般线程的正常优先级(5)后,其中的线程并未受到任何影响。也就是说,在线程组中允许线程的优先级可以高于线程组规定的最高值。但当我们再生成新的线程时或调用 `setPriority()` 重新设置线程的优先级时将要受到影响。上例中的最后两行说明了这个问题。值得注意的是, `setMaxPriority()` 并不改变在线程组中的线程组的最大优先级。

同样地,线程组的名字和 `daemon` 状态只对线程组本身有效。改变线程组的 `daemon` 属性并不能改变线程组中的任何线程的 `daemon` 属性。因为线程组的 `daemon` 属性和其中线程的 `daemon` 属性没有关联,我们可以把任意一个线程放在 `daemon` 线程组中。线程组的属性仅仅意味着当其中的所有线程都消亡的时候,该线程组将自动消亡。

■ 针对线程组中所有线程的操作函数。用这些函数可以改变在线程组中的所有线程的执行状态。它们是:

- 1) resume ()
- 2) stop ()
- 3) suspend ()

需要注意的是，这些操作对线程组中的所有子线程组中所有的线程有效。这一点与 setMaxPriority()有很大不同。

■ 存取控制。线程组类本身不提供任何的存取控制，如让一个线程检查或修改另一个线程组中的线程。但它与安全管理员 (java.lang.SecurityManager) 集合在一起，并通过安全管理员来提供安全检查。

线程和线程组中都有一个操作函数 checkAccess()，它调用安全管理员的 checkAccess()。安全管理员决定是否允许存取。如果不可以，则 checkAccess() 产生一个 SecurityException 对象；否则就直接返回。

以下是线程组中调用 checkAccess() 的操作函数列表：

- 1) ThreadGroup(ThreadGroup parent, String name)
- 2) setDaemon()
- 3) setMaxPriority()
- 4) stop()
- 5) suspend()
- 6) resume()
- 7) destroy()

下面是线程类中调用 checkAccess() 的操作函数列表：

- 1) 所有的构造函数
- 2) stop()
- 3) suspend()
- 4) resume()
- 5) setPriority()
- 6) setName()
- 7) setDaemon()

在独立应用程序中，缺省的安全策略是不加任何安全检查。因此，不管是在哪一个线程组中，一个线程都可以检查或修改其他线程。用户可以实现自己的安全策略，具体请参考“网络通讯和安全”一章。

一般地，在浏览器中有自己的安全管理员，来保证严格的安全策略。如在 HotJava 中，一个线程不能检查或修改其他线程组中的线程。不同的浏览器有不同的策略，因而，同样的 Java 嵌入小程序在不同的浏览器中可能会有不同的效果。

7.5 多进程同步控制

在多线程的程序中，不仅仅是独立、异步的线程，很多时候需要线程之间按造一定的规则和策略，共享一些资源，协调运行。此时，一个线程的运行要考虑到其他线程的运行状态，甚至需要其他线程计算的结果。这就是线程的同步控制问题。

下面我们通过生产者和消费者模型，研究多线程的同步控制问题。在生产者和消费者模型中，生产者和消费者共享一个缓冲区，如图 7.5 所示。生产者生产的产品放入缓冲区中，消费者从缓冲区中取出产品消费。当缓冲区满时，生产者就不能继续往里面放产品，生产者必须停止生产。当缓冲区空时，消费者将无产品可消费，必须等待下一个产品进入缓冲区中。



图 7.5 生产者和消费者模型示例

下面的程序将要实现这个模型。生产者、消费者和缓冲区分别由类 `Producer`、`Consumer`、`CubbyHole` 实现。主类是 `PCTest`，它生成了以上三个类，并启动了生产者和消费者对象。例子中的缓冲区的大小为 1，也就是说，每次只能往其中放一件产品。缓冲区能够记录产品号，所以可以反映产品之间的差别。

```

//*****
class PCTest {
public static void main(String args[] ) {

    // 生成缓冲区对象
    CubbyHole c = new CubbyHole();
    //生成生产者和消费者对象，通过参数指定共享同一个缓冲区
    Producer p1 = new Producer(c, 1);
    Consumer c1 = new Consumer(c, 1);

    // 启动生产者和消费者线程
    p1.start();
    c1.start();
}
}

//*****
class Producer extends Thread {
    private CubbyHole cubbyhole;
    private int number;

    public Producer(CubbyHole c, int number) {
cubbyhole = c; //把共享缓冲区的指针与本对象相连
this.number = number; //生产者代号
  
```



```

    }

    public void run() {
        // 连续生产五件产品
        for (int i = 0; i < 5; i++) {
            // 把第 i 个产品放入缓冲区并打印提示信息
            cubbyhole.put(i);
            System.out.println("Producer #" + this.number + " put: " + i);
            try {
                //等待一段随机长度的时间
                sleep((int)(Math.random() * 100));
            } catch (InterruptedException e) {
            }
        }
    }
}

//*****
class Consumer extends Thread {
    private CubbyHole cubbyhole;
    private int number;

    public Consumer(CubbyHole c, int number) {
        cubbyhole = c; //把共享缓冲区的指针与本对象相连
        this.number = number; //消费者代号
    }

    public void run() {
        int value = 0;
        //连续消费五件产品
        for (int i = 0; i < 5; i++) {
            // 检查缓冲区中的产品, 把产品号返回, 并打印结果
            value = cubbyhole.get();
            System.out.println("Consumer #" + this.number + " got: " + value);
        }
    }
}

//*****

```

```

class CubbyHole {
    private int seq; //产品号
    private boolean available = false; //缓冲区满标志

    public synchronized int get() {
        //当缓冲区空时，等待生产者放入产品后跳出循环
        while (available == false) {
            try {
                //等待生产者调用 notify() 唤醒
                wait();
            } catch (InterruptedException e) {
            }
        }
        //消费者取出产品，设置缓冲区满标志为假
        available = false;
        //通知生产者线程可以继续往缓冲区中放入产品
        notify();
        //返回产品号
        return seq;
    }

    public synchronized void put(int value) {
        //当缓冲区满时，等待消费者取出产品后跳出循环
        while (available == true) {
            try {
                //等待消费者调用 notify() 唤醒
                wait();
            } catch (InterruptedException e) {
            }
        }
        //设置产品号和缓冲区满标志为真
        seq = value;
        available = true;
        //通知消费者线程可以从缓冲区中取产品
        notify();
    }
}

```

编译运行后有如下的屏幕输出:

```
Producer #1 put: 0
Consumer #1 got: 0
Producer #1 put: 1
Consumer #1 got: 1
Producer #1 put: 2
Consumer #1 got: 2
Producer #1 put: 3
Consumer #1 got: 3
Producer #1 put: 4
Consumer #1 got: 4
```

在上面的例子中，生产者和消费者的流程有一些差别。生产者在放入产品后，等待了一小段时间，即

```
try {
    //等待一段随机长度的时间
    sleep((int) (Math.random() * 100));
} catch (InterruptedException e) {
}
```

如果去掉上面的这几行程序，有可能出现如下的屏幕输出：

```
Producer #1 put: 0
Consumer #1 got: 0
Producer #1 put: 1
Consumer #1 got: 1
Producer #1 put: 2
Producer #1 put: 3
Consumer #1 got: 2
Producer #1 put: 4
Consumer #1 got: 3
Consumer #1 got: 4
```

细心的读者会发现，在屏幕上消费者连续放进了产品 2 和 3。是不是同步控制出现了问题？我们在消费者的构造函数中加入如下一行，把生产者的优先级提到最大。

```
setPriority(10);
```

运行后发现如下的屏幕输出：

```
Producer #1 put: 0
Producer #1 put: 1
Consumer #1 got: 0
```

```
Producer #1 put: 2
Consumer #1 got: 1
Producer #1 put: 3
Consumer #1 got: 2
Producer #1 put: 4
Consumer #1 got: 3
Consumer #1 got: 4
```

现在问题看得更清楚了，生产者总是迫不及待地往缓冲区中加入产品，但是总不能全部加完。看来同步还是在起作用。我们再仔细地分析源程序，可以发现生产者在放入 0 号产品后，马上又“企图”放入 1 号产品，但此时，缓冲区已满，所以只能等待。消费者才有机会运行，从缓冲区中取出产品。一旦消费者从 `get()` 返回，运行控制权又被生产者夺取。生产者放入 1 号产品然后从 `put(i)` 返回，在消费者之前打印出了 `Producer #1 put: 1` 信息，又进入 `put(i)` 等待。

在缓冲区类中，`get()` 和 `put(int value)` 的申明中都有 `synchronized` 关键字修饰，它的意义是 `get()` 和 `put(int value)` 这两个操作函数是不能同时运行的。像 `CubbyHole` 这样的类，被两个线程共享，存取操作必须进行同步控制，我们称之为条件变量（`condition variables`）。

当多个线程想存取有限的系统资源时，有可能出现“死锁现象”（`Deadlock`）。此时，各个线程都忙于抢占系统资源。但是往往每个线程都不能满足所有的要求，因而，所有的线程都停步不前，对系统资源造成极大浪费。

解决死锁问题的最好办法是采取预防的办法，而不是实时监测，等问题出现后再解决。预防的最简单的方法是把条件变量有序化，把每个线程的存取顺序作出规定。

7.6 小 结

在这一章中，我们重点介绍了如下的内容：

- 线程的概念和特点，线程的状态迁移及触发条件，Java 线程类的属性和可用的操作函数。在此基础上，介绍了两类线程编程应用方法：一类是直接从 `Thread` 类引伸出自己的子类。另一类是，实现 `Runnable` 接口（`interface`）。

- 线程的优先级的设置和 Java 运行环境对不同优先级的线程的调度规则。高优先级的线程一定抢占低优先级的线程，直到其进入暂停状态或结束。相同优先级的线程在不同的平台上有不同的结果。对于分时系统，按时间片轮转；对非分时系统，其中一个线程独占运行资源。在编程时可以调用 `yield()` 来解决问题。

- 线程组是提供对一类相关线程进行管理、控制和安全检查的对象。通过合理地应用它的四类操作函数可以方便我们对多线程的编程。

- 多线程的同步控制和防止“死锁”（`Deadlock`）。恰当地控制条件变量（`condition variable`）的存取，可以把众多线程按造一定的规则运行。要防止“死锁”，预防是最有效的办法。

第八章 网络通讯和安全

在本章中，我们将介绍统一资源地址（URL）的概念、组成方法和编程的方法、面向连接的套接字和数据报套接字的概念及 Java 中这两类套接字的类库，并就两类不同的套接字分别示例客户端和服务端的应用编程。同时，对 FTP、NNTP 和 WWW 等应用协议作简单的介绍。最后介绍 Java 应用的安全概念、实现机制和相应的 SecurityManager 类的功能，示例怎样编写和安装自己的安全管理员线程。

8.1 URL 编程

8.1.1 什么是 URL?

经常在国际互连网络上漫游的人一定听说过 URL 并用 URL 访问某些结点的主页，但究竟什么是 URL 呢？URL 是 Uniform Resource Locator 的缩写，是国际互连网上资源的代号。例如

`http://java.sun.com`

`ftp://ftp.netscape.com`

一般地，URL 由两个主要的部分组成，两者之间用“://”相连。

1. 协议名，如 `http`，`ftp`，`gopher`，`news` 等。它说明了访问时要采用的协议。在 Netscape Navigator 中，如果不加说明，则认为是 `http` 协议。

2. 资源名，如，`java.sun.com`。它代表了服务器在国际互连网上的域名地址。我们也可以进一步地说明使用哪一个端口、要访问哪一个目录中的哪一个文件和在该文件中的参考位置。所以完整的资源名应有如下几个部分：

- 主机名或 IP 地址。它是不可缺少的一部分。
- 端口号。与主机名相连时用冒号分隔，一般可以省掉，对 `http` 来说是 80。
- 文件名，包括目录名。对 `http` 协议，有时只列目录也是可以的，此时会访问一个缺省的文件，一般称作 `default.html` 或 `index.html` 等，不同的 WWW 服务器会有不同的规定。
- 参考位置。对 `http` 协议而言，可以在 HTML 文件中定义多个标志点。在 URL 中加上标志点可以把很长的页面定位在标志点处。这个功能相当于书签。

一个完整的 URL 的例子为：

`http://home.netscape.com:80/comprod/one/white-paper.html#intro-1`

8.1.2 创建 URL 对象

建立 URL 对象的最简单的方法是采用人工可读的字符串来生成。如：

```
URL netscape = new URL("http://home.netscape.com/comprod/one/white-paper.html");
```

URL 字符串可以分成几个部分来写，如：

```
URL netscape=new URL(http,home.netscape.com,/comprod/one/white-paper.html);
```

其中，第一个参数是协议名，第二个是网络站点名，第三个是在站点上的文件名，与上面第一个 URL 的效果一样。

在这里，我们用了绝对地址的办法来产生 URL 对象，如：

`http://home.netscape.com/comprod/one/white-paper.html`。另外，我们还可以采用相对地址

```
URL netscape = new URL("http://home.netscape.com/ ");
```

```
URL NetscapeOne = new URL(netscape, "comprod/one/white-paper.html");
```

```
URL NetscapeOneIntro = new URL(NetscapeOne, "#intro-1");
```

在这个构造函数中，有两个参数，即，参考 URL 和相对地址。使用这个构造函数之前，必须有一个 URL 用来作参考点。合成的 NetscapeOneIntro 的 URL 是

```
http://home.netscape.com/comprod/one/white-paper.html#intro-1
```

这样的好处是可以不必每次都写上一长串的 URL 字符，使程序清楚了；而且在有多个 URL 时，写明相对位置，易于把握之间的关系。在创建时，为防止出现了错误，可以按如下的例子处理：

```
try {
    URL myURL = new URL("http://java.sun.com/")
} catch (MalformedURLException e) {
    System.out.println("Malformed URL Exception:" + e );
}
```

8.1.3 URL 应用编程

下面的这个程序将把 URL 的内容读出，并打印在计算机屏幕上。它首先创建一个 URL 对象，然后调用该对象的 `openStream()` 操作函数，把输入定位到一个数据输入流对象中，通过该数据输入流对象的 `readLine()` 操作函数来读入数据。

```
import java.net.*;
import java.io.*;

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

```

try {
    URL ibm = new URL("http://www.ibm.com/");
    DataInputStream is; //输入流变量
    String inputLine; //读入字符串变量

    //创建新的输入数据流，并把它于 URL 的输入流相连
    is = new DataInputStream(ibm.openStream());
    //从数据输入流中读入数据并打印在屏幕上，直到全部读完
    while ((inputLine = is.readLine()) != null) {
        System.out.println(inputLine);
    }
    //关闭输入数据流
    is.close();
} catch (MalformedURLException e) {
    System.out.println("MalformedURLException: " + e);
} catch (IOException e) {
    System.out.println("IOException: " + e);
}
}
}

```

读者可以任意更改 URL 构造函数中的值，以便访问相应的结点。如果用户给出的 URL 无效，则会打印出如下的信息；

```
IOException: java.net.UnknownHostException: URL 字符串
```

下面的例子与上面的例子有同样的功能，但增加了一个新类 `URLConnection`，它是一个与 URL 的连接对象，其中封装了许多有用的操作函数，可用来帮助实现与相关的 URL 建立通讯。我们的输入流是从该类中调用 `getInputStream()` 获得。

```

import java.net.*;
import java.io.*;

class ConnectionTest {
    public static void main(String args[]) {
        try { // 生成一个 URL 对象
            URL ibm = new URL("http://www.ibm.com/");
            // 由 URL 对象产生一个联接
            URLConnection ibmConnection = ibm.openConnection();
            DataInputStream is;

```

```

String inputLine;

is = new DataInputStream(ibmConnection.getInputStream());
// 从数据输入流中读入数据并打印在屏幕上, 直到全部读完
while ((inputLine = is.readLine()) != null) {
    System.out.println(inputLine);
}
is.close();    // 关闭输入流
} catch (MalformedURLException e) {
    System.out.println("MalformedURLException: " + e);
} catch (IOException c) {
    System.out.println("IOException: " + e);
}
}
}

```

通过 `URLConnection` 对象不仅可以从服务器读入数据, 还可以向服务器中写数据。下面的这个例子介绍了在客户端的一个 Java 程序与服务器侧的一个 CGI 程序建立连接, 并向它输入一串字符, 该 CGI 程序在作倒序处理后, 打印输出。

我们知道, 许多的 HTML 页面中包含有表格、文本域和其他的图形界面元素, 以便用户输入数据到服务器中。当你键入相应的信息后点击按钮, 浏览器就会把信息通过网络写回原 URL。在服务器一侧, 通常是一个 CGI 程序处理收到的数据, 然后把响应返回给浏览器, 一般是一个新的 HTML 页面。这种方法经常用来支持检索。

有的服务器侧的 CGI 程序用 `GET METHOD` 来读取数据, 但更多的 CGI 程序用 `POST METHOD` 从客户端读数据。这是因为 `POST METHOD` 更通用, 并且对通过连接发送的数据没有限制, 使 `GET METHOD` 成为一种过时的方法。

用 Java 程序也可以与服务器侧的 CGI 程序建立连接。编程的步骤是:

- 1) 构造一个 URL 对象。
- 2) 构造与该 URL 的连接。
- 3) 从连接中获得一个输出流, 该输出流与服务器侧的 CGI 程序的输入流相连。
- 4) 向输出流中写数据。
- 5) 关闭输出流。

下面是一个用 Perl 语言写的 CGI 程序, 用来从标准输入流中读入字符串。用户可以把该程序放在服务器上, 把它命名为 `backwards`, 相应地要改写在 Java 程序中的 URL。如果用户的网络环境很好, 可以用 Sun 公司服务器已有的 CGI 程序, 而不必再在自己的服务器上装下列的 Perl CGI 程序。Sun 公司服务器的 CGI 程序的地址是:
<http://www.javasoft.com/cgi-bin/backwards>。

```
#:/opt/internet/bin/perl
```



```

read(STDIN, , {'CONTENT-LENGTH'});
@pairs = split(/&/, );
foreach (@pairs)
{
    (, ) = split(/=/, );
    =~ tr/+// /;
    =~ s/%([a-zA-F0-9][a-zA-F0-9])/pack("C", hex( ))/eg;
    # Stop people from using subshells to execute commands
    =~ s/~!/~!/g;
    { } = ;
}
print "Content-type: text/plain\n\n";
print "'string' reversed is: ";
=reverse('string');
print "\n";
exit 0;

```

下面是 Java 程序的程序清单:

```

import java.io.*;
import java.net.*;

public class ReverseTest {
    public static void main(String args[ ]) {
        try {
            if (args.length != 1) {
                System.err.println("Usage:  java ReverseTest string-to-reverse");
                System.exit(1);
            }
            //把 args[0]字符串转换成 x-www 格式的字符串
            String stringToReverse = URLEncoder.encode(args[0]);

            URL url = new URL("http://www.javasoft.com/cgi-bin/backwards");
            URLConnection connection = url.openConnection();
            //创建输出数据流, 并把它与 URL 相连
            PrintStream outStream = new PrintStream(connection.getOutputStream());

            DataInputStream inStream;
            String inputLine;

```

```

//输出字符串到 URL 后, 关闭输出流
outStream.println("string=" + stringToReverse);
outStream.close();

//创建输入数据流, 并与 URL 相连
inStream = new DataInputStream(connection.getInputStream());
//读入数据后, 关闭输入流
while (null != (inputLine = inStream.readLine())) {
    System.out.println(inputLine);
}
inStream.close();
} catch (MalformedURLException me) {
    System.err.println("MalformedURLException: " + me);
} catch (IOException ioe) {
    System.err.println("IOException: " + ioe);
}
}
}

```

编译完成后, 键入如下的命令:

```
>java ReverseTest "Reverse me"
```

在屏幕上会有如下的输出:

```
Reverse Me
reversed is:
eM esreveR
```

如果使用网络代理, 则可以在 Java 的命令行中实现, 如

```
java -DproxySet=true -DproxyHost=proxyhost ReverseTest "Reverse Me"
```

其中的 proxyhost 要换成实际的站点名。

8.2 套接字编程

套接字是实现进程间通讯的标准的编程接口之一, 最初由 UNIX BSD 版本提供。现在已成为最流行的编程接口, 在各个操作系统平台上均有相应的支持。套接字的类型由传送数据的类型来划分, 有:

- 数据流式套接字 (Stream socket)。它提供面向连接 (Connection-oriented) 的服务 (TCP)。当连接意外中断时, 使用套接字的进程将被通知。

■ 数据报式套接字 (Datagram socket)。它提供非连接 (Connection-oriented) 的服务 (UDP)。它是一种不可靠的服务。但比前者要经济, 节省资源。

■ 原始套接字 (Raw socket)。它提供到底层协议软件的直接存取控制。一般很少用到, 只在调试网络协议软件时才用到。

套接字可以用在 UNIX 进程通讯、国际互连网通讯、X.25 通讯等的编程。

8.2.1 面向连接的套接字编程

套接字的应用编程接口 (API) 有 `socket()`, `bind()`, `listen()`, `accept()`; `connect()` 等系统调用组成, 这里不再赘述。在 Java 中, 这些系统调用用户不可见, 被封装到两个类中:

■ `Socket` 完成客户端的套接字操作。

■ `ServerSocket` 完成服务器端的套接字操作。

下面按照通讯的双方来分别介绍这两个类的使用编程。

1. 客户端编程

客户端的编程的流程是:

- 1) 新建一个套接字。
- 2) 为套接字建立一个输入和输出流。
- 3) 根据协议从套接字读入或向套接字写入。
- 4) 清除套接字和输入、输出流。

下面给出的例子是输入一行字符串后把它送到一个国际互连网站点 “`www.network.cn.ibm.com`”, 该站点运行着一个后台程序, 它接到后把字符串又原封不动地返回来。读者在键入以下程序时, 要把站点名改成就近的站点。如果在同一个域名中, 则后面的可以省掉。

在这里我们用了 7 号端口, 它是已经定义了的公认的服务口 (Well known port)。也就是说, 在系统中它被保留用来作一些常用的服务。如, 25 是电子邮件服务口, 21 是 ftp 服务口, 80 是 WWW 服务口, 513 是远程登录服务口等等。一般地, 在 1000 以下的端口为系统所保留, 建议用户在开发自己的服务器时不要使用, 否则可能会引起冲突。

```
//引入输入、输出库和网络支持库
import java.io.*;
import java.net.*;

public class EchoTest {

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

```

// 创建一个新的套接字，其目标站点是"www.network.cn.ibm.com",
// 所用的端口号是 7。
Socket echoSocket = new Socket("www.network.cn.ibm.com", 7);
// 得到一个套接字的输出流，并把它定向到一个输出流变量
OutputStream os = echoSocket.getOutputStream();
// 得到一个套接字的输入流，创建一个新的输入流，并把两者连接起来。
DataInputStream is = new DataInputStream(echoSocket.getInputStream());

    int c;
    String responseLine;

    while ((c = System.in.read()) != -1) {
        os.write((byte)c);
        if (c == '\n') {
            //强制马上输出
            os.flush();
            responseLine = is.readLine();
            System.out.println("echo: " + responseLine);
        }
    }

    // 关闭打开的流和套接字对象
    os.close();
    is.close();
    echoSocket.close();
} catch (Exception e) {
    System.err.println("Exception: " + e);
}
}
}

```

2. 服务器端编程

服务器侧的编程流程是：

- 1) 创建一个服务器型套接字和一个普通套接字
 - 2) 使服务器型套接字处于监听状态并把监听结果返回给普通套接字
 - 3) 为该普通套接字创建输入和输出流
 - 4) 从输入和输出流中读入或写入字节流，进行相应的处理
 - 5) 在完成后关闭所有的对象，如，服务器型的套接字、普通套接字、输入和输出流。
- 下面的例子列举了一个服务器侧的编程。它可以与前面的客户端编程相配合，在使用

时，要把客户端的代码中的套接字端口号改成 4000（或其他的端口号），以保证两侧的一致性，否则两边不能通讯。

这个程序是在端口 4000 监听来到的信息，在接到后再原封不动地返回去，就好像是一堵墙能反馈到来的声音一样。

```
//引入输入、输出库和网络支持库
import java.net.*;
import java.io.*;

class EchoServer {
    int i;
    public static void main(String args[] ) {
        // 申明一个 ServerSocket 类型变量
        ServerSocket serverSocket = null;
        try {
            //创建等待在 4000 端口的服务器套接字
            serverSocket = new ServerSocket(4000, 10);
        } catch (IOException e) {
            System.out.println("Could not listen on port: "+4000+", "+e);
            System.exit(1);
        }

        Socket clientSocket = null;
        try {
            System.out.println("before accept\n");
            //把服务器套接字的监听结果返回到客户套接字变量
            clientSocket = serverSocket.accept();
            System.out.println("after accept\n");
        } catch (IOException e) {
            System.out.println("Accept failed: "+4000+", "+e);
            System.exit(1);
        }

        try {
            //生成输入和输出数据流，并与套接字相连接
            DataInputStream is = new DataInputStream(
                new BufferedInputStream(clientSocket.getInputStream()));
            PrintStream os = new PrintStream(
                new BufferedOutputStream(clientSocket.getOutputStream(), 1024), false);
```

```

        String inputLine, outputLine;
        //读入数据，并原封不动地写回
        while ((inputLine = is.readLine()) != null) {
            for(int i=0; i<inputLine.length(); i++) os.write((byte)inputLine.charAt(i));
            os.write((byte)'\n');

            System.out.println(inputLine);
            os.flush();
        }
        //关闭打开的输入输出数据流和套接字
        os.close();
        is.close();
        clientSocket.close();
        serverSocket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

8.2.2 数据报套接字编程

前面的套接字是面向连接的，也就是说，在通信的两个站点之间有一个通道（channel）。为了通讯，两者首先要建立数据连接，然后发送或接收数据，最后要把连接关闭。所有的数据在网络上传送的时候都按照一定的顺序，先发的数据一定是先收到。对数据报而言则不一样，通讯的两端之间没有一个专有的通道。每个数据包在传送时，是按照网络的情况，选择可用的路径来发送。这样，有可能先送出的数据落后于后发送的数据到达接收方。

java.net 类库包包含有两个类来支持数据报的收发：DatagramSocket 和 DatagramPacket。DatagramSocket 支持在应用之间发送数据报的通讯连接。DatagramPacket 是通过 DatagramSocket 发送的消息对象。下面的这个例子是一个客户/服务器结构的查询系统。

1. 客户端编程

```

import java.io.*;
import java.net.*;
import java.util.*;

```

```

class QuoteClient {

    public static void main(String[] args) {
        int port; //服务端口
        InetAddress address; //Internet 地址
        DatagramSocket socket; //数据报套接字
        DatagramPacket packet; //数据报数据帧
        byte[] sendBuf = new byte[256]; //发送缓冲区

        //命令行参数检查
        if (args.length != 2) {
            System.out.println("Usage: java QuoteClient <hostname> <port#>");
            return;
        }
        try {
            //创建新的数据报套接字
            socket = new DatagramSocket();
            //读出命令行中的端口数值
            port = Integer.parseInt(args[1]);
            //把字符串方式的 internet 地址读到 InetAddress 对象中
            address = InetAddress.getByName(args[0]);
            //创建数据报数据帧对象，该对象包括了数据，目的地址，端口等内容
            packet = new DatagramPacket(sendBuf, 256, address, port);
            //发送数据帧对象
            socket.send(packet);

            //创建新的数据报对象，限定长度为 256
            packet = new DatagramPacket(sendBuf, 256);
            //让数据报套接字处于接收状态，直到它接收到一个数据帧。
            //否则，它就一直等待。
            socket.receive(packet);
            //从接收到的数据帧中取出数据
            String received = new String(packet.getData(), 0);
            //把结果打印在屏幕上
            System.out.println("Quote of the Moment: " + received);
            //关闭套接字
            socket.close();
        } catch (Exception e) {
            System.err.println("Exception: " + e);
        }
    }
}

```

```

e.printStackTrace();
    }
}

```

2. 服务器侧编程

```

import java.io.*;
import java.net.*;
import java.util.*;

class QuoteServer {
    public static void main(String[] args) {
        //启动服务器线程
        new QuoteServerThread().start();
    }
}

class QuoteServerThread extends Thread {
    //定义两个私有变量
    private DatagramSocket socket = null; //数据报套接字
    private DataInputStream qfs = null; //读入数据流

    QuoteServerThread() {
        //调用父类的构造函数
        super("QuoteServer");
        try {
            //创建数据报套接字，随机找一个可用的端口
            socket = new DatagramSocket();
            //打印出当前监听的端口号
            System.out.println("QuoteServer listening on
port: "+socket.getLocalPort());
        } catch (java.net.SocketException e) {
            System.err.println("Could not create datagram socket.");
        }
        //打开一个引用数据文件
        this.openInputFile();
    }
}

```



```

public void run() {
    //如果套接字还没有创建，则退出
    if (socket == null)
        return;
    //进入无限循环
    while (true) {
        try {
            byte[] buf = new byte[256];
            DatagramPacket packet;
            InetAddress address;
            int port;
            String dString = null;

            //创建数据报数据帧，限长 256
            packet = new DatagramPacket(buf, 256);
            // 接收客户端的请求数据帧
            socket.receive(packet);
            //从到来的数据报中读出 internet 地址
            address = packet.getAddress();
            //从到来的数据报中读出端口号
            port = packet.getPort();

            // 如果引用数据文件可用，则从中读出一句话
            // 否则，把当前时间转换成字符串放入缓冲区
            if (qfs == null)
                dString = new Date().toString();
            else
                dString = getNextQuote();
            //把缓冲区中的字符串转换后放入字节数组中
            dString.getBytes(0, dString.length(), buf, 0);
            // 把字节数组中的数据和刚接收到的数据中提供的
            // 源地址及端口号打包，放在数据报数据帧对象中
            packet = new DatagramPacket(buf, buf.length, address, port);
            // 把数据报数据帧发送出去
            socket.send(packet);
        } catch (Exception e) {
            System.err.println("Exception: "+e);
            e.printStackTrace();
        }
    }
}

```

```

    }
}

protected void finalize() {
    //如果套接字还存在，则关闭它，并把变量置空
    if (socket != null) {
        socket.close();
        socket = null;
        System.out.println("Closing datagram socket.");
    }
}

private void openInputFile() {
    try {
        // 打开一个名为 quote.txt 的数据文件，其中包含了我们要返回给客户端
        // 的一些句子。这个文件中的每个引用句子都在一行中，并且不能超过 256//
        // 个字节。每个句子结束后要加回车。
        qfs = new DataInputStream(new FileInputStream("quote.txt"));
    } catch (java.io.FileNotFoundException e) {
        System.err.println("Could not open quote file. Serving time instead.");
    }
}

private String getNextQuote() {
    String returnValue = null;
    try {
        //从文件中读出一行，如果读到了最后，则返回空
        if ((returnValue = qfs.readLine()) == null) {
            qfs.close();
            this.openInputFile();
            returnValue = qfs.readLine();
        }
    } catch (IOException e) {
        returnValue = "IOException occurred in server.";
    }
    // 把读出的字符串返回给调用者
    return returnValue;
}
}

```

编译这两个线程后，先在一台计算机 A 上运行服务器侧软件。屏幕上会有如下的显示，

其中的端口号是一个数字，比如 3997 等。

QuoteServer listening on port: 端口号

根据上面的端口号，在另一台计算机上运行客户侧软件。其命令格式如下：

java QuoteClient 计算机 A 的地址 端口号

如果在计算机 A 上没有名为 quote.txt 的文件，则显示出如下的信息。否则，显示从 quote.txt 文件中读出一行引用句子。

Client sent request packet.

Client received packet: Wed Nov 27 08:23:23 1996

8.3 协议和内容处理机

除了 Java 类库中提供的套接字协议等之外，sun 网络类库还提供了丰富的协议支持，如 ftp、NNTP 和 WWW 等协议。下面将仔细地介绍 ftp 类的使用方法，对其他的协议作简单地介绍。

8.3.1 FTP 协议

sun.net.ftp 软件包中包含了许多建立和使用 FTP 的操作函数，如登录、目录列表、设定传送的方式（字符方式还是二进制方式）、双向传送文件等。它的这些操作函数在很大程度上与 ftp 的命令集相对应。其中主要的类是 FtpClient。它的变量有：

- public static boolean useFtpProxy：标志是否采用 ftp 代理（Proxy）。代理是介于国际互连网和本机的一个服务器，它可以在缓存区中存放最近访问过的内容，提高访问速度。

- public static String ftpProxyHost：如果 useFtpProxy 为真，则存放代理的站点名。

- public static int ftpProxyPort：如果 useFtpProxy 为真，则存放代理的端口号。它的构造函数有：

- public FtpClientString(String host)：用主机名产生 FTP 连接。

- public FtpClientString(String host, int port)：用主机名和端口号产生 FTP 连接。

- public FtpClientString()：先产生一个实例，再用 openServer() 建立连接。

操作函数有：

- public void openServer(String host)：用主机名建立连接。

- public void openServer(String host, int port)：用主机名和端口号产生 FTP 连接。

- public void login(String user, String password)：登录。前者的字符串是用户名，后面的字符串是密码。

- public void cd(String remoteDirectory)：目录转换。

- public void binary()：设置为二进制传送方式。

- `public void ascii()`: 设置为字符传送方式。
- `public TelnetInputStream list()`: 服务器上的文件列表。
- `public TelnetInputStream get(String fileName)`: 把相应文件从服务器上传回本机。
- `public TelnetInputStream put(String fileName)`: 把相应文件从本机传到服务器上。

下面的一个例子是运用 `FtpClient` 类设计的一个应用，它把用户指定的文件下载到本机并打印在屏幕上。

```
import sun.net.ftp.*;
import sun.net.*;
import java.net.*;

class FtpTest {
    static FtpClient ftp;
    public static void main( String args[ ])
        throws java.io.IOException {
        StringBuffer buf = new StringBuffer();
        int ch;
        ftp = new FtpClient(args[0]);
        ftp.login("guest", "guest");
        ftp.ascii();
        TelnetInputStream t=ftp.list();
        while ( (ch= t.read())>=0)
            buf.append((char)ch);
        t.close();
        System.out.println(buf.toString());
        System.out.println("\nEnter File to Download:\n");
        buf.setLength(0);
        while((ch=System.in.read())!='\n')
            buf.append((char)ch);
        t=ftp.get(buf.toString());
        buf.setLength(0);
        while((ch=t.read())>=0)
            buf.append((char)ch);
        t.close();
        System.out.println(buf.toString());
    }
}
```

用户在使用这个程序时，要把 `ftp.login("guest", "guest")` 中的两个参数改成将要连接的 ftp 服务器的用户名（第一个参数）和密码（第二个参数），在编译完成后，按这样的语法来使用：

```
>java FtpTest 站点名
```

键入回车后，屏幕会列出当前服务器上的目录和文件名，然后系统提示用户要下载哪一个文件，用户输入后，屏幕上出现所要文件的内容。最后程序结束，退回系统状态。

8.3.2 NNTP 协议

NNTP 是网络新闻传输协议，用以在国际互连网上传送新闻消息。新闻组是按各类专题组织在一起的，用户可以读或发布信息。

Java 中的支持类是 `NntpClient` 类，它的使用方法与 `ftp` 类大体相同，下面仅列出有关的操作函数，帮助读者了解 `NntpClient` 类的功能。

```
public NntpClient(String host)
public NewsgroupInfo getGroup(String name)
public setGroup(String name)
public InputStream getArticle()
public PrintStream startPost()
public boolean finishPost()
```

有了以上的这些类我们就可以实现自己的新闻组阅读器。

8.3.3 WWW 协议

WWW 协议类库包可以分成几个不同的部分，每个部分分别担当不同的作用，如，处理 WWW 的页面，与 HTTP 服务器连接等。它的类库包有：

- `sun.net.www.auth`：包括了验证用的类 `Authenticator.class`，`basic.class` 等。
- `sun.net.www.content`：包括了处理图形和文字的类，如，`gif.class` `jpeg.class` `x-xbitmap.class`，`x-ypixmap.class`，`plain.class` 等。
- `sun.net.www.protocol`：包括了 `Handler.class`，`URLConnection.class`，`HttpURLConnection.class`，`HttpPostBufferStream` 等类支持。
- `sun.net.www.http`：包括了 `UnauthorizedHttpRequestException.class`，`HttpClient.class`，`Authentication.class` 等类，用以构造 Http 客户端。

等等。

有了这些类的支持，编写通讯软件，特别是采用已有协议的软件将变得很方便，在用户使用它们编程时，具体地参考相应的类说明文件。

8.4 安全机制

在编写与国际互连网有关的应用程序时，网络安全是大家普遍关心的一个问题。很难说与网络相连的计算机能绝对安全，但我们可以采取措施来保护计算机和其中的数据不受病毒和非法进入者的侵害。

在 Java 运行环境中有一个安全管理员 (Security Manager) 线程。只要 Java 系统运行，它就一直在工作。安全管理员的任务是监视程序的运行。一些有可能对系统安全造成威胁的代码在执行时要先经安全管理员批准。如果安全管理员不允许某种代码执行，它会产生一个 SecurityException 例外对象。

下面我们将讨论如何来编写自己的安全管理员并且在应用程序中安装它。有一点需要说明的是，在 Java 的嵌入小程序中我们不能设计、安装新的安全管理员。嵌入小程序的安全性必须受浏览器中的安全管理员的制约，因此有一些嵌入小程序的页面在不同的浏览器的表现有所不同。

我们自己的安全管理员必须是 SecurityManager 的子类。因此，在编写应用程序时，首先要从 SecurityManager 中衍生一个新类，然后根据自己应用的安全性需要，编写、覆盖其中的各种核实和批准的操作函数。

8.4.1 SecurityManager 类

SecurityManager 是一个抽象类，通过它进一步衍生出的新类来执行自己的安全策略。它允许对系统的执行堆栈进行检查。它有如下的操作函数：

- public void checkAccept(String hostname, int port)：检查到网上站点的套接字连接请求是否被接受。
- public void checkAccess(Thread threadname)：检查所选定的线程是否允许修改这个组。
- public void checkAccess(ThreadGroup groupname)：检查所选定的线程组是否允许修改这个组。
- public void checkConnect(String hostname, int port)：检查到网上站点的套接字是否已经建立。
- public void checkConnect(String hostname, int port, Object context)：检查当前运行上下文和选定的运行上下文是否都允许与相关的主机相连。
- public void checkCreateClassLoader()：检查类装载器 (ClassLoader) 是否已经建立。
- public void checkDelete(String file)：检查某种依赖于特殊文件能够被删除掉。
- public void checkExec(String cmd)：检查系统命令能够以可靠代码执行。
- public void checkExit(int status)：检查系统是否已经退出了 Java 虚拟机。如果 status 为 0，则说明是正常退出。
- public void checkLink(String lib)：检查特定的连接库是否存在。
- public void checkListen(int port)：检查服务器套接字是否在监听选定的端口

- `public void checkPackageAccess(String pkg)`: 检查嵌入小程序能否存取一个包。
- `public void checkPackageDefinition(String pkg)`: 检查嵌入小程序能否定义在一个包中的类。
- `public void checkPropertiesAccess()`: 检查是否能存取系统属性。
- `public void checkPropertyAccess(String key)`: 检查是否能存取名为 `key` 的系统属性。
- `public void checkPropertyAccess(String key, String def)`: 检查是否能存取名为 `key` 和 `def` 的系统属性。
- `checkRead(FileDescriptor fd)`: 检查一个特定文件描述的输入文件已经产生。
- `public void checkRead(String filename)`: 检查一个特定文件名的文件已经产生。
- `public void checkRead(String filename, Object context)`: 检查当前上下文和选定的上下文能否存取这个特定的文件。
- `public void checkSetFactory()`: 检查一个嵌入小程序能设置网络相关的对象定制。
- `public boolean checkTopLevelWindow(Object)`: 检查调用者能产生最顶层的窗口。
- `public void checkWrite(FileDescriptor fd)`: 检查特定文件描述符的输出文件已经产生。
- `public void checkWrite(String filename)`: 检查特定文件名的输出文件已经产生。
- `protected int classDepth(String name)`: 返回包含第一个出现的特定类的堆栈帧的位置。
- `protected int classLoaderDepth()`:
- `protected ClassLoader currentClassLoader()`: 返回当前执行堆栈上的类装载器。
- `protected Class[] getClassContext()`: 返回本类的上下文。
- `public boolean getInCheck()`: 确认是否有安全检查。
- `public Object getSecurityContext()`: 返回实现无关的对象，它封装了当前执行对象的有关信息，为以后的安全检查作准备。
- `protected boolean inClass(String name)`: 检查选定的字符串在这个类中。
- `protected boolean inClassLoader()`: 检查当前的类装载器是否等于空。

在以上的操作函数中最的一类是 `check***()`，它们有缺省的实现，但在安全性检查时比较宽容。在用户编写的应用中可以有选择地覆盖有关的函数，以满足特定的安全性需要。

几乎所有的 `check***()` 操作函数在检查通过时正常返回，否则会产生一个 `SecurityException` 安全例外对象。在编写覆盖操作函数时也要按造这样的方式返回。

8.4.2 编程实例

下面是几个 ThreadGroup 类的操作函数，它们在执行之前要获得安全管理员的批准。

- ThreadGroup(ThreadGroup parent, String name)
- setDaemon()
- setMaxPriority()
- stop()
- suspend()
- resume()
- destroy()

在 Thread 类中也有若干个操作函数需要安全管理员的批准方能执行。如

- 所有的构造函数
- stop()
- suspend()
- resume()
- setPriority()
- setName()
- setDaemon()

所有这些操作函数都调用了安全管理员的 checkAccess() 操作函数，以便进行安全检查。如果安全管理员批准了，checkAccess() 返回，否则 checkAccess() 产生一个 SecurityException 对象。

在我们的例子中，希望严格对线程存取的控制，我们必须覆盖 checkAccess() 操作函数。安全管理员提供了两个版本的 checkAccess()：一个采用 Thread 作参数；另一个采用 ThreadGroup 作参数。每个操作函数检查当前的线程能否执行某些有限制的操作。大多数的浏览器采用的策略是一个线程或线程组只能存取同一个组的线程或线程组。

下面的例子将实现一个安全管理员，来限制某些线程的存取。

```
import java.io.*;

class SMTest {

    private static final int NUMTHREADS = 2; //设定线程数目为 2

    public static void main(String args[]) {
        try {
            // 创建并安装安全管理员线程
            System.setSecurityManager(new SecretPasswordSMgr("password"));
        } catch (SecurityException se) {
            System.out.println("SecurityManager already set!");
        }
    }
}
```



```

    }

    // 开辟一个线程变量数组
    Thread someThreads[ ] = new Thread[NUMTHREADS];
    // 创建一个线程组对象
    ThreadGroup aGroup = new ThreadGroup("A Group of Threads");

    for (int i = 0; i < NUMTHREADS; i++) {
        // 在线程组中创建线程并启动它
        someThreads[i] = new Thread(aGroup, String.valueOf(i));
        someThreads[i].start();
    }
}

}

class SecretPasswordSMgr extends SecurityManager {
    private String password; //存放密码

    SecretPasswordSMgr(String password) {
        super();
        this.password = password;
    }

    private boolean accessOK() {
        int c;
        DataInputStream dis = new DataInputStream(System.in);
        String response;

        System.out.println("What's the secret password?");
        try {
            response = dis.readLine();
            // 检查读入的密码是否与设定的一样。如果一样则返回真，返回假
            if (response.equals(password))
                return true;
            else
                return false;
        } catch (IOException e) {
            return false;
        }
    }
}

```

```

    }
    public void checkAccess(Thread g) {
        // 如果不能存取, 则产生一个 SecurityException 安全例外对象
        if (!accessOK())
            throw new SecurityException("Not!");
    }
    public void checkAccess(ThreadGroup g) {
        if (!accessOK())
            throw new SecurityException("Not Even!");
    }
}

```

编译结束后, 键入命令:

```
>java SMTTest
```

屏幕出现提示输入密码, 用户键入三次密码 (对应了两个线程和一个线程组) 后, 屏幕如下:

```
What's the secret password?
```

```
password
```

```
What's the secret password?
```

```
password
```

```
What's the secret password?
```

```
password
```

如果输入密码不对, 则有如下的出错信息:

```
What's the secret password?
```

```
m
```

```
java.lang.SecurityException: Not Even!
```

```
    at SecretPasswordSMgr.checkAccess(SMTTest.java:60)
```

```
    at java.lang.ThreadGroup.checkAccess(ThreadGroup.java:166)
```

```
    at java.lang.ThreadGroup.<init>(ThreadGroup.java:76)
```

```
    at java.lang.ThreadGroup.<init>(ThreadGroup.java:63)
```

```
    at SMTTest.main(SMTTest.java:18)
```

8.5 小 结

在本章中, 我们介绍了如下的内容:

- 统一资源地址 (URL) 的概念和组成方法, Java 中对 URL 的类库支持。在此基础上示例了编程的方法。

- 面向连接的套接字和数据报套接字的概念和应用场合, Java 中这两类套接字的类库。就两类不同的套接字分别示例了客户端和服务端的应用编程。

■ FTP、NNTP 和 WWW 等应用协议的简单介绍。合理应用这些封装协议实现细节的类库，可以极大地方便我们对已有这些协议的编程。

■ Java 应用的安全概念、实现机制和相应的 SecurityManager 类的功能介绍。示例了怎样编写和安装自己的安全管理员线程。

第九章 Java 综合运用举例

在学习了前面几章的基础上，我们在这一章中再讲解一个综合的例子。此例涉及了用户界面设计、网络通讯和多线程编程等诸多方面，使读者对这些编程手段能有一个全面的认识。

9.1 概 述

这个例子是一个网上交互软件，用它可以在 Internet 网上下围棋。在两台国际互连网上的机器上运行该应用，出现图 9.1 所示的屏幕。在屏幕上有菜单条、棋盘区、棋手交谈区和信息提示区组成。在启动后，系统信息提示区上说明系统正在监听在缺省端口 4444。此时它已经可以接受另一台机器上同样应用的连接请求。

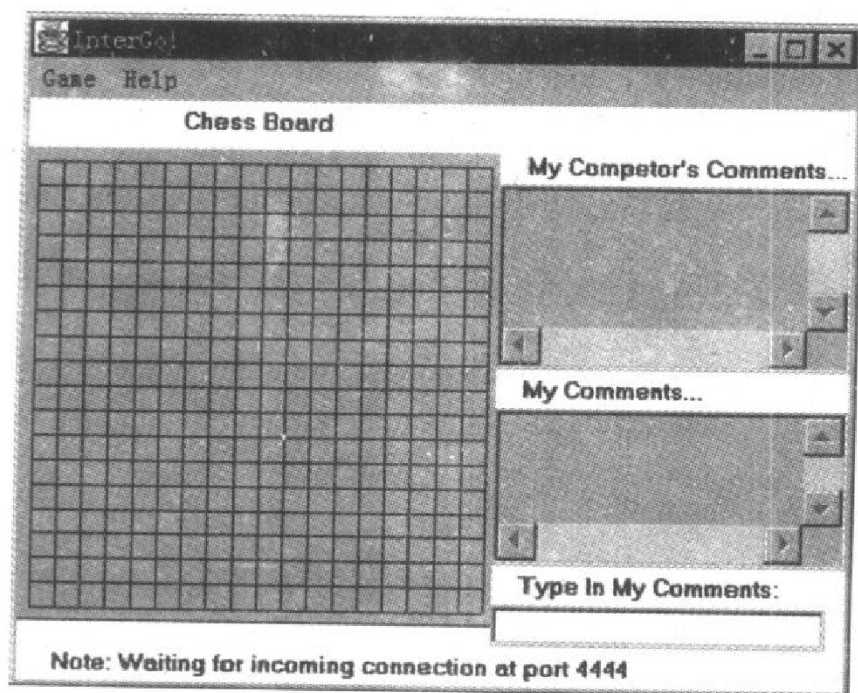


图 9.1 网上交互式围棋软件的主屏幕

本机的用户也可以主动地去建立连接。选择 Game 菜单中的 Start a connection... 菜单项，就进入一个对话框，提示用户输入对方的机器名和连接端口号如图 9.2 所示。输入完成后，点击 Set 按钮，如果连接成功，就会在信息提示区显示成功信息。否则警告用户失败。

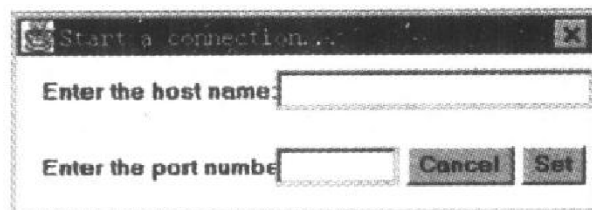


图 9.2 建立连接的对话框

连接成功后，用户才能在围棋的棋盘区中用鼠标点击来下子，否则在信息提示区会有警告信息告诉用户首先建立连接。下的子同时在对方的棋盘上显示，同时对方的信息提示区会告诉他该他走下一个子。在棋盘区上，棋手不能连着下两手棋，否则在信息提示区会有警告信息出现。

在下棋的同时，棋手还可以通过右侧的交谈区来交流。在右下侧的文本输入框中，键入自己的意见后，打回车。该信息就马上传送到对方的交谈区的上方的对手意见框中，同时在本机的交谈区中留一个备份，以备以后查找。

9.2 程序结构

该网上交互围棋软件有如下的若干个类：

- GoGoWindow：这是应用的主类，其中包含有 main() 函数。负责系统的初始化、其他类的启动和协调。
- StartDialog：主动建立连接时，设置对方结点号和端口的对话框。由主类的菜单项触发。
- ChangePortDialog：改变通讯端口对话框。由主类的菜单项触发。
- MessageBox：一般的信息提示对话框，它给用户提示提示信息。
- CommunicationServer：通讯服务器线程类。它负责建立可靠的双向数据连接。
- WaitingForConnectionThread：监听线程类。它在系统建立起数据连接之前负责监听套接字端口。
- ChessBoard：围棋棋盘类。

9.3 源程序

```

/*****
Software Name: InterGo
Author: Dr. 金尊和
zhjin@vnet.ibm.com
Date: June 1996
*****/

/** First import some of the available class from library
include Another Window Toolkit, Input/Output, network and some utilities

```

```

    */
    import java.awt.*;
    import java.io.*;
    import java.net.*;
    import java.util.*;
    /*****
    Here is the main class of this software. It extends the class Frame.
    It use several classes inside. They are CommunicationServer class which
    is in charge of setting up socket and maintaining the communication stream,
    ChessBoard which is in charge of monitoring and displaying the status of the
    chess board.
    *****/
    public class GoGoWindow extends Frame {
        boolean inAnApplet = true;
        int currentPlayMode=0;    // 0- disabled; 1- network; 2- wizard
        public TextArea textAreaFromCompetor, textAreaFromMyOwn;
        TextField    textFieldMyTypeIn;
        ChessBoard myChessBoard;
        static Label    messageLabel;
        public static CommunicationServer communicationServer;

        //constructor of this class, setup the layout and display the element of the frame.
        public GoGoWindow( ) {
            Panel talkPanel = new Panel( );
            Panel centerPanel = new Panel( );
            setLayout(new BorderLayout( ));

            //Set up the menu bar.
            MenuBar mb = new MenuBar( );
            Menu m = new Menu("Game");
            m.add(new MenuItem("Start a connection..."));
            m.add(new MenuItem("Chang the listening port..."));
            m.add(new CheckboxMenuItem("Leave"));
            m.addSeparator( );

            m.add(new MenuItem("Option..."));
            m.addSeparator( );
            m.add(new MenuItem("Exit"));
            Menu Help = new Menu("Help");

```

```

Help.add(new MenuItem("About"));
Help.add(new MenuItem("Rule"));
mb.add(m);
mb.add(Help);

setMenuBar(mb);

// set up a message line to remind user of some info.
messageLabel= new Label("Note: ",Label.LEFT);
messageLabel.setForeground(Color.blue);
add("South", messageLabel);

//Add chess board to the center area of the window.
//      BorderLayout border = new BorderLayout();
//      centerPanel.setLayout(border);
myChessBoard= new ChessBoard(this);
// myChessBoard.disable();
centerPanel.add(myChessBoard);
add("Center", centerPanel);

//Add talk area to the window.
GridBagLayout gridBag = new GridBagLayout();
talkPanel.setLayout(gridBag);
GridBagConstraints c = new GridBagConstraints();
c.gridwidth = GridBagConstraints.REMAINDER;
c.fill = GridBagConstraints.HORIZONTAL;
Label labelFromCompetor= new Label("My Competor's Comments...", Label.LEFT);
gridBag.setConstraints(labelFromCompetor, c);
talkPanel.add(labelFromCompetor);

c.gridwidth = GridBagConstraints.REMAINDER;
c.fill = GridBagConstraints.HORIZONTAL;
gridBag.setConstraints(labelFromCompetor, c);
textAreaFromCompetor = new TextArea(5,20);
textAreaFromCompetor.setBackground(Color.lightGray);
textAreaFromCompetor.setEditable(false);
gridBag.setConstraints(textAreaFromCompetor, c);
talkPanel.add(textAreaFromCompetor);

```

```

c.gridwidth = GridBagConstraints.REMAINDER;
c.anchor= GridBagConstraints.WEST;
c.fill = GridBagConstraints.HORIZONTAL;
Label labelFromMyOwn = new Label("My Comments...", Label.LEFT);
gridBag.setConstraints(labelFromMyOwn, c);
talkPanel.add(labelFromMyOwn);

c.gridwidth = GridBagConstraints.REMAINDER;
c.fill = GridBagConstraints.VERTICAL;
gridBag.setConstraints(labelFromCompetor, c);
textAreaFromMyOwn = new TextArea (4, 20);
textAreaFromMyOwn.setEditable(false);
textAreaFromMyOwn.setBackground(Color.lightGray);
gridBag.setConstraints(textAreaFromMyOwn, c);
talkPanel.add(textAreaFromMyOwn);

c.gridwidth = GridBagConstraints.REMAINDER;
c.anchor= GridBagConstraints.WEST;
c.fill = GridBagConstraints.HORIZONTAL;
Label labelMyCommentTypeInArea=new Label("Type In My Comments: ", Label.LEFT);
gridBag.setConstraints(labelMyCommentTypeInArea, c);
talkPanel.add(labelMyCommentTypeInArea);

c.gridwidth = GridBagConstraints.REMAINDER;
c.fill = GridBagConstraints.VERTICAL;
gridBag.setConstraints(labelFromCompetor, c);
textFieldMyTypeIn = new TextField (20);
gridBag.setConstraints(textFieldMyTypeIn, c);
talkPanel.add(textFieldMyTypeIn);

validate();

add("East", talkPanel);

add("North", new Label("      Chess Board      ", Label.LEFT));
) // end of main in class GoGoWindow

public boolean action(Event event, Object arg) {
if(event.target instanceof TextField) {

```



```

String text;
String responseLine;
if (currentPlayMode==0) {
messageLabel.setText("Note: Please setup network connection first or
                        wait for incoming request!");

return true;
}
text = textFieldMyTypeIn.getText();
textFieldMyTypeIn.selectAll();
textAreaFromMyOwn.appendText(text + "\n");
communicationServer.sendMessage(text);
}

if (event.target instanceof MenuItem) {
if (((String)arg).equals("About")) {
MessageBox fd = new MessageBox(this, "About Window",
                                " InterGoGo (beta), by Dr. Zunhe JIN, 1996", true);
fd.show();
}
if (((String)arg).equals("Exit")) {
System.exit(0);
}
if ( ( (String)arg).equals("Start a connection...")) {
StartDialog startDialog= new StartDialog(this, "Start a connection... ");
startDialog.show();
}
if ( ( (String)arg).equals("Chang the listening port...")) {
ChangePortDialog portDialog= new ChangePortDialog(this, "Start a connection... ");
portDialog.show();
}
} // end of menu process
return true;
}

public boolean handleEvent(Event event) {
//If we're running as an application, closing the window
//should quit the application.
if (event.id == Event.WINDOW_DESTROY) {
if (inAnApplet) {

```

```

dispose( );
} else {
    System.exit(0);
}
}
return super.handleEvent(event);
}

public Dimension minimumSize( ) {
    return new Dimension(500,400);
}

public Dimension preferredSize( ) {
    return minimumSize( );
}

public Dimension maximumSize( ) {
    return minimumSize( );
}

public void setupPlayMode(int mode) {
    currentPlayMode= mode;
    if( mode ==0 ) myChessBoard.disableChessBoard( );
    else myChessBoard.enableChessBoard( );
}

public void initializeAll( ) {
    setupPlayMode(0);
    textAreaFromCompetor.setText("\n");
    textAreaFromMyOwn.setText("\n");
    myChessBoard.clearBoard( );
}

public static void main(String args[ ]) {
    GoGoWindow window = new GoGoWindow( );
    window.inAnApplet = false;
    window.setTitle("InterGo! ");
    window.pack( );
    window.resize(500,400);
}

```

```

window.setResizable(true);
window.show( );
communicationServer = new CommunicationServer( window );
communicationServer.start( );
window.setupPlayMode(0);    // disabled
}
}

// This DialogBox window will enable users to setup the host and port to connect
class StartDialog extends Dialog {
private GoGoWindow parent;
TextField fieldPort;
TextField fieldHost;
private Button setButton;

StartDialog(GoGoWindow dw, String title) {
super(dw, title, true);
parent = dw;

GridBagLayout gridBag = new GridBagLayout( );
setLayout(gridBag);
GridBagConstraints c = new GridBagConstraints( );

//Create top row.
Label label = new Label("Enter the host name:");
c.anchor = GridBagConstraints.WEST;
gridBag.setConstraints(label, c);
add(label);

c.fill = GridBagConstraints.HORIZONTAL;
c.gridwidth = GridBagConstraints.REMAINDER;
c.weightx = 1.0;
c.weighty = 1.0;
fieldHost = new TextField(20);
gridBag.setConstraints(fieldHost, c);
add(fieldHost);

// create the center row
c.fill = GridBagConstraints.NONE;

```

```

c.gridwidth = 1;          //GridBagConstraints.REMAINDER;
label = new Label("Enter the port number:");
c.anchor = GridBagConstraints.WEST;
gridBag.setConstraints(label, c);
add(label);

c.fill = GridBagConstraints.HORIZONTAL;
c.gridwidth = 1;          //GridBagConstraints.REMAINDER;
c.weightx = 1.0;
c.weighty = 1.0;
fieldPort = new TextField(6);
gridBag.setConstraints(fieldPort, c);
add(fieldPort);

//Create bottom row.
Panel panel = new Panel();
panel.setLayout(new FlowLayout(FlowLayout.RIGHT));
Button b = new Button("Cancel");
setButton = new Button("Set");
panel.add(b);
panel.add(setButton);
c.weightx = 0.0;
c.weighty = 0.0;
gridBag.setConstraints(panel, c);
add(panel);
resize(350, 125);
}

public boolean action(Event event, Object arg) {
if ( (event.target == setButton) | (event.target instanceof TextField)) {
parent.communicationServer.setupHost(fieldHost.getText());
parent.communicationServer.setupPort(Integer.valueOf(fieldPort.getText())
                                   .intValue());
if (parent.communicationServer.initConnection()) {
parent.messageLabel.setText(" Note: ready for you to play on the board and
                           chat with"+parent.communicationServer.getHost()+"at"+
                           parent.communicationServer.getPort());
parent.setupPlayMode(1);
}
}

```

```

else {
    parent.messageLabel.setText("Note: "+parent.communicationServer.getHost( )
        +" may be down or the connecting port is not correct!" );
}
}
fieldHost.selectAll( );
fieldPort.selectAll( );
hide( );
return true;
}
}

```

// This DialogBox window will enable users to setup the host and port to connect

```

class ChangePortDialog extends Dialog {

```

```

    private GoGoWindow parent;

```

```

    TextField fieldPort;

```

```

    private Button setButton;

```

```

    ChangePortDialog(GoGoWindow dw, String title) {

```

```

        super(dw, title, true);

```

```

        parent = dw;

```

```

        GridBagLayout gridBag = new GridBagLayout( );

```

```

        setLayout(gridBag);

```

```

        GridBagConstraints c = new GridBagConstraints( );

```

```

        // create the top row

```

```

        Label label = new Label("Enter the port number: ");

```

```

        c.anchor = GridBagConstraints.WEST;

```

```

        gridBag.setConstraints(label, c);

```

```

        add(label);

```

```

        c.fill = GridBagConstraints.HORIZONTAL;

```

```

        c.gridwidth = 1;        //GridBagConstraints.REMAINDER;

```

```

        c.weightx = 1.0;

```

```

        c.weighty = 1.0;

```

```

        fieldPort = new TextField(6);

```

```

        gridBag.setConstraints(fieldPort, c);

```

```

        add(fieldPort);

```

```

//Create bottom row.
Panel panel = new Panel ( );
panel.setLayout(new FlowLayout(FlowLayout.RIGHT));
Button b = new Button("Cancel");
setButton = new Button("Set");
panel.add(b);
panel.add(setButton);
c.weightx = 0.0;
c.weighty = 0.0;
gridBag.setConstraints(panel, c);
add(panel);
resize(350, 125);
}

public boolean action(Event event, Object arg) {
if ( (event.target == setButton) | (event.target instanceof TextField)) {
parent.communicationServer.setupPort(Integer.valueOf(fieldPort.getText( )).intValue( ));
parent.communicationServer.listenNewPort( );
}
fieldPort.selectAll( );
hide( );
return true;
}
}

// This is a message window for common purpose
class MessageBox extends Dialog {
public MessageBox(Frame parent, String title, String message, boolean modal) {
super(parent, title, modal);
resize(300,100);
add("Center", new Label(message, Label.CENTER));
Button button = new Button("OK");
Panel panel = new Panel ( );
panel.add(button);
add("South", panel);
}
}

```

```

public boolean action(Event event, Object arg) {
    if(event.target instanceof Button) {
        dispose();
    }
    return true;
    //return super.handleEvent(event);
}
}

// This class is used to maintain the communication socket and transfer data
// between the two participants.
class CommunicationServer extends Thread{
    Socket toPeerSocket= null;
    WaitingForConnectionThread waitingForConnectionThread=null;
    OutputStream os= null;
    DataInputStream is= null;
    Label messageLabel;
    GoGoWindow parent;
    TextArea textAreaFromCompetor;
    ChessBoard myChessBoard;
    String host="IBM-DNS";
    int port=4444;
    boolean onNewPort= true;

    CommunicationServer(GoGoWindow goGoWindow) {
        this.messageLabel= goGoWindow.messageLabel;
        this.textAreaFromCompetor= goGoWindow.textAreaFromCompetor;
        this.myChessBoard= goGoWindow.myChessBoard;
        this.parent= goGoWindow;
    } // end of constructor

    public void setupHost(String h) {
        host= h;
    }

    public void setupPort(int i) {
        messageLabel.setText(" Note: at port " + port );
        port=i;
    }
}

```

```

public String getHost() {
    return host;
}

```

```

public int getPort() {
    return port;
}

```

```

public void listenNewPort() {
    waitingForConnectionThread.stop();
    messageLabel.setText(" Note: change the newport"+port );
    onNewPort= true;
}

```

```

public void run() {
    String inputLine= null;

```

```

    while ( true ){
        while ( ! isEverythingOK() ) {
            if(onNewPort) {
                if(waitingForConnectionThread!=null) waitingForConnectionThread.stop();
                System.out.println(" before new listen\n");
                waitingForConnectionThread = new WaitingForConnectionThread(port);
                messageLabel.setText("Note:Waiting for incoming connection at port"+ port);
                waitingForConnectionThread.start();
                // messageLabel.setText(" Note: after run " + port );
                onNewPort= false;
            }

```

```

            if( toPeerSocket != null) break;
            if(( toPeerSocket=waitingForConnectionThread.getIncomingConnection())==null

```

```

        )
        continue;
        try {
            messageLabel.setText(" Note: Got imcomming request!");
            waitingForConnectionThread.stop();
            waitingForConnectionThread= null;
            is = new DataInputStream(toPeerSocket.getInputStream());
            os= toPeerSocket.getOutputStream();
        } catch (IOException e) {

```



```

    System.out.println(" the is and os is not right\n");
    e.printStackTrace();
}
if ( isEverythingOK() ) break;
System.out.println("There are something wrong\n");
} // end of internal while

messageLabel.setText("Note: connection is setup, ready to play!");
parent.setupPlayMode(1);
// the normal intercourse
while ( isEverythingOK() ) {
    try {
        inputLine = is.readLine();
        if (inputLine.charAt(0) != (char) 1) {
            textAreaFromCompetor.appendText(inputLine+"\n");
            textAreaFromCompetor.validate();
            messageLabel.setText("Note: Got a line!");
        }
        else {
            int i= Integer.valueOf(inputLine.substring(1)).intValue();
            messageLabel.setText("Note: " + host + " Put one at"+ i/19+", " + i%19);
            myChessBoard.oneFromCompetor(i/19, i%19);
            System.out.println("Put one at"+ i/19+", " + i%19);
        }
    } catch (Exception e) {
        messageLabel.setText(" Note: An error happens when reading socket!");
        break;
    }
} // end of internal while
MessageBox fd = new MessageBox(parent, "Error Window"
, "An error happened. listen on the port again", true);
fd.show();
parent.setupPlayMode(0);
System.out.println("before the cleanup");
cleanup();
System.out.println("After the cleanup");
onNewPort= true;
parent.initializeAll();
} // end of external while(true)

```

```

} // end of this function

public boolean sendAMessage(String message) {
    if ( ! isEverythingOK() ) {
        messageLabel.setText("Note: No connection is available now!");
        parent.setupPlayMode(0);
        return false;
    }
    try {
        for ( int i=0; i < message.length(); i++) os.write((byte)message.charAt(i));
        os.write((byte)'\n');
        os.flush();
    } catch (Exception e) {
        System.err.println("Exception:  " + e);
    }
    return true;
} // end of send a message

// this method is used to check the availability of socket,
// input and output stream
public boolean isEverythingOK() {
    boolean isEverythingOK= true;
    if ( toPeerSocket== null) {
        isEverythingOK=false;
    }
    if(os==null) {
        isEverythingOK=false;
    }
    if ( is== null) {
        isEverythingOK=false;
    }
    return isEverythingOK;
}

// this method is used to initialize the socket connection
public boolean initConnection() {
    if (waitingForConnectionThread!=null)
        //System.out.println("The waiting thread is alive");
        waitingForConnectionThread.stop();
    try {

```

```

messageLabel.setText(" Note: Initilizing the connection with"+host+"at port:
                        "+port+"... ");
toPeerSocket = new Socket(host, port);
System.out.println("Setup the socket");
os = toPeerSocket.getOutputStream( );
is = new DataInputStream(toPeerSocket.getInputStream( ));

if (isEverythingOK( )) {System.out.println("init is ok \n");
return true; }
} catch (Exception e) {
System.err.println("Exception:  " + e);
} // end of catch exception processing
System.out.println(" failed to init the socket\n");
return false;
}

// deconstructor
public void finalize( ) {
cleanup( );
}

// this method is used to clean all the socket and input, output stream
public void cleanup( ) {
try { if(is != null ) {
is.close( );
is=null;
}
} catch ( Exception e ) { } // Ignore error
try { if(os != null ) {
os.close( );
os=null;
}
} catch ( Exception e ) { } // Ignore error
try { if(toPeerSocket != null ) {
toPeerSocket.close( );
toPeerSocket=null;
}
} catch ( Exception e ) { } // Ignore error
} // end of cleanup

```

```

} // end of communication server thread class

class WaitingForConnectionThread extends Thread {
    int port;
    Socket toPeerSocket=null;

    WaitingForConnectionThread(int port) {
        this.port=port;
    }

    public void run() {
        ServerSocket serverSocket = null;
        try {
            serverSocket = new ServerSocket(port);
        }
        catch (IOException e) {
            System.out.println("Could not listen on port: " + port + ", " + e);
            System.exit(1);
        }
        try {
            System.out.println("before accept\n");
            System.out.println("waiting before the accept!");
            toPeerSocket = serverSocket.accept();
            // messageLabel.setText("after the accept!");
            serverSocket.close();
            System.out.println("after accept--- in waiting thread\n");
        } catch (IOException e) {
            System.out.println("Accept failed: " + 4444 + ", " + e);
            System.exit(1);
        }
        try {
            wait();
        } catch (Exception e) {}
    } // end of run

    public Socket getIncomingConnection() {
        return toPeerSocket;
    }

```

```
}  
}
```

```
/******  
CLASS: ChessBoard. It is used to simulate the chessboard  
*****/  
class ChessBoard extends Canvas {  
    byte chessStatus[] [] = new byte[19][19];  
    boolean enabled= false;  
    Label messageLabel;  
    GoGoWindow parent;  
    boolean token= true;  
  
    public ChessBoard(GoGoWindow go) {  
        for (int i=0; i<19; i++) for (int j=0; j<19; j++) chessStatus[i][j] = (byte) 0;  
        this.parent= go;  
        this.messageLabel = go.messageLabel;  
    }  
  
    public void clearBoard() {  
        for (int i=0; i<19; i++) for (int j=0; j<19; j++) chessStatus[i][j] = (byte) 0;  
        repaint();  
        enabled = false;  
        token= true;  
    }  
  
    public void enableChessBoard() {  
        enabled= true;  
    }  
  
    public void disableChessBoard() {  
        enabled= false;  
    }  
  
    public void paint(Graphics g) {  
        int i, j;  
        int w = size().width;  
        int h = size().height;
```

```

// draw the lines on the chessboard.
g.setColor(Color.lightGray);
g.fillRect(0, 0, w, h);

g.setColor(Color.black);
g.drawLine(w-1, 1, w-1, h);
g.drawLine(1, h-1, w, h-1);
for ( i=0; i<19; i++) {
g.drawLine(8, 15*i+ 8, w - 9, 15*i+8 );
g.drawLine(15*i+8, 8, 15*i+8, (h - 9));
}
// draw the chess-man , including black & white.
for ( i=0; i<19; i++) for( j=0; j<19; j++) {
if (chessStatus[i][j]==(byte)1) {
g.setColor(Color.black);
g.fillOval(15*i+1, 15*j+1 , 14 , 14);
}
if (chessStatus[i][j]==(byte)2) {
g.setColor(Color.white);
g.fillOval(15*i+1, 15*j+1, 14, 14);
}
}
System.err.println("AFTER OVAL ");
}

//If we don't specify this, the canvas might not show up at all
//(depending on the layout manager).
public Dimension minimumSize( ) {
return new Dimension(286, 286);
}

//If we don't specify this, the canvas might not show up at all
//(depending on the layout manager).
public Dimension preferredSize( ) {
return minimumSize();
}

public void oneFromCompetor(int i, int j)

```

```

    {
        chessStatus[i][j]= (byte)2;
        repaint( );
        token= true;
        System.out.println("in oneFromCompetor");
    }

    public boolean mouseDown(java.awt.Event evt, int x, int y) {
        int i, j;

        System.out.println("inside the mousemove!");
        requestFocus( );
        if (enabled==false) {
            messageLabel.setText("Note: Please setup network connection at first
                                or wait for incoming request!");
            return false;
        }
        if ( token == false ) {
            messageLabel.setText(" Note:Please be reminded that it is the turn of your
                                competor! " );
            return false;
        }
        Rectangle PointCover = new Rectangle( x-5, y-5, 10 , 10 );
        for (i=0; i<19; i++) for (j=0; j<19; j++)
            if (PointCover.inside(15*i+8, 15*j+8 )) {
                if (chessStatus[i][j]!=0) {
                    chessStatus[i][j]= (byte) 1;
                    token = false;
                    messageLabel.setText("Note: put one at ("+i +", "+ j + ")");
                    int temp;
                    parent.communicationServer.sendMessage( "\ 01"+String.valueOf(19*i+j));
                    repaint( );
                }
                else {
                    messageLabel.setText("Note: It is already occupied!");
                }
                return true;
            }
        else {

```

```
// play(getCodeBase( ), "sounds/danger,danger...!.au " );  
}  
messageLabel.setText(" Note: please click a little closer to the cross  
                        point!");  
repaint( );  
return true;  
}  
}
```


第十章 Java 原生函数的应用

在 Java 程序中可以使用由其它编程语言实现的函数，这种函数称为原生 (native) 函数，目前，Java 还只是提供了将 C 程序代码集成到 Java 程序中的机制。这一章我们将分析在 Java 程序中使用原生函数的优缺点，由 C 语言编写的原生函数集成到 Java 程序中的方法，以及原生函数在数据库开发中的应用。

10.1 概 述

对于大多数 C/C++ 程序员来说，可能特别关心如何将 C 语言函数集成到 Java 程序中，因为如果能利用已有的程序代码无疑将省去大量的劳动。然而，事情并没有想象的那么好，在 Java 程序中使用由 C 语言编写的原生函数是要付出一定代价的。

首先，任何包含原生函数的 Java 类都不能在 Applet 中使用，出于安全性的考虑，几乎所有的 WWW 浏览器都不允许 Applet 调用原生函数。Java 安全管理机制难以防止来自原生函数的恶意攻击，唯一的解决办法就是不允许调用原生函数。

其次，使用原生函数将使你的应用程序失去可移植性。Java 的一项主要优势就是程序代码可在不同的平台移植，Java 采用的字节码格式使得你只需要编译一次就可以在各种平台上运行。一旦选择使用原生函数，程序就失去了这种能力，而且，你将必须为运行该 Java 程序的每种平台编写不同的 C 语言链接库。

既然原生函数有这么多不利的影响，那我们为什么还要使用它呢？最主要的原因是原生函数增加了 Java 标准类中不存在的功能。由于 Java 的可移植性，它不能利用与操作系统相关的特点，因此当要访问特定的硬件设备或使用新的网络驱动器时，就不得不使用原生函数提供的功能。Java 开发商正在努力提供满足各种所需功能的标准类，然而这是很难做到的，调用由 C 语言编写的原生函数毕竟得到了 Java 标准类不提供的功能，实际上，Java 标准类本身的许多功能也是通过原生函数调用来完成的。

10.2 Java 原生函数的实现过程

Java 原生函数首先必须在 Java 类中定义，任何的 Java 函数都可以变换为原生函数，这只需要删除该函数的实现代码，在函数前增加一个关键词 native。例如：

```
public int myMethod(byte[] data)
{
    ...
}
变为
public native int myMethod(byte[] data);
```

该函数将在 C 语言动态链接库中实现, 因此还需要在 Java 类中使用如下的语句来说明装载的动态链接库:

```
static
{
    System.loadLibrary("myMethodLibrary");
}
```

当 Java 类初始化时, 就将执行上面的代码, 这样即可载入相应的动态链接库。如果执行上面的代码失败, 则不能生成该 Java 类, 这就防止了没有动态链接库时去调用原生函数的可能。

下面我们用一个例子来说明 Java 原生函数的实现方法。

10.2.1 编写 Java 类

我们要实现的 Java 类主要完成文件输入/输出, 它实际上结合了 Java 类库中 `java.io.File` 和 `java.io.RandomAccessFile` 的一些简单功能, 该类的定义如下:

```
public class SimpleFile {
    public static final char separatorChar = '>';
    private protected String path;
    private protected int fd;

    public SimpleFile(Strings) {
        path = s;
    }

    // 从路径 path 中获取文件名
    public String getFileName() {
        int index = path.lastIndexOf(separatorChar);

        return (index < 0) ? path : path.substring(index + 1);
    }

    public String getPath() {
        return path;
    }

    // 下面是将用 C 程序实现的原生函数
    public native boolean open();
    public native void close();
}
```

```

public native int read(byte[] buffer, int length);
public native int write(byte[] buffer, int length);

static {
    System.loadLibrary("Simple");    // 在类初始化时执行
}
}

```

在 SimpleFile 类中定义了四个原生函数，它们将在后面用 C 程序代码来实现，并生成动态链接库 Simple.dll，因而这里需要载入该动态链接库。

SimpleFile 类的使用没有什么特别之处，只需在 Java 程序中按如下的方式调用：

```

SimpleFile f = new SimpleFile(">some>path>and>fileName");
f.open();
f.read(...);
f.write(...);
f.close();

```

10.2.2 生成 C 头文件和存根文件

C 程序库中供 Java 调用的入口称为存根 (stub)，当在 Java 程序中执行原生函数时，就进入了存根代码。为了简化从 Java 代码到 C 代码过渡的编程工作，Java 提供了一个工具来生成 C 的头文件和存根模块。

Javah 是用来为 Java 类生成 C 文件的工具，命令格式如下：

```
javah [option] class
```

表 10.1 列出了所有选项，缺省时，javah 将在当前目录下为命令行中的每个 class 文件生成一个 C 头文件。

表 10.1 javah 选项描述

选 项	描 述
-verbose	显示处理过程中的详细信息
-version	显示 javah 的版本
-o outputfile	使用指定的文件名作为输出
-d directory	在指定的目录生成输出文件
-td tempdirectory	指定另时使用的目录
-stubs	生成 C 代码存根文件
-classpath path	指定 Java 类库路径

要生成 C 头文件，首先要将 SimpleFile.java 编译成 SimpleFile.class，然后执行如下的命令：

```
javah SimpleFile
```

输出的 C 头文件 SimpleFile.h 如下:

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <native.h>
/* Header for class SimpleFile */

#ifndef _Included-SimpleFile
#define _Included-SimpleFile
struct Hjava_lang_String;

typedef struct ClassSimpleFile {
#define SimpleFile_separatorChar 62L
struct Hjava_lang_String *path;
long fd;
} ClassSimpleFile;
HandleTo(SimpleFile);
#ifdef __cplusplus
extern "C" {
#endif
extern /*boolean*/ long SimpleFile_open(struct HSimpleFile *);
extern void SimpleFile_close(struct HSimpleFile *);
extern long SimpleFile_read(struct HSimpleFile *, HArrayOfByte *, long);
extern long SimpleFile_write(struct HSimpleFile *, HArrayOfByte *, long);
#ifdef __cplusplus
}
#endif
#endif
```

同样地, 执行如下的命令可生成 C 存根文件:

```
javah -stubs SimpleFile
```

输出的 C 存根文件 SimpleFile.c 如下:

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <StubPreamble.h>

/* Stubs for class SimpleFile */
/* SYMBOL: "SimpleFile/open()Z", Java-SimpleFile-open-stub */
__declspec(dllexport) stack_item*Java-SimpleFile-open-stub(stack_item
*_P-, struct execenv *_EE-) {
```

```

extern long SimpleFile_open(void *);
_P-[0].i = (SimpleFile_open(_P-[0].p) ? TRUE : FALSE);
return _P- + 1;
}
/* SYMBOL: "SimpleFile/close()V", Java-SimpleFile-close-stub */
__declspec(dllexport) stack_item*Java-SimpleFile-close-stub(stack_item
*_P-, struct execenv*_EE-) {
extern void SimpleFile_close(void *);
(void) SimpleFile_close(_P-[0].p);
return _P-;
}
/* SYMBOL: "SimpleFile/read([BI])I", Java-SimpleFile-read-stub */
__declspec(dllexport) stack_item *Java-SimpleFile-read-stub(stack_item
*_P-, struct execenv *_EE-) {
extern long SimpleFile_read(void *, void *, long);
_P-[0].i = SimpleFile_read(_P-[0].p, ((_P-[1].p)), ((_P-[2].i)));
return _P- + 1;
}
/* SYMBOL: "SimpleFile/write([BI])I", Java-SimpleFile-write-stub */
__declspec(dllexport) stack_item *Java-SimpleFile-write-stub(stack_item
*_P-, struct execenv *_EE-) {
extern long SimpleFile_write(void *, void *, long);
_P-[0].i = SimpleFile_write(_P-[0].p, ((_P-[1].p)), ((_P-[2].i)));
return _P- + 1;
}

```

对生成的 C 头文件和存根文件不用作任何的修改，只需把它们编译和链接到你编写的 C 程序代码中。

10.2.3 编写 C 函数

现在就可以为 Java 原生函数编写 C 代码了。前面生成的 SimpleFile.h 给出了要实现的四个函数的原型，这里需要把它包含在所实现的 C 程序中。下面的 C 程序代码是在 UNIX 系统中的一个实现，我们把该文件命名为 SimpleFileNative.c。

```

#include "SimpleFile.h"
#include <sys/param.h> /* for MAXPATHLEN */
#include <fcntl.h> /* for O_RDWR and O_CREAT */

```

```

#define LOCAL_PATH_SEPARATOR

static void fixSeparators(char *p) {
    for ( ; *p != '\0'; ++p)
        if (*p == SimpleFile_separartorChar)
            *p = LOCAL_PATH_SEPARATOR;
}

long SimpleFile_open(struct HSimpleFile *this) {
    int fd;
    char buffer[MAXPATHLEN];

    javaString2CString(unhand(this)->path, buffer, sizeof(buffer));
    fixSeparators(buffer);
    if ((fd = open(buffer, O_RDWR | O_CREAT, 0664)) < 0)
        return(FALSE); /* or, SignalError( ) could "throw" an exception */
    unhand(this)->fd = fd; /* save fd in the Java world */
    return(TRUE);
}

void SimpleFile_close(struct HSimpleFile *this) {
    close(unhand(this)->fd);
    unhand(this)->fd = -1;
}

long SimpleFile_read ( struct HSimpleFile * this,
                        HArrayofByte*buffer,longcount) {
    char *data = unhand(buffer)->body; /* get array data */
    int len = obj_length(buffer); /* get array length */
    int numBytes = (len < count ? len : count);

    if ((numBytes = read(unhand(this)->fd, data, numBytes)) == 0)
        return(-1);
    return(numBytes); /* the number of bytes actually read */
}

long SimpleFile_write(struct HSimpleFile *this,
                      HArrayofByte *buffer, long count) {
    char *data = unhand(buffer)->body;
    int len = obj_length(buffer);

```

```

        return(write(unhand(this)->fd, data, (len < count ? len : count)));
    }

```

在上面的 C 程序代码中，用到了一些访问 Java 运行系统的宏和函数，我们在下面作些说明：

■ `Object *unhand(Handle *)`

返回一个对象数据的指针，指针类型不一定是 `Object *`，而是取决于参数 `Handle` 的类型。

■ `int obj_length(HArray *)`

返回一个数组的长度。

■ `SignalError(0, JAVAPKG "ExceptionClassName", "message")`

当从原生函数返回时，发出一个 Java 例外信息。

■ `javaString2CString(Hjava_lang_String *s, char *buf, int len)`

将 Java 字符串拷贝到预先分配的 C 缓冲区。

10.2.4 生成 C 动态链接库

最后需要将 `SimpleFile.c` 和 `SimpleFileNative.c` 编译、链接生成一个动态链接库 `Simple.dll`。在 UNIX 系统中，其命令如下：

```
cc -G SimpleFile.c SimpleFileNative.c -o Simple
```

至此就完成了 Java 原生函数的实现工作，你可以编写一段 Java 程序来进行测试。

10.3 在数据库开发中的应用

信息技术领域的许多应用都离不开对数据库的访问，然而由于标准的 Java 类还不提供数据库接口，因而数据库的读写只能采用 Java 原生函数的方法。这一节我们将提供两个 Java 类作为与数据库的接口，其中的一个类 `SQLStmt` 包含查询语句和返回的数据，另一个类 `Database` 则包括访问数据库的所有原生函数。

下面是类 `SQLStmt` 的实现代码，我们在后面用 C 语言实现的原生函数将直接访问该类的成员变量。

```

import java.io.*;
import java.lang.*;
import DBException;

/**
 * Class to contain an SQL statement and resulting data
 */
public class SQLStmt
{

```

```

// The query string
public String sqlStmt = null;

// The actual data from the query
private String result[ ][ ];
private int nRows, nCols;

// True if the query is successful
private boolean query = false;

/**
 * The long constructor, you must supply a query
 * string to use this constructor
 * @param stmt contains the query to execute
 */
SQLStmt(String stmt)
{
    sqlStmt = stmt;
    System.out.println("Statement: " + stmt);
}

/**
 * Return the number of rows in a query data set
 * @exception DBException if no query has been made
 */
public int numRows( )
    throws DBException
{
    if (!query)
        throw new DBException("No active query");
    return nRows;
}

/**
 * Return the number of cols in a query data set
 * @exception DBException if no query has been made
 */
public int numCols( )
    throws DBException

```



```

    {
    if (!query)
    throw new DBException("No active query");
    return nCols;
    }

    /**
     * Retrieve the contents of a row. Each column
     * is separated from the others by a pipe '|' character.
     * @param row is the row to retrieve
     * @exception DBException if invalid row
     */
    public String getRow(int row)
    throws DBException
    {
    if (!query)
    throw new DBException("No active query");
    else if (row >= nRows)
    throw new DBException("Row out of bounds");
    String buildResult = new String(" ");

    for (int x=0; x<nCols; x++)
        buildResult += (result[row])[x] + "|";
    return buildResult;
    }

    /**
     * Retrieve the contents of a column.
     * @param row, col is the column to retrieve
     * @exception DBException if invalid row or column
     */
    public String getColumn(int row, int col)
    throws DBException
    {
    if (!query)
    throw new DBException("No active query");
    else if (row >= nRows)
    throw new DBException("Row out of bounds");
    else if (col >= nCols)

```

```

        throw new DBException("Column out of bounds");
        return result[row][col];
    }

    public void allDone(String str)
    {
        System.out.println(str);
    }

    /**
     * Display the contents of the statement
     */
    public String toString()
    {
        String s = new String();

        if (query == false)
        {
            s += sqlStmt;
        }
        else {
            try {
                for (int x=0; x<nRows; x++)
                    s += getRow(x) + "\n";
            }
            catch (DBException de) {
                System.out.println(de);
            }
        }
        return s;
    }
}

```

SQLStmt 是一个完成双向功能的对象类，既要处理输入数据，又要处理输出数据。SQLStmt 类提供了几个按序提取数据的公用函数：numRows()、numCols()、getRow()、getColumn()，此外，在该类中还有一个存放输出数据的二维字符串数组，这就使得数据库接口函数要将表中的一行数据转换为字符串格式。

在 SQLStmt 类中使用了一个处理数据库例外的类 DBException，它的源代码如下：

```

public class DBException extends Exception
{
    DBException()
    {
        super();
    }
    DBException(String s)
    {
        super(s);
    }
}

```

虽然 SQLStmt 类包含所有的数据库信息，但它实际上并没有对数据库操作的接口，为了将功能专门化，特意实现了类 Database，它定义了访问数据库的原生函数的接口。Database 类的代码如下：

```

import java.lang.*;
import java.io.*;
import SQLStmt;
import DBException;

/**
 * Class to allow access to the database library
 */
public class Database
{
    // Table name to use (ODBC data source)
    public String tableName;

    /**
     * Lone constructor. A data source must be passed.
     * @param s holds the name of the data source to use
     */
    Database(String s)
    {
        tableName = s;
    }
}

```

```

    * Native method query
    * @param stmt holds the SQLStmt class to use
    * @exception DBException is thrown on any error
    */
    public synchronized native void query(SQLStmt stmt)
    throws DBException;
    public synchronized native SQLStmt sql(String stmt)
    throws DBException;

    static
    {
        System.loadLibrary("Database");
    }
}

```

Database 类中第一个原生函数使用一个 SQLStmt 类对象作为输入和输出的参数，而第二个原生函数以一个字符串作为输入参数并返回一个 SQLStmt 类对象作为输出。两个原生函数都定义为 synchronized，因为它们在动态链接库中的实现是单线程的，使用了全局变量作为缓存区，因此需要防止同时有多个线程进入 Java 原生函数。

按照 10.2 介绍的方法，下一步就是要生成 C 头文件和存根文件。在编译后执行 javah SQLStmt 命令得到输出的头文件 SQLStmt.h 如下：

```

/* DO NOT EDIT THIS FILE - it is machine generated */
#include <native.h>
/* Header for class SQLStmt */

#ifndef _Included_SQLStmt
#define _Included_SQLStmt
struct Hjava_lang_String;

typedef struct ClassSQLStmt {
    struct Hjava_lang_String *sqlStmt;
    struct HArrayOfArray *result;
    long nRows;
    long nCols;
    /*boolean*/ long query;
} ClassSQLStmt;
HandleTo (SQLStmt);

```

```

#ifdef __cplusplus
extern "C" {
#endif
#ifdef __cplusplus
}
#endif
#endif

```

虽然类 SQLStmt 中没有原生函数，但还是在标记原生函数的位置出现了 ifdef __cplusplus 语句，此外，二维数组 result 也转换成了 HArrayOfArray。

同样，执行命令 javah Database 输出的头文件 Database.h 如下：

```

/* DO NOT EDIT THIS FILE - it is machine generated */
#include <native.h>
/* Header for class Database */

#ifndef _Included_Database
#define _Included_Database
struct Hjava_lang_String;

typedef struct ClassDatabase {
    struct Hjava_lang_String *tableName;
} ClassDatabase;
HandleTo(Database);

#ifdef __cplusplus
extern "C" {
#endif
struct HSQLStmt;
extern void Database_query(struct HDatabase *, struct HSQLStmt *);
extern struct HSQLStmt *Database_sql(struct HDatabase *,
                                     struct Hjava_lang_String *);
#ifdef __cplusplus
}
#endif
#endif

```

两个原生函数出现在头文件的底部，由于该文件定义了原生函数，因而需要为它生成存根文件，执行命令 javah -stubs Database 输出的存根文件 Database.c 如下：

```

/* DO NOT EDIT THIS FILE - it is machine generated */
#include <StubPreamble.h>

/* Stubs for class Database */
/* SYMBOL: "Database/query(LSQLStmt;)V", Java_Database_query_stub */
__declspec(dllexport) stack_item *Java_Database_query_stub(stack_item
                    *_P_, struct execenv *_EE_) {
    extern void Database_query(void *,void *);
    (void) Database_query(_P_[0].p, ((_P_[1].p)));
    return _P_;
}
/*SYMBOL: "Database/sql(Ljava/lang/String;)LSQLStmt;", Java_Database_sql_stub */
__declspec(dllexport) stack_item *Java_Database_sql_stub(stack_item
                    *_P_, struct execenv *_EE_) {
    extern void* Database_sql(void *,void *);
    _P_[0].p = Database_sql(_P_[0].p, ((_P_[1].p)));
    return _P_ + 1;
}

```

现在剩下要做的就是实现访问数据库的原生函数，我们这里只是列出了文件 DatabaseImpl.c 的部分代码，它说明了如何操作 Java 数据结构。

```

#include <StubPreamble.h>
#include <javaString.h>
#include "Database.h"
#include "SQLStmt.h"
#include <stdio.h>
#include <sql.h>

#define MAX_WIDTH50
#define MAX_COLS 20
#define MAX_ROWS200

static SQLSALLINT nRows, nCols;
static SQLINTEGER namelen[MAX_COLS];
static char *cols[MAX_COLS];
static char *rows[MAX_ROWS][MAX_COLS];

```

```

bool_t throwDBError(char *description)
{
    SignalError(0, "DBException", description);
    return FALSE;
}

/*
 * Extract from local storage into the passed String array.
 */
void getTableRow(struct HDatabase *db, HArrayOfString *result, long row)
{
    int col;
    char *st;

    for (col=0; col<nCols; col++)
    {
        st = rows[row][col];
        unhand(result)->body[col] = makeJavaString(st, strlen(st));
    }
}

/*
 * Perform a database lookup using the passed HSQLStmt.
 */
void Database-query(struct HDatabase *db, struct HSQLStmt *stmt)
{
    int x;
    HArrayOfArray *all;
    HString *s;

    /* Read from the database into local stroage */
    if (doQuery(db, stmt) == FALSE)
        freeStorage();

    /* If we have data, store it in the class */
    if (nRows != 0 && nCols != 0)
    {
        /* Allocate the row array (1st dimension) */
        all = (HArrayOfArray *)ArrayAlloc(T-CLASS, nRows);
        if (!all)

```

```

{
    freeStorage();
    unhand(stmt)->query = FALSE;
    throwDBError("Unable to allocate result array");
    return;
}
/* Set the array into the HSQLStmt class object */
unhand(stmt)->result = all;

/* For each row, store the result string */
for (x=0; x<nRows; x++)
{
    /* Allocate the columns (2nd dimension) */
    all->obj->body[x] = ArrayAlloc(T-CLASS, nCols);

    /* Extract the data from local storage into the
     * HSQLStmt object.
     */
    getTableRow(db, (HArrayOfString *)all->obj->body[x], x);
}

/* Set final variables in the object to reflect the query */
unhand(stmt)->query = TRUE;
unhand(stmt)->nRows = nRows;
unhand(stmt)->nCols = nCols;

/* Print the results of the query by calling
 * allDone(HSQLStmt.toString());
 */
s = (HString *)execute-java-dynamic-method(0, (HObject *)stmt,
    "toString", "( )Ljava/lang/String;");
execute-java-dynamic-method(0, (HObject *)stmt, "allDone",
    "(Ljava/lang/String;)V", s);
}
else unhand(stmt)->query = FALSE;
}

/*
 * Create a HSQLStmt class object and pass it to the query routine.

```



```

    */
    struct HSQLStmt *Database_sql(struct HDatabase *db struct Hjava_lang_String *s)
    {
        HObject *ret;

        /* Create the Object by calling its constructor */
        ret = execute_java_constructor(0, "SQLStmt", FindClass(0, "SQLStmt", TRUE),
        "(Ljava/lang/String;)", s);
        if (!ret) return NULL;

        Database_query(db, (HSQLStmt *)ret);
        return (HSQLStmt *)ret;
    }

```

最后一步是编译链接生成动态链接库，你必须使用 Microsoft Visual C++ V2.0 以上的编译器，并在系统中安装了 ODBC32。下面是编译命令：

```
cl Database.c DatabaseImpl.c -FeDatabase.dll -MD -LD odb32.lib javai.lib
```

10.4 小 结

本章讨论了使用 Java 原生函数的优缺点及其实现的详细过程，并以数据库开发为例，说明了 Java 原生函数的应用。在你学完了本章的内容后，你就了解了 Java 运行环境本身是如何开发的，以及如何为你的应用增加一些较低层次的功能，这也许是 Java 语言开发中最复杂的内容之一。不过，随着 Java 功能的完善与强大，Java 原生函数的使用最终将退出历史舞台。

第十一章 Java 与虚拟现实

虚拟现实的目标是要让你对计算机模拟的世界产生栩栩如生的感觉，这就好象你亲身在现实世界当中游历一样，它常常需要依靠计算机三维图形和声音的效果。目前，在 Web 中描述虚拟现实是采用虚拟现实建模语言（VRML），正如 HTML 用来描述 Web 页面一样。而将 Java 与 VRML 结合起来使用，将为虚拟世界的模拟提供更加强大的功能。

本章将向你介绍一些 VRML 的基本知识、如何由 VRML 构造虚拟世界以及把 Java 与 VRML 结合在一起的应用例子。由于用于虚拟现实的 Java API 还处在发展初期，因此本章的内容具有一定的前沿性。

11.1 概 述

VRML 是在 1994 年春的第一次 WWW 会议上开始酝酿的，Tim Berners-Lee 和 Dave Ragget 等组织了一个称作“鸟羽”（BOF）的专题，讨论虚拟现实的问题和如何应用到 Web 上。一些研究人员描述了正在进行的与 Web 交互的三维图形虚拟工具的项目，与会者认为有必要为这些工具规定一种共同的语言，来指定三维场景描述和 WWW 链接，也就是说要为虚拟现实建立一种类似于 HTML 的语言。此后经过多次的提案和审议，终于建立了 VRML 标准。

VRML 是在 Internet 上描述三维对象的语言，由它构造的虚拟世界具有无限的广度和深度，其中的对象可以链接到文本、音频或视频文件、HTML 文件以及其它的虚拟世界。然而，VRML 并不是 HTML 的扩展，HTML 是为文本而不是为图形设计的，而且，VRML 需要更好地协调网络以达到最优化。一个典型的 VRML 场景往往要比 HTML 文本包含更多的内嵌对象，需要由更多的服务器来提供服务。此外，HTML 已成为普遍接受的标准，存在许多依赖于它的应用，如果由于 VRML 的问题而妨碍了 HTML 的应用，或由于考虑到与 HTML 的相容性而限制了 VRML 的设计都是不可取的，因而，VRML 是一种与 HTML 完全独立的网络语言。

为了领略虚拟现实世界的魅力，需要 VRML 浏览器的支持。VRML 浏览器与 VRML 的关系类似于标准浏览器（如 Mosaic 或 Netscape Navigator）与 HTML 的关系，它通过网络下载用 VRML 描述的虚拟世界，然后还原成三维图形并可让你在其中漫游。通过虚拟世界中的链接可以进入另一个虚拟世界或者传统的 HTML 页面。

目前，Internet 上比较流行的 VRML 浏览器有：

- Paper Software 公司的 WebFX （<http://www.paperinc.com>）
- SGI 公司的 WebSpace （<http://webpace.sgi.com>）
- Intervista 公司的 WorldView
- Chaco Communications 公司的 VR Scout

使用时，VRML 浏览器与标准的 Web 浏览器之间可互相通讯，因此当在虚拟世界中选择了一个指向 HTML 文件的链接，标准的 Web 浏览器将载入该 URL。同样，当在 Web 浏览器中选择一个 VRML 文件的链接时，它将把该 URL 或 VRML 文件传递给 VRML 浏览器。实际上，

Netscape Navigator 3.0 已提供了对 VRML 的支持, 你可以直接使用它来浏览虚拟世界。而且, 一些公司如 SGI、Intervista 等也实现了相应的 Netscape VRML 插件 (Plug-in), 这样就避免了在 VRML 浏览器和 Web 浏览器之间来回切换。

例如, 图 11.1 是在 Netscape Navigator 2.0 中安装了 SGI 的 Cosmo Player 插件后, 显示的 WWW 上的一个 VRML 场景, 显然, 它具有非常逼真的效果。

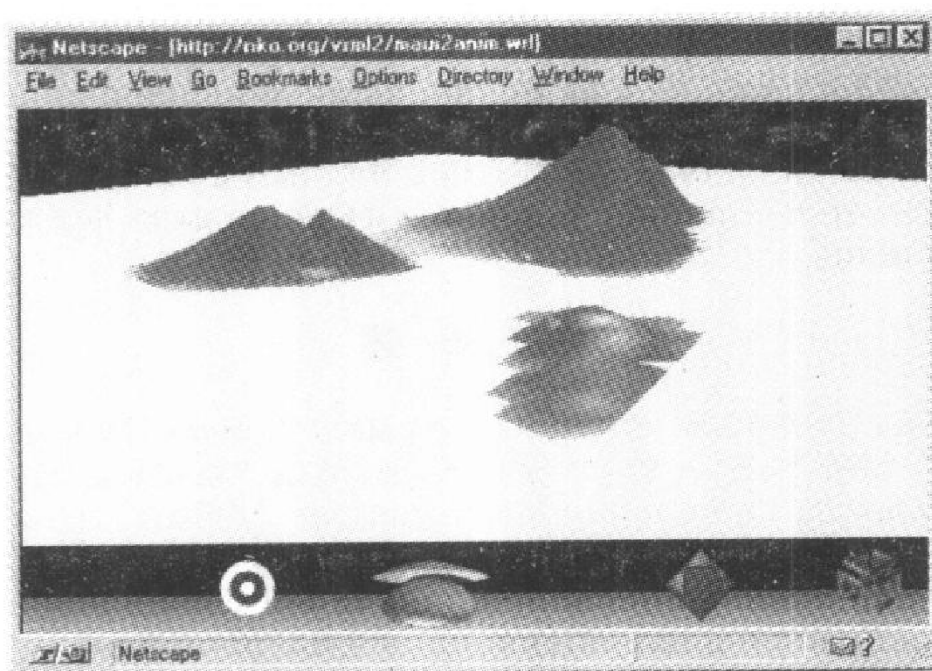


图 11.1 WWW 上的一个虚拟现实场景

11.2 虚拟现实世界的构造

构造虚拟现实世界非常具有挑战性, VRML 不象 HTML 那样容易使用, VRML 浏览器也不如 HTML 浏览器的容错性强。目前, 构造虚拟现实世界主要有两种方法, 方法之一是使用一个基于 Web 的 VRML 制作工具, 如 Aereal Phonts 等来构造虚拟现实世界; 方法之二是直接使用 VRML 代码来手工构造虚拟现实世界。

11.2.1 构造虚拟现实世界的流程

构造一个成功的虚拟现实世界需要按照一定的方法和步骤, 下面是一般的流程:

1) 定义项目

与其它的多媒体项目一样, 你首先需要对所建立的虚拟世界作一个很好的描述, 除了情节主线及应用定义, 你还需要清楚用户是谁, 用户将如何与虚拟世界交互, 以及用户会在什么样的平台上浏览。

2) 建立对象

在定义了项目以后, 你就要开始收集和建立构成虚拟世界的对象。当为三维世界新对

象建模时，一般使用多边形的模型器，这可让你比较容易地控制形状。此外，最好使用米制作为对象度量单位，因为虽然单位不太会影响用 VRML 1.0 编写的虚拟世界，但对 VRML 2.0 非常重要，不合适的单位会产生可笑的图形效果。

3) 动画与编排

当完成了虚拟世界的所有对象后，你还要给它们赋予生命力。对于动画序列，可使用 VRML 2.0 插补器 (Interpolator) 来实现主帧动画；对于虚拟世界的逻辑与行为，可使用 VRML Script 或 Java Script 节点来驱动。使用插补器和 Script 节点并不互相排斥，两者结合起来将使得虚拟世界具有更丰富的交互性。

4) 在目标平台上的测试

你用来建立虚拟世界的平台可能与用户使用的平台不一样，所以一定要在目标平台上对你制作的内容预览多次，通过修改来达到你希望的效果。目标平台的重要特征包括 CPU 能力、浏览器配置及调制解调器的连接等。如果没有做好这一步，有可能出现文件过大，图形变形等情况。

11.2.2 构造一个简单的虚拟现实世界

下面用一个简单的例子来说明虚拟现实世界的构造，这里我们使用了 VRML 2.0。

首先，需要在你的 Web 服务器中加入 VRML 的 MIME 类型，VRML 的 MIME 类型是 x-world/x-vrml。然后，我们就可以使用 VRML 来构造一个虚拟世界。

虚拟现实世界通常以下面一行作为开始，它是以 # 开头的一行注释，用来说明 VRML 的版本：

```
#VRML V2.0 utf8
```

按照 VRML 设计目的，在网络传送时注释将从 VRML 文件中摘除，如果你想在客户端显示一些诸如版权的信息，就要使用一个 INFO 节点。但实际上，目前许多 HTTP 服务器都还没有设置为将注释去掉。

每个虚拟现实世界都包括一个 Separator 节点，而它又包括一组各式各样的节点。节点可以表示为立方体、圆锥体等形状，颜色和纹理等属性。节点可以通过链接或内联对象与 WWW 建立联系，就象在 HTML 中通过 herfs 语句嵌入一个图形一样。例如，Material 就是一个用来指定颜色和透明度的属性节点。关于 VRML 节点的详细描述，可参考 VRML 规范 (<http://www.hyperreal.com/~mpesce/vrml/vrml.tech/vrml10-3.html>)。

下面，我们就以手工方式建立一个简单的虚拟世界，来说明 VRML 是如何工作的。图 11.2 是它在 Netscape Navigator 3.0 中的效果图。

虚拟现实世界的轮廓如下，它是一个有效的 VRML 文件，但把它载入 VRML 浏览器将看不到任何东西。

```
#VRML V2.0 utf8
```

```
Separator {  
}
```

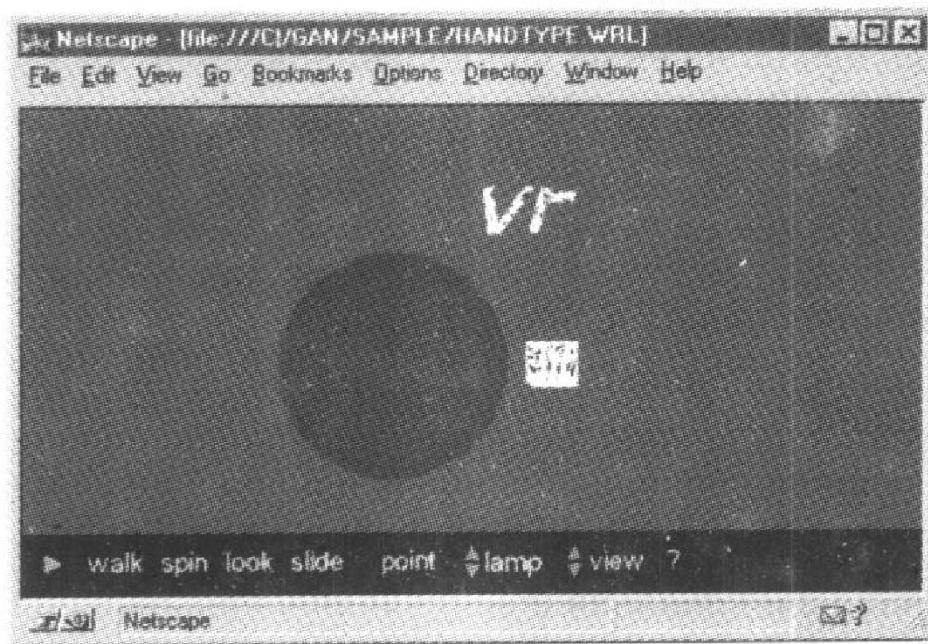


图 11.2 一个简单的虚拟现实世界

让我们在其中加入一个蓝色的球，使用 Material 节点将颜色设置为蓝色，然后增加一个 Sphere 节点，将半径设为 5 米。

```
#VRML V2.0 utf8
Separator {
  # blue sphere
  Material { diffuseColor 0 0 .7 } # sets the color to blue
  Sphere { radius 5 } # creates the sphere with radius 5 meters
}
```

现在在浏览器中就能看到一个蓝球了，我们还可以在其中增加更多的可视化的东西。VRML 浏览器可支持 JPEG、GIF 和 BMP 格式的图象文件，我们在虚拟世界中加入一个立方体，使用图象文件作为纹理，并通过使用一个 Translation 节点将它放在蓝球右边七米的地方。

```
#VRML V2.0 utf8
Separator {
  # blue sphere
  Material { diffuseColor 0 0 .7 } # sets the color to blue
  Sphere { radius 5 } # creates the sphere with radius 5 meters

  # textured cube
```

```

Translation { translation 7 0 0 } # moves away from the sphere
Texture2 { filename "image.gif" } # sets the texture to image.gif
Cube { } # creates the cube
)

```

其中 image.gif 是图象文件名, 你也可以使用 URL 的格式来表示。最后, 我们还要在其中加入一个内联对象。我们可以使用 WWWInline 节点嵌入指定 URL 的虚拟世界, 同样, 可通过 Translation 节点把它移到球和立方体上面中间的某个位置。

```

#VRML V2.0 utf8
Separator {
# blue sphere
Material { diffuseColor 0 0 .7 } # sets the color to blue
Sphere { radius 5 } # creates the sphere with radius 5 meters

# textured cube
Translation { translation 7 0 0 } # moves away from the sphere
Texture2 { filename "image.gif" } # sets the texture to image.gif
Cube { } # creates the cube

# WWWInline the Aereal Phonts world
Translation { translation -3.5 6 0 } # move above the sphere & cube
WWWInline { # render the following VRML world inside this world
name "vr.wrl"
}
}

```

这里, vr.wrl 是我们已构造的一个虚拟现实世界, 而它是用 VRML1.0 设计的, 其 VRML 格式如下:

```

#VRML V1.0 ascii
Separator {
Material { diffuseColor 0.89 0.47 0.20 }
Scale { scaleFactor 5 5 5 }
MatrixTransform { matrix 1 0 0 0 .3 1 0 0 0 0 1 0 0 0 0 1 }
Separator {
Translation { translation .1 .2 0 }
Separator {
Rotation { rotation 0 0 -1 -.4 }
DEF a Cylinder { radius .05 height .45 }

```

```

    }
    Translation { translation .2 0 0 }
    Separator
        Rotation { rotation 0 0 -1 .4 }
    USE a
}

Translation { translation .5 0 0 }
Separator {
    Translation { translation .05 .2 0 }
    Cylinder { radius .05 height .4 }
    Translation { translation .15 .12 0 }
    Separator {
        Rotation { rotation 0 0 -1 1.57 }
        Cylinder { radius .05 height .25 }
    }
}
Translation { translation .5 0 0 }
}

```

11.3 Java 在虚拟现实中的应用

按照 VRML 创始人之一 Mark Pesce 的说法：“Java 和 VRML 互相作了很好的补充，Java 主要讨论对象行为，但很少涉及外部特征，而 VRML 则着重于外表，不太考虑对象行为。可以说，VRML 所展现的正是 Java 所要做的，它们互相需要。”他认为，随着发展，Java 和 VRML 的关系将更加紧密，当人们谈到 Java 图象时，首先会想到 VRML；而当谈到如何驱动 VRML 时，则首先会想到使用 Java。

无论如何，Java 与虚拟现实的关系正越来越密切，许多公司都在竞相研制 Java 与 VRML 的接口，SGI、Dimension X 和 Paper Software 等公司已相继推出了用于虚拟现实的 Java API。

11.3.1 Java 与 VRML 的接口

Java 与 VRML 的接口是为虚拟现实世界与外部环境的通讯而设计的，它允许 Java Applet 使用 VRML 事件模型访问虚拟现实场景中的节点。接口分为以下四类：

- 访问 Browser Script 接口的功能。
- 给场景中节点的 EventIn 发送事件。

- 从节点的 EventOut 读最近发送的值。
- Applet 获取 EventOut 发送事件的通知。

Java Applet 与虚拟现实世界通讯时, 首先需要获取 Browser 类的实例。Browser 类是对虚拟现实世界所做的 Java 封装, 它包括整个 Browser Script 接口以及 getNode() 函数, Browser 类的定义如下:

```
// Specification of the External Interface for a VRML browser.
public class Browser {
    // Get the "name" and "version" of the VRML browser (browser-specific).
    public String getName();
    public String getVersion();

    // Get the current velocity of the bound viewpoint in meters/sec.
    public float getCurrentSpeed();

    // Get the current frame rate of the browser.
    public float getCurrentFrameRate();

    // Get the URL for the root of the current world, or an empty string.
    public String getWorldURL();

    // Replace the current world with the passed array of nodes.
    public void replaceWorld(Node[] nodes)
        throws IllegalArgumentException;

    // Load the given URL with the passed parameters ( as described in
    // the Anchor node ).
    public void loadURL(String[] url, String[] parameter);

    // Set the description of the current world in a browser-specific manner.
    public void setDescription(String description);

    // Parse STRING into a VRML scene and return the list of root nodes for
    // the resulting scene.
    public Node[] createVrmlFromString(String vrmlSyntax)
        throws InvalidVrmlException;

    // Tells the browser to load a VRML scene from the passed URL or URLs.
```



```

    public void createVrmlFromURL(String[] url, Node node, String event);

    // Get a DEFed node by name.
    public Node getNode(String name)
    throws InvalidNodeException;

    // Add and delete, respectively, a route between the specified eventOut
    // and eventIn
    // of the given nodes.
    public void addRoute(Node fromNode, String fromEventOut,
        Node toNode, String toEventIn)
        throws IllegalArgumentException;
    public void deleteRoute(Node fromNode, String fromEventOut,
        Node toNode, String toEventIn)
        throws IllegalArgumentException;

    // return an instance of the Browser class.
    static public Browser getBrowser();
}

```

1. 访问节点

一旦从 Browser 类的成员函数 `getNode()` 获得一个节点的实例, 就可访问该节点的 `EventIn` 和 `EventOut`。Node 类提供了成员函数 `getEventIn()` 和 `getEventOut()`, 它们用来获取指定名字的 `EventIn` 和 `EventOut`。Node 类的定义如下:

```

// Specification of the Java interface to a VRML node.
public class Node {
    // Get a string specifying the type of this node. May return the name of a PROTO,
    // or the class name.
    public String getType();

    // Means of getting a handle to an EventIn of this node
    public EventIn getEventIn(String name)
    throws InvalidEventInException;

    // Means of getting a handle to an EventOut of this node
    public EventOut getEventOut(String name)
    throws InvalidEventOutException;
}

```

2. 给 EventIn 发送事件

当获得 EventIn 的一个实例后, 就可以给它发送事件, 但 EventIn 是一个抽象类, 没有用于发送事件的成员函数, 因而首先必须把它转换成包含发送给定类型事件函数的 EventIn 子类。

我们用一个例子来说明, 假设 VRML 场景中包括如下的节点:

```
DEF Mover Transform {...}
```

下面的 Java 程序将给该节点发送事件来改变它的 *translation* 域(假定 *browser* 是前面调用中已得到的 Browser 类的实例)。

```
Node mover = browser.getNode("Mover");
EventInSFVec3f translation = (EventInSFVec3f) mover.getEventIn("set-translation");
float value[3] = new float[3];
value[0] = 5; value[1] = 0; value[2] = 1;
translation.setValue(value);
```

3. 访问 EventOut

同样, EventOut 也是一个抽象类, 为了获得 EventOut 的当前值, 必须把它转换成相应的 EventOut 子类。此外, EventOut 类的成员函数 *advise()* 可用来设置一个回调函数, 在生成 EventOut 对象时调用。

在上面的例子中, *translation* 域的当前值可通过如下的代码读取:

```
float current[] = ((EventOutSFVec3f) (mover.getEventOut(
    "translation-changed"))) .getValue();
```

4. EventOutObserver 类

当场景中节点输出事件时, Java Applet 为了得到通知, 必须定义为 EventOutObserver 的子类并实现 *callback* 函数。在以 EventOutObserver 类的实例和一个用户定义的对象作为参数调用 EventOut 的成员函数 *advise()* 后, 当节点输出事件时, 就将自动调用 *callback* 函数并把事件的值和时间标记传递给 Applet。

例如, 如果在上面的例子中加入如下的代码, 则当节点的 *translation* 域改变时, Applet 将得到通知:

```
public class MyObserver implements EventOutObserver {
    public void callback (EventOut value, double timeStamp, Object data)
    {
        // cast value into an EventOutSFVec3f and use it
    }
}
```

```

}
...
MyObserver observer = new MyObserver;
mover.getEventOut("translation-changed").advise(observer, null);

```

11.3.2 应用举例

这一节,我们将用一个比较典型的例子来说明 Applet 和虚拟现实世界之间是如何通讯的。该例包括一个 VRML 场景文件 sphere.wrl 和一个 Java Applet 程序 RGBTest.java, 它将在 Netscape Navigator 3.0 中显示为一个红色的球。

VRML 文件 sphere.wrl 的语句如下:

```

#VRML Draft #2 V2.0 utf8
Group {
  children [
    Shape {
      appearance Appearance {
        material DEF MAT Material {
          diffuseColor 0.8 0.2 0.2 # sets the color to red
        }
      }
      geometry Sphere { } # creates the sphere
    },
    DEF TOUCH TouchSensor {
  }
]
}

```

Java 程序 RGBTest.java 的源代码如下, 它涉及到事件的发送和接收以及回调函数的登记。

```

import java.awt.*;
import java.applet.*;
import vrml.field.EventOut;
import vrml.field.EventInSFColor;
import vrml.field.EventOutSFColor;
import vrml.field.EventOutSFTime;
import vrml.field.EventOutObserver;

```

```

import vrml.Node;
import vrml.Browser;
import vrml.exception.*;
import netscape.javascript.JSObject;

public class RGBTest extends Applet implements EventOutObserver {
    TextArea output = null;
    Browser browser = null;
    Node material = null;
    EventInSFColor diffuseColor = null;
    EventOutSFColor outputColor = null;
    EventOutSFTIME touchTime = null;
    boolean error = false;

    public void init() {
        // Set the layout of the applet
        add(new Button("Red"));
        add(new Button("Green"));
        add(new Button("Blue"));
        output = new TextArea(5, 40);
        add(output);

        // Get the handle to the VRML Browser
        JSObject win = JSObject.getWindow(this);
        JSObject doc = (JSObject) win.getMember("document");
        JSObject embeds = (JSObject) doc.getMember("embeds");
        browser = (Browser) embeds.getSlot(0);

        // Now we've got the handle to the VRML Browser.
        try {
            // Get the material node...
            material = browser.getNode("MAT");
            // Get the diffuseColor EventIn
            diffuseColor = (EventInSFColor) material.getEventIn("set_diffuseColor");
            // Get the Touch Sensor
            Node sensor = browser.getNode("TOUCH");
            // Get its touchTime EventOut
            touchTime = (EventOutSFTIME) sensor.getEventOut("touchTime");
            // Set up the callback

```

```

        touchTime.advise(this, new Integer(1));
        // Get its diffuseColor EventOut
        outputColor = (EventOutSFColor) material.getEventOut("diffuseColor");
        // Set up its callback
        outputColor.advise(this, new Integer(2));
    }
    catch (InvalidNodeException ne) {
        add(new TextField("Failed to get node: " + ne));
        error = true;
    }
    catch (InvalidEventInException ee) {
        add(new TextField("Failed to get EventIn: " + ee));
        error = true;
    }
    catch (InvalidEventOutException ee) {
        add(new TextField("Failed to get EventOut: " + ee));
        error = true;
    }
}

    public void callback(EventOut who, double when, Object which) {
Integer whichNum = (Integer) which;
if (whichNum.intValue() == 1) {
    // Make the timestamp and new time show up in the textarea.
    double val = touchTime.getValue();
    output.appendText("Got time " + val + " at time " + when + "\n");
}
if (whichNum.intValue() == 2) {
    // Make the new color of the sphere and timestamp
    // show up in the textarea.
    float[] val = outputColor.getValue();
    output.appendText("Got color " + val[0] + ", " + val[1] + ", " +
        val[2] + " at time " + when + "\n");
}
}

    public boolean action(Event event, Object what) {
if (error)
{

```

for

```

        showStatus("Problems! Had an error during initialization");
        return true; // Uh oh...
    }
    if (event.target instanceof Button)
    {
        Button b = (Button) event.target;
        if (b.getLabel() == "Red") // "Red" button is clicked
        {
            showStatus ("Red!");
            float[] val = new float[3];
            val[0] = 0.8f;
            val[1] = 0.2f;
            val[2] = 0.2f;
            diffuseColor.setValue(val); // change the color of the sphere to red
        }
        else if (b.getLabel() == "Green") // "Green" button is clicked
        {
            showStatus("Green!");
            float[] val = new float[3];
            val[0] = 0.2f;
            val[1] = 0.8f;
            val[2] = 0.2f;
            diffuseColor.setValue(val); // change the color of the sphere to green
        }
        else if (b.getLabel() == "Blue") // "Blue" button is clicked
        {
            showStatus("Blue!");
            float[] val = new float[3];
            val[0] = 0.2f;
            val[1] = 0.2f;
            val[2] = 0.8f;
            diffuseColor.setValue(val); // change the color of the sphere to blue
        }
    }
    return true;
}
}

```

在将 RGBTest.java 编译成 class 文件后，我们将它和 sphere.wrl 嵌入到同一 HTML

文件中，该 HTML 文件的格式如下：

```
<html>
<head>
<title> RGB Applet Test </title>
</head>
<center>
<embed src="sphere.wrl" border=0 height="250" width="375">
</p>
<applet code="RGBTest.class" mayscript>
</applet>
</center>
</html>
```

由于该 Applet 程序使用了基于虚拟现实的 Java API，因而它的运行需要特殊的 VRML 浏览器的支持。你可以从 Internet 上下载 SGI 公司的 Cosmo Player (<http://vrm1.sgi.com>)，这是一个支持虚拟现实 Java API 的 Netscape Plug-in 软件。安装了该软件后，在 Netscape 中打开上面的 HTML 文件，将看到图 11.3 所示的效果图。你如果按下“Red”、“Green”、“Blue”按钮作一测试，那么虚拟现实世界中球的颜色也将相应地变为红、绿、蓝色。

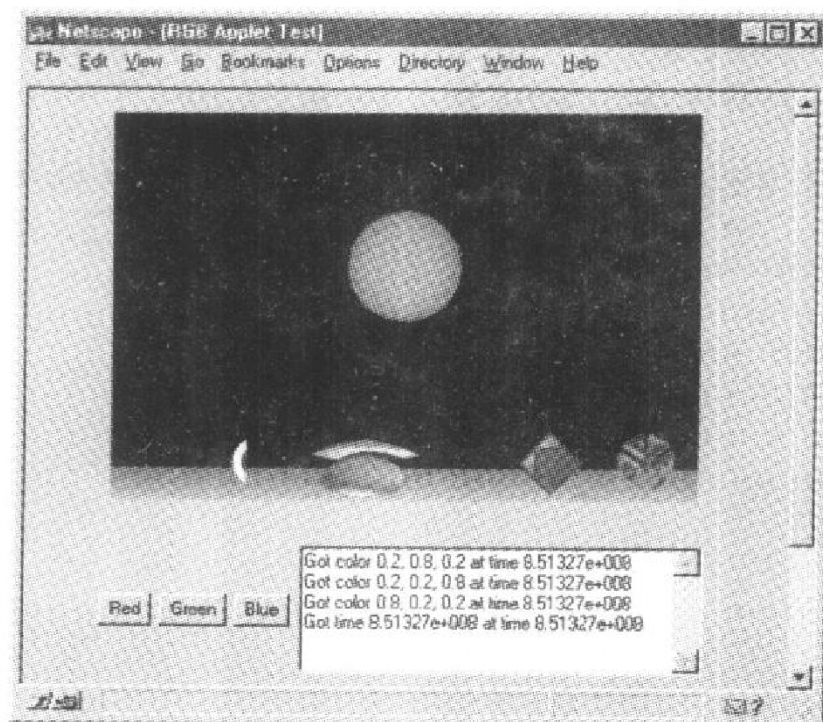


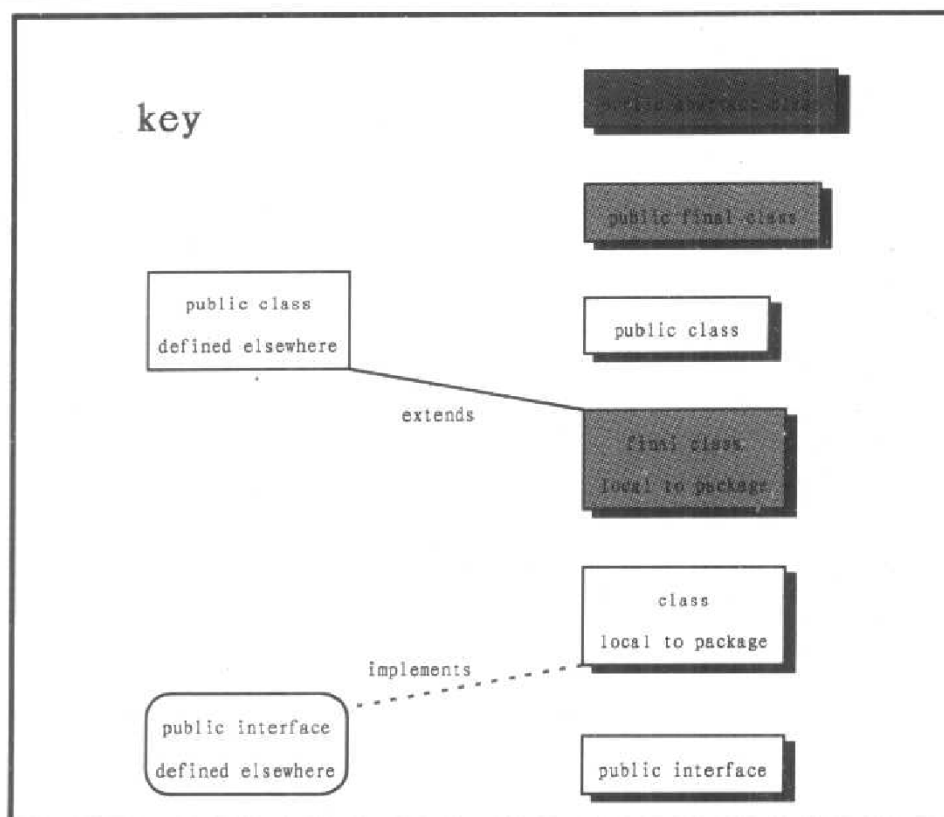
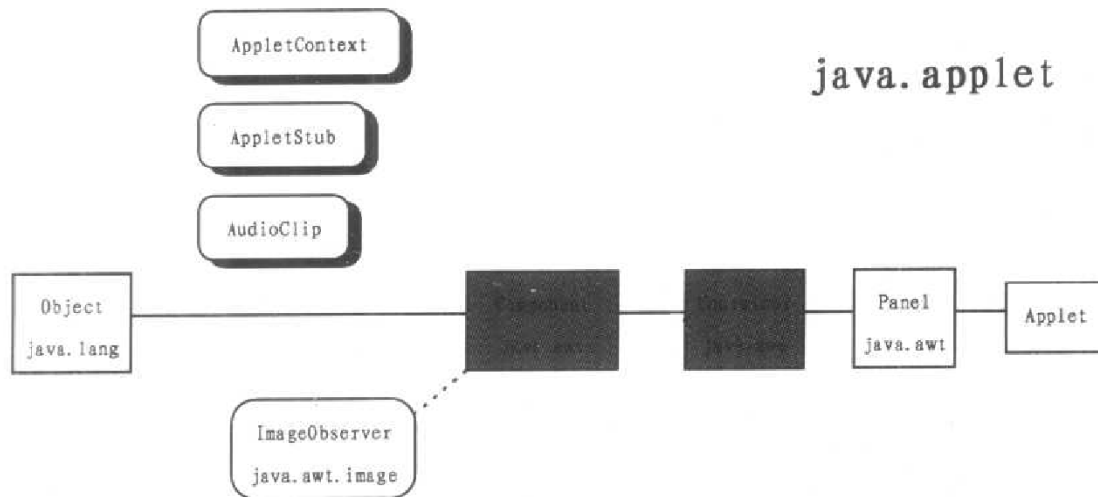
图 11.3 Java Applet 与虚拟现实世界的通讯

11.4 小 结

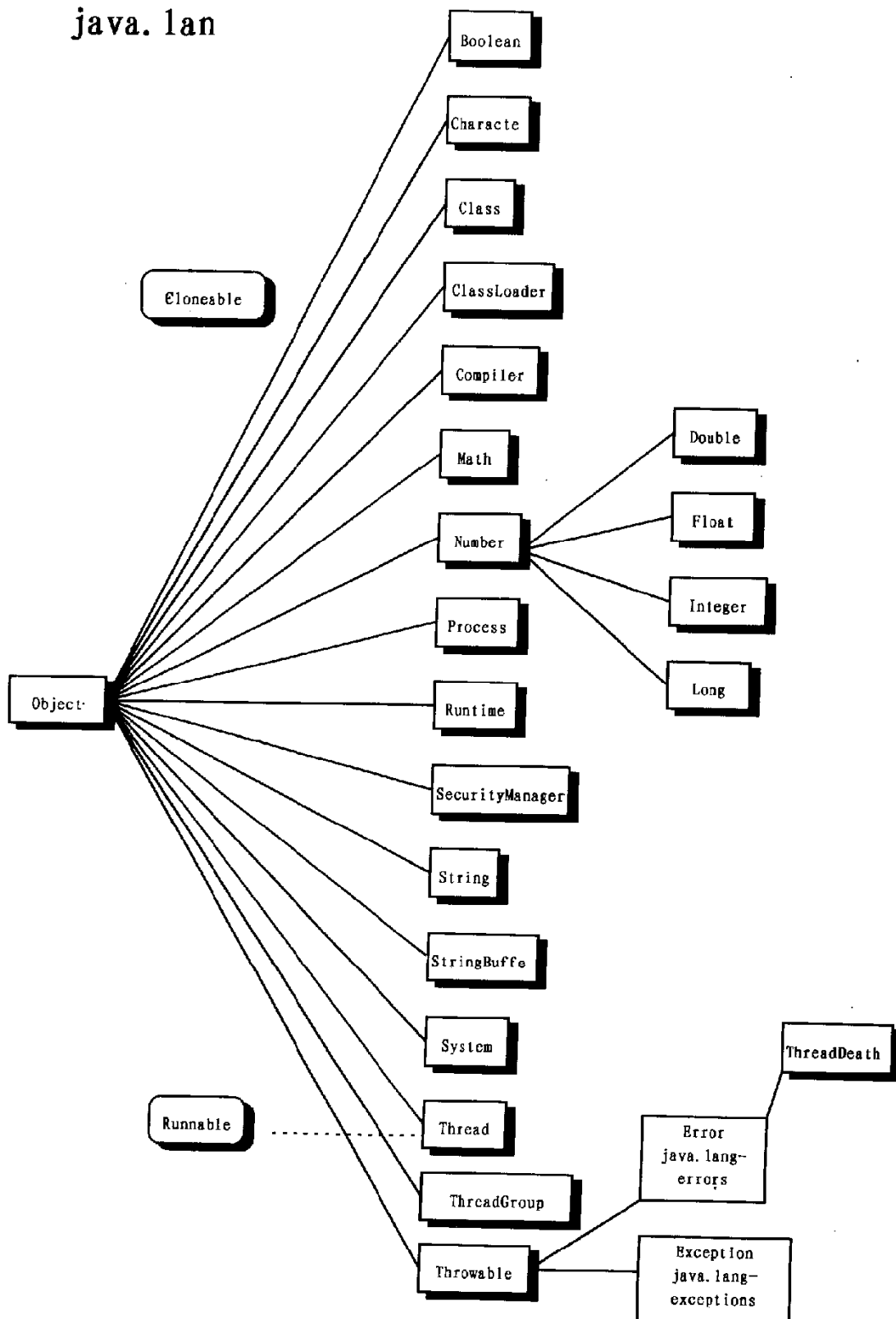
虚拟现实给 WWW 赋予了完美的交互性和生动感，随着 VRML 2.0 的推出，并与 Java 的强大功能相结合，将使虚拟现实更具有生命力，可以预见就象今天的 HTML 主页一样，Internet 上的虚拟现实世界将会如雨后春笋般地发展起来。

本章描述了虚拟现实的一些基本知识以及 Java 在虚拟现实中的应用，限于篇幅，不可能对 VRML 作比较详细的阐述，读者如果有兴趣，可参阅附录中列出的 Internet 上的一些资源。

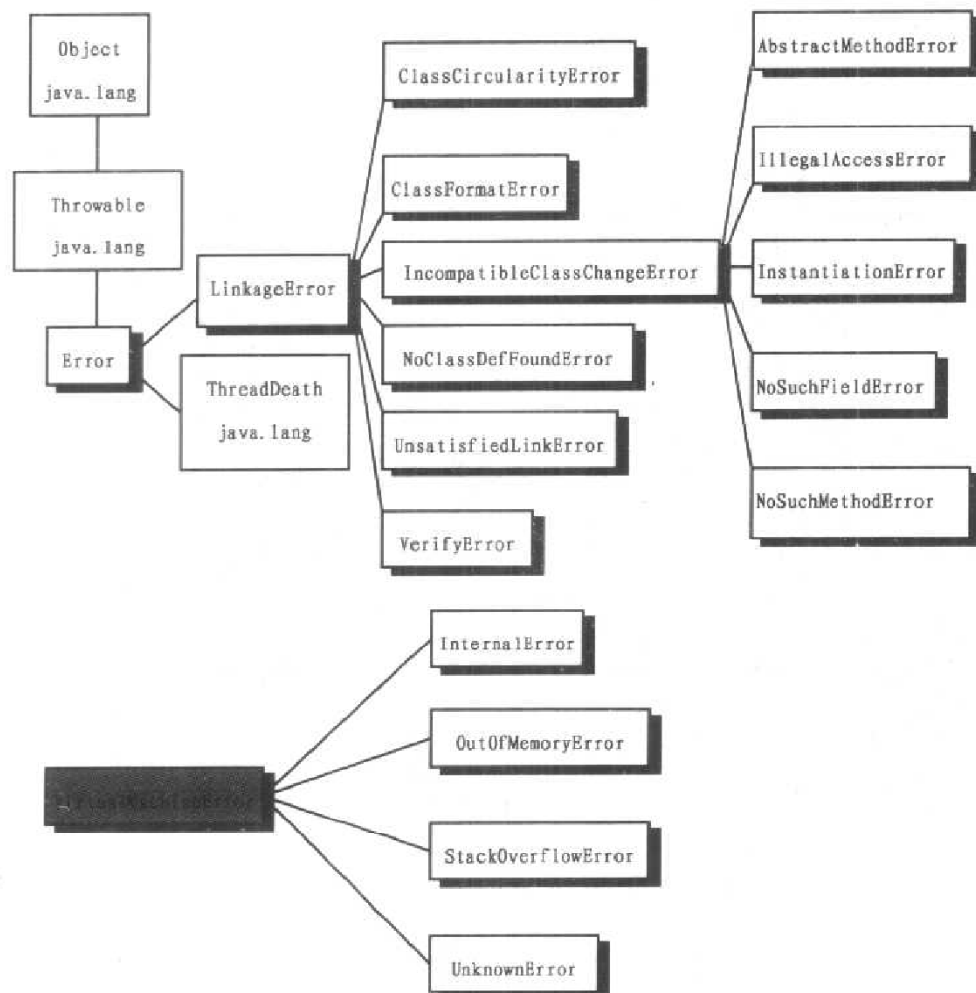
附录 A Java 类库层次关系图



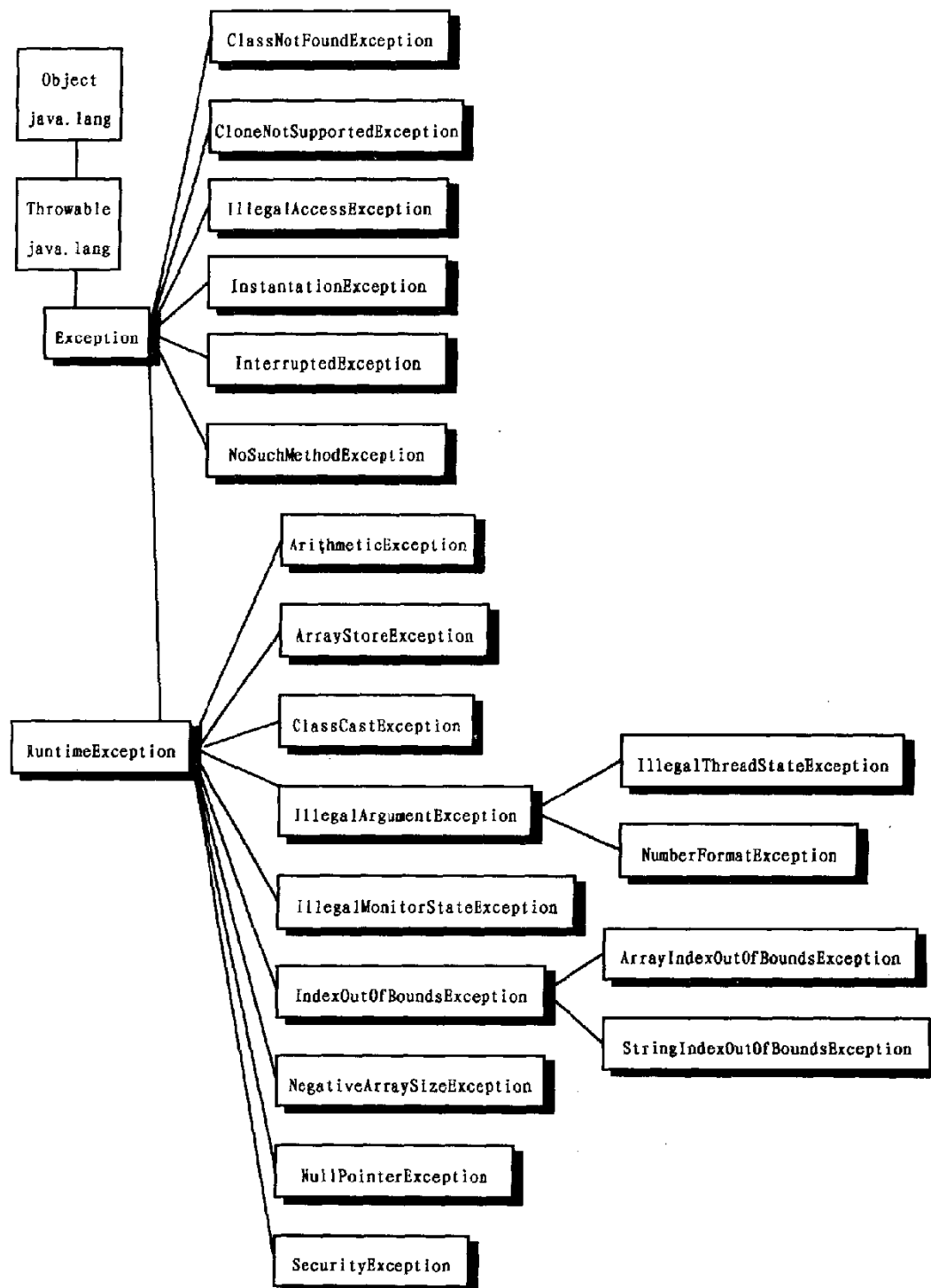
java. lan



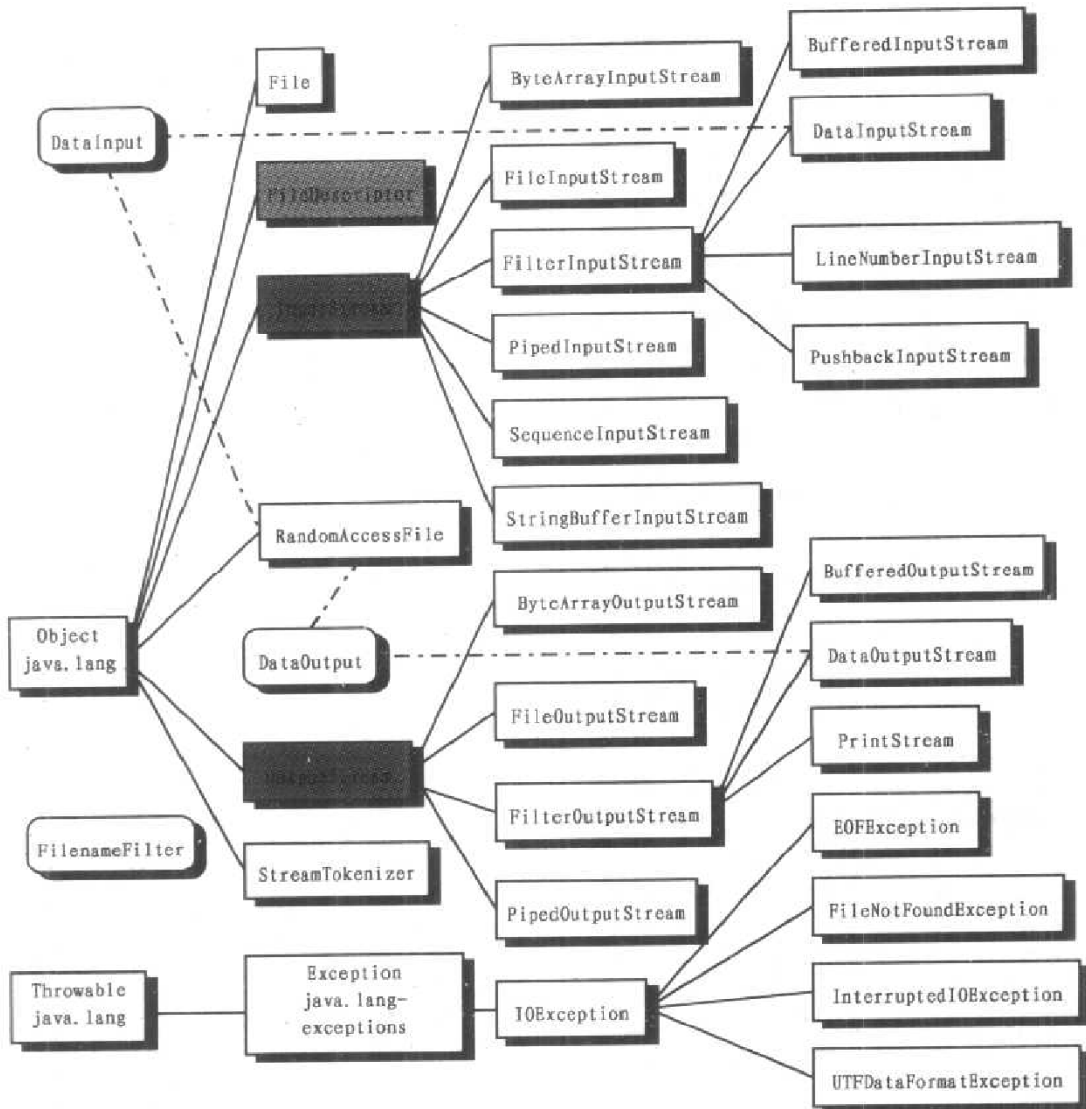
java.lang-



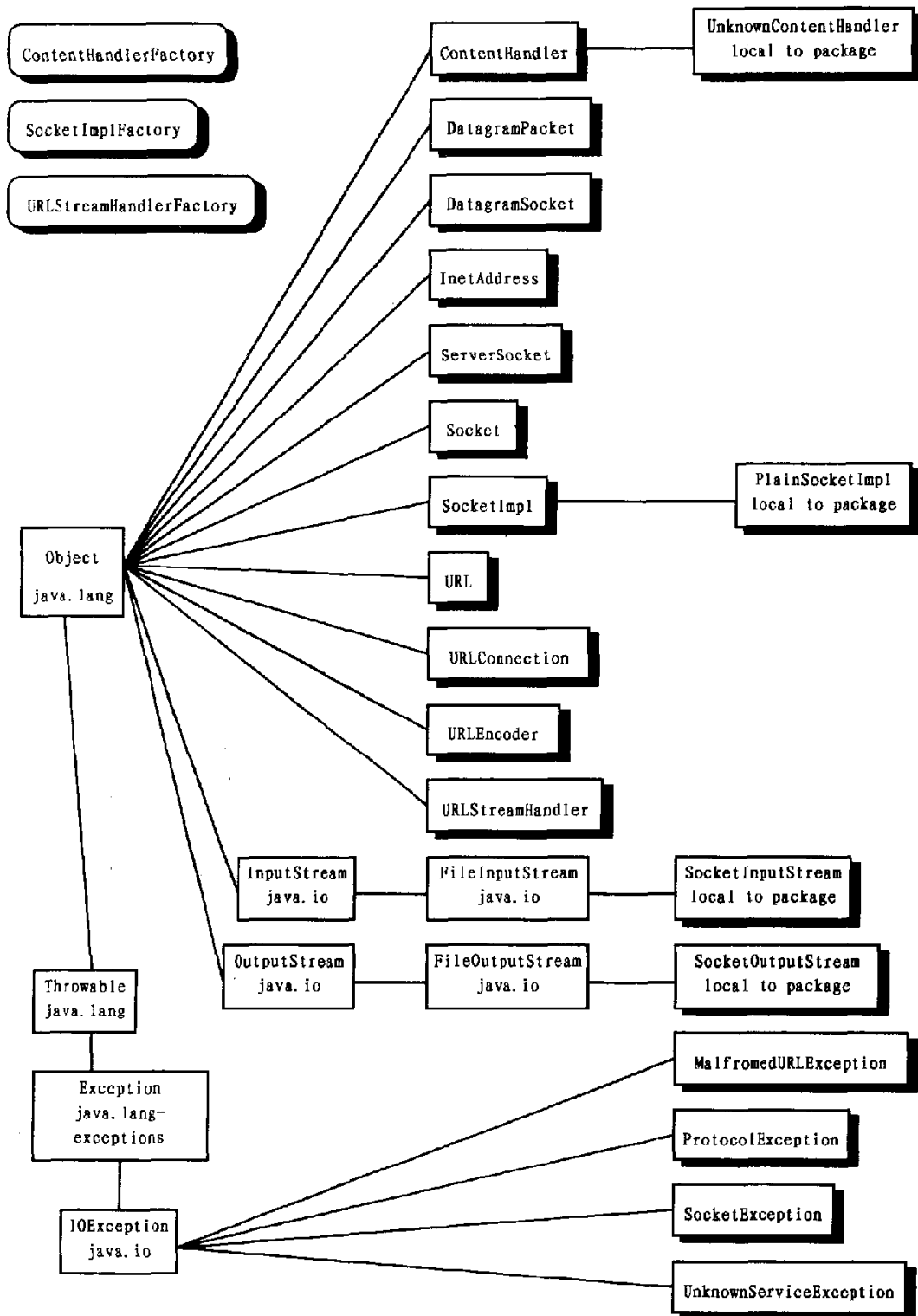
java.lang-



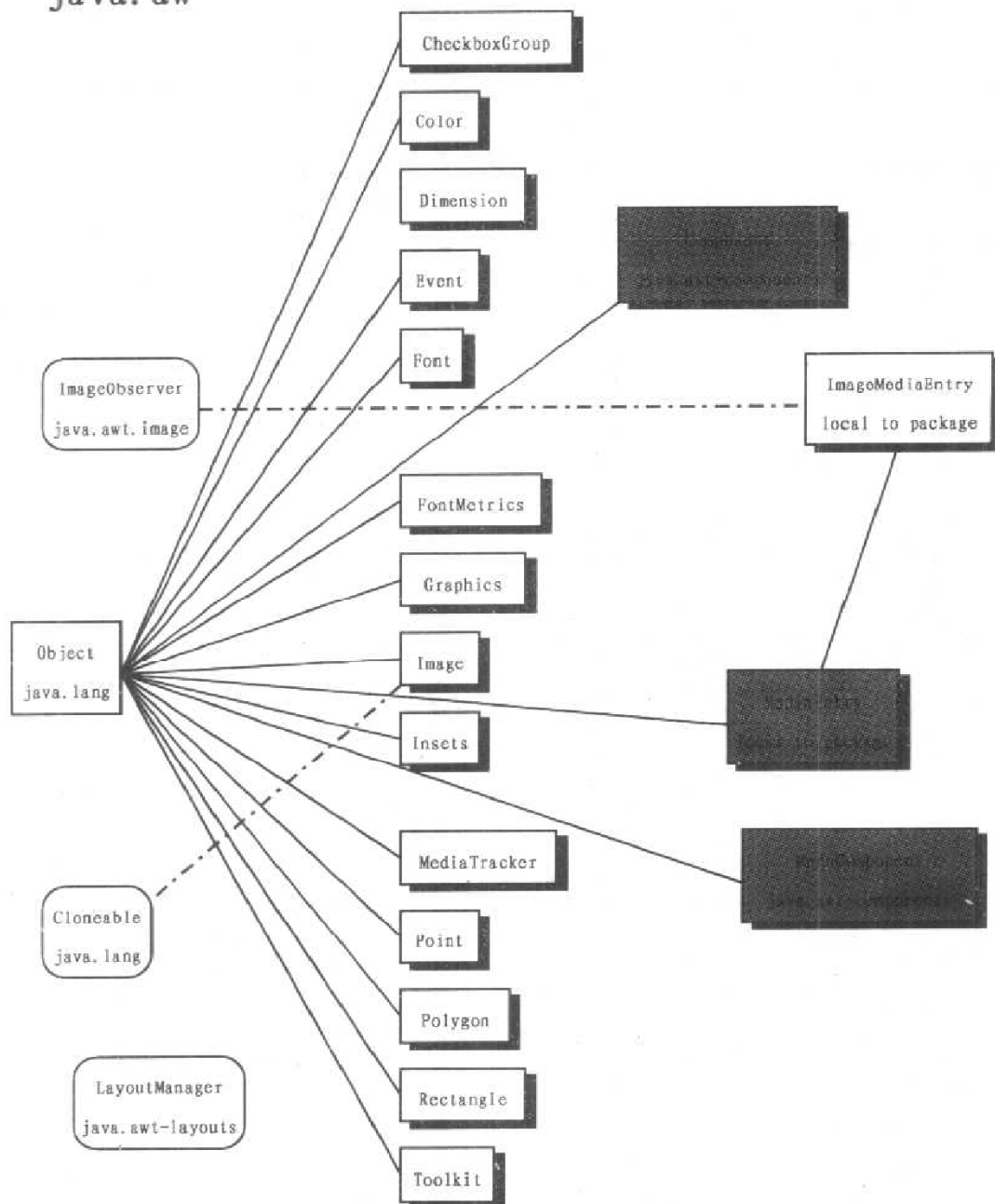
java. i



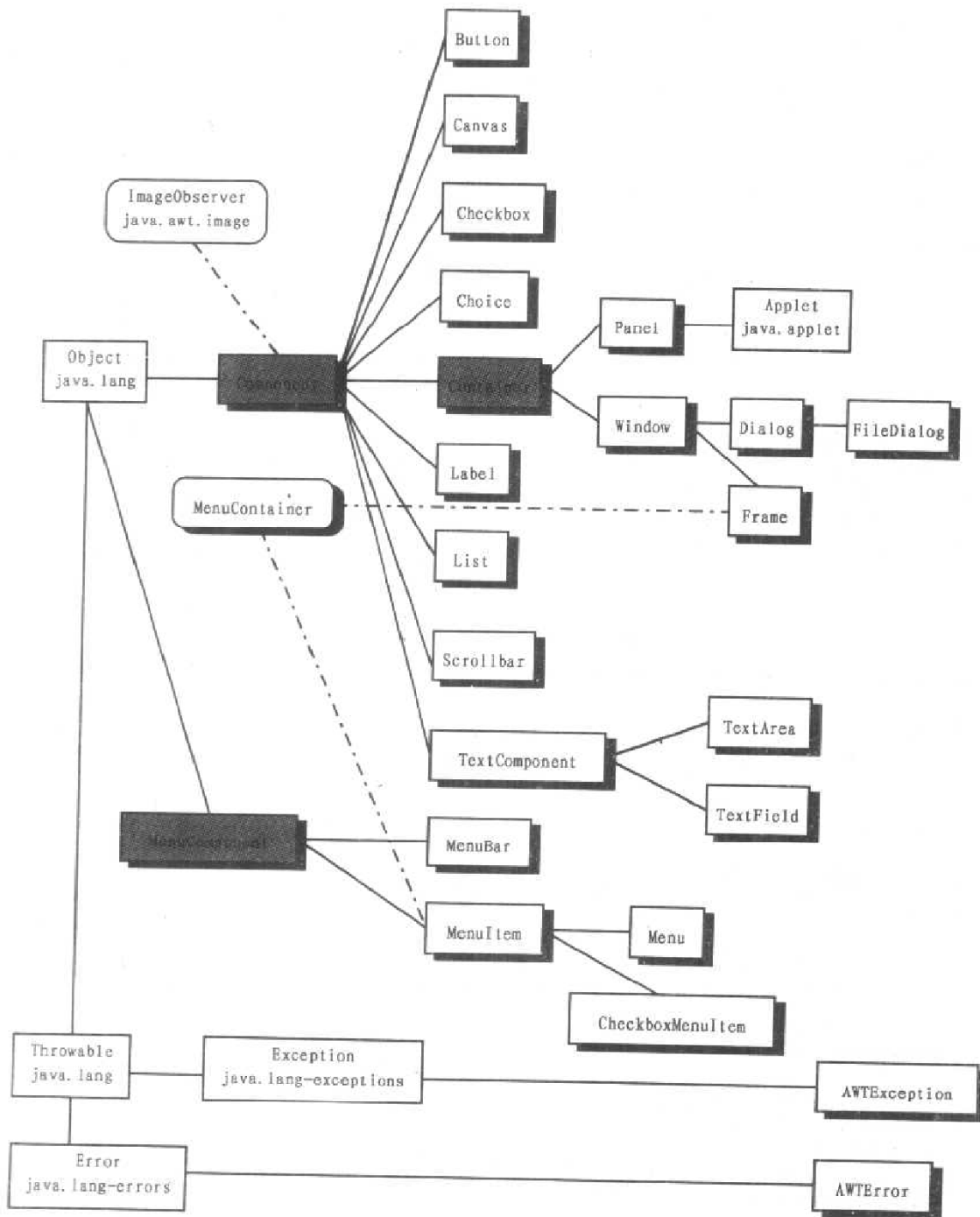
java. ne



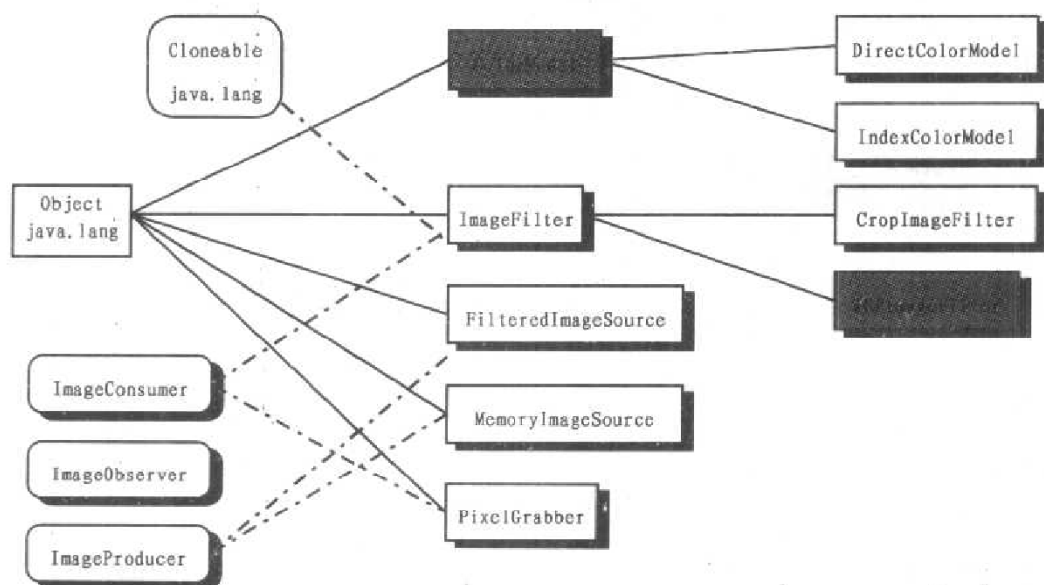
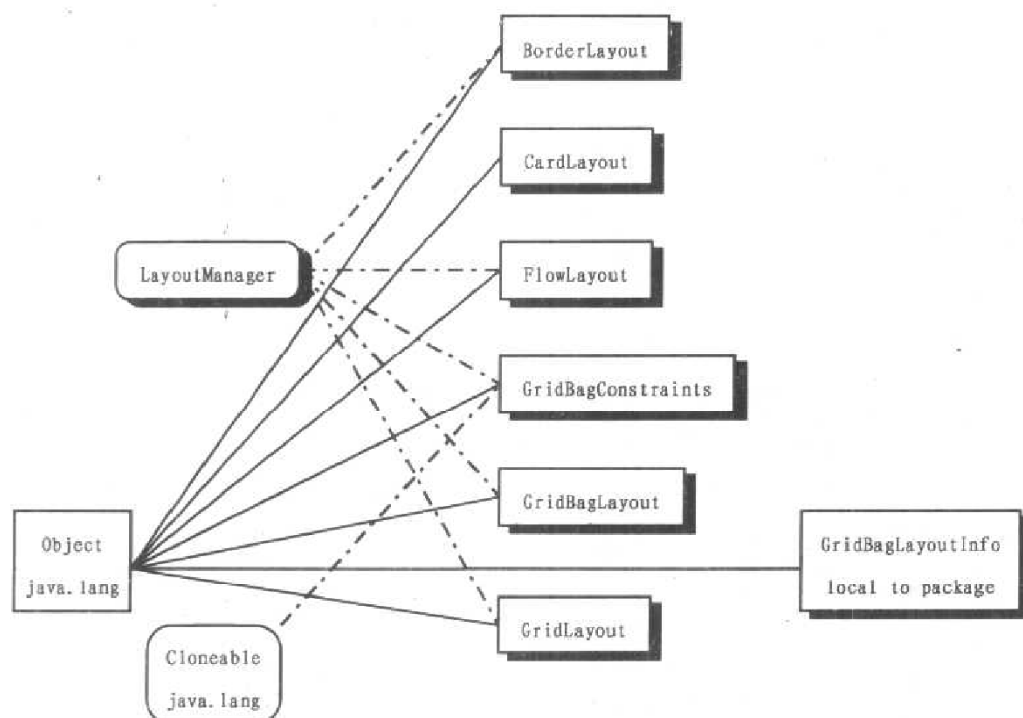
java. aw



java.awt-components

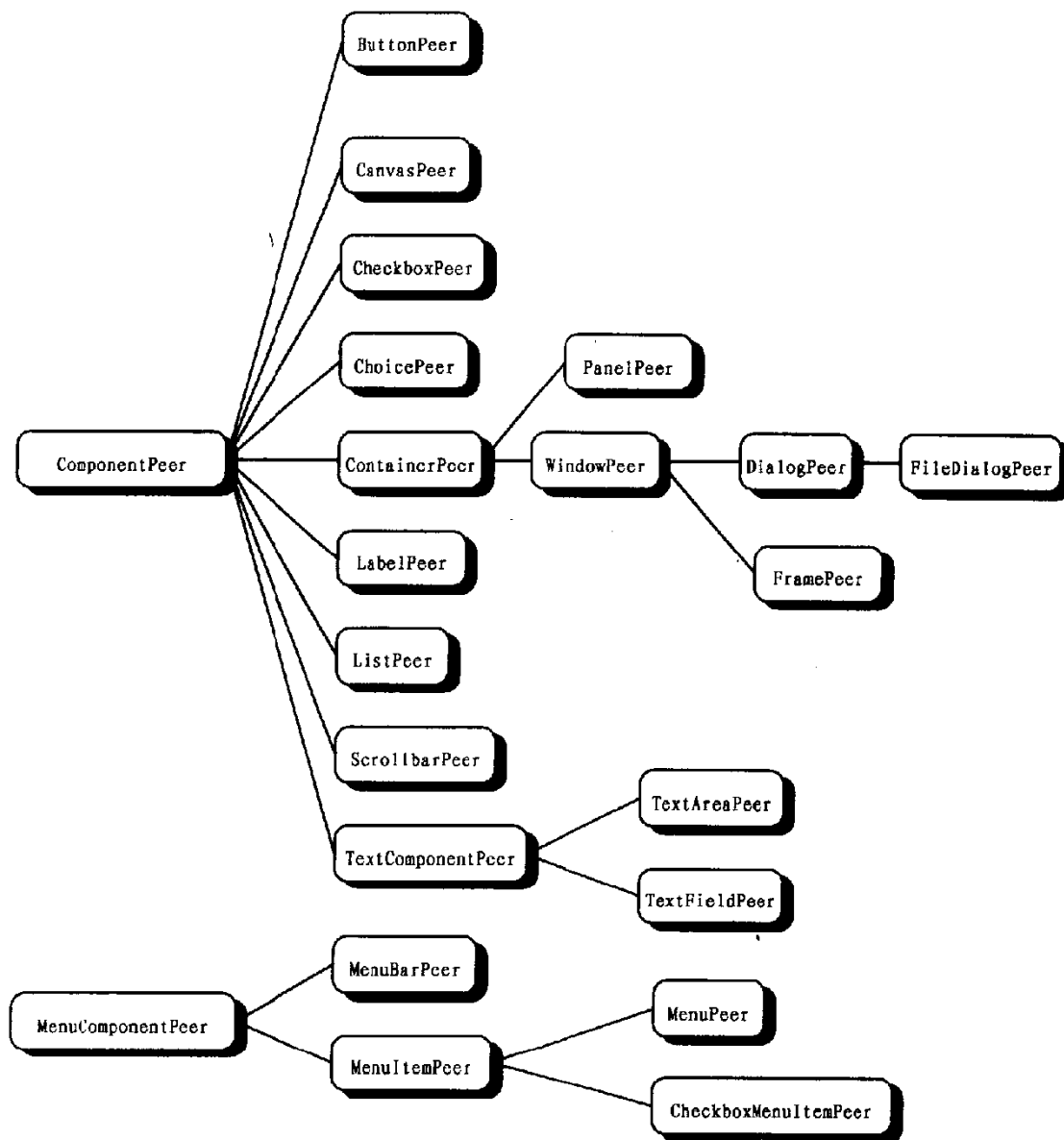


java.awt-

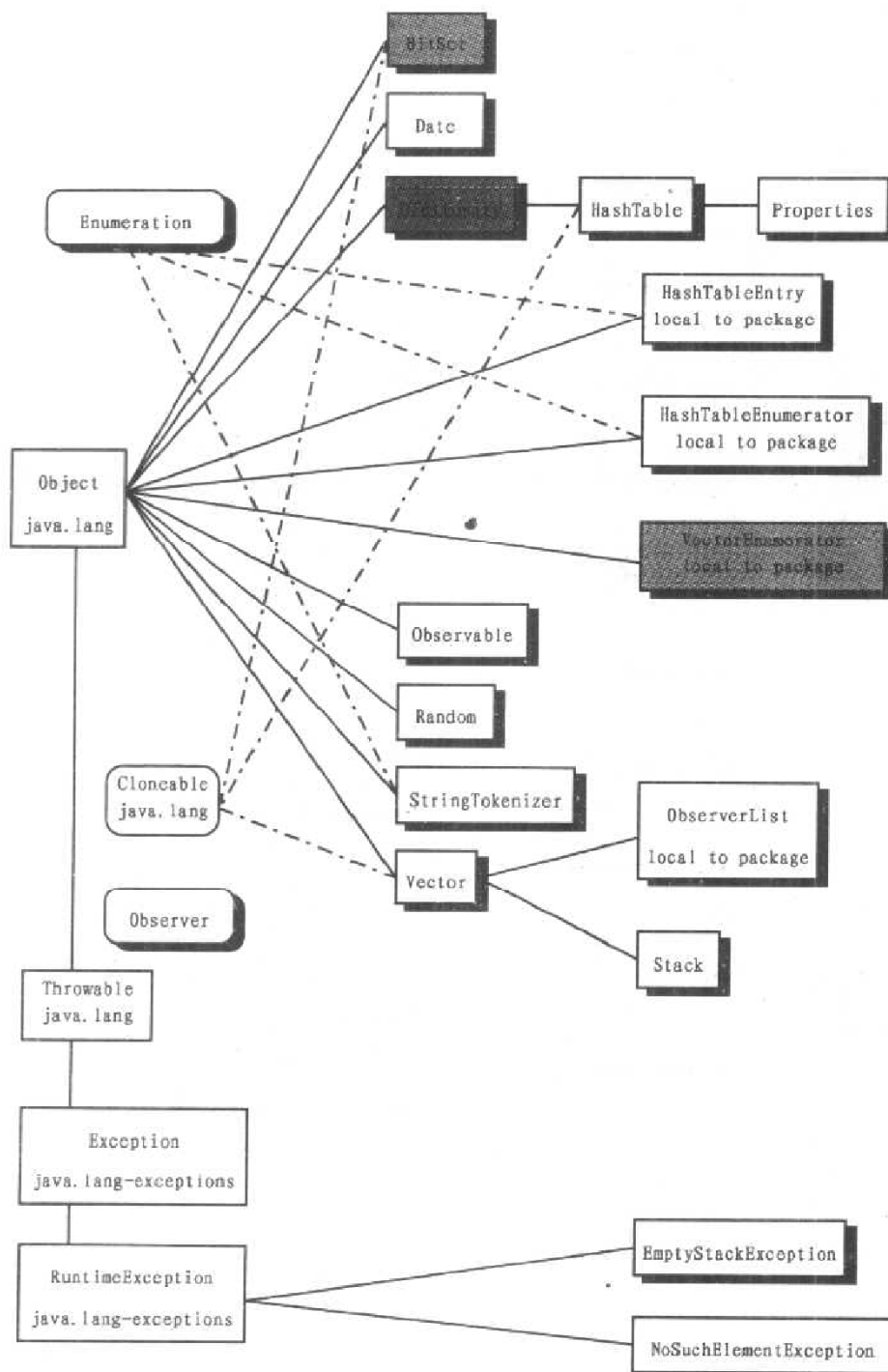


java.awt.image

java.awt.peer



java.util



附录 B HTML 简介

HTML 是一种具有广泛的链接能力的超文本语言标准。用 HTML 语言编写的文档一般在浏览器中阅读。但用普通的文本编辑器可以编写、查看其源文件。HTML 源文件的一个显著特征是其中有很多的标签。下面我们将介绍其中一些常用标签的功能。

现在有很多的辅助著作工具帮助人们编写 HTML 文档，使用起来很方便而且 HTML 的标准对编写者是透明的。从某种程度上讲，记忆 HTML 的这些标签是多余的。对于初学者而言，为了深入地理解 HTML 文档，有必要阅读以下内容。

在实际运用时，一个实用的办法是先找到一个与自己的要求相近的 HTML 文档，然后再替换其中的内容，如标题、图案等。此时需要对 HTML 文档的源文件有足够的了解。

最小的 HTML 文档

每个 HTML 文档包括许多标准的 HTML 标签（tag），如<HTML> <HEAD>等等。下面是一个最简单的 HTML 文档的例子。

```
<html>
<head>
<TITLE>A Simple HTML Example</TITLE>
</head>
<body>
<H1>HTML is Easy To Learn</H1>
<P>Welcome to the world of HTML.
This is the first paragraph. While short it is
still a paragraph!</P>
<P>And this is the second paragraph.</P>
</body>
</html>
```

HTML 的标签由左尖括号、标签名及右尖括号组成。标签一般地成对出现，如<H1>和</H1>就是一对。结束标签和开始标签很相象，只是结束标签名前多一个斜划线。有一些起始标签可能包含一些属性，如在 HTML 文档中可以在起始标签中说明图象的对齐方式（top, middle, or bottom）。

HTML 的标签是不区分大小写的。<title>、<TITLE> 和 <TiTlE>是等同的。如果某些标签不能被某一浏览器所解释，往往被忽略。

标 签

HTML

该标签告诉浏览器该文件包含了 HTML 格式的信息。当然该文件要以.html 或.htm 结尾。

HEAD

该标签标注 HTML 文档的头，其中包含有文档的标题。

TITLE

该标签说明了文档的标题。该标题将显示在浏览器窗口的上部，并出现在书签列表中，但不显示在浏览器的页面上。所以标题的内容应当简明扼要。

BODY

该标签标志 HTML 文档的主体，其中包含了文档的内容，该部分显示在浏览器的页面上。

Headings

在 HTML 主体中有六层的子标题，其中第一层是最上层。子标题用大号或/和加粗的字体显示。第一层的标题应标注为<H1>。相应地，第二层为<H2>，第三层为<H3>。在使用时应逐层使用，不能跳过其中的层次。

Paragraphs

该标签用来标注段落。在该标签标注的区域中，浏览器会忽略掉回车等，自动地把其中的内容自动回行。

在前面的例子中，有一个段落如下

```
<P>Welcome to the world of HTML.  
This is the first paragraph.  
While short it is  
still a paragraph!</P>
```

在源文件中每个句子之间都有一个回车，浏览器将忽略他们。只有当遇到新的标签的时候，才会新起一个段落。所以，我们要小心地在每个段落的前后使用该标签，否则整个正文会成为一个大段落（除了后面要讲述的用"preformatted"定义的以外）。

使用<P>和</P>作为段落的容器，我们可以进一步地定义段落的对齐方式。为此，要在源文件中插入 ALIGN=alignment 属性。如

```
<P ALIGN=CENTER>  
This is a centered paragraph. [See the formatted version below.]  
</P>  
This is a centered paragraph.
```

Lists

HTML 支持非记数 (unnumbered)、记数 (numbered) 和定义 (definition) 列表。可以使用嵌套的结构，但使用太多的嵌套可能会引起混淆。

非记数列表

1. 标记
2. 在每个列表项的前键入
3. 用 标签来结束列表

例如

```
<UL>
<LI> apples
<LI> bananas
<LI> grapefruit
</UL>
```

其在浏览器上的输出为:

```
apples
bananas
grapefruit
```

列表项可以包含有多个段落，用<P>来标注。

记数列表

其标注方法与上面的相仿，只是外层的标签为。例如

```
<OL>
<LI> oranges
<LI> peaches
<LI> grapes
</OL>
```

其输出为:

1. oranges
2. peaches
3. grapes

定义列表

由定义词（<DT>）和相应的定义（<DD>）组成。浏览器一般地会把定义部分放到新的一行中。如下例

```
<DL>
<DT> NCSA
<DD> NCSA, the National Center for Supercomputing Applications,
      is located on the campus of the University of Illinois
      at Urbana-Champaign.
<DT> Cornell Theory Center
<DD> CTC is located on the campus of Cornell University in Ithaca,
      New York.
</DL>
```

其输出如下

```
NCSA
NCSA, the National Center for Supercomputing Applications, is located on the
campus of the University of Illinois at Urbana-Champaign.
Cornell Theory Center CTC is located on the campus of Cornell University in
Ithaca, New York.
```

如果定义词很短，可以用 COMPACT 属性，如

```
<DL COMPACT>
<DT> -i
<DD>invokes NCSA Mosaic for Microsoft Windows using the
initialization file defined in the path
<DT> -k
<DD>invokes NCSA Mosaic for Microsoft Windows in kiosk mode
</DL>
```

输出为

```
-i invokes NCSA Mosaic for Microsoft Windows using the initialization file
defined in the path.
-k invokes NCSA Mosaic for Microsoft Windows in kiosk mode.
```

列表嵌套

列表可以嵌套。我们能在一个列表的表项中加入另一个列表。如

```
<UL>
```

```
<LI> A few New England states:
  <UL>
    <LI> Vermont
    <LI> New Hampshire
    <LI> Maine
  </UL>
<LI> Two Midwestern states:
  <UL>
    <LI> Michigan
    <LI> Indiana
  </UL>
</UL>
```

其输出效果为

```
A few New England states:
  Vermont
  New Hampshire
  Maine
Two Midwestern states:
  Michigan
  Indiana
```

Address

`<ADDRESS>`标签用来说明文档的作者和联系方式，如电子邮件地址等。它一般是文档中的最后一项。如

```
<ADDRESS>
A Beginner's Guide to HTML / NCSA / pubs@ncsa.uiuc.edu / revised April 96
</ADDRESS>
```

强制转行

`
` 标签强制一行转行。例如

```
National Center for Supercomputing Applications<BR>
605 East Springfield Avenue<BR>
Champaign, Illinois 61820-5518<BR>
```

输出为

National Center for Supercomputing Applications
605 East Springfield Avenue
Champaign, Illinois 61820-5518

水平分割线

<HR> 标签产生一个宽度可调的分割线。例如，
<HR SIZE=4 WIDTH="50%">

链接（ Linking ）

HTML 的一个威力来自于它把自身文档中的文本或图象链接到其他文档的能力。浏览器将把有链接的文本或图象用特殊颜色或加下划线标志。HTML 的链接是用<A>标签。例如，
Maine
它将把 Maine 链接到在同一个目录下的文档 MaineStats.html 上。

相对地址和绝对地址

在定义 URL 地址时，建议尽量使用相对地址，不要轻易用绝对地址（即指明服务器地址，例如：<http://www.ibm.com/pub/demo/demo.htm>）。因为若采用绝对地址，则当要把此 Home Page 移植到别的服务器上时，所有 HyperLink 的地址都要作修改，十分麻烦而且容易有遗漏。若采用相对地址，由于是用目录结构的相对关系来表示，因此只需将有关的目录结构照搬即可完成移植。

- 以“..”开头表示从此源文件所在子目录向上退一级。例如在 pub 目录的 demo 子目录下文件 demo.htm 中有这样一个 URL 地址：[href="../home/home.htm"](http://home/home.htm)，则说明 home 子目录和 demo 子目录是同在 pub 目录下的同级子目录。
- “../..”表示从此源文件所在子目录向上退两级，其它依此类推。
- 以“/”开头表示从服务器的缺省目录下算起。例如服务器的缺省目录为 doc 目录，则[href="/help/help.htm"](http://help/help.htm)表示 help 目录是 doc 目录的子目录。
- 若在 pub 目录的 demo 子目录下文件 demo.htm 中有这样一个 URL 地址：[href="staff/staff.htm"](http://staff/staff.htm)，则表明 staff 目录是 demo 目录的子目录。

锚点（ Anchors ）

锚点可以让用户标注同一文档或者是不同文档的特定点。

<H2>Acadia National Park</H2>

标注了一个有名锚点 ANP，下面的链接将直接把 Acadia National Park 显示在屏幕上。

Acadia National Park.

如果在同一个文件中，可以省略掉文件名，如

```
...More information about <A HREF="#ANP">Acadia National Park</a>  
is available elsewhere in this document.
```

Mailto

它可以是发送电子邮件给某个人变得很容易，其格式为：

```
<A HREF="mailto:emailinfo@host">Name</a>
```

例如

```
<A HREF="mailto:zhjin@vnet.ibm.com">Jin Zunhe's mailbox</a>
```

内插图象 (Inline Images)

大多数的浏览器能够显示内插的图象，如 X Bitmap (XBM)，GIF，JPEG 格式。图象的载入将减慢文档的显示。因而应当仔细选择图象，并控制数量。下面是嵌入图象的一般格式：

```
<IMG SRC=ImageName>
```

其中 ImageName 代表了图象的 URL。图象文件名必须包括文件的后缀，如.gif、.xbm、.jpg、.jpeg、.png。

在 HTML 文件中要定义图象的大小，如以点为单位的高度和宽度。如

```
<IMG SRC=SelfPortrait.gif HEIGHT=100 WIDTH=65>
```

需要注意的是当这里说明的高度和宽度与原有的图象的大小不符时，浏览器就要伸缩图象来符合图象框，这往往使图象变形。所以，最好有一个确定的值以后再写入。

对齐图象

显示图象时，在对齐的方式上可以有一定的柔性，如左对齐、右对齐或居中。

```
<IMG SRC = "BarHotlist.gif" ALIGN=CENTER>
```

ALT 属性在非图形化界面中很有用，它说明该图象的内容。例如，

```
<IMG SRC="UpArrow.gif" ALT="Up">
```

UpArrow.gif 是一个上指的箭头。在图形化的浏览器中只要把 image-loading 选项选中就可以看到一个上指的箭头图象。如果在 VT100 浏览器或把 image-loading 选项关掉，词 Up 就显示在同一个位置。

背景图象

新一代的浏览器可以装载图象，并把它当成背景使用。当使用背景的时候，要注意背景不能影响前台的字符的显示。背景图象可以是纹理图案或是特征标志。图象本身不用太大，浏览器可以反复地填充浏览器窗口。下面的例子，在文档的主体部分显示一个背景图

案。<BODY BACKGROUND="filename.gif">

背景颜色

浏览器缺省地在灰色的背景上显示文本。但你可以改变两者的颜色。下面的例子把窗口的底色设置为黑色，文本为白色，有链接的文本的颜色为银色。

```
<BODY BGCOLOR="#000000" TEXT="#FFFFFF" LINK="#9690CC">
```

代表颜色的六个数字分别代表了颜色的 RGB 值。

注 释

HTML 文档中的注释相计算机程序中的注释一样，只是给编写者一个备忘录，对最后的效果没有什么影响。访问者如果以源文件方式打开文件，那么可以看到注释。注释的格式是：

```
<!-- your comments here -->
```

其中的惊叹号和连字符都必不可少。

还有一些高级功能我们没有介绍，如 表格 (Table)、图象映射 (Map)、分栏 (Frame) 和动画功能等。请再后续的附录中查找相关的 HTML 标准说明书。

附录 C 相关信息的 Internet 地址

SUN 公司的 Java 资源

- <http://java.sun.com/> 及 <http://www.javasoft.com/>
提供了 Java 最详细的资料, 包括入门教材、常问问题解答以及 Java 的应用和开发者信息。

著名的 Java Web 节点

- <http://www.gamelan.com/>
收集了相当好的 Java 信息和大量的 Java Applet 示范程序, 并且经常更新。
- <http://www.iworld.com/InternetShopper/1Java-products.html>
描述了与 Java 相关的各种商用软件。
- <http://rendezvous.com/java/hierarchy/index.html>
提供了 Java 类层次关系图, 由 Charles L. Perkins 开发。
- <http://www.december.com/works/java/bib.html>
Java 在线 (Online) 参考手册, 列出了 Java 和相关技术的文章及出版物。

各大公司发布的 Java 信息

- <http://www.ibm.com/News/javapr.html>
IBM 公司购买了 Java 技术在 Internet 产品中的使用权。
- <http://www.lotus.com/corpcomm/3406.htm>
Lotus 公司已将 Notes 和 WWW 集成起来, 在 Notes Server 中提供对 HTTP、HTML 和 Java 技术的支持。
- <http://www.microsoft.com/internet/press.html>
Microsoft 公司欲在 Internet 上大展宏图, 并将 Java 作为一项核心支持技术。
- <http://home.netscape.com/newsref/pr/newsrelease67.html>
Netscape 公司和 SUN 公司宣布 Javascript, 一种用于企业网和 Internet 的开放、跨平台的对象副本 (Script) 语言。

VRML 资源

- <http://vag.vrml.org/>

提供了虚拟现实建模语言的详尽信息，包括最新发展、语言规范以及应用情况。

- <http://www.sgi.com/> 及 <http://vrml.sgi.com/>

SGI 公司提供对虚拟现实的全面支持，它实现了虚拟现实浏览器、VRML 制作工具以及用于虚拟现实的 Java API 等。

- <http://www.dnx.com/>

Dimension X 公司开发了 Liquid Reality，一个基于 Java 的 VRML 开发工具。

- <http://www.intervista.com/vrml/index.shtml>

Internet 上一个比较全面的 VRML 节点。