

# 译者序

《Java核心技术》的第8版又推出了，它已经在广大Java程序员和爱好者们当中产生了巨大的影响力。该书覆盖面广，几乎囊括了Java 2标准版的所有方面。其以接近实战的实例来展开内容的书写方式更是容易让读者理解和接受Java的精髓。

Java已经受到越来越多的程序员的青睐，但是Java语言包罗万象，而且其自身发展的速度更是惊人，我们在Sun的网站上几乎每个月都会看到有新的基于Java的规范出台。JDK 6.0使得Java又呈现出了新的面貌，其新添加的特性更好地能够应对Java在构建企业应用时所需要面对的挑战。因此，《Java核心技术》第8版在第7版的基础上，对JDK 6.0中的新特性进行了重点介绍，对原有部分章节进行了更新和调整，删除了一些过时的内容，并新增加了一些章节，而且对第7版的很多示例程序进行了调整，以此来使得程序员们能够更加透彻地理解和熟练地掌握这些新特性。

卷Ⅱ面向的是已经熟读并掌握了卷Ⅰ内容的读者，或者是已经对Java语言的基本特性相当熟悉的读者。卷Ⅱ的内容包含了流与文件、XML、网络、数据库编程、国际化、高级Swing、高级AWT、JavaBeans、安全、分布式对象、脚本编写、编译与注解处理，以及本地方法等内容，把读者引入了Java世界的更深处。

我们在翻译本书的过程中力求忠于原著。对于本书中出现的大量的专业术语尽量遵循标准的译法，并在有可能引起歧义之处注上了英文原文，以方便读者的对照理解。

全书的翻译由陈昊鹏、王浩、姚建平和龚斌合作完成，楼钢、李伟、郭嘉和方小丽对全书的翻译也做了大量的工作。由于译者水平有限，书中出现错误与不妥之处在所难免，恳请读者批评指正。

# 前　　言

## 致读者

您手中的这本书是按照Java SE 6完全更新后的《Java核心技术卷Ⅱ：高级特性（原书第8版）》。卷Ⅰ主要介绍了Java语言的一些关键特性；而本卷主要介绍编程人员进行专业软件开发时需要了解的高级主题。因此，与卷Ⅰ及之前的版本一样，我们仍将本书定位于为那些将Java技术运用于实际项目的编程人员提供帮助。

请注意：如果你是一个经验丰富的开发人员，能够灵活运用像内部类和泛型这样的高级语言特性，那么你就不需要阅读完卷Ⅰ再学习本卷。不过，本卷会根据适当情况去参考引用卷Ⅰ的有关内容，（我们希望读者会购买或者已经购买了卷Ⅰ，）当然，读者也可以在任何一本综合介绍Java平台的书中找到所需的背景知识。

最后要说明的一点是，编写任何一本书籍都难免会有一些错误或不准确的地方。我们非常乐意听到读者在本书中找到这方面的内容。当然，我们更希望对这些问题的报告只听到一次。为此，我们创建了一个FAQ、bug修正以及应急方案的网站<http://www.horstmann.com/corejava>。你可以在bug报告网页（该网页的目的是鼓励读者阅读以前的报告）的末尾处添加bug报告来发布bug和问题、给出建议，以便改进本书将来的版本。

## 内容提要

本书中的章节大部分是相互独立的。你可以研究最感兴趣的任何主题，并可以按照任意顺序阅读这些章节。

**第1章**的主题是输入输出处理。在Java中，所有I/O都是通过所谓“流”来处理的。流使你可以按照统一的方式来处理与各种数据源之间的通信，例如文件、网络连接或内存块。我们对各种读入器和写出器类进行了详细的讨论，它们使得对Unicode的处理变得很容易。我们还向你展示了：在使用对象序列化机制从而使保存和加载对象变得容易而方便时，其背后的原理是什么。最后，我们讨论了支持高效文件操作的“新I/O”类（它们曾作为最新内容添加到Java SE 1.4中）和正则表达式类库。

**第2章**介绍XML，介绍怎样解析XML文件，怎样生成XML以及怎样使用XSL转换。在一个实用示例中，我们将展示怎样在XML中指定Swing格式的布局。我们对该章进行了更新修正，将XPath API纳入其中，它使得“在XML的干草堆中寻找绣花针”变得更加容易。

**第3章**介绍网络API。Java使复杂的网络编程工作变得很容易实现。我们将介绍怎样创建连接到服务器上的网络连接，怎样实现你自己的服务器，以及怎样创建HTTP连接。

**第4章**介绍数据库编程，重点讲解JDBC，即Java数据库连接API，这是用于将Java程序与关系数据库进行连接的API。我们将介绍怎样通过使用JDBC API的核心子集，编写能够处理实际

的数据库日常操作事务的实用程序。(如果要完整介绍JDBC API的功能，可能需要编写一本像本书一样厚的书才行。)最后我们简要介绍了层次数据库，探讨了一下JNDI (Java命名及目录接口)以及LDAP (轻量级目录访问协议)。

第5章讨论了一个我们认为重要性将会不断提升的特性——国际化。Java编程语言是几种一开始就被设计为可以处理Unicode的语言之一，不过Java平台的国际化支持则走得更加深远。因此，你可以对Java应用程序进行国际化，使得它们不仅可以跨平台，而且还可以跨越国界。例如，我们会展示怎样编写一个退休金计算器的Applet，对它可以根据本地浏览器的情况使用英语、德语或者汉语进行浏览。

第6章涵盖了没有纳入卷 I 的所有Swing知识，尤其是重要但很复杂的树型构件和表格构件。随后我们介绍了编辑面板的基本用法、“多文档”界面的Java实现、在多线程程序中用到的进度指示器，以及诸如闪屏和支持系统托盘这样的“桌面集成特性”。我们仍着重介绍在实际编程中可能遇到的最为有用的构件，因为对Swing类库进行百科全书般的介绍可能会占据好几卷书的篇幅，并且只有专业编程人员才感兴趣。

第7章介绍Java 2D API，你可以用它来创建实际的图形和特殊的效果。该章还介绍了抽象窗口操作工具包 (AWT)的一些高级特性，这部分似乎应该在卷 I 中做专门介绍。虽然如此，这些技术还是应该成为每一个编程人员工具包的一部分。这些特性包括打印和用于剪切粘贴及拖放的API。

第8章介绍了用于Java平台的构件API——JavaBean。你将会看到怎样编写自己的Bean，以及其他编程人员怎样在集成构建环境中对它们进行操作。最后我们展示怎样使用JavaBean的持久性，以某种与适用于长期存储的对象序列化不同的格式来存储自己的数据。

第9章继续介绍Java安全模式。Java平台一开始就是基于安全而设计的，该章会带你深入内部，查看这种设计是怎样实现的。我们将展示怎样编写用于特殊目的的应用的类加载器以及安全管理器。然后介绍允许使用消息、代码签名、授权以及认证和加密等重要特性的安全API。最后，我们用一个使用AES和RSA加密算法的示例进行了总结。

第10章介绍分布式对象。我们详细介绍了RMI (远程方法调用)。这个API可以让你运行分布在多台机器上的Java对象。然后简要讨论了Web Service，并给出了一个实现了Java程序和Amazon Web Service之间进行通信的示例。

第11章讨论了三种处理代码的技术。脚本机制和编译器API是在Java SE 6中引入的，它们允许程序去调用使用诸如JavaScript或Groovy之类的脚本语言编写的代码，并且允许程序去编译Java代码。可以使用注释向Java程序中添加任意信息(有时称为元数据)。我们将展示注释处理器怎样在源码级别或者在类文件级别上收集这些注释，以及怎样运用这些注释来影响运行时的类行为。注释只有在工具的支持下才有用，因此，我们希望我们的讨论能够帮助你根据需要选择有用的注释处理工具。

第12章介绍本地化方法，它可以让你调用为微软Windows API这样的特殊机制而编写的各种调用方法。很显然，这种特性具有争议性：使用本地化方法，那么Java平台的跨平台本质将会随之消失。虽然如此，每个为特定平台编写Java应用程序的严谨的编程人员都需要了解这些技术。有时，当你与不支持Java平台的设备或服务进行交互时，为了你的目标平台，你可能需

要求助于操作系统API。我们将通过展示如何从某个Java程序访问Windows注册表API来阐明这一点。

所有章节都按照最新版本的Java进行了修订，过时的材料都删除了，Java SE 6的新API也都详细地进行了讨论。

## 约定

我们使用等宽字体表示计算机代码，这种格式在众多的计算机书籍中极为常见。各种图标的含义如下：

-  **注意：**需要引起注意的地方。
-  **提示：**有用的提示。
-  **警告：**关于缺陷或危险情况的警告信息。
-  **C++注意：**本书中有一些C++注释，用于解释Java程序设计语言和C++语言之间的不同。如果你对这部分不感兴趣，可以跳过。

## 应用编程接口

Java平台配备有大量的编程类库或者应用编程接口（API）。当第一次使用某个API时，我们添加了一个简短的描述，并用一个API图标进行标识。这些描述可能有点不太规范，但是比起那些正式的在线API文档来说要更具指导性一点。

其源代码包含在与本书相关的代码中的程序都被作为示例程序而将其代码列举了出来。例如，

### 程序清单11-1 ScriptTest.java

可以从网站<http://horstmann.com/corejava><sup>⊕</sup>下载相关代码。

## 致谢

写书总是需要付出极大的努力，而重写也并不像看上去那么容易，特别是在Java技术方面，要跟上其飞快的发展速率，更是如此。一本书的面世需要众多有奉献精神的人共同努力，我非常荣幸地在此向整个《Java核心技术》团队致谢。

Prentice Hall出版社和Sun Microsystems出版社的许多人都提供了颇有价值的帮助，但是他们甘愿居于幕后。我希望他们都能够知道我是多么感谢他们付出的努力。与以往一样，我要热切地感谢我的编辑，Prentice Hall出版社的Greg Doench，他对本书从编写到出版进行全程掌舵，并使我可以十分幸福地根本意识不到幕后那些人的存在。我的感谢还要送给本书以前版本的合著者Gary Cornell，他后来转向其他具有挑战性的领域了。

---

<sup>⊕</sup> 也可登录华章网站 (<http://www.hzbook.com>) 下载相关代码。——编辑注

我非常感谢找到了很多令人尴尬的错误并提出了许多颇具创见性的建议的早先版本的许多读者们。我特别要感谢十分出色的评审团队，他们用令人惊异的眼睛仔细浏览了所有原稿，并将我从许多令人尴尬的错误中拯救了出来。

这一版及以前版本是由以下人员评审的：Chuck Allison（特约编辑，《C/C++ Users Journal》）、Alec Beaton（PointBase, Inc.）、Cliff Berg（iSavvix Corporation）、Joshua Bloch（Sun Microsystems）、David Brown、Corky Cartwright、Frank Cohen（PushToTest）、Chris Crane（devXsolution）、Dr. Nicholas J. De Lillo（曼哈顿学院）、Rakesh Dhoopar（Oracle）、Robert Evans（资深教师，约翰霍·普金斯大学应用物理实验室）、David Geary（Sabreware）、Brian Goetz（首席顾问，Quiotix Corp.）、Angela Gordon（Sun Microsystems）、Dan Gordon（Sun Microsystems）、Rob Gordon、John Gray（Hartford大学）、Cameron Gregory（olabs.com）、Marty Hall（约翰斯·霍普金斯大学应用物理实验室）、Vincent Hardy（Sun Microsystems）、Dan Harkey（圣何塞州立大学）、William Higgins（IBM）、Vladimir Ivanovic（PointBase）、Jerry Jackson（ChannelPoint Software）、Tim Kimmet（Preview Systems）、Chris Laffra、Charlie Lai（Sun Microsystems）、Angelika Langer、Doug Langston、Hang Lau（McGill大学）、Mark Lawrence、Doug Lea（SUNY Oswego）、Gregory Longshore、Bob Lynch（Lynch Associates）、Philip Milne（顾问）、Mark Morrissey（俄勒冈研究院）、Mahesh Neelakanta（佛罗里达大西洋大学）、Hao Pham、Paul Philion、Blake Ragsdell、Ylber Ramadani（Ryerson大学）、Stuart Reges（亚利桑那大学）、Rich Rosen（Interactive Data Corporation）、Peter Sanders（ESSI大学, Nice, France）、Dr. Paul Sanghera（圣何塞州立大学和布鲁克学院）、Paul Sevinc（Teamup AG）、Devang Shah（Sun Microsystems）、Richard Slywczak（NASA/Glenn研究中心）、Bradley A. Smith、Steven Stelting（Sun Microsystems）、Christopher Taylor、Luke Taylor（Valtech）、George Thiruvathukal、Kim Topley（《Core JFC》的作者）、Janet Traub、Paul Tyma（顾问）Peter van der Linden（Sun Microsystems）和Burt Walsh。

Cay Horstmann

旧金山2008

# 目 录

|                           |            |
|---------------------------|------------|
| 译者序                       |            |
| 前言                        |            |
| <b>第1章 流与文件</b>           | <b>1</b>   |
| 1.1 流                     | 1          |
| 1.1.1 读写字节                | 1          |
| 1.1.2 完整的流家族              | 3          |
| 1.1.3 组合流过滤器              | 7          |
| 1.2 文本输入与输出               | 10         |
| 1.2.1 如何写出文本输出            | 10         |
| 1.2.2 如何读入文本输入            | 12         |
| 1.2.3 以文本格式存储对象           | 13         |
| 1.2.4 字符集                 | 17         |
| 1.3 读写二进制数据               | 20         |
| 1.4 ZIP文档                 | 28         |
| 1.5 对象流与序列化               | 35         |
| 1.5.1 理解对象序列化的文件格式        | 40         |
| 1.5.2 修改默认的序列化机制          | 45         |
| 1.5.3 序列化单例和类型安全的枚举       | 47         |
| 1.5.4 版本管理                | 48         |
| 1.5.5 为克隆使用序列化            | 50         |
| 1.6 文件管理                  | 52         |
| 1.7 新I/O                  | 57         |
| 1.7.1 内存映射文件              | 58         |
| 1.7.2 缓冲区数据结构             | 63         |
| 1.7.3 文件加锁机制              | 65         |
| 1.8 正则表达式                 | 67         |
| <b>第2章 XML</b>            | <b>76</b>  |
| 2.1 XML概述                 | 76         |
| 2.2 解析XML文档               | 80         |
| 2.3 验证XML文档               | 91         |
| 2.3.1 文档类型定义              | 92         |
| 2.3.2 XML Schema          | 98         |
| 2.3.3 实用示例                | 100        |
| 2.4 使用XPath来定位信息          | 113        |
| 2.5 使用命名空间                | 118        |
| 2.6 流机制解析器                | 120        |
| 2.6.1 使用SAX解析器            | 121        |
| 2.6.2 使用StAX解析器           | 125        |
| 2.7 生成XML文档               | 128        |
| 2.8 XSL转换                 | 138        |
| <b>第3章 网络</b>             | <b>148</b> |
| 3.1 连接到服务器                | 148        |
| 3.1.1 套接字超时               | 152        |
| 3.1.2 因特网地址               | 153        |
| 3.2 实现服务器                 | 154        |
| 3.2.1 为多个客户端服务            | 157        |
| 3.2.2 半关闭                 | 160        |
| 3.3 可中断套接字                | 161        |
| 3.4 发送E-mail              | 167        |
| 3.5 建立URL连接               | 172        |
| 3.5.1 URL和URI             | 172        |
| 3.5.2 使用URLConnection获取信息 | 173        |
| 3.5.3 提交表单数据              | 182        |
| <b>第4章 数据库编程</b>          | <b>190</b> |
| 4.1 JDBC的设计               | 190        |
| 4.1.1 JDBC驱动程序类型          | 191        |
| 4.1.2 JDBC的典型用法           | 192        |
| 4.2 结构化查询语言               | 193        |
| 4.3 JDBC配置                | 198        |
| 4.3.1 数据库URL              | 198        |
| 4.3.2 驱动程序JAR文件           | 199        |

|                           |     |                                |     |
|---------------------------|-----|--------------------------------|-----|
| 4.3.3 启动数据库 .....         | 199 | 5.7.2 属性文件 .....               | 289 |
| 4.3.4 注册驱动器类 .....        | 200 | 5.7.3 包类 .....                 | 290 |
| 4.3.5 连接到数据库 .....        | 200 | 5.8 一个完整的例子 .....              | 291 |
| 第4章 执行SQL语句 .....         | 203 | 第6章 高级Swing .....              | 305 |
| 4.4.1 管理连接、语句和结果集 .....   | 205 | 6.1 列表 .....                   | 305 |
| 4.4.2 分析SQL异常 .....       | 206 | 6.1.1 JList构件 .....            | 305 |
| 4.4.3 组装数据库 .....         | 208 | 6.1.2 列表模式 .....               | 310 |
| 4.5 执行查询操作 .....          | 211 | 6.1.3 插入和移除值 .....             | 315 |
| 4.5.1 预备语句 .....          | 212 | 6.1.4 值的绘制 .....               | 316 |
| 4.5.2 读写LOB .....         | 219 | 6.2 表格 .....                   | 321 |
| 4.5.3 SQL转义 .....         | 220 | 6.2.1 简单表格 .....               | 321 |
| 4.5.4 多结果集 .....          | 222 | 6.2.2 表格模型 .....               | 324 |
| 4.5.5 获取自动生成键 .....       | 222 | 6.2.3 对行和列的操作 .....            | 328 |
| 4.6 可滚动和可更新的结果集 .....     | 223 | 6.2.4 单元格的绘制和编辑 .....          | 340 |
| 4.6.1 可滚动的结果集 .....       | 223 | 6.3 树 .....                    | 351 |
| 4.6.2 可更新的结果集 .....       | 225 | 6.3.1 简单的树 .....               | 352 |
| 4.7 行集 .....              | 228 | 6.3.2 节点枚举 .....               | 365 |
| 4.8 元数据 .....             | 231 | 6.3.3 绘制节点 .....               | 367 |
| 4.9 事务 .....              | 240 | 6.3.4 监听树事件 .....              | 369 |
| 4.9.1 保存点 .....           | 241 | 6.3.5 定制树模型 .....              | 375 |
| 4.9.2 批量更新 .....          | 241 | 6.4 文本构件 .....                 | 383 |
| 4.9.3 高级SQL类型 .....       | 243 | 6.4.1 文本构件中的修改跟踪 .....         | 384 |
| 4.10 Web与企业应用中的连接管理 ..... | 244 | 6.4.2 格式化的输入框 .....            | 387 |
| 4.11 LDAP介绍 .....         | 245 | 6.4.3 JSpinner构件 .....         | 401 |
| 4.11.1 配置LDAP服务器 .....    | 247 | 6.4.4 用JEditorPane显示HTML ..... | 408 |
| 4.11.2 访问LDAP目录信息 .....   | 249 | 6.5 进度指示器 .....                | 414 |
| 第5章 国际化 .....             | 260 | 6.5.1 进度条 .....                | 414 |
| 5.1 Locales .....         | 260 | 6.5.2 进度监视器 .....              | 417 |
| 5.2 数字格式 .....            | 265 | 6.5.3 监视输入流的进度 .....           | 421 |
| 5.3 日期和时间 .....           | 271 | 6.6 构件组织器 .....                | 425 |
| 5.4 排序 .....              | 277 | 6.6.1 分割面板 .....               | 425 |
| 5.4.1 排序强度 .....          | 278 | 6.6.2 选项卡面板 .....              | 429 |
| 5.4.2 分解 .....            | 279 | 6.6.3 桌面面板和内部框架 .....          | 435 |
| 5.5 消息格式化 .....           | 284 | 6.6.4 级联与平铺 .....              | 437 |
| 5.6 文本文件和字符集 .....        | 287 | 6.6.5 否决属性设置 .....             | 440 |
| 5.7 资源包 .....             | 288 | 第7章 高级AWT .....                | 451 |
| 5.7.1 定位资源包 .....         | 288 | 7.1 绘图操作流程 .....               | 451 |

|                             |     |                          |     |
|-----------------------------|-----|--------------------------|-----|
| 7.2 形状                      | 453 | 第8章 JavaBean构件           | 590 |
| 7.3 区域                      | 467 | 8.1 为何使用Bean             | 590 |
| 7.4 笔划                      | 468 | 8.2 编写Bean的过程            | 591 |
| 7.5 着色                      | 475 | 8.3 使用Bean构造应用程序         | 594 |
| 7.6 坐标变换                    | 476 | 8.3.1 将Bean打包成JAR文件      | 594 |
| 7.7 剪切                      | 481 | 8.3.2 在开发环境中组合Bean       | 595 |
| 7.8 透明与组合                   | 483 | 8.4 Bean属性与事件的命名模式       | 600 |
| 7.9 绘图提示                    | 490 | 8.5 Bean属性的类型            | 602 |
| 7.10 图像的读取器和写入器             | 496 | 8.5.1 简单属性               | 603 |
| 7.10.1 获得图像文件类型的读取器<br>和写入器 | 497 | 8.5.2 索引属性               | 603 |
| 7.10.2 读取和写入带有多个图像的文件       | 498 | 8.5.3 绑定属性               | 603 |
| 7.11 图像处理                   | 506 | 8.5.4 约束属性               | 605 |
| 7.11.1 构建光栅图像               | 506 | 8.6 BeanInfo类            | 611 |
| 7.11.2 图像过滤                 | 512 | 8.7 属性编辑器                | 614 |
| 7.12 打印                     | 519 | 8.8 定制器                  | 623 |
| 7.12.1 图形打印                 | 520 | 8.9 JavaBean持久化          | 631 |
| 7.12.2 打印多页文件               | 528 | 8.9.1 JavaBean持久化可用于任何数据 | 634 |
| 7.12.3 打印预览                 | 529 | 8.9.2 一个JavaBean持久化的完整示例 | 640 |
| 7.12.4 打印服务程序               | 537 | 第9章 安全                   | 650 |
| 7.12.5 流打印服务程序              | 540 | 9.1 类加载器                 | 650 |
| 7.12.6 打印属性                 | 541 | 9.1.1 类加载器的层次结构          | 651 |
| 7.13 剪贴板                    | 547 | 9.1.2 将类加载器作为命名空间        | 653 |
| 7.13.1 数据传递的类和接口            | 548 | 9.1.3 编写你自己的类加载器         | 653 |
| 7.13.2 传递文本                 | 548 | 9.2 字节码校验                | 659 |
| 7.13.3 可传递的接口和数据风格          | 552 | 9.3 安全管理器与访问权限           | 663 |
| 7.13.4 构建一个可传递的图像           | 554 | 9.3.1 Java平台安全性          | 664 |
| 7.13.5 通过系统剪贴板传递Java对象      | 558 | 9.3.2 安全策略文件             | 667 |
| 7.13.6 使用本地剪贴板来传递对象引用       | 562 | 9.3.3 定制权限               | 672 |
| 7.14 拖放操作                   | 562 | 9.3.4 实现权限类              | 674 |
| 7.14.1 Swing对数据传递的支持        | 563 | 9.4 用户认证                 | 679 |
| 7.14.2 拖曳源                  | 567 | 9.5 数字签名                 | 692 |
| 7.14.3 放置目标                 | 569 | 9.5.1 消息摘要               | 693 |
| 7.15 平台集成                   | 576 | 9.5.2 消息签名               | 698 |
| 7.15.1 闪屏                   | 576 | 9.5.3 X.509证书格式          | 700 |
| 7.15.2 启动桌面应用程序             | 580 | 9.5.4 校验签名               | 701 |
| 7.15.3 系统托盘                 | 585 | 9.5.5 认证问题               | 703 |
|                             |     | 9.5.6 证书签名               | 705 |

|                                     |     |                          |     |
|-------------------------------------|-----|--------------------------|-----|
| 9.5.7 证书请求                          | 706 | 11.1.5 编译脚本              | 765 |
| 9.6 代码签名                            | 707 | 11.1.6 一个示例：用脚本处理GUI事件   | 766 |
| 9.6.1 JAR文件签名                       | 707 | 11.2 编译器API              | 770 |
| 9.6.2 软件开发者证书                       | 709 | 11.2.1 编译便捷之法            | 770 |
| 9.7 加密                              | 713 | 11.2.2 使用编译工具            | 771 |
| 9.7.1 对称密码                          | 713 | 11.2.3 一个示例：动态Java代码生成   | 775 |
| 9.7.2 密钥生成                          | 714 | 11.3 使用注解                | 779 |
| 9.7.3 密码流                           | 719 | 11.4 注解语法                | 785 |
| 9.7.4 公共密钥密码                        | 720 | 11.5 标准注解                | 788 |
| 第10章 分布式对象                          | 724 | 11.5.1 用于编译的注解           | 789 |
| 10.1 客户与服务器的角色                      | 724 | 11.5.2 用于管理资源的注解         | 790 |
| 10.2 远程方法调用                         | 726 | 11.5.3 元注解               | 790 |
| 10.3 配置远程方法调用                       | 728 | 11.6 源码级注解处理             | 792 |
| 10.3.1 接口与实现                        | 728 | 11.7 字节码工程               | 798 |
| 10.3.2 RMI注册表                       | 729 | 第12章 本地方法                | 805 |
| 10.3.3 部署程序                         | 733 | 12.1 从Java程序中调用C函数       | 805 |
| 10.3.4 记录RMI活动                      | 735 | 12.2 数值参数与返回值            | 810 |
| 10.4 远程方法中的参数和返回值                   | 736 | 12.3 字符串参数               | 812 |
| 10.4.1 传递远程对象                       | 736 | 12.4 访问域                 | 817 |
| 10.4.2 传递非远程对象                      | 736 | 12.4.1 访问实例域             | 817 |
| 10.4.3 动态类加载                        | 739 | 12.4.2 访问静态域             | 820 |
| 10.4.4 具有多重接口的远程引用                  | 743 | 12.5 编码签名                | 821 |
| 10.4.5 远程对象与equals、hashCode和clone方法 | 743 | 12.6 调用Java方法            | 822 |
| 10.5 远程对象激活                         | 744 | 12.6.1 实例方法              | 822 |
| 10.6 Web Services与JAX-WS            | 749 | 12.6.2 静态方法              | 823 |
| 10.6.1 使用JAX-WS                     | 749 | 12.6.3 构造器               | 824 |
| 10.6.2 Web服务的客户端                    | 752 | 12.6.4 替代方法调用            | 824 |
| 10.6.3 Amazon的E-Commerce服务          | 754 | 12.7 访问数组元素              | 828 |
| 第11章 脚本、编译与注解处理                     | 760 | 12.8 错误处理                | 831 |
| 11.1 Java平台的脚本                      | 760 | 12.9 使用调用API             | 835 |
| 11.1.1 获取脚本引擎                       | 760 | 12.10 完整的示例：访问Windows注册表 | 839 |
| 11.1.2 脚本赋值与绑定                      | 761 | 12.10.1 Windows注册表概述     | 840 |
| 11.1.3 重定向输入和输出                     | 763 | 12.10.2 访问注册表的Java平台接口   | 841 |
| 11.1.4 调用脚本的函数和方法                   | 764 | 12.10.3 以本地方法方式实现注册表     |     |
|                                     |     | 访问函数                     | 841 |

# 第1章 流与文件

- ▲ 流
- ▲ 文本输入与输出
- ▲ 读入和写出二进制数据
- ▲ ZIP文档
- ▲ 对象流与序列化
- ▲ 文件管理
- ▲ 新I/O
- ▲ 正则表达式

本章将介绍Java用于输入和输出的各种应用编程接口（Application Programming Interface，API）。你将要学习如何访问文件与目录，以及如何以二进制格式和文本格式来读写数据。本章还要向你展示对象序列化机制，它可以使存储对象像存储文本和数字数据一样容易。然后，我们将介绍在Java SE 1.4中引入的“新I/O”包java.nio所带来的种种改进。最后，本章将讨论正则表达式，尽管这部分内容实际上与流和文件并不相关，但是我们确实也找不到更合适的地方来处理这个话题。很明显，Java设计团队在这个问题的处理上和我们一样，因为正则表达式API的规格说明隶属于阐述Java SE 1.4的“新I/O”特性的规格说明。

## 1.1 流

在Java API中，可以从其中读入一个字节序列的对象称做输入流，而可以向其中写入一个字节序列的对象称做输出流。这些字节序列的来源地和目的地可以是文件，而且通常都是文件，但是也可以是网络连接，甚至是内存块。抽象类InputStream和OutputStream构成了有层次结构的输入/输出（I/O）类的基础。

因为面向字节的流不便于处理以Unicode形式（回忆一下，Unicode中每个字符都使用了多个字节来表示）存储的信息，所以从抽象类Reader和Writer中继承出来的专门用于处理Unicode字符的类构成了一个单独的层次结构。这些类拥有的读入和写出操作都是基于两字节的Unicode码元的，而不是基于单字节的字符。

### 1.1.1 读写字节

InputStream类有一个抽象方法：

```
abstract int read()
```

这个方法将读入一个字节，并返回读入的字节，或者在遇到输入源结尾时返回-1。在设计具体输入流类时，必须覆盖这个方法以提供适用的功能，例如，在FileInputStream类中，这个方法将从某个文件中读入一个字节，而System.in（这个InputStream的一个子类的预定义对象）却是从键盘读入信息的。

InputStream类还有若干个非抽象的方法，它们可以读入一个字节数组，或者跳过大量的

字节。这些方法都要调用抽象的**read**方法，因此，各个子类都只需覆盖这一个方法。

与此类似，**OutputStream**类定义了下面的抽象方法：

```
abstract void write(int b)
```

它可以向某个输出位置写出一个字节。

**read**和**write**方法在执行时都将阻塞，直至字节确实被读入或写出。这就意味着如果流不能被立即访问（通常是因为网络连接忙），那么当前的线程将被阻塞。这使得在这个方法等待指定的流变为可用的这段时间里，其他的线程就有机会去执行有用的工作。

**available**方法使我们可以去检查当前可用于读入的字节数量，这意味着像下面这样的代码片段就不可能被阻塞：

```
int bytesAvailable = in.available();
if (bytesAvailable > 0)
{
    byte[] data = new byte[bytesAvailable];
    in.read(data);
}
```

当你完成对流的读写时，应该通过调用**close**方法来关闭它，这个方法会释放掉十分有限的操作系统资源。如果一个应用程序打开了过多的流而没有关闭它们，那么系统资源将被耗尽。关闭一个输出流的同时也就是在清空用于该输出流的缓冲区：所有被临时置于缓冲区中，以便用更大的包的形式传递的字符在关闭输出流时都将被送出。特别是，如果不关闭文件，那么写出字节的最后一个包可能将永远也得不到传递。当然，我们还可以用**flush**方法来人为地清空这些输出。

即使某个流类提供了使用原生的**read**和**write**功能来工作的某些具体的方法，应用系统的程序员还是很少使用它们，因为大家感兴趣的数据可能包含数字、字符串和对象，而不是原生字节。

Java提供了众多从基本的**InputStream**和**OutputStream**类导出的类，这些类使我们可以处理那些以常用格式表示的数据，而不只是在字节级别上表示的数据。



### **java.io.InputStream 1.0**

- **abstract int read()**

从数据中读入一个字节，并返回该字节。这个**read**方法在碰到流的结尾时返回 -1。

- **int read(byte[] b)**

读入一个字节数组，并返回实际读入的字节数，或者在碰到流的结尾时返回 -1。这个**read**方法最多读入**b.length**个字节。

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

读入一个字节数组。这个**read**方法返回实际读入的字节数，或者在碰到流的结尾时返回 -1。

参数：**b** 数据读入的数组

**off** 第一个读入字节应该被放置的位置在**b**中的偏移量

len 读入字节的最大数量

- long skip(long n)

在输入流中跳过n个字节，返回实际跳过的字节数（如果碰到流的结尾，则可能小于n）。

- int available()

返回在不阻塞的情况下可用的字节数（回忆一下，阻塞意味着当前线程将失去它对资源的占用）。

- void close()

关闭这个输入流。

- void mark(int readlimit)

在输入流的当前位置打一个标记（并非所有的流都支持这个特性）。如果从输入流中已经读入的字节多于readlimit个，则这个流允许忽略这个标记。

- void reset()

返回到最后的标记，随后对read的调用将重新读入这些字节。如果当前没有任何标记，则这个流不被重置。

- boolean markSupported()

如果这个流支持打标记，则返回true。



## java.io.OutputStream 1.0

- abstract void write(int n)

写出一个字节的数据。

- void write(byte[] b)

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

写出所有字节或者某个范围的字节到数组b中。

参数：b 数据写出的数组

off 第一个写出字节在b中的偏移量

len 写出字节的最大数量

- void close()

清空并关闭输出流。

- void flush()

清空输出流，也就是将所有缓冲的数据发送到目的地。

### 1.1.2 完整的流家族

与C语言只有单一类型FILE\*即可工作良好不同，Java拥有一个包含各种流类型的流家族，其数量超过60个！请参见图1-1和图1-2。

让我们把流类家族中的成员按照它们的使用方法来进行划分，这样就形成了处理字节和字符的两个单独的层次结构。正如所见，InputStream和OutputStream类可以读写单个的字节或字节数组，这些类构成了图1-1所示的层次结构的基础。要想读写字符串和数字，就需要功能

更强大的子类，例如，`DataInputStream`和`DataOutputStream`可以以二进制格式读写所有的基本Java类型。最后，还包含了多个很有用的流，例如，`ZipInputStream`和`ZipOutputStream`可以以常见的ZIP压缩格式读写文件。

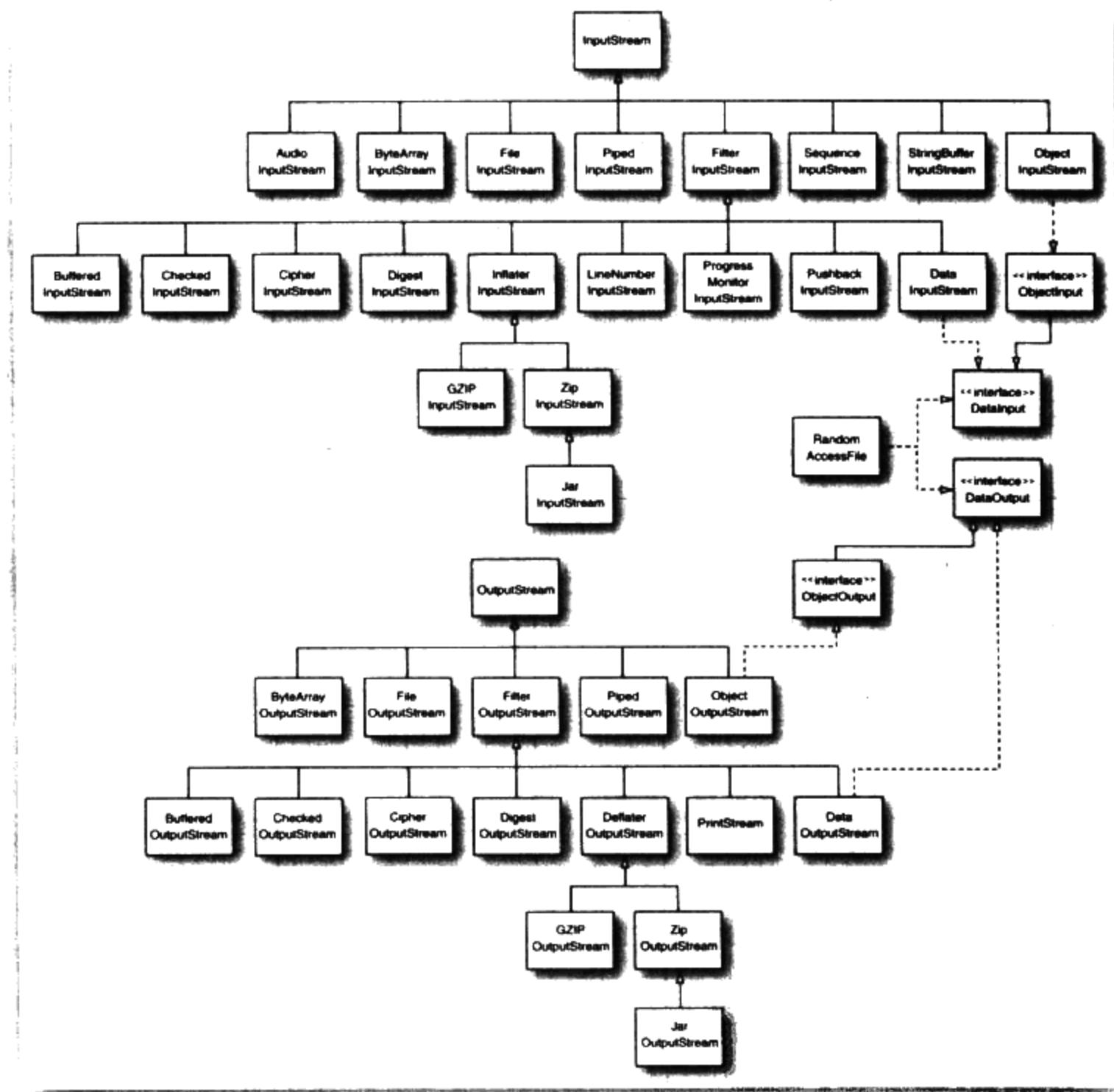


图1-1 输入流与输出流的层次结构

另一方面，对于Unicode文本，可以使用抽象类`Reader`和`Writer`的子类（请参见图1-2），`Reader`和`Writer`类的基本方法与`InputStream`和`OutputStream`中的方法类似。

```
abstract int read()
abstract void write(int c)
```

`read`方法将返回一个Unicode码元（作为一个在0~65535的整数），或者在碰到文件结尾时返回-1。`write`方法在调用时，要传递一个Unicode码元（请查看第1卷第3章有关Unicode码

元的讨论)。

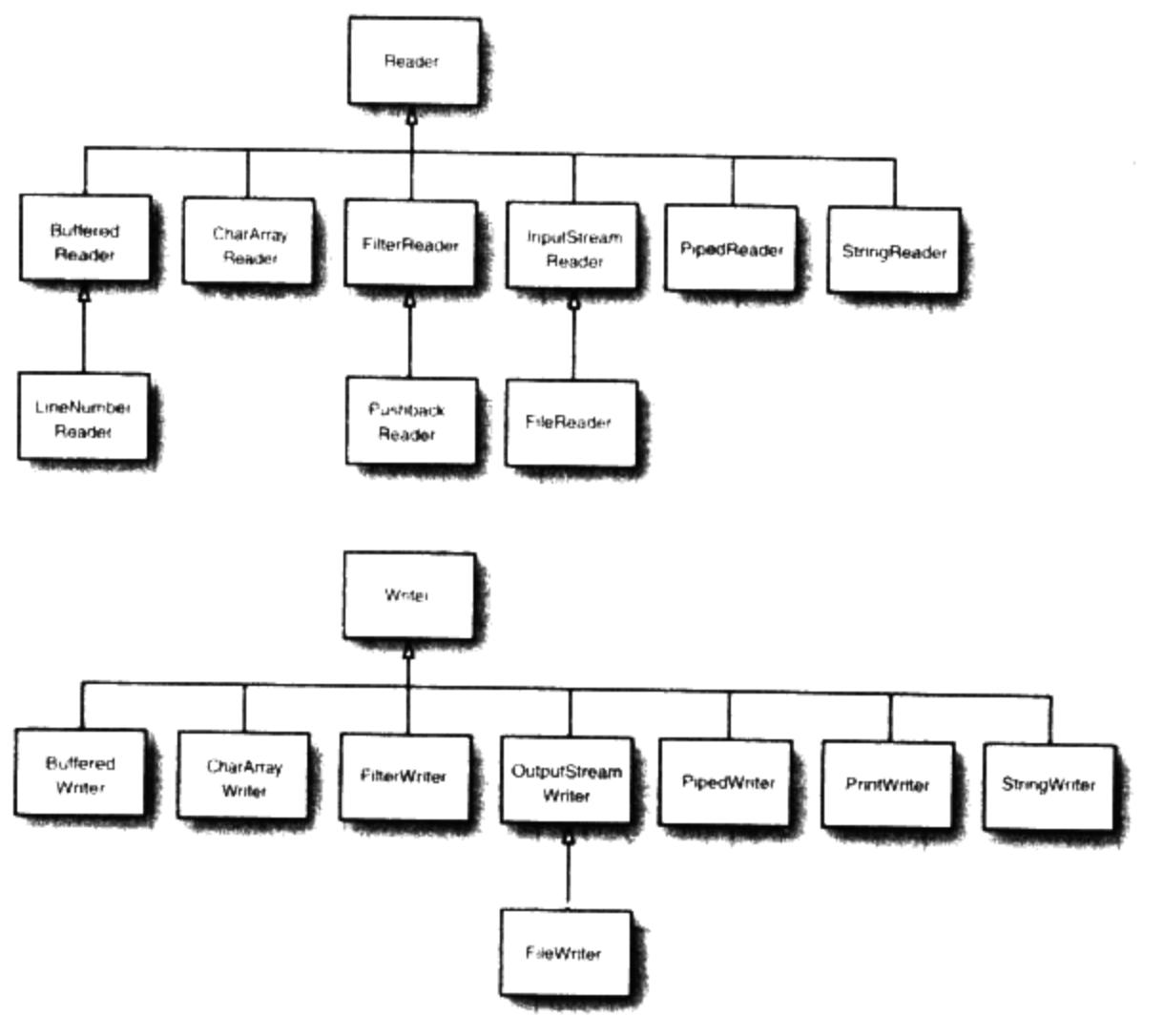


图1-2 Reader和Writer的层次结构

Java SE 5.0引入了4个附加的接口：**Closeable**、**Flushable**、**Readable**和**Appendable**（请查看图1-3）。前两个接口非常简单，它们分别拥有下面的方法：

```
void close() throws IOException
```

和

```
void flush()
```

**InputStream**、**OutputStream**、**Reader**和**Writer**都实现了**Closeable**接口，而**OutputStream**和**Writer**还实现了**Flushable**接口。

**Readable**接口只有一个方法：

```
int read(CharBuffer cb)
```

**CharBuffer**类拥有按顺序和随机地进行读写访问的方法，它表示一个内存中的缓冲区或者一个内存映像的文件（请查看第1.7.2节以了解细节）。

**Appendable**接口有两个用于添加单个字符和字符序列的方法：

```
Appendable append(char c)  
Appendable append(CharSequence s)
```

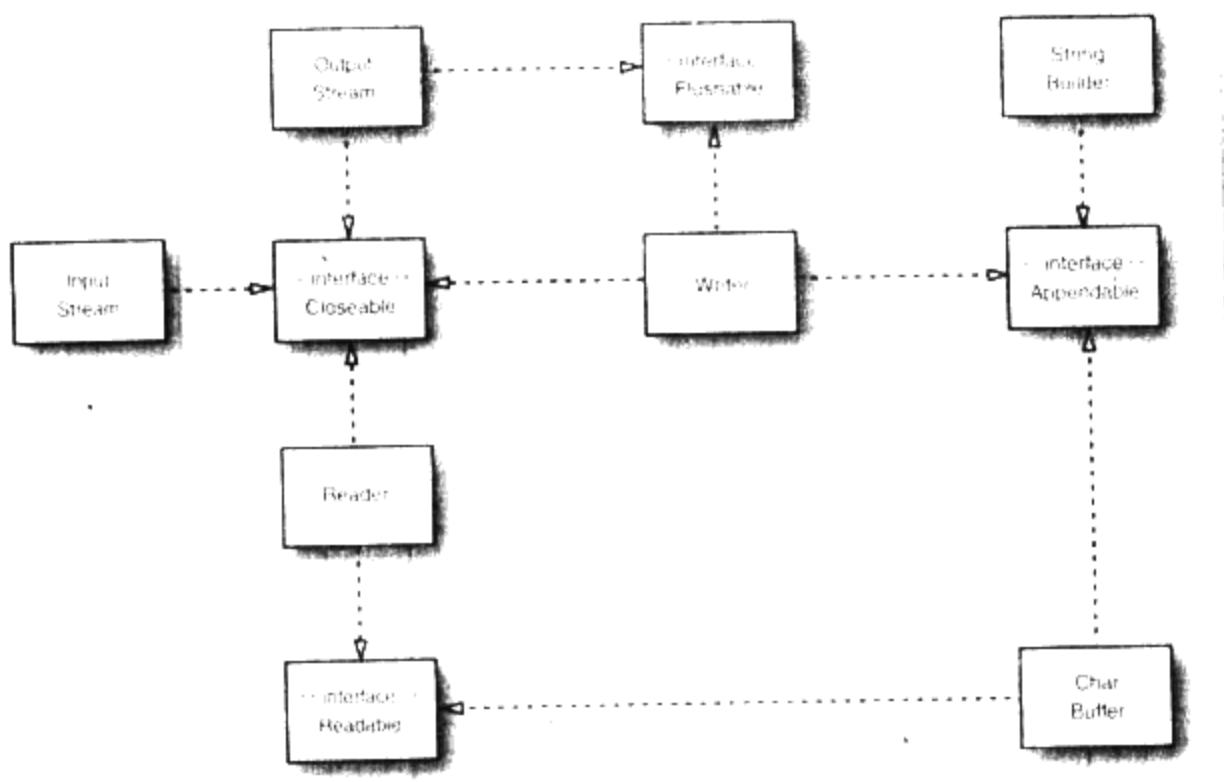


图1-3 Closeable、Flushable、Readable和Appendable接口

`CharSequence` 接口描述了一个`char` 值序列的基本属性，它是用 `String`、`CharBuffer`、`StringBuilder` 和 `StringBuffer` 来实现的。

在流类的家族中，只有 `Writer` 实现了 `Appendable`。

#### API `java.io.Closeable 5.0`

- `void close()`  
关闭这个 `Closeable`，这个方法可能会抛出 `IOException`。

#### API `java.io.Flushable 5.0`

- `void flush()`  
清空这个 `Flushable`。

#### API `java.lang.Readable 5.0`

- `int read(CharBuffer cb)`  
尝试着读入 `cb` 可以持有的数量的 `char` 值。返回读入的 `char` 值的数量，或者在从这个 `Readable` 中无法再获得更多的值时返回 `-1`。

#### API `java.lang.Appendable 5.0`

- `Appendable append(char c)`
- `Appendable append(CharSequence cs)`  
向这个 `Appendable` 中追加给定的码元或者给定的序列中的所有码元，返回 `this`。

**API** `java.lang.CharSequence` 1.4

- `char charAt(int index)`  
返回给定索引处的码元。
- `int length()`  
返回在这个序列中的码元的数量。
- `CharSequence subSequence(int startIndex, int endIndex)`  
返回由存储在startIndex到endIndex - 1处的所有码元构成的CharSequence。
- `String toString()`  
返回这个序列中所有码元构成的字符串。

### 1.1.3 组合流过滤器

`FileInputStream`和`FileOutputStream`可以提供附着在一个磁盘文件上的输入流和输出流，而你只需向其构造器提供文件名或文件的完整路径名。例如：

```
FileInputStream fin = new FileInputStream("employee.dat");
```

这行代码可以查看在用户目录下名为“employee.dat”的文件。

**!** 提示：因为所有在`java.io`中的类都将相对路径名解释为以用户工作目录开始，因此你可能希望了解这个目录。可以通过调用`System.getProperty("user.dir")`来获得这个信息。

与抽象类`InputStream`和`OutputStream`一样，这些类支持在字节级别的读写。也就是说，我们只能从`fin`对象中读入字节和字节数组。

```
byte b = (byte) fin.read();
```

正如下节中看到的，如果我们只有`DataInputStream`，那么我们就只能读入数字类型：

```
DataInputStream din = . . .;  
double s = din.readDouble();
```

但是正如`FileInputStream`没有任何读入数字类型的方法一样，`DataInputStream`也没有任何从文件中获取数据的方法。

Java使用了一种灵巧的机制来分离这两种职责。某些流（例如`FileInputStream`和由`URL`类的`openStream`方法返回的输入流）可以从文件和其他更外部的位置上获取字节，而其他的流（例如`DataInputStream`和`PrintWriter`）可以将字节组装到更有用的数据类型中。Java程序员必须对二者进行组合。例如，为了从文件中读入数字，首先需要创建一个`FileInputStream`，然后将其传递给`DataInputStream`的构造器：

```
FileInputStream fin = new FileInputStream("employee.dat");  
DataInputStream din = new DataInputStream(fin);  
double s = din.readDouble();
```

如果再次查看图1-1，你就会发现`FilterInputStream`和`FilterOutputStream`类。这些文件的子类用于添加原生字节流。

你可以通过嵌套过滤器来添加多重功能。例如，流在默认情况下是不被缓冲区缓存的，也就是说，每个对read的调用都会请求操作系统再分发一个字节。相比之下，请求一个数据块并将其置于缓冲区中会显得更加高效。如果我们想使用缓冲机制，以及用于文件的数据输入方法，那么就需要使用下面这种相当恐怖的构造器序列：

```
DataInputStream din = new DataInputStream(  
    new BufferedInputStream(  
        new FileInputStream("employee.dat")));
```

注意，我们把DataInputStream置于构造器链的最后，这是因为我们希望使用DataInputStream的方法，并且希望它们能够使用带缓冲机制的read方法。

有时当多个流链接在一起时，你需要跟踪各个中介流（intermediate stream）。例如，当读入输入时，你经常需要浏览下一个字节，以了解它是否是你想要的值。Java提供了用于此目的的PushbackInputStream：

```
PushbackInputStream pbin = new PushbackInputStream(  
    new BufferedInputStream(  
        new FileInputStream("employee.dat")));
```

现在你可以预读下一个字节：

```
int b = pbin.read();
```

并且在它并非你所期望的值时将其抛回。

```
if (b != '<') pbin.unread(b);
```

但是读入和不读入都只是应用于可回推（pushback）输入流的方法，如果你希望能够预先浏览并且还可以读入数字，那么你就需要一个既是可回推输入流，又是一个数据输入流的引用。

```
DataInputStream din = new DataInputStream(  
    pbin = new PushbackInputStream(  
        new BufferedInputStream(  
            new FileInputStream("employee.dat"))));
```

当然，在其他编程语言的流类库中，诸如缓冲机制和预览等细节都是自动处理的。因此相比较而言，Java就有一点麻烦，在这些情况下，它必须诉诸于将多个流过滤器组合起来。但是这种混合和匹配过滤器类以构建真正有用的流序列的能力，将带来极大的灵活性，例如，你可以从一个ZIP压缩文件中通过使用下面的流序列来读入数字（请参见图1-4）：

```
ZipInputStream zin = new ZipInputStream(new FileInputStream("employee.zip"));  
DataInputStream din = new DataInputStream(zin);
```

（请查看第1.4节以了解更多有关Java处理ZIP文件功能的知识。）



### java.io.FileInputStream 1.0

- `FileInputStream(String name)`
- `FileInputStream(File file)`

使用由name字符串或file对象指定路径名的文件创建一个新的文件输入流（File类在本章结尾处描述）。非绝对的路径名将按照相对于VM启动时所设置的工作目录来解析。

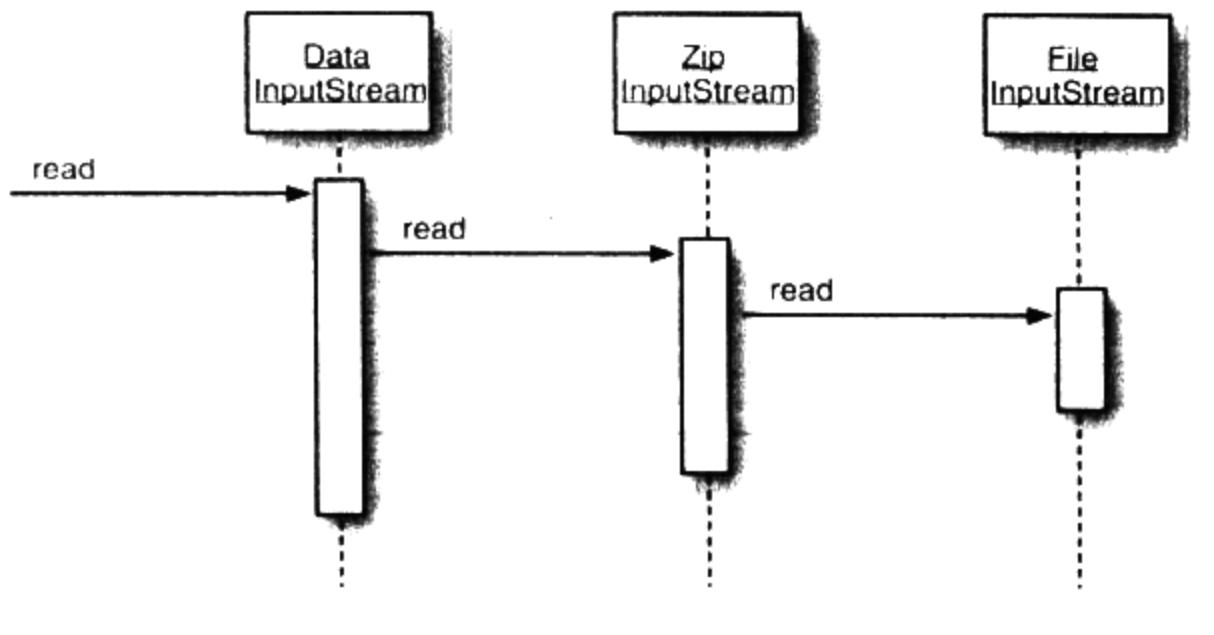


图1-4 过滤器流序列

**API** **java.io.FileOutputStream 1.0**

- `FileOutputStream(String name)`
- `FileOutputStream(String name, boolean append)`
- `FileOutputStream(File file)`
- `FileOutputStream(File file, boolean append)`

使用由`name`字符串或`file`对象指定路径名的文件创建一个新的文件输出流（`File`类在本章结尾处描述）。如果`append`参数为`true`，那么数据将被添加到文件尾，而具有相同名字的已有文件不会被删除；否则，这个方法会删除所有具有相同名字的已有文件。

**API** **java.io.BufferedReader 1.0**

- `BufferedReader(InputStream in)`

创建一个带缓冲区的流。带缓冲区的输入流在从流中读入字符时，不会每次都对设备访问。当缓冲区为空时，会向缓冲区中读入一个新的数据块。

**API** **java.io.BufferedWriter 1.0**

- `BufferedWriter(OutputStream out)`

创建一个带缓冲区的流。带缓冲区的输出流在收集要写出的字符时，不会每次都对设备访问。当缓冲区填满或当流被清空时，数据就被写出。

**API** **java.io.PushbackInputStream 1.0**

- `PushbackInputStream(InputStream in)`
- `PushbackInputStream(InputStream in, int size)`

构建一个可以预览一个字节或者具有指定尺寸的回推缓冲区的流。

- `void unread(int b)`

回推一个字节，它可以在下次调用`read`时被再次获取。

参数：b 要再次读入的字节。

## 1.2 文本输入与输出

在保存数据时，可以选择二进制或文本格式。例如，整数1234存储成二进制数时，它被写为由字节00 00 04 D2构成的序列（十六进制表示法），而存储成文本格式时，它被存成了字符串“1234”。尽管二进制I/O高速且高效，但是它不宜人们阅读。我们首先讨论文本I/O，然后在第1.3节中讨论二进制I/O。

在存储文本字符串时，需要考虑字符编码（character encoding）方式问题。在UTF-16编码方式中，字符串“1234”编码为00 31 00 32 00 33 00 34（十六进制）。但是，许多程序都希望文本文件按照其他的编码方式编码。在ISO 8859-1这种在美国和西欧最常用的编码方式中，字符串将写出为31 32 33 34，其中并没有任何0字节。

`OutputStreamWriter`类将使用选定的字符编码方式，把Unicode字符流转换为字节流。相反地，`InputStreamReader`类将包含字节（用某种字符编码方式表示的字符）的输入流转换为可以产生Unicode字符的读入器。

例如，下面的代码就展示了如何让一个输入读入器可以从控制台读入键盘敲击信息，并将其转换为Unicode：

```
InputStreamReader in = new InputStreamReader(System.in);
```

这个输入流读入器会假定主机系统所使用的默认字符编码方式，例如西欧采用ISO 8859-1。你可以通过在`InputStreamReader`的构造器中指定的方式来选择不同的编码方式。例如，

```
InputStreamReader in = new InputStreamReader(new FileInputStream("kremlin.dat"), "ISO8859_5");
```

请查看第1.2.4节以了解字符编码方式的更多信息。

因为将读入器或写出器附着到文件上非常普遍，所以就提供了`FileReader`和`FileWriter`这一对使用方便的类来专门用于此目的。例如，下面的写出器定义：

```
FileWriter out = new FileWriter("output.txt");
```

等同于：

```
FileWriter out = new FileWriter(new FileOutputStream("output.txt"));
```

### 1.2.1 如何写出文本输出

对于文本输出，你可能希望使用`PrintWriter`。这个类拥有以文本格式打印字符串和数字的方法，它甚至还有一个将`PrintWriter`链接到`FileWriter`的方便方法，下面的语句：

```
PrintWriter out = new PrintWriter("employee.txt");
```

等同于：

```
PrintWriter out = new PrintWriter(new FileWriter("employee.txt"));
```

为了写出到打印写出器，需要使用与使用`System.out`时相同的方法`print`、`println`和

`printf`。你可以用这些方法来打印数字（`int`、`short`、`long`、`float`、`double`）、字符、`boolean`值、字符串和对象。

例如，考虑下面的代码：

```
String name = "Harry Hacker";
double salary = 75000;
out.print(name);
out.print(' ');
out.println(salary);
```

它将把下面的字符：

Harry Hacker 75000.0

写出到写出器`out`，之后这些字符将会被转换成字节并最终写入`employee.txt`中。

`println`方法在行中添加了对目标系统来说恰当的行结束符（Windows系统是"`\r\n`"，UNIX系统是"`\n`"），也就是通过调用`System.getProperty("line.separator")`而获得的字符串。

如果写出器设置为自动清空模式，那么只要`println`被调用，缓冲区中的所有字符都会被发送到它们的目的地（打印写出器总是带缓冲区的）。默认情况下，自动清空机制是不使能的。你可以通过使用`PrintWriter(Writer out, Boolean autoFlush)`来使能或禁用自动清空机制：

```
PrintWriter out = new PrintWriter(new FileWriter("employee.txt"), true); // autoflush
```

`print`方法不抛出异常，你可以调用`checkError`方法来查看流是否出现了某些错误。



**注意：**Java的老手们可能会很想知道`PrintStream`类和`System.out`到底怎么了。在Java 1.0中，`PrintStream`类只是通过将高字节丢弃的方式把所有Unicode字符截断成ASCII字符。很明显，这并非一种干净利落和可移植的方式，这个问题在Java 1.1中通过引入读入器和写出器得到了修正。为了与已有的代码兼容，`System.in`、`System.out`和`System.err`仍旧是流而不是读入器和写出器。但是现在`PrintStream`类在内部采用与`PrintWriter`相同的方式将Unicode字符转换成了默认的主机编码方式。当你在使用`print`和`println`方法时，`PrintStream`类型的对象的行为看起来确实很像打印写出器，但是与打印写出器不同的是，它们允许我们用`write(int)`和`write(byte[])`方法输出原生字节。



### java.io.PrintWriter 1.1

- `PrintWriter(Writer out)`
- `PrintWriter(Writer out, boolean autoFlush)`

创建一个新的`PrintWriter`。

参数：`out` 一个用于字符输出的写出器

`autoflush` 如果为`true`，则`println`方法将清空输出缓冲区（默认值：`false`）

- `PrintWriter(OutputStream out)`
- `PrintWriter(OutputStream out, boolean autoflush)`

通过创建必需的中介`OutputStreamWriter`，从已有的`OutputStream`中创建一个新的`PrintWriter`。

- `PrintWriter(String filename)`

- `PrintWriter(File file)`

通过创建必需的中介`FileWriter`, 创建一个向给定的文件写出的新的`PrintWriter`。

- `void print(Object obj)`

通过打印从`toString`产生的字符串来打印一个对象。

参数: `obj` 要打印的对象

- `void print(String s)`

打印一个Unicode字符串。

- `void println(String s)`

打印一个字符串, 后面紧跟一个行终止符。如果这个流处于自动清空模式, 那么就会清空这个流。

- `void print(char[] s)`

打印在给定的字符串中的所有Unicode字符。

- `void print(char c)`

打印一个Unicode字符。

- `void print(int i)`

- `void print(long l)`

- `void print(float f)`

- `void print(double d)`

- `void print(boolean b)`

以文本格式打印给定的值。

- `void printf(String format, Object... args)`

按照格式字符串指定的方式打印给定的值。请查看第I卷第3章以了解格式化字符串的相关规范。

- `boolean checkError()`

如果产生格式化或输出错误, 则返回`true`。一旦这个流碰到了错误, 它就受到了感染, 并且所有对`checkError`的调用都将返回`true`。

## 1.2.2 如何读入文本输入

众所周知:

- 以二进制格式写出数据, 需要使用`DataOutputStream`。

- 以文本格式写出数据, 需要使用`PrintWriter`。

因此, 你可能认为存在着与`DataInputStream`类似的类允许我们以文本格式读入数据, 而在第I卷中我们广泛使用的与此最接近的类是`Scanner`。但是, 在Java SE 5.0之前, 处理文本输入的惟一方式就是通过`BufferedReader`类, 它拥有一个`readLine`方法, 使得我们可以读入一行文本。你需要将带缓冲区的读入器与输入源组合起来:

```
BufferedReader in = new BufferedReader(new FileReader("employee.txt"));
```

`readLine`方法在没有输入时返回`null`。然而，一个典型的输入循环类似于这样：

```
String line;
while ((line = in.readLine()) != null)
{
    do something with line
}
```

然而，`BufferedReader`没有任何用于读入数字的方法，我们建议你使用`Scanner`来读入文本输入。

### 1.2.3 以文本格式存储对象

在本节，我们将带你领略一个示例程序，它将一个`Employee`记录数组存储成了一个文本文档，其中每个记录都保存在单独的一行中，而实例的域彼此之间使用分隔符分离开，这里我们使用竖线（|）作为分隔符（冒号（:）是另一种流行的选择，有趣的是，每个人都会使用不同的分隔符）。因此，我们这里是在假设不会出现在我们要存储的字符串中确实会出现存在|的情况。

下面是一个记录集的样本：

```
Harry Hacker|35500|1989|10|1
Carl Cracker|75000|1987|12|15
Tony Tester|38000|1990|3|15
```

写出记录相当简单，因为我们要写出到一个文本文档中，所以我们使用`PrintWriter`类。我们直接写出所有的域，每个域后面跟着一个|，而最后一个域的后面跟着一个`\n`。这项工作是在下面这个我们添加到`Employee`类中的`writeData`方法里完成的：

```
public void writeData(PrintWriter out) throws IOException
{
    GregorianCalendar calendar = new GregorianCalendar();
    calendar.setTime(hireDay);
    out.println(name + "|"
        + salary + "|"
        + calendar.get(Calendar.YEAR) + "|"
        + (calendar.get(Calendar.MONTH) + 1) + "|"
        + calendar.get(Calendar.DAY_OF_MONTH));
}
```

为了读入记录，我们每次读入一行，然后分离所有的域。我们使用一个扫描器来读入每一行，然后用`String.split`方法将这一行断开成一组标记。

```
public void readData(Scanner in)
{
    String line = in.nextLine();
    String[] tokens = line.split("\\|");
    name = tokens[0];
    salary = Double.parseDouble(tokens[1]);
    int y = Integer.parseInt(tokens[2]);
    int m = Integer.parseInt(tokens[3]);
    int d = Integer.parseInt(tokens[4]);
    GregorianCalendar calendar = new GregorianCalendar(y, m - 1, d);
    hireDay = calendar.getTime();
}
```

split方法的参数是一个描述分隔符的正则表达式，我们在本章的末尾将详细讨论正则表达式。碰巧的是，竖线在正则表达式中具有特殊的含义，因此它需要用\字符来转义，而这个\又需要用另一个\来转义，这样就产生了“\\|”表达式。

完整的程序如程序清单1-1所示。静态方法：

```
void writeData(Employee[] e, PrintWriter out)
```

首先写出该数组的长度，然后写出每条记录。静态方法：

```
Employee[] readData(BufferedReader in)
```

首先读入该数组的长度，然后读入每条记录。这显得稍微有点棘手：

```
int n = in.nextInt();
in.nextLine(); // consume newline
Employee[] employees = new Employee[n];
for (int i = 0; i < n; i++)
{
    employees[i] = new Employee();
    employees[i].readData(in);
}
```

对nextInt的调用读入的是数组长度，而不是坐在行尾的换行字符，我们必须处理掉这个换行符，以便readData方法在调用nextLine方法时可以获得下一个输入行。

### 程序清单1-1 TextFileTest.java

```
1. import java.io.*;
2. import java.util.*;
3.
4. /**
5. * @version 1.12 2007-06-22
6. * @author Cay Horstmann
7. */
8. public class TextFileTest
9. {
10.     public static void main(String[] args)
11.     {
12.         Employee[] staff = new Employee[3];
13.
14.         staff[0] = new Employee("Carl Cracker", 75000, 1987, 12, 15);
15.         staff[1] = new Employee("Harry Hacker", 50000, 1989, 10, 1);
16.         staff[2] = new Employee("Tony Tester", 40000, 1990, 3, 15);
17.
18.         try
19.         {
20.             // save all employee records to the file employee.dat
21.             PrintWriter out = new PrintWriter("employee.dat");
22.             writeData(staff, out);
23.             out.close();
24.
25.             // retrieve all records into a new array
26.             Scanner in = new Scanner(new FileReader("employee.dat"));
27.             Employee[] newStaff = readData(in);
28.             in.close();
29.         }
```

```
30.         // print the newly read employee records
31.         for (Employee e : newStaff)
32.             System.out.println(e);
33.     }
34.     catch (IOException exception)
35.     {
36.         exception.printStackTrace();
37.     }
38. }
39.
40. /**
41. * Writes all employees in an array to a print writer
42. * @param employees an array of employees
43. * @param out a print writer
44. */
45. private static void writeData(Employee[] employees, PrintWriter out) throws IOException
46. {
47.     // write number of employees
48.     out.println(employees.length);
49.
50.     for (Employee e : employees)
51.         e.writeData(out);
52. }
53. /**
54. * Reads an array of employees from a scanner
55. * @param in the scanner
56. * @return the array of employees
57. */
58. private static Employee[] readData(Scanner in)
59. {
60.     // retrieve the array size
61.     int n = in.nextInt();
62.     in.nextLine(); // consume newline
63.
64.     Employee[] employees = new Employee[n];
65.     for (int i = 0; i < n; i++)
66.     {
67.         employees[i] = new Employee();
68.         employees[i].readData(in);
69.     }
70.     return employees;
71. }
72. }
73.
74. class Employee
75. {
76.     public Employee()
77.     {
78.     }
79.
80.     public Employee(String n, double s, int year, int month, int day)
81.     {
82.         name = n;
83.         salary = s;
84.         GregorianCalendar calendar = new GregorianCalendar(year, month - 1, day);
85.         hireDay = calendar.getTime();
86.     }
}
```

```
87.  
88.    public String getName()  
89.    {  
90.        return name;  
91.    }  
92.  
93.    public double getSalary()  
94.    {  
95.        return salary;  
96.    }  
97.  
98.    public Date getHireDay()  
99.    {  
100.        return hireDay;  
101.    }  
102.  
103.    public void raiseSalary(double byPercent)  
104.    {  
105.        double raise = salary * byPercent / 100;  
106.        salary += raise;  
107.    }  
108.  
109.    public String toString()  
110.    {  
111.        return getClass().getName() + "[name=" + name + ",salary=" + salary + ",hireDay="  
112.            + hireDay + "]";  
113.    }  
114.  
115.    /**  
116.     * Writes employee data to a print writer  
117.     * @param out the print writer  
118.     */  
119.    public void writeData(PrintWriter out)  
120.    {  
121.        GregorianCalendar calendar = new GregorianCalendar();  
122.        calendar.setTime(hireDay);  
123.        out.println(name + "|" + salary + "|" + calendar.get(Calendar.YEAR) + "|"  
124.            + (calendar.get(Calendar.MONTH) + 1) + "|" + calendar.get(Calendar.DAY_OF_MONTH));  
125.    }  
126.  
127.    /**  
128.     * Reads employee data from a buffered reader  
129.     * @param in the scanner  
130.     */  
131.    public void readData(Scanner in)  
132.    {  
133.        String line = in.nextLine();  
134.        String[] tokens = line.split("\\|");  
135.        name = tokens[0];  
136.        salary = Double.parseDouble(tokens[1]);  
137.        int y = Integer.parseInt(tokens[2]);  
138.        int m = Integer.parseInt(tokens[3]);  
139.        int d = Integer.parseInt(tokens[4]);  
140.        GregorianCalendar calendar = new GregorianCalendar(y, m - 1, d);  
141.        hireDay = calendar.getTime();  
142.    }  
143.
```

```
144.     private String name;  
145.     private double salary;  
146.     private Date hireDay;  
147. }
```

#### 1.2.4 字符集

在过去，国际化字符集已经得到了处理，但是处理得很不系统，散布在Java类库的各处。在Java SE 1.4中引入的java.nio包用引入的Charset类统一了对字符集的转换（注意s是小写的）。

字符集建立了两字节Unicode码元序列与使用本地字符编码方式的字节序列之间的映射。ISO-8859-1是最流行的字符编码方式之一，这是一种对Unicode前256个字符进行单字节编码的方式。ISO-8859-15则更加重要，它用法语和芬兰语中带有重音符号的字母取代了ISO-8859-1中某些不常用到的字符，更重要的是，它用欧元符号(€)取代了编码为0xA4的“国际货币”字符¤。通常用于日语和中文的可变字节编码方式是另一个字符编码方式的例子。

Charset类使用的是由IANA字符集注册中心标准化的字符集名字（<http://www.iana.org/assignments/character-sets>），这些名字与以前版本所使用的名字略有差异。例如，ISO-8859-1的“官方”名字现在是“ISO-8859-1”而不再是“ISO8859-1”，而后者是直到Java SE 1.3还在使用的名字。

 注意：在<http://czyborra.com/charsets/iso8859.html>处可以享受到一顿极佳的“ISO 8859字母大餐”。

你可以通过调用静态的forName方法来获得一个Charset，只需向这个方法传递一个官方名字或者是它的某个别名：

```
Charset cset = Charset.forName("ISO-8859-1");
```

字符集名字是大小写不敏感的。

为了兼容其他的命名惯例，每个字符集都可以拥有许多别名。例如，ISO-8859-1的别名有：

```
ISO8859-1  
ISO_8859_1  
ISO8859_1  
ISO_8859-1  
ISO_8859-1:1987  
8859_1  
latin1  
11  
csISOLatin1  
iso-ir-100  
cp819  
IBM819  
IBM-819  
819
```

aliases方法可以返回由别名构成的Set对象。下面是迭代遍历这些别名的代码：

```
Set<String> aliases = cset.aliases();  
for (String alias : aliases)  
    System.out.println(alias);
```

为了确定在某个特定实现中哪些字符集是可用的，可以调用静态的availableCharsets方法。使用下面的代码可以确定所有可用字符集的名字：

```
Map<String, Charset> charsets = Charset.availableCharsets();
for (String name : charsets.keySet())
    System.out.println(name);
```

表1-1列出了每一种Java实现都必须涵盖的字符编码方式，表1-2列出了Java开发包（JDK）默认安装的编码模式，表1-3中的字符集只能在使用非欧洲语言的操作系统上安装。

表1-1 必需的字符编码方式

| Charset标准名 | 遗 留 名 称               | 描 述                               |
|------------|-----------------------|-----------------------------------|
| US-ASCII   | ASCII                 | 美国信息交换标准码                         |
| ISO-8859-1 | ISO8859_1             | ISO 8859-1，拉丁文字符表1号               |
| UTF-8      | UTF8                  | 8位Unicode转换格式                     |
| UTF-16     | UTF-16                | 16位Unicode转换格式，字节顺序由可选的初始字节顺序标记指定 |
| UTF-16BE   | UnicodeBigUnmarked    | 16位Unicode转换格式，遵从高位在前顺序           |
| UTF-16LE   | UnicodeLittleUnmarked | 16位Unicode转换格式，遵从低位在前顺序           |

表1-2 基本的字符编码方式

| Charset标准名   | 遗 留 名      | 描 述                    |
|--------------|------------|------------------------|
| ISO8859-2    | ISO8859_2  | ISO 8859-2，拉丁文字符表2号    |
| ISO8859-4    | ISO8859_4  | ISO 8859-4，拉丁文字符表4号    |
| ISO8859-5    | ISO8859_5  | ISO 8859-5，拉丁文/西里尔文字符表 |
| ISO8859-7    | ISO8859_7  | ISO 8859-7，拉丁文/希腊文字符表  |
| ISO8859-9    | ISO8859_9  | ISO 8859-9，拉丁文字符表5号    |
| ISO8859-13   | ISO8859_13 | ISO 8859-13，拉丁文字符表7号   |
| ISO8859-15   | ISO8859_15 | ISO 8859-15，拉丁文字符表9号   |
| windows-1250 | Cp1250     | Windows东欧文             |
| windows-1251 | Cp1251     | Windows西里尔文            |
| windows-1252 | Cp1252     | Windows拉丁文-1           |
| windows-1253 | Cp1253     | Windows希腊文             |
| windows-1254 | Cp1254     | Windows土耳其文            |
| windows-1257 | Cp1257     | Windows波罗的文            |

表1-3 扩展的字符编码方式

| Charset标准名 | 遗 留 名      | 描 述                             |
|------------|------------|---------------------------------|
| Big5       | Big5       | Big5，繁体中文                       |
| Big5-HKSCS | Big5_HKSCS | Big5带香港扩展，繁体中文                  |
| EUC-JP     | EUC_JP     | JIS X 0201、0208、0212，EUC编码方式，日文 |
| EUC-KR     | EUC_KR     | KSC 5601，EUC编码方式，韩文             |
| GB18030    | GB18030    | 简体中文，中华人民共和国标准                  |
| GBK        | GBK        | GBK，简体中文                        |
| ISCII91    | ISCII91    | 印度文脚本的ISCII91编码方式               |

(续)

| Charset标准名     | 遗 留 名        | 描 述                                 |
|----------------|--------------|-------------------------------------|
| ISO-2022-JP    | ISO2022JP    | ISO 2022形式的JIS X 0201, 0208, 日文     |
| ISO-2022-KR    | ISO2022KR    | ISO 2022 KR, 韩文                     |
| ISO8859-3      | ISO8859_3    | ISO 8859-3, 拉丁文字母表3号                |
| ISO8859-6      | ISO8859_6    | ISO 8859-6, 拉丁文/阿拉伯文字符表             |
| ISO8859-8      | ISO8859_8    | ISO 8859-8, 拉丁文/希伯来文字符表             |
| Shift_JIS      | SJIS         | Shift-JIS, 日文                       |
| TIS-620        | TIS620       | TIS620, 泰文                          |
| windows-1255   | Cp1255       | Windows希伯来文                         |
| windows-1256   | Cp1256       | Windows阿拉伯文                         |
| windows-1258   | Cp1258       | Windows越南文                          |
| windows-31j    | MS932        | Windows日文                           |
| x-EUC-CN       | EUC_CN       | GB2312, EUC编码方式, 简体中文               |
| x-EUC-JP-LINUX | EUC_JP_LINUX | JIS X 0201, 0208, EUC编码方式, 日文       |
| x-EUC-TW       | EUC_TW       | CNS11643 (Plane 1-3), EUC编码方式, 繁体中文 |
| x-MS950-HKSCS  | MS950_HKSCS  | Windows繁体中文带香港扩展                    |
| x-mswin-936    | MS936        | Windows简体中文                         |
| x-windows-949  | MS949        | Windows韩文                           |
| x-windows-950  | MS950        | Windows繁体中文                         |

本地编码方式模式不能表示所有的Unicode字符，如果某个字符不能表示，它将被转换成？。

一旦有了字符集，就可以使用它将Unicode字符串转换成编码而成的字节序列，下面是如何编码Unicode字符串的代码：

```
String str = . . .;
ByteBuffer buffer = cset.encode(str);
byte[] bytes = buffer.array();
```

与之相反，要想解码字节序列，需要有字节缓冲区。使用`ByteBuffer`数组的静态方法`wrap`可以将一个字节数组转换成一个字节缓冲区。`decode`方法的结果是一个`CharBuffer`，调用它的`toString`方法可以获得一个字符串。

```
byte[] bytes = . . .;
ByteBuffer bbuf = ByteBuffer.wrap(bytes, offset, length);
CharBuffer cbuf = cset.decode(bbuf);
String str = cbuf.toString();
```

### API `java.nio.charset.Charset 1.4`

- `static SortedMap availableCharsets()`

获取这个虚拟机可用的所有字符集。返回一个映射表，它的键是字符集的名字，值是字符集。

- `static Charset forName(String name)`

获取给定名字的字符集。

- `Set aliases()`  
返回这个字符集的别名集。
- `ByteBuffer encode(String str)`  
将给定的字符串编码为字节序列。
- `CharBuffer decode(ByteBuffer buffer)`  
解码给定的字节序列。无法识别的输入将被转换成Unicode的“替代字符”('＼ufffd')。

#### API `java.nio.ByteBuffer 1.4`

- `byte[] array()`  
返回这个缓冲区所管理的字节数组。
- `static ByteBuffer wrap(byte[] bytes)`
- `static ByteBuffer wrap(byte[] bytes, int offset, int length)`  
返回管理给定的字节数组或给定字节数组的某个范围的字节缓冲区。

#### API `java.nio.CharBuffer`

- `char[] array()`  
返回这个缓冲区所管理的码元数组。
- `char charAt(int index)`  
返回给定索引处的码元。
- `String toString()`  
返回由这个缓冲区所管理的码元构成的字符串。

### 1.3 读写二进制数据

`DataOutput`接口定义了下面用于以二进制格式写数组、字符、`boolean`值和字符串的方法：

```
writeChars
writeByte
writeInt
writeShort
writeLong
writeFloat
writeDouble
writeChar
writeBoolean
writeUTF
```

例如，`writeInt`总是将一个整数写出为4字节的二进制数量值，而不管它有多少位，`writeDouble`总是将一个`double`值写出为8字节的二进制数量值。这样产生的结果并非人可阅读的，但是对于给定类型的每个值，所需的空间都是相同的，而且将其读回也比解析文本要更快。

 **注意：**根据你所使用的平台类型，有两种不同方法用来在内存存储整数和浮点数。例如，假设你使用的是4字节的`int`，如果有一个十进制数1234，也就是十六进制的4D2（ $1234 = 4 \times 256 + 13 \times 16 + 2$ ），那么它可以按照内存中4字节的第一个字节存储最高位字节的方

式来存储为：00 00 04 D2，这就是所谓的高位在前顺序；我们也可以从最低位字节开始：D2 04 00 00，这种方式自然就是所谓的低位在前顺序。例如，SPARC使用的是高位在前顺序，而Pentium使用的则是低位在前顺序。这就可能会带来问题，当存储C或者C++文件时，数据会精确地按照处理器存储它们的方式来存储，这就使得即使是最简单的数据在从一个平台迁移到另一个平台上时也是一种挑战。在Java中，所有的值都按照高位在前的模式写出，这使得Java数据文件可以独立于平台。

`writeUTF`方法使用修订版的8位Unicode转换格式写出字符串。与直接使用标准的UTF-8编码方式（如表1-4所示）不同，字符型字符串首先用UTF-16（请查看表1-5）表示，其结果之后使用UTF-8规则进行编码。修订后的编码方式对于编码大于0xFFFF的字符的处理有所不同，它用于向后兼容在Unicode还没有超过16位时构建的虚拟机。

表1-4 UTF-8编码方式

| 字符范围           | 编码方式   |
|----------------|--|
| 0...7F         | 0a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>  |
| 80...7FF       | 110a <sub>10</sub> a <sub>9</sub> a <sub>8</sub> a <sub>7</sub> a <sub>6</sub> 10a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>   |
| 800...FFFF     | 1110a <sub>15</sub> a <sub>14</sub> a <sub>13</sub> a <sub>12</sub> 10a <sub>11</sub> a <sub>10</sub> a <sub>9</sub> a <sub>8</sub> a <sub>7</sub> a <sub>6</sub> 10a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>  |
| 10000...10FFFF | 11110a <sub>20</sub> a <sub>19</sub> a <sub>18</sub> 10a <sub>17</sub> a <sub>16</sub> a <sub>15</sub> a <sub>14</sub> a <sub>13</sub> a <sub>12</sub> 10a <sub>11</sub> a <sub>10</sub> a <sub>9</sub> a <sub>8</sub> a <sub>7</sub> a <sub>6</sub> 10a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub> |

表1-5 UTF-16编码方式

| 字符范围           | 编码方式   |
|----------------|--|
| 0...FFFF       | a <sub>15</sub> a <sub>14</sub> a <sub>13</sub> a <sub>12</sub> a <sub>11</sub> a <sub>10</sub> a <sub>9</sub> a <sub>8</sub> a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>  |
| 10000...10FFFF | 110110b <sub>19</sub> b <sub>18</sub> b <sub>17</sub> b <sub>16</sub> a <sub>15</sub> a <sub>14</sub> a <sub>13</sub> a <sub>12</sub> a <sub>11</sub> a <sub>10</sub> 110111a <sub>9</sub> a <sub>8</sub> a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub><br>其中b <sub>19</sub> b <sub>18</sub> b <sub>17</sub> b <sub>16</sub> = a <sub>20</sub> a <sub>19</sub> a <sub>18</sub> a <sub>17</sub> a <sub>16</sub> - 1 |

因为没有其他方法会使用UTF-8的这种修订，所以你应该只在写出用于Java虚拟机的字符串时才使用`writeUTF`方法来，例如，当你需要编写一个生成字节码的程序时。对于其他的目的，都应该使用`writeChars`方法。

 注意：请查看RFC 2279 (<http://ietf.org/rfc/rfc2279.txt>) 和RFC 2781 (<http://ietf.org/rfc/rfc2781.txt>) 以了解UTF-8和UTF-16的定义。

为了读回数据，可以使用在`DataInput`接口中定义的下列方法：

```
readInt  
readShort  
readLong  
readFloat  
readDouble  
readChar  
readBoolean  
readUTF
```

`DataInputStream`类实现了`DataInput`接口，要想从文件中读入二进制数据，你需要将`DataInputStream`与某个字节源相组合，例如`FileInputStream`：

```
DataInputStream in = new DataInputStream(new FileInputStream("employee.dat"));
```

与此类似，要想写出二进制数据，你可以使用实现了DataOutput接口的DataOutputStream类：

```
DataOutputStream out = new DataOutputStream(new FileOutputStream("employee.dat"));
```

### API java.io.DataInput 1.0

- boolean readBoolean()
- byte readByte()
- char readChar()
- double readDouble()
- float readFloat()
- int readInt()
- long readLong()
- short readShort()

读入一个给定类型的值。

- void readFully(byte[] b)

将字节读入到数组b中，其间阻塞直至所有字节都读入。

参数：b 数据读入的缓冲区

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

将字节读入到数组b中，其间阻塞直至所有字节都读入。

参数：b 数据读入的缓冲区

off 数据起始位置的偏移量

len 读入字节的最大数量

- String readUTF()

读入由“修订过的UTF-8”格式的字符构成的字符串。

- int skipBytes(int n)

跳过n个字节，其间阻塞直至所有字节都被跳过。

参数：n 被跳过的字节数

### API java.io.DataOutput 1.0

- void writeBoolean(boolean b)
- void writeByte(int b)
- void writeChar(int c)
- void writeDouble(double d)
- void writeFloat(float f)
- void writeInt(int i)
- void writeLong(long l)

- `void writeShort(int s)`  
    写出一个给定类型的值。
- `void writeChars(String s)`  
    写出字符串中的所有字符。
- `void writeUTF(String s)`  
    写出由“修订过的UTF-8”格式的字符构成的字符串。

## 随机访问文件

`RandomAccessFile`类可以在文件中的任何位置查找或写入数据。磁盘文件都是随机访问的，但是从网络而来的数据流却不是。你可以打开一个随机访问文件，只用于读入或者同时用于读写，我们可以通过使用字符串“`r`”（用于读入访问）或“`rw`”（用于读入/写出访问）作为构造器的第二个参数来指定这个选项。

```
RandomAccessFile in = new RandomAccessFile("employee.dat", "r");
RandomAccessFile inOut = new RandomAccessFile("employee.dat", "rw");
```

当你将已有文件打开成`RandomAccessFile`时，这个文件并不会被删除。

随机访问文件有一个表示下一个将被读入或写出的字节所处位置的文件指针，`seek`方法可以将这个文件指针设置到文件内部的任意字节位置，`seek`的参数是一个`long`类型的整数，它的值位于0到文件按照字节来度量的长度之间。

`getFilePointer`方法将返回文件指针的当前位置。

`RandomAccessFile`类同时实现了`DataInput`和`DataOutput`接口。为了读写随机访问文件，可以使用在前面章节中讨论过的诸如`readInt/writeInt`和`readChar/writeChar`之类的方法。

我们现在要剖析一个将雇员记录存储到随机访问文件中的示例程序，其中每条记录都拥有相同的大小，这样我们可以很容易地读入任何记录。假设你希望将文件指针置于第三条记录处，那么你只需将文件指针置于恰当的字节位置，然后就可以开始读入了。

```
long n = 3;
in.seek((n - 1) * RECORD_SIZE);
Employee e = new Employee();
e.readData(in);
```

如果你希望修改记录，然后将其存回到相同的位置，那么就需要记住将文件指针置回到这条记录的开始处：

```
in.seek((n - 1) * RECORD_SIZE);
e.writeData(out);
```

要确定文件中的字节总数，可以使用`length`方法，而记录的总数则是用字节总数除以每条记录的大小。

```
long nbytes = in.length(); // length in bytes
int nrecords = (int) (nbytes / RECORD_SIZE);
```

整数和浮点值在二进制格式中都具有固定的尺寸，但是在处理字符串时就有些麻烦了，因此我们提供了两个方法来读写具有固定尺寸的字符串。

`writeFixedString`写出从字符串开头开始的指定数量的码元（如果码元过少，该方法将用

0值来补齐字符串)。

```
public static void writeFixedString(String s, int size, DataOutput out)
    throws IOException
{
    for (int i = 0; i < size; i++)
    {
        char ch = 0;
        if (i < s.length()) ch = s.charAt(i);
        out.writeChar(ch);
    }
}
```

`readFixedString`方法从输入流中读入字符，直至读入size个码元，或者直至遇到0值，然后跳过输入域中剩余的0值。为了提高效率，这个方法使用了`StringBuilder`类来读入字符串。

```
public static String readFixedString(int size, DataInput in)
    throws IOException
{
    StringBuilder b = new StringBuilder(size);
    int i = 0;
    boolean more = true;
    while (more && i < size)
    {
        char ch = in.readChar();
        i++;
        if (ch == 0) more = false;
        else b.append(ch);
    }
    in.skipBytes(2 * (size - i));
    return b.toString();
}
```

我们将`writeFixedString`和`readFixedString`方法放到了`DataIO`助手类的内部。

为了写出一条固定尺寸的记录，我们直接以二进制方式写出所有的域：

```
public void writeData(DataOutput out) throws IOException
{
    DataIO.writeFixedString(name, NAME_SIZE, out);
    out.writeDouble(salary);
    GregorianCalendar calendar = new GregorianCalendar();
    calendar.setTime(hireDay);
    out.writeInt(calendar.get(Calendar.YEAR));
    out.writeInt(calendar.get(Calendar.MONTH) + 1);
    out.writeInt(calendar.get(Calendar.DAY_OF_MONTH));
}
```

读回数据也很简单：

```
public void readData(DataInput in) throws IOException
{
    name = DataIO.readFixedString(NAME_SIZE, in);
    salary = in.readDouble();
    int y = in.readInt();
    int m = in.readInt();
    int d = in.readInt();
    GregorianCalendar calendar = new GregorianCalendar(y, m - 1, d);
```

```
    hireDay = calendar.getTime();
}
```

让我们来计算每条记录的大小：我们将使用40个字符来表示姓名字符串，因此，每条记录包含100个字节：

- 40字符 = 80 字节，用于姓名。
- 1 double = 8 字节，用于薪水。
- 3 int = 12字节，用于日期。

程序清单1-2中所示的程序将三条记录写到了一个数据文件中，然后以逆序将它们从文件中读回。为了高效地执行，这里需要使用随机访问，因为我们需要首先读入第三条记录。

### 程序清单1-2 RandomFileTest.java

```
1. import java.io.*;
2. import java.util.*;
3.
4. /**
5. * @version 1.11 2004-05-11
6. * @author Cay Horstmann
7. */
8.
9. public class RandomFileTest
10. {
11.     public static void main(String[] args)
12.     {
13.         Employee[] staff = new Employee[3];
14.
15.         staff[0] = new Employee("Carl Cracker", 75000, 1987, 12, 15);
16.         staff[1] = new Employee("Harry Hacker", 50000, 1989, 10, 1);
17.         staff[2] = new Employee("Tony Tester", 40000, 1990, 3, 15);
18.
19.         try
20.         {
21.             // save all employee records to the file employee.dat
22.             DataOutputStream out = new DataOutputStream(new FileOutputStream("employee.dat"));
23.             for (Employee e : staff)
24.                 e.writeData(out);
25.             out.close();
26.
27.             // retrieve all records into a new array
28.             RandomAccessFile in = new RandomAccessFile("employee.dat", "r");
29.             // compute the array size
30.             int n = (int)(in.length() / Employee.RECORD_SIZE);
31.             Employee[] newStaff = new Employee[n];
32.
33.             // read employees in reverse order
34.             for (int i = n - 1; i >= 0; i--)
35.             {
36.                 newStaff[i] = new Employee();
37.                 in.seek(i * Employee.RECORD_SIZE);
38.                 newStaff[i].readData(in);
39.             }
40.             in.close();
41.         }
```

```
42     // print the newly read employee records
43     for (Employee e : newStaff)
44         System.out.println(e);
45     }
46     catch (IOException e)
47     {
48         e.printStackTrace();
49     }
50 }
51 }
52
53 class Employee
54 {
55     public Employee() {}
56
57     public Employee(String n, double s, int year, int month, int day)
58     {
59         name = n;
60         salary = s;
61         GregorianCalendar calendar = new GregorianCalendar(year, month - 1, day);
62         hireDay = calendar.getTime();
63     }
64
65     public String getName()
66     {
67         return name;
68     }
69
70     public double getSalary()
71     {
72         return salary;
73     }
74
75     public Date getHireDay()
76     {
77         return hireDay;
78     }
79
80     /**
81      * Raises the salary of this employee.
82      * @byPercent the percentage of the raise
83      */
84     public void raiseSalary(double byPercent)
85     {
86         double raise = salary * byPercent / 100;
87         salary += raise;
88     }
89
90     public String toString()
91     {
92         return getClass().getName()
93             + "[name=" + name
94             + ",salary=" + salary
95             + ",hireDay=" + hireDay
96             + "]";
97     }
```

```
98.  
99.     /**  
100.        Writes employee data to a data output  
101.        @param out the data output  
102.     */  
103.    public void writeData(DataOutput out) throws IOException  
104.    {  
105.        DataIO.writeFixedString(name, NAME_SIZE, out);  
106.        out.writeDouble(salary);  
107.  
108.        GregorianCalendar calendar = new GregorianCalendar();  
109.        calendar.setTime(hireDay);  
110.        out.writeInt(calendar.get(Calendar.YEAR));  
111.        out.writeInt(calendar.get(Calendar.MONTH) + 1);  
112.        out.writeInt(calendar.get(Calendar.DAY_OF_MONTH));  
113.    }  
114.  
115.    /**  
116.        Reads employee data from a data input  
117.        @param in the data input  
118.     */  
119.    public void readData(DataInput in) throws IOException  
120.    {  
121.        name = DataIO.readFixedString(NAME_SIZE, in);  
122.        salary = in.readDouble();  
123.        int y = in.readInt();  
124.        int m = in.readInt();  
125.        int d = in.readInt();  
126.        GregorianCalendar calendar = new GregorianCalendar(y, m - 1, d);  
127.        hireDay = calendar.getTime();  
128.    }  
129.  
130.    public static final int NAME_SIZE = 40;  
131.    public static final int RECORD_SIZE = 2 * NAME_SIZE + 8 + 4 + 4 + 4;  
132.  
133.    private String name;  
134.    private double salary;  
135.    private Date hireDay;  
136.}  
137.  
138. class DataIO  
139. {  
140.    public static String readFixedString(int size, DataInput in)  
141.        throws IOException  
142.    {  
143.        StringBuilder b = new StringBuilder(size);  
144.        int i = 0;  
145.        boolean more = true;  
146.        while (more && i < size)  
147.        {  
148.            char ch = in.readChar();  
149.            i++;  
150.            if (ch == 0) more = false;  
151.            else b.append(ch);  
152.        }  
153.        in.skipBytes(2 * (size - i));  
154.        return b.toString();  
}
```

```

155.    }
156.
157.    public static void writeFixedString(String s, int size, DataOutput out)
158.        throws IOException
159.    {
160.        for (int i = 0; i < size; i++)
161.        {
162.            char ch = 0;
163.            if (i < s.length()) ch = s.charAt(i);
164.            out.writeChar(ch);
165.        }
166.    }
167.}
```

### **API** java.io.RandomAccessFile 1.0

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

参数: file 要打开的文件

mode “r” 表示只读模式; “rw” 表示读/写模式; “rws” 表示每次更新时, 都对数据和元数据的写磁盘操作进行同步的读/写模式; “rwd” 表示每次更新时, 只对数据的写磁盘操作进行同步的读/写模式

- long getFilePointer()

返回文件指针的当前位置。

- void seek(long pos)

将文件指针从文件的开始设置到pos个字节处。

- long length()

返回文件按照字节来度量的长度。

## 1.4 ZIP文档

ZIP文档(通常)以压缩格式存储了一个或多个文件, 每个ZIP文档都有一个包含诸如文件名字和所使用的压缩方法等信息的头。在Java中, 可以使用ZipInputStream来读入ZIP文档。你可能需要浏览文档中每个单独的项, getNextEntry方法就可以返回一个描述这些项的ZipEntry类型的对象。ZipInputStream的read方法被修改为在碰到当前项的结尾时返回-1(而不是碰到ZIP文件的末尾), 然后你必须调用closeEntry来读入下一项。下面是典型的通读ZIP文件的代码序列:

```

ZipInputStream zin = new ZipInputStream(new FileInputStream(zipname));
ZipEntry entry;
while ((entry = zin.getNextEntry()) != null)
{
    analyze entry;
    read the contents of zin;
    zin.closeEntry();
}
zin.close();
```

当希望读入某个ZIP项的内容时，我们可能并不想使用原生的read方法，通常，我们将使用某个更能胜任的流过滤器的方法。例如，为了读入ZIP文件内部的一个文本文件，我们可以使用下面的循环：

```
Scanner in = new Scanner(zin);
while (in.hasNextLine())
    do something with in.nextLine();
```

 注意：ZIP输入流在读入ZIP文件发生错误时，会抛出ZipException。通常这种错误在ZIP文件被破坏时发生。

要写出到ZIP文件，可以使用ZipOutputStream，而对于你希望放入到ZIP文件中的每一项，都应该创建一个ZipEntry对象，并将文件名传递给ZipEntry的构造器，它将设置其他诸如文件日期和解压缩方法等参数。如果需要，你可以覆盖这些设置。然后，你需要调用ZipOutputStream的putNextEntry方法来开始写出新文件，并将文件数据发送到ZIP流中。当完成时，需要调用closeEntry。然后，你需要对所有你希望存储的文件都重复这个过程。下面是代码框架：

```
FileOutputStream fout = new FileOutputStream("test.zip");
ZipOutputStream zout = new ZipOutputStream(fout);
for all files
{
    ZipEntry ze = new ZipEntry(filename);
    zout.putNextEntry(ze);
    send data to zout;
    zout.closeEntry();
}
zout.close();
```

 注意：JAR文件（在第I卷第10章中讨论过）只是带有另一个项的ZIP文件，这个项称作清单。你可以使用JarInputStream和JarOutputStream类来读写清单项。

ZIP流是一个能够展示流的抽象化的强大之处的实例。当你读入以压缩格式存储的数据时，不必操心边请求边解压数据。而且ZIP格式的字节源并非必须是文件，也可以是来自网络连接的ZIP数据。事实上，只要Applet的类加载器能够读入JAR文件，那么它就能够读入和解压来自网络的数据。

 注意：在<http://www.javaworld.com/javaworld/jw-10-2000/jw-1027-toolbox.html>处的文章说明了如何修改ZIP文档。

程序清单1-3所示的程序可以让我们打开一个ZIP文件，它还在屏幕底部的一个组合框中显示了存储在ZIP文档中的文件。如果你选中这些文件中的某一个，那么这个文件的内容就会显示在文本域中，如图1-5所示。

### 程序清单1-3 ZipTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.io.*;
```

```
4. import java.util.*;
5. import java.util.List;
6. import java.util.zip.*;
7. import javax.swing.*;
8.
9. /**
10. * @version 1.32 2007-06-22
11. * @author Cay Horstmann
12. */
13. public class ZipTest
14. {
15.     public static void main(String[] args)
16.     {
17.         EventQueue.invokeLater(new Runnable()
18.         {
19.             public void run()
20.             {
21.                 ZipTestFrame frame = new ZipTestFrame();
22.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23.                 frame.setVisible(true);
24.             }
25.         });
26.     }
27. }
28.
29. /**
30. * A frame with a text area to show the contents of a file inside a ZIP archive, a combo
31. * box to select different files in the archive, and a menu to load a new archive.
32. */
33. class ZipTestFrame extends JFrame
34. {
35.     public ZipTestFrame()
36.     {
37.         setTitle("ZipTest");
38.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
39.
40.         // add the menu and the Open and Exit menu items
41.         JMenuBar menuBar = new JMenuBar();
42.         JMenu menu = new JMenu("File");
43.
44.         JMenuItem openItem = new JMenuItem("Open");
45.         menu.add(openItem);
46.         openItem.addActionListener(new ActionListener()
47.         {
48.             public void actionPerformed(ActionEvent event)
49.             {
50.                 JFileChooser chooser = new JFileChooser();
51.                 chooser.setCurrentDirectory(new File("."));
52.                 int r = chooser.showOpenDialog(ZipTestFrame.this);
53.                 if (r == JFileChooser.APPROVE_OPTION)
54.                 {
55.                     zipname = chooser.getSelectedFile().getPath();
56.                     fileCombo.removeAllItems();
57.                     scanZipFile();
58.                 }
59.             }
59.         });
59.     }
59.
```

```
60     });
61
62     JMenuItem exitItem = new JMenuItem("Exit");
63     menu.add(exitItem);
64     exitItem.addActionListener(new ActionListener()
65     {
66         public void actionPerformed(ActionEvent event)
67         {
68             System.exit(0);
69         }
70     });
71
72     menuBar.add(menu);
73     setJMenuBar(menuBar);
74
75     // add the text area and combo box
76     fileText = new JTextArea();
77     fileCombo = new JComboBox();
78     fileCombo.addActionListener(new ActionListener()
79     {
80         public void actionPerformed(ActionEvent event)
81         {
82             loadZipFile((String) fileCombo.getSelectedItem());
83         }
84     });
85
86     add(fileCombo, BorderLayout.SOUTH);
87     add(new JScrollPane(fileText), BorderLayout.CENTER);
88 }
89
90 /**
91 * Scans the contents of the ZIP archive and populates the combo box.
92 */
93 public void scanZipFile()
94 {
95     new SwingWorker<Void, String>()
96     {
97         protected Void doInBackground() throws Exception
98         {
99             ZipInputStream zin = new ZipInputStream(new FileInputStream(zipname));
100            ZipEntry entry;
101            while ((entry = zin.getNextEntry()) != null)
102            {
103                publish(entry.getName());
104                zin.closeEntry();
105            }
106            zin.close();
107            return null;
108        }
109
110        protected void process(List<String> names)
111        {
112            for (String name : names)
113                fileCombo.addItem(name);
114        }
115    }.execute();
```

```
117.    }
118.
119.    /**
120.     * Loads a file from the ZIP archive into the text area
121.     * @param name the name of the file in the archive
122.     */
123.    public void loadZipFile(final String name)
124.    {
125.        fileCombo.setEnabled(false);
126.        fileText.setText("");
127.        new SwingWorker<Void, Void>()
128.        {
129.            protected Void doInBackground() throws Exception
130.            {
131.                try
132.                {
133.                    ZipInputStream zin = new ZipInputStream(new FileInputStream(zipname));
134.                    ZipEntry entry;
135.
136.                    // find entry with matching name in archive
137.                    while ((entry = zin.getNextEntry()) != null)
138.                    {
139.                        if (entry.getName().equals(name))
140.                        {
141.                            // read entry into text area
142.                            Scanner in = new Scanner(zin);
143.                            while (in.hasNextLine())
144.                            {
145.                                fileText.append(in.nextLine());
146.                                fileText.append("\n");
147.                            }
148.                        }
149.                        zin.closeEntry();
150.                    }
151.                    zin.close();
152.                }
153.                catch (IOException e)
154.                {
155.                    e.printStackTrace();
156.                }
157.                return null;
158.            }
159.
160.            protected void done()
161.            {
162.                fileCombo.setEnabled(true);
163.            }
164.        }.execute();
165.    }
166.
167.    public static final int DEFAULT_WIDTH = 400;
168.    public static final int DEFAULT_HEIGHT = 300;
169.    private JComboBox fileCombo;
170.    private JTextArea fileText;
171.    private String zipname;
172. }
```

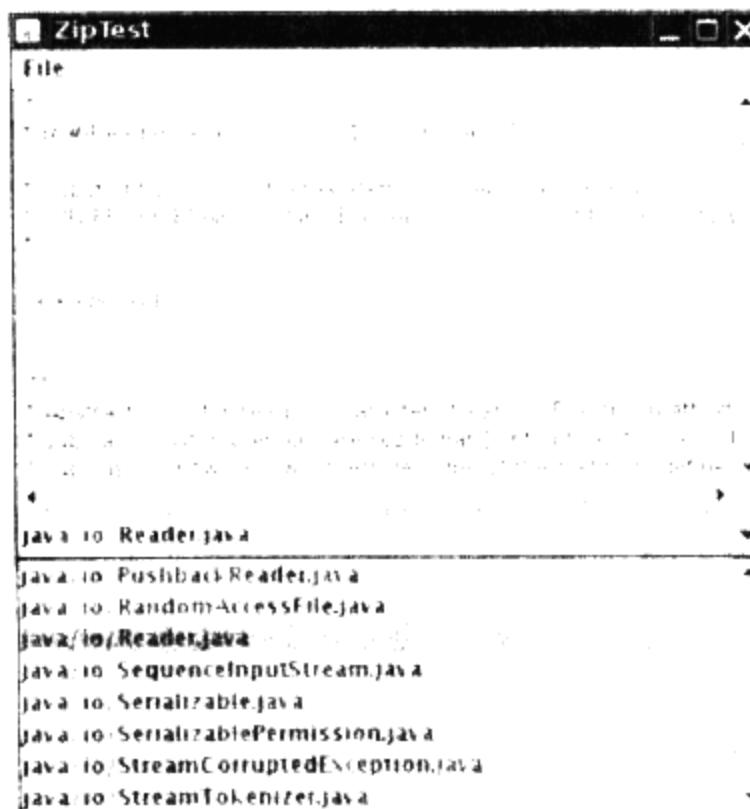


图1-5 ZipTest程序

**API** `java.util.zip.ZipInputStream 1.1`

- `ZipInputStream(InputStream in)`

创建一个ZipInputStream，使得我们可以从给定的InputStream向其中填充数据。

- `ZipEntry getNextEntry()`

为下一项返回ZipEntry对象，否则没有更多的项时返回null。

- `void closeEntry()`

关闭这个ZIP文件中当前打开的项。之后可以通过使用getNextEntry()读入下一项。

**API** `java.util.zip.ZipOutputStream 1.1`

- `ZipOutputStream(OutputStream out)`

创建一个将压缩数据写出到指定的OutputStream的ZipOutputStream。

- `void putNextEntry(ZipEntry ze)`

将给定的ZipEntry中的信息写出到流中，并定为用于写出数据的流；然后这些数据可以通过write()写出到这个流中。

- `void closeEntry()`

关闭这个ZIP文件中当前打开的项。使用putNextEntry方法可以开始下一项。

- `void setLevel(int level)`

设置后续的各个DEFLATED项的默认压缩级别。这里默认值是Deflater.DEFAULT\_COMPRESSION。如果级别无效，则抛出IllegalArgumentException。

参数：level 压缩级别，从0(NO\_COMPRESSION)到9(BEST\_COMPRESSION)

- `void setMethod(int method)`

设置用于这个ZipOutputStream的默认压缩方法，这个压缩方法会作用于所有没有指定压缩方法的项上。

参数：method 压缩方法，DEFLATED或STORED

#### API java.util.zip.ZipEntry 1.1

- ZipEntry(String name)

参数：name 这一项的名字

- long getCrc()

返回用于这个ZipEntry的CRC32校验和的值。

- String getName()

返回这一项的名字。

- long getSize()

返回这一项不被压缩的大小，或者在不被压缩的大小不可知的情况下返回-1。

- boolean isDirectory()

当这一项是目录时返回true。

- void setMethod(int method)

参数：method 用于这一项的压缩方法，必须是DEFLATED或STORED

- void setSize(long size)

设置这一项的大小，只有在压缩方法是STORED时才是必需的。

参数：size 这一项不被压缩的大小

- void setCrc(long crc)

给这一项设置CRC32校验和，这个校验和是使用CRC32类计算的。只有在压缩方法是STORED时才是必需的。

参数：crc 这一项的校验和

#### API java.util.zip.ZipFile 1.1

- ZipFile(String name)

- ZipFile(File file)

创建一个ZipFile，用于从给定的字符串或File对象中读入数据。

- Enumeration entries()

返回一个Enumeration对象，它枚举了描述这个ZipFile中各个项的ZipEntry对象。

- ZipEntry getEntry(String name)

返回给定名字所对应的项，或者在没有对应项的时候返回null。

参数：name 项名

- InputStream getInputStream(ZipEntry ze)

返回用于给定项的InputStream。

参数：ze 这个ZIP文件中的一个ZipEntry

- `String getName()`  
返回这个ZIP文件的路径。

## 1.5 对象流与序列化

当你需要存储相同类型的数据时，使用固定长度的记录格式是一个不错的选择。但是，在面向对象程序中创建的对象很少全部都具有相同的类型。例如，你可能有一个称为`staff`的数组，它名义上是一个`Employee`记录数组，但是实际上却包含诸如`Manager`这样的子类实例。

我们当然可以自己设计出一种数据格式来存储这种多态集合，但是幸运的是，我们并不需要这么做。Java语言支持一种称为对象序列化（object serialization）的非常通用的机制，它可以将任何对象写出到流中，并在之后将其读回。（你将在本章稍后看到“序列化”这个术语的出处。）

为了保存对象数据，首先需要打开一个`ObjectOutputStream`对象：

```
ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("employee.dat"));
```

现在，为了保存对象，我们可以像下面的代码段一样直接使用`ObjectOutputStream`的`writeObject`方法：

```
Employee harry = new Employee("Harry Hacker", 50000, 1989, 10, 1);
Manager boss = new Manager("Carl Cracker", 80000, 1987, 12, 15);
out.writeObject(harry);
out.writeObject(boss);
```

为了将这些对象读回，首选需要获得一个`ObjectInputStream`对象：

```
ObjectInputStream in = new ObjectInputStream(new FileInputStream("employee.dat"));
```

然后，用`readObject`方法以这些对象被写出时的顺序获得它们：

```
Employee e1 = (Employee) in.readObject();
Employee e2 = (Employee) in.readObject();
```

但是，你必须对希望在对象流中存储或恢复的所有类都进行一下修改，这些类必须实现`Serializable`接口：

```
class Employee implements Serializable { . . . }
```

`Serializable`接口没有任何方法，因此你不需要对这些做任何改动。在这一点上，它与在第I卷第6章中讨论过的`Cloneable`接口很相似。但是，为了使类可克隆，你仍旧需要覆盖`Object`类中的`clone`方法，而为了使类可序列化，你不需要做任何事。

 **注意：**你只有在写出对象时才能用`writeObject/readObject`方法，对于基本类型值，你需要使用诸如`writeInt/readInt`或`writeDouble/readDouble`这样的方法。（对象流类都实现了`DataInput/DataOutput`接口。）

表面上，是`ObjectOutputStream`在浏览对象的所有域，并存储它们的内容。例如，当写出一个`Employee`对象时，其名字、日期和薪水域都会被写出到输出流中。

但是，有一种重要的情况需要考虑：当一个对象被多个对象共享，作为它们各自状态的一

部分时，会发生什么呢？

为了说明这个问题，我们对Manager类稍微做些修改，假设每个经理都有一个秘书：

```
class Manager extends Employee
{
    ...
    private Employee secretary;
}
```

现在每个Manager对象都包含一个对描述秘书的Employee对象的引用，当然，两个经理可以共用一个秘书，正如图1-6和下面的代码所示的那样：

```
harry = new Employee("Harry Hacker", ...);
Manager carl = new Manager("Carl Cracker", ...);
carl.setSecretary(harry);
Manager tony = new Manager("Tony Tester", ...);
tony.setSecretary(harry);
```

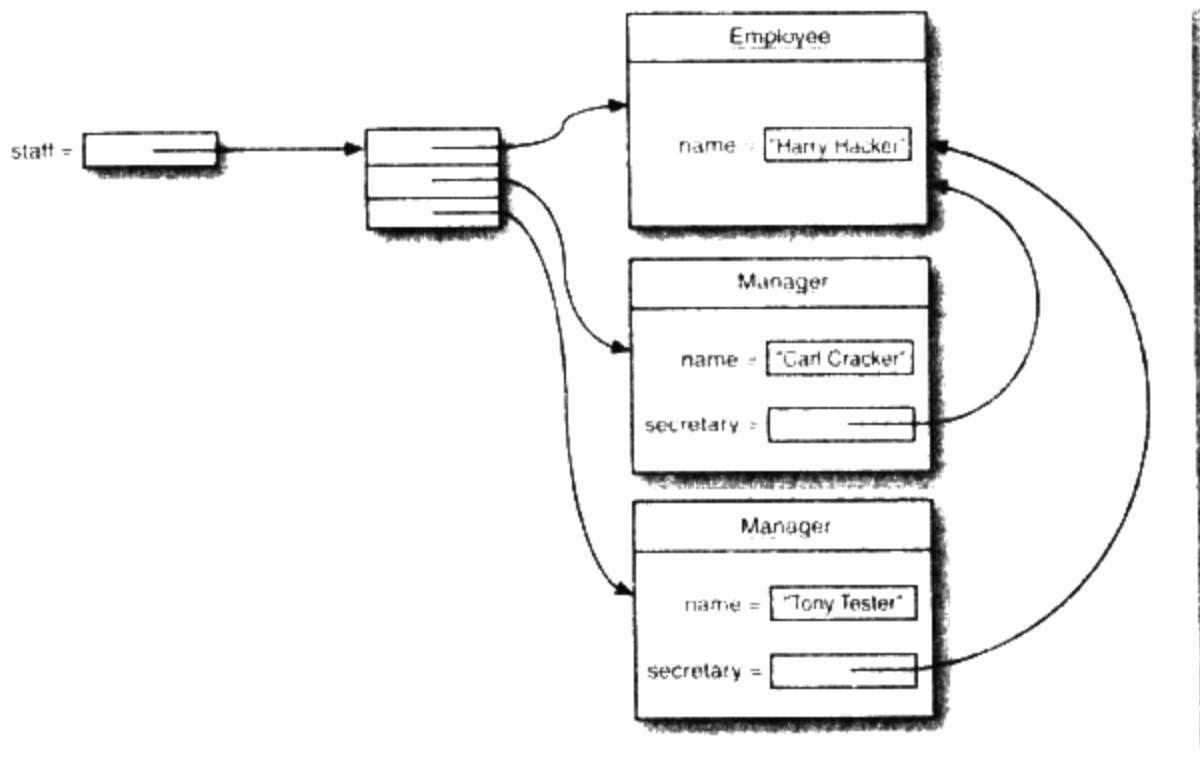


图1-6 两个经理可以共用一个共有的雇员

保存这样的对象网路是一种挑战，在这里我们当然不能去保存和恢复秘书对象的地址，因为当对象被重新加载时，它可能占据的是与原来完全不同的内存地址。

与此不同的是，每个对象都是用一个序列号（serial number）保存的，这就是这种机制之所以称为对象序列化的原因。下面是其算法：

- 对你遇到的每一个对象引用都关联一个序列号（如图1-7所示）。
- 对于每个对象，当第一次遇到时，保存其对象数据到流中。
- 如果某个对象之前已经被保存过，那么只写出“与之前保存过的序列号为x的对象相同”。在读回对象时，整个过程是反过来的。
- 对于流中的对象，在第一次遇到其序列号时，构建它，并使用流中数据来初始化它，然

后记录这个顺序号和新对象之间的关联。

- 当遇到“与之前保存过的序列号为x的对象相同”标记时，获取与这个顺序号相关联的对象引用。

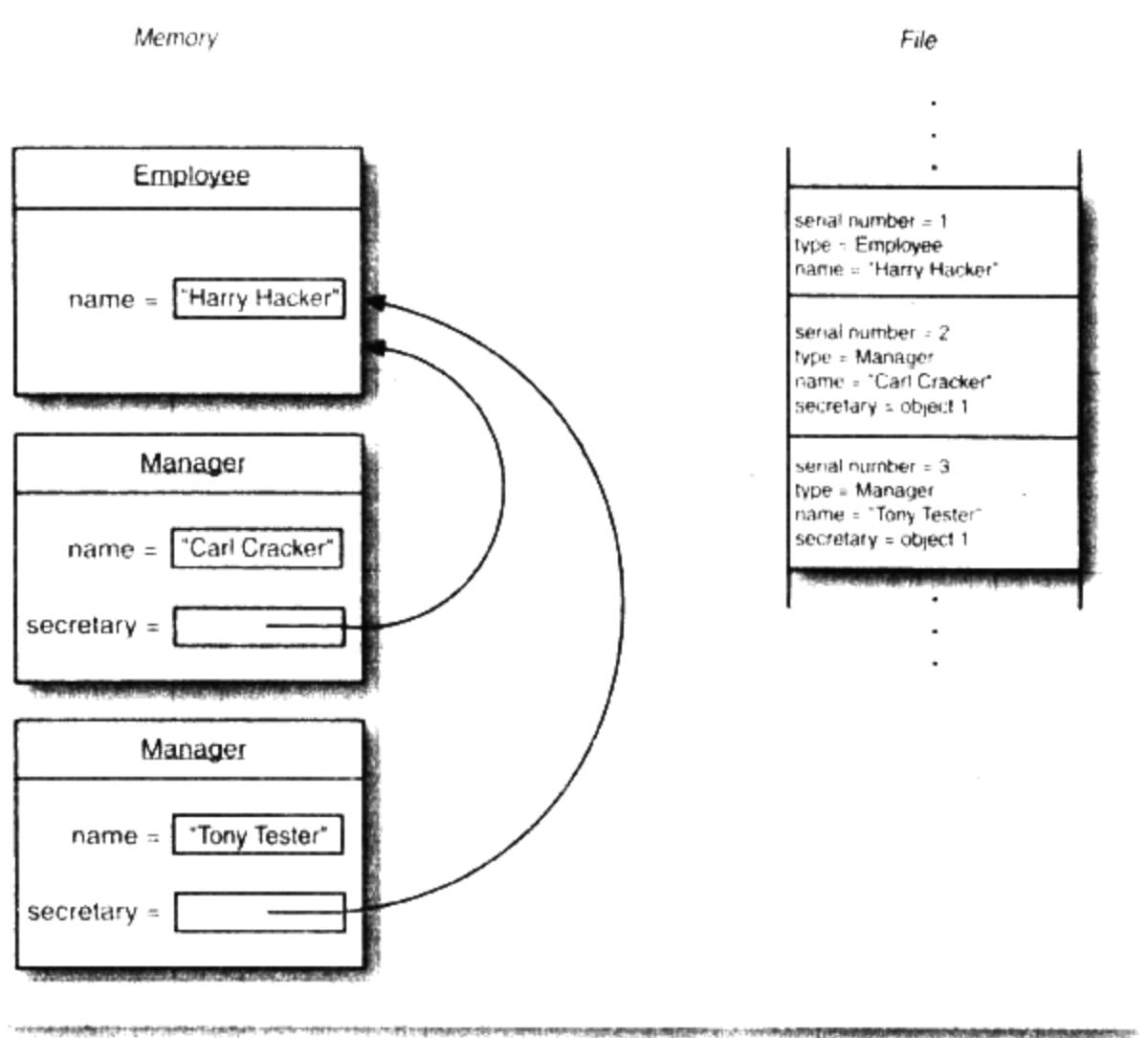


图1-7 一个对象序列化的实例



**注意：**在本章中，我们使用序列化将对象集合保存到磁盘文件中，并按照它们被存储的样子获取它们。序列化的另一种非常重要的应用是通过网络将对象集合传送到另一台计算机上。正如在文件中保存原生的内存地址毫无意义一样，它们对于在不同的处理器之间的通信也是毫无意义的。因为序列化用序列号代替了内存地址，所以它允许将对象集合从一台机器传送到另一台机器。我们将在第5章讨论远程方法调用时学习序列化的这种用法。

程序清单1-4是保存和重新加载Employee和Manager对象网络的代码（有些对象共享相同的表示秘书的雇员）。注意，秘书对象在重新加载之后是惟一的，当newStaff[1]被恢复时，它会反映到经理们的secretary域中。

#### 程序清单1-4 ObjectStreamTest.java

```

1. import java.io.*;
2. import java.util.*;
3.

```

```
4. /**
5. * @version 1.10 17 Aug 1998
6. * @author Cay Horstmann
7. */
8. class ObjectStreamTest
9. {
10.     public static void main(String[] args)
11.     {
12.         Employee harry = new Employee("Harry Hacker", 50000, 1989, 10, 1);
13.         Manager carl = new Manager("Carl Cracker", 80000, 1987, 12, 15);
14.         carl.setSecretary(harry);
15.         Manager tony = new Manager("Tony Tester", 40000, 1990, 3, 15);
16.         tony.setSecretary(harry);
17.
18.         Employee[] staff = new Employee[3];
19.
20.         staff[0] = carl;
21.         staff[1] = harry;
22.         staff[2] = tony;
23.
24.         try
25.         {
26.             // save all employee records to the file employee.dat
27.             ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("employee.dat"));
28.             out.writeObject(staff);
29.             out.close();
30.
31.             // retrieve all records into a new array
32.             ObjectInputStream in = new ObjectInputStream(new FileInputStream("employee.dat"));
33.             Employee[] newStaff = (Employee[]) in.readObject();
34.             in.close();
35.
36.             // raise secretary's salary
37.             newStaff[1].raiseSalary(10);
38.
39.             // print the newly read employee records
40.             for (Employee e : newStaff)
41.                 System.out.println(e);
42.         }
43.         catch (Exception e)
44.         {
45.             e.printStackTrace();
46.         }
47.     }
48. }
49.
50. class Employee implements Serializable
51. {
52.     public Employee()
53.     {
54.     }
55.
56.     public Employee(String n, double s, int year, int month, int day)
57.     {
58.         name = n;
59.         salary = s;
```

```
60     GregorianCalendar calendar = new GregorianCalendar(year, month - 1, day);
61     hireDay = calendar.getTime();
62 }
63
64 public String getName()
65 {
66     return name;
67 }
68
69 public double getSalary()
70 {
71     return salary;
72 }
73
74 public Date getHireDay()
75 {
76     return hireDay;
77 }
78
79 public void raiseSalary(double byPercent)
80 {
81     double raise = salary * byPercent / 100;
82     salary += raise;
83 }
84
85 public String toString()
86 {
87     return getClass().getName() + "[name=" + name + ",salary=" + salary + ",hireDay="
88         + hireDay + "]";
89 }
90
91 private String name;
92 private double salary;
93 private Date hireDay;
94 }
95
96 class Manager extends Employee
97 {
98     /**
99      * Constructs a Manager without a secretary
100     * @param n the employee's name
101     * @param s the salary
102     * @param year the hire year
103     * @param month the hire month
104     * @param day the hire day
105     */
106     public Manager(String n, double s, int year, int month, int day)
107     {
108         super(n, s, year, month, day);
109         secretary = null;
110     }
111
112     /**
113      * Assigns a secretary to the manager
114      * @param s the secretary
115      */
```

```

116. public void setSecretary(Employee s)
117. {
118.     secretary = s;
119. }
120.
121. public String toString()
122. {
123.     return super.toString() + "[secretary=" + secretary + "]";
124. }
125.
126. private Employee secretary;
127. }

```

### **API** `java.io.ObjectOutputStream` 1.1

- `ObjectOutputStream(OutputStream out)`

创建一个`ObjectOutputStream`使得你可以将对象写出到指定的`OutputStream`。

- `void writeObject(Object obj)`

写出指定的对象到`ObjectOutputStream`，这个方法将存储指定对象的类、类的签名以及这个类及其超类中所有非静态和非瞬时的域的值。

### **API** `java.io.ObjectInputStream` 1.1

- `ObjectInputStream(InputStream in)`

创建一个`ObjectInputStream`用于从指定的`InputStream`中读回对象信息。

- `Object readObject()`

从`ObjectInputStream`中读入一个对象。特别是，这个方法会读回对象的类、类的签名以及这个类及其超类中所有非静态和非瞬时的域的值。它执行的反序列化允许恢复多个对象引用。

#### 1.5.1 理解对象序列化的文件格式

对象序列化是以特殊的文件格式存储对象数据的，当然，你不必了解文件中表示对象的确切的字节序列，就可以使用`writeObject/readObject`方法。但是，我们发现研究这种数据格式对于洞察对象流化的处理过程非常有益。因为其细节显得有些专业，所以如果你对其实现不感兴趣，则可以跳过这一节。

每个文件都是以下面这两个字节的“魔幻数字”开始的

AC ED

后面紧跟着对象序列化格式的版本号，目前是

00 05

(我们在本节中统一使用十六进制数字来表示字节。)然后，是它包含的对象序列，其顺序即它们存储的顺序。

字符串对象被存为

74 两字节表示的字符串长度 所有字符

例如，字符串“Harry”被存为

74 00 05 Harry

字符串中的Unicode字符被存储为修订过的UTF-8格式。

当存储一个对象时，这个对象所属的类也必须存储。这个类的描述包含

- 类名。
- 序列化的版本惟一的ID，它是数据域类型和方法签名的指纹。
- 描述序列化方法的标志集。
- 对数据域的描述。

指纹是通过对类、超类、接口、域类型和方法签名按照规范方式排序，然后将安全散列算法（SHA）应用于这些数据而获得的。

SHA是一种可以为较大的信息块提供指纹的快速算法，不论最初的数据块尺寸有多大，这种指纹总是20个字节的数据包。它是通过在数据上执行一个灵巧的位操作序列而创建的，这个序列在本质上可以百分之百地保证无论这些数据以何种方式发生变化，其指纹也都会跟着变化。（关于SHA的更多细节，可以查看一些参考资料，例如William Stallings所著的《Cryptography and Network Security: Principles and Practice》[Prentice Hall, 2002]。）但是，序列化机制只使用了SHA码的前8个字节作为类的指纹。即便这样，当类的数据域或方法发生变化时，其指纹跟着变化的可能性还是非常大。

在读入一个对象时，会拿其指纹与它所属的类的当前指纹进行比对，如果它们不匹配，那么就说明这个类的定义在该对象被写出之后发生过变化，因此会产生一个异常。在实际情况下，类当然是会演化的，因此对于程序来说，读入较旧版本的对象可能是必需的。我们将在第1.5.4节中讨论这个问题。

下面表示了类标识符是如何存储的：

72

2字节长的类名

类名

8字节长的指纹

1字节长的标志

2字节长的数据域描述符的计数值

数据域描述符

78（结束标记）

超类类型（如果没有就是70）

标志字节是由在java.io.ObjectStreamConstants中定义的3位掩码构成的：

```
static final byte SC_WRITE_METHOD = 1;
    // class has writeObject method that writes additional data
static final byte SC_SERIALIZABLE = 2;
    // class implements Serializable interface
static final byte SC_EXTERNALIZABLE = 4;
    // class implements Externalizable interface
```

我们会在本章稍后讨论Externalizable接口。可外部化的类像客户提供了可以导出其实例

域的输出的读写方法，我们要写出的这些类实现了`Serializable`接口，并且其标志值为02，而可序列化的`java.util.Date`类定义了它自己的`readObject/writeObject`方法，并且其标志值为03。

每个数据域描述符的格式如下：

1字节长的类型编码

2字节长的域名

域名

类名（如果域是对象）

其中类型编码是下列取值之一：

B byte

C char

D double

F float

I int

J long

L 对象

S short

Z boolean

[ 数组

当类型编码为L时，域名后面紧跟域的类型。类名和域名字符串不是以字符串编码74开头的，而域类型却是。域类型使用的是与域名稍有不同的编码机制，即本地方法使用的格式。

例如，`Employee`类的薪水域被编码为：

D 00 06 salary

下面是`Employee`类完整的类描述符：

72 00 08 Employee

|                             |                              |
|-----------------------------|------------------------------|
| E6 D2 86 7D AE AC 18 1B 02  | 指纹和标志                        |
| 00 03                       | 实例域的数量                       |
| D 00 06 salary              | 实例域的类型和名字                    |
| L 00 07 hireDay             | 实例域的类型和名字                    |
| 74 00 10 Ljava/util/Date;   | 实例域的类名—— <code>Date</code>   |
| L 00 04 name                | 实例域的类型和名字                    |
| 74 00 12 Ljava/lang/String; | 实例域的类名—— <code>String</code> |
| 78                          | 结束标记                         |
| 70                          | 无超类                          |

这些描述符相当长，如果在文件中再次需要相同的类描述符，可以使用一种缩写版：

71 4字节长的序列号

这个序列号将引用到前面已经描述过的类描述符，我们稍后将讨论编号模式。

对象将被存储为：

73 类描述符 对象数据

例如，下面展示的就是Employee对象如何存储：

|                                  |                    |
|----------------------------------|--------------------|
| 40 E8 6A 00 00 00 00 00          | salary域的值——double  |
| 73                               | hireDay域的值——新对象    |
| 71 00 7E 00 08                   | 已有的类Java.util.Date |
| 77 08 00 00 00 91 1B 4E B1 80 78 | 外部存储——稍后讨论细节       |
| 74 00 0C Harry Hacker            | name域的值——String    |

正如你所看见的，数据文件包含了足够的信息来恢复这个Employee对象。

数组总是被存储成下面的格式：

75      类描述符      4字节长的数组项的数量      数组项

在类描述符中的数组类名的格式与本地方法中使用的格式相同（它与在其他类的描述符中的类名稍微有些差异）。在这种格式中，类名以L开头，以分号结束。

例如，3个Employee对象构成的数组写出时就像下面一样：

|                            |                       |
|----------------------------|-----------------------|
| 75                         | 数组                    |
| 72 00 0B [LEmployee;       | 新类、字符串长度、类名Employee[] |
| FC BF 36 11 C5 91 11 C7 02 | 指纹和标志                 |
| 00 00                      | 实例域的数量                |
| 78                         | 结束标记                  |
| 70                         | 无超类                   |
| 00 00 00 03                | 数组项的数量                |

注意，Employee对象数组的指纹与Employee类自身的指纹并不相同。

所有对象（包含数组和字符串）和所有类的描述符在存储到输出文件时都被赋予了一个序列号，这个数字以00 7E 00 00开头。

我们已经看到过，任何给定的类其完整的类描述符只保存一次，后续的描述符将引用它。例如，在前面的示例中，对Date类的重复引用就被编码为：

71 00 7E 00 08

相同的机制还被用于对象。如果要写出一个对之前存储过的对象的引用，那么这个引用就会以完全相同的方式存储，即，71后面跟随序列号，从上下文中可以很清楚地了解这个特殊的序列引用表示的是类描述符还是对象。

最后，空引用被存储为：

70

下面是前面小节中ObjectRefTest程序的带注释的输出。如果你喜欢，可以运行这个程序，然后查看其数据文件employee.dat的十六进制码，并将其与注释列表比较。在输出中接近结束部分的几行重要编码展示了对之前存储过的对象的引用。

|                            |                              |
|----------------------------|------------------------------|
| AC ED 00 05                | 文件头                          |
| 75                         | 数组staff (序列#1)               |
| 72 00 0B [LEmployee;       | 新类、字符串长度、类名Employee[] (序列#0) |
| FC BF 36 11 C5 91 11 C7 02 | 指纹和标志                        |

|                             |                          |
|-----------------------------|--------------------------|
| 00 00                       | 实例域的数量                   |
| 78                          | 结束标记                     |
| 70                          | 无超类                      |
| 00 00 00 03                 | 数组项的数量                   |
| 73                          | staff[0]——新对象(序列#7)      |
| 72 00 07 Manager            | 新类、字符串长度、类名(序列#2)        |
| 36 06 AE 13 63 8F 59 B7 02  | 指纹和标志                    |
| 00 01                       | 实例域的数量                   |
| L 00 09 secretary           | 实例域的类型和名字                |
| 74 00 0A LEmployee;         | 实例域的类名——String(序列#3)     |
| 78                          | 结束标记                     |
| 72 00 08 Employee           | 超类——新类、字符串长度、类名(序列#4)    |
| E6 02 86 7D AE AC 18 1B 02  | 指纹和标志                    |
| 00 03                       | 实例域的数量                   |
| D 00 06 salary              | 实例域的类型和名字                |
| L 00 07 hireDay             | 实例域的类型和名字                |
| 74 00 10 Ljava/util/Date;   | 实例域的类名——String(序列#5)     |
| L 00 04 name                | 实例域的类型和名字                |
| 74 00 12 Ljava/lang/String; | 实例域的类名——String(序列#6)     |
| 78                          | 结束标记                     |
| 70                          | 无超类                      |
| 40 F3 88 00 00 00 00 00     | salary域的值——double        |
| 73                          | hireDay域的值——新对象(序列#9)    |
| 72 00 0E java.util.Date     | 新类、字符串长度、类名(序列#8)        |
| 68 6A 81 01 4B 59 74 19 03  | 指纹和标志                    |
| 00 00                       | 无实例变量                    |
| 78                          | 结束标记                     |
| 70                          | 无超类                      |
| 77 08                       | 外部存储、字节的数量               |
| 00 00 00 83 E9 39 E0 00     | 日期                       |
| 78                          | 结束标记                     |
| 74 00 0C Carl Cracker       | name域的值——String(序列#10)   |
| 73                          | secretary域的值——新对象(序列#11) |
| 71 00 7E 00 04              | 已有的类(使用序列#4)             |
| 40 E8 6A 00 00 00 00 00     | salary域的值——double        |
| 73                          | hireDay域的值——新对象(序列#12)   |
| 71 00 7E 00 08              | 已有的类(使用序列#8)             |
| 77 08                       | 外部存储、字节的数量               |
| 00 00 00 91 1B 4E B1 80     | 日期                       |
| 78                          | 结束标记                     |
| 74 00 0C Harry Hacker       | name域的值——String(序列#13)   |

|                            |                              |
|----------------------------|------------------------------|
| 71 00 7E 00 0B             | staff[1]——已有的对象（使用序列#11）     |
| 73                         | staff[2]——新对象（序列#14）         |
| 71 00 7E 00 02             | 已有的类（使用序列#2）                 |
| 40 E3 88 00 00 00 00 00    | salary域的值——double            |
| 73                         | hireDay域的值——新对象（序列#15）       |
| 71 00 7E 00 0B             | 已有的类（使用序列#8）                 |
| 77 08                      | 外部存储、字节的数量                   |
| 00 00 00 94 6D 3E EC 00 00 | 日期                           |
| 78                         | 结束标记                         |
| 74 00 0B Tony Tester       | name域的值——String（序列#16）       |
| 71 00 7E 00 0B             | secretary域的值——已有的对象（使用序列#11） |

当然，研究这些编码大概与阅读常用的电话号码簿一样枯燥。了解确切的文件格式确实不那么重要（除非你试图通过修改数据来达到不可告人的目的），但是关于对象流包含它所包含的所有对象的详细描述，并且这些充足的细节可以用来重构对象和对象数组，因此了解它还是大有益处的。

你应该记住：

- 对象流输出中包含所有对象的类型和数据域。
- 每个对象都被赋予一个序列号。
- 相同对象的重复出现将被存储为对这个对象的序列号的引用。

### 1.5.2 修改默认的序列化机制

某些数据域永远都不应该被序列化，例如，只对本地访问有意义的存储文件句柄或窗口句柄的整数值，这种信息在稍后重新加载对象或将其传送到其他机器上时都是没有用处的。事实上，这种域的值如果不恰当，还会引起本地方法崩溃。Java拥有一种很简单的机制来防止这种域被序列化，那就是将它们标记成是transient的。如果这些域属于不可序列化的类，你也需要将它们标记成transient的。瞬时的域在对象被序列化时总是被跳过的。

序列化机制为单个的类提供了一种方式，去向默认的读写行为添加验证或任何其他想要的行为。可序列化的类可以定义具有下列签名的方法：

```
private void readObject(ObjectInputStream in)
    throws IOException, ClassNotFoundException;
private void writeObject(ObjectOutputStream out)
    throws IOException;
```

之后，数据域就再也不被自动序列化，取而代之的是调用这些方法。

下面是一个典型的示例。在java.awt.geom包中有大量的类都是不可序列化的，例如Point2D.Double。现在假设你想要序列化一个LabeledPoint类，它存储了一个String和一个Point2D.Double。首先，你需要将Point2D.Double标记成transient，以避免抛出NotSerializableException。

```
public class LabeledPoint implements Serializable
{
```

```

private String label;
private transient Point2D.Double point;
}

```

在`writeObject`方法中，我们首先通过调用`defaultWriteObject`方法写出对象描述符和`String`域及其状态，这是`ObjectOutputStream`类中的一个特殊的方法，它只能在可序列化类的`writeObject`方法中被调用。然后，我们使用标准的`DataOutput`调用写出点的坐标。

```

private void writeObject(ObjectOutputStream out)
    throws IOException
{
    out.defaultWriteObject();
    out.writeDouble(point.getX());
    out.writeDouble(point.getY());
}

```

在`readObject`方法中，我们反过来执行上述过程：

```

private void readObject(ObjectInputStream in)
    throws IOException
{
    in.defaultReadObject();
    double x = in.readDouble();
    double y = in.readDouble();
    point = new Point2D.Double(x, y);
}

```

另一个例子是`java.util.Date`类，它提供了自己的`readObject`和`writeObject`方法，这些方法将日期写出为从纪元（UTC时间1970年1月1日0点）开始的毫秒数。`Date`类有一个复杂的内部表示，为了便于查询，它存储了一个`Calendar`对象和一个毫秒计数值。`Calendar`状态是冗余的，因此并不需要保存。

`readObject`和`writeObject`方法只需要保存和加载它们的数据域，而不需要关心超类数据和任何其他类的信息。

除了让序列化机制来保存和恢复对象数据，类还可以定义它自己的机制。为了做到这一点，这个类必须实现`Externalizable`接口，这需要它定义两个方法：

```

public void readExternal(ObjectInputStream in)
    throws IOException, ClassNotFoundException;
public void writeExternal(ObjectOutputStream out)
    throws IOException;

```

与前面一节描述的`readObject`和`writeObject`不同，这些方法对包括超类数据在内的整个对象的存储和恢复负全责，而序列化机制在流中仅仅只是记录该对象所属的类。在读入可外部化的类时，对象流将用默认的构造器创建一个对象，然后调用`readExternal`方法。下面展示了如何为`Employee`类实现这些方法：

```

public void readExternal(ObjectInput s)
    throws IOException
{
    name = s.readUTF();
    salary = s.readDouble();
    hireDay = new Date(s.readLong());
}

```

```
public void writeExternal(ObjectOutput s)
    throws IOException
{
    s.writeUTF(name);
    s.writeDouble(salary);
    s.writeLong(hireDay.getTime());
}
```

**!** 提示：序列化有些慢是因为虚拟机必须了解每个对象的结构。如果你关心性能，并且需要读写某个特定类的大量对象，那么你就应该研究一下**Externalizable**接口。在<http://java.sun.com/developer/TechTips/2000/tt0425.html>处的教程提示验证了在雇员类的情况下，使用外部化读写大约比默认的序列化要快35%~40%。

**X** 警告：**readObject**和**writeObject**方法是私有的，并且只能被序列化机制调用。与此不同的是，**readExternal**和**writeExternal**方法是公共的。特别是，**readExternal**还潜在地允许修改现有对象的状态。

### 1.5.3 序列化单例和类型安全的枚举

在序列化和反序列化对象被认为是惟一时，你必须加倍当心，这通常会在实现单例和类型安全的枚举时发生。

如果你使用Java SE 5.0的**enum**结构，那么你就不必担心序列化，它能够正常工作。但是，假设你在维护遗留代码，其中包含下面这样的枚举类型：

```
public class Orientation
{
    public static final Orientation HORIZONTAL = new Orientation(1);
    public static final Orientation VERTICAL = new Orientation(2);
    private Orientation(int v) { value = v; }
    private int value;
}
```

这种风格在枚举被添加到Java语言中之前是很普遍的。注意，其构造器是私有的。因此，不可能创建出超出**Orientation.HORIZONTAL**和**Orientation.VERTICAL**之外的对象。特别是，你可以使用==操作符来测试对象的等同性：

```
if (orientation == Orientation.HORIZONTAL) ...
```

当类型安全的枚举实现了**Serializable**接口时，你必须牢记存在着一种曲解。此时，默认的序列化机制是不适用的。假设我们写出一个**Orientation**类型的值，并再次将其读回：

```
Orientation original = Orientation.HORIZONTAL;
ObjectOutputStream out = ...;
out.write(value);
out.close();
ObjectInputStream in = ...;
Orientation saved = (Orientation) in.read();
```

现在，下面的测试：

```
if (saved == Orientation.HORIZONTAL) ...
```

将失败。事实上，**saved**的值是**Orientation**类型的一个全新的对象，它与任何预定义的常量都

不等同。即使构造器是私有的，序列化机制也可以创建新的对象！

为了解决这个问题，你需要定义另外一种称为`readResolve`的特殊序列化方法。如果定义了`readResolve`方法，在对象被序列化之后就会调用它。它必须返回一个对象，而该对象之后会成为`readObject`的返回值。在上面的情况下，`readResolve`方法将检查`value`域并返回恰当的枚举常量：

```
protected Object readResolve() throws ObjectStreamException
{
    if (value == 1) return Orientation.HORIZONTAL;
    if (value == 2) return Orientation.VERTICAL;
    return null; // this shouldn't happen
}
```

请记住向遗留代码中所有类型安全的枚举以及向所有支持单例设计模式的类中添加`readResolve`方法。

#### 1.5.4 版本管理

如果使用序列化来保存对象，就需要考虑在程序演化时会有什么问题。例如，1.1版本可以读入旧文件吗？仍旧使用1.0版本的用户可以读入新版本产生的文件吗？显然，如果对象文件可以处理类的演化问题，那它正是我们想要的。

乍一看，这好像是不可能的。无论类的定义产生了什么样的变化，它的SHA指纹也会跟着变化，而我们都知道对象流将拒绝读入具有不同指纹的对象。但是，类可以表明它对其早期版本保持兼容，要想这样做，就必须首先获得这个类的早期版本的指纹。我们可以使用JDK中的单机程序`serialver`来获得这个数字，例如，运行下面的命令

```
serialver Employee
```

将会打印出

```
Employee: static final long serialVersionUID = -1814239825517340645L;
```

如果在运行`serialver`程序时添加`-show`选项，那么这个程序就会产生下面的图形化对话框（请查看图1-8）。

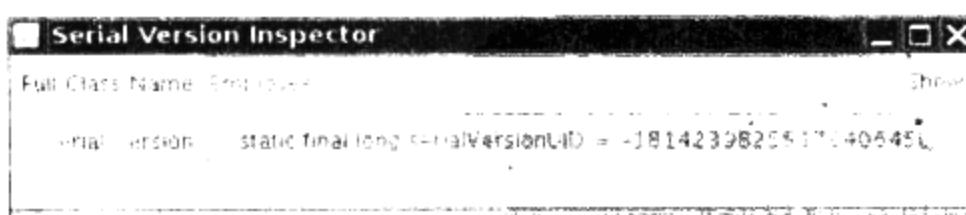


图1-8 serialver程序的图形化版本

这个类的所有较新的版本都必须把`serialVersionUID`常量定义为与最初版本的指纹相同。

```
class Employee implements Serializable // version 1.1
{
    ...
    public static final long serialVersionUID = -1814239825517340645L;
}
```

如果一个类具有名为`serialVersionUID`的静态数字成员，它就不需要再人工地计算其指纹，

而只需直接使用这个值。

一旦这个静态数据成员被置于某个类的内部，那么序列化系统将可以读入这个类的对象的不同版本。

只有在这个类的方法产生变化时，在读入新对象数据时才不会有任何问题。但是，如果数据域产生了变化，那么就可能会有问题。例如，旧文件对象可能比程序中的对象具有更多或更少的数据域，或者数据域的类型可能有所不同。在这些情况下，对象流将尽力将流对象转换成这个类当前的版本。

对象流会将这个类当前版本的数据域与流中版本的数据域进行比较，当然，对象流只会考虑非瞬时和非静态的数据域。如果这两部分数据域之间名字匹配而类型不匹配，那么对象流不会尝试将一种类型转换成另一种类型，因为这两个对象不兼容；如果流中的对象具有在当前版本中所没有的数据域，那么对象流会忽略这些额外的数据；如果当前版本具有在流化对象中所没有的数据域，那么这些新添加的域将被设置成它们的默认值（如果是对象则是null，如果是数字则为0，如果是boolean值则是false）。

下面是一个示例：假设我们已经用雇员类的最初版本（1.0）在磁盘上保存了大量的雇员记录，现在我们在Employee类中添加了称为department的数据域，从而将其演化到了2.0版本。图1-9展示了将1.0版的对象读入到使用2.0版对象的程序中的情形，可以看到department域被设置成了null。图1-10展示了相反的情况：一个使用1.0版对象的程序读入了2.0版的对象，可以看到额外的department域被忽略。

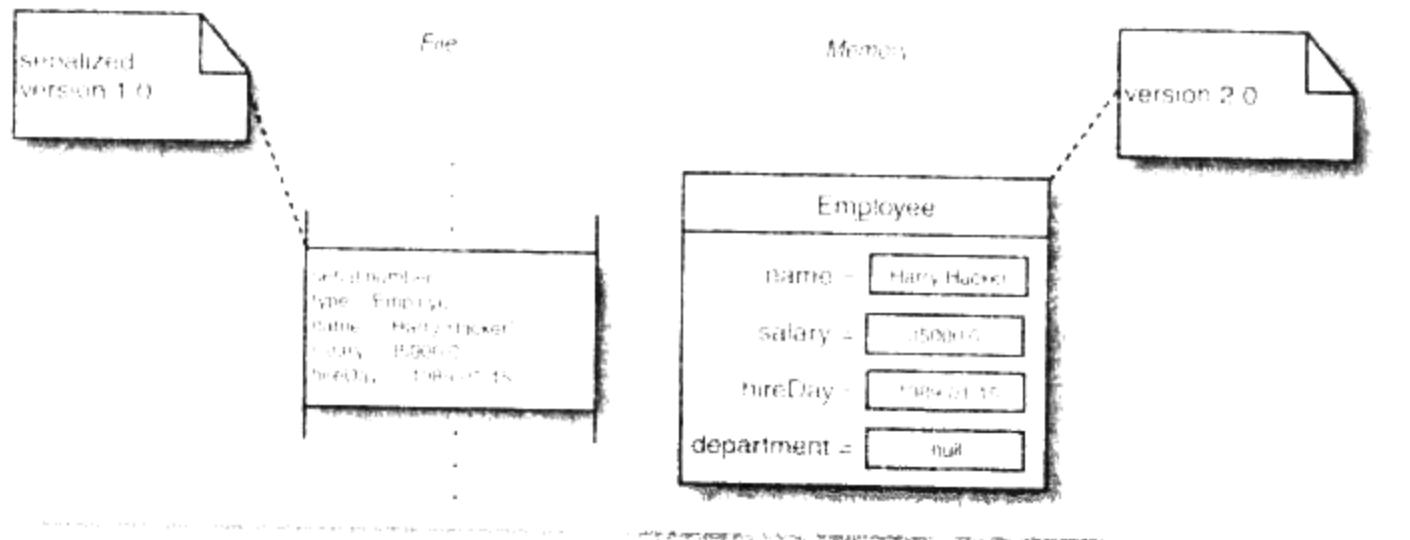


图1-9 读入具有较少数据域的对象

这种处理是安全的吗？视情况而定。丢掉数据域看起来是无害的，因为接收者仍旧拥有它知道如何处理的所有数据，但是将数据域设置为null却有可能并不那么安全。许多类都费尽心思地在其所有的构造器中将所有的数据域都初始化为非null的值，以使得其各个方法都不必去处理null数据。因此，这个问题取决于类的设计者是否能够在readObject方法中实现额外的代码去订正版本不兼容问题，或者是否能够确保所有的方法在处理null数据时都足够健壮。

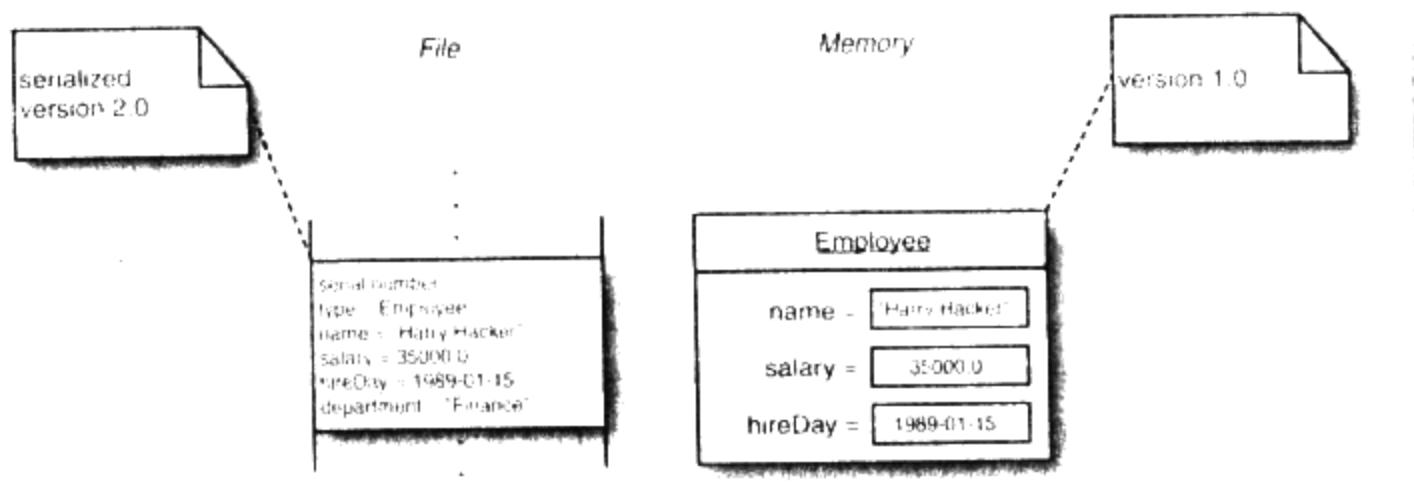


图1-10 读入具有较多数据域的对象

### 1.5.5 为克隆使用序列化

序列化机制有一种很有趣的用法：它提供了一种克隆对象的简便途径，只要对应的类是可序列化的即可。其做法很简单：直接将对象序列化到输出流中，然后将其读回。这样产生的新对象是对现有对象的一个深拷贝（deep copy）。在此过程中，我们不必将对象写出到文件中，因为可以用`ByteArrayOutputStream`将数据保存到字节数组中。

正如程序清单1-5所示，要想得到`clone`，只需扩展`Serializable`类，这样就完事了。

我们应该当心这个方法，尽管它很灵巧，但是它通常会比显式地构建新对象并复制或克隆数据域的克隆方法慢得多。

#### 程序清单1-5 SerialCloneTest.java

```

1. import java.io.*;
2. import java.util.*;
3.
4. public class SerialCloneTest
5. {
6.     public static void main(String[] args)
7.     {
8.         Employee harry = new Employee("Harry Hacker", 35000, 1989, 10, 1);
9.         // clone harry
10.        Employee harry2 = (Employee) harry.clone();
11.
12.        // mutate harry
13.        harry.raiseSalary(10);
14.
15.        // now harry and the clone are different
16.        System.out.println(harry);
17.        System.out.println(harry2);
18.    }
19. }
20.
21. /**
22.  * A class whose clone method uses serialization.
23. */

```

```
24. class SerialCloneable implements Cloneable, Serializable
25. {
26.     public Object clone()
27.     {
28.         try
29.         {
30.             // save the object to a byte array
31.             ByteArrayOutputStream bout = new ByteArrayOutputStream();
32.             ObjectOutputStream out = new ObjectOutputStream(bout);
33.             out.writeObject(this);
34.             out.close();
35.
36.             // read a clone of the object from the byte array
37.             ByteArrayInputStream bin = new ByteArrayInputStream(bout.toByteArray());
38.             ObjectInputStream in = new ObjectInputStream(bin);
39.             Object ret = in.readObject();
40.             in.close();
41.
42.             return ret;
43.         }
44.         catch (Exception e)
45.         {
46.             return null;
47.         }
48.     }
49. }
50.
51. /**
52. * The familiar Employee class, redefined to extend the
53. * SerialCloneable class.
54. */
55. class Employee extends SerialCloneable
56. {
57.     public Employee(String n, double s, int year, int month, int day)
58.     {
59.         name = n;
60.         salary = s;
61.         GregorianCalendar calendar = new GregorianCalendar(year, month - 1, day);
62.         hireDay = calendar.getTime();
63.     }
64.
65.     public String getName()
66.     {
67.         return name;
68.     }
69.
70.     public double getSalary()
71.     {
72.         return salary;
73.     }
74.
75.     public Date getHireDay()
76.     {
77.         return hireDay;
78.     }
79.
```

```
80. public void raiseSalary(double byPercent)
81. {
82.     double raise = salary * byPercent / 100;
83.     salary += raise;
84. }
85.
86. public String toString()
87. {
88.     return getClass().getName()
89.         + "[name=" + name
90.         + ",salary=" + salary
91.         + ",hireDay=" + hireDay
92.         + "]";
93. }
94.
95. private String name;
96. private double salary;
97. private Date hireDay;
98 }
```

## 1.6 文件管理

你已经学习了如何从文件中读写数据。但是，文件管理的内涵远比读写要多。`File`类封装了在用户机器上处理文件系统所需的所有功能。例如，可以使用`File`类来查询文件最后修改的时间，移除或重命名文件。换句话说，流类关心的是文件的内容，而`File`类关心的是在磁盘上文件的存储。

 **注意：**像在Java中常见的`File`类采用了最小公分母方式。例如，在Windows下，你可以查询（或设置）一个文件的只读标志，然而尽管你可以查询这个文件是否是隐藏文件，但是你却不能在不使用本地方法的情况下将其隐藏。

`File`对象最简单的构造器将接受一个（完全的）文件名。如果你没有提供路径名，那么Java就会使用当前路径。例如：

```
File f = new File("test.txt");
```

将提供一个在当前目录下具有这个名字的文件对象。（“当前目录”是执行虚拟机的进程的当前目录。如果你是通过命令行启动虚拟机的，那么它就是你启动java可执行文件的目录。）

 **警告：**因为反斜线字符在Java字符串中是转义字符，所以请保证在Windows风格的路径名中使用`\`（“C:\\Windows\\win.ini”）。在Windows中，还可以使用单斜线（“C:/Windows/win.ini”），因为大多数对Windows文件处理系统的调用都将斜线解释成为文件分隔符。但是，这种做法并不推荐，Windows系统功能的行为在不断地变化，并且在其他操作系统上，文件分隔符也可能会有所不同。因此，考虑到移植问题，应该使用程序运行平台所使用的文件分隔符，它是以常量字符串`File.separator`的形式存储的。

对这个构造器的调用不会在指定文件不存在情况下创建一个具有指定文件名的文件。实际上，从`File`对象中创建文件是由文件流类的构造器或`File`类中的`createNewFile`方法完成的，`createNewFile`方法只有在具有指定文件名的文件不存在的情况下才会创建文件，并且会返回

一个boolean值，来说明文件是否被成功创建。

另一方面，一旦有了File对象，File类中exist方法就可以告知具有这个文件名的文件是否存在。例如，下面的实验程序就几乎可以肯定在任何人的机器上都会打印“false”，并且它仍旧可以打印出这个并不存在的文件的路径名。

```
import java.io.*;  
  
public class Test  
{  
    public static void main(String args[])  
    {  
        File f = new File("afilethatprobablydoesntexist");  
        System.out.println(f.getAbsolutePath());  
        System.out.println(f.exists());  
    }  
}
```

File对象还有另外两个构造器：

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

它可以在由path参数指定的目录中创建具有给定名字的File对象。（如果path参数为null，这个构造器将使用当前目录创建File对象。）

最后，你可以在下面的构造器中使用已有的File对象：

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

其中，File对象表示一个目录，与前面一样，如果dir为null，这个构造器就会在当前目录中创建一个File对象。

有点令人疑惑的是，File对象既可以表示文件，也可以表示目录（可能是因为Java设计者们最熟悉的操作系统碰巧就是把目录当作文件实现的）。你可以使用isDirectory和isFile方法来了解一个文件对象表示的到底是文件还是目录。这有些令人惊讶，在面向对象的系统中，你可能曾经期望有个单独的Directory类，它可能扩展自File类。

为了让一个对象表示目录，我们只需在File构造器中提供目录名：

```
File tempDir = new File(File.separator + "temp");
```

如果这个目录不存在，你可以使用mkdir方法创建它：

```
tempDir.mkdir();
```

如果一个文件对象表示的是目录，使用list()方法可以获得由这个目录下的文件名构成的数组。程序清单1-6中的程序使用了所有这些方法打印出了在命令行中输入的任何路径的目录子结构。（我们很容易就可以将这个程序修改为一个实用类，让其返回可以用于进一步处理的子目录列表。）



**提示：**在处理文件或目录名时，应该总是使用File对象而不是字符串。这样做的好处很多，例如，File类的equals方法知道某些文件系统不是大小写敏感的，或者在目录名尾部的（是无关紧要的）。

**程序清单1-6 FindDirectories.java**

```

1. import java.io.*;
2.
3. /**
4. * @version 1.00 05 Sep 1997
5. * @author Gary Cornell
6. */
7. public class FindDirectories
8. {
9.     public static void main(String[] args)
10.    {
11.        // if no arguments provided, start at the parent directory
12.        if (args.length == 0) args = new String[] { ".." };
13.
14.        try
15.        {
16.            File pathName = new File(args[0]);
17.            String[] fileNames = pathName.list();
18.
19.            // enumerate all files in the directory
20.            for (int i = 0; i < fileNames.length; i++)
21.            {
22.                File f = new File(pathName.getPath(), fileNames[i]);
23.
24.                // if the file is again a directory, call the main method recursively
25.                if (f.isDirectory())
26.                {
27.                    System.out.println(f.getCanonicalPath());
28.                    main(new String[] { f.getPath() });
29.                }
30.            }
31.        }
32.        catch (IOException e)
33.        {
34.            e.printStackTrace();
35.        }
36.    }
37. }

```

可以使用**FileNameFilter**对象作为**list**方法的参数来减小列表长度，而不是列出目录中所有的文件。这些对象是满足**FileNameFilter**接口的类的实例。

一个实现**FileNameFilter**接口的类需要定义**accept**方法。下面是一个简单的**FileNameFilter**类的示例，它只允许带有特定扩展的文件。

```

public class ExtensionFilter implements FilenameFilter
{
    public ExtensionFilter(String ext)
    {
        extension = "." + ext;
    }

    public boolean accept(File dir, String name)
    {
        return name.endsWith(extension);
    }
}

```

```
    }  
  
    private String extension;  
}
```

在编写可移植的程序时，指定带有子目录的文件名就成为了一种挑战。正如之前我们提到的，已经证明在Windows中也可以使用斜线（UNIX分隔符）作为目录分隔符，但是其他操作系统也许并不允许这么做，因此我们不推荐使用斜线。

**X 警告：**如果你在构建File对象时，确实在Windows中使用了斜线作为目录分隔符，那么getAbsolutePath方法就会返回包含斜线的文件名，这对于Windows用户来说有些奇怪。与此不同的是，使用getCanonicalPath方法就可以将斜线替换成反斜线。

使用File类中存储在名为separator的静态实例域中的有关当前目录分隔符的信息就要好得多。在Windows环境中，它是反斜线（\）；在UNIX环境中，它是斜线（/）。例如：

```
File foo = new File("Documents" + File.separator + "data.txt")
```

当然，如果使用File的第二个可选版本的构造器

```
File foo = new File("Documents", "data.txt")
```

那么这个构造器将提供恰当的分隔符。

下面的API注释是我们认为File类中最重要的其他方法，它们的用法都很直观。

#### **API** java.io.File 1.0

- boolean canRead()
- boolean canWrite()

- boolean canExecute() 6

表明文件是否可读、可写或可执行。

- boolean setReadable(boolean state, boolean ownerOnly) 6

- boolean setWritable(boolean state, boolean ownerOnly) 6

- boolean setExecutable(boolean state, boolean ownerOnly) 6

设置这个文件的可读、可写或可执行状态。如果ownerOnly为true，状态设置只对文件拥有者有效，否则，对所有人有效。这些方法在设置状态成功后返回true。

- static boolean createTempFile(String prefix, String suffix) 1.2

- static boolean createTempFile(String prefix, String suffix, File directory) 1.2

在系统的默认临时目录或给定目录中创建一个临时文件，并使用给定的前缀或后缀来生成文件名。

参数：prefix

至少3个字符长的前缀字符串

suffix

可选的后缀，如果为null，就是用.tmp

directory

创建文件的目录。如果为null，就在当前工作目录中创建文件

- boolean delete()

尝试删除这个文件，如果文件被删除，则返回true，否则返回false。

- void deleteOnExit()

请求在虚拟机关闭时将文件删除。

- boolean exists()

如果这个文件或目录存在，则返回true，否则返回false。

- **String getAbsolutePath()**

返回包含绝对路径名的字符串。提示：应使用getCanonicalPath来替代它。

- **File getCanonicalFile() 1.2**

返回包含这个文件的规范路径名的File对象。特别是它会移除冗余的"."目录，使用恰当的目录分隔符，并可以获得底层文件系统所选择的大小写处理方式。

- **String getCanonicalPath() 1.1**

返回包含这个文件的规范路径名的字符串。特别是它会移除冗余的"."目录，使用恰当的目录分隔符，并可以获得底层文件系统所选择的大小写处理方式。

- **String getName()**

返回包含这个File对象的文件名的字符串（不包含路径信息）。

- **String getParent()**

返回这个File对象的父亲名字的字符串。如果这个File对象是一个文件，那么它的父亲就是包含它的目录；如果它是一个目录，那么它的父亲就是其父目录，如果它没有父目录，那么它的父亲就是null。

- **File getParentFile() 1.2**

返回这个File目录的父目录的File对象。请查看getParent中关于“父亲”的定义。

- **String getPath()**

返回包含这个文件的路径名的字符串。

- **boolean isDirectory()**

如果这个File表示一个目录，则返回true，否则返回false。

- **boolean isFile()**

如果这个File对象表示一个文件而不是一个目录或一个设备，则返回true。

- **boolean isHidden() 1.2**

如果这个File对象表示的是一个隐藏文件或目录，则返回true。

- **long lastModified()**

返回这个文件最后被修改的时间（记录从GMT时间1970年1月1日0时以来的毫秒数），或者在文件不存在时返回0。使用Date(long)构造器可以将这个值转换成一个日期。

- **long length()**

返回这个文件按照字节度量的长度，或者在文件不存在时返回0。

- **String[] list()**

返回由这个File对象包含的文件名和目录名构成的字符串数组，或者在这个File不表示目录时返回null。

- **String[] list(FilenameFilter filter)**

返回由这个File对象包含的满足过滤器条件的文件名和目录名构成的字符串数组，或者在不存在符合条件的内容时返回null。

- **File[] listFiles() 1.2**

返回由这个File对象包含的文件和目录所对应的File对象构成的数组，或者在这个File不表示目录时返回null。

- **File[] listFiles(FilenameFilter filter) 1.2**

返回由这个File对象包含的满足过滤器条件的文件和目录所对应的File对象构成的数组，

或者在不存在符合条件的内容时返回null。

- **static File[] listRoots() 1.2**

返回由所有可获得的文件根对应的File对象构成的数组（例如，在Windows系统上，可以获得表示所安装的硬盘驱动器的多个File对象，包括本地驱动器和网络映射驱动器。在UNIX系统上，你只能获得"/"）。

- **boolean createNewFile() 1.2**

自动创建一个由File对象给定名字的新文件。也就是说，检查文件名和创建文件不会被文件系统的其他行为所中断。在这个方法创建文件成功后返回true。

- **boolean mkdir()**

创建一个由这个File对象给定名字的子目录，如果子目录创建成功则返回true，否则返回false。

- **boolean mkdirs()**

与mkdir不同，这个方法在必要时将创建父目录，只要有必需的目录未能创建成功，则返回false。

- **boolean renameTo(File newName)**

如果文件名被修改，则返回true，否则返回false。

- **boolean setLastModified(long time) 1.2**

设置这个文件的最后修改时间，如果设置成功则返回true，否则返回false。time是一个表示从GMT时间1970年1月1日0时以来的毫秒数的长整数，可以使用Date类的getTime方法来计算这个值。

- **boolean setReadOnly() 1.2**

将这个文件设置成只读，如果设置成功则返回true，否则返回false。

- **URL toURL() 1.2**

将这个File对象转换成一个文件的URL。

- **long getTotalSpace() 6**

- **long getFreeSpace() 6**

- **long getUsableSpace() 6**

获得由File对象所描述的分区的总大小、未分配字节的数量和可用字节的数量。如果这个File对象描述的不是一个分区，这些方法将返回0。



### java.io.FilenameFilter 1.0

- **boolean accept(File dir, String name)**

应该定义为在文件能够匹配过滤器标准时返回true。

参数：dir 一个表示包含文件目录的File对象

name 文件名

## 1.7 新I/O

Java SE 1.4引入了大量用于改进输入/输出处理机制的特性，它们位于java.nio包中，合称“新I/O”。（当然，“新”这个称号有些令人遗憾，因为多年以后，这个包也就不再新了。）

这个包中包含对下列特性的支持：

- 字符集编码器和解码器

- 非阻塞的I/O

- 内存映射文件
- 文件加锁机制

我们已经在第1.2.4节中讨论过了字符编码机制和解码机制，非阻塞的I/O将在第3章中讨论，因为它在跨网络的通信中显得特别重要。在下面的小节中，我们将详细探讨内存映射文件和文件加锁机制。

### 1.7.1 内存映射文件

大多数操作系统都可以利用虚拟内存实现来将一个文件或者文件的一部分“映射”到内存中。然后，这个文件就可以当作是内存数组一样地访问，这比传统的文件操作要快得多。

在本节的末尾，你可以发现一个使用传统的文件输入和内存映射文件来计算文件的CRC32校验和的程序。在同一台机器上，我们对JDK的jre/lib目录中的37MB的rt.jar文件用不同的方式来计算校验和，记录下来的时间数据如表1-6所示。

正如你所见，在这台特定的机器上，内存映射比使用带缓冲的顺序输入要稍微快一点，但是比使用RandomAccessFile快很多。

当然，精确的值因机器不同会产生很大的差异，但是很明显，如果需要使用随机访问，那么性能受损是不可避免的。另一方面，对于中等尺寸文件的顺序读入没有必要使用内存映射。

java.nio包使内存映射变得十分简单，下面就是我们需要做的。

首先，从文件中获得一个通道（channel），通道是用于磁盘文件的一种抽象，它使我们可以访问诸如内存映射、文件加锁机制以及文件间快速数据传递等操作系统特性。我们可以通过调用getChannel方法来获得通道，这个方法已经添加到了FileInputStream、FileOutputStream和RandomAccessFile类中。

```
FileInputStream in = new FileInputStream(...);
FileChannel channel = in.getChannel();
```

然后，通过调用FileChannel类的map方法从这个通道中获得一个MappedByteBuffer。你可以指定想要映射的文件区域与映射模式，有三种模式可以得到支持：

- FileChannel.MapMode.READ\_ONLY：所产生的缓冲区是只读的，任何对该缓冲区写入的尝试都会导致ReadOnlyBufferException异常。
- FileChannel.MapMode.READ\_WRITE：所产生的缓冲区是可写的，任何修改都会在某个时刻写回到文件中。注意，其他映射同一个文件的程序可能不能立即看到这些修改，多个程序同时进行文件映射的确切行为是依赖于操作系统的。
- FileChannel.MapMode.PRIVATE：所产生的缓冲区是可写的，但是任何修改对这个缓冲区来说都是私有的，不会传播到文件中。

一旦有了缓冲区，就可以使用ByteBuffer类和Buffer超类的方法读写数据了。

缓冲区支持顺序和随机数据访问，它有一个可以通过get和put操作来推动的位置。例如，像下面这样顺序遍历缓冲区中的所有字节：

```
while (buffer.hasRemaining())
{
```

表1-6 文件操作的处理时间数据

| 方 法     | 时 间  |
|---------|------|
| 普通输入流   | 110秒 |
| 带缓冲的输入流 | 9.9秒 |
| 随机访问文件  | 162秒 |
| 内存映射文件  | 7.2秒 |

```
byte b = buffer.get();
...
}
```

或者，像下面这样进行随机访问：

```
for (int i = 0; i < buffer.limit(); i++)
{
    byte b = buffer.get(i);
    ...
}
```

你可以用下面的方法来读写字节数组

```
get(byte[] bytes)
get(byte[], int offset, int length)
```

最后，还有下面的方法：

```
getInt
getLong
getShort
getChar
getFloat
getDouble
```

用来读入在文件中存储为二进制值的基本类型值。正如我们提到的，Java对二进制数据使用高位在前的排序机制，但是，如果需要处理的文件包含低位在前的二进制数字，那么只需调用

```
buffer.order(ByteOrder.LITTLE_ENDIAN);
```

要查询缓冲区内当前的字节顺序，可以调用：

```
ByteOrder b = buffer.order()
```



**警告：**这一对方法没有使用set/get命名惯例。

要向缓冲区写数字，可以使用下列的方法：

```
putInt
putLong
putShort
putChar
putFloat
putDouble
```

程序清单1-7用于计算文件的32位的循环冗余校验和（CRC32），这个数值就是经常用来判断一个文件是否已损坏的校验和，因为文件损坏极有可能导致校验和改变。`java.util.zip`包中包含一个CRC32类，可以使用下面的循环来计算一个字节序列的校验和：

```
CRC32 crc = new CRC32();
while (more bytes)
    crc.update(next byte)
long checksum = crc.getValue();
```



**注意：**对CRC算法有一个很精细的解释，请查看<http://www.relisoft.com/Science/CrcMath.html>。

CRC计算的细节并不重要，我们只是将它作为一个有用的文件操作的实例来使用。

应该像下面这样运行程序：

```
java NIOTest filename
```

### 程序清单1-7 NIOTest.java

```
1. import java.io.*;
2. import java.nio.*;
3. import java.nio.channels.*;
4. import java.util.zip.*;
5.
6. /**
7. * This program computes the CRC checksum of a file. <br>
8. * Usage: java NIOTest filename
9. * @version 1.01 2004-05-11
10. * @author Cay Horstmann
11. */
12. public class NIOTest
13. {
14.     public static long checksumInputStream(String filename) throws IOException
15.     {
16.         InputStream in = new FileInputStream(filename);
17.         CRC32 crc = new CRC32();
18.
19.         int c;
20.         while ((c = in.read()) != -1)
21.             crc.update(c);
22.         return crc.getValue();
23.     }
24.
25.     public static long checksumBufferedInputStream(String filename) throws IOException
26.     {
27.         InputStream in = new BufferedInputStream(new FileInputStream(filename));
28.         CRC32 crc = new CRC32();
29.
30.         int c;
31.         while ((c = in.read()) != -1)
32.             crc.update(c);
33.         return crc.getValue();
34.     }
35.
36.     public static long checksumRandomAccessFile(String filename) throws IOException
37.     {
38.         RandomAccessFile file = new RandomAccessFile(filename, "r");
39.         long length = file.length();
40.         CRC32 crc = new CRC32();
41.
42.         for (long p = 0; p < length; p++)
43.         {
44.             file.seek(p);
45.             int c = file.readByte();
46.             crc.update(c);
47.         }
48.         return crc.getValue();
49.     }
50. }
```

```
49.    }
50.
51.   public static long checksumMappedFile(String filename) throws IOException
52.   {
53.       FileInputStream in = new FileInputStream(filename);
54.       FileChannel channel = in.getChannel();
55.
56.       CRC32 crc = new CRC32();
57.       int length = (int) channel.size();
58.       MappedByteBuffer buffer = channel.map(FileChannel.MapMode.READ_ONLY, 0, length);
59.
60.       for (int p = 0; p < length; p++)
61.       {
62.           int c = buffer.get(p);
63.           crc.update(c);
64.       }
65.       return crc.getValue();
66.   }
67.
68.   public static void main(String[] args) throws IOException
69.   {
70.       System.out.println("Input Stream:");
71.       long start = System.currentTimeMillis();
72.       long crcValue = checksumInputStream(args[0]);
73.       long end = System.currentTimeMillis();
74.       System.out.println(Long.toHexString(crcValue));
75.       System.out.println((end - start) + " milliseconds");
76.
77.       System.out.println("Buffered Input Stream:");
78.       start = System.currentTimeMillis();
79.       crcValue = checksumBufferedInputStream(args[0]);
80.       end = System.currentTimeMillis();
81.       System.out.println(Long.toHexString(crcValue));
82.       System.out.println((end - start) + " milliseconds");
83.
84.       System.out.println("Random Access File:");
85.       start = System.currentTimeMillis();
86.       crcValue = checksumRandomAccessFile(args[0]);
87.       end = System.currentTimeMillis();
88.       System.out.println(Long.toHexString(crcValue));
89.       System.out.println((end - start) + " milliseconds");
90.
91.       System.out.println("Mapped File:");
92.       start = System.currentTimeMillis();
93.       crcValue = checksumMappedFile(args[0]);
94.       end = System.currentTimeMillis();
95.       System.out.println(Long.toHexString(crcValue));
96.       System.out.println((end - start) + " milliseconds");
97.   }
98. }
```



### java.io.FileInputStream 1.0

- `FileChannel getChannel() 1.4`

返回用于访问这个流的通道。

**API** `java.io.FileOutputStream` 1.0

- `FileChannel getChannel()` 1.4

返回用于访问这个流的通道。

**API** `java.io.RandomAccessFile` 1.0

- `FileChannel getChannel()` 1.4

返回用于访问这个流的通道。

**API** `java.nio.channels.FileChannel` 1.4

- `MappedByteBuffer map(FileChannel.MapMode mode, long position, long size)`

将文件的一个区域映射到内存中。

参数: mode `FileChannel.MapMode`类中的常量`READ_ONLY`、`READ_WRITE`、或`PRIVATE`之一

position 映射区域的起始位置

size 映射区域的大小

**API** `java.nio.Buffer` 1.4

- `boolean hasRemaining()`

如果当前的缓冲区位置没有达到这个缓冲区的界限位置则返回`true`。

- `int limit()`

返回这个缓冲区的界限位置, 即没有任何值可用的第一个位置。

**API** `java.nio.ByteBuffer` 1.4

- `byte get()`

从当前位置获得一个字节, 并将当前位置推到下一个字节。

- `byte get(int index)`

从指定索引处获得一个字节。

- `ByteBuffer put(byte b)`

向当前位置推入一个字节, 并将当前位置推到下一个字节。返回对这个缓冲区的引用。

- `ByteBuffer put(int index, byte b)`

向指定索引处推入一个字节。返回对这个缓冲区的引用。

- `ByteBuffer get(byte[] destination)`

- `ByteBuffer get(byte[] destination, int offset, int length)`

用缓冲区中的字节来填充字节数组, 或者字节数组的某个区域, 并将当前位置向前推读入的字节数个位置。如果缓冲区不够, 那么就不会读入任何字节, 并抛出`BufferUnderflowException`。返回对这个缓冲区的引用。

参数: destination 要填充的字节数组

offset 要填充区域的偏移量

length 要填充区域的长度

- ByteBuffer put(byte[] source)
- ByteBuffer put(byte[] source, int offset, int length)

将字节数组中的所有字节或者给定区域的字节都推入缓冲区中，并将当前位置向前推写出的字节数个位置。如果缓冲区不够，那么就不会读入任何字节，并抛出BufferUnderflowException。返回对这个缓冲区的引用。

参数：source 要写出的数组

offset 要写出区域的偏移量

length 要写出区域的长度

- Xxx getXxx()
- Xxx getXxx(int index)
- ByteBuffer putXxx(xxx value)
- ByteBuffer putXxx(int index, xxx value)

获得或放置一个二进制数。Xxx Int、Long、Short、Char、Float或Double中的一个。

- ByteBuffer order(ByteOrder order)
- ByteOrder order()

设置或获得字节顺序，order的值是ByteOrder类的常量BIG\_ENDIAN或LITTLE\_ENDIAN中的一个。

### 1.7.2 缓冲区数据结构

在使用内存映射时，我们创建了单一的缓冲区横跨整个文件或我们感兴趣的文件区域。我们还可以使用更多的缓冲区来读写大小适度的信息块。

本节将简要地介绍Buffer对象上的基本操作。缓冲区是由具有相同类型的数值构成的数组，Buffer类是一个抽象类，它有众多的具体子类，包括ByteBuffer、CharBuffer、DoubleBuffer、IntBuffer、LongBuffer和ShortBuffer。

 注意：StringBuffer类与这些缓冲区没有关系。

在实践中，最常用的将是ByteBuffer和CharBuffer。如图1-11所示，每个缓冲区都具有：

- 一个容量，它永远不能改变。
- 一个读写位置，下一个值将在此进行读写。
- 一个界限，超过它进行读写是没有意义的。
- 一个可选的标记，用于重复一个读入或写出操作。

这些值满足下面的条件

$$0 \leq \text{标记} \leq \text{位置} \leq \text{界限} \leq \text{容量}$$

使用缓冲区的主要目的是执行“写，然后读入”循环。假设我们有一个缓冲区；在一开始，它的位置为0，界限等于容量。我们不断地调用put将值添加到这个缓冲区中，当我们耗尽所有的数据或者写出的数据量达到容量大小时，就该切换到读入操作了。

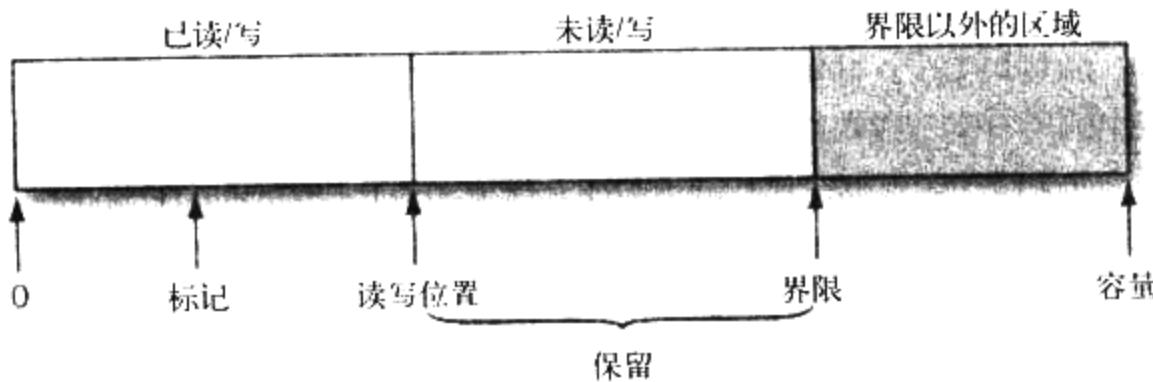


图1-11 一个缓冲区

这时调用`flip`方法将界限设置到当前位置，并把位置复位到0。现在在`remaining`方法返回正数时（它返回的值是“界限 - 位置”），不断地调用`get`。在我们将缓冲区中所有的值都读入之后，调用`clear`使缓冲区为下一次写循环做好准备。`clear`方法将位置复位到0，并将界限复位到容量。

如果你想重新读入缓冲区，可以使用`rewind`或`mark/reset`方法，详细内容请查看API注释。

#### API `java.nio.Buffer 1.4`

- `Buffer clear()`

通过将位置复位到0，并将界限复位到容量，使这个缓冲区为写出做好准备。返回`this`。

- `Buffer flip()`

通过将界限设置到位置，并将位置复位到0，使这个缓冲区为读入做好准备。返回`this`。

- `Buffer rewind()`

通过将读写位置复位到0，并保持界限不变，使这个缓冲区为重新读入相同的值做好准备。返回`this`。

- `Buffer mark()`

将这个缓冲区的标记设置到读写位置，返回`this`。

- `Buffer reset()`

将这个缓冲区的位置设置到标记，从而允许被标记的部分可以再次被读入或写出，返回`this`。

- `int remaining()`

返回剩余可读入或可写出的值的数量，即界限与位置之间的差异。

- `int position()`

返回这个缓冲区的位置。

- `int capacity()`

返回这个缓冲区的容量。



### java.nio.CharBuffer 1.4

- `char get()`
- `CharBuffer get(char[] destination)`
- `CharBuffer get(char[] destination, int offset, int length)`

从这个缓冲区的位置处开始，获得一个char值，或者某个范围的char值，然后将位置向前推过所读入的字符。最后两个方法返回this。

- `CharBuffer put(char c)`
- `CharBuffer put(char[] source)`
- `CharBuffer put(char[] source, int offset, int length)`
- `CharBuffer put(String source)`
- `CharBuffer put(CharBuffer source)`

从这个缓冲区的位置处开始，推入一个char值，或者某个范围的char值，然后将位置向前推过所写出的字符。

- `CharBuffer read(CharBuffer destination)`

从这个缓冲区中获得char值，然后将它们推入目标缓冲区，直至达到目标缓冲区的界限。返回this。

### 1.7.3 文件加锁机制

考虑一下多个同时执行的文件需要修改同一个文件的情形，很明显，这些程序需要以某种方式进行通信，不然这个文件很容易被损坏。

文件锁可以控制对文件或文件中某个范围的字节的访问，但是，文件加锁机制在不同的操作系统之间变化很大，这也就解释了为什么在JDK以前的版本中没有文件加锁能力。

文件加锁在应用程序中并非总是很常见，许多应用使用数据库进行数据存储，而数据库具有解决并发访问的机制。如果需要在平面文件中存储信息，就必须注意并发访问，你可能会发现使用数据库比设计复杂的文件加锁机制要更简单。

但是，在某些情况下，文件加锁机制仍然是必需的。假设你的应用将用户的偏好存储在一个配置文件中，当用户调用这个应用的两个实例时，这两个实例就有可能会同时希望写这个配置文件。在这种情况下，第一个实例应该锁定这个文件，当第二个实例发现这个文件被锁定时，它必须决策是等待直至这个文件解锁，还是直接跳过这个写操作过程。

要锁定一个文件，可以调用FileChannel类的lock或tryLock方法：

```
FileLock lock = channel.lock();
```

或

```
FileLock lock = channel.tryLock();
```

第一个调用会阻塞直至可获得锁，而第二个调用将立即返回，要么返回锁，要么在锁不可获得的情况下返回null。这个文件将保持锁定状态，直至这个通道关闭，或者在锁上再调用了release方法。

你还可以通过下面的调用锁定文件的一部分：

```
FileLock lock(long start, long size, boolean exclusive)
```

或

```
FileLock tryLock(long start, long size, boolean exclusive)
```

如果`exclusive`标志为`true`，则锁定文件的目的读写，而如果为`false`，则这是一个共享锁，它允许多个进程从文件中读入，并阻止任何进程获得独占的锁。并非所有的操作系统都支持共享锁，因此你可能会在请求共享锁的时候得到的却是独占的锁。调用`FileLock`类的`isShared`方法可以查询你所持有的锁的类型。

 注意：如果你锁定了文件的尾部，而这个文件的长度随后增长超过了锁定的部分，那么增长出来的额外区域是不锁定的，要想锁定所有的字节，可以使用`Long.MAX_VALUE`来表示尺寸。

请记住，文件加锁机制是依赖于操作系统的，下面是需要注意的几点：

- 在某些系统中，文件加锁仅仅是建议性的，如果一个应用未能得到锁，它仍旧可以向被另一个应用并发锁定的文件执行写操作。
- 在某些系统中，不能在锁定一个文件的同时将其映射到内存中。
- 文件锁是由整个Java虚拟机持有的。如果有两个程序是由同一个虚拟机启动的（例如Applet和应用程序启动器），那么它们不可能每一个都获得一个在同一个文件上的锁。当调用`lock`和`tryLock`方法时，如果虚拟机已经在同一个文件上持有另一个重叠的锁，那么这两个方法将抛出`OverlappingFileLockException`。
- 在一些系统中，关闭一个通道会释放由Java虚拟机持有的潜在文件上的所有锁。因此，在同一个加锁文件上应避免使用多个锁。
- 在网络化的文件系统上锁定文件是高度依赖于系统的，因此应该尽量避免。

#### `java.nio.channels.FileChannel` 1.4

- `FileLock lock()`

在整个文件上获得一个独占的锁，这个方法将阻塞直至获得锁。

- `FileLock tryLock()`

在整个文件上获得一个独占的锁，或者在无法获得锁的情况下返回`null`。

- `FileLock lock(long position, long size, boolean shared)`

- `FileLock tryLock(long position, long size, boolean shared)`

在文件的一个区域上获得锁。第一个方法将阻塞直至获得锁，而第二个方法将在无法获得锁时返回`null`。

参数：`position` 要锁定区域的起始位置

`size` 要锁定区域的尺寸

`shared` `true`为共享锁，`false`为独占锁

**API** `java.nio.channels.FileLock 1.4`

- `void release()`

释放这个锁。

## 1.8 正则表达式

正则表达式 (regular expression) 用于指定字符串的模式，你可以在任何需要定位匹配某种特定模式的字符串的情况下使用正则表达式。例如，我们有一个示例程序就是用来定位HTML文件中的所有超级链接的，它是通过查找`<a href = "...>`模式的字符串来实现此目的的。

当然，对于指定模式来说，`...`标记法并不够精确。你需要精确地指定什么样的字符序列才是合法的匹配，这就要求无论何时，当你要描述一个模式时，都需要使用某种特定的语法。

下面是一个简单的示例，正则表达式

`[Jj]ava.+`

匹配下列形式的所有字符串：

- 第一个字母是J或j。
- 接下来的三个字母是ava。
- 字符串的其余部分由一个或多个任意的字符构成。

例如，字符串“javane”就匹配这个特定的正则表达式，但是字符串“core java”就不匹配。

正如你所见，你需要了解一点这种语法，以理解正则表达式的含义。幸运的是，对于大多数情况，一小部分很直观的语法结构就足够用了。

- 字符类 (character class) 是一个括在括号中的可选择的字符集，例如，`[Jj]`、`[0-9]`、`[A-Za-z]`或`[^0-9]`。这里—表示是一个范围（所有Unicode值落在两个边界范围之内的字符），而`^`表示补集（除了指定字符之外的所有字符）。
- 有许多预定的字符类，例如`\d`（数字）和`\p{Sc}`（Unicode货币符号）。请查看表1-7和表1-8。

表1-7 正则表达式语法

| 语 法   | 解 释  |
|---|--|
| 字符  |  |
| <code>c</code>                                | 字符c  |
| <code>\unnnnn, \xnnn, \0n, \0nn, \0nnn</code> | 具有给定十六进制或十进制值的码元   |
| <code>\t, \n, \r, \f, \a, \e</code>           | 控制符：制表符、换行符、回车符、换页符、警告符、逃逸符  |
| <code>\cc</code>                              | 与字符c相关的控制符   |
| 字符类   |  |
| <code>[C<sub>1</sub>C<sub>2</sub>...]</code>  | 任何由C <sub>1</sub> 、C <sub>2</sub> ，...表示的字符，其中C <sub>i</sub> 可以是多个字符、字符范围(c <sub>1</sub> -c <sub>2</sub> )和字符类 |
| <code>[^...]</code>                           | 字符类的补集   |
| <code>[... &amp; ...]</code>                  | 两个字符集的交集   |

(续)

## 语 法

## 解 释

## 预定义字符类

除了行终止符之外的所有字符（在DOTALL标志被设置时，则表示所有字符）

|          |                      |
|----------|----------------------|
| \d       | 一个数字[0-9]            |
| \D       | 一个非数字[^0-9]          |
| \s       | 一个空白字符[\t\n\r\f\x0B] |
| \S       | 一个非空白字符              |
| \w       | 一个词语字符[a-zA-Z0-9_]   |
| \W       | 一个非词语字符              |
| \p{name} | 一个命名字符类——请查看表1-8     |
| \P{name} | 一个命名字符类的补集           |

## 边界匹配符

|      |                           |
|------|---------------------------|
| ^ \$ | 输入的开头和结尾（或者在多行模式下行的开头和结尾） |
| \b   | 一个词语边界                    |
| \B   | 一个非词语边界                   |
| \A   | 输入的开头                     |
| \Z   | 输入的结尾                     |
| \z   | 除了行终止符之外的输入结尾             |
| \G   | 前一个匹配的结尾                  |

## 量词

|                    |                  |
|--------------------|------------------|
| X?                 | 可选的X             |
| X*                 | X, 0或多次          |
| X+                 | X, 1或多次          |
| X{n} X{n,} X{n, m} | Xn次，至少n次，在n到m次之间 |

## 量词后缀

|   |                  |
|---|------------------|
| ? | 将默认（贪婪）匹配转变为勉强匹配 |
| + | 将默认（贪婪）匹配转变为占有匹配 |

## 集合操作

|     |                       |
|-----|-----------------------|
| XY  | 任何X中的字符串，后面跟随任何Y中的字符串 |
| X Y | 任何X或Y中的字符串            |

## 群组

|              |                   |
|--------------|-------------------|
| (X)          | 捕获将X作为群组匹配的字符串    |
| \\$ <i>n</i> | 第 <i>n</i> 个群组的匹配 |

## 转义

|           |                         |
|-----------|-------------------------|
| \c        | 字符c（必须是不在字母表中的字符）       |
| \Q ... \E | 逐字地引用...                |
| (? ... )  | 特殊结构——请查看Pattern类的API注释 |

表1-8 预定义的字符类名字

| 字符类名字               | 解 释   |
|---------------------|---|
| Lower               | ASCII的小写字母[a-z]   |
| Upper               | ASCII的大写字母[A-Z]   |
| Alpha               | ASCII的字母[A-Za-z]  |
| Digit               | ASCII的数字[0-9]   |
| Alnum               | ASCII的字母或数字[A-Za-z0-9]  |
| XDigit              | 十六进制数字[0-9A-Fa-f]   |
| Print或Graph         | 可打印的ASCII字符[\x21-\x7E]  |
| Punct               | ASCII的非字母和数字字符[\p{Print} && \P{Alnum}]  |
| ASCII               | 所有ASCII字符[\x00-\x7F]  |
| Cntrl               | ASCII的控制字符[\x00-\x1F]   |
| Blank               | 空格字符或制表符[\t]  |
| Space               | 空白字符[\t\n\r\f\0x0B]   |
| javaLowerCase       | 小写字母，正如Character.isLowerCase()确定的字符   |
| javaUpperCase       | 大写字母，正如Character.isUpperCase()确定的字符   |
| JavaWhitespace      | 空白字符，正如Character.isWhitespace()确定的字符  |
| javaMirrored        | 镜像字符，正如Character.isMirrored()确定的字符  |
| InBlock             | Block是Unicode字符块的名字，不过要剔除名字中的空格，例如BasicLatin和Mongolian。请查看 <a href="http://www.unicode.org">http://www.unicode.org</a> 以了解字符块名字列表 |
| Category或InCategory | Category是Unicode字符类别的名字，例如，L（字母）和Sc（货币符号）。请查看 <a href="http://www.unicode.org">http://www.unicode.org</a> 以了解字符类别名字列表             |

- 大部分字符都可以与它们自身匹配，例如在前面示例中的ava字符。
- 符号可以匹配任何字符（有可能不包括行终止符，这取决于标志的设置）。
- 使用\作为转义字符，例如，\.匹配句号而\\匹配反斜线。
- ^和\$分别匹配一行的开头和结尾。
- 如果X和Y是正则表达式，那么XY表示“任何X的匹配后面跟随Y的匹配”，X|Y表示“任何X或Y的匹配”。
- 你可以将量词运用到表达式X：X+（1个或多个）、X\*（0个或多个）与X?（0个或1个）。
- 默认情况下，量词要匹配能够使整个匹配成功的最大可能的重复次数。你可以修改这种行为，方法是使用后缀？（使用勉强或吝啬匹配，也就是匹配最小的重复次数）或使用后缀+（使用占有或贪婪匹配，也就是即使让整个匹配失败，也要匹配最大的重复次数）。

例如，字符串cab匹配[a-z]\*ab，但是不匹配[a-z]++ab。在第一种情况中，表达式[a-z]\*只匹配字符c，使得字符ab匹配模式的剩余部分；但是贪婪版本[a-z]++将匹配字符cab，模式的剩余部分将无法匹配。

- 我们使用群组来定义子表达式，其中群组用括号()括起来。例如，([+-]?)([0-9]+)。然后你可以询问模式匹配器，让其返回每个组的匹配，也可以用\ n来引用某个群组，其中n是群组号（从\1开始）。

例如，下面是一个有些复杂但是却可能很有用的正则表达式，它描述了十进制和十六进制整数：

[+-]?[0-9]+|[0][Xx][0-9A-Fa-f]+

遗憾的是，在使用正则表达式的各种程序和类库之间，表达式语法并未完全标准化。尽管在基本结构上达成了一致，但是它们在细节上仍旧存在着许多令人抓狂的差异。Java正则表达式类使用的语法与Perl语言使用的语法十分相似，但是并不完全一样。表1-7展示了Java语法中的所有结构。关于正则表达式语法的更多信息，可以求教于Pattern类的API文档和Jeffrey E. F. Friedl的《Mastering Regular Expressions》(O'Reilly and Associates, 1997)。

正则表达式的最简单用法就是测试某个特定的字符串是否与它匹配。下面展示了如何用Java来编写这种测试，首先用表示正则表达式的字符串构建一个Pattern对象。然后从这个模式中获得一个Matcher，并调用它的matches方法：

```
Pattern pattern = Pattern.compile(patternString);
Matcher matcher = pattern.matcher(input);
if (matcher.matches()) . . .
```

这个匹配器的输入可以是任何实现了CharSequence接口的类的对象，例如String、StringBuilder和CharBuffer。

在编译这个模式时，你可以设置一个或多个标志，例如：

```
Pattern pattern = Pattern.compile(patternString,
    Pattern.CASE_INSENSITIVE + Pattern.UNICODE_CASE);
```

下面是所支持的六个标志：

- CASE\_INSENSITIVE：匹配字符时忽略字母的大小写，默认情况下，这个标志只考虑US ASCII字符。
- UNICODE\_CASE：当CASE\_INSENSITIVE组合时，用Unicode字母大小写来匹配。
- MULTILINE：^和\$匹配行的开头和结尾，而不是整个输入的开头和结尾。
- UNIX\_LINES：在多行模式中匹配^和\$时，只有'\n'被识别成行终止符。
- DOTALL：当使用这个标志时，.符号匹配所有字符，包括行终止符。
- CANON\_EQ：考虑Unicode字符规范的等价性，例如，u后面跟随”（分音符号）匹配ü。

如果正则表达式包含群组，那么Matcher对象可以揭示群组的边界。下面的方法

```
int start(int groupIndex)
int end(int groupIndex)
```

将产生指定群组的开始索引和结束之后的索引。

你可以直接通过调用下面的方法抽取匹配的字符串

```
String group(int groupIndex)
```

群组0是整个输入，而用于第一个实际群组的群组索引是1。调用groupCount方法可以获得全部群组的数量。

嵌套群组是按照前括号排序的，例如，假设我们有下面的模式

```
((1?[0-9]):([0-5][0-9]))[ap]m
```

和下面的输出

11:59am

那么，匹配器会报告下面的群组

| 群组索引 | 开始 | 结束 | 字符串     |
|------|----|----|---------|
| 0    | 0  | 7  | 11:59am |

```
1      0      5      11:59
2      0      2      11
3      3      5      59
```

程序清单1-8的程序提示输入一个模式，然后提示输入用于匹配的字符串，随后将打印出输入是否与模式相匹配。如果输入匹配模式，并且模式包含群组，那么这个程序将用括号打印出群组边界，例如

```
((11):(59))am
```

### 程序清单1-8    RegexTest.java

```
1 import java.util.*;
2 import java.util.regex.*;
3
4 /**
5 * This program tests regular expression matching.
6 * Enter a pattern and strings to match, or hit Cancel
7 * to exit. If the pattern contains groups, the group
8 * boundaries are displayed in the match.
9 * @version 1.01 2004-05-11
10 * @author Cay Horstmann
11 */
12 public class RegExTest
13 {
14     public static void main(String[] args)
15     {
16         Scanner in = new Scanner(System.in);
17         System.out.println("Enter pattern: ");
18         String patternString = in.nextLine();
19
20         Pattern pattern = null;
21         try
22         {
23             pattern = Pattern.compile(patternString);
24         }
25         catch (PatternSyntaxException e)
26         {
27             System.out.println("Pattern syntax error");
28             System.exit(1);
29         }
30
31         while (true)
32         {
33             System.out.println("Enter string to match: ");
34             String input = in.nextLine();
35             if (input == null || input.equals("")) return;
36             Matcher matcher = pattern.matcher(input);
37             if (matcher.matches())
38             {
39                 System.out.println("Match");
40                 int g = matcher.groupCount();
41                 if (g > 0)
42                 {
43                     for (int i = 0; i < input.length(); i++)
```

```

44         {
45             for (int j = 1; j <= g; j++)
46                 if (i == matcher.start(j))
47                     System.out.print('(');
48             System.out.print(input.charAt(i));
49             for (int j = 1; j <= g; j++)
50                 if (i + 1 == matcher.end(j))
51                     System.out.print(')');
52         }
53     System.out.println();
54 }
55 }
56 else
57     System.out.println("No match");
58 }
59 }
60 }

```

通常，你不希望用正则表达式来匹配全部输入，而只是想找出输入中一个或多个匹配的字符串。这时可以使用Matcher类的find方法来查找下一个匹配，如果它返回true，再使用start和end方法来查找匹配的内容。

```

while (matcher.find())
{
    int start = matcher.start();
    int end = matcher.end();
    String match = input.substring(start, end);
    ...
}

```

程序清单1-9对这种机制进行了应用，它定位一个Web页面上的所有超文本引用，并打印它们。为了运行这个程序，你需要在命令行中提供一个URL，例如

```
java HrefMatch http://www.horstmann.com
```

### 程序清单1-9 HrefMatch.java

```

1. import java.io.*;
2. import java.net.*;
3. import java.util.regex.*;
4.
5. /**
6. * This program displays all URLs in a web page by matching a regular expression that describes
7. * the <a href=...> HTML tag. Start the program as <br>
8. * java HrefMatch URL
9. * @version 1.01 2004-06-04
10. * @author Cay Horstmann
11. */
12. public class HrefMatch
13. {
14.     public static void main(String[] args)
15.     {
16.         try
17.         {
18.             // get URL string from command line or use default

```

```

19.     String urlString;
20.     if (args.length > 0) urlString = args[0];
21.     else urlString = "http://java.sun.com";
22.
23.     // open reader for URL
24.     InputStreamReader in = new InputStreamReader(new URL(urlString).openStream());
25.
26.     // read contents into string builder
27.     StringBuilder input = new StringBuilder();
28.     int ch;
29.     while ((ch = in.read()) != -1)
30.         input.append((char) ch);
31.
32.     // search for all occurrences of pattern
33.     String patternString = "<a\\s+href\\s*=\\s*(\"[^\"\\n]+|[\\n\\s]+\")\\s*>";
34.     Pattern pattern = Pattern.compile(patternString, Pattern.CASE_INSENSITIVE);
35.     Matcher matcher = pattern.matcher(input);
36.
37.     while (matcher.find())
38.     {
39.         int start = matcher.start();
40.         int end = matcher.end();
41.         String match = input.substring(start, end);
42.         System.out.println(match);
43.     }
44. }
45. catch (IOException e)
46. {
47.     e.printStackTrace();
48. }
49. catch (PatternSyntaxException e)
50. {
51.     e.printStackTrace();
52. }
53. }
54. }

```

Matcher类的replaceAll方法将正则表达式出现的所有地方都用替换字符串来替换。例如，下面的指令将所有的数字序列都替换成#字符。

```

Pattern pattern = Pattern.compile("[0-9]+");
Matcher matcher = pattern.matcher(input);
String output = matcher.replaceAll("#");

```

替换字符串可以包含对模式中群组的引用：\$n表示替换成第n个群组，因此我们需要用\\$来表示在替换文本中包含一个\$字符。

replaceFirst方法将只替换模式的第一次出现。

最后，Pattern类有一个split方法，它可以用正则表达式来匹配边界，从而将输入分割成字符串数组。例如，下面的指令可以将输入分割成标号，其中分隔符是由可选的空白字符包围的标点符号。

```

Pattern pattern = Pattern.compile("\\s*\\p{Punct}\\s*");
String[] tokens = pattern.split(input);

```

**API** **java.util.regex.Pattern 1.4**

- static Pattern compile(String expression)
- static Pattern compile(String expression, int flags)

把正则表达式字符串编译到一个用于快速处理匹配的模式对象中。

参数: expression 正则表达式

flags CASE\_INSENSITIVE、UNICODE\_CASE、MULTILINE、  
UNIX\_LINES、DOTALL和CANON\_EQ标志中的一个

- Matcher matcher(CharSequence input)

返回一个matcher对象，你可以用它在输入定位模式的匹配。

- String[] split(CharSequence input)

- String[] split(CharSequence input, int limit)

将输入分割成标号，其中模式指定了分隔符的形式。返回标号数组，分隔符并非标号的一部分。

参数: input 要分割成标号的字符串

limit 所产生的字符串的最大数量。如果已经发现了limit - 1个匹配的分隔符，那么返回的数组中的最后一项就包含所有剩余未分割的输入。如果limit≤0，那么这个输入都被分割；如果limit为0，那么坠尾的空字符串将不会置于返回的数组中

**API** **java.util.regex.Matcher 1.4**

- boolean matches()

如果输入匹配模式，则返回true。

- boolean lookingAt()

如果输入的开头匹配模式，则返回true。

- boolean find()

- boolean find(int start)

尝试查找下一个匹配，如果找到了另一个匹配，则返回true。

参数: start 开始查找索引位置

- int start()

- int end()

返回当前匹配的开始索引和结尾之后的索引。

- String group()

返回当前的匹配。

- int groupCount()

返回输入模式中的群组数量。

- int start(int groupIndex)

- int end(int groupIndex)

返回当前匹配中给定群组的开始和结尾之后的位置。

参数: groupIndex 群组索引 (从1开始), 或者表示整个匹配的0

- `String group(int groupIndex)`

返回匹配给定群组的字符串。

参数: groupIndex 群组索引 (从1开始), 或者表示整个匹配的0。

- `String replaceAll(String replacement)`

- `String replaceFirst(String replacement)`

返回从匹配器输入获得的通过将所有匹配或第一个匹配用替换字符串替换之后的字符串。

参数: replacement 替换字符串, 它可以包含用\$*n*表示的对群组的引用, 这时需要用\\$来表示包括一个\$符号

- `Matcher reset()`

- `Matcher reset(CharSequence input)`

复位匹配器的状态。第二个方法将使匹配器作用于另一个不同的输入。这两个方法都返回this。

你现在已经看到了在Java中输入输出操作是如何实现的, 也对作为“新I/O”包一部分的正则表达式有了概略的了解。在下一章中, 我们将转而研究XML数据的处理。

# 第2章 XML

- ▲ XML概述
- ▲ 解析XML文档
- ▲ 验证XML文档
- ▲ 使用XPath来定位信息
- ▲ 使用命令空间
- ▲ 流机制解析器
- ▲ 生成XML文档
- ▲ XSL转换

Don Box等人在其合著的《*Essential XML*》(Addison-Wesley出版社2000年出版)的前言中半开玩笑地说道：“可扩展标记语言(Extensible Markup Language, XML)已经取代Java、设计模式、对象技术成为软件行业解决世界饥荒的方案。”确实，正如你将在本章看到的，XML是一个非常有用的数据结构化信息的技术。XML工具使处理和转化信息十分容易。但是，XML并不是银弹。需要特定领域的标准和代码库才能有效地使用XML。此外，XML非但没有使Java技术过时，XML恰恰与Java配合得很好。从20世纪90年代末以来，IBM、Apache和其他许多公司一直在帮助开发用于XML处理的高质量Java库。从JDK SE 1.4开始，Sun公司将大部分重要的代码库都整合到了Java平台标准版中。

本章将介绍XML，并涵盖了Java库的XML特性。一如既往，我们将指出何时大量地使用XML是正确的；而何时必须有保留地使用XML，通过利用良好的设计和代码，来采用老办法解决问题。

## 2.1 XML概述

在第I卷第10章中，你已经看见过用属性文件(property file)来描述程序配置。属性文件包含了一组名/值对，例如：

```
fontname=Times Roman  
fontsize=12  
windowsize=400 200  
color=0 50 100
```

你可以在一个Properties类中仅使用一个方法调用即可读入这个属性文件。这是一个很好的特性，但这还不够。在许多情况下，想要描述的信息的结构比较复杂，属性文件不能很方便地处理它。例如下面例子中的fontname/fontsize项，使用以下的单一项将更符合面向对象的要求：

```
font=Times Roman 12
```

但是，这时对字体描述的分析就变得很讨厌了，必须确定字体名何时结束，字体大小何时开始。

属性文件采用的是一种简单的平面层次结构。你常常会看到程序员用如下的键名来努力解

解决这种局限性：

```
title.fontname=Helvetica  
title fontsize=36  
body.fontname=Times Roman  
body fontsize=12
```

属性文件格式的另一个缺点是由键的惟一性引起的。要存放一连串的值，你需要另一个变通方法，例如：

```
menu.item.1=Times Roman  
menu.item.2=Helvetica  
menu.item.3=Goudy Old Style
```

XML格式解决了这些问题，因为它能够表示层次结构，这比属性文件的平面表结构更灵活。描述程序配置的XML文件可能会像这样：

```
<configuration>  
  <title>  
    <font>  
      <name>Helvetica</name>  
      <size>36</size>  
    </font>  
  </title>  
  <body>  
    <font>  
      <name>Times Roman</name>  
      <size>12</size>  
    </font>  
  </body>  
  <>window>  
    <width>400</width>  
    <height>200</height>  
  </window>  
  <color>  
    <red>0</red>  
    <green>50</green>  
    <blue>100</blue>  
  </color>  
  <menu>  
    <item>Times Roman</item>  
    <item>Helvetica</item>  
    <item>Goudy Old Style</item>  
  </menu>  
</configuration>
```

XML格式能够表达层次结构和重复的元素而不会被曲解。

正如上面看到的，XML文件的格式非常直观，它与HTML文件非常相似。这完全是有理由的，因为XML和HTML格式是古老的标准通用标记语言（Standard Generalized Markup Language, SGML）的衍生语言。

SGML从20世纪70年代开始就用于描述复杂文件的结构。它的使用在一些要求对海量文献进行持续维护的产业中取得了成功，特别是在飞机制造业中。但是，SGML相当复杂，所以它从未风行。造成SGML如此复杂的主要原因是SGML有两个相互矛盾的目标。它既想要确保文档能够根据其文档类型的规则来形成，又想要通过减少数据键入的快捷方式使数据的输入能够

简单易行。XML设计成了一个使用于因特网上SGML的简化版本。和通常情况一样，越简单的东西越好，XML立即得到了长期以来一直在躲避SGML的用户的热情采用。

 **注意：**在<http://www.xml.com/axml/axml.html>可以找到一个由Tim Bray注释的XML标准的极佳版本。

尽管HTML和XML来源相同，但是两者之间存在着重要的区别。

- 与HTML不同，XML是大小写敏感的。例如，`<H1>`和`<h1>`是不同的XML标签。
- 在HTML中，如果从上下文可以分清哪里是段落或列表项的结尾，那么结束标签（如`</p>`或`</li>`）就可以省略，而在XML中结束标签绝对不能省略。
- 在XML中，只有一个标签而没有相对应的结束标签的元素必须以/结尾，比如``。这样，解析器就知道不需要查找`</img>`标签了。
- 在XML中，属性值必须用引号括起来。在HTML中，引号是可有可无的。例如，`<applet code="MyApplet.class" width=300 height=300>`对HTML来说是合法的，但是对XML来说则是不合法的。在XML中，必须使用引号，比如，`width="300"`。
- 在HTML中，属性名可以没有值。例如，`<input type="radio" name="language" value="Java" checked>`。在XML中，所有属性必须都有属性值。比如，`checked="true"`或`checked="checked"`。

 **注意：**目前由万维网协会（W3C）推荐的网络文档是基于XHTML标准的，该标准强化了HTML标准，使之能够符合XML标准。你可以在<http://www.w3.org/TR/xhtml1/>找到XHTML标准的拷贝。XHTML与当前的浏览器是向后兼容，但是并非所有的HTML编写工具都支持它。随着XHTML变得更加普及，您就可以使用在本章所描述的XML工具来分析网络文档了。

## XML文档的结构

XML文档应当以一个文档头开始，例如：

```
<?xml version="1.0"?>
```

或者

```
<?xml version="1.0" encoding="UTF-8"?>
```

严格说来，文档头是可有可无的，但是强烈推荐你使用文档头。

 **注意：**因为建立SGML是为了处理真正的文档，因此XML文件叫做文档，尽管许多XML文件是用来描述通常不称为文档的数据集的。

文档头之后通常是文档类型定义（Document Type Definition，DTD），例如：

```
<!DOCTYPE web-app PUBLIC  
        "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"  
        "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
```

文档类型定义是确保文档正确的一个重要机制，但是这不是必需的。我们将在本章的后面讨论这个问题。

最后，XML文档的正文包含根元素，根元素包含其他一些元素。例如：

```
<?xml version="1.0"?>
<!DOCTYPE configuration . . .>
<configuration>
    <title>
        <font>
            <name>Helvetica</name>
            <size>36</size>
        </font>
    </title>
    . . .
</configuration>
```

元素可以有子元素（child element）、文本或两者皆有。在上述例子中，`font`元素有两个子元素，它们是`name`和`size`。`name`元素包含文本“Helvetica”。

**!** 提示：在设计XML文档结构时，最好使元素要么包含子元素，要么包含文本。换句话说，你应该避免以下情况：

```
<font>
    Helvetica
    <size>36</size>
</font>
```

在XML规范中，这叫做混合式内容（mixed content）。本章中，稍后你将会看到，如果避免了混合式内容，就可以简化解析过程。

XML元素可以包含属性，例如：

```
<size unit="pt">36</size>
```

何时用元素，何时用属性，在XML设计人员中存在一些分歧。例如，将`font`做如下描述：

```
<font name="Helvetica" size="36"/>
```

似乎比下面更简单一些：

```
<font>
    <name>Helvetica</name>
    <size>36</size>
</font>
```

但是，属性的灵活性要差很多。假设你想把单位添加到`size`的值中去，如果使用属性，那么必须把单位添加到属性值中去：

```
<font name="Helvetica" size="36 pt"/>
```

嗨！现在必须对字符串“36 pt”进行解析，而这正是XML设计用来避免的那种麻烦。把属性加到`size`元素中则简单多了：

```
<font>
    <name>Helvetica</name>
    <size unit="pt">36</size>
</font>
```

一个通常的经验法则是，属性只应该在修改值的解释时使用，而不是在指定值时使用。如果你发现自己卷入了关于某个设置是否是对某个值的解释所作的修改的讨论之中，那么你就应该使用属性，而应该完全使用元素。许多有用的文档根本就不使用属性。

 注意：在HTML中属性的使用规则很简单：凡是不显示在网页上的都是属性。例如以下的超链接：

```
<a href="http://java.sun.com">Java Technology</a>
```

字符串Java Technology要在网页上显示，但是这个链接的URL并不是显示页面的一部分。然而，这个规则对于大多数XML并不那么管用，因为XML文件中的数据并非像通常意义那样是让人浏览的。

元素和文本是XML文档“主要的支撑要素”，以下是你会遇到的其他一些标记的说明：

- 字符引用 (character reference) 的形式是&#十进制值；或&#x十六进制值；。例如，字符é可以用下面两种形式表示：

```
&#233;  
&#xD9;
```

- 实体引用 (entity reference) 的形式是&name;。下面这些实体引用：

```
&lt;  
&gt;  
&amp;  
&quot;  
&apos;
```

都有预定义的含义：小于、大于、&、引号、省略号等字符。可以在DTD中定义其他的实体引用。

- CDATA部分 (CDATA Section) 用<![CDATA[和]]>来限定其界限。它们是字符数据的一种特殊形式。你可以使用它们来包含那些含有<、>、&之类字符的字符串，而不必将它们解释为标记，例如：

```
<![CDATA[< &gt; are my favorite delimiters]]>
```

CDATA部分不能包含字符串]]>。使用这一特性时要特别小心。它常用来当作将传统数据偷偷纳入XML文档的一个后门。

- 处理指令 (processing instruction) 是那些专门在处理XML文档的应用程序中使用的指令，它们将用<?和?>来限定其界限，例如：

```
<?xml-stylesheet href="mystyle.css" type="text/css"?>
```

每个XML都以一个处理指令开头：

```
<?xml version="1.0"?>
```

- 注释 (comment) 用<!--和-->限定其界限，例如：

```
<!-- This is a comment. -->
```

注释不能含有字符串--。注释只是为了给文档的读者提供信息，其中绝不含有隐藏的命令，命令是由处理指令来实现的。

## 2.2 解析XML文档

要处理XML文档，就要先解析 (parse) 它。解析器是这样一个程序，它读入一个文件，确认这个文件具有正确的格式，然后将其分解成各种元素，使得程序员能够得到这些元素。

Java库提供了两种XML解析器：

- 像文档对象模型（Document Object Model, DOM）解析器这样的树型解析器（tree parser），它们将读入的XML文档转换成树结构。
- 像用于XML的简单API（Simple API for XML, SAX）解析器这样的流机制解析器（streaming parser），它们在读入XML文档时生成相应的事件。

DOM解析器对于实现我们的大多数目的来说都很容易，所以我们首先介绍它。如果你要处理很长的文档，用它生成树结构将会消耗大量内存，或者如果你只是对于某些元素感兴趣，而不关心它们的上下文，那么在这些情况下你应该考虑使用流机制解析器。更多的信息可以查看第2.6节。

DOM解析器的接口已经被W3C标准化了。`org.w3c.dom`包包含了接口类型的定义，比如：`Document`和`Element`等。不同的提供者，比如Apache Organization和IBM都编写了实现这些接口的DOM解析器。Sun公司的XML处理Java API（Java API for XML Processing, JAXP）库实际上可以插入到这些解析器中的任意一个中。但是Sun公司也在Java SDK中包含了自己的DOM解析器。在本章中，我们使用的就是Sun公司的解析器。

要读入一个XML文档，首先需要一个`DocumentBuilder`对象，你可以从`DocumentBuilderFactory`中得到这个对象，例如：

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
```

现在，可以从文件中读入某个文档：

```
File f = ...;
Document doc = builder.parse(f);
```

或者，可以用一个URL：

```
URL u = ...;
Document doc = builder.parse(u);
```

甚至可以指定一个任意的输入流：

```
InputStream in = ...;
Document doc = builder.parse(in);
```



**注意：**如果你使用输入流作为输入源，那么对于那些通过相对与文档的位置关系来引用的文件，解析器将无法定位，比如在同一个目录中的DTD。但是，你可以通过安装一个“实体分解器（entity resolver）”来解决这个问题。

`Document`对象是XML文档的树型结构在内存中的表现。它由实现`Node`接口及其多个子接口的类的对象构成。图2-1显示了各个子接口的层次结构。

可以通过调用`getDocumentElement`方法来分析文档的内容，它将返回根元素。

```
Element root = doc.getDocumentElement();
```

例如，你要处理下面的文档：

```
<?xml version="1.0"?>
<font>
```

```
...
```

```
</font>
```

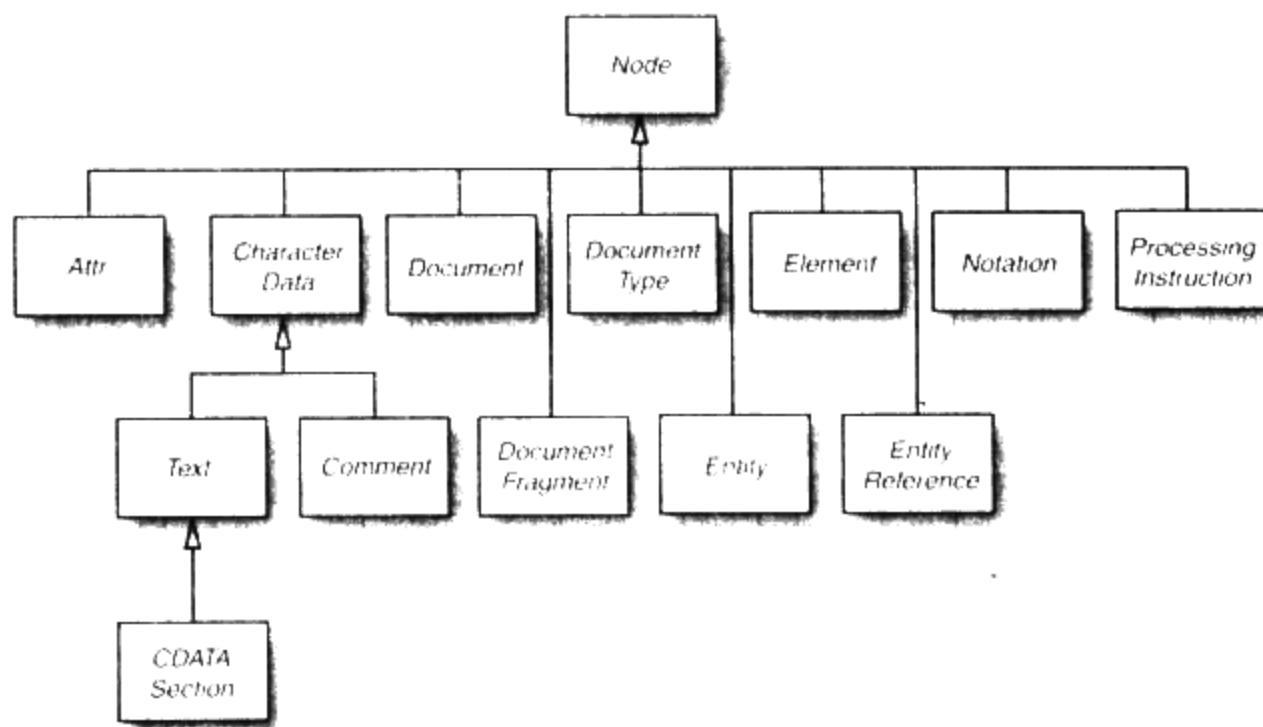


图2-1 Node接口及其子接口

那么，调用`getDocumentElement`方法可以返回`font`元素。`getTagName`方法可以返回元素的标签名。在前面这个例子中，`root.getTagName()`返回字符串"font"。

如果要得到该元素的子元素（可以是子元素、文本、注释或其他节点），请使用`getChildNodes`方法。这个方法返回一个类型为`NodeList`的集合。这个类型在标准的Java集合类创建之前就建立了，它有一个与众不同的访问协议。`item`方法将得到指定索引号的项。`getLength`方法则提供了项的总数，你可以像这样枚举所有子元素：

```

NodeList children = root.getChildNodes();
for (int i = 0; i < children.getLength(); i++)
{
    Node child = children.item(i);
    ...
}
  
```

分析子元素时要很仔细。例如，假设你正在处理以下文档：

```

<font>
  <name>Helvetica</name>
  <size>36</size>
</font>
  
```

你期望`font`有两个子元素，但是解析器却报告说有5个：

- `<font>`和`<name>`之间的空白字符
- `name`元素
- `</name>`和`<size>`之间的空白字符
- `size`元素

- </size>和</font>之间的空白字符

图2-2显示了其DOM树。

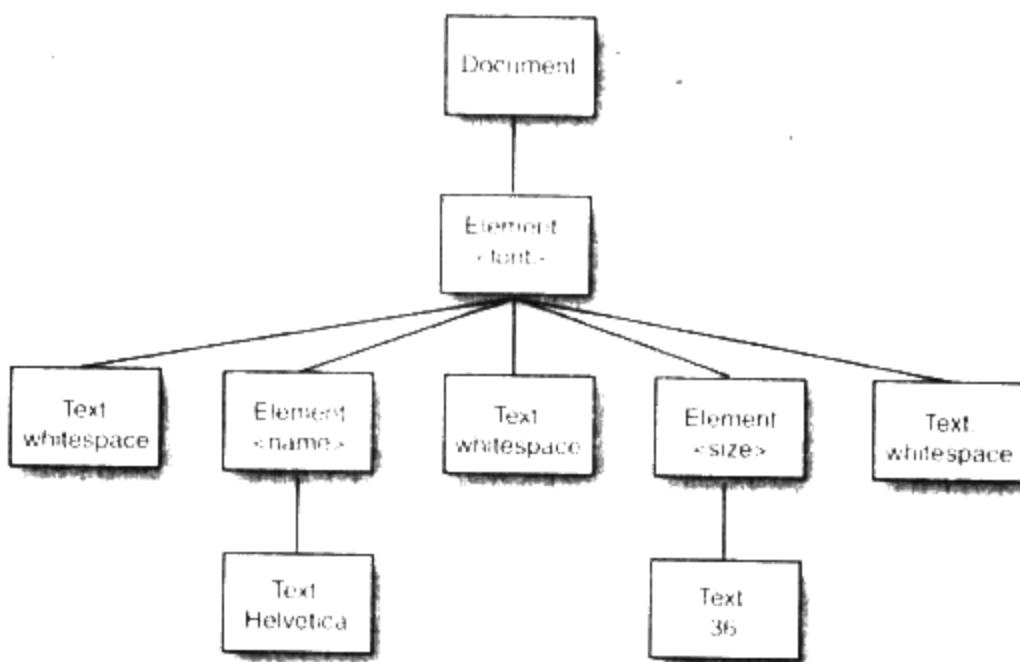


图2-2 一棵简单的DOM树

如果只希望得到子元素，那么可以忽略空白字符：

```

for (int i = 0; i < children.getLength(); i++)
{
    Node child = children.item(i);
    if (child instanceof Element)
    {
        Element childElement = (Element) child;
        ...
    }
}
  
```

现在，只会看到两个元素，它们的标签名是name和size。

正如你将在下一节中所看到的那样，如果你的文档有DTD，那么你就可以做得更好。这时，解析器知道哪些元素没有文本节点的子元素，而且它会帮你禁止掉空白字符。

当你分析name和size元素时，想检索到它们包含的文本字符串。这些文本字符串本身都包含在Text类型的子节点中。既然知道了这些Text节点是唯一的子元素，就可以用getFirstChild方法而不用再遍历一个NodeList。然后可以用getData方法检索存储在Text节点中的字符串。

```

for (int i = 0; i < children.getLength(); i++)
{
    Node child = children.item(i);
    if (child instanceof Element)
    {
        Element childElement = (Element) child;
        Text textNode = (Text) childElement.getFirstChild();
        String text = textNode.getData().trim();
        if (childElement.getTagName().equals("name"))
  
```

```

        name = text;
    else if (childElement.getTagName().equals("size"))
        size = Integer.parseInt(text);
    }
}

```

**!** 提示：对getData的返回值调用trim方法是个好主意。如果XML文件的作者将起始和结束的标签放在不同的行上，例如：

```

<size>
    36
</size>

```

那么，解析器将会把所有的换行符和空格都包含到文本节点中去。调用trim方法可以把实际数据前后的空白字符删掉。

也可以用getLastChild方法得到最后一项子元素，用getNextSibling得到下一个兄弟节点。这样，另一种遍历子节点集的方法就是：

```

for (Node childNode = element.getFirstChild();
    childNode != null;
    childNode = childNode.getNextSibling())
{
    ...
}

```

如果要枚举节点的属性，请调用getAttributes方法。它返回一个NamedNodeMap对象，其中包含了描述属性的节点对象。可以用和遍历NodeList一样的办法在NamedNodeMap中遍历各个节点。调用getNodeName和getNodeValue方法可以得到属性名和属性值。

```

NamedNodeMap attributes = element.getAttributes();
for (int i = 0; i < attributes.getLength(); i++)
{
    Node attribute = attributes.item(i);
    String name = attribute.getNodeName();
    String value = attribute.getNodeValue();
    ...
}

```

或者，如果知道属性名，则可以直接得到相应的属性值：

```
String unit = element.getAttribute("unit");
```

现在你已经知道怎么分析DOM树了。程序清单2-1中的程序将这些技术都运用了一遍。你可以使用File->Open菜单选项来读入一个XML文件。DocumentBuilder对象会解析这个XML文件，并产生一个Document对象。该程序将Document对象显示为一个JTree（参见图2-3）。

该树形结构清楚地显示了子元素是怎样被空白字符和注释包围起来的。为了更清楚起见，这个程序显示了换行和回车字符，即\n和\r。（否则，它们将显示为空的文本框，这是Swing在字符串中不能绘制的字符的默认符号）。

在第6章你将会学习到该程序中用来显示树形结构和属性表的技术。DOMTreeModel类实现了treeModel接口。getRoot方法返回文档的根元素，getChild方法得到子元素的节点列表，返回被请求的索引号的项。表的单元格渲染器显示了以下内容：



图2-3 一个XML文档的解析树

- 对元素来说，显示的是所有标签名和由所有的属性构成的一张表。
- 对字符数据来说，显示的是界面（文本、注释、CDATA部分），后面跟随数据，其中换行和回车字符将被\n和\r取代。
- 对其他所有的节点类型来说，显示的是类名，后面跟随toString的结果。

#### 程序清单2-1 DOMTreeTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.io.*;
4. import javax.swing.*;
5. import javax.swing.event.*;
6. import javax.swing.table.*;
7. import javax.swing.tree.*;
8. import javax.xml.parsers.*;
9. import org.w3c.dom.*;
10.
11. /**
12. * This program displays an XML document as a tree.
13. * @version 1.11 2007-06-24
14. * @author Cay Horstmann
15. */
16. public class DOMTreeTest
17. {
18.     public static void main(String[] args)
19.     {
20.         EventQueue.invokeLater(new Runnable()
21.         {
22.             public void run()
23.             {
24.                 JFrame frame = new DOMTreeFrame();
25.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
26.                 frame.setVisible(true);
27.             }
28.         });
29.     }
30. }
```

```
27.         }
28.     });
29. }
30. }
31.
32. /**
33. * This frame contains a tree that displays the contents of an XML document.
34. */
35. class DOMTreeFrame extends JFrame
36. {
37.     public DOMTreeFrame()
38.     {
39.         setTitle("DOMTreeTest");
40.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
41.
42.         JMenu fileMenu = new JMenu("File");
43.         JMenuItem openItem = new JMenuItem("Open");
44.         openItem.addActionListener(new ActionListener()
45.         {
46.             public void actionPerformed(ActionEvent event)
47.             {
48.                 openFile();
49.             }
50.         });
51.         fileMenu.add(openItem);
52.
53.         JMenuItem exitItem = new JMenuItem("Exit");
54.         exitItem.addActionListener(new ActionListener()
55.         {
56.             public void actionPerformed(ActionEvent event)
57.             {
58.                 System.exit(0);
59.             }
60.         });
61.         fileMenu.add(exitItem);
62.
63.         JMenuBar menuBar = new JMenuBar();
64.         menuBar.add(fileMenu);
65.         setJMenuBar(menuBar);
66.     }
67.
68. /**
69. * Open a file and load the document.
70. */
71. public void openFile()
72. {
73.     JFileChooser chooser = new JFileChooser();
74.     chooser.setCurrentDirectory(new File("."));
75.
76.     chooser.setFileFilter(new javax.swing.filechooser.FileFilter()
77.     {
78.         public boolean accept(File f)
79.         {
80.             return f.isDirectory() || f.getName().toLowerCase().endsWith(".xml");
81.         }
82.     });
83. }
```

```
83.         public String getDescription()
84.         {
85.             return "XML files";
86.         }
87.     });
88.     int r = chooser.showOpenDialog(this);
89.     if (r != JFileChooser.APPROVE_OPTION) return;
90.     final File file = chooser.getSelectedFile();
91.
92.     new SwingWorker<Document, Void>()
93.     {
94.         protected Document doInBackground() throws Exception
95.         {
96.             if (builder == null)
97.             {
98.                 DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
99.                 builder = factory.newDocumentBuilder();
100.            }
101.            return builder.parse(file);
102.        }
103.
104.        protected void done()
105.        {
106.            try
107.            {
108.                Document doc = get();
109.                JTree tree = new JTree(new DOMTreeModel(doc));
110.                tree.setCellRenderer(new DOMTreeCellRenderer());
111.
112.                setContentPane(new JScrollPane(tree));
113.                validate();
114.            }
115.            catch (Exception e)
116.            {
117.                JOptionPane.showMessageDialog(DOMTreeFrame.this, e);
118.            }
119.        }
120.    }.execute();
121. }
122.
123. private DocumentBuilder builder;
124. private static final int DEFAULT_WIDTH = 400;
125. private static final int DEFAULT_HEIGHT = 400;
126. }
127.
128. /**
129. * This tree model describes the tree structure of an XML document.
130. */
131. class DOMTreeModel implements TreeModel
132. {
133.     /**
134.     * Constructs a document tree model.
135.     * @param doc the document
136.     */
137.     public DOMTreeModel(Document doc)
138.     {
```

```
139.     this.doc = doc;
140. }
141
142. public Object getRoot()
143{
144.     return doc.getDocumentElement();
145. }
146
147. public int getChildCount(Object parent)
148. {
149.     Node node = (Node) parent;
150.     NodeList list = node.getChildNodes();
151.     return list.getLength();
152. }
153
154. public Object getChild(Object parent, int index)
155. {
156.     Node node = (Node) parent;
157.     NodeList list = node.getChildNodes();
158.     return list.item(index);
159. }
160
161. public int getIndexOfChild(Object parent, Object child)
162. {
163.     Node node = (Node) parent;
164.     NodeList list = node.getChildNodes();
165.     for (int i = 0; i < list.getLength(); i++)
166.         if (getChild(node, i) == child) return i;
167.     return -1;
168. }
169
170. public boolean isLeaf(Object node)
171. {
172.     return getChildCount(node) == 0;
173. }
174
175. public void valueForPathChanged(TreePath path, Object newValue)
176. {
177. }
178
179. public void addTreeModelListener(TreeModelListener l)
180. {
181. }
182
183. public void removeTreeModelListener(TreeModelListener l)
184. {
185. }
186
187. private Document doc;
188. }
189
190. /**
191. * This class renders an XML node.
192. */
193. class DOMTreeCellRenderer extends DefaultTreeCellRenderer
194. {
```

```
195 public Component getTreeCellRendererComponent(JTree tree, Object value, boolean selected,
196     boolean expanded, boolean leaf, int row, boolean hasFocus)
197 {
198     Node node = (Node) value;
199     if (node instanceof Element) return elementPanel((Element) node);
200
201     super.getTreeCellRendererComponent(tree, value, selected, expanded, leaf, row, hasFocus);
202     if (node instanceof CharacterData) setText(characterString((CharacterData) node));
203     else setText(node.getClass() + ": " + node.toString());
204     return this;
205 }
206
207 public static JPanel elementPanel(Element e)
208 {
209     JPanel panel = new JPanel();
210     panel.add(new JLabel("Element: " + e.getTagName()));
211     final NamedNodeMap map = e.getAttributes();
212     panel.add(new JTable(new AbstractTableModel()
213     {
214         public int getRowCount()
215         {
216             return map.getLength();
217         }
218
219         public int getColumnCount()
220         {
221             return 2;
222         }
223
224         public Object getValueAt(int r, int c)
225         {
226             return c == 0 ? map.item(r).getNodeName() : map.item(r).getNodeValue();
227         }
228     }));
229     return panel;
230 }
231
232 public static String characterString(CharacterData node)
233 {
234     StringBuilder builder = new StringBuilder(node.getData());
235     for (int i = 0; i < builder.length(); i++)
236     {
237         if (builder.charAt(i) == '\r')
238         {
239             builder.replace(i, i + 1, "\\\r");
240             i++;
241         }
242         else if (builder.charAt(i) == '\n')
243         {
244             builder.replace(i, i + 1, "\\\n");
245             i++;
246         }
247         else if (builder.charAt(i) == '\t')
248         {
249             builder.replace(i, i + 1, "\\\t");
250             i++;
251         }
252     }
253     return builder.toString();
254 }
```

```
251     }
252     }
253     if (node instanceof CDATASection) builder.insert(0, "CDATASEction: ");
254     else if (node instanceof Text) builder.insert(0, "Text: ");
255     else if (node instanceof Comment) builder.insert(0, "Comment: ");
256
257     return builder.toString();
258   }
259 }
```

**API javax.xml.parsers.DocumentBuilderFactory 1.4**

- static DocumentBuilderFactory newInstance()  
返回DocumentBuilderFactory类的一个实例。
- DocumentBuilder newDocumentBuilder()  
返回DocumentBuilder类的一个实例。

**API javax.xml.parsers.DocumentBuilder 1.4**

- Document parse(File f)
- Document parse(String url)
- Document parse(InputStream in)  
解析来自给定文件、URL或输入流的XML文档，返回解析后的文档。

**API org.w3c.dom.Document 1.4**

- Element getDocumentElement()  
返回文档的根元素。

**API org.w3c.dom.Element 1.4**

- String getTagName()  
返回元素的名字。
- String getAttribute(String name)  
返回给定名字的属性值，没有该属性时返回空字符串。

**API org.w3c.dom.Node 1.4**

- NodeList getChildNodes()  
返回包含所有子元素节点的节点列表。
- Node getFirstChild()
- Node getLastChild()  
获取该节点的第一个或最后一个子节点，在该节点没有子节点时返回null。
- Node getNextSibling()
- Node getPreviousSibling()  
获取该节点的下一个或上一个兄弟节点，在该节点没有兄弟节点时返回null。

- `Node getParentNode()`

获取该节点的父节点，在该节点是文档节点时返回`null`。

- `NamedNodeMap getAttributes()`

返回含有描述该节点所有属性的Attr节点的映射表。

- `String getNodeName()`

返回该节点的名字。当该节点是Attr节点时，该名字就是属性名。

- `String getNodeValue()`

返回该节点的值。当该节点是Attr节点时，该值就是属性值。



#### `org.w3c.dom.CharacterData 1.4`

- `String getData()`

返回存储在节点中的文本。



#### `org.w3c.dom.NodeList 1.4`

- `int getLength()`

返回列表中的节点数。

- `Node item(int index)`

返回给定索引号的节点。索引号范围在0到`getLength()-1`之间。



#### `org.w3c.dom.NamedNodeMap 1.4`

- `int getLength()`

返回该节点映射表中的节点数。

- `Node item(int index)`

返回给定索引号的节点。索引号范围在0到`getLength()-1`之间。

## 2.3 验证XML文档

在前一节中，我们了解了如何遍历DOM文档的树形结构。然而，如果仅仅按照这种方法来操作，会发现需要大量冗长的编程和错误检查工作。你不但需要处理元素间的空白字符，还要检查该文档包含的节点是否和你期望的一样。例如，当你读入下面这个元素：

```
<font>
  <name>Helvetica</name>
  <size>36</size>
</font>
```

你将得到第一个子元素。这是一个含有空白字符“\n”的文本节点。你跳过文本节点找到第一个元素节点。然后，你要检查它的标签名是不是“name”。还要检查它是否有一个Text类型的子节点。接下去，转到下一个非空白字符的子元素，并进行同样的检查。那么，当文档作者改变了子元素的顺序或是加入另一个子元素时又会怎样呢？要是对所有的错误检查进行编码，显得太琐碎麻烦了，而跳过这些检查又是不慎重的。

幸好，XML解析器的一个很大的好处就是它能自动检验某个文档结构是否正确。这样，解

析就变得简单多了。例如，如果知道font片段已经通过了验证，那么你不用进一步检查就能得到其两个孙节点，并把它们转换成Text节点，得到它们的文本数据。

如果要规范文档结构，可以提供一个文档类型定义（DTD）或一个XML Schema定义。DTD或schema包含了用于解释文档是如何构成的规则。这些规则规范了每个元素的合法子元素和属性。例如，某个DTD可能含有一个规则：

```
<!ELEMENT font (name,size)>
```

这个规则表明，一个font元素总是有两个子元素，分别是name和size。XML Schema语言用于表示同样约束的形式如下：

```
<xsd:element name="font">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="size" type="xsd:int"/>
  </xsd:sequence>
</xsd:element>
```

与DTD相比，XML Schema可以表达更加复杂的验证条件（比如size元素必须包含一个整数）。与DTD语法不同，Schema使用XML，这将为你处理Schema文件带来方便。

XML Schema语言是设计用来替代DTD的。然而，到我们撰写本章时为止，DTD仍然具有旺盛的生命力。XML Schema很复杂，而且还没有得到普遍的支持。事实上，某些XML用户对XML Schema的复杂性感到很烦恼，以致他们采用替代的验证语言。最常用的一种就是Relax NG (<http://www.relaxng.org>)。

在以下的章节中，我们将详细讨论DTD。接着简要介绍XML Schema的一些基础知识。最后我们为你展示一个完整的应用程序来演示验证是如何简化XML编程的。

### 2.3.1 文档类型定义

提供DTD的方式有多种。可以像下面这样将其纳入一个XML文档中：

```
<?xml version="1.0"?>
<!DOCTYPE configuration [
  <!ELEMENT configuration . . .>
  more rules
  .
  .
]>
<configuration>
  .
  .
</configuration>
```

正如你看到的，这些规则被纳入到了DOCTYPE声明中，该代码块使用[...]来限定其界限。文档类型必须符合根元素的名字，比如我们例子中的configuration那样。

在XML文档内部提供DTD不是很普遍，因为DTD会变得很长。把DTD存储在外面可能更具意义。SYSTEM声明可以用来实现这个目标。你可以设定一个包含DTD的URL，例如：

```
<!DOCTYPE configuration SYSTEM "config.dtd">
```

或者

```
<!DOCTYPE configuration SYSTEM "http://myserver.com/config.dtd">
```

**X** 警告：如果你使用的是DTD的相对URL（比如“config.dtd”），那么要给解析器一个文件或URL对象，而不是InputStream。如果必须从一个输入流来解析，请提供一个实体渲染器，请看下面的说明。

最后，有一个来源于SGML的用于识别“众所周知的”DTD的机制，下面是一个例子：

```
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
```

如果XML处理器知道如何定位带有公共标识符的DTD，那么就不需要URL了。

**✓** 注意：如果你使用的是DOM解析器，并且想要支持公共标识符，请调用DocumentBuilder类的setEntityResolver方法来安装EntityResolver接口的某个实现类的一个对象。该接口只有一个方法：resolveEntity。下面是一个典型实现的代码框架：

```
class MyEntityResolver implements EntityResolver
{
    public InputSource resolveEntity(String publicID,
                                     String systemID)
    {
        if (publicID.equals(a known ID))
            return new InputSource(DTD data);
        else
            return null; // use default behavior
    }
}
```

你可以从InputStream、Reader或字符串构建输入源。

既然你已经知道解析器怎样定位DTD了，下面让我们来看看不同类型的规则。

ELEMENT规则用于规范某个元素可以拥有什么样的子元素。可以设定一个正则表达式，它由表2-1中所示组件构成。

表2-1 元素内容的规则

| 规 则  | 含 义   |
|--|---|
| E*   | 0或多个E   |
| E+   | 1或多个E   |
| E?   | 0或1个E   |
| E <sub>1</sub>  E <sub>2</sub>  ... E <sub>n</sub>                         | E <sub>1</sub> , E <sub>2</sub> , ..., E <sub>n</sub> 中的一个                |
| E <sub>1</sub> , E <sub>2</sub> , ..., E <sub>n</sub>                      | E <sub>1</sub> 随后是E <sub>2</sub> , ..., E <sub>n</sub>                    |
| #PCDATA  | 文本  |
| (#PCDATA E <sub>1</sub>  E <sub>2</sub>  ... E <sub>n</sub> ) <sup>*</sup> | 0或多个任意顺序的文本和E <sub>1</sub> , E <sub>2</sub> , ..., E <sub>n</sub> （混合式内容） |
| ANY  | 允许任意子元素   |
| EMPTY  | 不允许有子元素   |

下面是一些简单而典型的例子。下面的规则说明了menu元素包含0或多个item元素：

```
<!ELEMENT menu (item)*>
```

下面这组规则说明font是用一个name后跟一个size来描述的，它们都包含了文本：

```
<!ELEMENT font (name,size)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT size (#PCDATA)>
```

缩写PCDATA标识已解析的字符数据。这些数据被称为“已解析的”是因为解析器解释了该文本字符串，并正在寻找表示一个新标签起始的<字符或表示一个实体起始的&字符。

元素的规范可以包含嵌套的和复杂的正则表达式，例如，下面是一个描述了本书中一章的结构的规则：

```
<!ELEMENT chapter (intro,(heading,(para|image|table|note)+)+)
```

每章都以简介开头，其后是1或多个小节，每个小节由一个标题和1个或多个段落、图片、表格或说明。

然而，有一种常见的情况，不能把规则定义得像你希望的那样灵活。当一个元素可以包含文本时，那么就只有两种情况。要么元素只包含文本，比如：

```
<!ELEMENT name (#PCDATA)>
```

要么元素包含任意顺序的文本和标签，比如：

```
<!ELEMENT para (#PCDATA|em|strong|code)*>
```

设定其他包含#PCDATA规则的类型都是不合法的。例如，以下规则是非法的：

```
<!ELEMENT captionedImage (image,#PCDATA)>
```

你不得不重写一个规则，以引入另一个caption元素或者允许使用image标签和文本的组合。

这种限制简化了XML解析器在解析混合式内容（标签和文本的混合）时的工作。因为在允许混合式内容时会失去一些控制，所以最好在设计DTD时，让其中所有的元素要么包含其他元素，要么只有文本。



**注意：**实际上，可以在DTD中设定元素的任意正则表达式。XML解析器会拒绝某些导致非确定性的复杂规则。例如，正表达式 $((x,y)|(x,z))$ 是非确定性的。当解析器看到X时，它不知道在两个选择中取哪一个。这个表达式可以改写成确定性的形式，如 $(x,(y|z))$ 。然而，有一些表达式不能被改写，如 $((x,y)^*|x?)$ 。Sun解析器在遇到歧义的DTD时，不会给出警告。在解析时，它仅仅在两者中选取第一个匹配项，这将导致它拒绝一些正确的输入。当然，解析器有权这么做，因为XML标准允许解析器假设DTD都是非二义性的。在实际应用中，这不是一个会让你睡不着觉的问题，因为大多数DTD都非常简单，根本不会遇上二义性问题。

还要设定描述合法元素属性的规则。通常语法是：

```
<!ATTLIST element attribute type default>
```

表2-2显示了合法的属性类型，表2-3显示了属性默认值的语法。

以下是两个典型的属性规范：

```
<!ATTLIST font style (plain|bold|italic|bold-italic) "plain">
<!ATTLIST size unit CDATA #IMPLIED>
```

第一个规范描述font元素的style属性。它有4个合法的属性值，默认值是plain。第二个

规范表示size元素的unit属性可以包含任意字符数据序列。

表2-2 属性类型

| 类 型   | 含 义   |
|---|---|
| CDATA   | 任意字符串   |
| (A <sub>1</sub>  A <sub>2</sub>  ... A <sub>n</sub> ) | A <sub>1</sub> A <sub>2</sub> ...   A <sub>n</sub> 之一 |
| NMTOKEN, NMTOKENS                                     | 或多个名字标记   |
| ID  | 一个惟一的ID   |
| IDREF, IDREFS   | 或多个惟一ID的引用  |
| ENTITY, ENTITIES                                      | 或多个未经解析的实体  |

表2-3 属性的默认值

| 默 认 值     | 含 义                             |
|-----------|---------------------------------|
| #REQUIRED | 属性是必须的                          |
| #IMPLIED  | 属性是可选的                          |
| A         | 属性是可选的；若未指定，解析器报告的属性是A          |
| #FIXED A  | 属性必须是未设定的或者是A；两者情况下，解析器报告的属性都是A |

 **注意：**一般情况下，我们推荐用元素而非属性来描述数据。按照这个推荐，font style应该是一个独立的元素，例如<font><style>plain</style>...</font>。然而，对于枚举型的类型，属性有一个不可否认的优点，那就是解析器能够确认它的值是否合法。例如，如果font style是一个属性，解析器就会检查它是不是4个允许值之一，如果没有提供属性值，解析器会提供一个默认值。

C DATA属性值的处理与你前面看到的处理#PCDATA有着微妙的差别，并且与<! [CDATA[...]]>部分没有多大关系。属性值首先被范式化，也就是说，解析器要处理字符和实体的引用（比如&#233;或&lt;），并且要用空格来替换空白字符。

NMTOKEN（即名字标记）与C DATA相似，但是大多数非字母数字字符和内部空白字符是不允许使用的，而且解析器会删除起始和结尾的空白字符。NMTOKENS是一个空白字符分隔的名字标记列表。

ID结构是很有用的，ID是在文档中惟一的名字标记，解析器会检查其惟一性。在下一个示例程序中，你会看到它的应用。IDREF是对同一文档中存在的ID的引用。解析器也会对它进行检查。IDREFS是空白字符分隔的ID引用列表。

ENTITY属性值是指一个“未经解析的外部实体”。这是从SGML那里沿用下来的，在实际应用中很少见到。<http://www.xml.com/axml/axml.html>的带注释的XML规范有该属性的一个例子。

DTD也可以定义实体，或者定义解析过程中被替换的缩写。可以在Mozilla/Netscape 6浏览器的用户接口描述中找到一个使用实体的很好的例子。这些描述是用XML格式化的，它们包含了如下的实体定义：

```
<!ENTITY back.Label "Back">
```

在其他地方，文本可以包含实体引用，例如：

```
<menuitem label=&back.label;/>
```

解析器用替代字符串来替换实体引用。如果要对应用程序进行国际化，只需修改实体定义中的字符串。其他实体的使用更加复杂，不太常用。详细说明参见XML规范。

这样我们就结束了对DTD的介绍。现在你已经知道如何使用DTD了，你可以配置你的解析器来利用它们。首先，通知文档生成工厂打开验证特性。

```
factory.setValidating(true);
```

该工厂生成的所有文档生成器都将根据DTD来验证它们的输入。验证的最大好处是忽略元素内容中的空白字符。例如，请看以下XML代码片段：

```
<font>
  <name>Helvetica</name>
  <size>36</size>
</font>
```

一个非验证的解析器会报告font、name和size元素之间的空白字符，因为它无法知道font的子元素是否是：

```
(name,size)
(#PCDATA,name,size)*
```

还是：

ANY

一旦DTD设定了子元素是(name, size)，解析器就知道它们之间的空白字符不是文本。调用下面的代码：

```
factory.setIgnoringElementContentWhitespace(true);
```

这样，生成器将不会报告文本节点中的空白字符。这意味着，你可以依赖font节点只有2个子元素这一事实。你再也不用编写下面这样的单调冗长的循环代码：

```
for (int i = 0; i < children.getLength(); i++)
{
    Node child = children.item(i);
    if (child instanceof Element)
    {
        Element childElement = (Element) child;
        if (childElement.getTagName().equals("name")) . . .
        else if (childElement.getTagName().equals("size")) . . .
    }
}
```

而是仅仅通过如下代码访问第一个和第二个子元素：

```
Element nameElement = (Element) children.item(0);
Element sizeElement = (Element) children.item(1);
```

这就是DTD如此有用的原因。你不会为了检查规则而使程序负担过重。在得到文档之前，解析器已经做完了这些工作。

**提示：**许多刚刚开始使用XML的程序员都对验证不习惯，并且始终在系统运行过程中分析DOM树。如果说服你的同事使他们相信使用验证过的文档的好处，就给他们看上述两种不同的编码方式，这样才能使他们相信你。

当解析器报告错误时，应用程序希望对该错误执行某些操作。例如，记录到日志中，把它显示给用户，或是抛出一个异常以放弃解析。因此，在验证时，应该安装一个错误处理器，这需要应该提供一个实现了ErrorHandler接口的对象。这个接口有三个方法：

```
void warning(SAXParseException exception)  
void error(SAXParseException exception)  
void fatalError(SAXParseException exception)
```

可以通过DocumentBuilder类的setErrorHandler方法来安装错误处理器：

```
builder.setErrorHandler(handler);
```

#### **API** javax.xml.parsers.DocumentBuilder 1.4

- void setEntityResolver(EntityResolver resolver)

设置解析器，来定位要分析的XML文档中引用的实体。

- void setErrorHandler(ErrorHandler handler)

设置报告解析过程中出现的错误和警告的处理器。

#### **API** org.xml.sax.EntityResolver 1.4

- public InputSource resolveEntity(String publicID, String systemID)

返回包含被指定ID引用数据的一个输入源，或者，当解析器不知道如何解析某个特定名字时，返回null。如果没有提供公共ID，那么参数publicID可以为null。

#### **API** org.xml.sax.InputSource 1.4

- InputSource(InputStream in)
- InputSource(Reader in)
- InputSource(String systemID)

根据流、读入器、或系统ID（通常是相对或绝对URL）构建输入源。

#### **API** org.xml.sax.ErrorHandler 1.4

- void fatalError(SAXParseException exception)
- void error(SAXParseException exception)
- void warning(SAXParseException exception)

覆盖这些方法以提供处理器，对致命错误、非致命错误和警告进行处理。

#### **API** org.xml.sax.SAXParseException 1.4

- int getLineNumber()
- int getColumnNumber()

返回引起异常的已处理的输入信息末尾的行号和列号。

#### **API** javax.xml.parsers.DocumentBuilderFactory 1.4

- boolean isValidation()
- void setValidation(boolean value)

获取和设置工厂的 validating 属性。当它设为 true 时，工厂生成的解析器会验证它们的输入信息。

- boolean isIgnoringElementContentWhitespace()
- void setIgnoringElementContentWhitespace(boolean value)

获取和设置工厂的 ignoringElementContentWhitespace 属性。当它设为 true 时，工厂生成的解析器会忽略没有混合内容（即，元素与#PCDATA 混合）的元素节点之间的空白字符。

### 2.3.2 XML Schema

因为 XML Schema 比起 DTD 语法要复杂许多，所以我们只涉及其基本知识。更多信息请参考 <http://www.w3.org/TR/xmlschema-0> 上的指南。

如果要在文档中引用 Schema 文件，需要在根元素中加上属性，例如：

```
<?xml version="1.0"?>
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="config.xsd">
    ...
</configuration>
```

这个声明说明 Schema 文件 config.xsd 会被用来验证文档。如果你使用命名空间，语法就更加复杂了。详细请参见 XML Schema 指南（前缀 xsi 是一个命名空间别名（namespace alias），请查看第 2.5 节以了解的更多信息）。

Schema 定义了每个元素的类型。类型可以是简单类型、格式受限的字符串或复杂类型。一些简单类型已经被内建到了 XML Schema 内，包括：

```
xsd:string
xsd:int
xsd:boolean
```

 注意：我们用前缀 xsd: 来表示 XSL Schema 定义的命名空间。一些作者代之以 xs:。

可以自己定义简单类型。例如，下面是一个枚举类型：

```
<xsd:simpleType name="StyleType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="PLAIN" />
        <xsd:enumeration value="BOLD" />
        <xsd:enumeration value="ITALIC" />
        <xsd:enumeration value="BOLD_ITALIC" />
    </xsd:restriction>
</xsd:simpleType>
```

当定义元素时，要设定它的类型：

```
<xsd:element name="name" type="xsd:string"/>
<xsd:element name="size" type="xsd:int"/>
<xsd:element name="style" type="StyleType"/>
```

类型约束了元素的内容。例如，下面的元素将被正确验证：

```
<size>10</size>
<style>PLAIN</style>
```

但是，下面的元素会被解析器拒绝：

```
<size>default</size>
<style>SLANTED</style>
```

你可以把类型组合成复杂类型，例如：

```
<xsd:complexType name="FontType">
  <xsd:sequence>
    <xsd:element ref="name"/>
    <xsd:element ref="size"/>
    <xsd:element ref="style"/>
  </xsd:sequence>
</xsd:complexType>
```

FontType是name、size和style的序列。在这个类型定义中，我们用ref属性来引用在Schema中位于别处的定义。也可以嵌套定义，像这样：

```
<xsd:complexType name="FontType">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="size" type="xsd:int"/>
    <xsd:element name="style" type="StyleType">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="PLAIN" />
          <xsd:enumeration value="BOLD" />
          <xsd:enumeration value="ITALIC" />
          <xsd:enumeration value="BOLD_ITALIC" />
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```

请注意style元素的匿名定义。

xsd:sequence和DTD中的连接符号等价。xsd:choice和|操作符等价，例如：

```
<xsd:complexType name="contactinfo">
  <xsd:choice>
    <xsd:element ref="email"/>
    <xsd:element ref="phone"/>
  </xsd:choice>
</xsd:complexType>
```

这和DTD中的类型email|phone类型是等价的。

如果要允许重复元素，可以使用minoccurs和maxoccurs属性，例如，与DTD类型item\*的等价形式如下：

```
<xsd:element name="item" type="..." minoccurs="0" maxoccurs="unbounded">
```

如果要设定属性，可以把xsd:attribute元素加到complexType定义中去：

```
<xsd:element name="size">
  <xsd:complexType>
    ...
    <xsd:attribute name="unit" type="xsd:string" use="optional" default="cm"/>
  </xsd:complexType>
</xsd:element>
```

这是与DTD声明等价的形式：

```
<!ATTLIST size unit CDATA #IMPLIED "cm">
```

可以把Schema的元素和类型定义封装在xsd:schema元素中：

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
  . . .
</xsd:schema>
```

解析带有Schema的XML文件和解析带有DTD的文件相似，但有3点差别：

1. 必须打开对命名空间的支持，即使在XML文件里你不使用它。

```
factory.setNamespaceAware(true);
```

2. 必须通过如下“魔咒”来准备处理Schema的工厂。

```
final String JAXP_SCHEMA_LANGUAGE = "http://java.sun.com/xml/jaxp/properties/schemaLanguage";
final String W3C_XML_SCHEMA = "http://www.w3.org/2001/XMLSchema";
factory.setAttribute(JAXP_SCHEMA_LANGUAGE, W3C_XML_SCHEMA);
```

3. 解析器不会丢弃元素中的空白字符，这确实很令人恼火，关于这是否是一个bug，人们看法不一。有一种变通方法，请参看程序清单2-4中的代码。

### 2.3.3 实用示例

在本节中，我们将要介绍一个实用的示例程序，来说明在实际环境中XML的用法。请回忆一下第I卷第9章，GridBagLayout是Swing构件中最有用的布局管理器。然而，人们都很畏惧它，这不仅是因为它的复杂，而且因为其编码冗长乏味。把布局指令放到一个文本文件中来替代大量重复代码将会带来很大便利。在本节中，你将看到怎样用XML来描述网格组（grid bag）布局和怎样解析布局文件。

网格组是由行和列构成的，它和HTML表格非常相似。与HTML表格相似的是，我们把它描述成一个行的序列，每个行都包含许多单元格：

```
<gridbag>
  <row>
    <cell>...</cell>
    <cell>...</cell>
    . . .
  </row>
  <row>
    <cell>...</cell>
    <cell>...</cell>
    . . .
  </row>
  . . .
</gridbag>
```

gridbag.dtd指定了以下规则：

```
<!ELEMENT gridbag (row)*>
<!ELEMENT row (cell)*>
```

一些单元格可以跨多行多列。在网格组布局中，这是通过将gridwidth和gridheight设置为大于1的值来实现的。这里使用相同的名字作为属性名：

```
<cell gridwidth="2" gridheight="2">
```

同样，我们将属性应用于网格组的其他约束：fill、anchor、gridx、gridy、weightx、weighty、ipadx和ipady。我们不处理 insets 约束，因为它的值不是简单类型，但是要支持它也是很简单的。例如：

```
<cell fill="HORIZONTAL" anchor="NORTH">
```

对大多数属性，我们都提供了与 GridBagConstraints 的构造器所提供的相同的默认值：

```
<!ATTLIST cell gridwidth CDATA "1">
<!ATTLIST cell gridheight CDATA "1">
<!ATTLIST cell fill (NONE|BOTH|HORIZONTAL|VERTICAL) "NONE">
<!ATTLIST cell anchor (CENTER|NORTH|NORTHEAST|EAST
    |SOUTHEAST|SOUTH|SOUTHWEST|WEST|NORTHWEST) "CENTER">
...

```

gridx 和 gridy 的值受到了特殊处理，因为如果手工设定会很冗长且易于出错。我们用以下的可选项来提供它们的值：

```
<!ATTLIST cell gridx CDATA #IMPLIED>
<!ATTLIST cell gridy CDATA #IMPLIED>
```

- 如果没有提供这些值，程序会通过如下的试探法来确定它们：在第0列，gridx 的默认值是0。否则，它是前面的gridx 加上前面的gridwidth。gridy 的默认值总是与行数相同。这样，大多数情况下你不必设置gridx 和 gridy 的值。但是，如果一个构件跨越多列，那么，每当要跳过这个构件时，就必须设置gridx。

 **注意：**网格组专家可能会奇怪，我们为什么不使用RELATIVE 和 REMAINDER 机制让网格组布局自动确定gridx 和 gridy 的位置呢？我们试过这种方法，但是怎么也不能产生图2-4中那个字体对话框示例的布局。阅读了GridBagLayout 的源代码后，我们发现，很明显，它的算法无法完成恢复绝对地址所必需的繁重任务。

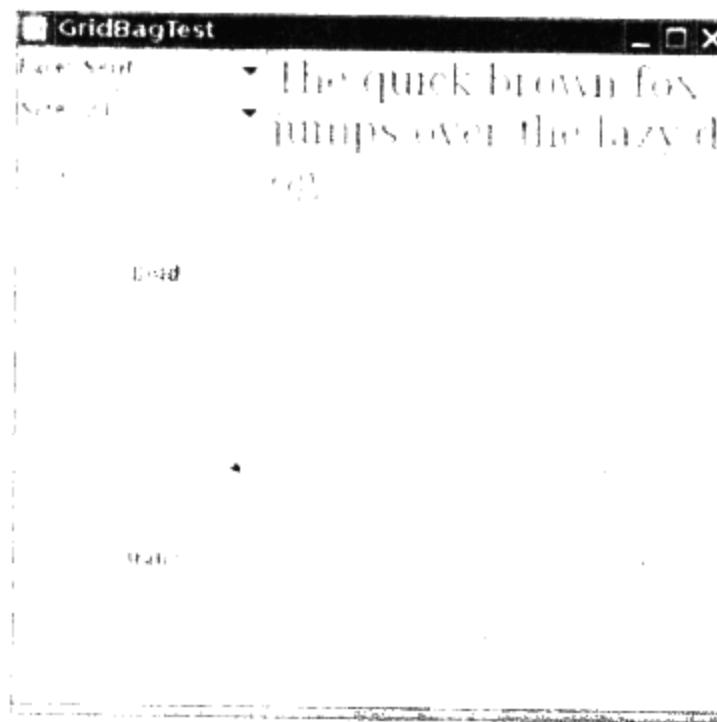


图2-4 由XML布局定义的字体对话框

这个程序对属性进行分析，并且设置了网格组的约束条件。例如，要读取网格宽度，程序

需要包含下面这行代码：

```
constraints.gridx = Integer.parseInt(e.getAttribute("gridx"));
```

程序不必担心属性的缺失，因为当文档中没有设定值时，解析器会自动提供默认值。

如果要测试gridx或gridy属性是否被设定了，我们可以调用getAttribute方法来检查它是否返回空串：

```
String value = e.getAttribute("gridy");
if (value.length() == 0) // use default
    constraints.gridy = r;
else
    constraints.gridx = Integer.parseInt(value);
```

我们发现允许单元格包含任意对象会显得很方便，这使我们能够设定如边界那样的非构件类型。这里只要求这些对象属于遵循JavaBean惯例的某个类，即具有一个默认构造器，而对每个属性都提供了相应的获取器（getter）/设置器（setter）对。（我们将在第8章详细讨论JavaBean。）

bean是由一个类名和0到多个属性定义的：

```
<!ELEMENT bean (class, property*)>
<!ELEMENT class (#PCDATA)>
```

属性包含一个名字和一个值。

```
<!ELEMENT property (name, value)>
<!ELEMENT name (#PCDATA)>
```

该值可以是整数、布尔值、字符串或者其他bean：

```
<!ELEMENT value (int|string|boolean|bean)>
<!ELEMENT int (#PCDATA)>
<!ELEMENT string (#PCDATA)>
<!ELEMENT boolean (#PCDATA)>
```

下面是一个典型示例，这是一个JLabel对象的实例，它的文本属性被设为"Face: "。

```
<bean>
    <class>javax.swing.JLabel</class>
    <property>
        <name>text</name>
        <value><string>Face:</string></value>
    </property>
</bean>
```

把字符串用<string>标签围起来似乎有点麻烦。为什么不只用#PCDATA表示字符串而把那个标签留给别的类型呢？因为那样我们必须使用混合式内容，这样会把value元素的规则弱化为：

```
<!ELEMENT value (#PCDATA|int|boolean|bean)*>
```

然而，这样的规则允许任意混合的文本和标签。

程序可以使用BeanInfo类来设置属性，而BeanInfo可以枚举bean的属性描述符。我们用匹配名字的方式来查找属性，然后调用它的setter方法来设置其值。

当我们的程序读入一个用户界面描述时，它有足够的信息来构建和安排用户界面构件。但是，当然，这个界面是死的，因为它没有事件监听器。如果要添加事件监听器，我们必须先定位构件。因为这个缘故，所以我们为每个bean提供了ID类型的可选属性：

```
<!ATTLIST bean id ID #IMPLIED>
```

例如，下面是一个带有ID的组合框：

```
<bean id="face">
  <class>javax.swing.JComboBox</class>
</bean>
```

请回想一下，我们说过解析器会检查ID是否惟一。

程序员可以用下面的方式来附加事件处理器：

```
gridbag = new GridBagPane("fontdialog.xml");
setContentPane(gridbag);
JComboBox face = (JComboBox) gridbag.get("face");
face.addListener(listener);
```

 注意：在这个示例中，我们只使用XML来描述构件布局，而把在Java代码中附加事件处理器的工作留给了程序员。你可以更进一步，将该代码添加到XML描述中去。最有前途的方式是将JavaScript这样的脚本语言运用到该代码中。如果你想添加这样的增强功能，请参考<http://www.mozilla.org/rhino>的Rhino解释器。

程序清单2-2的程序显示了如何使用GridBagPane类来完成设定网格组布局时所有的无聊工作，这个布局是在程序清单2-3中定义的。图2-4显示了运行结果。程序只初始化了组合框（这项工作对于GridBagPane支持的bean属性设定机制来说过于复杂，以致无法完成）和添加事件监听器；程序清单2-4中的GridBagPane类用于解析XML文件，构造构件并安置它们；程序清单2-5显示的是DTD文件。

除了DTD，如果像下面这样运行，程序也可以处理Schema：

```
java GridBagTest fontdialog-schema.xml
```

程序清单2-6就包含了一个Schema。

这个例子是XML的典型用法。XML格式十分健壮，足以表达复杂的关系。XML解析器是通过接管有效性检查和提供默认值等例行工作来添加值的。

### 程序清单2-2 GridBagTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.swing.*;
4.
5. /**
6.  * This program shows how to use an XML file to describe a gridbag layout
7.  * @version 1.01 2007-06-25
8.  * @author Cay Horstmann
9. */
10. public class GridBagTest
11. {
12.     public static void main(final String[] args)
13.     {
14.         EventQueue.invokeLater(new Runnable()
15.         {
16.             public void run()
17.             {
```

```
18         String filename = args.length == 0 ? "fontdialog.xml" : args[0];
19         JFrame frame = new FontFrame(filename);
20         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21         frame.setVisible(true);
22     }
23 }
24 }
25 }
26
27 /**
28 * This frame contains a font selection dialog that is described by an XML file.
29 * @param filename the file containing the user interface components for the dialog.
30 */
31 class FontFrame extends JFrame
32 {
33     public FontFrame(String filename)
34     {
35         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
36         setTitle("GridBagTest");
37
38         gridbag = new GridBagPane(filename);
39         add(gridbag);
40
41         face = (JComboBox) gridbag.get("face");
42         size = (JComboBox) gridbag.get("size");
43         bold = (JCheckBox) gridbag.get("bold");
44         italic = (JCheckBox) gridbag.get("italic");
45
46         face.setModel(new DefaultComboBoxModel(new Object[] { "Serif", "SansSerif",
47             "Monospaced", "Dialog", "DialogInput" }));
48
49         size.setModel(new DefaultComboBoxModel(new Object[] { "8", "10", "12", "15", "18", "24",
50             "36", "48" }));
51
52         ActionListener listener = new ActionListener()
53         {
54             public void actionPerformed(ActionEvent event)
55             {
56                 setSample();
57             }
58         };
59
60         face.addActionListener(listener);
61         size.addActionListener(listener);
62         bold.addActionListener(listener);
63         italic.addActionListener(listener);
64         setSample();
65     }
66
67 /**
68 * This method sets the text sample to the selected font.
69 */
70     public void setSample()
71     {
72         String fontFace = (String) face.getSelectedItem();
73         int fontSize = Integer.parseInt((String) size.getSelectedItem());
```

```
74.     JTextArea sample = (JTextArea) gridbag.get("sample");
75.     int fontStyle = (bold.isSelected() ? Font.BOLD : 0)
76.         + (italic.isSelected() ? Font.ITALIC : 0);
77.
78.     sample.setFont(new Font(fontFace, fontStyle, fontSize));
79.     sample.repaint();
80. }
81.
82. private GridBagPane gridbag;
83. private JComboBox face;
84. private JComboBox size;
85. private JCheckBox bold;
86. private JCheckBox italic;
87. private static final int DEFAULT_WIDTH = 400;
88. private static final int DEFAULT_HEIGHT = 400;
89. }
```

### 程序清单2-3 fontdialog.xml

```
1. <?xml version="1.0"?>
2. <!DOCTYPE gridbag SYSTEM "gridbag.dtd">
3. <gridbag>
4.   <row>
5.     <cell anchor="EAST">
6.       <bean>
7.         <class>javax.swing.JLabel</class>
8.         <property>
9.           <name>text</name>
10.          <value><string>Face: </string></value>
11.        </property>
12.      </bean>
13.    </cell>
14.    <cell fill="HORIZONTAL" weightx="100">
15.      <bean id="face">
16.        <class>javax.swing.JComboBox</class>
17.      </bean>
18.    </cell>
19.    <cell gridheight="4" fill="BOTH" weightx="100" weighty="100">
20.      <bean id="sample">
21.        <class>javax.swing.JTextArea</class>
22.        <property>
23.          <name>text</name>
24.          <value><string>The quick brown fox jumps over the lazy dog</string></value>
25.        </property>
26.        <property>
27.          <name>editable</name>
28.          <value><boolean>false</boolean></value>
29.        </property>
30.        <property>
31.          <name>lineWrap</name>
32.          <value><boolean>true</boolean></value>
33.        </property>
34.        <property>
35.          <name>border</name>
36.          <value>
37.            <bean>
```

```

38.          <class>javax.swing.border.EtchedBorder</class>
39.          </bean>
40.        </value>
41.      </property>
42.    </bean>
43.  </cell>
44. </row>
45. <row>
46.   <cell anchor="EAST">
47.     <bean>
48.       <class>javax.swing.JLabel</class>
49.       <property>
50.         <name>text</name>
51.         <value><string>Size: </string></value>
52.       </property>
53.     </bean>
54.   </cell>
55.   <cell fill="HORIZONTAL" weightx="100">
56.     <bean id="size">
57.       <class>javax.swing.JComboBox</class>
58.     </bean>
59.   </cell>
60. </row>
61. <row>
62.   <cell gridwidth="2" weighty="100">
63.     <bean id="bold">
64.       <class>javax.swing.JCheckBox</class>
65.       <property>
66.         <name>text</name>
67.         <value><string>Bold</string></value>
68.       </property>
69.     </bean>
70.   </cell>
71. </row>
72. <row>
73.   <cell gridwidth="2" weighty="100">
74.     <bean id="italic">
75.       <class>javax.swing.JCheckBox</class>
76.       <property>
77.         <name>text</name>
78.         <value><string>Italic</string></value>
79.       </property>
80.     </bean>
81.   </cell>
82. </row>
83. </gridbag>

```

#### 程序清单2-4 GridBagPane.java

```

1. import java.awt.*;
2. import java.beans.*;
3. import java.io.*;
4. import java.lang.reflect.*;
5. import javax.swing.*;
6. import javax.xml.parsers.*;
7. import org.w3c.dom.*;

```

```
8.  
9. /**  
10. * This panel uses an XML file to describe its components and their grid bag layout positions.  
11. * @version 1.10 2004-09-04  
12. * @author Cay Horstmann  
13. */  
14. public class GridBagPane extends JPanel  
15. {  
16.     /**  
17.      * Constructs a grid bag pane.  
18.      * @param filename the name of the XML file that describes the pane's components and their  
19.      * positions  
20.      */  
21.     public GridBagPane(String filename)  
22.     {  
23.         setLayout(new GridBagLayout());  
24.         constraints = new GridBagConstraints();  
25.  
26.         try  
27.         {  
28.             DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();  
29.             factory.setValidating(true);  
30.  
31.             if (filename.contains("-schema"))  
32.             {  
33.                 factory.setNamespaceAware(true);  
34.                 final String JAXP_SCHEMA_LANGUAGE = "http://java.sun.com/xml/jaxp/properties/  
35.                                     schemaLanguage";  
36.                 final String W3C_XML_SCHEMA = "http://www.w3.org/2001/XMLSchema";  
37.                 factory.setAttribute(JAXP_SCHEMA_LANGUAGE, W3C_XML_SCHEMA);  
38.             }  
39.  
40.             factory.setIgnoringElementContentWhitespace(true);  
41.  
42.             DocumentBuilder builder = factory.newDocumentBuilder();  
43.             Document doc = builder.parse(new File(filename));  
44.  
45.             if (filename.contains("-schema"))  
46.             {  
47.                 int count = removeElementContentWhitespace(doc.getDocumentElement());  
48.                 System.out.println(count + " whitespace nodes removed.");  
49.             }  
50.  
51.             parseGridbag(doc.getDocumentElement());  
52.         }  
53.         catch (Exception e)  
54.         {  
55.             e.printStackTrace();  
56.         }  
57.     }  
58.  
59.     /**  
60.      * Removes all (heuristically determined) element content whitespace nodes  
61.      * @param e the root element  
62.      * @return the number of whitespace nodes that were removed.  
63.      */
```

```
64. private int removeElementContentWhitespace(Element e)
65. {
66.     NodeList children = e.getChildNodes();
67.     int count = 0;
68.     boolean allTextChildrenAreWhiteSpace = true;
69.     int elements = 0;
70.     for (int i = 0; i < children.getLength() && allTextChildrenAreWhiteSpace; i++)
71.     {
72.         Node child = children.item(i);
73.         if (child instanceof Text && ((Text) child).getData().trim().length() > 0)
74.             allTextChildrenAreWhiteSpace = false;
75.         else if (child instanceof Element)
76.         {
77.             elements++;
78.             count += removeElementContentWhitespace((Element) child);
79.         }
80.     }
81.     if (elements > 0 && allTextChildrenAreWhiteSpace) // heuristics for element content
82.     {
83.         for (int i = children.getLength() - 1; i >= 0; i--)
84.         {
85.             Node child = children.item(i);
86.             if (child instanceof Text)
87.             {
88.                 e.removeChild(child);
89.                 count++;
90.             }
91.         }
92.     }
93.     return count;
94. }
95.
96. /**
97. * Gets a component with a given name
98. * @param name a component name
99. * @return the component with the given name, or null if no component in this grid bag
100. * pane has the given name
101. */
102. public Component get(String name)
103. {
104.     Component[] components = getComponents();
105.     for (int i = 0; i < components.length; i++)
106.     {
107.         if (components[i].getName().equals(name)) return components[i];
108.     }
109.     return null;
110. }
111.
112. /**
113. * Parses a gridbag element.
114. * @param e a gridbag element
115. */
116. private void parseGridbag(Element e)
117. {
118.     NodeList rows = e.getChildNodes();
119.     for (int i = 0; i < rows.getLength(); i++)
```

```
120.    {
121.        Element row = (Element) rows.item(i);
122.        NodeList cells = row.getChildNodes();
123.        for (int j = 0; j < cells.getLength(); j++)
124.        {
125.            Element cell = (Element) cells.item(j);
126.            parseCell(cell, i, j);
127.        }
128.    }
129. }
130.
131. /**
132. * Parses a cell element.
133. * @param e a cell element
134. * @param r the row of the cell
135. * @param c the column of the cell
136. */
137. private void parseCell(Element e, int r, int c)
138. {
139.     // get attributes
140.
141.     String value = e.getAttribute("gridx");
142.     if (value.length() == 0) // use default
143.     {
144.         if (c == 0) constraints.gridx = 0;
145.         else constraints.gridx += constraints.gridwidth;
146.     }
147.     else constraints.gridx = Integer.parseInt(value);
148.
149.     value = e.getAttribute("gridy");
150.     if (value.length() == 0) // use default
151.         constraints.gridy = r;
152.     else constraints.gridy = Integer.parseInt(value);
153.
154.     constraints.gridwidth = Integer.parseInt(e.getAttribute("gridwidth"));
155.     constraints.gridheight = Integer.parseInt(e.getAttribute("gridheight"));
156.     constraints.weightx = Integer.parseInt(e.getAttribute("weightx"));
157.     constraints.weighty = Integer.parseInt(e.getAttribute("weighty"));
158.     constraints.ipadx = Integer.parseInt(e.getAttribute("ipadx"));
159.     constraints.ipady = Integer.parseInt(e.getAttribute("ipady"));
160.
161.     // use reflection to get integer values of static fields
162.     Class<GridBagConstraints> cl = GridBagConstraints.class;
163.
164.     try
165.     {
166.         String name = e.getAttribute("fill");
167.         Field f = cl.getField(name);
168.         constraints.fill = f.getInt(cl);
169.
170.         name = e.getAttribute("anchor");
171.         f = cl.getField(name);
172.         constraints.anchor = f.getInt(cl);
173.     }
174.     catch (Exception ex) // the reflection methods can throw various exceptions
175.     {
176.         ex.printStackTrace();
```

```
177.     }
178.
179.     Component comp = (Component) parseBean((Element) e.getFirstChild());
180.     add(comp, constraints);
181. }
182.
183. /**
184. * Parses a bean element.
185. * @param e a bean element
186. */
187. private Object parseBean(Element e)
188. {
189.     try
190.     {
191.         NodeList children = e.getChildNodes();
192.         Element classElement = (Element) children.item(0);
193.         String className = ((Text) classElement.getFirstChild()).getData();
194.
195.         Class<?> cl = Class.forName(className);
196.
197.         Object obj = cl.newInstance();
198.
199.         if (obj instanceof Component) ((Component) obj).setName(e.getAttribute("id"));
200.
201.         for (int i = 1; i < children.getLength(); i++)
202.         {
203.             Node propertyElement = children.item(i);
204.             Element nameElement = (Element) propertyElement.getFirstChild();
205.             String propertyName = ((Text) nameElement.getFirstChild()).getData();
206.
207.             Element valueElement = (Element) propertyElement.getLastChild(),
208.             Object value = parseValue(valueElement);
209.             BeanInfo beanInfo = Introspector.getBeanInfo(cl);
210.             PropertyDescriptor[] descriptors = beanInfo.getPropertyDescriptors();
211.             boolean done = false;
212.             for (int j = 0; !done && j < descriptors.length; j++)
213.             {
214.                 if (descriptors[j].getName().equals(propertyName))
215.                 {
216.                     descriptors[j].getWriteMethod().invoke(obj, value);
217.                     done = true;
218.                 }
219.             }
220.         }
221.     }
222.     return obj;
223. }
224. catch (Exception ex) // the reflection methods can throw various exceptions
225. {
226.     ex.printStackTrace();
227.     return null;
228. }
229. }
230.
231. /**
232. * Parses a value element.
```

```

233.     * @param e a value element
234.     */
235.     private Object parseValue(Element e)
236.     {
237.         Element child = (Element) e.getFirstChild();
238.         if (child.getTagName().equals("bean")) return parseBean(child);
239.         String text = ((Text) child.getFirstChild()).getData();
240.         if (child.getTagName().equals("int")) return new Integer(text);
241.         else if (child.getTagName().equals("boolean")) return new Boolean(text);
242.         else if (child.getTagName().equals("string")) return text;
243.         else return null;
244.     }
245.
246.     private GridBagConstraints constraints;
247. }
```

**程序清单2-5 gridbag.dtd**

```

1. <!ELEMENT gridbag (row)*>
2. <!ELEMENT row (cell)*>
3. <!ELEMENT cell (bean)>
4. <!ATTLIST cell gridx CDATA #IMPLIED>
5. <!ATTLIST cell gridy CDATA #IMPLIED>
6. <!ATTLIST cell gridwidth CDATA "1">
7. <!ATTLIST cell gridheight CDATA "1">
8. <!ATTLIST cell weightx CDATA "0">
9. <!ATTLIST cell weighty CDATA "0">
10. <!ATTLIST cell fill (NONE|BOTH|HORIZONTAL|VERTICAL) "NONE">
11. <!ATTLIST cell anchor
12.     (CENTER|NORTH|NORTHEAST|EAST|SOUTHEAST|SOUTH|SOUTHWEST|WEST|NORTHWEST) "CENTER">
13. <!ATTLIST cell ipadx CDATA "0">
14. <!ATTLIST cell ipady CDATA "0">
15.
16. <!ELEMENT bean (class, property*)>
17. <!ATTLIST bean id ID #IMPLIED>
18.
19. <!ELEMENT class (#PCDATA)>
20. <!ELEMENT property (name, value)>
21. <!ELEMENT name (#PCDATA)>
22. <!ELEMENT value (int|string|boolean|bean)>
23. <!ELEMENT int (#PCDATA)>
24. <!ELEMENT string (#PCDATA)>
25. <!ELEMENT boolean (#PCDATA)>
```

**程序清单2-6 gridbag.xsd**

```

1. <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
2.
3.     <xsd:element name="gridbag" type="GridBagType"/>
4.
5.     <xsd:element name="bean" type="BeanType"/>
6.
7.     <xsd:complexType name="GridBagType">
8.         <xsd:sequence>
9.             <xsd:element name="row" type="RowType" minOccurs="0" maxOccurs="unbounded"/>
10.            </xsd:sequence>
```

```
11. </xsd:complexType>
12.
13. <xsd:complexType name="RowType">
14.   <xsd:sequence>
15.     <xsd:element name="cell" type="CellType" minOccurs="0" maxOccurs="unbounded"/>
16.   </xsd:sequence>
17. </xsd:complexType>
18.
19. <xsd:complexType name="CellType">
20.   <xsd:sequence>
21.     <xsd:element ref="bean"/>
22.   </xsd:sequence>
23.     <xsd:attribute name="gridx" type="xsd:int" use="optional"/>
24.     <xsd:attribute name="gridy" type="xsd:int" use="optional"/>
25.     <xsd:attribute name="gridwidth" type="xsd:int" use="optional" default="1" />
26.     <xsd:attribute name="gridheight" type="xsd:int" use="optional" default="1" />
27.     <xsd:attribute name="weightx" type="xsd:int" use="optional" default="0" />
28.     <xsd:attribute name="weighty" type="xsd:int" use="optional" default="0" />
29.     <xsd:attribute name="fill" use="optional" default="NONE">
30.       <xsd:simpleType>
31.         <xsd:restriction base="xsd:string">
32.           <xsd:enumeration value="NONE" />
33.           <xsd:enumeration value="BOTH" />
34.           <xsd:enumeration value="HORIZONTAL" />
35.           <xsd:enumeration value="VERTICAL" />
36.         </xsd:restriction>
37.       </xsd:simpleType>
38.     </xsd:attribute>
39.     <xsd:attribute name="anchor" use="optional" default="CENTER">
40.       <xsd:simpleType>
41.         <xsd:restriction base="xsd:string">
42.           <xsd:enumeration value="CENTER" />
43.           <xsd:enumeration value="NORTH" />
44.           <xsd:enumeration value="NORTHEAST" />
45.           <xsd:enumeration value="EAST" />
46.           <xsd:enumeration value="SOUTHEAST" />
47.           <xsd:enumeration value="SOUTH" />
48.           <xsd:enumeration value="SOUTHWEST" />
49.           <xsd:enumeration value="WEST" />
50.           <xsd:enumeration value="NORTHWEST" />
51.         </xsd:restriction>
52.       </xsd:simpleType>
53.     </xsd:attribute>
54.     <xsd:attribute name="ipady" type="xsd:int" use="optional" default="0" />
55.     <xsd:attribute name="ipadx" type="xsd:int" use="optional" default="0" />
56.   </xsd:complexType>
57.
58. <xsd:complexType name="BeanType">
59.   <xsd:sequence>
60.     <xsd:element name="class" type="xsd:string"/>
61.     <xsd:element name="property" type=".PropertyType" minOccurs="0" maxOccurs="unbounded"/>
62.   </xsd:sequence>
63.   <xsd:attribute name="id" type="xsd:ID" use="optional" />
64. </xsd:complexType>
65.
66. <xsd:complexType name=".PropertyType">
```

```
67      <xsd:sequence>
68          <xsd:element name="name" type="xsd:string"/>
69          <xsd:element name="value" type="ValueType"/>
70      </xsd:sequence>
71  </xsd:complexType>
72
73  <xsd:complexType name="ValueType">
74      <xsd:choice>
75          <xsd:element ref="bean"/>
76          <xsd:element name="int" type="xsd:int"/>
77          <xsd:element name="string" type="xsd:string"/>
78          <xsd:element name="boolean" type="xsd:boolean"/>
79      </xsd:choice>
80  </xsd:complexType>
81 </xsd:schema>
```

## 2.4 使用XPath来定位信息

如果要定位某个XML文档中的一段特定信息，那么，通过遍历DOM树的众多节点来进行查找显得有些麻烦。XPath语言使得访问树节点变得很容易。例如，假设有如下XML文档：

```
<configuration>
    ...
    <database>
        <username>dbuser</username>
        <password>secret</password>
    ...
</database>
</configuration>
```

通过对XPath表达式`/configuration/database/username`求值来得到`database`中的`username`的值。

下列操作比普通的DOM方式要简单得多：

1. 得到文档节点。
2. 枚举它的子元素。
3. 定位`database`元素。
4. 获取`database`元素的第一个子元素，即`username`元素。
5. 获取`username`元素的第一个子元素，即Text节点。
6. 获取Text节点中的数据。

XPath可以描述XML文档中的一组节点，例如，下面的XPath：

`/gridbag/row`

描述了根元素`gridbag`的子元素中所有的`row`元素。可以用`[]`操作符来选择特定元素：

`/gridbag/row[1]`

这表示的是第一行（索引号从1开始）。

使用`@`操作符可以得到属性值。XPath表达式

`/gridbag/row[1]/cell[1]/@anchor`

描述了第一行第一个单元格的`anchor`属性。XPath表达式

/gridbag/row/cell/@anchor

描述了作为根元素gridbag的子元素的那些行元素中的所有单元格的anchor属性节点。

XPath有很多有用的函数，例如：

`count(/gridbag/row)`

返回gridbag根元素的row元素的数量。精心设计的XPath表达式还有很多，请参见<http://www.w3c.org/TR/xpath>的规范，或者在<http://www.zvon.org/xxl/XPathTutorial/General/examples.html>上的一个非常好的在线指南。

Java SE 5.0增加了一个API来计算XPath表达式，需要先从XPathFactory创建一个XPath对象：

```
XPathFactory xpfactory = XPathFactory.newInstance();
path = xpfactory.newXPath();
```

然后，调用evaluate方法来计算XPath表达式：

```
String username = path.evaluate("/configuration/database/username", doc);
```

你可以用同样的XPath对象来计算多个表达式。

这种形式的evaluate方法将返回一个字符串。这很适合用来检索文本，比如前面的例子中的username节点中的文本。如果XPath表达式产生了一组节点，请做如下调用：

```
NodeList nodes = (NodeList) path.evaluate("/gridbag/row", doc, XPathConstants.NODESET);
```

如果结果只有一个节点，则以XPathConstants.NODE代替：

```
Node node = (Node) path.evaluate("/gridbag/row[1]", doc, XPathConstants.NODE);
```

如果结果是一个数字，则使用XPathConstants.NUMBER：

```
int count = ((Number) path.evaluate("count(/gridbag/row)", doc, XPathConstants.NUMBER)).intValue();
```

不必从文档的根节点开始搜索，可以从任意一个节点或节点列表开始。例如，如果你有前一次计算得到的一个节点，那么就可以调用：

```
result = path.evaluate(expression, node);
```

程序清单2-7中的程序演示了XPath表达式求值。只要载入一个XML文件，键入一个表达式，选择表达式的类型，点击计算按钮，表达式的结果就会在框架底部显示出来了（见图2-5）。

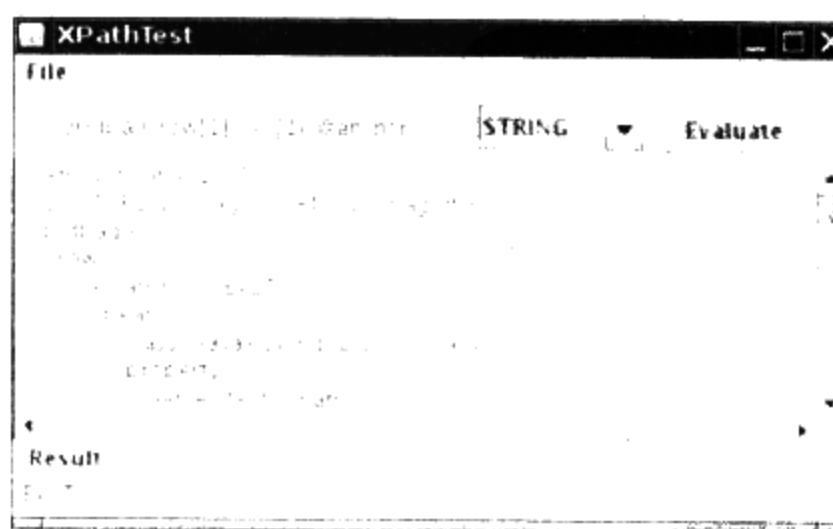


图2-5 计算XPath表达式

**程序清单2-7 XPathTest.java**

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.io.*;
4. import javax.swing.*;
5. import javax.swing.border.*;
6. import javax.xml.namespace.*;
7. import javax.xml.parsers.*;
8. import javax.xml.xpath.*;
9. import org.w3c.dom.*;
10. import org.xml.sax.*;

11.
12. /**
13. * This program evaluates XPath expressions
14. * @version 1.01 2007-06-25
15. * @author Cay Horstmann
16. */
17. public class XPathTest
18. {
19.     public static void main(String[] args)
20.     {
21.         EventQueue.invokeLater(new Runnable()
22.         {
23.             public void run()
24.             {
25.                 JFrame frame = new XPathFrame();
26.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27.                 frame.setVisible(true);
28.             }
29.         });
30.     }
31. }
32.
33. /**
34. * This frame shows an XML document, a panel to type an XPath expression, and a text field
35. * to display the result.
36. */
37. class XPathFrame extends JFrame
38. {
39.     public XPathFrame()
40.     {
41.         setTitle("XPathTest");
42.
43.         JMenu fileMenu = new JMenu("File");
44.         JMenuItem openItem = new JMenuItem("Open");
45.         openItem.addActionListener(new ActionListener()
46.         {
47.             public void actionPerformed(ActionEvent event)
48.             {
49.                 openFile();
50.             }
51.         });
52.         fileMenu.add(openItem);
53.
54.         JMenuItem exitItem = new JMenuItem("Exit");
55.         exitItem.addActionListener(new ActionListener()
```

```
56.     {
57.         public void actionPerformed(ActionEvent event)
58.         {
59.             System.exit(0);
60.         }
61.     });
62.     fileMenu.add(exitItem);
63.
64.     JMenuBar menuBar = new JMenuBar();
65.     menuBar.add(fileMenu);
66.     setJMenuBar(menuBar);
67.
68.     ActionListener listener = new ActionListener()
69.     {
70.         public void actionPerformed(ActionEvent event)
71.         {
72.             evaluate();
73.         }
74.     };
75.     expression = new JTextField(20);
76.     expression.addActionListener(listener);
77.     JButton evaluateButton = new JButton("Evaluate");
78.     evaluateButton.addActionListener(listener);
79.
80.     typeCombo = new JComboBox(new Object[] { "STRING", "NODE", "NODESET", "NUMBER",
81.                                     "BOOLEAN" });
82.     typeCombo.setSelectedItem("STRING");
83.
84.     JPanel panel = new JPanel();
85.     panel.add(expression);
86.     panel.add(typeCombo);
87.     panel.add(evaluateButton);
88.     docText = new JTextArea(10, 40);
89.     result = new JTextField();
90.     result.setBorder(new TitledBorder("Result"));
91.
92.     add(panel, BorderLayout.NORTH);
93.     add(new JScrollPane(docText), BorderLayout.CENTER);
94.     add(result, BorderLayout.SOUTH);
95.
96.     try
97.     {
98.         DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
99.         builder = factory.newDocumentBuilder();
100.    }
101.   catch (ParserConfigurationException e)
102.   {
103.     JOptionPane.showMessageDialog(this, e);
104.   }
105.
106.   XPathFactory xpfactory = XPathFactory.newInstance();
107.   path = xpfactory.newXPath();
108.   pack();
109. }
110.
111. /**
112. * Open a file and load the document.
```

```
113.  */
114. public void openFile()
115. {
116.     JFileChooser chooser = new JFileChooser();
117.     chooser.setCurrentDirectory(new File("."));
118.
119.     chooser.setFileFilter(new javax.swing.filechooser.FileFilter()
120.     {
121.         public boolean accept(File f)
122.         {
123.             return f.isDirectory() || f.getName().toLowerCase().endsWith(".xml");
124.         }
125.
126.         public String getDescription()
127.         {
128.             return "XML files";
129.         }
130.     });
131.     int r = chooser.showOpenDialog(this);
132.     if (r != JFileChooser.APPROVE_OPTION) return;
133.     File f = chooser.getSelectedFile();
134.     try
135.     {
136.         byte[] bytes = new byte[(int) f.length()];
137.         new FileInputStream(f).read(bytes);
138.         docText.setText(new String(bytes));
139.         doc = builder.parse(f);
140.     }
141.     catch (IOException e)
142.     {
143.         JOptionPane.showMessageDialog(this, e);
144.     }
145.     catch (SAXException e)
146.     {
147.         JOptionPane.showMessageDialog(this, e);
148.     }
149. }
150.
151. public void evaluate()
152. {
153.     try
154.     {
155.         String typeName = (String) typeCombo.getSelectedItem();
156.         QName returnType = (QName) XPathConstants.class.getField(typeName).get(null);
157.         Object evalResult = path.evaluate(expression.getText(), doc, returnType);
158.         if (typeName.equals("NODESET"))
159.         {
160.             NodeList list = (NodeList) evalResult;
161.             StringBuilder builder = new StringBuilder();
162.             builder.append("{");
163.             for (int i = 0; i < list.getLength(); i++)
164.             {
165.                 if (i > 0) builder.append(", ");
166.                 builder.append("") + list.item(i));
167.             }
168.             builder.append("}");
169.     }
170. }
```

```

169.         result.setText("'" + builder);
170.     }
171.     else result.setText("'" + evalResult);
172.   }
173.   catch (XPathExpressionException e)
174.   {
175.     result.setText("'" + e);
176.   }
177.   catch (Exception e) // reflection exception
178.   {
179.     e.printStackTrace();
180.   }
181. }
182.
183. private DocumentBuilder builder;
184. private Document doc;
185. private XPath path;
186. private JTextField expression;
187. private JTextField result;
188. private JTextArea docText;
189. private JComboBox typeCombo;
190. }

```



### **javax.xml.xpath.XPathFactory 5.0**

- **static XPathFactory newInstance()**  
返回XPathFactory实例来创建XPath对象。
- **XPath newPath()**  
构建XPath对象来计算XPath表达式。



### **javax.xml.xpath.XPath 5.0**

- **String evaluate(String expression, Object startingPoint)**  
从给定的起点计算表达式。起点可以是一个节点或节点列表。如果结果是一个节点或节点集，则返回的字符串包含所有文本节点子元素的数据。
- **Object evaluate(String expression, Object startingPoint, QName resultType)**  
从给定的起点计算表达式。起点可以是一个节点或节点列表。**resultType**是**XPathConstants**类的常量**STRING**、**NODE**、**NODESET**、**NUMBER**或**BOOLEAN**之一。返回值是**String**、**Node**、**NodeList**、**Number**或**Boolean**。

## 2.5 使用命名空间

Java语言使用包来避免名字冲突。程序员可以在不同的类中使用相同的名字，只要它们不在同一个包中即可。XML也有类似的命名空间（namespace）机制，可以用于元素名和属性名。名字空间是由统一资源标识符（Uniform Resource Identifier, URI）来标识的，比如：

<http://www.w3.org/2001/XMLSchema>  
uuid:1c759aed-b748-475c-ab68-10679700c4f2  
urn:com:books-r-us

HTTP的URL格式是最常用的。注意，URL只用作标识符字符串，而不是一个文件的定位器。例如，名字空间标识符：

```
http://www.horstmann.com/corejava  
http://www.horstmann.com/corejava/index.html
```

表示了不同的命名空间，尽管Web服务器将为这两个URL提供相同的文档。

在命名空间的URL不需要有任何文档，XML解析器不会尝试去该处找到任何东西。然而，为了给遇到不熟悉的命名空间的程序员一些帮助，人们习惯于将解释该命名空间的文档放在URL位置上。例如，如果你把浏览器指向XML Schema的命名空间URL (<http://www.w3.org/2001/XMLSchema>)，就会发现一个描述XML Schema标准的文档。

为什么要用HTTP URL作为命名空间的标识符？这是因为这样容易确保它们是独一无二的。如果使用一个实际的URL，那么主机部分的惟一性就将由域名系统来保证。然后，你的组织可以安排URL余下部分的惟一性，这和Java包名中的反向域名是一个原理。

为了惟一性，你可能想要用长命名空间标识符，当然你肯定是不想处理长标识符的。在Java编程语言中，可以用import机制来设定很长的包名，然后只是用较短的类名。在XML中有一个类似的机制，比如：

```
<element xmlns="namespaceURI">  
    children  
</element>
```

现在，元素和它的子元素都是给定命名空间的一部分。

子元素可以提供自己的命名空间，例如：

```
<element xmlns="namespaceURI1">  
    <child xmlns="namespaceURI2">  
        grandchildren  
    </child>  
    more children  
</element>
```

这时，第一个子元素和孙元素都是第二个命名空间的一部分。

无论是只需要一个命名空间，还是命名空间本质上是嵌套的，这个简单机制都工作得很好。否则，就需要使用第二种机制，Java中没有类似的机制。你可以拥有命名空间的别名，即为特定文档选取的一个短的标识符。下面是一个典型的例子：

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
    <xsd:element name="gridbag" type="GridBagType"/>  
    . . .  
</xsd:schema>
```

下面的属性：

```
xmlns:alias="namespaceURI"
```

用于定义命名空间和别名。在我们的例子中，别名是xsd。这样，xsd:schema实际上指的是“命名空间<http://www.w3.org/2001/XMLSchema>中的schema”。

 注意：只有子元素继承了它们父元素的命名空间，而不带显式别名前缀的属性不是命名空间的一部分。请看下面这个构造的例子：

```

<configuration xmlns="http://www.horstmann.com/corejava"
  xmlns:si="http://www.bipm.fr/enus/3_SI/si.html">
  <size value="210" si:unit="mm"/>
  ...
</configuration>

```

在这个示例中，元素`configuration`和`size`是URI `http://www.horstmann.com/corejava`的命名空间的一部分。属性`si:unit`是URI `http://www.bipm.fr/enus/3_SI/si.html`命名空间的一部分。然而，属性值不是任何命名空间的一部分。

你可以控制解析器对命名空间的处理。默认地，Sun公司的DOM解析器是没有“命名空间意识的”。

要打开命名空间处理特性，请调用`DocumentBuilderFactory`类的`setNamespaceAware`方法：

```
factory.setNamespaceAware(true);
```

这样工厂生产的所有生成器便都支持命名空间了。每个节点有三个属性：

- 带有别名前缀的限定名（qualified），由`getnodeName`和`getTagName`等方法返回。
- 命名空间URI，由`getNamespaceURI`方法返回。
- 不带别名前缀和命名空间的本地名（local name），由`getLocalName`方法返回。

下面是一个例子。假设解析器看到了以下元素：

```
<xsd:schema xmlns:xsl="http://www.w3.org/2001/XMLSchema">
```

它会报告：

- 限定名 = `xsd:schema`
- 命名空间 URI = `http://www.w3.org/2001/XMLSchema`
- 本地名 = `schema`

**注意：**如果命名空间特性被关闭，`getLocalName`和`getNamespaceURI`方法将返回null。

#### **org.w3c.dom.Node 1.4**

- `String getLocalName()`  
返回本地名（不带别名前缀），或者在解析器不支持命名空间时，返回null。
- `String getNamespaceURI()`  
返回命名空间URI，或者在解析器不支持命名空间时，返回null。

#### **javax.xml.parsers.DocumentBuilderFactory 1.4**

- `boolean isNamespaceAware()`
- `void setNamespaceAware(boolean value)`  
获取或设置工厂的`namespaceAware`属性。当设为true时，工厂产生的解析器支持命名空间。

## 2.6 流机制解析器

DOM解析器读入的是一个完整的XML文档，然后将其转换成一个树形的数据结构。对于大多数应用，DOM都运行得很好。但是，当文档很大，并且处理算法非常简单，可以在运行

时解析节点，而不必看到所有的树形结构时，DOM可能就会显得效率低下了。在这种情况下，我们应该使用流机制解析器（streaming parser）。

在下面的小节中，我们将讨论Java类库提供的流机制解析器：老而弥坚的SAX解析器和添加到Java SE 6中的更现代化的StAX解析器。SAX解析器使用的是事件回调（event callback），而StAX解析器提供了解析事件的迭代器，后者用起来通常更方便一些。

### 2.6.1 使用SAX解析器

SAX解析器在解析XML输入的构件时就报告事件，但不会以任何方式存储文档。由事件处理器决定是否要建立数据结构。实际上，DOM解析器是在SAX解析器的基础上建立起来的，它在接收到解析器事件时建立DOM树。

在使用SAX解析器时，需要一个处理器来定义不同的解析器事件的事件动作。ContentHandler接口定义了若干个回调方法。下面是最重要的几个：

- startElement和endElement在每当遇到起始或终止标签时调用。
- characters每当遇到字符数据时调用。
- startDocument和endDocument分别在文档开始和结束时各调用一次。

例如，解析以下片断时：

```
<font>
  <name>Helvetica</name>
  <size units="pt">36</size>
</font>
```

解析器确保产生以下调用：

1. startElement，元素名：font
2. startElement，元素名：name
3. characters，内容：Helvetica
4. endElement，元素名：name
5. startElement，元素名：size，属性：units="pt"
6. characters，内容：36
7. endElement，元素名：size
8. endElement，元素名：font

处理器必须覆盖这些方法，让它们执行在解析文件时想要执行的动作。本节最后的程序会打印出一个HTML文件中的所有链接[。它只覆盖了处理器的startElement方法，用来检查名为a，属性为href的链接，其潜在用途是用于实现“网络爬虫”，即一个沿着链接到达越来越多网页的程序。](...)

 注意：遗憾的是，大多数HTML页面都与良构的XML差别很大，以至于示例程序无法解析它们。正如前面提过的，W3C推荐Web设计人员使用XHTML，它是一个既可以被当前的Web浏览器显示的HTML方言，又是良构的XML。因为W3C采用了自己提出的方案，而且他们的网页都是用XHTML写的，所以可以用这些页面来测试示例程序。例如，运行：

```
java SAXTest http://www.w3c.org/MarkUp
```

将看到那一面上所有链接的URL列表。

示例程序是使用SAX的很好的例子。我们根本不在乎a元素出现的上下文环境，而且不必存储树形结构。

下面是如何得到SAX解析器的代码：

```
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser parser = factory.newSAXParser();
```

现在可以处理文档了：

```
parser.parse(source, handler);
```

这里的source可以是一个文件、一个URL字符串或者是一个输入流。处理器属于DefaultHandler的一个子类，DefaultHandler类为以下四个接口定义了空的方法：

```
ContentHandler
DTDHandler
EntityResolver
ErrorHandler
```

示例程序定义了一个处理器，它覆盖了ContentHandler接口的startElement方法来观察带有href属性的a元素。

```
DefaultHandler handler = new
DefaultHandler()
{
    public void startElement(String namespaceURI, String lname, String qname, Attributes attrs)
    throws SAXException
    {
        if (lname.equalsIgnoreCase("a") && attrs != null)
        {
            for (int i = 0; i < attrs.getLength(); i++)
            {
                String aName = attrs.getLocalName(i);
                if (aName.equalsIgnoreCase("href"))
                    System.out.println(attrs.getValue(i));
            }
        }
    }
};
```

startElement方法有3个描述元素名的参数，其中qname参数以alias:localname的形式报告限定名。如果命名空间处理特性已经打开，那么namespaceURI和lname参数描述的就是命名空间和本地（非限定）名。

与DOM解析器一样，命名空间处理特性默认是关闭的，可以调用工厂类的setNamespaceAware方法来激活命名空间处理特性：

```
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setNamespaceAware(true);
SAXParser saxParser = factory.newSAXParser();
```

程序清单2-8包含了网络爬虫程序的代码。在本章的后续部分，将会看到SAX的另一个有趣用法，即将非XML数据源转换成XML的一个简单方法是报告XML解析器将要报告的SAX事件。详细请参见第2.8节。

**程序清单2-8 SAXTest.java**

```
1. import java.io.*;
2. import java.net.*;
3. import javax.xml.parsers.*;
4. import org.xml.sax.*;
5. import org.xml.sax.helpers.*;
6.
7. /**
8. * This program demonstrates how to use a SAX parser. The program prints all hyperlinks links
9. * of an XHTML web page. <br>
10. * Usage: java SAXTest url
11. * @version 1.00 2001-09-29
12. * @author Cay Horstmann
13. */
14. public class SAXTest
15. {
16.     public static void main(String[] args) throws Exception
17.     {
18.         String url;
19.         if (args.length == 0)
20.         {
21.             url = "http://www.w3c.org";
22.             System.out.println("Using " + url);
23.         }
24.         else url = args[0];
25.
26.         DefaultHandler handler = new DefaultHandler()
27.         {
28.             public void startElement(String namespaceURI, String lname, String qname,
29.             Attributes attrs)
30.             {
31.                 if (lname.equals("a") && attrs != null)
32.                 {
33.                     for (int i = 0; i < attrs.getLength(); i++)
34.                     {
35.                         String aname = attrs.getLocalName(i);
36.                         if (aname.equals("href")) System.out.println(attrs.getValue(i));
37.                     }
38.                 }
39.             }
40.         };
41.
42.         SAXParserFactory factory = SAXParserFactory.newInstance();
43.         factory.setNamespaceAware(true);
44.         SAXParser saxParser = factory.newSAXParser();
45.         InputStream in = new URL(url).openStream();
46.         saxParser.parse(in, handler);
47.     }
48. }
```

**Javax.xml.parsers.SAXParserFactory 1.4**

- static SAXParserFactory newInstance()

返回SAXParserFactory类的一个实例。

- `SAXParser newSAXParser()`

返回`SAXParser`类的一个实例。

- `boolean isNamespaceAware()`

- `void setNamespaceAware(boolean value)`

获取和设置工厂的`namespaceAware`属性。当设为`true`时，该工厂生成的解析器是支持命名空间特性的。

- `boolean isValidating()`

- `void setValidating(boolean value)`

获取和设置工厂的`validating`属性。当设为`true`时，该工厂生成的解析器将要验证其输入。

### **API** `javax.xml.parsers.SAXParser 1.4`

- `void parse(File f, DefaultHandler handler)`

- `void parse(String url, DefaultHandler handler)`

- `void parse(InputStream in, DefaultHandler handler)`

解析来自给定文件、URL或输入流的XML文档，并把解析事件报告给指定的处理器。

### **API** `org.xml.sax.ContentHandler 1.4`

- `void startDocument()`

- `void endDocument()`

在文档的起始或结束时被调用。

- `void startElement(String uri, String lname, String qname, Attributes attr)`

- `void endElement(String uri, String lname, String qname)`

在元素的起始或结束时被调用。

参数: `uri` 命名空间的URI (如果解析器支持命名空间特性)

`lname` 不带别名前缀的本地名 (如果解析器支持命名空间特性)

`qname` 元素名 (如果解析器不支持命名空间特性)，或者是带有别名前缀的限定名 (如果解析器除了报告本地名之外还报告限定名)

- `void characters(char[] data, int start, int length)`

解析器报告字符数据时被调用。

参数: `data` 字符数据数组

`start` 作为被报告的字符的一部分的字符数组，它的第一个字符的索引

`length` 被报告的字符串的长度

### **API** `org.xml.sax.Attributes 1.4`

- `int getLength()`

返回存储在该属性集合中属性数量。

- `String getLocalName(int index)`

返回给定索引的属性的本地名 (无别名前缀)，或当解析器不支持命名空间特性时返回空

字符串。

- `String getURI(int index)`

返回给定索引的属性的命名空间URI，或者，当该节点不是命名空间的一部分，或解析器不支持命名空间特性时返回空字符串。

- `String getQName(int index)`

返回给定索引的属性的限定名（带别名前缀），或当解析器不报告限定名时返回空字符串。

- `String getValue(int index)`

- `String getValue(String qname)`

- `String getValue(String uri, String lname)`

根据给定索引、限定名或命名空间URI+本地名，返回属性值；该值不存在时，返回null。

## 2.6.2 使用StAX解析器

StAX解析器是一种“拉解析器（pull parser）”，与安装事件处理器不同，你只需使用下面这样的基本循环来迭代所有的事件：

```
InputStream in = url.openStream();
XMLInputFactory factory = XMLInputFactory.newInstance();
XMLStreamReader parser = factory.createXMLStreamReader(in);
while (parser.hasNext())
{
    int event = parser.next();
    Call parser methods to obtain event details
}
```

例如，在解析下面的片断时

```
<font>
    <name>Helvetica</name>
    <size units="pt">36</size>
</font>
```

解析器将产生下面的事件：

1. START\_ELEMENT，元素名：font
2. CHARACTERS，内容：空白字符
3. START\_ELEMENT，元素名：name
4. CHARACTERS，内容：Helvetica
5. END\_ELEMENT，元素名：name
6. CHARACTERS，内容：空白字符
7. START\_ELEMENT，元素名：size
8. CHARACTERS，内容：36
9. END\_ELEMENT，元素名：size
10. CHARACTERS，内容：空白字符
11. END\_ELEMENT，元素名：font

要分析这些属性值，需要调用`XMLStreamReader`类的恰当方法，例如：

```
String units = parser.getAttributeValue(null, "units");
```

获取当前元素的units属性。

默认情况下，命名空间处理是使能的，你可以通过修改下面的工厂来使其无效：

```
XMLInputFactory factory = XMLInputFactory.newInstance();
factory.setProperty(XMLInputFactory.IS_NAMESPACE_AWARE, false);
```

程序清单2-9包含了用StAX解析器实现的网络爬虫程序。正如你所见，这段代码比等效的SAX代码要简短了许多，因为此时我们不必操心事件处理问题。

### 程序清单2-9 StAXTest.java

```
1. import java.io.*;
2. import java.net.*;
3. import javax.xml.stream.*;
4.
5. /**
6. * This program demonstrates how to use a StAX parser. The program prints all hyperlinks links
7. * of an XHTML web page. <br>
8. * Usage: java StAXTest url
9. * @author Cay Horstmann
10. * @version 1.0 2007-06-23
11. */
12. public class StAXTest
13 {
14     public static void main(String[] args) throws Exception
15     {
16         String urlString;
17         if (args.length == 0)
18         {
19             urlString = "http://www.w3c.org";
20             System.out.println("Using " + urlString);
21         }
22         else urlString = args[0];
23         URL url = new URL(urlString);
24         InputStream in = url.openStream();
25         XMLInputFactory factory = XMLInputFactory.newInstance();
26         XMLStreamReader parser = factory.createXMLStreamReader(in);
27         while (parser.hasNext())
28         {
29             int event = parser.next();
30             if (event == XMLStreamConstants.START_ELEMENT)
31             {
32                 if (parser.getLocalName().equals("a"))
33                 {
34                     String href = parser.getAttributeValue(null, "href");
35                     if (href != null)
36                         System.out.println(href);
37                 }
38             }
39         }
40     }
41 }
```



### javax.xml.stream.XMLInputFactory 6

- static XMLInputFactory newInstance()

返回XMLInputFactory类的一个实例。

- void setProperty(String name, Object value)

设置这个工厂的属性，或者在要设置的属性不支持设置成给定值时，抛出IllegalArgumentException。Java SE实现支持下列Boolean类型的属性：

"javax.xml.stream.

为false（默认值）时，这个文档不被验证  
(规范不需要)。

isValidating

为true（默认值）时，将处理命名空间  
(规范不需要)。

"javax.xml.stream.

为false（默认值）时，邻近的字符不进行  
连接。

isNamespaceAware"

为true（默认值）时，实体引用将作为字符  
数据被替换和报告。

"javax.xml.stream.

为false（默认值）时，邻近的字符不进行  
连接。

isCoalescing"

为true（默认值）时，实体引用将作为字符  
数据被替换和报告。

"javax.xml.stream.

为true（默认值）时，外部实体将被解析。  
规范对于这个属性没有给出默认值。

isReplacingEntityReferences"

为true（默认值）时，DTD将作为事件被  
报告。

"javax.xml.stream.

为true（默认值）时，DTD将作为事件被  
报告。

isSupportingExternalEntities"

"javax.xml.stream.  
supportDTD"

- XMLStreamReader createXMLStreamReader(InputStream in)

- XMLStreamReader createXMLStreamReader(InputStream in, String character  
Encoding)

- XMLStreamReader createXMLStreamReader(Reader in)

- XMLStreamReader createXMLStreamReader(Source in)

创建一个从给定的流、阅读器或JAXP源读入的解析器。



### javax.xml.stream.XMLStreamReader 6

- boolean hasNext()

如果有另一个解析事件，则返回true。

- int next()

将解析器的状态设置为下一个解析事件，并返回下列常量之一：START\_ELEMENT、  
END\_ELEMENT、CHARACTERS、START\_DOCUMENT、END\_DOCUMENT、CDATA、COMMENT、  
SPACE（可忽略的空白字符）、PROCESSING\_INSTRUCTION、ENTITY\_REFERENCE、DTD。

- boolean isStartElement()

- boolean isEndElement()

- boolean isCharacters()

- boolean isWhiteSpace()

如果当前事件是一个开始元素、结束元素、字符数据或空白字符，则返回true。

- `QName getName()`
- `String getLocalName()`  
获取在START\_ELEMENT或END\_ELEMENT事件中的元素的名字。
- `String getText()`  
返回一个CHARACTERS、COMMENT或CDATA事件，或一个ENTITY\_REFERENCE的替换值，或者一个DTD的内部子集所对应的字符。
- `int getAttributeCount()`
- `QName getAttributeName(int index)`
- `String getAttributeLocalName(int index)`
- `String getAttributeValue(int index)`  
只要当前事件是START\_ELEMENT，则获取属性数量和属性的名字与值。
- `String getAttributeValue(String namespaceURI, String name)`  
只要当前事件是START\_ELEMENT，则获取给定属性的值。如果namespaceURI为null，则不检查名字空间。

## 2.7 生成XML文档

现在你已经知道怎样编写读取XML的Java程序了。下面让我们开始介绍它的反向过程，即输出XML。当然，你可以直接通过一系列print调用，打印出各元素、属性和文本内容，来编写XML文件，但这并不是一个好主意。这样代码会非常冗长复杂，对属性值和文本内容中的那些特殊符号（如：“和<），一不注意就会出错。

一个比较好的方法是用文档的内容构建一棵DOM树，然后再写出该树的所有内容。要建立一棵DOM树，你可以从一个空的文档开始。通过调用DocumentBuilder类的newDocument方法可以得到一个空文档。

```
Document doc = builder.newDocument();
```

使用Document类的createElement方法可以构建文档里的元素：

```
Element rootElement = doc.createElement(rootName);
Element childElement = doc.createElement(childName);
```

使用createTextNode方法可以构建文本节点：

```
Text textNode = doc.createTextNode(textContents);
```

使用以下方法可以给文档加上根元素，给父结点加上子节点：

```
doc.appendChild(rootElement);
rootElement.appendChild(childElement);
childElement.appendChild(textNode);
```

在建立DOM树时，可能还需要设置元素属性，这只需调用Element类的setAttribute方法即可：

```
rootElement.setAttribute(name, value);
```

有些奇怪的是，DOM API目前还不支持把DOM树写到输出流。为了克服这个限制，我们

使用可扩展的格式页转换（Extensible Stylesheet Language Transformations, XSLT）API。关于XSLT的更多信息，请参见第2.8节。当下，我们将这段代码视为可以生成XML输出的“魔咒”。

我们把“不做任何操作”的转换应用于文档，并且捕获它的输出。为了将DOCTYPE节点纳入输出，你还需要将SYSTEM和PUBLIC标识符设置为输出属性。

```
// construct the "do nothing" transformation
Transformer t = TransformerFactory.newInstance().newTransformer();
// set output properties to get a DOCTYPE node
t.setOutputProperty(OutputKeys.DOCUMENT_TYPE_NAME, "xml");
t.setOutputProperty(OutputKeys.DOCUMENT_URI, "http://www.w3.org/TR/2000/CR-SVG-20000802/DTD/svg-20000802.dtd");
t.setOutputProperty(OutputKeys.INDENT, "yes");
t.setOutputProperty(OutputKeys.METHOD, "xml");
t.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "2");
// apply the "do nothing" transformation and send the output to a file
t.transform(new DOMSource(doc), new StreamResult(new FileOutputStream(file)));
```

 注意：产生的XML文件不包含空白字符（也就是说，没有换行或缩进）。如果你喜欢空白字符，请把OutputKeys.INDENT属性设为"yes"。

程序清单2-10是一个生成XML输出的典型程序。该程序绘制了一幅现代画，即一组随机的彩色矩形（参见图2-6）。我们使用可伸缩向量图形（Scalable Vector Graphics, SVG）来保存作品。SVG是XML格式的，它使用设备无关的方式描述复杂图形。你可以在<http://www.w3c.org/Graphics/SVG>找到更多关于SVG的信息。要查看SVG文件，请从<http://xml.apache.org/batik>下载Apache Batik查看器（图2-7）。

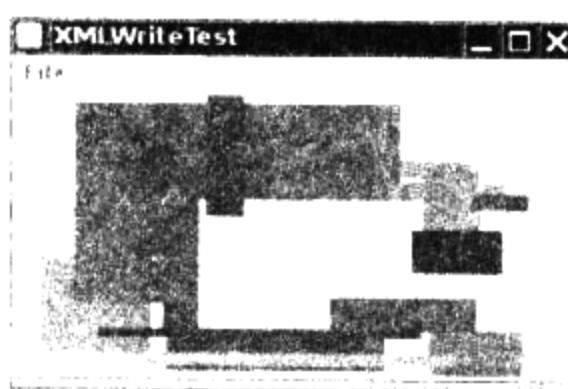


图2-6 生成现代艺术

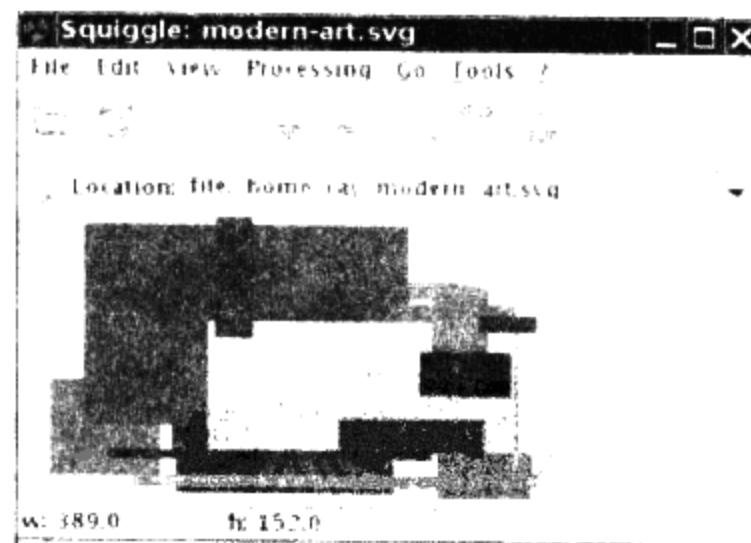


图2-7 Apache Batik的SVG查看器

我们没有涉及SVG的细节。如果你对SVG感兴趣，我们建议你从Adobe网站上的指南开始。就我们的目的而言，我们只需要知道怎样表达一组彩色的矩形。下面是一个例子：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20000802//EN"
  "http://www.w3.org/TR/2000/CR-SVG-20000802/DTD/svg-20000802.dtd">
<svg width="300" height="150">
<rect x="231" y="61" width="9" height="12" fill="#6e4a13"/>
<rect x="107" y="106" width="56" height="5" fill="#c406be"/>
...

```

```
</svg>
```

正如你看到的，每个矩形都描述为一个rect结点。它有位置、宽度、长度和填充色等属性，其中填充色以十六进制RGB值表示。

 **注意：**SVG大量使用了属性。实际上，某些属性相当复杂。例如，下面的path元素：

```
<path d="M 100 100 L 300 100 L 200 300 z">
```

M是指“moveto”命令、L是指“lineto”、z是指“closepath”(!)。显然，该数据格式的设计者不太信任XML的设计结构化数据能力。在你自己的XML格式中，应该使用元素来替代复杂的属性。

**API** `javax.xml.parsers.DocumentBuilder 1.4`

- `Document newDocument()`

返回一个空文档。

**API** `org.w3c.dom.Document 1.4`

- `Element createElement(String name)`

返回具有给定名字的元素。

- `Text createTextNode(String data)`

返回具有给定数据的文本节点。

**API** `org.w3c.dom.Node 1.4`

- `Node appendChild(Node child)`

将一个节点附加到该节点的子节点列表，返回该节点。

**API** `org.w3c.dom.Element 1.4`

- `void setAttribute(String name, String value)`

将有给定名字的属性设置为指定的值。

- `void setAttributeNS(String uri, String qname, String value)`

将带有给定命名空间URI和限定名的属性设置为指定的值。

参数：uri 名字空间的URI或null

qname 限定名。如果有别名前缀，uri不能为null

value 属性值

**API** `javax.xml.transform.TransformerFactory 1.4`

- `static TransformerFactory newInstance()`

返回TransformerFactory类的一个实例。

- `Transformer newTransformer()`

返回Transformer类的一个实例，它带有标识符或“无操作”的转换。

**API** **javax.xml.transform.Transformer 1.4**

- void setOutputProperty(String name, String value)

设置输出属性。标准输出属性参见<http://www.w3.org/TR/xslt#output>，其中最有用的几个如下所示：

参数：doctype-public DOCTYPE声明中使用的公共ID

doctype-system DOCTYPE声明中使用的系统ID

Indent “yes” 或者 “no”

method “xml”、“html”、“text” 或定制的字符串

- void transform(Source from, Result to)

转换一个XML文档。

**API** **javax.xml.transform.dom.DOMSource 1.4**

- DOMSource(Node n)

根据指定的节点构建一个源。通常，n是文档节点。

**API** **javax.xml.transform.stream.StreamResult 1.4**

- StreamResult(File f)
- StreamResult(OutputStream out)
- StreamResult(Writer out)
- StreamResult(String systemID)

根据文件、流、写入程序或系统ID（通常是相对或绝对URL）来构建数据流结果。

## 使用StAX写XML文档

在前一节中，你看到了如何通过写DOM树的方法来产生XML文件。如果这个DOM树没有其他任何用途，那么这种方式就不是很高效。

StAX API使我们可以直接将XML树写出，这需要从某个OutputStream中构建一个XMLStreamWriter，就像下面这样：

```
XMLOutputFactory factory = XMLOutputFactory.newInstance();
XMLStreamWriter writer = factory.createXMLStreamWriter(out);
```

要产生XML文件头，需要调用

```
writer.writeStartDocument()
```

然后调用

```
writer.writeStartElement(name);
```

添加属性需要调用

```
writer.writeAttribute(name, value);
```

现在，可以通过再次调用writeStartElement添加新的子节点，或者用下面的语句写出字符

```
writer.writeCharacters(text);
```

在添加完所有子节点之后，调用

```
writer.writeEndElement();
```

这会导致当前元素被关闭。

要写出没有子节点的元素（例如<img .../>），可以使用下面的调用

```
writer.writeEmptyElement(name);
```

最后，在文档的结尾，调用

```
writer.writeEndDocument();
```

这个调用将关闭所有的元素。

与使用DOM/XSLT方式一样，我们不必担心属性值和字符数据中的转义字符。但是，我们仍旧有可能会产生非良构的XML，例如具有多个根节点的文档。并且，StAX当前的版本还没有任何对产生缩进输出的支持。

程序清单2-10中的程序展示了写出XML的两种方式。

### 程序清单2-10 XMLWriteTest.java

```
1. import java.awt.*;
2. import java.awt.geom.*;
3. import java.io.*;
4. import java.util.*;
5. import java.awt.event.*;
6. import javax.swing.*;
7. import javax.xml.parsers.*;
8. import javax.xml.stream.*;
9. import javax.xml.transform.*;
10. import javax.xml.transform.dom.*;
11. import javax.xml.transform.stream.*;
12. import org.w3c.dom.*;
13.
14. /**
15. * This program shows how to write an XML file. It saves a file describing a modern drawing
16. * in SVG format.
17. * @version 1.10 2004-09-04
18. * @author Cay Horstmann
19. */
20. public class XMLWriteTest
21. {
22.     public static void main(String[] args)
23.     {
24.         EventQueue.invokeLater(new Runnable()
25.         {
26.             public void run()
27.             {
28.                 XMLWriteFrame frame = new XMLWriteFrame();
29.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
30.                 frame.setVisible(true);
31.             }
32.         });
33.     }
34. }
35.
36. /**
37. * A frame with a component for showing a modern drawing.
```

```
38. */
39. class XMLWriteFrame extends JFrame
40. {
41.     public XMLWriteFrame()
42.     {
43.         setTitle("XMLWriteTest");
44.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
45.
46.         chooser = new JFileChooser();
47.
48.         // add component to frame
49.
50.         comp = new RectangleComponent();
51.         add(comp);
52.
53.         // set up menu bar
54.
55.         JMenuBar menuBar = new JMenuBar();
56.         setJMenuBar(menuBar);
57.
58.         JMenu menu = new JMenu("File");
59.         menuBar.add(menu);
60.
61.         JMenuItem newItem = new JMenuItem("New");
62.         menu.add(newItem);
63.         newItem.addActionListener(new ActionListener()
64.         {
65.             public void actionPerformed(ActionEvent event)
66.             {
67.                 comp.newDrawing();
68.             }
69.         });
70.
71.         JMenuItem saveItem = new JMenuItem("Save with DOM/XSLT");
72.         menu.add(saveItem);
73.         saveItem.addActionListener(new ActionListener()
74.         {
75.             public void actionPerformed(ActionEvent event)
76.             {
77.                 try
78.                 {
79.                     saveDocument();
80.                 }
81.                 catch (Exception e)
82.                 {
83.                     JOptionPane.showMessageDialog(XMLWriteFrame.this, e.toString());
84.                 }
85.             }
86.         });
87.
88.         JMenuItem saveStAXItem = new JMenuItem("Save with StAX");
89.         menu.add(saveStAXItem);
90.         saveStAXItem.addActionListener(new ActionListener()
91.         {
92.             public void actionPerformed(ActionEvent event)
93.             {
```

```
94.         try
95.         {
96.             saveStAX();
97.         }
98.         catch (Exception e)
99.         {
100.             JOptionPane.showMessageDialog(XMLWriteFrame.this, e.toString());
101.         }
102.     }
103. );
104.
105. JMenuItem exitItem = new JMenuItem("Exit");
106. menu.add(exitItem);
107. exitItem.addActionListener(new ActionListener()
108. {
109.     public void actionPerformed(ActionEvent event)
110.     {
111.         System.exit(0);
112.     }
113. });
114. }
115.
116. /**
117. * Saves the drawing in SVG format, using DOM/XSLT
118. */
119. public void saveDocument() throws TransformerException, IOException
120. {
121.     if (chooser.showSaveDialog(this) != JFileChooser.APPROVE_OPTION) return;
122.     File f = chooser.getSelectedFile();
123.     Document doc = comp.buildDocument();
124.     Transformer t = TransformerFactory.newInstance().newTransformer();
125.     t.setOutputProperty(OutputKeys.DOCTYPE_SYSTEM,
126.                         "http://www.w3.org/TR/2000/CR-SVG-20000802/DTD/svg-20000802.dtd");
127.     t.setOutputProperty(OutputKeys.DOCTYPE_PUBLIC, "-//W3C//DTD SVG 20000802//EN");
128.     t.setOutputProperty(OutputKeys.INDENT, "yes");
129.     t.setOutputProperty(OutputKeys.METHOD, "xml");
130.     t.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "2");
131.     t.transform(new DOMSource(doc), new StreamResult(new FileOutputStream(f)));
132. }
133.
134. /**
135. * Saves the drawing in SVG format, using StAX
136. */
137. public void saveStAX() throws FileNotFoundException, XMLStreamException
138. {
139.     if (chooser.showSaveDialog(this) != JFileChooser.APPROVE_OPTION) return;
140.     File f = chooser.getSelectedFile();
141.     XMLOutputFactory factory = XMLOutputFactory.newInstance();
142.     XMLStreamWriter writer = factory.createXMLStreamWriter(new FileOutputStream(f));
143.     comp.writeDocument(writer);
144.     writer.close();
145. }
146.
147. public static final int DEFAULT_WIDTH = 300;
148. public static final int DEFAULT_HEIGHT = 200;
149.
150. private RectangleComponent comp;
```

XML

```
151.     private JFileChooser chooser;
152. }
153.
154. /**
155. * A component that shows a set of colored rectangles
156. */
157. class RectangleComponent extends JComponent
158. {
159.     public RectangleComponent()
160.     {
161.         rects = new ArrayList<Rectangle2D>();
162.         colors = new ArrayList<Color>();
163.         generator = new Random();
164.
165.         DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
166.         try
167.         {
168.             builder = factory.newDocumentBuilder();
169.         }
170.         catch (ParserConfigurationException e)
171.         {
172.             e.printStackTrace();
173.         }
174.     }
175.
176. /**
177. * Create a new random drawing.
178. */
179. public void newDrawing()
180. {
181.     int n = 10 + generator.nextInt(20);
182.     rects.clear();
183.     colors.clear();
184.     for (int i = 1; i <= n; i++)
185.     {
186.         int x = generator.nextInt(getWidth());
187.         int y = generator.nextInt(getHeight());
188.         int width = generator.nextInt(getWidth() - x);
189.         int height = generator.nextInt(getHeight() - y);
190.         rects.add(new Rectangle(x, y, width, height));
191.         int r = generator.nextInt(256);
192.         int g = generator.nextInt(256);
193.         int b = generator.nextInt(256);
194.         colors.add(new Color(r, g, b));
195.     }
196.     repaint();
197. }
198.
199. public void paintComponent(Graphics g)
200. {
201.     if (rects.size() == 0) newDrawing();
202.     Graphics2D g2 = (Graphics2D) g;
203.     // draw all rectangles
204.     for (int i = 0; i < rects.size(); i++)
205.     {
206.         g2.setPaint(colors.get(i));
```

```
207.         g2.fill(rects.get(i));
208.     }
209. }
210.
211. /**
212. * Creates an SVG document of the current drawing.
213. * @return the DOM tree of the SVG document
214. */
215. public Document buildDocument()
216. {
217.     Document doc = builder.newDocument();
218.     Element svgElement = doc.createElement("svg");
219.     doc.appendChild(svgElement);
220.     svgElement.setAttribute("width", "" + getWidth());
221.     svgElement.setAttribute("height", "" + getHeight());
222.     for (int i = 0; i < rects.size(); i++)
223.     {
224.         Color c = colors.get(i);
225.         Rectangle2D r = rects.get(i);
226.         Element rectElement = doc.createElement("rect");
227.         rectElement.setAttribute("x", "" + r.getX());
228.         rectElement.setAttribute("y", "" + r.getY());
229.         rectElement.setAttribute("width", "" + r.getWidth());
230.         rectElement.setAttribute("height", "" + r.getHeight());
231.         rectElement.setAttribute("fill", colorToString(c));
232.         svgElement.appendChild(rectElement);
233.     }
234.     return doc;
235. }
236.
237. /**
238. * Writes an SVG document of the current drawing.
239. * @param writer the document destination
240. */
241. public void writeDocument(XMLStreamWriter writer) throws XMLStreamException
242. {
243.     writer.writeStartDocument();
244.     writer.writeDTD("<!DOCTYPE svg PUBLIC \"-//W3C//DTD SVG 20000802//EN\" "
245.                 + "\\"http://www.w3.org/TR/2000/CR-SVG-20000802/DTD/svg-20000802.dtd\\>");
246.     writer.writeStartElement("svg");
247.     writer.writeAttribute("width", "" + getWidth());
248.     writer.writeAttribute("height", "" + getHeight());
249.     for (int i = 0; i < rects.size(); i++)
250.     {
251.         Color c = colors.get(i);
252.         Rectangle2D r = rects.get(i);
253.         writer.writeEmptyElement("rect");
254.         writer.writeAttribute("x", "" + r.getX());
255.         writer.writeAttribute("y", "" + r.getY());
256.         writer.writeAttribute("width", "" + r.getWidth());
257.         writer.writeAttribute("height", "" + r.getHeight());
258.         writer.writeAttribute("fill", colorToString(c));
259.     }
260.     writer.writeEndDocument(); // closes svg element
261. }
262.
263. /**
```

```
264.     * Converts a color to a hex value.  
265.     * @param c a color  
266.     * @return a string of the form #rrggbb  
267.     */  
268.    private static String colorToString(Color c)  
269.    {  
270.        StringBuffer buffer = new StringBuffer();  
271.        buffer.append(Integer.toHexString(c.getRGB() & 0xFFFFFFFF));  
272.        while (buffer.length() < 6)  
273.            buffer.insert(0, '0');  
274.        buffer.insert(0, '#');  
275.        return buffer.toString();  
276.    }  
277.  
278.    private ArrayList<Rectangle2D> rects;  
279.    private ArrayList<Color> colors;  
280.    private Random generator;  
281.    private DocumentBuilder builder;  
282. }
```

### API javax.xml.stream.XMLOutputFactory 6

- static XMLOutputFactory newInstance()  
    返回这个XMLOutputFactory类的一个实例。
- XMLStreamWriter createXMLStreamWriter(OutputStream in)
- XMLStreamWriter createXMLStreamWriter(OutputStream in, String characterEncoding)
- XMLStreamWriter createXMLStreamWriter(Writer in)
- XMLStreamWriter createXMLStreamWriter(Result in)  
    创建写入给定流、写出器和JAXP结果的写出器。

### API javax.xml.stream.XMLStreamWriter 6

- void writeStartDocument()
- void writeStartDocument(String xmlVersion)
- void writeStartDocument(String encoding, String xmlVersion)  
    在文档的顶部写入XML处理指令。注意，encoding参数只被用来写入这个属性，它不会设置输出的字符编码机制。
- void setDefaultNamespace(String namespaceURI)
- void setPrefix(String prefix, String namespaceURI)  
    设置默认的命名空间，或者具有前缀的命名空间。这种声明的作用域只是当前元素，在没有写入任何元素的情况下，其作用域为文档的根。
- void writeStartElement(String localName)
- void writeStartElement(String namespaceURI, String localName)  
    写出一个开始标签，其中namespaceURI将用相关联的前缀来代替。

- `void writeEndElement()`  
关闭当前元素。
- `void writeEndDocument()`  
关闭所有打开的元素。
- `void writeEmptyElement(String localName)`
- `void writeEmptyElement(String namespaceURI, String localName)`  
写出一个自闭合的标签，其中namespaceURI将用相关联的前缀来代替。
- `void writeAttribute(String localName, String value)`
- `void writeAttribute(String namespaceURI, String localName, String value)`  
写出一个用于当前元素的属性，其中namespaceURI将用相关联的前缀来代替。
- `void writeCharacters(String text)`  
写出字符数据。
- `void writeCData(String text)`  
写出CDATA块。
- `void writeDTD(String dtd)`  
写出dtd字符串，该字串需要包含一个DOCTYPE声明。
- `void writeComment(String comment)`  
写出一个注释。
- `void close()`  
关闭这个写出器。

## 2.8 XSL转换

XSL转换（XSLT）机制可以指定将XML文档转换为其他格式的规则，例如，纯文本、 XHTML或其他任何XML格式。XSLT通常用来将一个机器可读的XML格式转译为另一种机器可读的XML格式，或者将XML转译为适于人类阅读的表示格式。

你需要提供XSLT样式表，它描述了XML文档向某种其他格式转换的规则。XSLT处理器将读入XML文档和这个样式表，并产生所要的输出（请查看图2-8）。

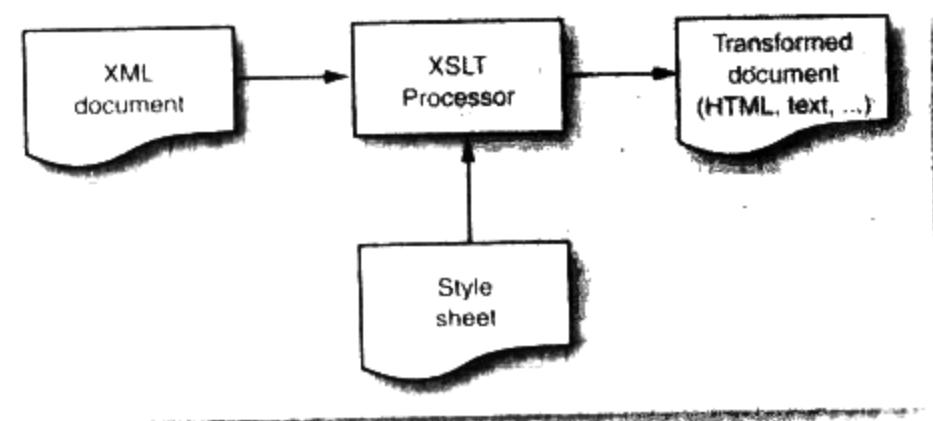


图2-8 应用XSL 转换

下面是一个典型的例子。我们想要把有雇员记录的XML文件转换成HTML文件。请看这个输入文件：

```
<staff>
  <employee>
    <name>Carl Cracker</name>
    <salary>75000</salary>
    <hiredate year="1987" month="12" day="15"/>
  </employee>
  <employee>
    <name>Harry Hacker</name>
    <salary>50000</salary>
    <hiredate year="1989" month="10" day="1"/>
  </employee>
  <employee>
    <name>Tony Tester</name>
    <salary>40000</salary>
    <hiredate year="1990" month="3" day="15"/>
  </employee>
</staff>
```

我们希望的输出是一张HTML表格：

```
<table border="1">
<tr>
<td>Carl Cracker</td><td>$75000.0</td><td>1987-12-15</td>
</tr>
<tr>
<td>Harry Hacker</td><td>$50000.0</td><td>1989-10-1</td>
</tr>
<tr>
<td>Tony Tester</td><td>$40000.0</td><td>1990-3-15</td>
</tr>
</table>
```

XSLT规范是很复杂的，已经有很多书描述了该主题。我们不可能讨论XSLT的全部特性，所以我们只能介绍一个有代表性的例子。你可以在Don Box等人合著的《Essential XML》一书（Addison-Wesley Professional 2000）中找到更多的信息。XSLT规范可以在<http://www.w3.org/TR/xslt>获得。

具有转换模板的样式表形式如下：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:output method="html"/>
  template1
  template2
  ...
</xsl:stylesheet>
```

在我们的例子中，`xsl:output`元素将方法设定为HTML。而其他有效的方法设置是`xml`和`text`。

下面是一个典型的模板：

```
<xsl:template match="/staff/employee">
  <tr><xsl:apply-templates/></tr>
</xsl:template>
```

**match**属性的值是一个XPath表达式。该模板说明，每当看到一个XPath集/staff/employee中的一个节点时，做以下操作：

1. 产生字符串<tr>。
2. 对于要处理的子元素继续应用模板。
3. 当处理完所有子元素后，产生字符串</tr>。

换句话说，该模板围绕每个雇员记录生成HTML表格的行标记。

XSLT处理器由检查根元素开始其处理过程。每当一个节点匹配某个模板时，就会应用该模板（如果匹配多个模板，就会使用最佳匹配的那个，详情请参见<http://www.w3.org/TR/xslt>）。如果没有匹配的模板，处理器执行默认操作。对于文本节点，默认操作是把它的内容包含到输出中去。对于元素，默认操作是不产生输出的，但会继续处理其子节点。

下面是一个原来转换雇员记录文件中的名字节点的模板：

```
<xsl:template match="/staff/employee/name">
  <td><xsl:apply-templates/></td>
</xsl:template>
```

正如你知道的，模板产生定界符<td>...</td>，并且请求处理器递归访问name元素的子节点。它只有一个子节点，即文本节点。当处理器访问该节点时，它产生文本内容（当然，前提是没有任何其他匹配的模板）。

如果想要把属性值复制到输出中去，就不得不做稍微复杂一些的操作了。下面是一个例子：

```
<xsl:template match="/staff/employee/hiredate">
  <td><xsl:value-of select="@year"/>-<xsl:value-of
    select="@month"/>-<xsl:value-of select="@day"/></td>
</xsl:template>
```

当处理hiredate节点时，该模板会产生：

- 字符串<td>
- year属性的值
- 一个连字符
- month属性的值
- 一个连字符
- day属性的值
- 一个连字符
- 字符串</td>

**xsl:value-of**语句用于计算节点集的字符串值。节点集由**select**属性的XPath值设定。在这个例子中，该路径是相对于当前处理节点的相对路径。节点集通过连接各个节点的字符串值被转换成一个字符串。属性节点的字符串值就是它的值，文本节点的字符串值是它的内容，元素节点的字符串值是它子节点的字符串值的连接（但不是它的属性）。

程序清单2-11包含了将带有雇员记录的XML文件转换成HTML表格的样式表。

**程序清单2-11 makehtml.xsl**

```
1. <?xml version="1.0" encoding="ISO-8859-1"?>
2.
3. <xsl:stylesheet
4.   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
5.   version="1.0">
6.
7.   <xsl:output method="html"/>
8.
9.   <xsl:template match="/staff">
10.     <table border="1"><xsl:apply-templates/></table>
11.   </xsl:template>
12.
13.   <xsl:template match="/staff/employee">
14.     <tr><xsl:apply-templates/></tr>
15.   </xsl:template>
16.
17.   <xsl:template match="/staff/employee/name">
18.     <td><xsl:apply-templates/></td>
19.   </xsl:template>
20.
21.   <xsl:template match="/staff/employee/salary">
22.     <td>$<xsl:apply-templates/></td>
23.   </xsl:template>
24.
25.   <xsl:template match="/staff/employee/hiredate">
26.     <td><xsl:value-of select="@year"/>-<xsl:value-of
27.       select="@month"/>-<xsl:value-of select="@day"/></td>
28.   </xsl:template>
29.
30. </xsl:stylesheet>
```

程序清单2-12显示了一组不同的转换。其输入是同一个XML文件，其输出是我们熟悉的属性文件格式的纯文本。

**程序清单2-12 makeprop.xsl**

```
1. <?xml version="1.0"?>
2.
3. <xsl:stylesheet
4.   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
5.   version="1.0">
6.
7.   <xsl:output method="text"/>
8.   <xsl:template match="/staff/employee">
9.     employee.<xsl:value-of select="position()"/>.name=<xsl:value-of select="name/text()" />
10.    employee.<xsl:value-of select="position()"/>.salary=<xsl:value-of select="salary/text()" />
11.    employee.<xsl:value-of select="position()"/>.hiredate=<xsl:value-of select="hiredate/@year" />
12.      -<xsl:value-of select="hiredate/@month" />-<xsl:value-of select="hiredate/@day" />
13.   </xsl:template>
14.
15. </xsl:stylesheet>
```

```
employee.1.name=Carl Cracker
employee.1.salary=75000.0
```

```

employee.1.hiredate=1987-12-15
employee.2.name=Harry Hacker
employee.2.salary=50000.0
employee.2.hiredate=1989-10-1
employee.3.name=Tony Tester
employee.3.salary=40000.0
employee.3.hiredate=1990-3-15

```

该示例使用position()函数来产生以其父节点的角度来看的当前节点的位置。我们只要切换样式表就可以得到一个完全不同的输出。这样，就可以安全地使用XML来描述数据了，即便一些应用程序需要的是其他格式的数据，我们只要用XSLT来产生该种替代格式即可。在Java平台上产生XML的转换极其简单，只需为每个样式表设置一个转换器工厂，然后得到一个转换器对象，并告诉它把一个源转换成结果。

```

File styleSheet = new File(filename);
StreamSource styleSource = new StreamSource(styleSheet);
Transformer t = TransformerFactory.newInstance().newTransformer(styleSource);
t.transform(source, result);

```

Transform方法的参数是Source和Result接口的实现类的对象。Source接口有3个实现类：  
DOMSource  
SAXSource  
StreamSource

可以根据一个文件、流、阅读器、URL或来自DOM树节点的DOMSource来构建一个StreamSource。例如，在上一节中，我们调用了如下的身份转换：

```
t.transform(new DOMSource(doc), result);
```

在示例程序中，我们做了一些更有趣的事情。我们不是从一个现有的XML文件开始工作，我们要产生一个SAX XML阅读器，通过产生适当的SAX事件，给人以解析XML文件的错觉。实际上，XML阅读器读入的是一个如第1章所描述的平面文件，而输入文件看上去是这样的：

```

Carl Cracker|75000.0|1987|12|15
Harry Hacker|50000.0|1989|10|1
Tony Tester|40000.0|1990|3|15

```

处理输入时，XML阅读器将产生SAX事件。下面是实现XMLReader接口的EmployeeReader类的parse方法的一部分。

```

AttributesImpl attributes = new AttributesImpl();
handler.startDocument();
handler.startElement("", "staff", "staff", attributes);
while ((line = in.readLine()) != null)
{
    handler.startElement("", "employee", "employee", attributes);
    StringTokenizer t = new StringTokenizer(line, "|");
    handler.startElement("", "name", "name", attributes);
    String s = t.nextToken();
    handler.characters(s.toCharArray(), 0, s.length());
    handler.endElement("", "name", "name");
    ...
    handler.endElement("", "employee", "employee");
}
handler.endElement("", rootElement, rootElement);
handler.endDocument();

```

用于转换器的SAXSource是根据XML阅读器构建的：

```
t.transform(new SAXSource(new EmployeeReader(),
    new InputSource(new FileInputStream(filename))), result);
```

这是一个将非XML的遗留数据转换成XML的一个小技巧。当然，大多数XSLT应用程序都已经有了XML格式的输入数据，只需要在一个StreamSource对象上调用transform方法即可，例如：

```
t.transform(new StreamSource(file), result);
```

其转换结果是Result接口实现类的一个对象。Java库提供了3个类：

```
DOMResult  
SAXResult  
StreamResult
```

如果要把结果存储到DOM树中，请使用DocumentBuilder产生一个新的文档节点，并将其包装到DOMResult中：

```
Document doc = builder.newDocument();
t.transform(source, new DOMResult(doc));
```

如果要将输出保存为文件，请使用StreamResult：

```
t.transform(source, new StreamResult(file));
```

程序清单2-13包含了完整的源代码。

### 程序清单2-13 TransformTest.java

```
1. import java.io.*;
2. import java.util.*;
3. import javax.xml.transform.*;
4. import javax.xml.transform.sax.*;
5. import javax.xml.transform.stream.*;
6. import org.xml.sax.*;
7. import org.xml.sax.helpers.*;
8.
9. /**
10. * This program demonstrates XSL transformations. It applies a transformation to a set
11. * of employee records. The records are stored in the file employee.dat and turned into XML
12. * format. Specify the stylesheet on the command line, e.g. java TransformTest makeprop.xsl
13. * @version 1.01 2007-06-25
14. * @author Cay Horstmann
15. */
16. public class TransformTest
17. {
18.     public static void main(String[] args) throws Exception
19.     {
20.         String filename;
21.         if (args.length > 0) filename = args[0];
22.         else filename = "makehtml.xsl";
23.         File styleSheet = new File(filename);
24.         StreamSource styleSource = new StreamSource(styleSheet);
25.
26.         Transformer t = TransformerFactory.newInstance().newTransformer(styleSource);
27.         t.setOutputProperty(OutputKeys.INDENT, "yes");
28.         t.setOutputProperty(OutputKeys.METHOD, "xml");
```

```
29.     t.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "2");
30.
31.     t.transform(new SAXSource(new EmployeeReader(), new InputSource(new FileInputStream(
32.         "employee.dat"))), new StreamResult(System.out));
33. }
34. */
35. /**
36. * This class reads the flat file employee.dat and reports SAX parser events to act as if
37. * it was parsing an XML file.
38. */
39. class EmployeeReader implements XMLReader
40. {
41.     public void parse(InputSource source) throws IOException, SAXException
42.     {
43.         InputStream stream = source.getByteStream();
44.         BufferedReader in = new BufferedReader(new InputStreamReader(stream));
45.         String rootElement = "staff";
46.         AttributesImpl atts = new AttributesImpl();
47.
48.         if (handler == null) throw new SAXException("No content handler");
49.
50.         handler.startDocument();
51.         handler.startElement("", rootElement, rootElement, atts);
52.         String line;
53.         while ((line = in.readLine()) != null)
54.         {
55.             handler.startElement("", "employee", "employee", atts);
56.             StringTokenizer t = new StringTokenizer(line, "|");
57.
58.             handler.startElement("", "name", "name", atts);
59.             String s = t.nextToken();
60.             handler.characters(s.toCharArray(), 0, s.length());
61.             handler.endElement("", "name", "name");
62.
63.             handler.startElement("", "salary", "salary", atts);
64.             s = t.nextToken();
65.             handler.characters(s.toCharArray(), 0, s.length());
66.             handler.endElement("", "salary", "salary");
67.
68.             atts.addAttribute("", "year", "year", "CDATA", t.nextToken());
69.             atts.addAttribute("", "month", "month", "CDATA", t.nextToken());
70.             atts.addAttribute("", "day", "day", "CDATA", t.nextToken());
71.             handler.startElement("", "hiredate", "hiredate", atts);
72.             handler.endElement("", "hiredate", "hiredate");
73.             atts.clear();
74.
75.             handler.endElement("", "employee", "employee");
76.         }
77.
78.         handler.endElement("", rootElement, rootElement);
79.         handler.endDocument();
80.     }
81.
82.     public void setContentHandler(ContentHandler newValue)
83.     {
84.         handler = newValue;
```

```
85.    }
86.
87.   public ContentHandler getContentHandler()
88.   {
89.     return handler;
90.   }
91.
92.   // the following methods are just do-nothing implementations
93.   public void parse(String systemId) throws IOException, SAXException
94.   {
95.   }
96.
97.   public void setErrorHandler(ErrorHandler handler)
98.   {
99.   }
100.
101.  public ErrorHandler getErrorHandler()
102.  {
103.    return null;
104.  }
105.
106.  public void setDTDHandler(DTDHandler handler)
107.  {
108.  }
109.
110.  public DTDHandler getDTDHandler()
111.  {
112.    return null;
113.  }
114.
115.  public void setEntityResolver(EntityResolver resolver)
116.  {
117.  }
118.
119.  public EntityResolver getEntityResolver()
120.  {
121.    return null;
122.  }
123.
124.  public void setProperty(String name, Object value)
125.  {
126.  }
127.
128.  public Object getProperty(String name)
129.  {
130.    return null;
131.  }
132.
133.  public void setFeature(String name, boolean value)
134.  {
135.  }
136.
137.  public boolean getFeature(String name)
138.  {
139.    return false;
```

```
141.  
142.     private ContentHandler handler;  
143. }
```

**API** **javax.xml.transform.TransformerFactory 1.4**

- `transformer newTransformer(Source stylesheet)`

返回一个`transformer`类的实例，用来从指定的源中读取样式表。

**API** **javax.xml.transform.stream.StreamSource 1.4**

- `StreamSource(File f)`
- `StreamSource(InputStream in)`
- `StreamSource(Reader in)`
- `StreamSource(String systemID)`

根据一个文件、流、阅读器或系统ID（通常是相对或绝对URL）来构建一个数据流源。

**API** **javax.xml.transform.sax.SAXSource 1.4**

- `SAXSource(XMLReader reader, InputSource source)`

构建一个SAX数据源，以便从给定输入源获取数据，并使用给定的阅读器来解析输入数据。

**API** **org.xml.sax.XMLReader 1.4**

- `void setContentHandler(ContentHandler handler)`

设置在输入被解析时会被告知解析事件的处理器。

- `void parse(InputSource source)`

根据给定输入源解析输入数据，并将解析事件发送到内容处理器。

**API** **javax.xml.transform.dom.DOMResult 1.4**

- `DOMResult(Node n)`

根据给定节点构建一个数据源。通常，n是一个新文档节点。

**API** **org.xml.sax.helpers.AttributesImpl 1.4**

- `void addAttribute(String uri, String lname, String qname, String type, String value)`

将一个属性添加到该属性集合。

参数：uri 名字空间的URI

lname 无别名前缀的本地名

qname 带别名前缀的限定名

type 类型，“CDATA”、“ID”、“IDREF”、“IDREFS”、“NMTOKEN”、“NMTOKENS”、“ENTITY”、“ENTITIES”或“NOTATION”之一

value 属性值

- `void clear()`

删除属性集合中的所有属性。

我们以该示例结束对Java库中的XML支持特性的讨论。现在，你应该对XML的强大功能有了很好的了解，尤其是它的自动解析、验证和强大的转换机制。当然，所有这些技术只有在你很好地设计了XML格式之后才能发挥作用。你必须确保那些格式足够丰富，能够表达全部业务需求，随着时间的推移也依旧稳定，你的业务伙伴也愿意接受你的XML文档。这些问题要远比处理解析器、DTD或转换特性更具挑战。

在下一章，我们将讨论在Java平台上的网络编程，从最基础的网络套接字开始，逐渐过渡到用于E-mail和万维网的更高层协议。

# 第3章 网络

- ▲ 连接到服务器
- ▲ 实现服务器
- ▲ 可中断套接字

- ▲ 发送E-mail
- ▲ 建立URL连接

本章的开头部分将首先回顾一下网络方面的基本概念，然后进一步介绍如何编写连接网络服务的Java程序，并演示网络客户端和服务器是如何实现的，最后将介绍如何通过Java程序发送E-mail，以及如何从Web服务器获得信息。

## 3.1 连接到服务器

在编写第一个网络程序之前，首先让我们来了解一下telnet这个你已经拥有的工具，它对我们调试网络程序非常有帮助。大多数系统都已经预装了telnet，你可以在命令shell中输入telnet来启动它。

 **注意：**在Windows Vista中，telnet是安装了的，但是默认情况下是未激活的。要激活它，需要到“控制面板”，选择“程序”，点击“打开/关闭Windows特性”，然后选择“Telnet客户端”复选框。Windows防火墙将会阻止我们在本章中使用的很多网络端口，你可能需要管理账户才能解除对它们的禁用。

你可能曾经使用过telnet来连接远程计算机，但其实你也可以用它与因特网主机所提供的其他服务进行通信。下面是一个可以操作的例子。请输入：

```
telnet time-A.timefreq.bldrdoc.gov 13
```

如图3-1所示，你可以得到与下面这一行相似的信息：

```
54276 07-06-25 21:37:31 50 0 0 659.0 UTC(NIST) *
```

上面例子说明了什么？它说明你已经连接到了大多数UNIX计算机都支持的“当日时间”服务。而你刚才所连接的那台服务器就设在美国科罗拉多州博尔德市的国家标准与技术研究所，这家研究所负责提供铯原子钟的计量时间。（当然，由于网络延迟的缘故，原子钟反馈过来的时间并不完全准确。）

按照惯例，“当日时间”服务总是连接到端口13。

 **注意：**在网络术语中，端口并不是指物理设备，而是为了便于实现服务器与客户端之间通信所使用的抽象概念（见图3-2）。

运行在远程计算机上的服务器软件不停地等待那些希望与端口13连接的网络请求。当远程计算机上的操作系统接收到一个请求与端口13连接的网络数据包时，它便唤醒正在监听网络连接请

求的服务器进程，并为两者建立连接。这种连接将一直保持下去，直到被其中任何一方中止。

```
Terminal
File Edit View Terminal Tabs Help
$ telnet time-A.timefreq.bldrdoc.gov 13
Trying 132.163.4.103...
Connected to time-A.timefreq.bldrdoc.gov.
Escape character is '^]'.
54276 07-06-25 21:37:31 50 0 0 659.0 UTC(NIST) *
Connection closed by foreign host.
$
```

图3-1 “当日时间”服务的输出

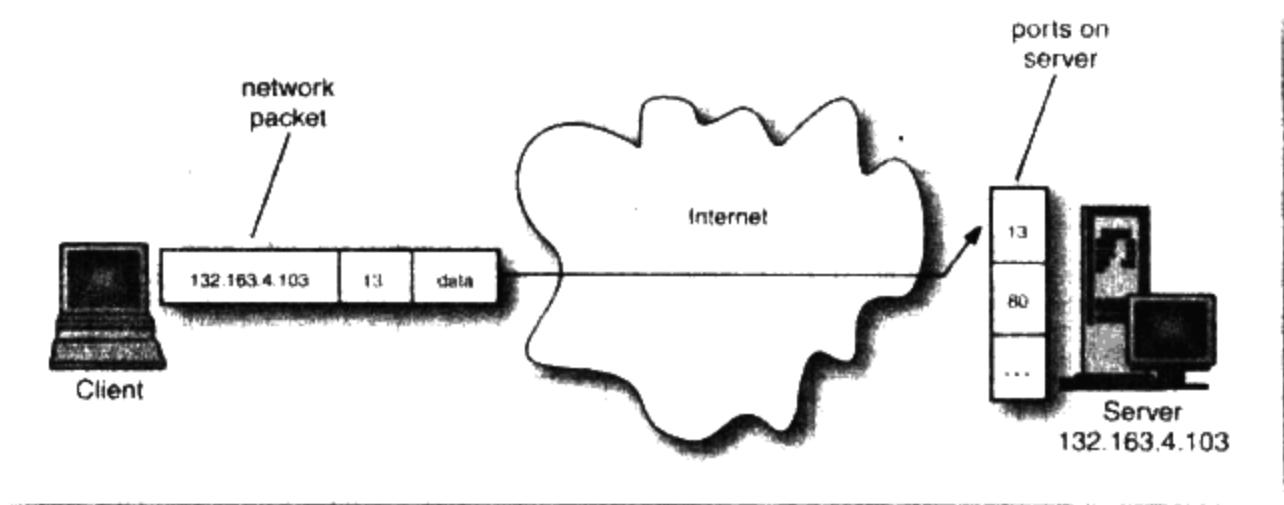


图3-2 连接到服务器端口的客户端

当你开始用time-A.timefreq.bldrdoc.gov在端口13上建立telnet会话时，有一个网络软件非常清楚地知道应该将字符串“time-A.timefreq.bldrdoc.gov”转换为正确的IP地址132.163.4.103。随后，这个telnet软件发送一个连接请求给该地址，请求一个到端口13的连接。一旦建立连接，远程程序便发送回一行数据，然后关闭该连接。当然，一般而言，客户端和服务器在其中一方关闭连接之前，会进行更多的对话。

下面是另一个同类的试验，但它更加有趣。请执行以下操作：

1. 使用telnet连接到java.sun.com上的端口80。
2. 请准确输入下面这行文字，勿按回车键。注意只在第一个斜杠两侧有空格，而第二个斜杠两侧没有空格。

GET / HTTP/1.0

3. 现在按两次Enter键。

图3-3显示了以上操作返回的结果。它看上去应该是你非常熟悉的——你得到的是一个HTML格式的文本页，即Java技术的主页。

上述操作与Web浏览器访问某个网页所经历的过程是完全一致的，它使用HTTP从服务器请求Web页面。当然，浏览器能够更精致地显示了HTML代码。

```

Terminal
File Edit View Terminal Help
$ telnet java.sun.com 80
Trying 72.5.124.55...
Connected to java.sun.com.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Server: Sun-Java-System-Web-Server-6.1
Date: Mon, 25 Jun 2007 21:42:38 GMT
Content-type: text/html; charset=ISO-8859-1
Set-cookie: JSESSIONID=14188FFAC3EAC882A0C92643F21B2FDA; Path=/
Connection: close

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Java Technology</title>
<meta name="keywords" content="Java, platform">
<meta name="collection" content="reference">
<meta name="description" content="Java technology is a portfolio of products that are based on the power of networks and the idea that the same software should run on many different kinds of systems and devices.">
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

```

图3-3 使用telnet访问HTTP端口

注意：如果你尝试此过程时所使用的Web服务器使用相同的IP地址为多个域提供宿主环境，那么你将无法得到想要的Web页面（一般多个较小的Web网站共享单一服务器就属于这种情况，如horstmann.com）。当连接到这样的服务器上时，需要指定想要连接的主机名，例如：

```
GET / HTTP/1.1
Host: horstmann.com
```

然后连续按下回车键两次（注意这里的HTTP版本是1.1）。

程序清单3-1是我们的第一个网络程序。它的作用与我们使用telnet工具是一致的，即连接到某个端口并打印出它所找到的信息。

### 程序清单3-1 SocketTest.java

```

1. import java.io.*;
2. import java.net.*;
3. import java.util.*;
4.
5. /**
6. * This program makes a socket connection to the atomic clock in Boulder, Colorado, and
7. * prints the time that the server sends.
8. * @version 1.20 2004-08-03
9. * @author Cay Horstmann
10.*/
11. public class SocketTest
12. {
13.     public static void main(String[] args)
14.     {
15.         try
16.         {
17.             Socket s = new Socket("time-A.timefreq.bldrdoc.gov", 13);
18.             try
19.             {

```

```
20.     InputStream inStream = s.getInputStream();
21.     Scanner in = new Scanner(inStream);
22.
23.     while (in.hasNextLine())
24.     {
25.         String line = in.nextLine();
26.         System.out.println(line);
27.     }
28. }
29. finally
30. {
31.     s.close();
32. }
33. }
34. catch (IOException e)
35. {
36.     e.printStackTrace();
37. }
38. }
39. }
```

下面是这个简单程序的几行关键代码：

```
Socket s = new Socket("time-A.timefreq.bldrdoc.gov", 13);
InputStream inStream = s.getInputStream();
```

第一行代码用于打开一个套接字，它也是网络软件中的一个抽象概念，负责使能该程序内部和外部之间的通信。我们将远程地址和端口号传递给套接字的构造器，如果连接失败，它将抛出一个`UnknownHostException`异常；如果存在其他问题，它将抛出一个`IOException`异常。因为`UnknownHostException`是`IOException`的一个子类，况且这只是一个示例程序，所以我们在这里仅仅捕获超类的异常。

一旦套接字被打开，`java.net.Socket`类中的`getInputStream`方法就会返回一个`InputStream`对象。而一旦获取了这个流，该程序将直接把每一行打印到标准输出。这个过程将一直持续到流发送完毕且服务器断开连接为止。

该程序只适用于非常简单的服务器，比如“当日时间”之类的服务程序。在比较复杂的网络程序中，客户端发送请求数据给服务器，而服务器可能在响应结束时并不立刻断开连接。在本章的若干个示例程序中，都会看到我们是如何实现这种行为的。

`Socket`类非常简单易用，因为Java技术隐藏了建立网络连接和通过连接发送数据的复杂过程。实际上，`java.net`包提供的编程接口与操作文件时所使用的接口基本相同。



**注意：**本书所介绍的内容仅适用于TCP（传输控制协议）网络协议。Java平台另外还支持所谓的UDP（用户数据报协议）协议，该协议可以用于发送数据包（也称为数据报），它所需付出的开销要比TCP少得多。UDP有一个重要的缺点：数据包无需按照顺序进行传递，它们甚至可能在传输过程中全部丢失。UDP要求数据包的接受者对它们进行排序，并请求发送者重新发送那些丢失的数据包。UDP比较适合于那些可以忍受数据包丢失的应用，例如用于音频流和视频流的传输，或者用于连续测量的应用领域。

**API** **java.net.Socket 1.0**

- `Socket(String host, int port)`

构建一个套接字，用来连接给定的主机和端口。

- `InputStream getInputStream()`

- `OutputStream getOutputStream()`

获取可以从套接字中读取数据的流，以及可以向套接字写出数据的流。

### 3.1.1 套接字超时

从套接字读取信息时，在可以访问数据之前，读操作将会被阻塞。如果此时主机不可达，那么应用将要等待很长的时间，并且因为受底层操作系统的限制而最终会导致超时。

对于不同的应用，应该确定合理的超时值。然后调用`setSoTimeout`方法设置这个超时值（单位：毫秒）。

```
Socket s = new Socket(. . .);
s.setSoTimeout(10000); // time out after 10 seconds
```

如果已经为套接字设置了超时值，并且之后的读操作和写操作在没有完成之前就超过了时间限制，那么这些操作就会抛出`SocketTimeoutException`异常。你可以捕获这个异常，并对超时做出反应。

```
try
{
    InputStream in = s.getInputStream(); // read from in
    ...
}
catch (InterruptedIOException exception)
{
    react to timeout
}
```

另外还有一个超时问题是必须解决的。下面这个构造器：

```
Socket(String host, int port)
```

会一直无限期地阻塞下去，直到建立了到达主机的初始连接为止。

可以通过先构建一个无连接的套接字，然后再使用一个超时来进行连接的方法解决这个问题。

```
Socket s = new Socket();
s.connect(new InetSocketAddress(host, port), timeout);
```

如果你希望允许用户在任何时刻都可以中断套接字连接，请查看从第3.3节。

**API** **java.net.Socket 1.0**

- `Socket()`

创建一个还未被连接的套接字。

- `void connect(SocketAddress address) 1.4`

将该套接字连接到给定的地址。

- `void connect(SocketAddress address, int timeoutInMilliseconds) 1.4`

将套接字连接到给定的地址。如果在给定的时间内没有响应，则返回。

- void setSoTimeout(int timeoutInMilliseconds) 1.1

设置该套接字上读请求的阻塞时间。如果超出给定时间，则抛出一个InterruptedIOException异常。

- boolean isConnected() 1.4

如果该套接字已被连接，则返回true。

- boolean isClosed() 1.4

如果套接字已经被关闭，则返回true。

### 3.1.2 因特网地址

因特网地址是指用一串数字表示的主机地址，它由4个字节组成，IPv6规定为16个字节，比如132.163.4.102。通常，不用过多考虑因特网地址的问题。但是，如果需要在主机名和因特网地址之间进行转换时，可以使用 InetAddress类。

只要主机操作系统支持IPv6格式的因特网地址，java.net包也将支持它。

静态的getByName方法可以返回代表某个主机的InetAddress对象。例如，

```
InetAddress address = InetAddress.getByName("time-A.timefreq.bldrdoc.gov");
```

将返回一个InetAddress对象，该对象封装了一个4字节的序列：132.163.4.104。然后，可以使用getAddress方法来访问这些字节。

```
byte[] addressBytes = address.getAddress();
```

一些访问量较大的主机名通常会对应于多个因特网地址，以实现负载均衡。例如，在撰写本书时，主机名java.sun.com就对应着3个不同的因特网地址。当访问主机时，其因特网地址将从这三者中随机产生。可以通过调用getAllByName方法来获得所有主机。

```
InetAddress[] addresses = InetAddress.getAllByName(host);
```

最后需要说明的是，有时可能需要知道本地主机的地址。如果只是要求得到localhost的地址，那总会得到地址127.0.0.1，但是其他程序无法用这个地址来连接到这台机器上。此时，可以使用静态的getLocalHost方法来得到本地主机的地址。

```
InetAddress address = InetAddress.getLocalHost();
```

程序清单3-2是一段比较简单的程序代码。如果不在命令行中设置任何参数，那么它将打印出本地主机的因特网地址。反之，如果在命令行中指定了主机名，那么它将打印出该主机的所有因特网地址，例如：

```
java InetAddressTest java.sun.com
```

#### 程序清单3-2 InetAddressTest.java

```
1. import java.net.*;
2.
3. /**
4. * This program demonstrates the InetAddress class. Supply a host name as command line
5. * argument, or run without command line arguments to see the address of the local host.
6. * @version 1.01 2001-06-26
7. * @author Cay Horstmann
```

```

8. */
9. public class InetAddressTest
10. {
11.     public static void main(String[] args)
12.     {
13.         try
14.         {
15.             if (args.length > 0)
16.             {
17.                 String host = args[0];
18.                 InetAddress[] addresses = InetAddress.getAllByName(host);
19.                 for (InetAddress a : addresses)
20.                     System.out.println(a);
21.             }
22.             else
23.             {
24.                 InetAddress localHostAddress = InetAddress.getLocalHost();
25.                 System.out.println(localHostAddress);
26.             }
27.         }
28.         catch (Exception e)
29.         {
30.             e.printStackTrace();
31.         }
32.     }
33. }

```

### API java.net.InetAddress 1.0

- `static InetAddress getByName(String host)`
- `static InetAddress[] getAllByName(String host)`

为给定的主机名，创建一个InetAddress对象，或者一个包含了该主机名所对应的所有因特网地址的数组。

- `static InetAddress getLocalHost()`

为本地主机创建一个InetAddress对象。

- `byte[] getAddress()`

返回一个包含数字型地址的字节数组。

- `String getHostAddress()`

返回一个由十进制数组成的字符串，各数字间用圆点符号隔开，例如，“132.163.4.102”。

- `String getHostName()`

返回主机名。

## 3.2 实现服务器

既然我们已经实现了一个基本的网络客户端，并且用它从因特网上获取了数据，那么现在就让我们再来实现一个简单的服务器，它可以向客户端发送信息。一旦启动服务器程序，它便

等待某个客户端连接到它的端口。我们选择端口号8189，因为所有标准服务都不使用这个端口。`ServerSocket`类用于建立一个套接字。在我们的示例中，下面这行命令：

```
ServerSocket s = new ServerSocket(8189);
```

用于建立一个负责监控端口8189的服务器。以下命令：

```
Socket incoming = s.accept();
```

用于告诉程序不停地等待，直到有客户端连接到这个端口。一旦有人通过网络向该端口发送了正确的连接请求，该方法就会返回一个表示连接已经建立的`Socket`对象。你可以使用这个对象来得到输入流和输出流，代码如下：

```
InputStream inStream = incoming.getInputStream();
OutputStream outStream = incoming.getOutputStream();
```

服务器发送给服务器输出流的所有信息都会成为客户端程序的输入，同时来自客户端程序的所有输出都会被包含在服务器输入流中。

因为在本章的所有示例程序中，我们都要通过套接字来发送文本，所以我们将流转换成扫描器和写入器。

```
Scanner in = new Scanner(inStream);
PrintWriter out = new PrintWriter(outStream, true /* autoFlush */);
```

以下代码将给客户端发送一条问候信息：

```
out.println("Hello! Enter BYE to exit.");
```

当使用telnet通过端口8189连接到这个服务器程序时，将会在终端屏幕上看到上述问候信息。

在这个简单的服务器程序中，它仅仅只是读取客户端输入，每次读取一行，并回送这一行。这表明程序接收到了客户端的输入。当然，实际应用中的服务器都会对输入进行计算并返回处理结果。

```
String line = in.nextLine();
out.println("Echo: " + line);
if (line.trim().equals("BYE")) done = true;
```

在代码的最后，我们关闭连接进来的套接字。

```
incoming.close();
```

这就是整个示例代码的大致情况。每一个服务器程序，比如一个HTTP Web服务器，都不间断地执行下面这个循环操作：

1. 通过输入数据流从客户端接收一个命令（“get me this information”）。
2. 解码这个命令。
3. 收集客户端所请求的信息。
4. 通过输出数据流发送信息给客户端。

程序清单3-3给出了这个程序的完整代码。

### 程序清单3-3 EchoServer.java

```
1. import java.io.*;
2. import java.net.*;
3. import java.util.*;
4.
```

```
5. /**
6. * This program implements a simple server that listens to port 8189 and echoes back all
7. * client input.
8. * @version 1.20 2004-08-03
9. * @author Cay Horstmann
10.*/
11. public class EchoServer
12. {
13.     public static void main(String[] args)
14.     {
15.         try
16.         {
17.             // establish server socket
18.             ServerSocket s = new ServerSocket(8189);
19.
20.             // wait for client connection
21.             Socket incoming = s.accept();
22.             try
23.             {
24.                 InputStream inStream = incoming.getInputStream();
25.                 OutputStream outStream = incoming.getOutputStream();
26.
27.                 Scanner in = new Scanner(inStream);
28.                 PrintWriter out = new PrintWriter(outStream, true /* autoFlush */);
29.
30.                 out.println("Hello! Enter BYE to exit.");
31.
32.                 // echo client input
33.                 boolean done = false;
34.                 while (!done && in.hasNextLine())
35.                 {
36.                     String line = in.nextLine();
37.                     out.println("Echo: " + line);
38.                     if (line.trim().equals("BYE")) done = true;
39.                 }
40.             }
41.             finally
42.             {
43.                 incoming.close();
44.             }
45.         }
46.         catch (IOException e)
47.         {
48.             e.printStackTrace();
49.         }
50.     }
51. }
```

想要试一下这个例子，就请编译并运行这个程序。然后使用telnet连接到服务器localhost（或IP地址127.0.0.1）和端口8189。

如果你直接连接到因特网上，那么世界上任何人都可以访问到你的回送服务器，只要他们知道你的IP地址和端口号。

当你连接到该端口时，将看到如图3-4的信息：

Hello! Enter BYE to exit.

```
Terminal
File Edit View Terminal Help
$ telnet localhost 8189
Trying 127.0.0.1...
Connected to localhost
Escape character is '^]'.
Hello! Enter BYE to exit
Hello Sailor!
Echo: Hello Sailor!
BYE
Echo: BYE
Connection closed by foreign host.
-$
```

图3-4 访问一个回送服务器

随意键入一条信息，观察屏幕上的回送信息。输入BYE（全为大写字母）可以断开连接，同时，服务器程序也会终止运行。

#### API `java.net.ServerSocket` 1.0

- `ServerSocket(int port)`

创建一个监控端口的服务器套接字。

- `Socket accept()`

等待连接。该方法阻塞（即，使之空闲）当前线程直到建立连接为止。该方法返回一个套接字对象，程序可以通过这个对象与连接中的客户端进行通信。

- `void close()`

关闭服务器套接字。

### 3.2.1 为多个客户端服务

前面例子中的简单服务器存在一个问题。我们可能希望有多个客户端同时连接到我们的服务器上。通常，服务器总是不间断地运行在服务器计算机上，来自整个因特网的用户希望同时使用服务器。拒绝多客户端连接将使得某个用户可能会因长时间地连接服务而独占服务。其实我们可以运用线程的魔力把这个问题解决得更好。

每当程序建立一个新的套接字连接，也就是说当成功调用`accept`的时候，将创建一个新的线程来处理服务器和该客户端之间的连接，而主程序将立即返回并等待下一个连接。为了实现这个机制，服务器应该具有类似以下代码的循环操作：

```
while (true)
{
    Socket incoming = s.accept();
    Runnable r = new ThreadedEchoHandler(incoming);

    Thread t = new Thread(r);
    t.start();
}
```

`ThreadedEchoHandler`类实现了`Runnable`接口，而且在它的`run`方法中包含了与客户端循环通信的代码。

```
class ThreadedEchoHandler implements Runnable
{
    ...
    public void run()
    {
        try
        {
            InputStream inStream = incoming.getInputStream();
            OutputStream outStream = incoming.getOutputStream();
            process input and send response
            incoming.close();
        }
        catch(IOException e)
        {
            handle exception
        }
    }
}
```

由于每一个连接都会启动一个新的线程，因而多个客户端就可以同时连接到服务器了。对此可以做个简单的测试：

1. 编译和运行服务器程序（程序清单3-4）。
2. 如图3-5打开数个telnet窗口。
3. 在这些窗口之间切换，并键入命令。注意你可以同时通过这些窗口进行通信。
4. 当你完成之后，切换到你启动服务器程序的窗口，并使用`CTRL+C`强行关闭它。

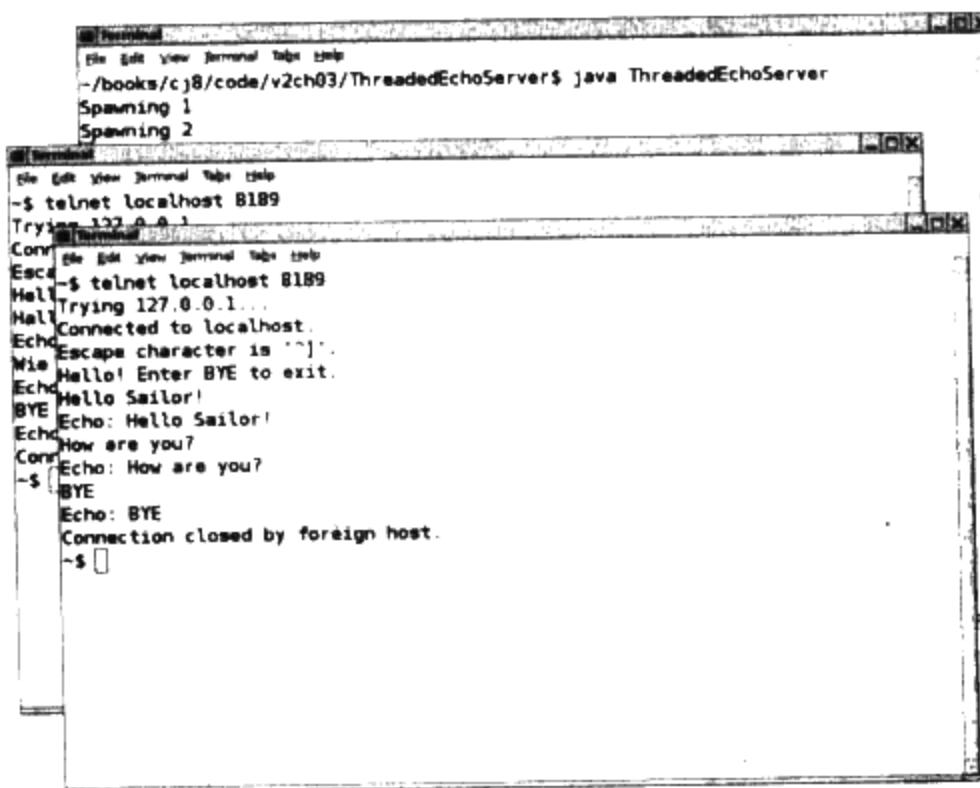


图3-5 多个同时通信的telnet窗口



**注意：**在这个程序中，我们为每个连接生成一个独立的线程。这种方法并不能满足高性能服务器的要求。为使服务器实现更高的吞吐量，你可以使用`java.nio`包中一些特性。

详情请参见以下链接：<http://www-106.ibm.com/developerworks/java/library/j-javaio.html>。

### 程序清单3-4 ThreadedEchoServer.java

```
1. import java.io.*;
2. import java.net.*;
3. import java.util.*;
4.
5. /**
6.  * This program implements a multithreaded server that listens to port 8189 and echoes back
7.  * all client input.
8.  * @author Cay Horstmann
9.  * @version 1.20 2004-08-03
10. */
11 public class ThreadedEchoServer
12 {
13     public static void main(String[] args)
14     {
15         try
16         {
17             int i = 1;
18             ServerSocket s = new ServerSocket(8189);
19.
20             while (true)
21             {
22                 Socket incoming = s.accept();
23                 System.out.println("Spawning " + i);
24                 Runnable r = new ThreadedEchoHandler(incoming);
25                 Thread t = new Thread(r);
26                 t.start();
27                 i++;
28             }
29         }
30         catch (IOException e)
31         {
32             e.printStackTrace();
33         }
34     }
35 }
36.
37 /**
38  * This class handles the client input for one server socket connection.
39. */
40 class ThreadedEchoHandler implements Runnable
41 {
42     /**
43      * Constructs a handler.
44      * @param i the incoming socket
45      * @param c the counter for the handlers (used in prompts)
46. */
47     public ThreadedEchoHandler(Socket i)
48     {
49         incoming = i;
50     }
51.
52     public void run()
```

```

53. {
54.     try
55.     {
56.         try
57.         {
58.             InputStream inStream = incoming.getInputStream();
59.             OutputStream outStream = incoming.getOutputStream();
60.
61.             Scanner in = new Scanner(inStream);
62.             PrintWriter out = new PrintWriter(outStream, true /* autoFlush */);
63.
64.             out.println("Hello! Enter BYE to exit.");
65.
66.             // echo client input
67.             boolean done = false;
68.             while (!done && in.hasNextLine())
69.             {
70.                 String line = in.nextLine();
71.                 out.println("Echo: " + line);
72.                 if (line.trim().equals("BYE"))
73.                     done = true;
74.             }
75.         }
76.         finally
77.         {
78.             incoming.close();
79.         }
80.     }
81.     catch (IOException e)
82.     {
83.         e.printStackTrace();
84.     }
85. }
86.
87. private Socket incoming;
88. }

```

### 3.2.2 半关闭

半关闭 (half-close) 提供了这样一种能力：套接字连接的一端可以终止其输出，同时仍旧可以接收来自另一端的数据。

这是一种很典型的情况，例如我们在向服务器传输数据，但是并不知道要传输多少数据。在写一个文件时，我们只需在数据写入后关闭文件即可。但是，如果关闭一个套接字，那么与服务器的连接将立刻断开，因而也就无法读取响应。

使用半关闭的方法就可以解决上述问题。可以通过关闭一个套接字的输出流来表示发送给服务器的请求数据已经结束，但是必须保持输入流处于打开状态。

如下代码演示了如何在客户端使用半关闭方法：

```

Socket socket = new Socket(host, port);
Scanner in = new Scanner(socket.getInputStream());
PrintWriter writer = new PrintWriter(socket.getOutputStream());
// send request data

```

```
writer.print(...);
writer.flush();
socket.shutdownOutput();
// now socket is half closed
// read response data
while (in.hasNextLine() != null) { String line = in.nextLine(); ... }
socket.close();
```

服务器端将读取输入信息，直到到达输入流的结尾，然后它再发送响应。

当然，该协议只适用于一站式（one-shot）的服务，例如HTTP服务，在这种服务中，客户端连接服务器，发送一个请求，捕获响应信息，然后断开连接。

### java.net.Socket 1.0

- `void shutdownOutput()` 1.3  
将输出流设为“流结束”。
- `void shutdownInput()` 1.3  
将输入流设为“流结束”。
- `boolean isOutputShutdown()` 1.4  
如果输出已被关闭，则返回`true`。
- `boolean isInputShutdown()` 1.4  
如果输入已被关闭，则返回`true`。

### 3.3 可中断套接字

当连接到一个套接字时，当前线程将会被阻塞直到建立连接或产生超时为止。同样地，当通过套接字读写数据时，当前线程也会被阻塞直到操作成功或产生超时为止。

在交互式的应用中，也许会考虑为用户提供一个功能，用以取消那些看似不会成功的连接。但是，当线程因套接字长时间无法响应而发生阻塞时，无法通过调用`interrupt`来解除阻塞。

为了中断套接字操作，可以使用`java.nio`包提供的一个特性——`SocketChannel`类。可以使用如下方法打开`SocketChannel`：

```
SocketChannel channel = SocketChannel.open(new InetSocketAddress(host, port));
```

通道（channel）并没有与之相关联的流。实际上，它所拥有的`read`和`write`方法都是通过调用`Buffer`对象来实现的（关于NIO缓存的相关信息请参见第1章）。`ReadableByteChannel`接口和`WritableByteChannel`接口都声明了这两个方法。

如果不处理缓存，可以使用`Scanner`类来读取信息，因为`Scanner`有一个带`ReadableByteChannel`参数的构造器：

```
Scanner in = new Scanner(channel);
```

通过调用静态方法`Channels.newOutputStream`，可以从通道中获取输出流。

```
OutputStream outStream = Channels.newOutputStream(channel);
```

上述操作都是必须做的。假设线程正在执行打开、读取或写入操作，此时如果线程发生中断，那么这些操作将不会陷入阻塞，而是以抛出异常的方式结束。

程序清单3-5的程序对比了可中断套接字和阻塞套接字：服务器将连续发送数字，并在每发送十个数字之后停滞一下。点击两个按钮中的任何一个，都会启动一个线程来连接服务器并打印输出。第一个线程使用可中断套接字，而第二个线程使用阻塞套接字。如果在第一批的十个数字的读取过程中点击“Cancel”按钮，这两个线程都会中断。

但是，在第一批十个数字之后，就只能中断第一个线程了，第二个线程将保持阻塞直到服务器最终关闭连接（参见图3-6）。

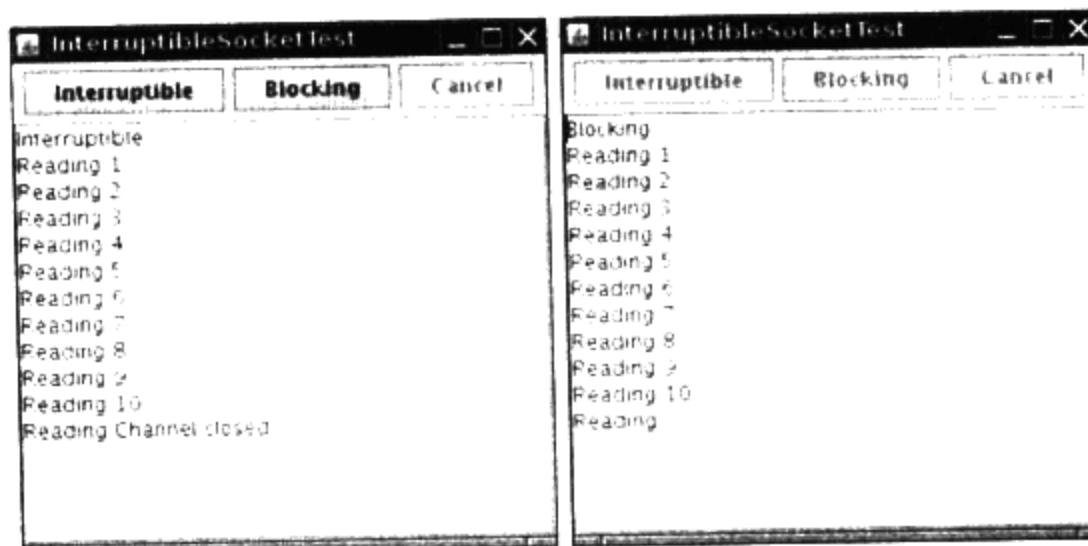


图3-6 中断一个套接字

### 程序清单3-5 InterruptibleSocketTest.java

```

1. import java.awt.*;
2. import java.awt.event.*;
3. import java.util.*;
4. import java.net.*;
5. import java.io.*;
6. import java.nio.channels.*;
7. import javax.swing.*;

8.
9. /**
10. * This program shows how to interrupt a socket channel.
11. * @author Cay Horstmann
12. * @version 1.01 2007-06-25
13. */
14. public class InterruptibleSocketTest
15. {
16.     public static void main(String[] args)
17.     {
18.         EventQueue.invokeLater(new Runnable()
19.         {
20.             public void run()
21.             {
22.                 JFrame frame = new InterruptibleSocketFrame();
23.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24.                 frame.setVisible(true);
25.             }
26.         });
27.     }
28. }
```

```
29.  
30. class InterruptibleSocketFrame extends JFrame  
31. {  
32.     public InterruptibleSocketFrame()  
33.     {  
34.         setSize(WIDTH, HEIGHT);  
35.         setTitle("InterruptibleSocketTest");  
36.  
37.         JPanel northPanel = new JPanel();  
38.         add(northPanel, BorderLayout.NORTH);  
39.  
40.         messages = new JTextArea();  
41.         add(new JScrollPane(messages));  
42.  
43.         interruptibleButton = new JButton("Interruptible");  
44.         blockingButton = new JButton("Blocking");  
45.  
46.         northPanel.add(interruptibleButton);  
47.         northPanel.add(blockingButton);  
48.  
49.         interruptibleButton.addActionListener(new ActionListener()  
50.             {  
51.                 public void actionPerformed(ActionEvent event)  
52.                 {  
53.                     interruptibleButton.setEnabled(false);  
54.                     blockingButton.setEnabled(false);  
55.                     cancelButton.setEnabled(true);  
56.                     connectThread = new Thread(new Runnable()  
57.                         {  
58.                             public void run()  
59.                             {  
60.                                 try  
61.                                 {  
62.                                     connectInterruptibly();  
63.                                 }  
64.                                 catch (IOException e)  
65.                                 {  
66.                                     messages.append("\nInterruptibleSocketTest.connectInterruptibly: "  
67.                                         + e);  
68.                                 }  
69.                             }  
70.                         });  
71.                     connectThread.start();  
72.                 }  
73.             });  
74.         blockingButton.addActionListener(new ActionListener()  
75.             {  
76.                 public void actionPerformed(ActionEvent event)  
77.                 {  
78.                     interruptibleButton.setEnabled(false);  
79.                     blockingButton.setEnabled(false);  
80.                     cancelButton.setEnabled(true);  
81.                     connectThread = new Thread(new Runnable()  
82.                         {  
83.                             public void run()  
84.                             {  
85. 
```

```
86             try
87             {
88                 connectBlocking();
89             }
90             catch (IOException e)
91             {
92                 messages.append("\nInterruptibleSocketTest.connectBlocking: " + e);
93             }
94         });
95     );
96     connectThread.start();
97 }
98 );
99
100 cancelButton = new JButton("Cancel");
101 cancelButton.setEnabled(false);
102 northPanel.add(cancelButton);
103 cancelButton.addActionListener(new ActionListener()
104 {
105     public void actionPerformed(ActionEvent event)
106     {
107         connectThread.interrupt();
108         cancelButton.setEnabled(false);
109     }
110 });
111 server = new TestServer();
112 new Thread(server).start();
113 }
114
115 /**
116 * Connects to the test server, using interruptible I/O
117 */
118 public void connectInterruptibly() throws IOException
119 {
120     messages.append("Interruptible:\n");
121     SocketChannel channel = SocketChannel.open(new InetSocketAddress("localhost", 8189));
122     try
123     {
124         in = new Scanner(channel);
125         while (!Thread.currentThread().isInterrupted())
126         {
127             messages.append("Reading ");
128             if (in.hasNextLine())
129             {
130                 String line = in.nextLine();
131                 messages.append(line);
132                 messages.append("\n");
133             }
134         }
135     }
136     finally
137     {
138         channel.close();
139         EventQueue.invokeLater(new Runnable()
140         {
141             public void run()
```

```
142.         {
143.             messages.append("Channel closed\n");
144.             interruptibleButton.setEnabled(true);
145.             blockingButton.setEnabled(true);
146.         }
147.     });
148. }
149. }
150.
151. /**
152. * Connects to the test server, using blocking I/O
153. */
154. public void connectBlocking() throws IOException
155. {
156.     messages.append("Blocking:\n");
157.     Socket sock = new Socket("localhost", 8189);
158.     try
159.     {
160.         in = new Scanner(sock.getInputStream());
161.         while (!Thread.currentThread().isInterrupted())
162.         {
163.             messages.append("Reading ");
164.             if (in.hasNextLine())
165.             {
166.                 String line = in.nextLine();
167.                 messages.append(line);
168.                 messages.append("\n");
169.             }
170.         }
171.     }
172.     finally
173.     {
174.         sock.close();
175.         EventQueue.invokeLater(new Runnable()
176.         {
177.             public void run()
178.             {
179.                 messages.append("Socket closed\n");
180.                 interruptibleButton.setEnabled(true);
181.                 blockingButton.setEnabled(true);
182.             }
183.         });
184.     }
185. }
186.
187. /**
188. * A multithreaded server that listens to port 8189 and sends numbers to the client,
189. * simulating a hanging server after 10 numbers.
190. */
191. class TestServer implements Runnable
192. {
193.     public void run()
194.     {
195.         try
196.         {
197.             ServerSocket s = new ServerSocket(8189);
```

```
198.         while (true)
199.     {
200.         Socket incoming = s.accept();
201.         Runnable r = new TestServerHandler(incoming);
202.         Thread t = new Thread(r);
203.         t.start();
204.     }
205. }
206. }
207. catch (IOException e)
208. {
209.     messages.append("\nTestServer.run: " + e);
210. }
211. }
212. }
213.
214. /**
215. * This class handles the client input for one server socket connection.
216. */
217. class TestServerHandler implements Runnable
218. {
219.     /**
220.     * Constructs a handler.
221.     * @param i the incoming socket
222.     */
223.     public TestServerHandler(Socket i)
224.     {
225.         incoming = i;
226.     }
227.
228.     public void run()
229.     {
230.         try
231.         {
232.             OutputStream outStream = incoming.getOutputStream();
233.             PrintWriter out = new PrintWriter(outStream, true /* autoFlush */);
234.             while (counter < 100)
235.             {
236.                 counter++;
237.                 if (counter <= 10) out.println(counter);
238.                 Thread.sleep(100);
239.             }
240.             incoming.close();
241.             messages.append("Closing server\n");
242.         }
243.         catch (Exception e)
244.         {
245.             messages.append("\nTestServerHandler.run: " + e);
246.         }
247.     }
248.
249.     private Socket incoming;
250.     private int counter;
251. }
252.
253. private Scanner in;
254. private JButton interruptibleButton;
```

```
255.     private JButton blockingButton;
256.     private JButton cancelButton;
257.     private JTextArea messages;
258.     private TestServer server;
259.     private Thread connectThread;
260.
261.     public static final int WIDTH = 300;
262.     public static final int HEIGHT = 300;
263. }
```

### API **java.net.InetSocketAddress 1.4**

- `InetSocketAddress(String hostname, int port)`

通过主机和端口参数创建一个地址对象，并在创建过程中解析主机名。如果主机名不能被解析，那么该地址对象的`unresolved`属性将被设为true。

- `boolean isUnresolved()`

如果不能解析该地址对象，则返回true。

### API **java.nio.channels.SocketChannel 1.4**

- `static SocketChannel open(SocketAddress address)`

打开一个套接字通道，并将其连接到远程地址。

### API **java.nio.channels.Channels 1.4**

- `static InputStream newInputStream(ReadableByteChannel channel)`

创建一个输入流，用以从指定的通道读取数据。

- `static OutputStream newOutputStream(WritableByteChannel channel)`

创建一个输出流，用以向指定的通道写入数据。

## 3.4 发送E-mail

在这一节中，我们将介绍一个套接字编程的实际例子：一个发送E-mail给远程站点的程序。

为了发送E-mail，必须建立一个到端口25（即SMTP端口）的套接字连接。简单邮件传输协议用于描述E-mail消息的格式。你可以连接到任何一个提供SMTP服务的服务器，不过首先得确定服务器是否愿意接受你的请求。SMTP服务器通常总是愿意发送来自任何人的E-mail，但是在垃圾邮件泛滥的今天，大多数服务器都内置有检查功能，它们只接受那些来自它们所信任的用户或IP地址段的请求。

一旦连接到服务器，就可以发送一个邮件报头（采用SMTP格式，该格式很容易生成）。紧随其后的是邮件消息。

以下是操作的详细过程：

1. 打开一个到达主机的套接字：

```
Socket s = new Socket("mail.yourserver.com", 25); // 25 is SMTP
PrintWriter out = new PrintWriter(s.getOutputStream());
```

## 2. 发送以下信息到打印流：

```

HELO sending host
MAIL FROM: <sender e-mail address>
RCPT TO: <recipient e-mail address>
DATA
mail message
(any number of lines)
QUIT

```

SMTP规范（RFC 821）规定，每一行都要以\r再紧跟一个\n来结尾。

有些SMTP服务器并不检查信息的真实性，你可以随意填写任何你喜欢使用的发件人名字。（请牢记这一点，下次你可能会收到一封来自president@whitehouse.gov的E-mail，邀请你参加在白宫前草坪上举行的正式聚会。找到一个可以转发虚假消息的SMTP服务器相当容易。）

程序清单3-6是一个简单的E-mail程序。如图3-7所示，你可以输入发件人、收件人、邮件消息和SMTP服务器，然后点击发送按钮，你的邮件就这样被发送出去了。

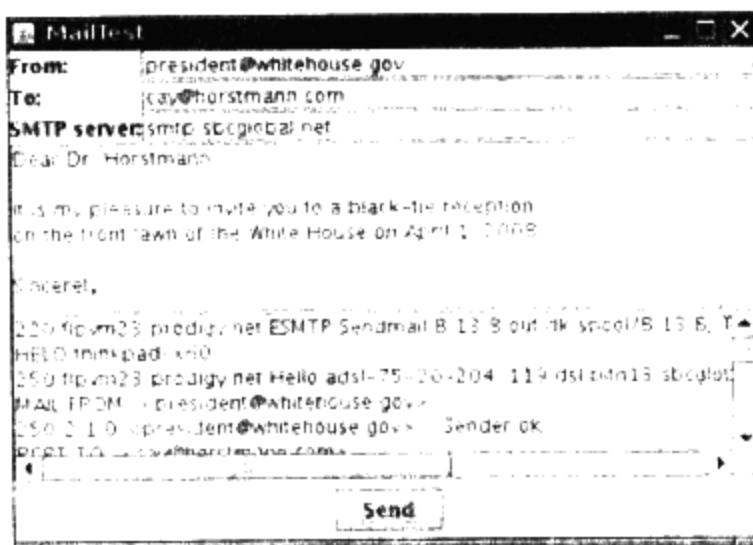


图3-7 MailTest程序

该程序创建了一个到SMTP服务器的套接字连接，然后发送了一个刚才讨论的命令序列，并显示了这些命令和收到的响应。



**注意：**在1996年这个程序出现《Java核心技术》（第1版）中时，大多数SMTP服务器都可以接受来自任何地方的连接，而根本不做任何检查。现在，大多数服务器都变得更加严苛了，你可能会发现运行这个程序变得更困难了。但是只要你是在家里或者某个受信任的IP地址进行连接，那么你的因特网提供商的邮件服务器仍旧有可能是可以访问的。其他服务器将使用“SMTP之前先POP”规则，即要求你在发送任何消息之前先下载你的E-mail（这要求提供密码）。因此，在用这个程序发送邮件之前，应该先尝试获取E-mail。有一种要求提供加密的密码的SMTP扩展（<http://tools.ietf.org/html/rfc2554>）正在变得越来越普及。我们这个简单的程序并不支持这种认证机制。

在本节的最后，你已经了解了如何使用套接字级的编程来连接SMTP服务器以及发送E-mail消息。知道这些操作是可以实现的，并且大致了解一下诸如E-mail之类的因特网服务的

“内幕情况”，是很有好处的。如果你正在设计一个应用并且这个应用涉及E-mail技术，那么为了让自己在更高层次上工作，你或许会希望可以使用一个封装了网络协议实现细节的类库。例如，Sun公司开发了一个Java平台的标准扩展——JavaMail API。在JavaMail API中，你只需简单地调用下面这个方法：

```
Transport.send(message);
```

就可以发送一条邮件消息了。该类库负责实现邮件协议、认证和附件处理等。

### 程序清单3-6 MailTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.util.*;
4. import java.net.*;
5. import java.io.*;
6. import javax.swing.*;

7.
8. /**
9. * This program shows how to use sockets to send plain text mail messages.
10. * @author Cay Horstmann
11. * @version 1.11 2007-06-25
12. */
13. public class MailTest
14. {
15.     public static void main(String[] args)
16.     {
17.         EventQueue.invokeLater(new Runnable()
18.         {
19.             public void run()
20.             {
21.                 JFrame frame = new MailTestFrame();
22.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23.                 frame.setVisible(true);
24.             }
25.         });
26.     }
27. }

28.
29. /**
30. * The frame for the mail GUI.
31. */
32. class MailTestFrame extends JFrame
33. {
34.     public MailTestFrame()
35.     {
36.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
37.         setTitle("MailTest");
38.
39.         setLayout(new GridBagLayout());
40.
41.         // we use the GBC convenience class of Core Java Volume I, Chapter 9
42.         add(new JLabel("From:"), new GBC(0, 0).setFill(GBC.HORIZONTAL));
43.
44.         from = new JTextField(20);
45.         add(from, new GBC(1, 0).setFill(GBC.HORIZONTAL).setWeight(100, 0));
```

```
46.        add(new JLabel("To:"), new GBC(0, 1).setFill(GBC.HORIZONTAL));
47.
48.        to = new JTextField(20);
49.        add(to, new GBC(1, 1).setFill(GBC.HORIZONTAL).setWeight(100, 0));
50.
51.        add(new JLabel("SMTP server:"), new GBC(0, 2).setFill(GBC.HORIZONTAL));
52.
53.        smtpServer = new JTextField(20);
54.        add(smtpServer, new GBC(1, 2).setFill(GBC.HORIZONTAL).setWeight(100, 0));
55.
56.
57.        message = new JTextArea();
58.        add(new JScrollPane(message), new GBC(0, 3, 2, 1).setFill(GBC.BOTH).setWeight(100, 100));
59.
60.
61.        comm = new JTextArea();
62.        add(new JScrollPane(comm), new GBC(0, 4, 2, 1).setFill(GBC.BOTH).setWeight(100, 100));
63.
64.        JPanel buttonPanel = new JPanel();
65.        add(buttonPanel, new GBC(0, 5, 2, 1));
66.
67.        JButton sendButton = new JButton("Send");
68.        buttonPanel.add(sendButton);
69.        sendButton.addActionListener(new ActionListener()
70.        {
71.            public void actionPerformed(ActionEvent event)
72.            {
73.                new SwingWorker<Void, Void>()
74.                {
75.                    protected Void doInBackground() throws Exception
76.                    {
77.                        comm.setText("");
78.                        sendMail();
79.                        return null;
80.                    }
81.                }.execute();
82.            }
83.        });
84.
85.    /**
86.     * Sends the mail message that has been authored in the GUI.
87.     */
88.    public void sendMail()
89.    {
90.        try
91.        {
92.            Socket s = new Socket(smtpServer.getText(), 25);
93.
94.            InputStream inStream = s.getInputStream();
95.            OutputStream outStream = s.getOutputStream();
96.
97.            in = new Scanner(inStream);
98.            out = new PrintWriter(outStream, true /* autoFlush */);
99.
100.           String hostName = InetAddress.getLocalHost().getHostName();
101.
102.           receive();
103.       }
```

```
103         send("HELO " + hostName);
104         receive();
105         send("MAIL FROM: <" + from.getText() + ">");
106         receive();
107         send("RCPT TO: <" + to.getText() + ">");
108         receive();
109         send("DATA");
110         receive();
111         send(message.getText());
112         send(".");
113         receive();
114         s.close();
115     }
116     catch (IOException e)
117     {
118         comm.append("Error: " + e);
119     }
120 }
121
122 /**
123 * Sends a string to the socket and echoes it in the comm text area.
124 * @param s the string to send.
125 */
126 public void send(String s) throws IOException
127 {
128     comm.append(s);
129     comm.append("\n");
130     out.print(s.replaceAll("\n", "\r\n"));
131     out.print("\r\n");
132     out.flush();
133 }
134
135 /**
136 * Receives a string from the socket and displays it in the comm text area.
137 */
138 public void receive() throws IOException
139 {
140     String line = in.nextLine();
141     comm.append(line);
142     comm.append("\n");
143 }
144
145 private Scanner in;
146 private PrintWriter out;
147 private JTextField from;
148 private JTextField to;
149 private JTextField smtpServer;
150 private JTextArea message;
151 private JTextArea comm;
152
153 public static final int DEFAULT_WIDTH = 300;
154 public static final int DEFAULT_HEIGHT = 300;
155 }
```

## 3.5 建立URL连接

为了在Java程序中访问Web服务器，你可能希望在更高的级别上进行处理，而不只是创建套接字连接和发送HTTP请求。在下面的各个小节中，我们将讨论专用于此目的的Java类库中的各个类。

### 3.5.1 URL和URI

URL和URLConnection类封装了大量复杂的实现细节，这些细节涉及如何从远程站点获取信息。例如，可以通过传递字符串来构建一个URL对象：

```
URL url = new URL(urlString);
```

如果只是想获得该资源的内容，可以使用URL类中的openStream方法。该方法返回一个InputStream对象，然后就可以按照一般的用法来使用这个对象了，比如用它构建一个Scanner对象：

```
InputStream inStream = url.openStream();
Scanner in = new Scanner(inStream);
```

java.net包对统一资源定位符（uniform resource locator, URL）和统一资源标识符（uniform resource identifier, URI）作了非常有用的区别。

URI是个纯粹的句法结构，用于指定标识Web资源的字符串的各个不同部分。URL是URI的一个特例，它包含了用于定位Web资源的足够信息。其他URI，比如

`mailto:cay@horstmann.com`

则不属于定位符，因为根据该标识符我们无法定位任何数据。像这样的URI我们称之为URN（uniform resource name，统一资源名称）。

在Java类库中，URI类不包含任何用于访问资源的方法，它的惟一作用就是解析。相反的是，URL类可以打开一个到达资源的流。因此，URL类只能作用于那些Java类库知道该如何处理的模式，例如http:、https:、ftp:、本地文件系统（file:）和JAR文件（jar:）。

为了了解对URI进行解析并不是可有可无的，要考虑到它也许会变得非常复杂。例如，

```
http://maps.yahoo.com/py/maps.py?csz=Cupertino+CA
ftp://username:password@ftp.yourserver.com/pub/file.txt
```

URI规范给出了标记这些标识符的规则。一个URI具有以下句法：

`[scheme:]schemeSpecificPart[#fragment]`

上式中，[...]表示可选部分，它与：和#可以被包含在标识符内。

包含scheme:部分的URI被称为绝对URI。否则，被称为相对URI。

如果绝对URI的schemeSpecificPart不是以/开头的，我们就称它是不透明的。例如：

`mailto:cay@horstmann.com`

所有绝对的透明URI和所有相对URI都是分层的（hierarchical）。例如：

```
http://java.sun.com/index.html
../../java/net/Socket.html#Socket()
```

一个分层URI的schemeSpecificPart具有以下结构：

[//*authority*][*path*][?*query*]

在这里，[...]同样表示可选。

对于那些基于服务器的URI，*authority*部分采用以下形式

[*user-info@*]*host*[*:port*]

*port*必须是一个整数。

RFC 2396（标准化URI的文献）还支持一种基于注册表的机制，此时*authority*采用了一种不同的格式。不过，这种情况并不常见。

URI类的作用之一是解析标识符并将它分解成各种不同的组成部分。你可以用以下方法读取它们：

```
getScheme  
getSchemeSpecificPart  
getAuthority  
getUserInfo  
getHost  
getPort  
getPath  
getQuery  
getFragment
```

URI类的另一个作用是处理绝对标识符和相对标识符。如果存在一个如下的绝对URI：  
<http://docs.mycompany.com/api/java/net/ServerSocket.html>

和一个如下的相对URI：

[.../java/net/Socket.html#Socket\(\)](#)

那么可以将它们合并成一个绝对URI：

[http://docs.mycompany.com/api/java/net/Socket.html#Socket\(\)](http://docs.mycompany.com/api/java/net/Socket.html#Socket())

这个过程被称为相对URL的转换（resolving）。

与此相反的过程称为相对化（relativization）。例如，假设有一个基本URI：

<http://docs.mycompany.com/api>

和另一个URI：

<http://docs.mycompany.com/api/java/lang/String.html>

那么相对化之后的URI就是：

<java/lang/String.html>

URI类同时支持以下两个操作：

```
relative = base.relativize(combined);  
combined = base.resolve(relative);
```

### 3.5.2 使用URLConnection获取信息

如果想从某个Web资源获取更多信息，那么应该使用URLConnection类，它能得到比基本的URL类更多的控制功能。

当操作一个URLConnection对象时，必须像下面这样非常小心地安排操作步骤：

1. 调用URL类中的openConnection方法获得URLConnection对象；

```
URLConnection connection = url.openConnection();
```

2. 使用以下方法来设置任意的请求属性：

```
setDoInput  
setDoOutput  
setIfModifiedSince  
setUseCaches  
setAllowUserInteraction  
setRequestProperty  
setConnectTimeout  
setReadTimeout
```

我们将在本节的稍后部分以及API说明中讨论这些方法。

3. 调用connect方法连接远程资源：

```
connection.connect();
```

除了与服务器建立套接字连接外，该方法还可用于向服务器查询头信息（header information）。

4. 与服务器建立连接后，你可以查询头信息。getHeaderFieldKey和getHeaderField两个方法列举了消息头的所有字段。getHeaderFields方法返回一个包含了消息头中所有字段的标准Map对象。为了方便使用，以下方法可以查询各标准字段：

```
getContentType  
getContentLength  
getContentEncoding  
getDate  
getExpiration  
getLastModified
```

5. 最后，访问资源数据。使用getInputStream方法获取一个输入流用以读取信息（这个输入流与URL类中的openStream方法所返回的流相同）。另一个方法getContent在实际操作中并不是很有用。由标准内容类型（比如text/plain和image/gif）所返回的对象需要使用com.sun层次结构中的类来进行处理。也可以注册自己的内容处理器，但是在本书中我们不讨论这项技术。



**警告：**一些程序员在使用URLConnection类的过程中形成了错误的观念，他们认为URLConnection类中的getInputStream和getOutputStream方法与Socket类中的这些方法相似，但是这种想法并不十分正确。URLConnection类具有很多表面之外的神奇功能，尤其在处理请求和响应消息头时。正因为如此，严格遵循建立连接的每个步骤都显得非常重要。

下面将详细介绍一下URLConnection类中的一些方法。有几个方法可以在与服务器建立连接之前设置连接属性，其中最重要的是setDoInput和setDoOutput。在默认情况下建立的连接只有从服务器读取信息的输入流，并没有任何执行写操作的输出流。如果想获得输出流（例如，向一个Web服务器提交数据），那么你需要调用：

```
connection.setDoOutput(true);
```

接下来，也许想设置某些请求头（request header）。请求头是与请求命令一起被发送到服

务器的。例如：

```
GET www.server.com/index.html HTTP/1.0
Referer: http://www.somewhere.com/links.html
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.4)
Host: www.server.com
Accept: text/html, image/gif, image/jpeg, image/png, */
Accept-Language: en
Accept-Charset: iso-8859-1,*;utf-8
Cookie: orangemilano=192218887821987
```

`setIfModifiedSince`方法用于告诉连接你只对自某个特定日期以来被修改过的数据感兴趣；`setUseCaches`和`setAllowUserInteraction`这两个方法只作用于Applet；`setUseCaches`方法用于命令浏览器首先检查它的缓存；`setAllowUserInteraction`方法则用于在访问有密码保护的资源时弹出对话框，以便查询用户名和口令（见图3-8）。

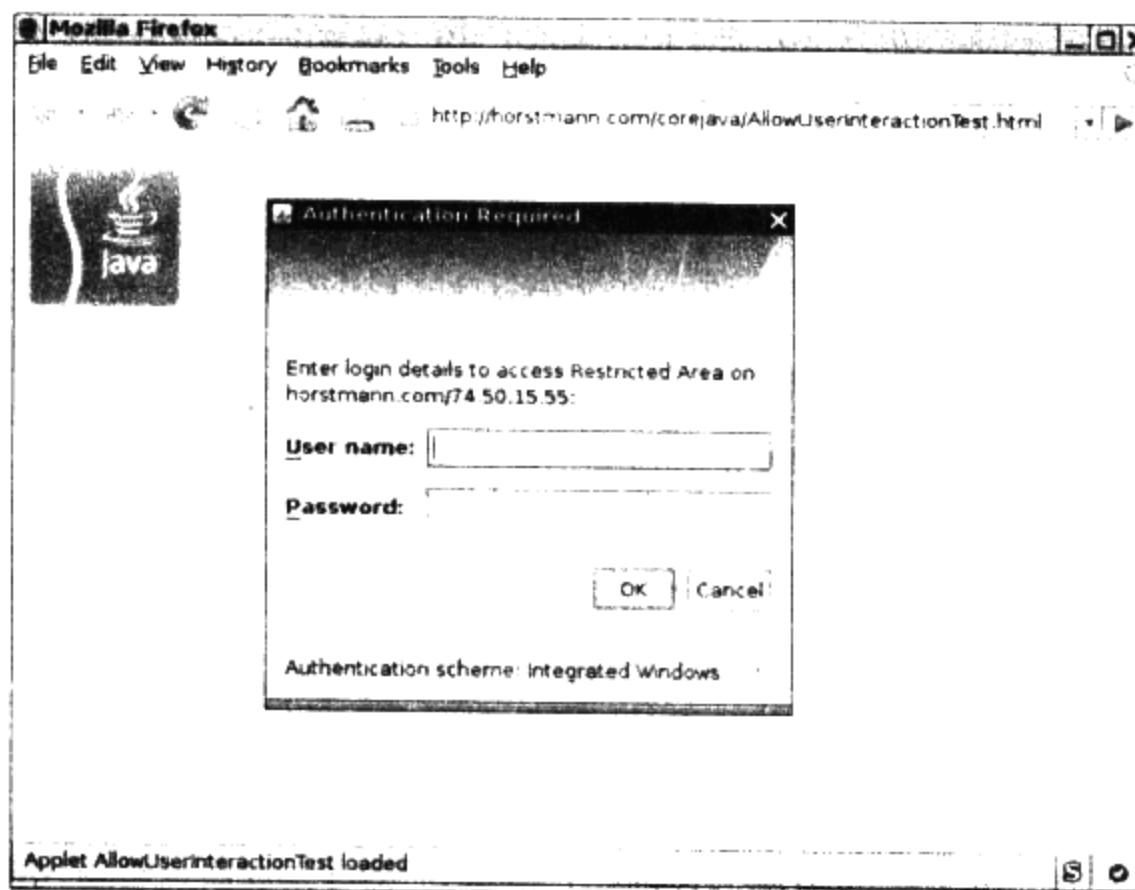


图3-8 网络口令对话框

最后我们再介绍一个总揽全局的方法：`setRequestProperty`，它可以用来设置对特定协议起作用的任何“名-值（name/value）对”。关于HTTP请求头的格式，请参见RFC 2616，其中的某些参数没有很好地记录在文档中，它们通常在程序员之间口头传授。例如，如果你想访问一个有口令保护的Web页，那么就必须按如下步骤操作：

1. 将用户名、冒号和口令以字符串形式连接在一起。

```
String input = username + ":" + password;
```

2. 计算上一步骤所得字符串的base64编码。（base64编码用于将字节流编码成可打印的ASCII字符流。）

```
String encoding = base64Encode(input);
```

3. 调用setRequestProperty方法，设置name参数的值为“Authorization”、value参数的值为“Basic” + encoding：

```
connection.setRequestProperty("Authorization", "Basic " + encoding);
```

**!** 提示：我们上面介绍的是如何访问一个有口令保护的Web页。如果想要通过FTP访问一个有口令保护的文件时，要采用一种完全不同的方法。可以直接构建一个如下格式的URL：

```
ftp://username:password@ftp.yourserver.com/pub/file.txt
```

一旦调用了connect方法，就可以查询响应头信息。首先，我们将介绍如何列举所有响应头的字段。似乎是为了展示自己的个性，该操作采用了另一种迭代方式。调用如下方法：

```
String key = connection.getHeaderFieldKey(n);
```

可以获得响应头的第n个键，其中n从1开始。如果n为0或大于消息头的字段总数，该方法将返回null值。没有哪种方法可以返回字段的数量，你必须反复调用getHeaderFieldKey方法直到返回null为止。同样地，调用以下方法：

```
String value = connection.getHeaderField(n);
```

可以得到第n个值。

getHeaderFields方法可以返回一个封装了响应头字段的Map对象，关于如何访问Map，我们已经在本书的第2章中作了介绍。

```
Map<String, List<String>> headerFields = connection.getHeaderFields();
```

下面是一组来自典型的HTTP请求的响应头字段。

```
Date: Wed, 27 Aug 2008 00:15:48 GMT
Server: Apache/2.2.2 (Unix)
Last-Modified: Sun, 22 Jun 2008 20:53:38 GMT
Accept-Ranges: bytes
Content-Length: 4813
Connection: close
Content-Type: text/html
```

为了简便起见，Java提供了6个方法用以访问大多数常用的消息头类型的值，并在需要的时候将它们转换成数字类型，这些方法的详细信息请参加表3-1。返回类型为long的方法返回的是从格林尼治时间1970年1月1日开始计算的秒数。

表3-1 用于访问响应头值的简便方法

| 键 名              | 方 法 名              | 返 回 类 型 |
|------------------|--------------------|---------|
| Date             | getDate            | long    |
| Expires          | getExpiration      | long    |
| Last-Modified    | getLastModified    | long    |
| Content-Length   | getContentLength   | int     |
| Content-Type     | getContentType     | String  |
| Content-Encoding | getContentEncoding | String  |

通过程序清单3-7的程序，可以对URL连接做一些试验。程序运行起来后，请在命令行中

输入一个URL以及用户名和口令（可选），例如：

```
java URLConnectionTest http://www.yourserver.com user password
```

该程序将输出以下内容：

- 消息头中的所有键和值。
- 表3-1中6个简便方法的返回值。
- 被请求资源的前10行信息。

除了base64编码的计算稍显复杂之外，程序的其他部分非常简单明了。有一个类sun.misc.BASE64Encoder，虽然没有被归档，但是可以用它来代替我们在示例程序中提供的类。只需用以下代码替代对Base64Encode的调用即可：

```
String encoding = new sun.misc.BASE64Encoder().encode(input.getBytes());
```

这里之所以提供自己的类，是因为我们不想依赖于未公开（undocumented）的类。

 **注意：**JavaMail标准扩展包中的javax.mail.internet.MimeUtility类也有一个用于base64编码的方法。出于同样的目的，JDK中还有一个java.util.prefs.Base64类，但是这个类不是公共类，你并不能在自己的代码中使用它。

### 程序清单3-7 URLConnectionTest.java

```
1. import java.io.*;
2. import java.net.*;
3. import java.util.*;
4.
5. /**
6. * This program connects to a URL and displays the response header data and the first 10
7. * lines of the requested data.
8. *
9. * Supply the URL and an optional username and password (for HTTP basic authentication) on
10. * the command line.
11. * @version 1.11 2007-06-26
12. * @author Cay Horstmann
13. */
14. public class URLConnectionTest
15. {
16.     public static void main(String[] args)
17.     {
18.         try
19.         {
20.             String urlName;
21.             if (args.length > 0) urlName = args[0];
22.             else urlName = "http://java.sun.com";
23.
24.             URL url = new URL(urlName);
25.             URLConnection connection = url.openConnection();
26.
27.             // set username, password if specified on command line
28.
29.             if (args.length > 2)
30.             {
31.                 String username = args[1];
32.                 String password = args[2];
```

```
33.         String input = username + ":" + password;
34.         String encoding = base64Encode(input);
35.         connection.setRequestProperty("Authorization", "Basic " + encoding);
36.     }
37.
38.     connection.connect();
39.
40.     // print header fields
41.
42.     Map<String, List<String>> headers = connection.getHeaderFields();
43.     for (Map.Entry<String, List<String>> entry : headers.entrySet())
44.     {
45.         String key = entry.getKey();
46.         for (String value : entry.getValue())
47.             System.out.println(key + ": " + value);
48.     }
49.
50.     // print convenience functions
51.
52.     System.out.println("-----");
53.     System.out.println("getContentType: " + connection.getContentType());
54.     System.out.println("getContentLength: " + connection.getContentLength());
55.     System.out.println("getContentEncoding: " + connection.getContentEncoding());
56.     System.out.println("getDate: " + connection.getDate());
57.     System.out.println("getExpiration: " + connection.getExpiration());
58.     System.out.println("getLastModified: " + connection.getLastModified());
59.     System.out.println("-----");
60.
61.     Scanner in = new Scanner(connection.getInputStream());
62.
63.     // print first ten lines of contents
64.
65.     for (int n = 1; in.hasNextLine() && n <= 10; n++)
66.         System.out.println(in.nextLine());
67.     if (in.hasNextLine()) System.out.println("... ");
68. }
69. catch (IOException e)
70. {
71.     e.printStackTrace();
72. }
73. }
74. /**
75. * Computes the Base64 encoding of a string
76. * @param s a string
77. * @return the Base 64 encoding of s
78. */
79. public static String base64Encode(String s)
80. {
81.     ByteArrayOutputStream bOut = new ByteArrayOutputStream();
82.     Base64OutputStream out = new Base64OutputStream(bOut);
83.     try
84.     {
85.         out.write(s.getBytes());
86.         out.flush();
87.     }
88.     catch (IOException e)
89.     {
```

```
90     }
91     return bOut.toString();
92   }
93 }
94
95 /**
96 * This stream filter converts a stream of bytes to their Base64 encoding.
97 *
98 * Base64 encoding encodes 3 bytes into 4 characters. |11111122|22223333|33444444| Each set
99 * of 6 bits is encoded according to the toBase64 map. If the number of input bytes is not a
100 * multiple of 3, then the last group of 4 characters is padded with one or two = signs. Each
101 * output line is at most 76 characters.
102 */
103 class Base64OutputStream extends FilterOutputStream
104 {
105   /**
106    * Constructs the stream filter
107    * @param out the stream to filter
108    */
109   public Base64OutputStream(OutputStream out)
110   {
111     super(out);
112   }
113
114   public void write(int c) throws IOException
115   {
116     inbuf[i] = c;
117     i++;
118     if (i == 3)
119     {
120       super.write(toBase64[(inbuf[0] & 0xFC) >> 2]);
121       super.write(toBase64[((inbuf[0] & 0x03) << 4) | ((inbuf[1] & 0xF0) >> 4)]);
122       super.write(toBase64[((inbuf[1] & 0x0F) << 2) | ((inbuf[2] & 0xC0) >> 6)]);
123       super.write(toBase64[inbuf[2] & 0x3F]);
124       col += 4;
125       i = 0;
126       if (col >= 76)
127       {
128         super.write('\n');
129         col = 0;
130       }
131     }
132   }
133
134   public void flush() throws IOException
135   {
136     if (i == 1)
137     {
138       super.write(toBase64[(inbuf[0] & 0xFC) >> 2]);
139       super.write(toBase64[(inbuf[0] & 0x03) << 4]);
140       super.write('=');
141       super.write('=');
142     }
143     else if (i == 2)
144     {
145       super.write(toBase64[(inbuf[0] & 0xFC) >> 2]);
```

```

146.         super.write(toBase64[((inbuf[0] & 0x03) << 4) | ((inbuf[1] & 0xF0) >> 4)]);
147.         super.write(toBase64[(inbuf[1] & 0x0F) << 2]);
148.         super.write('=');
149.     }
150.     if (col > 0)
151.     {
152.         super.write('\n');
153.         col = 0;
154.     }
155. }
156.
157. private static char[] toBase64 = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K',
158.     'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b',
159.     'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's',
160.     't', 'u', 'v', 'w', 'x', 'y', 'z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
161.     '+', '/' };
162.
163. private int col = 0;
164. private int i = 0;
165. private int[] inbuf = new int[3];
166. }

```

**注意：**一个常会遇到的问题是Java平台是否支持对安全Web网页的访问 (<https://URL>)。从Java SE 1.4开始，对安全套接字层SSL的支持已经成为标准程序库的一部分。在Java SE 1.4之前，你只能利用浏览器中实现的SSL通过Applet建立SSL连接。

### **java.net.URL 1.0**

- **InputStream openStream()**

打开一个用于读取资源数据的输入流。

- **URLConnection openConnection()**

返回一个URLConnection对象，该对象负责管理与资源之间的连接。

### **java.netURLConnection 1.0**

- **void setDoInput(boolean doInput)**

- **boolean getDoInput()**

如果doInput为true，那么用户可以接收来自该URLConnection的输入。

- **void setDoOutput(boolean doOutput)**

- **boolean getDoOutput()**

如果doOutput为true，那么用户可以将输出发送到该URLConnection。

- **void setIfModifiedSince(long time)**

- **long getIfModifiedSince()**

属性ifModifiedSince用于配置URLConnection对象，使它只获取那些自从某个给定时间以来被修改过的数据。调用方法时需要传入的time参数指的是从格林尼治时间1970年1月1日午夜开始计算的秒数。

- **void setUseCaches(boolean useCaches)**

- `boolean getUseCaches()`

如果useCaches为true，那么数据可以从本地缓存中得到。请注意，URLConnection本身并不维护这样一个缓存，缓存必须由浏览器之类的外部程序提供。

- `void setAllowUserInteraction(boolean allowUserInteraction)`

- `boolean getAllowsUserInteraction()`

如果allowUserInteraction为true，那么可以查询用户的口令。请注意，URLConnection本身并不提供这种查询功能。查询必须由浏览器或浏览器插件之类的外部程序实现。

- `void setConnectTimeout(int timeout) 5.0`

- `int getConnectTimeout() 5.0`

设置或得到连接超时时限（单位：毫秒）。如果在连接建立之前就已经达到了超时的时限，那么输入流的connect方法就会抛出一个SocketTimeoutException异常。

- `void setReadTimeout(int timeout) 5.0`

- `int getReadTimeout() 5.0`

设置读取数据的超时时限（单位：毫秒）。如果在一个读操作成功之前就已经达到了超时的时限，那么read方法就会抛出一个SocketTimeoutException异常。

- `void setRequestProperty(String key, String value)`

设置请求头的一个字段。

- `Map<String, List<String>> getRequestProperties() 1.4`

返回请求头属性的一个映射表。相同的键对应的所有值被放置在同一个映射表中。

- `void connect()`

连接远程资源并获取响应头信息。

- `Map<String, List<String>> Map getHeaderFields() 1.4`

返回响应的一个映射表。相同的键对应的所有值被放置在同一个映射表中。

- `String getHeaderFieldKey(int n)`

得到响应头第n个字段的键。如果n等于0或大于响应头字段的总数，该方法返回null值。

- `String getHeaderField(int n)`

得到响应头第n个字段的值。如果n等于0或大于响应头字段的总数，该方法返回null值。

- `int getContentLength()`

如果知道内容长度，则返回该长度值，否则返回-1。

- `String getContentType`

获取内容的类型，比如text/plain或image/gif。

- `String getContentEncoding()`

获取内容的编码，比如gzip。这个值不太常用，因为默认的identity编码并不是用Content-Encoding头来设定的。

- `long getDate()`

- `long getExpiration()`

- `long getLastModified()`

获取创建日期、过期日以及最后一次被修改的日期。这些日期指的是从格林尼治时间1970年1月1日午夜开始计算的秒数。

- `InputStream getInputStream()`

- `OutputStream getOutputStream()`

返回从资源读取信息或向资源写入信息的流。

- `Object getContent()`

选择适当的内容处理器，以便读取资源数据并将它转换成对象。该方法不能用于读取诸如`text/plain`或`image/gif`之类的标准内容类型，除非你安装了自己的内容处理器。

### 3.5.3 提交表单数据

在上一节中，我们介绍了如何从一个Web服务器读取数据。现在，我们将介绍如何让程序再将数据反馈回Web服务器和那些被Web服务器调用的程序。

为了将信息从Web浏览器发送到Web服务器，用户需要填写一个类似图3-9中所示的表单。

The screenshot shows a Mozilla Firefox browser window with the title "World Population Prospects: The 2006 Revision Population Database - Mozilla Firefox". The address bar contains the URL <http://esa.un.org/unpp/>. The main content area displays a form for selecting variables and countries. On the left, there are several links: "Panel 1 Basic data", "Panel 2 Detailed data", "Country profile", "Assumptions", "Definition of regions", "Sources", and "Glossary". In the center, there are two main sections: "Select Variables (up to 5):" and "Select Country/Region (up to 5):". The "Select Variables" section has a dropdown menu with options: Population, Population density, Percentage urban, and Percentage rural. The "Select Country/Region" section has a dropdown menu with options: Least developed countries, Less developed regions, excluding least developed countries, Less developed regions, excluding China, Sub-Saharan Africa, Africa, Eastern Africa, Burundi, Comoros, Djibouti, Eritrea, Ethiopia, Kenya, Madagascar, Malawi, and Mauritius. At the bottom of the form, there are buttons for "Select Variant:" (Medium variant), "Select Start Year:" (1950), "Select End Year:" (2050), "Display", and "Download as .CSV File". The footer of the page includes copyright information: "Copyright © United Nations, 2007", navigation links: "Basic Data | Detailed Data | Country profile | Assumptions | Definition of regions | Sources | Glossary", and a note: "This website is last updated on 20-Sept-2007".

图3-9 HTML表单

当用户点击提交按钮时，文本框中的文本以及复选框和单选按钮的设定值都被发送到了Web服务器。此时，Web服务器调用程序对用户的输入进行处理。

有许多技术可以让Web服务器实现对程序的调用。其中最广人所知的是Java Servlet、JavaServer Face、微软ASP（Active Server Pages，动态服务器主页）以及CGI（Common Gateway Interface，通用网关接口）脚本。为简便起见，我们在说明问题时不考虑所用的具体技术，而一律使用通用术语：服务器端程序脚本。

服务器端脚本用于处理表单数据并生成另一个HTML页，该页会被Web服务器发回给浏览器，这个操作过程我们在图3-10中作了说明。返回给浏览器的响应页可能包含新的信息（例如，信息检索程序中的响应页）或者仅仅只是一个确认。

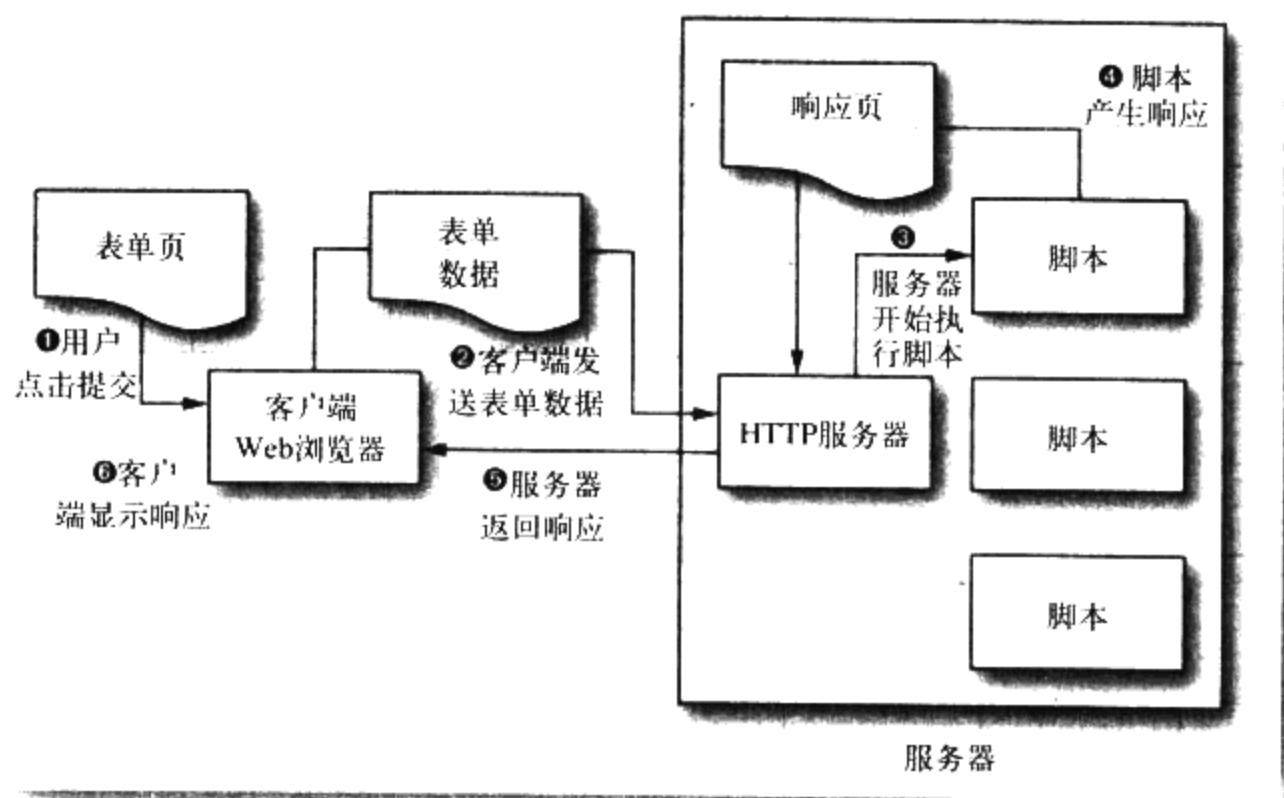


图3-10 执行服务器端脚本过程中的数据流

我们不会在本书中介绍应该如何实现服务器端脚本，而将侧重点放在如何编写客户端程序使之与已有的服务器端脚本进行交互。

当表单数据被发送到Web服务器时，数据到底由谁来处理并不重要，可能是Servlet或CGI脚本，也可能是其他服务器端技术。客户端将数据发送给Web服务器，而Web服务器则负责将数据传递给具体的响应程序。

在向Web服务器发送信息时，通常有两个命令会被用到：GET和POST。

在使用GET命令时，只需将参数附在URL的结尾处即可。该URL的格式如下：

`http://host/script?parameters`

其中，每个参数都具有“名字=值”的形式，而这些参数之间用&字符分隔。参数的值将遵循下面的规则，使用URL编码模式进行编码：

- 保留字符A-Z、a-z、0-9以及 - \* \_。

- 用+字符替换所有的空格。
- 将其他所有字符编码为UTF-8，并将每个字节都编码为%后面紧跟一个两位的十六进制数字。例如，若要发送街道名S. Main，可以使用S%2e+Main，因为十六进制数2e（即十进制数46）是“.”的ASCII码值。

这种编码方式使得在任何中间程序中都不会混入空格，并且也不需要对其他特殊字符进行转换。

例如，就在编写本书的时候，Yahoo的Web站点在主机maps.yahoo.com上放置了一个脚本程序：py/maps.py。运行该脚本需要两个参数：addr和csz。

为了得到1 Infinite Loop, Cupertino, CA的地图，只需访问下面的URL即可：

```
http://maps.yahoo.com/py/maps.py?addr=1+Infinite+Loop&csz=Cupertino+CA
```

GET命令很简单，但是它有一个重要的局限性，正是由于这个局限性使得它并不是非常受欢迎，大多数浏览器都对GET请求中可以包含的字符数作了限制。

在使用POST命令时，并不需要在URL中添加任何参数，而是从URLConnection中获取输出流，并将名-值对写入该流中。当然，仍然需要对这些值进行URL编码，并用&字符将它们隔开。

下面，我们将详细介绍这个过程。在提交数据给脚本之前，首先需要创建一个URLConnection对象。

```
URL url = new URL("http://host/script");
URLConnection connection = url.openConnection();
```

然后，调用setDoOutput方法建立一个用于输出的连接。

```
connection.setDoOutput(true);
```

接着，调用getOutputStream方法获得一个流，可以通过这个流向服务器发送数据。如果要向服务器发送文本信息，那么可以非常方便地将流包装在PrintWriter对象中。

```
PrintWriter out = new PrintWriter(connection.getOutputStream());
```

现在，可以向服务器发送数据了。

```
out.print(name1 + "=" + URLEncoder.encode(value1, "UTF-8") + "&");
out.print(name2 + "=" + URLEncoder.encode(value2, "UTF-8"));
```

之后，关闭输出流。

```
out.close();
```

最后，调用getInputStream方法读取服务器的响应。

下面我们来实际操作一个例子。地址为http://esa.un.org/unpp/的Web站点包含一个请求访问人口数据的表单（见图3-9）。如果查看过该页的HTML源码，不难找到下面这个HTML标签：

```
<form action="p2k0data.asp" method="post">
```

该标签表明：用户点击提交按钮后，名为p2k0data.asp的脚本将会被执行，并且必须是使用POST命令将数据发送给脚本。

接着，需要找出脚本程序将要处理的字段名。观察用户界面中的构件，会发现每个构件都有一个name属性，例如，

```
<select name="Variable">
<option value="12;">Population</option>
```

```
more options ...
</select>
```

从上述几行代码可以看出，该字段名为Variable，这个字段用于设置人口数量表的类型。如果设表的类型为12，那么将得到一个年中的人口总数表。接着看下去，可以找到一个表示地域的字段名：location。它的值如果是900，则表示整个世界；如果是404，则表示肯尼亚。

另外还有其他一些字段需要设置，例如，要得到肯尼亚自1950年到2050年间的人口估计值，可以构建下面的字符串：

```
Panel=1&Variable=12%3b&Location=404&Variant=2&StartYear=1950&EndYear=2050&
DoWhat=Download+as+%2eCSV+File
```

将这个字符串发送给下面这个URL

<http://esa.un.org/unpp/p2k0data.asp>

脚本程序返回响应信息如下：

```
"Country","Variable","Variant","Year","Value"
"Kenya","Population (thousands)","Medium variant","1950",6077
"Kenya","Population (thousands)","Medium variant","1955",6984
"Kenya","Population (thousands)","Medium variant","1960",8115
"Kenya","Population (thousands)","Medium variant","1965",9524
"Kenya","Population (thousands)","Medium variant","1970",11273
...
"Kenya","Population (thousands)","Medium variant","1975",13512
"Kenya","Population (thousands)","Medium variant","1980",16202
"Kenya","Population (thousands)","Medium variant","1985",18774
"Kenya","Population (thousands)","Medium variant",1990,21447
"Kenya","Population (thousands)","Medium variant",1995,23950
"Kenya","Population (thousands)","Medium variant",2000,26152
"Kenya","Population (thousands)","Medium variant",2005,28534
"Kenya","Population (thousands)","Medium variant",2010,314574
"Kenya","Population (thousands)","Medium variant",2015,346167
"Kenya","Population (thousands)","Medium variant",2020,381691
"Kenya","Population (thousands)","Medium variant",2025,417176
"Kenya","Population (thousands)","Medium variant",2030,452762
"Kenya","Population (thousands)","Medium variant",2035,48464
```

就像你所看到的，返回响应信息的脚本程序发送回一个由逗号分隔的数据文件，所以我们才选它作为我们的例子，因为这样很容易了解这个脚本是如何运行的。但是，如果要解释其他脚本程序所产生的复杂的HTML标签，那将是非常麻烦的。

程序清单3-8用于将POST数据发送给任何脚本。我们提供了一个简单的图形用户界面用以设置表单数据和查看输出（见图3-11）。

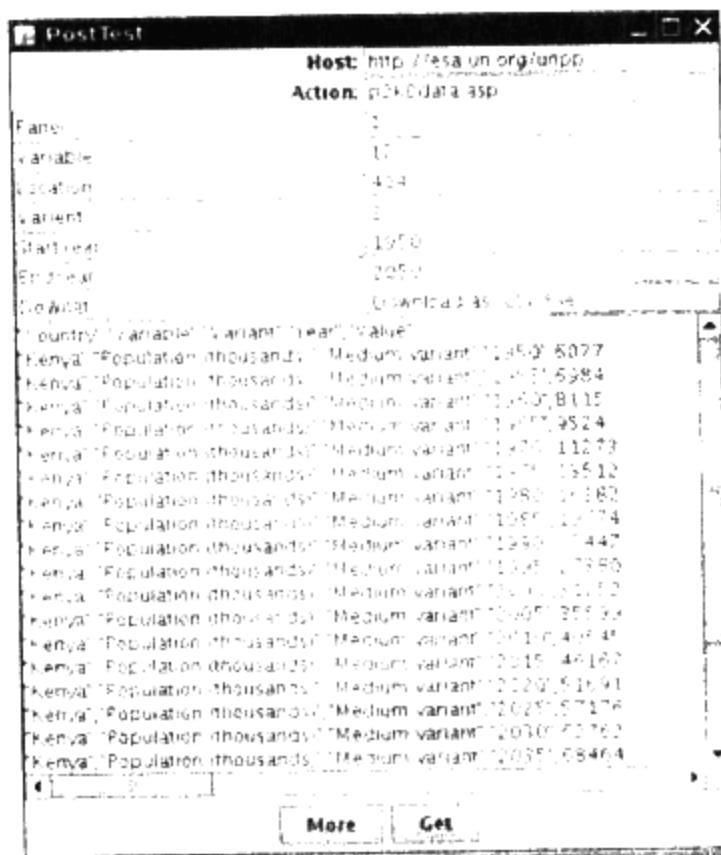


图3-11 从服务器获取信息

在doPost方法中，我们首先打开连接、调用setDoOutput(true)以及打开输出流。然后，枚举Map对象中的名字和值。对每一个名-值对，我们发送name、=字符、value和&分隔符：

```
out.print(name);
out.print('=');
out.print(URLEncoder.encode(value, "UTF-8"));
if (more pairs) out.print('&');
```

最后，我们从服务器读取响应信息。

在读取响应过程中会碰到一个问题。如果脚本运行出现错误，那么调用connection.getInputStream()时就会抛出一个FileNotFoundException异常。但是，此时服务器仍然会向浏览器返回一个错误页面（例如，常见的“错误404—找不到该页”）。为了捕捉这个错误页，可以将URLConnection对象转型为HttpURLConnection类并调用它的getErrorStream方法：

```
InputStream err = ((HttpURLConnection) connection).getErrorStream();
```

更多的是出于好奇而非实际需要，人们常会问：除了我们提供的数据之外，URLConnection到底还向服务器发送了什么信息。

URLConnection对象首先向服务器发送一个请求头。当提交表单数据时，该请求头必须包含下面这行内容：

```
Content-Type: application/x-www-form-urlencoded
```

而POST的请求头还必须包括内容的长度，例如：

```
Content-Length: 124
```

请求头必须以空自行结尾，紧跟其后的才是数据部分。Web服务器滤去请求头后将数据部分发送给服务器端脚本程序。

请注意，URLConnection对象会把你发送到输出流的所有数据都缓存起来，这是因为在真正发送之前它必须首先确定内容的总长度。

每当需要从某个现有的Web站点查询信息时，就可以使用该程序所展示的处理技术。只需找出所要发送的参数（通常可以通过查看执行同一查询操作的Web页的HTML源代码来发现），然后从响应信息中剔除HTML标签和其他不必要的信息。

### 程序清单3-8 PostTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.io.*;
4. import java.net.*;
5. import java.util.*;
6. import javax.swing.*;
7.
8. /**
9. * This program demonstrates how to use the URLConnection class for a POST request.
10 * @version 1.20 2007-06-25
11 * @author Cay Horstmann
12 */
13. public class PostTest
14 {
```

```
15. public static void main(String[] args)
16. {
17.     EventQueue.invokeLater(new Runnable()
18.     {
19.         public void run()
20.         {
21.             JFrame frame = new PostTestFrame();
22.             frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23.             frame.setVisible(true);
24.         }
25.     });
26. }
27 }
28
29 class PostTestFrame extends JFrame
30 {
31. /**
32. * Makes a POST request and returns the server response.
33. * @param urlString the URL to post to
34. * @param nameValuePairs a map of name/value pairs to supply in the request.
35. * @return the server reply (either from the input stream or the error stream)
36. */
37 public static String doPost(String urlString, Map<String, String> nameValuePairs)
38     throws IOException
39 {
40     URL url = new URL(urlString);
41     URLConnection connection = url.openConnection();
42     connection.setDoOutput(true);
43
44     PrintWriter out = new PrintWriter(connection.getOutputStream());
45     boolean first = true;
46     for (Map.Entry<String, String> pair : nameValuePairs.entrySet())
47     {
48         if (first) first = false;
49         else out.print('&');
50         String name = pair.getKey();
51         String value = pair.getValue();
52         out.print(name);
53         out.print('=');
54         out.print(URLEncoder.encode(value, "UTF-8"));
55     }
56
57     out.close();
58     Scanner in;
59     StringBuilder response = new StringBuilder();
60     try
61     {
62         in = new Scanner(connection.getInputStream());
63     }
64     catch (IOException e)
65     {
66         if (!(connection instanceof HttpURLConnection)) throw e;
67         InputStream err = ((HttpURLConnection) connection).getErrorStream();
68         if (err == null) throw e;
69         in = new Scanner(err);
70     }
71 }
```

```
72     while (in.hasNextLine())
73     {
74         response.append(in.nextLine());
75         response.append("\n");
76     }
77
78     in.close();
79     return response.toString();
80 }
81
82 public PostTestFrame()
83 {
84     setTitle("PostTest");
85
86     northPanel = new JPanel();
87     add(northPanel, BorderLayout.NORTH);
88     northPanel.setLayout(new GridLayout(0, 2));
89     northPanel.add(new JLabel("Host: ", SwingConstants.TRAILING));
90     final JTextField hostField = new JTextField();
91     northPanel.add(hostField);
92     northPanel.add(new JLabel("Action: ", SwingConstants.TRAILING));
93     final JTextField actionField = new JTextField();
94     northPanel.add(actionField);
95     for (int i = 1; i <= 8; i++)
96         northPanel.add(new JTextField());
97
98     final JTextArea result = new JTextArea(20, 40);
99     add(new JScrollPane(result));
100
101    JPanel southPanel = new JPanel();
102    add(southPanel, BorderLayout.SOUTH);
103    JButton addButton = new JButton("More");
104    southPanel.add(addButton);
105    addButton.addActionListener(new ActionListener()
106    {
107        public void actionPerformed(ActionEvent event)
108        {
109            northPanel.add(new JTextField());
110            northPanel.add(new JTextField());
111            pack();
112        }
113    });
114
115    JButton getButton = new JButton("Get");
116    southPanel.add(getButton);
117    getButton.addActionListener(new ActionListener()
118    {
119        public void actionPerformed(ActionEvent event)
120        {
121            result.setText("");
122            final Map<String, String> post = new HashMap<String, String>();
123            for (int i = 4; i < northPanel.getComponentCount(); i += 2)
124            {
125                String name = ((JTextField) northPanel.getComponent(i)).getText();
126                if (name.length() > 0)
127                {
```

```
128.             String value = ((JTextField) northPanel.getComponent(i + 1)).getText();
129.             post.put(name, value);
130.         }
131.     }
132.     new SwingWorker<Void, Void>()
133.     {
134.         protected Void doInBackground() throws Exception
135.         {
136.             try
137.             {
138.                 String urlString = hostField.getText() + "/" + actionField.getText();
139.                 result.setText(doPost(urlString, post));
140.             }
141.             catch (IOException e)
142.             {
143.                 result.setText("") + e);
144.             }
145.             return null;
146.         }
147.         }.execute();
148.     }
149.   });
150. }
151. pack();
152. }
153.
154. private JPanel northPanel;
155. }
```

**API** **java.net.HttpURLConnection 1.0**

- `InputStream getErrorStream()`

返回一个流，通过这个流可以读取Web服务器的错误信息。

**API** **java.net.URLEncoder 1.0**

- `static String encode(String s, String encoding)` 1.4

采用指定的字符编码模式（推荐使用“UTF-8”）对字符串s进行编码，并返回它的URL编码形式。在URL编码中，'A' - 'Z'，'a' - 'z'，'0' - '9'，'-'，'\_'，'.'和'\*'等字符保持不变，空格被编码成'+'，所有其他字符被编码成"%XY"形式的字节序列，其中0xXY为该字节十六进制数。

**API** **java.net.URLDecoder 1.2**

- `static string decode(String s, String encoding)` 1.4

采用指定编码模式对已编码字符串s进行解码，并返回结果。

在本章中，你已经看到了如何用Java编写网络客户端和服务器，以及如何从Web服务器上获取数据。下一章将讨论数据库连接行为，你将会学习如何通过使用JDBC API来实现用Java操作关系型数据库。同时，下一章还将简要地介绍层次型数据库（例如LDAP目录）和JNDI API。

# 第4章 数据库编程

- ▲ JDBC的设计
- ▲ 结构化查询语言
- ▲ JDBC配置
- ▲ 执行SQL语句
- ▲ 执行查询操作
- ▲ 可滚动和可更新的结果集

- ▲ 行集
- ▲ 元数据
- ▲ 事务
- ▲ Web和企业应用中的连接管理
- ▲ LDAP简介

1996年，Sun公司发布了第1版的Java数据库连接（JDBC）API，使编程人员可以通过这个API接口连接到数据库，并使用结构化查询语言（即SQL）完成对数据库的查找与更新。（SQL通常发音为“sequel”，它是数据库访问的业界标准。）JDBC自此成为了Java类库中最常使用的API之一。

到目前为止，JDBC的版本已更新过数次。作为Java SE 1.2的一部分，Sun公司于1998年发布了JDBC第2版。JDBC 3已经被囊括到了Java SE 1.4和5.0，而在本书出版之际，最新版的JDBC 4也被囊括到了Java SE 6中。

在本章中，我们将阐述JDBC幕后的关键思想，并将介绍（或者是复习）一下SQL（Structured Query Language，结构化查询语言），它是关系数据库的业界标准。我们还将提供了足够的详细信息和代码，使你可以将JDBC融入到日常的编程中。最后，本章会简单介绍一下分层数据库、轻量级目录访问协议（Lightweight Directory Access Protocol，LDAP）以及JNDI（Java Naming and Directory Interface，Java命名和目录接口）。

 **注意：**根据Sun的声明，JDBC是一个注册了商标的术语，而并非Java Database Connectivity的首字母缩写。对它的命名体现了对ODBC的致敬，后者是微软开创的标准数据库API，并因此而并入了SQL标准中。

## 4.1 JDBC的设计

从一开始，Sun公司的Java技术开发人员就意识到了Java在数据库应用方面的巨大潜力。从1995年开始，他们就致力于扩展Java标准类库，使之可以应用SQL访问数据库。他们最初希望，通过扩展Java，人们就可以用“纯”Java语言与任何数据库进行通信。但是，他们很快发现这是一项无法完成的任务：因为业界存在许多不同的数据库，且它们所使用的协议也各不相同。很多数据库供应商都表示支持Sun公司提供一套数据库访问的标准网络协议，因为他们每一家企业都希望Sun公司能采用自己的网络协议。

所有的数据库供应商和工具开发商都认为，如果Sun公司能够为SQL访问提供一套“纯”Java

API，同时提供一个驱动管理器，以允许第三方驱动程序可以连接到特定的数据库，那它们会显得非常有用。这样，数据库供应商就可以提供自己的驱动程序，并插入到驱动管理器中。另外还需要一套简单的机制，以使得第三方驱动程序可以向驱动管理器注册。因此，Sun公司制定了两套接口。应用程序开发者使用JDBC API，而数据库供应商和工具开发商则使用JDBC驱动API。

这种接口组织方式遵循了微软公司非常成功的ODBC模式，ODBC为C语言访问数据库提供了一套编程接口。JDBC和ODBC都基于同一个思想：根据API编写的程序都可以与驱动管理器进行通信，而驱动管理器则通过驱动程序与实际数据库进行通信。

所有这些都意味着JDBC API是大部分程序员不得不使用的接口（参见图4-1）。

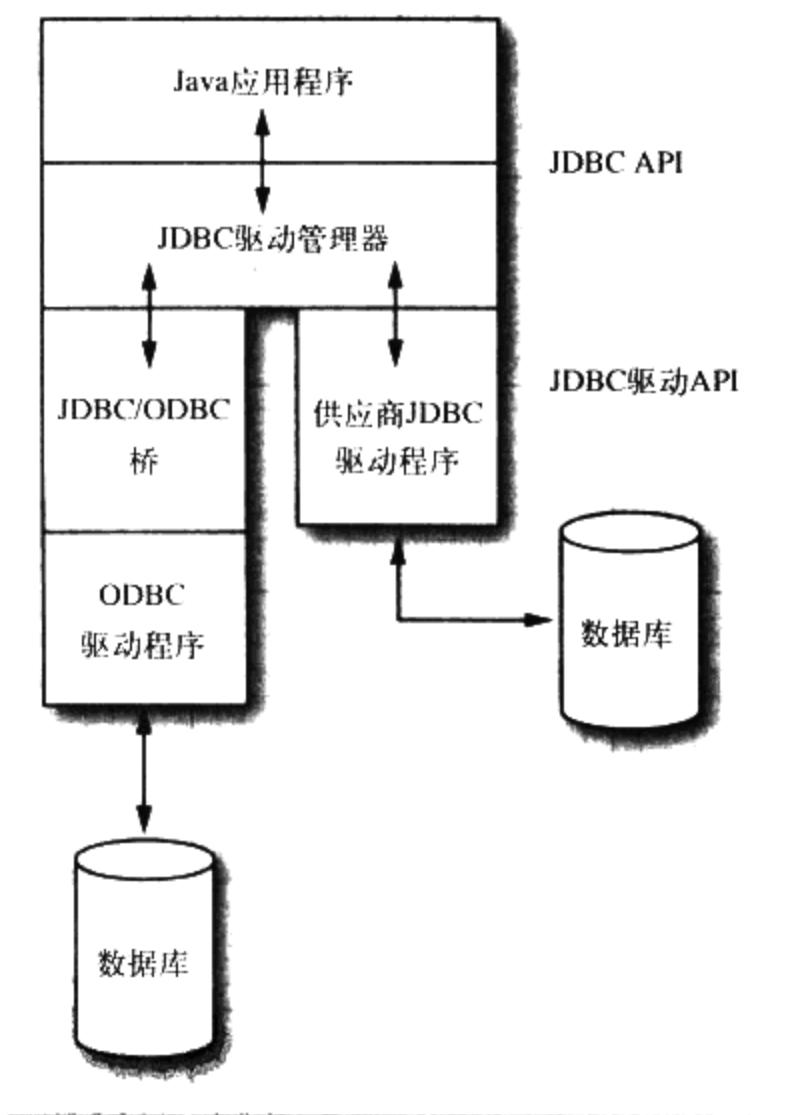


图4-1 JDBC到数据库的通信路径

注意：你可以在以下网址找到一份目前最常用的JDBC驱动程序的清单：  
<http://developers.sun.com/product/jdbc/drivers>。

#### 4.1.1 JDBC驱动程序类型

JDBC规范将驱动程序归结为以下几类：

- 第1类驱动程序将JDBC翻译成ODBC，然后使用一个ODBC驱动程序与数据库进行通信。Sun公司发布的较早版本的JDK中包含了一个这样的驱动程序：JDBC/ODBC桥，不过在

使用这个桥接器之前需要对ODBC进行相应的部署和正确的设置。在JDBC面世之初，桥接器可以方便地用于测试，却不太适用于产品的开发。目前我们可以得到许多更好的驱动程序，因此我们不建议使用JDBC/ODBC桥。

- 第2类驱动程序是由部分Java程序和部分本地代码组成的，用于与数据库的客户端API进行通信。在使用这种驱动程序之前，不仅需要安装Java类库，还需要安装一些与平台相关的代码。
- 第3类驱动程序是纯Java客户端类库，它使用一种与具体数据库无关的协议将数据库请求发送给服务器构件，然后该构件再将数据库请求翻译成特定数据库协议。
- 第4类驱动程序是纯Java类库，它将JDBC请求直接翻译成特定的数据库协议。

大部分数据库供应商都为他们的产品提供第3类或第4类驱动程序。与数据库供应商提供的驱动程序相比，许多第三方公司开发了很多更符合标准的产品，它们支持更多的平台、运行性能也更佳，某些情况下甚至具有更高的可靠性。

总之，JDBC最终是为了实现以下目标：

- 通过使用SQL语句，甚至是专有的SQL扩展，程序员可以利用Java语言开发访问数据库的应用。需要说明的是，扩展SQL仍然需要遵守Java语言的相关约定。
- 数据库供应商和数据库工具开发商可以提供底层的驱动程序。因此，他们有能力优化各自数据库产品的驱动程序。



**注意：**也许你会问为什么Sun公司没有采用ODBC模式，下面就是Sun公司在1996年5月举行的JavaOne研讨会上给出的说法：

- ODBC很难学会。
- ODBC中有几个命令需要配置很多复杂的选项，而在Java编程语言中所采用的风格是要具有简单而直观的方法，而且数量巨大。
- ODBC依赖于void\*指针和其他C语言特性，而这些特性并不太适用于Java编程语言。
- 与纯Java的解决方案相比，基于ODBC的解决方案天生就缺乏安全性，且难于部署。

#### 4.1.2 JDBC的典型用法

在传统的客户端/服务器模式中，通常是在服务器端配置数据库，而在客户端安装内容丰富的GUI界面（参见图4-2）。在此模型中，JDBC驱动程序应该部署在客户端。

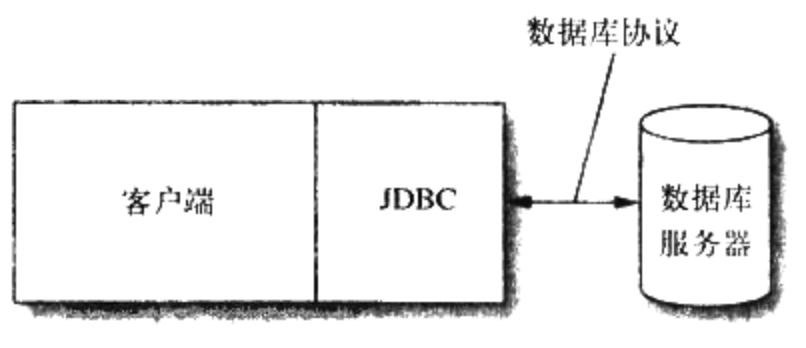


图4-2 传统的客户端/服务器应用

但是，如今全世界都在从客户端/服务器模式转向“三层应用模式”，甚至更高级的“n层应用模式”。在三层应用模式中，客户端不直接调用数据库，而是调用服务器上的中间件层，最后由中间件层完成数据库查询操作。这种三层应用模式有以下优点：它将可视化表示（位于客户端）从业务逻辑（位于中间层）和原始数据（位于数据库）中分离出来。因此，我们就可以从不同的客户端，如Java应用、Applet或者Web表单，来访问相同的数据和相同的业务规则。

客户端和中间层之间的通信可以通过HTTP（在将Web浏览器用作客户端时），或诸如远程方法调用RMI（在使用应用或Applet时，请参见第10章）这样的其他机制来完成。JDBC负责在中间层和后台数据库之间进行通信，图4-3展示了这种通信模式的基本架构。当然，这种模式有多种变体，尤其是Java企业版（Java EE）为应用服务器定义了一种结构，用于管理称为企业级JavaBean（EJB）的代码模块，并且提供了许多重要的服务，比如高安全性、负载平衡、访问请求的高速缓存以及对象-关系映射等。在此架构中，JDBC仍然扮演了重要的角色，即完成复杂的数据库查询。（关于Java企业版的更多信息请参<http://java.sun.com/javase/>。）

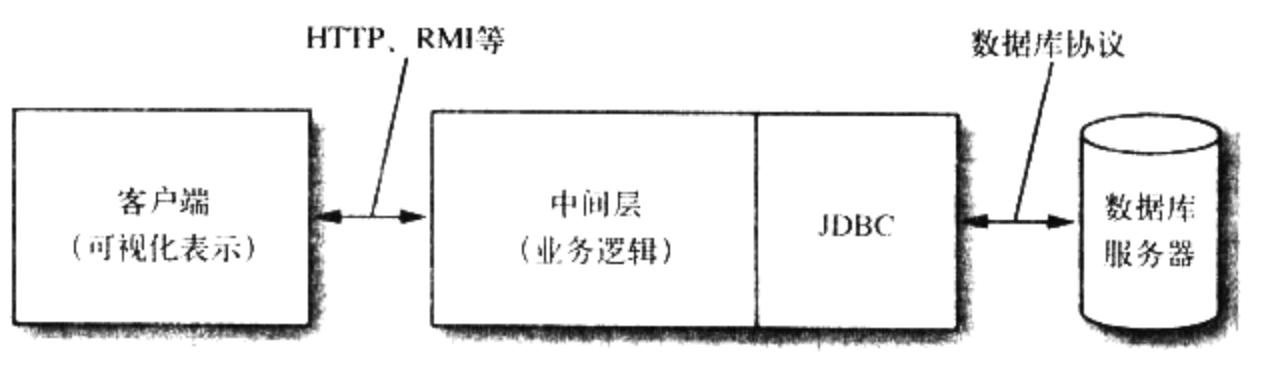


图4-3 三层结构的应用



**注意：**你可以在Applet中使用JDBC，但是也许你并不想这样做。在默认情况下，安全管理器只允许Applet与这个Applet的下载来源服务器（即Applet是从此服务器下载的）建立数据库连接。这就意味着Web服务器和数据库服务器必须在同一台机器上，但是这种配置并不典型。因此，你需要通过对该Applet进行签名来解决这个问题。

## 4.2 结构化查询语言

SQL是对所有现代关系型数据库都至关重要的命令行语言，JDBC则使得我们可以通过SQL与数据库进行通信。桌面数据库通常都有一个图形用户界面；使用这种界面，用户可以直接操作数据。但是，基于服务器的数据库只能使用SQL进行访问。

我们可以将JDBC包看作是一个用于将SQL语句传递给数据库的应用编程接口（API）。在本节中，我们将简单介绍一下SQL。如果之前没有接触过SQL，你会发现这些介绍是远远不够的，你可以参阅关于SQL的其他著作。我们推荐Alan Beaulieu所著的《Learning SQL》（2005年由O'Reilly出版社出版），或者还可以参考C. J. Date和Hugh Darwen合著的《A Guide to the SQL Standard》（1997年由Addison-Wesley出版社出版）。

可以将数据库想象成一组由行和列组成的表格，其中每一列都有列名（column name），而

每一行则包含了实际的数据。

作为本书的数据库实例，我们将使用一系列数据库表格来描述一组经典的计算机著作（请参见表4-1到表4-4）。

表4-1 Authors表

| Author_ID | Name      | Fname        |
|-----------|-----------|--------------|
| ALEX      | Alexander | Christopher  |
| BROOKS    | Brooks    | Frederick P. |
| ...       | ...       | ...          |

表4-2 Books表

| Title  | ISBN          | Publisher_ID | Price |
|--|---------------|--------------|-------|
| A Guide to the SQL Standard                        | 0-201-96426-0 | 0201         | 47.95 |
| A Pattern Language: Towns, Buildings, Construction | 0-19-501919-9 | 019          | 65.00 |
| ...  | ...           | ...          | ...   |

表4-3 BooksAuthors表

| ISBN          | Author_ID | Seq_No |
|---------------|-----------|--------|
| 0-201-96426-0 | DATE      | 1      |
| 0-201-96426-0 | DARW      | 2      |
| 0-19-501919-9 | ALEX      | 1      |
| ...           | ...       | ...    |

表4-4 Publishers表

| Publisher_ID | Name              | URL           |
|--------------|-------------------|---------------|
| 0201         | Addison-Wesley    | www.aw-bc.com |
| 0407         | John Wiley & Sons | www.wiley.com |
| ...          | ...               | ...           |

图4-4显示的是一个Books表的视图，而图4-5显示了对Books表和Publishers表执行连接操作后的结果。Books表和Publishers表都包含了一个表示出版社的字段。当我们利用出版社编号对这两个表进行连接操作时，我们就得到了由连接后的表格的值所组成的查询结果。结果中的每一行都包含了图书的信息、出版社名称及其Web页的URL地址。注意，有的出版社名称和URL地址重复出现在数行中，因为这些行都对应于同一个出版社。

对表格进行连接操作的好处是能避免在数据库表格中出现不必要的重复数据。例如，有一种比较原始的数据库设计是在Books表中设置出版社名称和URL地址字段。但是这样一来，数据库本身，而非查询结果，将出现许多重复数据。如果出版社的Web地址发生了改变，就需要更新所有的重复数据。显然，有时这很容易导致错误。在关系模型中，我们将数据分配到多个表格中，使得所有信息都不会出现不必要的重复。例如，每个出版社的URL地址只在出版社表中出现一次。如果需要将此信息与其他信息合并，我们只需对表格进行连接操作。

The screenshot shows a window titled "COREJAVA Books" with a menu bar: File, Edit, View, Tools, Window, Help. The main area displays a table with columns: Title, ISBN, Publisher\_ID, and Price. The data includes titles like "UNIX System Administration Handbook", "The C Programming Language", and "Design Patterns". The table has 20 records.

|   | Title   | ISBN          | Publisher_ID | Price  |
|---|---|---------------|--------------|--------|
| > | UNIX System Administration Handbook                         | 0-13-020601-6 | 013          | 68.00  |
|   | The C Programming Language                                  | 0-13-110362-8 | 013          | 42.00  |
|   | A Pattern Language: Towns, Buildings, Construction          | 0-19-501919-9 | 019          | 65.00  |
|   | Introduction to Automata Theory, Languages, and Computation | 0-201-44124-1 | 0201         | 105.00 |
|   | Design Patterns   | 0-201-63361-2 | 0201         | 54.99  |
|   | The C++ Programming Language                                | 0-201-70073-5 | 0201         | 64.99  |
|   | The Mythical Man Month                                      | 0-201-83595-9 | 0201         | 29.95  |
|   | Computer Graphics: Principles and Practice                  | 0-201-84840-6 | 0201         | 79.99  |
|   | The Art of Computer Programming vol. 1                      | 0-201-89683-4 | 0201         | 59.99  |
|   | The Art of Computer Programming vol. 2                      | 0-201-89684-2 | 0201         | 59.99  |
|   | The Art of Computer Programming vol. 3                      | 0-201-89685-0 | 0201         | 59.99  |
|   | A Guide to the SQL Standard                                 | 0-201-96426-0 | 0201         | 47.95  |
|   | Introduction to Algorithms                                  | 0-262-03293-7 | 0262         | 80.00  |
|   | Applied Cryptography  | 0-471-11709-9 | 0471         | 60.00  |
|   | JavaScript: The Definitive Guide                            | 0-596-00048-0 | 0596         | 44.95  |
|   | the Cathedral and the Bazaar                                | 0-596-00108-8 | 0596         | 16.95  |
|   | The Soul of a New Machine                                   | 0-579-60261-5 | 0679         | 18.95  |
|   | The Codebreakers  | 0-684-83130-9 | 07434        | 70.00  |
|   | Cuckoo's Egg  | 0-7434-1146-3 | 07434        | 13.95  |
|   | The UNIX Hater's Handbook                                   | 1-56884-203-1 | 0471         | 16.95  |

图4-4 包含图书信息的示例表格

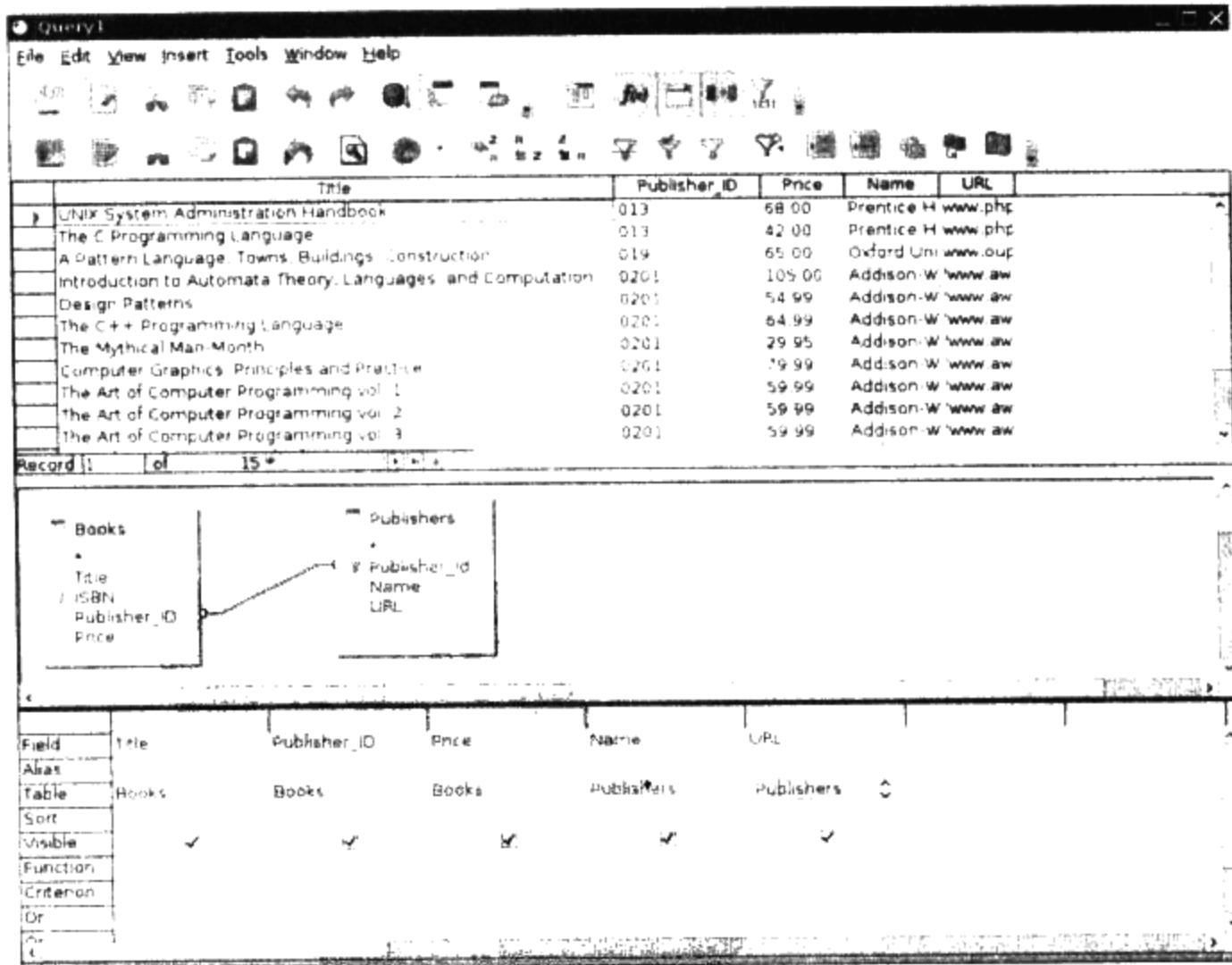


图4-5 对两个表进行连接操作

在上述两幅插图中，可以看到一个用于查看和链接表格的图形工具。许多数据库供应商都会提供工具让用户以简单的方式查询数据库，其方法是将列名连接起来，并将信息填入各表单

中，这种工具通常称为实例查询（Query by Example, QBE）工具。而使用SQL的查询则是利用SQL语法以文本方式编写的。例如，

```
SELECT Books.Title, Books.Publisher_Id, Books.Price, Publishers.Name, Publishers.URL  
FROM Books, Publishers  
WHERE Books.Publisher_Id = Publishers.Publisher_Id
```

在本节的余下部分中，我们将介绍如何编写这样的查询语句。如果你已经熟悉SQL了，就可以跳过这部分内容。

按照惯例，SQL关键字全部使用大写字母。当然，也可以不这样做。

SELECT语句相当灵活。仅使用下面这个查询语句，就可以查出Books表中的所有记录：

```
SELECT * FROM Books
```

在每一个SQL的SELECT语句中，FROM子句都是必不可少的。FROM子句用于告知数据库应该在哪个表格上查询数据。

我们还可以选择所需要的列：

```
SELECT ISBN, Price, Title  
FROM Books
```

并且还可以在查询语句中使用WHERE子句来限定所要选择的行：

```
SELECT ISBN, Price, Title  
FROM Books  
WHERE Price <= 29.95
```

请小心使用“相等”这个比较操作。与Java编程语言不同，SQL使用=和<>而非==和!=来进行相等比较。

 注意：有些数据库供应商的产品支持在进行不等于比较时使用!=。这不符合标准SQL的语法，所以我们建议不要使用这种方法。

WHERE子句也可以使用LIKE操作符来实现模式匹配。不过，这里的通配符并不是通常所说的\*和?，而是用%表示0或0以上个字符，用下划线表示单个字符。例如，

```
SELECT ISBN, Price, Title  
FROM Books  
WHERE Title NOT LIKE '%n_x%'
```

这条语句排除了所有书名中包含UNIX或者Linux的图书。

请注意，字符串都是用单引号括起来的，而非双引号。字符串中的单引号则需要用一对单引号代替。例如，

```
SELECT Title  
FROM Books  
WHERE Title LIKE '%''%'
```

上述语句返回所有包含单引号的书名。

你也可以从多个表格中选取数据：

```
SELECT * FROM Books, Publishers
```

如果没有WHERE子句，上述查询语句意义不大，它只是罗列了两个表中所有记录的组合。在我们这个例子中，Books表有20行记录，Publishers表有8行记录，合并的结果将产生 $20 \times 8$

条记录，其中不乏大量重复数据。实际上我们需要对查询结果进行限制，只对那些图书与出版社相匹配的数据感兴趣。

```
SELECT * FROM Books, Publishers
WHERE Books.Publisher_Id = Publishers.Publisher_Id
```

这条语句的查询结果共有20行记录，每一条记录对应于一本书，因为每本书都在Publisher表中只对应一个出版社。

每当查询语句涉及多个表格时，相同的列名可能会出现在两个不同的地方。在我们的例子中也存在这种情况，Books表和Publishers表都拥有一个列名为Publisher\_Id的列。当出现歧义时，可以在每个列名前添加它所在表格的表名作为前缀，比如Books.Publisher\_Id。

也可以使用SQL来改变数据库中的数据。例如，假设现在要将所有书名中包含“C++”的图书降价5美元。

```
UPDATE Books
SET Price = Price - 5.00
WHERE Title LIKE '%C++%'
```

类似地，要删除所有的C++图书，可以使用下面的DELETE查询。

```
DELETE FROM Books
WHERE Title LIKE '%C++%'
```

此外，SQL中还有许多内置函数，用于对某一列计算平均值、查找最大值和最小值以及其他许多功能。有关这方面的信息可以参阅网址<http://sqlzoo.net>。（此站点还提供了一套既风趣幽默又颇具交互性的SQL指南。）

通常，可以使用INSERT语句向表格中插入值：

```
INSERT INTO Books
VALUES ('A Guide to the SQL Standard', '0-201-96426-0', '0201', 47.95)
```

我们必须为每一条插入到表格中的记录使用一次INSERT语句。

当然，在查询、修改和插入数据之前，必须要有存储数据的空间。可以使用CREATE TABLE语句创建一个新的表格，还可以为每一列指定列名和数据类型。

```
CREATE TABLE Books
(
    Title CHAR(60),
    ISBN CHAR(13),
    Publisher_Id CHAR(6),
    Price DECIMAL(10,2)
)
```

表4-5给出了最常见的SQL数据类型。

表4-5 SQL数据类型

| 数据类型                                 | 说 明                  |
|--------------------------------------|----------------------|
| INTEGER或INT                          | 通常为32位的整数            |
| SMLINT                               | 通常为16位的整数            |
| NUMERIC(m,n), DECIMAL(m,n)或 DEC(m,n) | m位长的定点十进制数，其中小数点后为n位 |
| FLOAT(n)                             | 运算精度为n位二进制数的浮点数      |

(续)

| 数据类型                 | 说明                |
|----------------------|-------------------|
| REAL                 | 通常为32位浮点数         |
| DOUBLE               | 通常为64位浮点数         |
| CHARACTER(n)或CHAR(n) | 固定长度为n的字符串        |
| VARCHAR(n)           | 最大长度为n的字符串        |
| BOOLEAN              | 布尔值               |
| DATE                 | 日历日期（与具体的实现有关）    |
| TIME                 | 当前时间（与具体的实现有关）    |
| TIMESTAMP            | 当前日期和时间（与具体的实现有关） |
| BLOB                 | 二进制大对象            |
| CLOB                 | 字符大对象             |

在本书中，我们不再介绍更多的子句，比如可以应用于CREATE TABLE命令的主键和限制性语句。

### 4.3 JDBC配置

当然，你需要有一个可获得其JDBC驱动程序的数据库软件。目前这方面有许多出色的软件可供选择，比如IBM DB2、Microsoft SQL Server、MySQL、Oracle和PostgreSQL。

为了练习本部分内容，还需要创建一个数据库，我们假定将这个数据库命名为COREJAVA。请创建一个新的数据库，或者让数据库管理员创建一个拥有适当权限的数据库，因为你需要拥有对这个数据库进行创建、更新和删除表的权限。

如果你以前从未安装过采用客户端/服务器模式的数据库，那么就会发现配置这样一个数据库会稍显复杂并且难于判断故障的原因。如果安装的数据库无法正常运行，那么最好请专家来帮忙。

如果第一次接触数据库，我们建议使用Apache Derby，它已经成为了JDK 6的一部分。（如果你使用的JDK不包含它，可以从<http://db.apache.org/derby>处下载Apache Derby）。

 注意：Sun将包含在JDK中的Apache Derby版本称为JavaDB，为了避免混淆，我们在本章中称其为Derby。

在编写第一个数据库程序之前，你需要收集大量的信息和文件，下面将讨论这些内容。

#### 4.3.1 数据库URL

在连接数据库时，我们必须使用各种与数据库类型相关的参数，例如主机名、端口号和数据库名。

JDBC使用了一种与普通URL相类似的语法来描述数据源。下面是这种语法的两个实例：

```
jdbc:derby://localhost:1527/COREJAVA;create=true
jdbc:postgresql:COREJAVA
```

上述JDBC URL指定了名为COREJAVA的一个Derby数据库和一个PostgreSQL数据库。JDBC URL的一般语法为：

```
jdbc:subprotocol:other stuff
```

其中，*subprotocol*用于指明连接到数据库的特定驱动程序。

*other stuff*参数的格式随所使用的*subprotocol*不同而不同。如果要了解具体格式，你需要查阅数据库供应商提供的相关文档。

### 4.3.2 驱动程序JAR文件

你需要获得包含了你所使用的数据库的驱动程序的JAR文件。如果你使用的是Derby，那么就需要derbeclient.jar；如果你使用的是其他的数据库，那么就需要去寻找恰当的驱动程序。例如，PostgreSQL的驱动程序可以在<http://jdbc.postgresql.org>处找到。

在运行访问数据库的程序时，需要将驱动程序的JAR文件包括到类路径中（编译时并不需要整个JAR文件）。

在从命令行启动程序时，只需要使用下面的命令：

```
java -classpath .:driverjar ProgramName
```

在Windows上，可以使用分号将当前路径（即由.字符表示的路径）与驱动程序JAR文件分隔开。

### 4.3.3 启动数据库

数据库服务器在连接之前需要先启动，启动的细节取决于所使用的数据库。

在使用Derby数据库时，需要遵循下面的步骤：

1. 打开命令shell，并转到将来存有数据库文件的目录中。
2. 定位derbyrun.jar。对于某些JDK版本，它包含在jdk/db/lib目录中，而另一些版本则将其置于单独的JavaDB安装目录中。我们用derby来表示包含lib/derbyrun.jar的目录。
3. 运行下面的命令：

```
java -jar derby/lib/derbyrun.jar server start
```

4. 仔细检查数据库是否正确工作了。然后创建一个名为ij.properties并包含下面各行的文件：

```
ij.driver=org.apache.derby.jdbc.ClientDriver  
ij.protocol=jdbc:derby://localhost:1527/  
ij.database=COREJAVA;create=true
```

在另一个命令shell中，通过执行下面的命令来运行Derby的交互式脚本执行工具（称为ij）：

```
java -jar derby/lib/derbyrun.jar ij -p ij.properties
```

现在，可以发布像下面这样的SQL命令了：

```
CREATE TABLE Greetings (Message CHAR(20));  
INSERT INTO Greetings VALUES ('Hello, World!');  
SELECT * FROM Greetings;  
DROP TABLE Greetings;
```

注意，每条命令都需要以分号结尾，要退出编辑器，可以键入  
EXIT；

5. 在使用完数据库之后，可以用下面的命令关闭服务器：

```
java -jar derby/lib/derbyrun.jar server shutdown
```

如果使用其他的数据库，则需要查看文档，以了解如何启动和关闭数据库服务器，以及如何连接到数据库和发布SQL命令。

#### 4.3.4 注册驱动器类

某些JDBC的JAR文件（例如包含在Java SE 6中的Derby驱动程序）将自动注册驱动器类，在这种情况下，可以跳过本节所描述的手动注册步骤。包含META-INF/services/java.sql.Driver文件的JAR文件可以自动注册，解压缩驱动程序JAR文件就可以检查其是否包含该文件。

 **注意：**这种注册机制使用的是JAR规范中几乎不为人知的特性，请参见<http://java.sun.com/javase/6/docs/technotes/guides/jar/jar.html#Service%20Provider>。自动注册对于遵循JDBC4的驱动程序是必需具备的特性。

如果驱动程序JAR不支持自动注册，那就需要找出数据库提供商使用的JDBC驱动器类的名字。典型的驱动器名字如下：

```
org.apache.derby.jdbc.ClientDriver  
org.postgresql.Driver
```

通过使用DriverManager，可以用两种方式来注册驱动器。一种方式是在Java程序中加载驱动器类，例如：

```
Class.forName("org.postgresql.Driver"); // force loading of driver class
```

这条语句将使得驱动器类被加载，由此将执行可以注册驱动器的静态初始化器。

另一种方式是设置jdbc.drivers属性。可以用命令行参数来指定这个属性，例如：

```
java -Djdbc.drivers=org.postgresql.Driver ProgramName
```

或者在应用中用下面这样的调用来设置系统属性

```
System.setProperty("jdbc.drivers", "org.postgresql.Driver");
```

在这种方式中可以提供多个驱动器，用冒号将它们分隔开，例如

```
org.postgresql.Driver:org.apache.derby.jdbc.ClientDriver
```

#### 4.3.5 连接到数据库

在Java程序中，我们可以在代码中打开一个数据库连接，例如：

```
String url = "jdbc:postgresql:COREJAVA";  
String username = "dbuser";  
String password = "secret";  
Connection conn = DriverManager.getConnection(url, username, password);
```

驱动管理器遍历所有注册过的驱动程序，以便找到一个能够使用数据库URL中指定的子协议的驱动程序。

getConnection方法返回一个Connection对象。在下一节中，我们将详细介绍如何使用Connection对象来执行SQL语句。

要连接到数据库，我们还需要知道数据库的名字和口令。

注意：在默认情况下，Derby允许我们使用任何用户名进行连接，并且不检查口令。它会为每个用户生成一个单独的模式，而默认的用户名是app。

程序清单4-1中的测试程序将所有这些步骤放到了一起：它从名为database.properties的文件中加载连接参数，并连接到数据库。示例代码中提供的database.properties文件包含的是关于Derby数据库的连接信息，如果使用其他的数据库，则需要将与数据库相关的连接信息放到整个文件中。下面是一个用于连接到PostgreSQL数据库的示例：

```
jdbc.drivers=org.postgresql.Driver  
jdbc.url=jdbc:postgresql:COREJAVA  
jdbc.username=dbuser  
jdbc.password=secret
```

在连接到数据库之后，这个测试程序执行了下面的SQL语句：

```
CREATE TABLE Greetings (Message CHAR(20))  
INSERT INTO Greetings VALUES ('Hello, World!')  
SELECT * FROM Greetings
```

SELECT的结果将被打印出来，你应该可以看到如下的输出：

```
Hello, World!
```

然后，通过执行下面的语句移除了这张表

```
DROP TABLE Greetings
```

要运行这个测试程序，需要启动数据库，并像下面这样启动这个程序

```
java -classpath .:driverJAR TestDB
```

提示：调试与JDBC相关的问题时，最好的方法是使能JDBC的跟踪特性。调用DriverManager.setLogWriter方法可以将跟踪信息发送给PrintWriter，而PrintWriter将输出JDBC活动的详细列表。大多数JDBC驱动程序的实现都提供了用于跟踪的附加特性，例如，在使用Derby时，可以在JDBC的URL中添加traceFile选项，如jdbc:derby://localhost:1527/COREJAVA;create=true;traceFile=trace.out。

#### 程序清单4-1 TestDB.java

```
1. import java.sql.*;  
2. import java.io.*;  
3. import java.util.*;  
4.  
5. /**  
6.  * This program tests that the database and the JDBC driver are correctly configured.  
7.  * @version 1.01 2004-09-24  
8.  * @author Cay Horstmann  
9.  */  
10. class TestDB  
11. {  
12.     public static void main(String args[])  
13.     {  
14.         try  
15.         {  
16.             runTest();  
17.         }  
18.     }
```

```
18.     catch (SQLException ex)
19.     {
20.         for (Throwable t : ex)
21.             t.printStackTrace();
22.     }
23.     catch (IOException ex)
24.     {
25.         ex.printStackTrace();
26.     }
27. }
28.
29. /**
30. * Runs a test by creating a table, adding a value, showing the table contents, and
31. * removing the table.
32. */
33. public static void runTest() throws SQLException, IOException
34. {
35.     Connection conn = getConnection();
36.     try
37.     {
38.         Statement stat = conn.createStatement();
39.
40.         stat.executeUpdate("CREATE TABLE Greetings (Message CHAR(20))");
41.         stat.executeUpdate("INSERT INTO Greetings VALUES ('Hello, World!')");
42.
43.         ResultSet result = stat.executeQuery("SELECT * FROM Greetings");
44.         if (result.next())
45.             System.out.println(result.getString(1));
46.         result.close();
47.         stat.executeUpdate("DROP TABLE Greetings");
48.     }
49.     finally
50.     {
51.         conn.close();
52.     }
53. }
54.
55. /**
56. * Gets a connection from the properties specified in the file database.properties
57. * @return the database connection
58. */
59. public static Connection getConnection() throws SQLException, IOException
60. {
61.     Properties props = new Properties();
62.     FileInputStream in = new FileInputStream("database.properties");
63.     props.load(in);
64.     in.close();
65.
66.     String drivers = props.getProperty("jdbc.drivers");
67.     if (drivers != null) System.setProperty("jdbc.drivers", drivers);
68.     String url = props.getProperty("jdbc.url");
69.     String username = props.getProperty("jdbc.username");
70.     String password = props.getProperty("jdbc.password");
71.
72.     return DriverManager.getConnection(url, username, password);
73. }
74. }
```

**java.sql.DriverManager 1.1**

- static Connection getConnection(String url, String user, String password)
- 建立一个到指定数据库的连接，并返回一个Connection对象。

## 4.4 执行SQL语句

在执行SQL命令之前，首先需要创建一个Statement对象。要创建statement对象，需要使用调用DriverManager.getConnection方法所获得的Connection对象。

```
Statement stat = conn.createStatement();
```

接着，将要执行的SQL语句放入字符串中，例如：

```
String command = "UPDATE Books"
    + " SET Price = Price - 5.00"
    + " WHERE Title NOT LIKE '%Introduction%'";
```

然后，调用Statement类中的executeUpdate方法：

```
stat.executeUpdate(command);
```

executeUpdate方法将返回受SQL命令影响的行数，或者对于不返回行数的语句返回0。例如，在先前的例子中调用executeUpdate方法将返回那些降价5美元的行数。

executeUpdate方法既可以执行诸如INSERT、UPDATE和DELETE之类的操作，也可以执行诸如CREATE TABLE和DROP TABLE之类的数据定义语句。但是，执行SELECT查询时必须使用executeQuery方法。另外还有一个execute方法可以执行任意的SQL语句，此方法通常只用于用户提供的交互式查询。

当我们执行查询操作时，通常最感兴趣的是查询结果。executeQuery方法返回一个ResultSet对象，可以通过它来每次一行地迭代遍历所有查询结果。

```
ResultSet rs = stat.executeQuery("SELECT * FROM Books")
```

分析结果集时通常可以使用类似如下循环语句的代码：

```
while (rs.next())
{
    look at a row of the result set
}
```



**警告：**ResultSet类的迭代方法与java.util.Iterator接口稍有不同。对于ResultSet类，迭代器初始化时被设定在第一行之前的位置，必须调用next方法将它移动到第一行。另外，它没有hasNext方法，我们需要不断地调用next，直至该方法返回false。

结果集中行的顺序是任意的。除非使用ORDER BY子句指定行的顺序，否则不能为行序强加任何意义。

查看每一行时，可能希望知道其中每一列的内容，有许多访问器（accessor）方法可以用于获取这些信息。

```
String isbn = rs.getString(1);
double price = rs.getDouble("Price");
```

不同的数据类型有不同的访问器，比如`getString`和`getdouble`。每个访问器都有两种形式，一种接受数字参数，另一种接受字符串参数。当使用数字参数时，我们指的是该数字所对应的列。例如，`rs.getString(1)`返回的是当前行中第一列的值。

 **警告：**与数组的索引不同，数据库的列序号是从1开始计算的。

当使用字符串参数时，指的是结果集中以该字符串为列名的列。例如，`rs.getDouble("Price")`返回列名为Price的列所对应的值。使用数字参数效率更高一些，但是使用字符串参数可以使代码易于阅读和维护。

当`get`方法的类型和列的数据类型不一致时，每个`get`方法都会进行合理的类型转换。例如，调用`rs.getString("Price")`时，该方法会将Price列的浮点值转换成字符串。

 **API** `java.sql.Connection 1.1`

- `Statement createStatement()`

创建一个`Statement`对象，用以执行不带参数的SQL查询和更新。

- `void close()`

用于立即关闭当前的连接以及释放由它所创建的JDBC资源。

 **API** `java.sql.Statement 1.1`

- `ResultSet executeQuery(String sqlQuery)`

执行给定字符串中的SQL语句，并返回一个用于查看查询结果的`ResultSet`对象。

- `int executeUpdate(String sqlStatement)`

执行字符串中指定的INSERT、UPDATE或DELETE等SQL语句。还可以执行数据定义语言(Data Definition Language, DDL)的语句，如CREATE TABLE。返回受影响的记录总数，如果是没有更新计数的语句，则返回-1。

- `boolean execute(String sqlStatement)`

执行字符串中指定的SQL语句。可能会产生多个结果集和更新数。如果第一个执行结果是结果集，则返回true；反之，返回false。调用`getResultSet`或`getUpdateCount`方法可以得到第一个执行结果。请参见第4.5.4节中关于处理多结果集的详细信息。

- `ResultSet getResultSet()`

返回前一条查询语句的结果集。如果前一条语句未产生结果集，则返回null值。对于每一条执行过的语句，该方法只能被调用一次。

- `int getUpdateCount()`

返回受前一条更新语句影响的行数。如果前一条语句未更新数据库，则返回-1。对于每一条执行过的语句，该方法只能被调用一次。

- `void close()`

关闭`Statement`对象以及它所对应的结果集。

- `boolean isClosed()`

如果语句被关闭，则返回true。

#### API java.sql.ResultSet 1.1

- boolean next()

将结果集中的当前行向前移动一行。如果已经到达最后一行的后面，则返回false。注意，初始情况下必须调用该方法才能转到第一行。

- Xxx getXxx(int columnNumber)

- Xxx getXxx(String columnName)

(Xxx指数据类型，例如int、double、String、Date等。)

用给定的列序号或列标签返回该列的值，并将值转换成指定类型。列标签是SQL的AS字句中指定的标签，在没有使用AS时，它就是列名。

- int findColumn(String columnName)

根据给定的列名，返回该列的序号。

- void close()

立即关闭当前的结果集。

- boolean isClosed() 6

如果语句被关闭，则返回true。

#### 4.4.1 管理连接、语句和结果集

每个Connection对象都可以创建一个或一个以上的Statement对象。同一个Statement对象可以用于多个不相关的命令和查询。但是，一个Statement对象最多只能打开一个结果集。如果需要执行多个查询操作，且需要同时分析查询结果，那么必须创建多个Statement对象。

需要说明的是，至少有一种常用的数据库（Microsoft SQL Server）的JDBC驱动程序只允许同时存在一个激活的Statement对象。使用DatabaseMetaData类中的getMaxStatements方法可以获取JDBC驱动程序同时支持的语句对象的总数。

这看上去似乎很有局限性。但实际上，我们通常并不需要同时处理多个结果集。如果结果集相互关联，我们就可以使用组合查询，这样就只需要分析一个结果。对数据库进行组合查询比使用Java程序遍历多个结果集要高效得多。

当使用完ResultSet、Statement或Connection对象时，应立即调用close方法。这些对象都使用了规模较大的数据结构，所以我们不应该等待垃圾回收器来处理它们。

如果Statement对象上有一个打开的结果集，那么调用close方法将自动关闭该结果集。同样地，调用Connection类的close方法将关闭该连接上的所有语句。

如果所用连接都是短时的，那么无需考虑关闭语句和结果集。只需将close语句放在finally块中，以便确保最终关闭连接对象。

```
try
{
    Connection conn = . . .
    try
    {
```

```

    Statement stat = conn.createStatement();
    ResultSet result = stat.executeQuery(queryString);
    process query result
}
finally
{
    conn.close();
}
}
catch (SQLException ex)
{
    handle exception
}

```

**!** 提示：我们可以使用try/finally块关闭连接，而使用一个单独的try/catch块处理异常。分割try程序块可以提高代码的可读性和可维护性。

#### 4.4.2 分析SQL异常

每个SQLException都有一个由多个SQLException对象构成的链，这些对象可以通过getNextException方法获取。这个异常链是每个异常都具有由Throwable对象构成的“成因”链之外的异常链（请参见第I卷第11章以了解Java异常的详细信息），因此，我们需要用两个迭代器的循环来完全枚举所有的异常。幸运的是，Java SE 6改进了SQLException类，让其实现了Iterable<Throwable>接口，其iterator()方法可以产生一个Iterable<Throwable>，这个迭代器可以迭代这两个链，首先迭代第一个SQLException的成因链，然后迭代下一个SQLException，以此类推。我们可以简单地使用下面这个改进的for循环：

```

for (Throwable t : sqlException)
{
    do something with t
}

```

可以在SQLException上调用getSQLState和getErrorCode方法来进一步分析它，其中第一个方法将产生符合X/Open或SQL:2003标准的字符串（调用DatabaseMetaData的方法getSQLStateType可以查出驱动程序所使用的标准）。而错误代码是提供商相关的。

从Java SE 6开始，SQL异常按照层次结构树的方式组织到了一起（如图4-6所示），这使得我们可以按照提供商无关的方式来捕获具体的错误类型。

另外，数据库驱动程序可以将非致命问题作为警告报告，我们可以从连接、语句和结果集中获取这些警告。SQLWarning类是SQLException的子类（尽管SQLWarning不会被当作异常抛出），我们可以调用getSQLState和getErrorCode来获取有关警告的更多信息。与SQL异常类似，警告也是串成链的。要获得所有的警告，可以使用下面的循环：

```

SQLWarning w = stat.getWarning();
while (w != null)
{
    do something with w
    w = w.nextWarning();
}

```

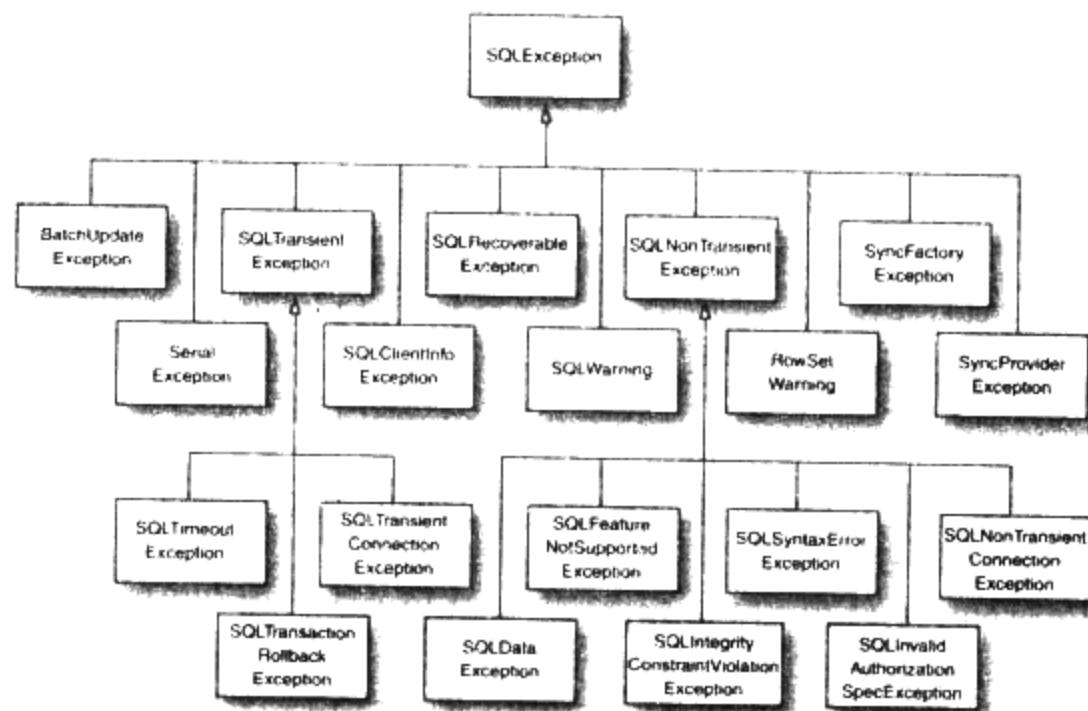


图4-6 SQL异常类型

当数据从数据库中读出并意外被截断时，SQLWarning的DataTruncation子类就派上用场了。如果数据截断发生更新语句中，那么DataTruncation将会被当作异常抛出。

#### **API** java.sql.SQLException 1.1

- SQLException getNextException()

返回链接到该SQL异常的下一个SQL异常，或者在到达链尾时返回null。

- Iterator<Throwable> iterator() 6

获取迭代器，可以迭代链接的SQL异常和它们的成因。

- String getSQLState()

获取“SQL 状态”，即标准化的错误代码。

- int getErrorCode()

获取提供商相关的错误代码。

#### **API** java.sql.Warning 1.1

- SQLWarning getNextWarning()

返回链接到该警告的下一个警告，或者在到达链尾时返回null。

#### **API** java.sql.Connection 1.1

java.sql.Statement 1.1

java.sql.ResultSet 1.1

- SQLWarning getWarnings()

- SQLWarning getWarnings()

返回未处理警告中的第一个，或者在没有未处理警告时返回null。

#### **java.sql.DataTruncation 1.1**

- `boolean getParameter()`

如果在参数上进行了数据截断，则返回true，如果在列上进行了数据截断，则返回false。

- `int getIndex()`

返回被截断的参数和列的索引。

- `int getDataSize()`

返回应该被传输的字节数量，或者在该值未知的情况下返回-1。

- `int getTransferSize()`

返回实际被传输的字节数量，或者在该值未知的情况下返回-1。

### 4.4.3 组装数据库

至此，大家也许都迫不及待地想编写一个真正实用的JDBC程序了。如果我们可以编写一段程序来执行之前所介绍的那些巧妙的查询，那当然很好。不过，在此之前我们还有一个问题没有解决：目前数据库中还没有数据。我们需要组装数据库，并且也确实存在一种简单方法可以实现此目的：用一系列的SQL指令来创建数据表并向其中插入数据。大多数数据库系统都可以从文本文件中执行一系列的SQL指令，但是在语句终止符和其他一些文法问题上，这些数据库系统之间存在着令人讨厌的差异。

正是由于这个原因，我们使用JDBC创建了一个简单的程序，它从文件中读取SQL指令，其中一条指令占据一行，然后执行它们。

该程序专门用于从下列格式的文本文件中读取数据：

```
CREATE TABLE Publisher (Publisher_Id CHAR(6), Name CHAR(30), URL CHAR(80));
INSERT INTO Publishers VALUES ('0201', 'Addison-Wesley', 'www.aw-bc.com');
INSERT INTO Publishers VALUES ('0471', 'John Wiley & Sons', 'www.wiley.com');
...
```

程序清单4-2是用以读取SQL语句文件以及执行这些语句的程序代码。通读这些代码并不重要，我们在这里只是提供了这样的程序，使你能够组装数据库并运行本章剩余部分的代码。

请确认你的数据库服务器是在运行的，然后可以使用如下方法运行程序：

```
java -classpath .:driverPath ExecSQL Books.sql
java -classpath .:driverPath ExecSQL Authors.sql
java -classpath .:driverPath ExecSQL Publishers.sql
java -classpath .:driverPath ExecSQL BooksAuthors.sql
```

在运行程序之前，请检查一下**database.properties**文件是否已经为你的运行环境进行了正确设置。请查看第4.3.5节。



注意：你的数据库可能也包含直接从SQL文件读取的工具，例如，在使用Derby时，可以运行下面的命令：

```
java -jar derby/lib/derbyrun.jar ij -p ij.properties Books.sql
```

( *ij.properties* 文件在第 4.3.3 节中描述过。)

或者，如果你熟悉 Ant，也可以使用 Ant 的 sql 任务。

在用于 ExecSQL 命令的数据格式中，我们允许每行的结尾都可以有一个可选的分号，因为大多数数据库工具和 Ant 都希望使用这种格式。

下面将简要介绍一下 ExecSQL 程序的操作步骤：

1. 连接数据库。getConnection 方法读取 database.properties 文件中的属性信息，并将属性 jdbc.drivers 添加到系统属性中。驱动管理器使用属性 jdbc.drivers 加载正确的驱动程序。getConnection 方法使用 jdbc.url、jdbc.username 和 jdbc.password 等属性打开数据库连接。
2. 使用 SQL 命令打开文件。如果未提供任何文件名，则在控制台中提示用户输入命令。
3. 使用通用的 execute 方法执行每条命令。如果它返回 true，则说明该命令产生了一个结果集。我们为图书数据库提供的 4 个 SQL 文件都以一个 SELECT \* 语句结束，这样就可以看到数据是否已成功插入到了数据库中。
4. 如果产生了结果集，则打印出结果。因为这是一个通用的结果集，所以我们必须使用元数据来确定该结果的列数。更多的信息请查看第 4.8 节。
5. 如果运行过程中出现 SQL 异常，则打印出这个异常以及所有可能包含在其中的与其链接在一起的相关异常。
6. 关闭数据库连接。

程序清单 4-2 给出了该程序的代码。

### 程序清单 4-2 ExecSQL.java

```
1. import java.io.*;
2. import java.util.*;
3. import java.sql.*;
4.
5. /**
6.  * Executes all SQL statements in a file. Call this program as <br>
7.  * > java -classpath driverPath:. ExecSQL commandFile
8.  * > @version 1.30 2004-08-05
9.  * > @author Cay Horstmann
10. */
11 class ExecSQL
12 {
13     public static void main(String args[])
14     {
15         try
16         {
17             Scanner in;
18             if (args.length == 0) in = new Scanner(System.in);
19             else in = new Scanner(new File(args[0]));
20
21             Connection conn = getConnection();
22             try
23             {
24                 Statement stat = conn.createStatement();
```

```
25
26     while (true)
27     {
28         if (args.length == 0) System.out.println("Enter command or EXIT to exit:");
29
30         if (!in.hasNextLine()) return;
31
32         String line = in.nextLine();
33         if (line.equalsIgnoreCase("EXIT")) return;
34         if (line.trim().endsWith(";")) // remove trailing semicolon
35         {
36             line = line.trim();
37             line = line.substring(0, line.length() - 1);
38         }
39         try
40         {
41             boolean hasResultSet = stat.execute(line);
42             if (hasResultSet) showResultSet(stat);
43         }
44         catch (SQLException ex)
45         {
46             for (Throwable e : ex)
47                 e.printStackTrace();
48         }
49     }
50     }
51     finally
52     {
53         conn.close();
54     }
55 }
56 catch (SQLException e)
57 {
58     for (Throwable t : e)
59         t.printStackTrace();
60 }
61 catch (IOException e)
62 {
63     e.printStackTrace();
64 }
65 }
66
67 /**
68 * Gets a connection from the properties specified in the file database.properties
69 * @return the database connection
70 */
71 public static Connection getConnection() throws SQLException, IOException
72 {
73     Properties props = new Properties();
74     FileInputStream in = new FileInputStream("database.properties");
75     props.load(in);
76     in.close();
77
78     String drivers = props.getProperty("jdbc.drivers");
79     if (drivers != null) System.setProperty("jdbc.drivers", drivers);
80 }
```

```
81.     String url = props.getProperty("jdbc.url");
82.     String username = props.getProperty("jdbc.username");
83.     String password = props.getProperty("jdbc.password");
84.
85.     return DriverManager.getConnection(url, username, password);
86. }
87.
88. /**
89. * Prints a result set.
90. * @param stat the statement whose result set should be printed
91. */
92. public static void showResultSet(Statement stat) throws SQLException
93. {
94.     ResultSet result = stat.getResultSet();
95.     ResultSetMetaData metaData = result.getMetaData();
96.     int columnCount = metaData.getColumnCount();
97.
98.     for (int i = 1; i <= columnCount; i++)
99.     {
100.         if (i > 1) System.out.print(", ");
101.         System.out.print(metaData.getColumnLabel(i));
102.     }
103.     System.out.println();
104.
105.     while (result.next())
106.     {
107.         for (int i = 1; i <= columnCount; i++)
108.         {
109.             if (i > 1) System.out.print(", ");
110.             System.out.print(result.getString(i));
111.         }
112.         System.out.println();
113.     }
114.     result.close();
115. }
116. }
```

## 4.5 执行查询操作

在这一节中，我们将编写一段用于对COREJAVA数据库执行查询操作的程序。为了使程序可以正常运行，必须按照上一节中的说明将表格填入COREJAVA数据库。图4-7所示为运行中的QueryDB应用程序。

可以选择作者和出版社，或者将它们设置为“Any”。点击Query按钮，文本区中将显示所有符合查询条件的图书。

还可以修改数据库中的数据。选择一家出版社，然后在Change prices按钮旁的文本框中输入数值。点击Change prices按钮后，该出版社对应的所有价格都将按照填入的数值进行调整，同时文本区中将显示被修改的行数。为了减少不必要的数据库修改，不能一次修改所有的价格。修改价格时，我们没有考虑作者字段。修改完以后，可以运行一个查询操作，以核实新的价格。



图4-7 QueryDB应用程序

#### 4.5.1 预备语句

在这个程序中，我们使用了一个新的特性，即预备语句（prepared statement）。如果不考虑作者字段，我们要查询某个出版社的所有图书，那么该查询的SQL语句如下：

```
SELECT Books.Price, Books.Title
FROM Books, Publishers
WHERE Books.Publisher_Id = Publishers.Publisher_Id
AND Publishers.Name = the name from the list box
```

我们没有必要在每次开始一个这样的查询时都建立新的查询语句，而是准备一个带有宿主变量的查询语句，每次查询时只需为该变量填入不同的字符串就可以反复多次地使用该语句。这一技术改进了查询性能，每当数据库执行一个查询时，它总是首先通过计算来确定查询策略，以便高效地执行查询操作。通过事先准备好查询并多次重用它，我们就可以确保查询所需的准备步骤只被执行一次。

在预备查询语句中，每个宿主变量都用“?”来表示。如果存在一个以上的变量，那么在设置变量值时必须注意“?”的位置。例如，

```
String publisherQuery =
    "SELECT Books.Price, Books.Title" +
    " FROM Books, Publishers" +
    " WHERE Books.Publisher_Id = Publishers.Publisher_Id AND Publishers.Name = ?";
PreparedStatement publisherQueryStat = conn.prepareStatement(publisherQuery);
```

在执行预备语句之前，必须使用set方法将变量绑定到实际的值上。和ResultSet方法中的get方法类似，针对不同的数据类型也有不同的set方法。在本例中，我们为出版社名称设置了一个字符串值。

```
publisherQueryStat.setString(1, publisher);
```

第一个参数指的是需要赋值的宿主变量的位置，位置1表示第一个“？”；第二个参数指的是赋予宿主变量的值。

如果想要重用已经执行过的预备查询语句，那么除非使用set方法或调用clearParameters

方法，否则所有宿主变量的绑定都不会改变。这就意味着，在从一个查询到另一个查询过程中，只需使用setXxx方法重新绑定那些需要改变的变量即可。

一旦为所有变量绑定了具体的值，就可以执行查询操作了：

```
ResultSet rs = publisherQueryStat.executeQuery();
```

**!** 提示：通过连接字符串来手动地构建查询显得非常枯燥乏味，而且存在潜在的危险。你必须注意像引号这样的特殊字符，而且如果查询中涉及用户的输入，那就还需要警惕注入攻击。因此，只有查询涉及变量时，你才应该使用预备语句。

价格更新操作可以由UPDATE语句实现。请注意，我们调用executeUpdate方法，而非executeQuery方法，因为UPDATE语句不返回结果集。executeUpdate的返回值为被修改过的行数。我们将这个行数显示在文本区中。

```
int r = priceUpdateStmt.executeUpdate();
result.setText(r + " rows updated");
```

**✓** 注意：在相关的Connection对象关闭之后，PreparedStatement对象也变得无效了。不过，许多数据库驱动程序通常都会自动缓存预备语句。如果要执行两次相同的查询，数据库通常会直接重用查询策略。因此，无需过多考虑调用prepareStatement的开销。

下面的列表简要说明了示例程序的结构。

- 通过执行两个查询可以得到数据库中所有的作者和出版社名称，根据这些查询结果，在文本框中填入作者和出版社。
- 监听Query按钮的监听器检查需要执行哪个类型的查询。如果该查询类型是第一次被执行，那么设置预备语句的变量值为null，同时创建一个这样的语句。然后，将变量的值绑定到查询并执行该查询。
- 涉及作者的查询比较复杂。因为一本书可能有多个作者，BooksAuthors表给出了作者和图书之间的对应关系。例如，ISBN号为0-201-96426-0的图书有两个作者，其代号为：DATE和DARW。以下为BooksAuthors表中的两行记录：

```
0-201-96426-0, DATE, 1
0-201-96426-0, DARW, 2
```

BooksAuthors表中第三列指的是作者的顺序（我们不能只看表中记录的位置，在关系表中没有固定的记录顺序）。因此，查询时需要联合Books表、BooksAuthors表和Authors表，以便和用户所选的作者名进行比较。

```
SELECT Books.Price, Books.Title FROM Books, BooksAuthors, Authors, Publishers
WHERE Authors.Author_Id = BooksAuthors.Author_Id AND BooksAuthors.ISBN = Books.ISBN
AND Books.Publisher_Id = Publishers.Publisher_Id AND Authors.Name = ? AND Publishers.Name = ?
```

**!** 提示：许多程序员都不喜欢使用如此复杂的SQL语句。比较常见的方法是使用大量的Java代码来遍历多个结果集，但是这种方法效率非常低。通常，使用数据库的查询代码要比使用Java程序好得多，这是数据库的一个重要优点。一般而言，可以使用SQL解决的问题，就不要使用Java程序。

- Change prices按钮的监听器执行UPDATE语句。注意，UPDATE语句中的WHERE子句需要使用出版社代码，而我们只知道出版社名称。这个问题可以使用嵌套子查询的方法解决。

```
UPDATE Books
SET Price = Price + ?
WHERE Books.Publisher_Id = (SELECT Publisher_Id FROM Publishers WHERE Name = ?)
```

- 我们在构造器中初始化了连接对象和语句对象，程序将在整个生命周期内一直持有这些对象。只有在程序退出之前，在我们捕获“window closing”（窗口关闭）事件的时候，才关闭上述这些对象。

程序清单4-3给出了程序的完整代码。

程序清单4-3 QueryDB.java

```
1. import java.sql.*;
2. import java.awt.*;
3. import java.awt.event.*;
4. import java.io.*;
5. import java.util.*;
6. import javax.swing.*;
7.
8. /**
9. * This program demonstrates several complex database queries.
10. * @version 1.23 2007-06-28
11. * @author Cay Horstmann
12. */
13. public class QueryDB
14. {
15.     public static void main(String[] args)
16.     {
17.         EventQueue.invokeLater(new Runnable()
18.         {
19.             public void run()
20.             {
21.                 JFrame frame = new QueryDBFrame();
22.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23.                 frame.setVisible(true);
24.             }
25.         });
26.     }
27. }
28.
29. /**
30. * This frame displays combo boxes for query parameters, a text area for command results,
31. * and buttons to launch a query and an update.
32. */
33. class QueryDBFrame extends JFrame
34. {
35.     public QueryDBFrame()
36.     {
37.         setTitle("QueryDB");
38.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
39.         setLayout(new GridBagLayout());
40.
41.         authors = new JComboBox();
```

```
42.     authors.setEditable(false);
43.     authors.addItem("Any");
44.
45.     publishers = new JComboBox();
46.     publishers.setEditable(false);
47.     publishers.addItem("Any");
48.
49.     result = new JTextArea(4, 50);
50.     result.setEditable(false);
51.
52.     priceChange = new JTextField(8);
53.     priceChange.setText("-5.00");
54.
55.     try
56.     {
57.         conn = getConnection();
58.         Statement stat = conn.createStatement();
59.         String query = "SELECT Name FROM Authors";
60.         ResultSet rs = stat.executeQuery(query);
61.         while (rs.next())
62.             authors.addItem(rs.getString(1));
63.         rs.close();
64.
65.         query = "SELECT Name FROM Publishers";
66.         rs = stat.executeQuery(query);
67.         while (rs.next())
68.             publishers.addItem(rs.getString(1));
69.         rs.close();
70.         stat.close();
71.     }
72.     catch (SQLException e)
73.     {
74.         for (Throwable t : e)
75.             result.append(t.getMessage());
76.     }
77.     catch (IOException e)
78.     {
79.         result.setText("") + e;
80.     }
81.
82.     // we use the GBC convenience class of Core Java Volume I, Chapter 9
83.     add(authors, new GBC(0, 0, 2, 1));
84.
85.     add(publishers, new GBC(2, 0, 2, 1));
86.
87.     JButton queryButton = new JButton("Query");
88.     queryButton.addActionListener(new ActionListener()
89.     {
90.         public void actionPerformed(ActionEvent event)
91.         {
92.             executeQuery();
93.         }
94.     });
95.     add(queryButton, new GBC(0, 1, 1, 1).setInsets(3));
96.
97.     JButton changeButton = new JButton("Change prices");
```

```
98.     changeButton.addActionListener(new ActionListener()
99.     {
100.         public void actionPerformed(ActionEvent event)
101.         {
102.             changePrices();
103.         }
104.     });
105.    add(changeButton, new GBC(2, 1, 1, 1).setInsets(3));
106.
107.    add(priceChange, new GBC(3, 1, 1, 1).setFill(GBC.HORIZONTAL));
108.
109.    add(new JScrollPane(result), new GBC(0, 2, 4, 1).setFill(GBC.BOTH).setWeight(100, 100));
110.
111.    addWindowListener(new WindowAdapter()
112.    {
113.        public void windowClosing(WindowEvent event)
114.        {
115.            try
116.            {
117.                if (conn != null) conn.close();
118.            }
119.            catch (SQLException e)
120.            {
121.                for (Throwable t : e)
122.                    t.printStackTrace();
123.            }
124.        }
125.    });
126. }
127.
128. /**
129. * Executes the selected query.
130. */
131. private void executeQuery()
132. {
133.     ResultSet rs = null;
134.     try
135.     {
136.         String author = (String) authors.getSelectedItem();
137.         String publisher = (String) publishers.getSelectedItem();
138.         if (!author.equals("Any") && !publisher.equals("Any"))
139.         {
140.             if (authorPublisherQueryStmt == null) authorPublisherQueryStmt = conn
141.                 .prepareStatement(authorPublisherQuery);
142.             authorPublisherQueryStmt.setString(1, author);
143.             authorPublisherQueryStmt.setString(2, publisher);
144.             rs = authorPublisherQueryStmt.executeQuery();
145.         }
146.         else if (!author.equals("Any") && publisher.equals("Any"))
147.         {
148.             if (authorQueryStmt == null) authorQueryStmt = conn.prepareStatement(authorQuery);
149.             authorQueryStmt.setString(1, author);
150.             rs = authorQueryStmt.executeQuery();
151.         }
152.         else if (author.equals("Any") && !publisher.equals("Any"))
153.         {
```

```
154.         if (publisherQueryStmt == null) publisherQueryStmt = conn
155.             .prepareStatement(publisherQuery);
156.         publisherQueryStmt.setString(1, publisher);
157.         rs = publisherQueryStmt.executeQuery();
158.     }
159.     else
160.     {
161.         if (allQueryStmt == null) allQueryStmt = conn.prepareStatement(allQuery);
162.         rs = allQueryStmt.executeQuery();
163.     }
164.
165.     result.setText("");
166.     while (rs.next())
167.     {
168.         result.append(rs.getString(1));
169.         result.append(", ");
170.         result.append(rs.getString(2));
171.         result.append("\n");
172.     }
173.     rs.close();
174. }
175. catch (SQLException e)
176. {
177.     for (Throwable t : e)
178.         result.append(t.getMessage());
179. }
180. }
181.
182. /**
183. * Executes an update statement to change prices.
184. */
185. public void changePrices()
186. {
187.     String publisher = (String) publishers.getSelectedItem();
188.     if (publisher.equals("Any"))
189.     {
190.         result.setText("I am sorry, but I cannot do that.");
191.         return;
192.     }
193.     try
194.     {
195.         if (priceUpdateStmt == null) priceUpdateStmt = conn.prepareStatement(priceUpdate);
196.         priceUpdateStmt.setString(1, priceChange.getText());
197.         priceUpdateStmt.setString(2, publisher);
198.         int r = priceUpdateStmt.executeUpdate();
199.         result.setText(r + " records updated.");
200.     }
201.     catch (SQLException e)
202.     {
203.         for (Throwable t : e)
204.             result.append(t.getMessage());
205.     }
206. }
207.
208. /**
209. * Gets a connection from the properties specified in the file database.properties
```

```
210.     * @return the database connection
211.     */
212.    public static Connection getConnection() throws SQLException, IOException
213.    {
214.        Properties props = new Properties();
215.        FileInputStream in = new FileInputStream("database.properties");
216.        props.load(in);
217.        in.close();
218.
219.        String drivers = props.getProperty("jdbc.drivers");
220.        if (drivers != null) System.setProperty("jdbc.drivers", drivers);
221.        String url = props.getProperty("jdbc.url");
222.        String username = props.getProperty("jdbc.username");
223.        String password = props.getProperty("jdbc.password");
224.
225.        return DriverManager.getConnection(url, username, password);
226.    }
227.
228.    public static final int DEFAULT_WIDTH = 400;
229.    public static final int DEFAULT_HEIGHT = 400;
230.
231.    private JComboBox authors;
232.    private JComboBox publishers;
233.    private JTextField priceChange;
234.    private JTextArea result;
235.    private Connection conn;
236.    private PreparedStatement authorQueryStmt;
237.    private PreparedStatement authorPublisherQueryStmt;
238.    private PreparedStatement publisherQueryStmt;
239.    private PreparedStatement allQueryStmt;
240.    private PreparedStatement priceUpdateStmt;
241.
242.    private static final String authorPublisherQuery = "SELECT Books.Price,
243.        Books.Title FROM Books, BooksAuthors, Authors, Publishers"
244.        + " WHERE Authors.Author_Id = BooksAuthors.Author_Id AND
245.        BooksAuthors.ISBN = Books.ISBN" + " AND Books.Publisher_Id =
246.        Publishers.Publisher_Id AND Authors.Name = ?" + " AND Publishers.Name = ?";
247.
248.    private static final String authorQuery = "SELECT Books.Price, Books.Title FROM Books,
249.        BooksAuthors, Authors" + " WHERE Authors.Author_Id =
250.        BooksAuthors.Author_Id AND BooksAuthors.ISBN = Books.ISBN"
251.        + " AND Authors.Name = ?";
252.
253.    private static final String publisherQuery = "SELECT Books.Price, Books.Title FROM Books,
254.        Publishers" + " WHERE Books.Publisher_Id = Publishers.Publisher_Id
255.        AND Publishers.Name = ?";
256.
257.    private static final String allQuery = "SELECT Books.Price, Books.Title FROM Books";
258.
259.    private static final String priceUpdate = "UPDATE Books " + "SET Price = Price + ? "
260.        + " WHERE Books.Publisher_Id = (SELECT Publisher_Id FROM Publishers WHERE Name = ?)";
261. }
```



### java.sql.Connection 1.1

- `PreparedStatement prepareStatement(String sql)`

返回一个含预编译语句的PreparedStatement对象。字符串sql代表了一个SQL语句，该语句可以包含一个或多个由?字符指明的参数占位符。

### API java.sql.PreparedStatement 1.1

- void setXxx(int n, Xxx x)  
(Xxx指int、double、String、Date之类的数据类型) 设置第n个参数值为x。
- void clearParameters()  
清除预备语句中的所有当前参数。
- ResultSet executeQuery()  
执行预备SQL查询，并返回一个ResultSet对象。
- int executeUpdate()  
执行预备SQL语句INSERT、UPDATE或DELETE，这些语句由PreparedStatement对象表示。该方法返回在执行上述语句过程中所有受影响的记录总数。如果执行的是数据定义语言(DDL)中的语句，如CREATE TABLE，则该方法返回0。

## 4.5.2 读写LOB

除了数字、字符串和日期之外，许多数据库都可以存储大对象，例如图片或其他数据。在SQL中，二进制大对象称为BLOB，字符型大对象称为CLOB。

要读取LOB，需要执行SELECT语句，然后在ResultSet上调用getBlob和getBlob方法，这样就可以获得Blob和Clob类型的对象。要从Blob中获取二进制数据，可以调用getBytes或getInputStream。例如，如果你有一张保存图书封面图像的表，那么就可以像下面这样获取一张图像：

```
PreparedStatement stat = conn.prepareStatement("SELECT Cover FROM BookCovers WHERE ISBN=?");
stat.setInt(1, isbn);
ResultSet result = stat.executeQuery();
if (result.next())
{
    Blob coverBlob = result.getBlob(1);
    Image coverImage = ImageIO.read(coverBlob.getInputStream());
}
```

类似地，如果获取了Clob对象，那么就可以通过调用getSubString或getCharacterStream来获取其中的字符数据。

要将LOB置于数据库中，需要在Connection对象上调用createBlob或createClob，然后获取一个用于该LOB的输出流或写出器，并将该对象存储到数据库中。例如，下面展示了如何存储一张图像：

```
Blob coverBlob = connection.createBlob();
int offset = 0;
OutputStream out = coverBlob.setBinaryStream(offset);
ImageIO.write(coverImage, "PNG", out);
PreparedStatement stat = conn.prepareStatement("INSERT INTO Cover VALUES (?, ?)");
stat.setInt(1, isbn);
```

```
stat.setInt(2, coverBlob);
stat.executeUpdate();
```

### **java.sql.ResultSet 1.1**

- `Blob getBlob(int columnIndex)` 1.2
- `Blob getBlob(String columnLabel)` 1.2
- `Clob getClob(int columnIndex)` 1.2
- `Clob getClob(String columnLabel)` 1.2

获取给定列的BLOB或CLOB。

### **java.sql.Blob 1.2**

- `long length()`  
获取该BLOB的长度。
- `byte[] getBytes(long startPosition, long length)`  
获取该BLOB中给定范围的数据。
- `InputStream getBinaryStream()`
- `InputStream getBinaryStream(long startPosition, long length)`  
返回一个输入流，用于读取该BLOB中全部或给定范围的数据。
- `OutputStream setBinaryStream(long startPosition)` 1.4  
返回一个输出流，用于从给定位置开始写入该BLOB。

### **java.sql.Clob 1.4**

- `long length()`  
获取该CLOB中的字符总数。
- `String getSubString(long startPosition, long length)`  
获取该CLOB中给定范围的字符。
- `Reader getCharacterStream()`
- `Reader getCharacterStream(long startPosition, long length)`  
返回一个读入器（而不是流），用于读取CLOB中全部或给定范围的数据。
- `Writer setCharacterStream(long startPosition)` 1.4  
返回一个写出器（而不是流），用于从给定位置开始写入该CLOB。

### **java.sql.Connection 1.1**

- `Blob createBlob()` 6
  - `Clob createClob()` 6
- 创建一个空的BLOB或CLOB。

#### 4.5.3 SQL转义

“转义”语法支持各种数据库普遍支持的特性，但是具有一些与数据库相关的语法变体，

因此，将转义语法转译为特定数据库的语法是JDBC驱动程序的任务之一。

转义主要用于下列特性：

- 日期和时间字面常量
- 调用标量函数
- 调用存储过程
- 外连接
- 在LIKE子句中的转义字符

日期和时间字面常量在随数据库的不同而变化很大。要嵌入日期或时间字面常量，需要按照ISO 8601格式（<http://www.cl.cam.ac.uk/~mgk25/iso-time.html>）指定它的值，之后驱动程序会将其转译为本地格式。应该使用d、t、ts来表示DATE、TIME和TIMESTAMP值：

```
{d '2008-01-24'}  
{t '23:59:59'}  
{ts '2008-01-24 23:59:59.999'}
```

标量函数（scalar function）是指仅返回一个数值的函数。在数据库中包含大量的函数，但是不同的数据库中这些函数名存在着差异。JDBC规范提供了标准的名字，并将其转译为数据库相关的名字。要调用函数，需要像下面这样嵌入标准的函数名和参数：

```
{fn left(?, 20)}  
{fn user()}
```

在JDBC规范中可以找到它支持的函数名的完整列表。

存储过程（stored procedure）是在数据库中执行的用数据库相关的语言编写的过程。要调用存储过程，需要使用call转义命令，其中在存储过程没有任何参数的时候，就不用加上括号。另外，应该用=来捕获存储过程的返回值：

```
{call PROC1(?, ?)}  
{call PROC2}  
{call ? = PROC3(?)}
```

两个表的外连接（outer join）并不要求每个表的行都要根据连接条件进行匹配，例如，假设有如下的查询：

```
SELECT * FROM {oj Books LEFT OUTER JOIN Publishers ON Books.Publisher_Id = Publisher.Publisher_Id}
```

这个查询的执行结果中将包含有Publisher\_Id在Publishers表中没有任何匹配的书，其中，Publisher\_ID为NULL值的行，就表示不存在任何匹配。如果应该使用RIGHT OUTER JOIN，就可以囊括没有任何匹配图书的出版商，而使用FULL OUTER JOIN可以同时返回这两类没有任何匹配的信息。由于并非所有的数据库对于这些连接都使用标准的写法，因此需要使用转义语法。

最后一种情况，\_和%字符在LIKE子句中具有特殊含义，用来匹配一个字符或一个字符序列。目前并不存在任何在字面上使用它们的标准方式，所以如果想要匹配所有包含\_字符的字符串，就必须使用下面的结构：

```
... WHERE ? LIKE %!_% {escape '!'}
```

这里我们将!定义为转义字符，而!\_组合表示字面常量下划线。

#### 4.5.4 多结果集

在执行存储过程，或者在使用允许在单个查询中提交多个SELECT语句的数据库时，一个查询有可能会返回多个结果集。下面是获取所有结果集的步骤：

1. 使用execute方法来执行SQL语句。
2. 获取第一个结果集或更新计数。
3. 重复调用getMoreResults方法以移动到下一个结果集（这个调用会自动关闭前一个结果集）。
4. 当不存在更多的结果集或更新计数时，完成操作。

如果由多结果集构成的链中的下一项是结果集，execute和getMoreResults方法将返回true，而如果在链中的下一项不是更新计数，getUpdateCount方法将返回-1。

下面的循环可以遍历所有的结果：

```
boolean done = false;
boolean isResult = stmt.execute(command);
while (!done)
{
    if (isResult)
    {
        ResultSet result = stmt.getResultSet();
        do something with result
    }
    else
    {
        int updateCount = stmt.getUpdateCount();
        if (updateCount >= 0)
            do something with updateCount
        else
            done = true;
    }
    isResult = stmt.getMoreResults();
}
```

#### API java.sql.Statement 1.1

- boolean getMoreResults()

获取该语句的下一个结果集，如果存在下一个结果集，并且它确实是一个结果集，则返回true。

#### 4.5.5 获取自动生成键

大多数数据库都支持某种在数据库中对行自动计数的机制。但是，不同的提供商所提供的机制之间存在着很大的差异，而这些自动计数的值经常用作主键。尽管JDBC没有提供独立于提供商的自动生成键的解决方案，但是它提供了获取自动生成键的有效途径。当我们向数据表中插入一个新行，而其键自动生成时，可以实现下面的代码来获取这个键：

```
stmt.executeUpdate(insertStatement, Statement.RETURN_GENERATED_KEYS);
ResultSet rs = stmt.getGeneratedKeys();
if (rs.next())
```

```

{
    int key = rs.getInt(1);
    ...
}

```



### java.sql.Statement 1.1

- boolean execute(String statement, int autogenerated) 1.4
- int executeUpdate(String statement, int autogenerated) 1.4

像前面描述的那样执行给定的SQL语句，如果autogenerated被设置为Statement.RETURN\_GENERATED\_KEYS，并且该语句是一条INSERT语句，那么第一列中就是自动生成的键。

## 4.6 可滚动和可更新的结果集

我们前面已经介绍过，使用ResultSet类中的next方法可以迭代遍历结果集中的所有行。对于一个只需要分析数据的程序来说，这显然已经足够了。不过，如果是用于显示一张表或查询结果的可视化数据显示（参见图4-5），我们通常会希望用户可以在结果集上前后移动。

另外，一旦向用户显示了结果集中的内容，他们就可能希望编辑这些内容。在可更新的结果集中，可以以编程方式来更新其中的项，而数据库将自动地更新。我们将在下面的小节中讨论这些功能。

### 4.6.1 可滚动的结果集

默认情况下，结果集是不可滚动和不可更新的。为了从查询中获取可滚动的结果集，必须使用以下方法得到一个不同的Statement对象：

```
Statement stat = conn.createStatement(type, concurrency);
```

如果要获得预备语句，请调用以下方法：

```
PreparedStatement stat = conn.prepareStatement(command, type, concurrency);
```

表4-6和表4-7列出了type和concurrency的所有可能值，可以有以下几种选择：

- 是否希望结果集是可滚动的？如果不需要，则使用ResultSet.TYPE\_FORWARD\_ONLY。
- 如果结果集是可滚动的，且数据库在查询生成结果集之后发生了变化，那么是否希望结果集反映出这些变化？（在我们的讨论中，我们假设将可滚动的结果集设置为ResultSet.TYPE\_SCROLL\_INSENSITIVE。这个设置将使结果集“感应”不到查询结束后出现的数据库变化。）
- 是否希望通过编辑结果集就可以更新数据库？（详细说明请参见下一节内容。）

表4-6 ResultSet类的type值

| 值                       | 解 释                |
|-------------------------|--------------------|
| TYPE_FORWARD_ONLY       | 结果集不能滚动            |
| TYPE_SCROLL_INSENSITIVE | 结果集可以滚动，但对数据库变化不敏感 |
| TYPE_SCROLL_SENSITIVE   | 结果集可以滚动，且对数据库变化敏感  |

表4-7 ResultSet类的Concurrency值

| 值                | 解 释               |
|------------------|-------------------|
| CONCUR_READ_ONLY | 结果集不能用于更新数据库（默认值） |
| CONCUR_UPDATABLE | 结果集可以用于更新数据库      |

例如，如果只想滚动遍历结果集，而不想编辑它的数据，那么可以使用以下语句：

```
Statement stat = conn.createStatement(
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
```

现在，通过调用以下方法获得的所有结果集都将是可滚动的。

```
ResultSet rs = stat.executeQuery(query)
```

可滚动的结果集有一个光标，用以指示当前位置。

 **注意：**并非所有的数据库驱动程序都支持可滚动和可更新的结果集。（使用DatabaseMetaData类中的supportsResultSetType和supportsResultSetConcurrency方法，我们可以获知在使用特定的驱动程序时，某个数据库究竟支持哪些结果集类型以及哪些并发模式。）即便是数据库支持所有的结果集模式，某个特定的查询也可能无法产生带有所请求的所有属性的结果集。（例如，一个复杂查询的结果集就有可能是不可更新的结果集。）在这种情况下，executeQuery方法将返回一个功能较少的ResultSet对象，并添加一个SQLWarning给连接对象。（参见第4.4.2节有关如果获取警告信息的内容）或者，也可以使用ResultSet类中的getType和getConcurrency方法查看结果集实际支持的模式。如果不检查结果集的功能就发起一个不支持的操作，比如对不可滚动的结果集调用previous方法，那么程序将抛出一个SQLException异常。

在结果集上的滚动是非常简单的，可以使用

```
if (rs.previous()) . . .
```

向后滚动。如果游标位于一个实际的行上，那么该方法将返回true；如果游标位于第一行之前，那么返回false。

可以使用以下调用将游标向后或向前移动多行：

```
rs.relative(n);
```

如果n为正数，光标将向前移动。如果n为负数，光标将向后移动。如果n为0，那么调用该方法将不起任何作用。如果试图将游标移动到当前行集的范围之外，那么根据n值的正负号，游标将被设置在最后一行之后或第一行之前。然后，该方法将返回false，且不移动游标。如果游标位于一个实际的行上，那么该方法将返回true。

或者，还可以将游标设置到指定的行号上：

```
rs.absolute(n);
```

调用以下方法将返回当前行的行号：

```
int currentRow = rs.getRow();
```

结果集中第一行的行号为1。如果返回值为0，那么当前光标不在任何行上，它要么位于第一行之前，要么位于最后一行之后。

`first`、`last`、`beforeFirst`和`afterLast`这些简便方法用于将光标移动到第一行、最后一行、第一行之前或最后一行之后。

最后，`isFirst`、`isLast`、`isBeforeFirst`和`isAfterLast`用于测试光标是否位于这些特殊位置上。

使用一个可滚动的结果集是非常简单的，将查询数据放入缓存中的复杂工作是由数据库驱动程序在后台完成的。

#### 4.6.2 可更新的结果集

如果希望编辑结果集中的数据，并且将结果集上的数据变更自动反映到数据库中，那么就必须使用可更新的结果集。可更新的结果集并不一定是可滚动的，但如果将数据提供给用户去编辑，那么通常也会希望结果集是可滚动的。

如果要获得可更新的结果集，应该使用以下方法创建一条语句：

```
Statement stat = conn.createStatement(  
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
```

这样，调用`executeQuery`方法返回的结果集就将是可更新的结果集。



**注意：**并非所有的查询都会返回可更新的结果集。如果查询涉及多个表格的连接操作，那么它所产生的结果集将是不可更新的。如果查询只涉及一个表格，或者在查询时是使用主键连接多个表格的，那么它所产生的结果集将是可更新的结果集。可以调用`ResultSet`类中的`getConcurrency`方法来确定结果集是否是可更新的。

例如，假设有提高某些图书的价格，但是在执行`UPDATE`命令时又没有一个简单而统一的提价标准。此时，就可以根据任意的条件，迭代遍历所有的图书并更新它们的价格。

```
String query = "SELECT * FROM Books";  
ResultSet rs = stat.executeQuery(query);  
while (rs.next())  
{  
    if (...)  
    {  
        double increase = ...;  
        double price = rs.getDouble("Price");  
        rs.updateDouble("Price", price + increase);  
        rs.updateRow(); // make sure to call updateRow after updating fields  
    }  
}
```

所有对应于SQL类型的数据类型都配有一个`updateXxx`方法，比如`updateDouble`、`updateString`等。与`getXxx`方法相同，在使用`updateXxx`方法时必须指定列的名称或序号。然后，你可以给该字段设置新的值。



**注意：**在使用第一个参数为列序号的`updateXxx`方法时，请注意这里的列序号指的是该列在结果集中的序号。它的值可以与数据库中的列序号不同。

`updateXxx`方法改变的只是结果集中的行值，而非数据库中的值。当更新完行中的字段值后，必须调用`updateRow`方法，这个方法将当前行中的所有更新信息发送给数据库。如果没有

调用`updateRow`方法就将光标移动到其他行上，那么所有的更新信息都将被行集丢弃，而且永远也不会被传递给数据库。还可以调用`cancelRowUpdates`方法来取消对当前行的更新。

我们在前面的例子中已经介绍过如何修改一个现有的行。如果想在数据库中添加一条新的记录，首先需要使用`moveToInsertRow`方法将光标移动到特定的位置，我们称之为插入行(*insert row*)。然后，调用`updateXxx`方法在插入行的位置上创建一个新的行。在上述操作全部完成之后，还需要调用`insertRow`方法将新建的行发送给数据库。完成插入操作后，再调用`moveToCurrentRow`方法将光标移回到调用`moveToInsertRow`方法之前的位置。下面是一段示例程序：

```
rs.moveToInsertRow();
rs.updateString("Title", title);
rs.updateString("ISBN", isbn);
rs.updateString("Publisher_Id", pubid);
rs.updateDouble("Price", price);
rs.insertRow();
rs.moveToCurrentRow();
```

请注意，你无法控制在结果集或数据库中添加新数据的位置。

对于在插入行中没有指定值的列，将被设置为SQL的NULL。但是，如果这个列有NOT NULL约束，那么将会抛出异常，而这一行也无法插入。

最后需要说明的是，你可以使用以下方法删除光标所指的行。

```
rs.deleteRow();
```

`deleteRow`方法会立即将该行从结果集和数据库中删除。

`ResultSet`类中的`updateRow`、`insertRow`和`deleteRow`方法的执行效果等同于SQL命令中的UPDATE、INSERT和DELETE。不过，习惯于Java编程语言的程序员通常会觉得使用结果集来操控数据库要比使用SQL语句自然得多。

 **警告：**如果不小心处理的话，就很有可能在使用可更新的结果集时编写出非常低效的代码。执行UPDATE语句，要比建立一个查询，然后一边遍历一边修改数据显得高效得多。对于用户能够任意修改数据的交互式程序来说，使用可更新的结果集是非常有意义的。但是，对大多数程序性的修改而言，使用SQL的UPDATE语句更合适一些。

 **注意：**JDBC 2对结果集做了进一步的改进，例如，如果数据被其他的并发数据库连接所修改，那么它可以用最新的数据来更新结果集。JDBC 3添加了另一种优化，可以指定结果集在事务提交时的行为。但是，这些高级特性超出了本章的范围。我们推荐你参考Maydene Fisher、Jon Ellis和Jonathan Bruce所著的《JDBC API Tutorial and Reference》(Addison-Wesley出版社2003年出版)和<http://java.sun.com/javase/technologies/database>处的JDBC规范，以了解更多的信息。

#### **API** `java.sql.Connection 1.1`

- `Statement createStatement(int type, int concurrency)` 1.2
- `PreparedStatement prepareStatement(String command, int type, int`

concurrency) 1.2

创建一个语句或预备语句，且该语句可以产生指定类型和并发模式的结果集。

参数：command

要预备的命令

type

ResultSet接口中的下述常量之一：TYPE\_FORWARD\_ONLY、  
TYPE\_SCROLL\_INSENSITIVE或者TYPE\_SCROLL\_SENSITIVE

concurrency

ResultSet接口中的下述常量之一：CONCUR\_READ\_ONLY或者  
CONCUR\_UPDATABLE



### java.sql.ResultSet 1.1

• int getType() 1.2

返回结果集的类型。返回值为以下常量之一：TYPE\_FORWARD\_ONLY、  
TYPE\_SCROLL\_INSENSITIVE或TYPE\_SCROLL\_SENSITIVE。

• int getConcurrency() 1.2

返回结果集的并发设置。返回值为以下常量之一：CONCUR\_READ\_ONLY或  
CONCUR\_UPDATABLE

• boolean previous() 1.2

将光标移动到前一行。如果光标位于某一行上，则返回true；如果游标位于第一行之前  
的位置，则返回false。

• int getRow() 1.2

得到当前行的序号。所有行从1开始编号。

• boolean absolute(int r) 1.2

移动光标到第r行。如果光标位于某一行上，则返回true。

• boolean relative(int d) 1.2

将光标移动d行。如果d为负数，则光标向后移动。如果光标位于某一行上，则返回true。

• boolean first() 1.2

• boolean last() 1.2

移动光标到第一行或最后一行。如果光标位于某一行上，则返回true。

• void beforeFirst() 1.2

• void afterLast() 1.2

移动光标到第一行之前或最后一行之后的位置。

• boolean isFirst() 1.2

• boolean isLast() 1.2

测试光标是否在第一行或最后一行。

• boolean isBeforeFirst() 1.2

• boolean isAfterLast() 1.2

测试光标是否在第一行之前或最后一行之后的位置。

• void moveToInsertRow() 1.2

移动光标到插入行。插入行是一个特殊的行，可以在该行上使用updateXxx和insertRow方法来插入新数据。

- void moveToCurrentRow() 1.2

将光标从插入行移回到调用moveToInsertRow方法之前它所在的那一行。

- void insertRow() 1.2

将插入行上的内容插入到数据库和结果集中。

- void deleteRow() 1.2

从数据库和结果集中删除当前行。

- void updateXxx(int column, Xxx data) 1.2

- void updateXxx(String columnName, Xxx data) 1.2

(Xxx指数据类型，比如int、double、String、Date等) 更新结果中当前行上的某个字段值。

- void updateRow() 1.2

将当前行的更新信息发送到数据库。

- void cancelRowUpdates() 1.2

撤销对当前行的更新。



### **java.sql.DatabaseMetaData 1.1**

- boolean supportsResultSetType(int type) 1.2

如果数据库支持给定类型的结果集，则返回true。type是ResultSet接口中的常量之一：TYPE\_SCROLL\_INSENSITIVE或者TYPE\_SCROLL\_SENSITIVE。

- boolean supportsResultSetConcurrency(int type, int concurrency) 1.2

如果数据库支持给定类型和并发模式的结果集，则返回true。

参数：type                    ResultSet接口中的下述常量之一：TYPE\_FORWARD\_ONLY、

TYPE\_SCROLL\_INSENSITIVE或者TYPE\_SCROLL\_SENSITIVE

concurrency                ResultSet接口中的下述常量之一：CONCUR\_READ\_ONLY或者

CONCUR\_UPDATABLE

## 4.7 行集

可滚动的结果集虽然功能强大，却有一个重要的缺陷：在与用户的整个交互过程中，必须始终与数据库保持连接。用户也许会离开电脑旁很长一段时间，而在此期间却始终占有着数据库连接。这种方式存在很大的问题，因为数据库连接属于稀有资源。在这种情况下，我们可以使用行集。RowSet接口继承了ResultSet接口，却无需始终保持与数据库的连接。

行集还适用于将查询结果移动到复杂应用的其他层，或者是诸如手机之类的其他设备中。你可能从未考虑过移动一个结果集，因为它的数据结构非常庞大，且依赖于数据连接。

如下所示为javax.sql.rowset包提供的接口，它们都扩展了RowSet接口：

- CachedRowSet允许在断开连接的状态下执行相关操作。关于被缓存的行集我们将在下一

节中讨论。

- **WebRowSet**对象代表了一个被缓存的行集，该行集可以保存为XML文件。该文件可以移动到Web应用的其他层中，只要在该层中使用**WebRowSet**重新打开该文件即可。
- **FilteredRowSet**和**JoinRowSet**接口支持对行集的轻量级操作，它们等同于SQL中的**SELECT**和**JOIN**操作。上述两个接口的操作对象是存储在行集中的数据，因此运行时无需建立数据库连接。
- **JdbcRowSet**是**ResultSet**接口的一个瘦包装器。它从**RowSet**中继承了**get**方法和**set**方法，从而将一个结果集转换成一个“bean”。(关于bean的更多信息请参见第8章。)

Sun公司希望数据库供应商可以高效地实现上述接口。幸运的是，Sun公司为这些接口提供了可供参考的实现，因此即便有的数据库供应商不支持这些接口也可以使用行集。所有的参考实现都放在包**com.sun.rowset**中，它们的类名都以**Impl**结尾，例如，**CachedRowSetImpl**。

### 被缓存的行集

一个被缓存的行集包含了一个结果集中所有的数据。**CachedRowSet**是**ResultSet**接口的子接口，所以你完全可以像使用结果集一样来使用被缓存的行集。被缓存的行集有一个非常重要的优点：断开数据库连接后仍然可以使用行集。你将在程序清单4-4的示例程序中看到，这种做法大大简化了交互式应用的实现。在执行每个用户命令时，我们只需打开数据库连接、执行查询操作、将查询结果放入被缓存的行集，然后关闭数据库连接即可。

我们甚至可以修改被缓存的行集中的数据。当然，这些修改不会立即反馈到数据库中。相反，必须发起一个显式的请求，以便让数据库真正接受所有修改。此时**CachedRowSet**类会重新连接到数据库，并通过执行SQL命令向数据库中写入所有修改后的数据。

可以使用一个结果集来填充**CachedRowSet**对象：

```
ResultSet result = . . .;
CachedRowSet crs = new com.sun.rowset.CachedRowSetImpl();
    // or use an implementation from your database vendor
crs.populate(result);
conn.close(); // now ok to close the database connection
```

或者，也可以让**CachedRowSet**对象自动创建一个数据库连接。首先，设置数据库参数：

```
crs.setURL("jdbc:derby://localhost:1527/COREJAVA");
crs.setUsername("dbuser");
crs.setPassword("secret");
```

然后，设置查询命令和所有参数。

```
crs.setCommand("SELECT * FROM Books WHERE PUBLISHER = ?");
crs.setString(1, publisherName);
```

最后，将查询结果填充到行集：

```
crs.execute();
```

这个方法负责建立数据库连接、执行查询操作、填充行集、最后断开连接。

如果查询结果非常大，那我们肯定不想将其全部放入行集中。毕竟，用户可能只是想浏览器中的几行而已。在这种情况下，可以指定每一页的尺寸：

```
CachedRowSet crs = . . .;
crs.setCommand(command);
crs.setPageSize(20);
. . .
crs.execute();
```

现在就只能获得20行了。要获取下一批数据，可以调用：

```
crs.nextPage();
```

可以使用与结果集中相同的命令来查看和修改行集中的数据。如果修改了行集中的内容，那么必须调用以下方法将修改写回到数据库中：

```
crs.acceptChanges(conn);
```

或

```
crs.acceptChanges();
```

只有在行集中设置了连接数据库所需的信息（如URL、用户名和密码）时，上述第二个方法调用才会有效。

在第4.6.2节中，我们曾经介绍过，并非所有的结果集都是可更新的。同样，如果一个行集对应的是复杂查询的查询结果，那么我们就无法将对行集数据的修改写回到数据库中。不过，如果行集上的数据都来自于同一张数据库表，我们就可以安全地写回数据。

**X 警告：**如果是使用结果集来填充行集，那么行集就无从获知需要更新数据的数据库表名。此时，必须调用setTable方法来设置表名称。

另外一个导致问题复杂化的情况是：在填充了行集之后，数据库中的数据发生了改变，这显然容易造成数据不一致性。为了解决这个问题，参考实现会首先检查行集中的原始值（即，修改前的值）是否与数据库中的当前值一致。如果一致，那么修改后的值将覆盖数据库中的当前值。否则，将抛出SyncProviderException异常，且不向数据库写回任何值。在实现行集接口时其他实现也可以采用不同的同步策略。

#### API javax.sql.RowSet 1.4

- `String getURL()`  
获取或设置数据库的URL。
- `void setURL(String url)`  
获取或设置连接数据库所需的用户名。
- `String getUsername()`  
获取或设置连接数据库所需的密码。
- `void setUsername(String username)`  
获取或设置连接数据库所需的密码。
- `String getPassword()`  
获取或设置向行集中填充数据时需要执行的命令。
- `void setPassword(String password)`  
获取或设置向行集中填充数据时需要执行的命令。
- `String getCommand()`  
获取或设置向行集中填充数据时需要执行的命令。
- `void setCommand(String command)`  
获取或设置向行集中填充数据时需要执行的命令。

- void execute()

通过执行使用setCommand方法设置的命令集来填充行集。为了使驱动管理器可以获得连接，必须事先设定URL、用户名和密码。



#### javax.sql.rowset.CachedRowSet 5.0

- void execute(Connection conn)

通过执行使用setCommand方法设置的命令集来填充行集。该方法使用给定的连接，并负责关闭它。

- void populate(ResultSet result)

将指定的结果集中的数据填充到被缓存的行集中。

- String getTableName()

- void setTableName(String tableName)

获取或设置数据库表名称，填充被缓存的行集时所需的数据来自于该表。

- int getPageSize()

- void setPageSize(int size)

获取和设置页的尺寸。

- boolean nextPage()

- boolean previousPage()

加载下一页或上一页，如果要加载的页存在，则返回ture。

- void acceptChanges()

- void acceptChanges(Connection conn)

重新连接数据库，并写回行集中修改过的数据。如果因为数据库中的数据已经被修改而导致无法写回行集中的数据，该方法可能会抛出SyncProviderException异常。

## 4.8 元数据

在前几节中，我们介绍了如何填写、查询和更新数据库表。JDBC可以提供关于数据库结构和表的详细信息。例如，可以获取某个数据库的所有表的列表，也可以获得某个表中所有列的名称及其数据类型。如果是在开发业务应用时使用事先定义好的数据库，那么数据库结构和表信息就不是非常有用了。毕竟，在设计数据库表时，就已经知道了它们的结构。但是，对于那些编写数据库工具的程序员来说，数据库的结构信息却是极其有用的。

在SQL中，描述数据库或其组成部分的数据称为元数据（区别于那些存在数据库中的实际数据）。我们可以获得三类元数据：关于数据库的元数据、关于结果集的元数据以及关于预备语句参数的元数据。

如果要了解数据库的更多信息，可以从数据库连接中获取一个DatabaseMetaData对象。

```
DatabaseMetaData meta = conn.getMetaData();
```

现在就可以获取某些元数据了。例如，调用

```
ResultSet mrs = meta.getTables(null, null, null, new String[] { "TABLE" });
```

将返回一个包含所有数据库表信息的结果集（如果要了解该方法的其他参数，请参见本节末尾的API说明）。

该结果集中的每一行都包含了数据库中一张表的详细信息。第三列是表的名称。（同样，如果要了解其他列的信息，请参阅API说明。）下面的循环可以获取所有的表名：

```
while (mrs.next())
    tableNames.addItem(mrs.getString(3));
```

数据库元数据还有第二个重要应用。数据库是非常复杂的，SQL标准为数据库的多样性提供了很大的空间。DatabaseMetaData类中有上百个方法可以用于查询数据库的相关信息，包括一些使用奇特的名字进行调用的方法，如：

```
meta.supportsCatalogsInPrivilegeDefinitions()
```

和

```
meta.nullPlusNonNullIsNull()
```

显然，这些方法主要是针对有特殊要求的高级用户的，尤其是那些需要编写涉及多个数据库且具有高可移植性的代码的编程人员。

DatabaseMetaData类用于提供有关数据库的数据。第二个元数据类ResultSetMetaData则用于提供结果集的相关信息。每当通过查询得到一个结果集，我们都可以获取该结果集的列数以及每一列的名称、类型和字段宽度。下面是一个典型的循环：

```
ResultSet mrs = stat.executeQuery("SELECT * FROM " + tableName);
ResultSetMetaData meta = mrs.getMetaData();
for (int i = 1; i <= meta.getColumnCount(); i++)
{
    String columnName = meta.getColumnName(i);
    int columnWidth = meta.getColumnDisplaySize(i);
    ...
}
```

在这一节中，我们将介绍如何编写一个简单的数据库工具，程序清单4-4中的程序通过使用元数据来浏览数据库中的所有表。

顶部的组合框用于显示数据库中的所有表。选中其中一个表，框中央就会显示出该表的所有字段名及其第一条记录的值，见图4-8。点击Next和Previous按钮可以滚动遍历表格中的所有记录，还可以删除一行或编辑行的值。点击Save按钮可以将各种修改保存到数据库中。

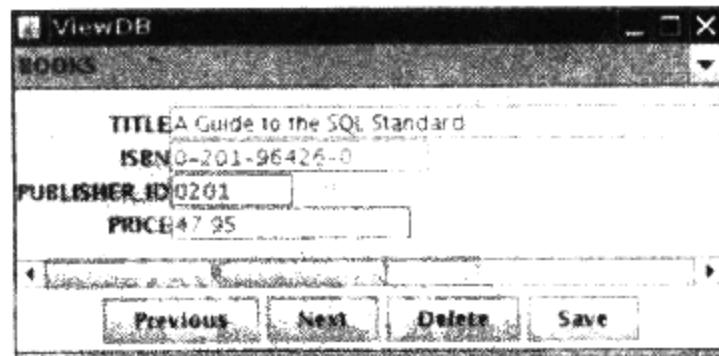


图4-8 ViewDB应用程序



**注意：**许多数据库都配有非常成熟的工具，用于查看和编辑数据库表格。如果你使用的数据库没有这样的工具，那么可以求助于iSQL-Viewer (<http://isql.sourceforge.net>) 或者 SQuirreL (<http://squirrel-sql.sourceforge.net>)。这两个工具可以查看任何JDBC数据库中的表。我们编写示例程序并非为了取代这些工具，而是为了向你演示如何编写工具用于处理任意的数据库表。

**程序清单4-4 ViewDB.java**

```
1. import com.sun.rowset.*;
2. import java.sql.*;
3. import java.awt.*;
4. import java.awt.event.*;
5. import java.io.*;
6. import java.util.*;
7. import javax.swing.*;
8. import javax.sql.*;
9. import javax.sql.rowset.*;

10.
11. /**
12. * This program uses metadata to display arbitrary tables in a database.
13. * @version 1.31 2007-06-28
14. * @author Cay Horstmann
15. */
16. public class ViewDB
17. {
18.     public static void main(String[] args)
19.     {
20.         EventQueue.invokeLater(new Runnable()
21.         {
22.             public void run()
23.             {
24.                 JFrame frame = new ViewDBFrame();
25.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
26.                 frame.setVisible(true);
27.             }
28.         });
29.     }
30. }
31.
32. /**
33. * The frame that holds the data panel and the navigation buttons.
34. */
35. class ViewDBFrame extends JFrame
36. {
37.     public ViewDBFrame()
38.     {
39.         setTitle("ViewDB");
40.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
41.
42.         tableNames = new JComboBox();
43.         tableNames.addActionListener(new ActionListener()
44.         {
45.             public void actionPerformed(ActionEvent event)
46.             {
47.                 showTable((String) tableNames.getSelectedItem());
48.             }
49.         });
50.         add(tableNames, BorderLayout.NORTH);
51.
52.         try
53.         {
54.             readDatabaseProperties();
```

```
55     Connection conn = getConnection();
56     try
57     {
58         DatabaseMetaData meta = conn.getMetaData();
59         ResultSet mrs = meta.getTables(null, null, null, new String[] { "TABLE" });
60         while (mrs.next())
61             tableNames.addItem(mrs.getString(3));
62     }
63     finally
64     {
65         conn.close();
66     }
67 }
68 catch (SQLException e)
69 {
70     JOptionPane.showMessageDialog(this, e);
71 }
72 catch (IOException e)
73 {
74     JOptionPane.showMessageDialog(this, e);
75 }
76
77 JPanel buttonPanel = new JPanel();
78 add(buttonPanel, BorderLayout.SOUTH);
79
80 previousButton = new JButton("Previous");
81 previousButton.addActionListener(new ActionListener()
82 {
83     public void actionPerformed(ActionEvent event)
84     {
85         showPreviousRow();
86     }
87 });
88 buttonPanel.add(previousButton);
89
90 nextButton = new JButton("Next");
91 nextButton.addActionListener(new ActionListener()
92 {
93     public void actionPerformed(ActionEvent event)
94     {
95         showNextRow();
96     }
97 });
98 buttonPanel.add(nextButton);
99
100 deleteButton = new JButton("Delete");
101 deleteButton.addActionListener(new ActionListener()
102 {
103     public void actionPerformed(ActionEvent event)
104     {
105         deleteRow();
106     }
107 });
108 buttonPanel.add(deleteButton);
109
110 saveButton = new JButton("Save");
```

```
111.     saveButton.addActionListener(new ActionListener()
112.     {
113.         public void actionPerformed(ActionEvent event)
114.         {
115.             saveChanges();
116.         }
117.     });
118.     buttonPanel.add(saveButton);
119. }
120.
121. /**
122. * Prepares the text fields for showing a new table, and shows the first row.
123. * @param tableName the name of the table to display
124. */
125. public void showTable(String tableName)
126. {
127.     try
128.     {
129.         // open connection
130.         Connection conn = getConnection();
131.         try
132.         {
133.             // get result set
134.             Statement stat = conn.createStatement();
135.             ResultSet result = stat.executeQuery("SELECT * FROM " + tableName);
136.             // copy into cached row set
137.             CRS = new CachedRowSetImpl();
138.             CRS.setTableName(tableName);
139.             CRS.populate(result);
140.         }
141.         finally
142.         {
143.             conn.close();
144.         }
145.
146.         if (scrollPane != null) remove(scrollPane);
147.         dataPanel = new DataPanel(CRS);
148.         scrollPane = new JScrollPane(dataPanel);
149.         add(scrollPane, BorderLayout.CENTER);
150.         validate();
151.         showNextRow();
152.     }
153.     catch (SQLException e)
154.     {
155.         JOptionPane.showMessageDialog(this, e);
156.     }
157. }
158.
159. /**
160. * Moves to the previous table row.
161. */
162. public void showPreviousRow()
163. {
164.     try
165.     {
166.         if (CRS == null || CRS.isFirst()) return;
```

```
167.         crs.previous();
168.         dataPanel.showRow(crs);
169.     }
170.     catch (SQLException e)
171.     {
172.         for (Throwable t : e)
173.             t.printStackTrace();
174.     }
175. }
176.
177. /**
178. * Moves to the next table row.
179. */
180. public void showNextRow()
181. {
182.     try
183.     {
184.         if (crs == null || crs.isLast()) return;
185.         crs.next();
186.         dataPanel.showRow(crs);
187.     }
188.     catch (SQLException e)
189.     {
190.         JOptionPane.showMessageDialog(this, e);
191.     }
192. }
193.
194. /**
195. * Deletes current table row.
196. */
197. public void deleteRow()
198. {
199.     try
200.     {
201.         Connection conn = getConnection();
202.         try
203.         {
204.             crs.deleteRow();
205.             crs.acceptChanges(conn);
206.             if (!crs.isLast()) crs.next();
207.             else if (!crs.isFirst()) crs.previous();
208.             else crs = null;
209.             dataPanel.showRow(crs);
210.         }
211.         finally
212.         {
213.             conn.close();
214.         }
215.     }
216.     catch (SQLException e)
217.     {
218.         JOptionPane.showMessageDialog(this, e);
219.     }
220. }
221.
222. /**
```

```
223.     * Saves all changes.  
224.     */  
225. public void saveChanges()  
{  
226.     try  
227.     {  
228.         Connection conn = getConnection();  
229.         try  
230.         {  
231.             dataPanel.setRow(crs);  
232.             crs.acceptChanges(conn);  
233.         }  
234.         finally  
235.         {  
236.             conn.close();  
237.         }  
238.     }  
239.     catch (SQLException e)  
240.     {  
241.         JOptionPane.showMessageDialog(this, e);  
242.     }  
243. }  
244.  
245.  
246. private void readDatabaseProperties() throws IOException  
247. {  
248.     props = new Properties();  
249.     FileInputStream in = new FileInputStream("database.properties");  
250.     props.load(in);  
251.     in.close();  
252.     String drivers = props.getProperty("jdbc.drivers");  
253.     if (drivers != null) System.setProperty("jdbc.drivers", drivers);  
254. }  
255.  
256. /**  
257. * Gets a connection from the properties specified in the file database.properties  
258. * @return the database connection  
259. */  
260. private Connection getConnection() throws SQLException  
261. {  
262.     String url = props.getProperty("jdbc.url");  
263.     String username = props.getProperty("jdbc.username");  
264.     String password = props.getProperty("jdbc.password");  
265.  
266.     return DriverManager.getConnection(url, username, password);  
267. }  
268.  
269. public static final int DEFAULT_WIDTH = 400;  
270. public static final int DEFAULT_HEIGHT = 200;  
271.  
272. private JButton previousButton;  
273. private JButton nextButton;  
274. private JButton deleteButton;  
275. private JButton saveButton;  
276. private DataPanel dataPanel;  
277. private Component scrollPane;  
278. private JComboBox tableNames;
```

```
279.     private Properties props;
280.     private CachedRowSet crs;
281. }
282.
283. /**
284. * This panel displays the contents of a result set.
285. */
286. class DataPanel extends JPanel
287. {
288.     /**
289.      * Constructs the data panel.
290.      * @param rs the result set whose contents this panel displays
291.      */
292.     public DataPanel(ResultSet rs) throws SQLException
293.     {
294.         fields = new ArrayList<JTextField>();
295.         setLayout(new GridBagLayout());
296.         GridBagConstraints gbc = new GridBagConstraints();
297.         gbc.gridwidth = 1;
298.         gbc.gridheight = 1;
299.
300.         ResultSetMetaData rsmd = rs.getMetaData();
301.         for (int i = 1; i <= rsmd.getColumnCount(); i++)
302.         {
303.             gbc.gridy = i - 1;
304.
305.             String columnName = rsmd.getColumnName(i);
306.             gbc.gridx = 0;
307.             gbc.anchor = GridBagConstraints.EAST;
308.             add(new JLabel(columnName), gbc);
309.
310.             int columnWidth = rsmd getColumnDisplaySize(i);
311.             JTextField tb = new JTextField(columnWidth);
312.             if (!rsmd.getColumnClassName(i).equals("java.lang.String"))
313.                 tb.setEditable(false);
314.
315.             fields.add(tb);
316.
317.             gbc.gridx = 1;
318.             gbc.anchor = GridBagConstraints.WEST;
319.             add(tb, gbc);
320.         }
321.     }
322.
323. /**
324. * Shows a database row by populating all text fields with the column values.
325. */
326. public void showRow(ResultSet rs) throws SQLException
327. {
328.     for (int i = 1; i <= fields.size(); i++)
329.     {
330.         String field = rs.getString(i);
331.         JTextField tb = (JTextField) fields.get(i - 1);
332.         tb.setText(field);
333.     }
334. }
```

```

336. /**
337. * Updates changed data into the current row of the row set
338. */
339. public void setRow(ResultSet rs) throws SQLException
340. {
341.     for (int i = 1; i <= fields.size(); i++)
342.     {
343.         String field = rs.getString(i);
344.         JTextField tb = (JTextField) fields.get(i - 1);
345.         if (!field.equals(tb.getText()))
346.             rs.updateString(i, tb.getText());
347.     }
348.     rs.updateRow();
349. }
350.
351. private ArrayList<JTextField> fields;
352. }

```

**API** **java.sql.Connection 1.1**

- `DatabaseMetaData getMetaData()`

返回一个`DatabaseMetaData`对象，该对象封装了有关数据库连接的元数据。

**API** **java.sql.DatabaseMetaData 1.1**

- `ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String types[])`

返回某个目录（catalog）中的所有表的描述，该目录必须符合给定的模式（schema）、表名字模式以及类型标准。（模式用于描述一组相关的表和访问权限，而目录描述的是一组相关的模式，这些概念对组织大型数据库非常重要。）

`catalog`和`schema`参数可以为空，用于检索那些没有目录和模式的表。如果不考虑目录和模式，也可以将上述参数设为`null`。

`types`数组包含了所需的表类型的名称。通常表类型有`TABLE`、`VIEW`、`SYSTEM TABLE`、`GLOBAL TEMPORARY`、`LOCAL TEMPORARY`、`ALIAS`和`SYNONYM`。如果`types`为`null`，则返回所有类型的表。

如表4-8所示，返回的结果集共有5列，均为`String`类型。

- `int getJDBCMajorVersion()1.4`
- `int getJDBCMinorVersion()1.4`

返回建立数据库连接的JDBC驱动程序的主版本号和次版本号。例如，一个JDBC 3.0的驱动程序有一个主版本号3和一个次版本号0。

- `int getMaxConnections()`

返回可同时连接到数据库的最大连接数。

- `int getMaxStatements()`

表4-8 `getTables`方法返回的结果集

| 行<br>名<br>称   | 解<br>释                        |
|---------------|-------------------------------|
| 1 TABLE_CAT   | 表目录（可以为 <code>null</code> ）   |
| 2 TABLE_SCHEM | 表结构模式（可以为 <code>null</code> ） |
| 3 TABLE_NAME  | 表名称                           |
| 4 TABLE_TYPE  | 表类型                           |
| 5 REMARKS     | 关于表的注释                        |

返回单个数据库连接允许同时打开的最大语句数。如果对允许打开的语句数目没有限制或者不可知，则返回0。

**API** **java.sql.ResultSet 1.1**

- `ResultSetMetaData getMetaData()`

返回与当前ResultSet对象中的列相关的元数据。

**API** **java.sql.ResultSetMetaData 1.1**

- `int getColumnCount()`

返回当前ResultSet对象中的列数。

- `int getColumnDisplaySize(int column)`

返回给定列序号的列的最大宽度。

参数: `column` 列序号

- `String getColumnLabel(int column)`

返回该列所建议的名称。

参数: `column` 列序号

- `String getColumnName(int column)`

返回指定的列序号所对应的列名。

参数: `column` 列序号

## 4.9 事务

我们可以将一组语句构建成一个事务 (transaction)。当所有语句都顺利执行之后，事务可以被提交 (commit)。否则，如果其中某个语句遇到错误，那么事务将被回滚，就好像没有任何命令被执行过一样。

将多个命令组合成事务的主要原因是为了确保数据库完整性 (database integrity)。例如，假设我们需要将钱从一个银行账号转移到另一个账号。此时，一个非常重要的问题就是我们必须同时将钱从一个账号取出并且存入另一个账号。如果在将钱存入其他账号之前系统发生崩溃，那么我们必须撤销取款操作。

如果将更新语句组合成一个事务，那么事务要么成功地执行所有操作并被提交，要么在中间某个位置发生失败。在这种情况下，可以执行回滚 (rollback) 操作，则数据库将自动撤销上次提交事务以来的所有更新操作产生的影响。

默认情况下，数据库连接处于自动提交模式 (autocommit mode)。每个SQL命令一旦被执行便被提交给数据库。一旦命令被提交，就无法对它进行回滚操作。在使用事务时，需要关闭这个默认值：

```
conn.setAutoCommit(false);
```

现在可以使用通常的方法创建一个语句对象：

```
Statement stat = conn.createStatement();
```

然后任意多次地调用executeUpdate方法：

```
stat.executeUpdate(command1);
stat.executeUpdate(command2);
stat.executeUpdate(command3);
...
```

执行了所有命令之后，调用commit方法：

```
conn.commit();
```

如果出现错误，请调用：

```
conn.rollback();
```

此时，程序将自动撤销自上次提交以来的所有命令。当事务被SQLException异常中断时，通常的办法是发起回滚操作。

#### 4.9.1 保存点

在使用某些驱动程序时，使用保存点（Save Point）可以更好地控制回滚操作。创建一个保存点意味着稍后只需返回到这个点，而非事务的开头。例如，

```
Statement stat = conn.createStatement(); // start transaction; rollback() goes here
stat.executeUpdate(command1);
Savepoint svpt = conn.setSavepoint(); // set savepoint; rollback(svpt) goes here
stat.executeUpdate(command2);
if (...) conn.rollback(svpt); // undo effect of command2
...
conn.commit();
```

当不再需要保存点时，必须释放它：

```
conn.releaseSavepoint(svpt);
```

#### 4.9.2 批量更新

假设有一个程序需要执行许多INSERT语句，以便将数据填入数据库表中。可以使用批量更新的方法来提高程序性能。在使用批量更新（batch update）时，一个命令序列作为一批操作将同时被收集和提交。



**注意：** 使用DatabaseMetaData类中的supportsBatchUpdates方法可以获知数据库是否支持这种特性。

处于同一批中的容器可以是INSERT、UPDATE和DELETE等操作，也可以是数据库定义命令，如CREATE TABLE和DROP TABLE。但是，在批量处理中添加SELECT命令会抛出异常（从概念上讲，批量处理中的SELECT语句没有意义，因为它会返回结果集，而并不更新数据库）。

为了执行批量处理，首先必须使用通常的办法创建一个Statement对象：

```
Statement stat = conn.createStatement();
```

现在，应该调用addBatch方法，而非executeUpdate方法：

```
String command = "CREATE TABLE . . ."
stat.addBatch(command);
```

```
while (. . .)
```

```
{  
    command = "INSERT INTO . . . VALUES (" + . . . + ")";  
    stat.addBatch(command);  
}
```

最后，提交整个批量更新语句：

```
int[] counts = stat.executeBatch();
```

调用executeBatch方法将为所有已提交的命令返回一个记录数的数组。

为了在批量模式下正确地处理错误，必须将批量执行的操作视为单个事务。如果批量更新在执行过程中失败，那么必须将它回滚到批量操作开始之前的状态。

首先，关闭自动提交模式，然后收集批量操作，执行并提交该操作，最后恢复最初的自动提交模式：

```
boolean autoCommit = conn.getAutoCommit();  
conn.setAutoCommit(false);  
Statement stat = conn.createStatement();  
  
// keep calling stat.addBatch(. . .);  
  
stat.executeBatch();  
conn.commit();  
conn.setAutoCommit(autoCommit);
```

 注意：只能在批量操作中执行更新语句。如果发起一个SELECT查询，程序将抛出异常。

#### **java.sql.Connection 1.1**

- `boolean getAutoCommit()`
- `void setAutoCommit(boolean b)`

获取将该连接中的自动提交模式，或将其设置为b。如果自动更新为true，那么所有语句将在执行结束后立刻被提交。

- `void commit()`

提交自上次提交以来所有执行过的语句。

- `void rollback()`

撤销自上次提交以来所有执行过的语句所产生的影响。

- `Savepoint setSavepoint() 1.4`
- `Savepoint setSavepoint(String name) 1.4`

设置一个匿名或具名的保存点。

- `void rollback(Savepoint svpt) 1.4`

回滚到给定保存点。

- `void releaseSavepoint(Savepoint svpt) 1.4`

释放给定的保存点。

**API** **java.sql.Savepoint 1.4**

- int getSavepointId()

获取该匿名保存点的ID号。如果该保存点具有名字，则抛出一个SQLException异常。

- String getSavepointName()

获取该保存点的名称。如果该对象为匿名保存点，则抛出一个SQLException异常。

**API** **java.sql.Statement 1.1**

- void addBatch(String command) 1.2

添加命令到当前批量命令中。

- int[] executeBatch() 1.2

执行当前批量更新中的所有命令。返回一个记录数的数组，其中每一个元素都对应一条命令，代表受该命令影响的记录总数。

**API** **java.sql.DatabaseMetaData 1.1**

- boolean supportsBatchUpdates() 1.2

如果驱动程序支持批量更新，则返回true。

表4-9 SQL数据类型及其对应的Java类型

| SQL数据类型   | Java数据类型             |
|---|----------------------|
| INTEGER或INT   | int                  |
| SMALLINT  | short                |
| NUMERIC( <i>m,n</i> ),DECIMAL<br>( <i>m,n</i> )或DEC( <i>m,n</i> ) | java.math.BigDecimal |
| FLOAT( <i>n</i> )   | double               |
| REAL  | float                |
| DOUBLE  | double               |
| CHARACTER( <i>n</i> )或CHAR( <i>n</i> )                            | String               |
| VARCHAR( <i>n</i> ),LONGVARCHAR                                   | String               |
| BOOLEAN   | boolean              |
| DATE  | java.sql.Date        |
| TIME  | java.sql.Time        |
| TIMESTAMP   | java.sql.Timestamp   |
| BLOB  | java.sql.Blob        |
| CLOB  | java.sql.Clob        |
| ARRAY   | java.sql.Array       |
| ROWID   | java.sql.RowId       |
| NCHAR( <i>n</i> ), NVARCHAR( <i>n</i> ),<br>LONG NVARCHAR         | String               |
| NCLOB   | java.sql.NClob       |
| SQLXML  | java.sql.SQLXML      |

### 4.9.3 高级SQL类型

表4-9列举了JDBC支持的SQL数据类型以及它们在Java语言中对应的数据类型。

SQL ARRAY (SQL数组) 指的是值的序列。例如，Student表中通常都会有一个Scores列，这个列就应该是ARRAY OF INTEGER (整数数组)。getArray方法返回一个类型为java.sql.Array，该接口中有许多方法可以用于获取数组的值。

从数据库中获得一个BLOB或数组并不等于获取了它的实际内容，只有在访问具体的值时它们才会从数据库中被读取出来。这对改善性能非常有好处，因为通常这些数据的数据量都非常大。

某些数据库支持描述行位置的ROWID值，这样就可以非常快捷地获取这个值。JDBC 4引入了java.sql.RowId接口，并提供了用于在查询中提供行ID，以及从结果中获取该值的方法。

具有国家属性字符串 (NCHAR及其变体)

按照本地字符编码机制存储字符串，并使用本地排序惯例对这些字符串进行排序。JDBC 4提供

了方法，用于在查询和结果中进行Java的String对象和国家属性字符串之间的双向转换。

有些数据库可以存储用户自定义的结构化类型。JDBC 3提供了一种机制用于将SQL结构化类型自动映射成Java对象。

有些数据库提供用于XML数据的本地存储。JDBC 4引入了SQLXML接口，它可以在内部XML表示和DOM的Source/Result接口或二进制流之间起到中介作用。请查看SQLXML类的API文档以了解详细信息。

我们不再更深入地讨论这些高级SQL类型了，你可以在《JDBC API Tutorial and Reference》和JDBC 4的规范中找到更多有关这些主题的信息。

## 4.10 Web与企业应用中的连接管理

我们在前面几节中曾经介绍过，使用database.properties文件可以对数据库连接进行非常简单的设置。这种方法适用于小型的测试程序，但是不适用于规模较大的应用。

在企业环境中部署JDBC应用时，数据库连接管理与Java名字和目录接口（JNDI）是集成在一起的。遍布企业的数据源的属性可以被存储在同一个目录中，采用这种方式使得我们可以集中管理用户名、密码、数据库名和JDBC URL。

在这样的环境中，可以使用下列代码创建数据库连接：

```
Context jndiContext = new InitialContext();
DataSource source = (DataSource) jndiContext.lookup("java:comp/env/jdbc/corejava");
Connection conn = source.getConnection();
```

请注意，我们不再使用`DriverManager`，而是使用JNDI服务来定位数据源。一个数据源就是一个能够提供简单的JDBC连接和更多高级服务的接口，比如执行涉及多个数据库的分布式事务。`javax.sql`标准扩展包定义了`DataSource`接口。

 **注意：**在Java EE 5的容器中，甚至不必编程进行JNDI查找，只需在`DataSource`域上使用`Resource`注解，当加载应用时，这个数据源引用将被设置：

```
@Resource("jdbc/corejava")
private DataSource source;
```

当然，我们必须在某个地方设置数据源。如果你编写的数据库程序将在Servlet容器中运行，比如Apache Tomcat，或在应用服务器中运行，比如GlassFish，那么必须将数据库配置信息（包括JNDI名字、JDBC URL、用户名和密码）放置在配置文件中，或者在管理员GUI中进行设置。

用户名管理和数据库登录只是众多需要特别关注的问题之一。另一个重要问题则涉及建立数据库连接所需的开销。我们的示例数据库程序使用了两种策略来获取数据库连接：程序清单4-3中的QueryDB程序在程序的开头建立了到数据库的单个连接，并在程序结尾处关闭它，而程序清单4-4中的ViewDB程序在每次需要时都打开一个新连接。

但是，这两种方式都不令人满意：因为数据库连接是有限的资源，如果用户在某个时刻离开了应用，那么他占用的连接就不应该保持开放状态；另一方面，每次查询都获取连接并在随后关闭它的代价也是相当高的。

解决上述问题的方法是建立数据库连接池。这意味着数据库连接在物理上并未被关闭，而

是保留在一个队列中并被反复重用。连接池是一种非常重要的服务，JDBC规范为实现者提供了用以实现连接池服务的手段。不过，JDK本身并未实现这项服务，数据库供应商提供的JDBC驱动程序中通常也不包含这项服务。相反，Web容器和应用服务器的开发商通常会提供连接池服务的实现。

连接池的使用对程序员来说是完全透明的，可以通过获取数据源并调用`getConnection`方法来得到连接池中的连接。使用完连接后，需要调用`close`方法。该方法并不在物理上关闭连接，而只是告诉连接池已经使用完该连接。连接池通常还会将池机制作用于预备语句上。

至此，你已经学会了JDBC的基本知识，并且已经知道如何实现简单的数据库应用。然而，正如我们在本章的开头所强调的那样，数据库的相关技术非常复杂；本章属于介绍性章节，相当多的高级话题已经超出了本章的范围。如果要全面了解JDBC的高级功能，请参阅《JDBC API Tutorial and Reference》或JDBC规范。

## 4.11 LDAP介绍

在前几节中，我们已经介绍了如何与关系数据库进行交互。本节将简要介绍一下如何使用轻量级目录访问协议（Lightweight Directory Access Protocol，LDAP）的分层数据库。本节主要改编自Geary和Horstmann合著的《Core JavaServer Faces》第2版（2007年由Prentice Hall出版社出版）。

如果应用的数据使用树状结构，且读操作远远多于写操作，那么此时应首选层次型数据库而非关系型数据库。LDAP通常主要应用于目录存储，且该目录包含了诸如用户名、密码和权限之类的数据。



**注意：**如果要深入了解LDAP，我们建议你参阅“LDAP圣经”：由Timothy Howes等人所著的《Understanding and Deploying LDAP Directory Services》（第2版）（2003年由Macmillan出版社出版）。

LDAP数据库将所有数据存储在一个树状结构中，而不是像关系型数据库那样存储在一组表中。树中的每一项都包含了以下内容：

- 0或0个以上属性（attribute）。每个属性都有一个ID和一个值。例如，属性`cn= John Q. Public`。（`cn`作为属性ID存储了“通用名”。关于常用的LDAP属性的意义，请参见表4-10。）
- 一个或一个以上的对象类（object class）。一个对象类定义了该元素上的一组属性，这些属性可能是必需的，也可能是可选的。例如，对象类`person`定义了一个必需属性`cn`和一个可选属性`telephoneNumber`。对象类虽然不同于Java类，但是它们同样都支持继承的概念。例如，`organizationalPerson`是`person`的子类，它在`person`的基础上添加了一些额外的属性。
- 一个专有名称（distinguished name）。例如，`uid=jqpublic, ou=people, dc=mycompany`，

表4-10 通用的LDAP属性

| 属性ID | 意义    |
|------|-------|
| dc   | 域构件   |
| cn   | 通用名   |
| sn   | 姓     |
| dn   | 专有名称  |
| o    | 组织    |
| ou   | 组织单元  |
| uid  | 唯一标识符 |

`dc=com`。专有名称是一组描述路径的属性序列，该路径将该条目与树的根连接在一起。也许会存在许多不同的可选路径，但是其中必须要有一个被指定为专有的。

图4-9所示是目录树的一个实例。

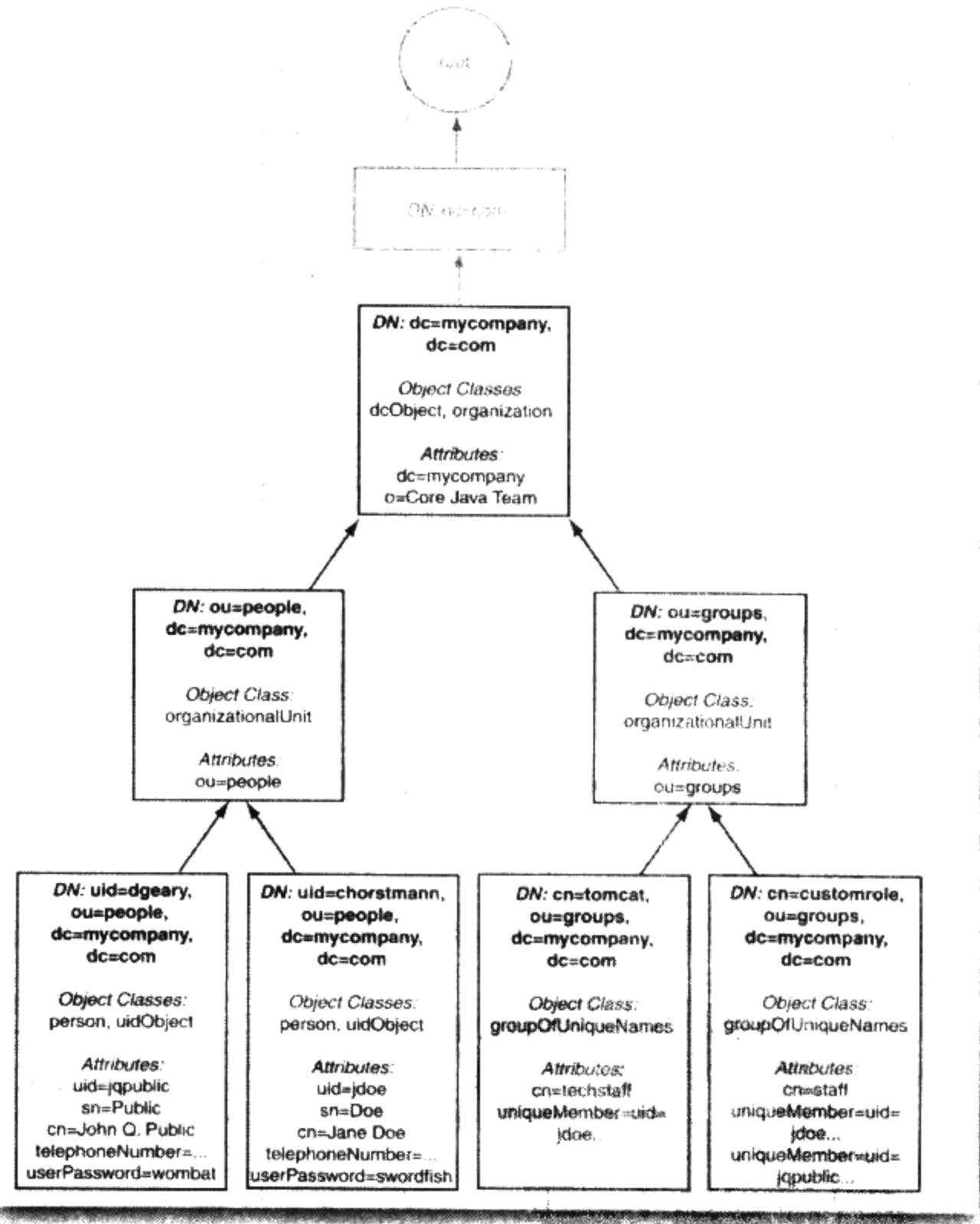


图4-9 目录树

如何组织目录树以及在目录树上应该放置哪些信息，都是非常容易引起激烈争论的问题。在此，我们并不讨论这些问题。相反，我们只简单地假设目录树的组织结构已经事先建立好了，并且包含了相关的用户数据。

#### 4.11.1 配置LDAP服务器

有数种可选的LDAP服务器可供运行本节的示例程序。以下为最常用的几种LDAP服务器：

- IBM Tivoli Directory Server
- Microsoft Active Directory
- Novell eDirectory
- OpenLDAP
- Sun Java System Directory Server for Solaris

我们将为你简要说明一下如何配置OpenLDAP服务器 (<http://openldap.org>)，这是一种可用于Linux和Windows系统，并可内建到Mac OS X系统中的免费服务器。其他目录服务器的基本配置步骤也都大同小异。

如果使用OpenLDAP服务器，那么在启动它之前必须首先编辑slapd.conf文件。（在Linux系统中，slapd.conf文件的默认路径为/etc/ldap、/etc/openldap或/usr/local/etc/openldap。）编辑slapd.conf文件中的suffix项使之与示例数据集匹配。该项指明了该服务器的专有名称，具体内容如下：

```
suffix "dc=mycompany,dc=com"
```

还需要使用管理员权限配置一个LDAP用户，使之可以编辑目录数据。如果是OpenLDAP服务器，可以在slapd.conf文件中加入以下几行内容：

```
rootdn "cn=Manager,dc=mycompany,dc=com"  
rootpw secret
```

我们推荐你指定授权设置，尽管这么做对于运行本节中的示例来说，并非严格必需的。下面在slapd.conf中的设置允许Manager用户读写密码，而其他任何人都只能读取其余的全部属性。

```
access to attr=userPassword  
  by dn.base="cn=Manager,dc=mycompany,dc=com" write  
  by self write  
  by * none  
access to *\n  by dn.base="cn=Manager,dc=mycompany,dc=com" write  
  by self write  
  by * read
```

现在可以启动LDAP服务器了。如果是在Linux系统上，请运行slapd服务（通常在/usr/sbin或/usr/local/libexec目录中）。

接下来，我们可以将示例数据填入服务器。大多数LDAP服务器都允许导入LDIF (Lightweight Directory Interchange Format，轻量级目录交换格式) 数据，LDIF是一种可供阅读的格式，它直接罗列了所有目录条目，包括它们的专有名称、对象类和属性。程序清单4-5是一个描述本节所用的示例数据的LDIF文件。

例如，对于OpenLDAP服务器，可以使用ldapadd工具将数据添加到服务器中：

```
ldapadd -f sample.ldif -x -D "cn=Manager,dc=mycompany,dc=com" -w secret
```

**程序清单4-5 sample.ldif**

```
1. # Define top-level entry
2. dn: dc=mycompany,dc=com
3. objectClass: dcObject
4. objectClass: organization
5. dc: mycompany
6. o: Core Java Team
7.
8. # Define an entry to contain people
9. # searches for users are based on this entry
10. dn: ou=people,dc=mycompany,dc=com
11. objectClass: organizationalUnit
12. ou: people
13.
14. # Define a user entry for John Q. Public
15. dn: uid=jqpublic,ou=people,dc=mycompany,dc=com
16. objectClass: person
17. objectClass: uidObject
18. uid: jqpublic
19. sn: Public
20. cn: John Q. Public
21. telephoneNumber: +1 408 555 0017
22. userPassword: wombat
23.
24. # Define a user entry for Jane Doe
25. dn: uid=jdoe,ou=people,dc=mycompany,dc=com
26. objectClass: person
27. objectClass: uidObject
28. uid: jdoe
29. sn: Doe
30. cn: Jane Doe
31. telephoneNumber: +1 408 555 0029
32. userPassword: heffalump
33.
34. # Define an entry to contain LDAP groups
35. # searches for roles are based on this entry
36. dn: ou=groups,dc=mycompany,dc=com
37. objectClass: organizationalUnit
38. ou: groups
39.
40. # Define an entry for the "techstaff" group
41. dn: cn=techstaff,ou=groups,dc=mycompany,dc=com
42. objectClass: groupOfUniqueNames
43. cn: techstaff
44. uniqueMember: uid=jdoe,ou=people,dc=mycompany,dc=com
45.
46. # Define an entry for the "staff" group
47. dn: cn=staff,ou=groups,dc=mycompany,dc=com
48. objectClass: groupOfUniqueNames
49. cn: staff
50. uniqueMember: uid=jqpublic,ou=people,dc=mycompany,dc=com
51. uniqueMember: uid=jdoe,ou=people,dc=mycompany,dc=com
```

在继续接下来的操作之前，最好再确认一次目录中是否已经包含了你所需要的数据。为此，我们建议你从网址 <http://www.jxplore.org> 处下载 JXplore 或者从网址 <http://www->

unix.mcs.anl.gov/~gawor/ldap/处下载Jarek Gawor开发的LDAP Browser\Editor。使用这些Java程序，可以方便地浏览任何LDAP服务器中的内容。启动该程序，并提供以下选项进行配置：

- Host: localhost
- Port: 389
- Base DN: dc=mycompany,dc=com
- User DN: cn=Manager,dc=mycompany,dc=com
- Password: secret

启动并连接LDAP服务器。如果一切顺利，你将看到一棵类似如图4-10所示的目录树。

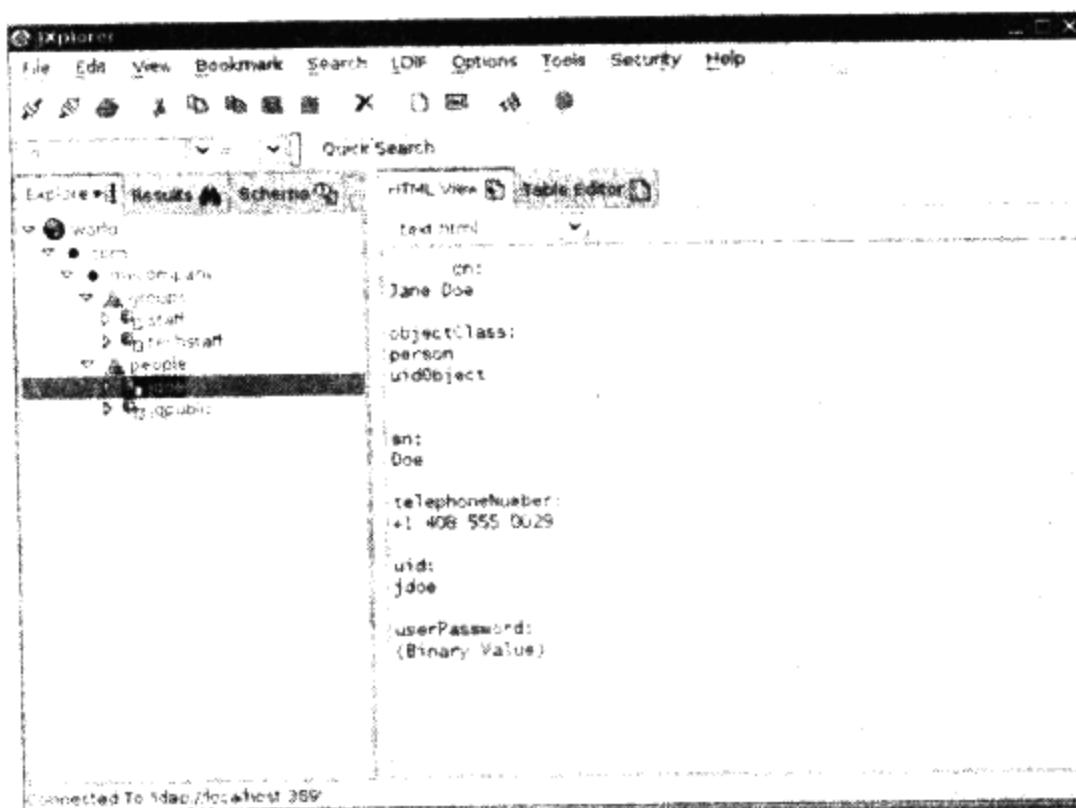


图4-10 查看一棵LDAP目录树

#### 4.11.2 访问LDAP目录信息

一旦将数据填入LDAP数据库中，就可以使用Java程序连接它了。使用以下语句可以获得LDAP目录的目录上下文（directory context）：

```
Hashtable env = new Hashtable();
env.put(Context.SECURITY_PRINCIPAL, username);
env.put(Context.SECURITY_CREDENTIALS, password);
DirContext initial = new InitialDirContext(env);
DirContext context = (DirContext) initial.lookup("ldap://localhost:389");
```

这里，我们连接的是本地的LDAP服务器。端口号389是默认的LDAP端口。

如果使用一个无效的用户名/密码连接LDAP数据库，程序将抛出一个AuthenticationException异常。



注意：Sun公司的JNDI教程建议使用另一种方法连接服务器：

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://localhost:389");
```

```
env.put(Context.SECURITY_PRINCIPAL, userDN);
env.put(Context.SECURITY_CREDENTIALS, password);
DirContext context = new InitialDirContext(env);
```

不过，没有必要在代码中套用Sun公司的LDAP提供者，JNDI提供了用于配置提供者信息的机制，你无法直接绕过这些配置。

如果要列出给定条目的所有属性，首先需要指明它的专有名称，然后调用`getAttributes`方法：

```
Attributes attrs = context.getAttributes("uid=jqpublic,ou=people,dc=mycompany,dc=com");
```

我们可以使用`get`方法获取指定的属性，例如：

```
Attribute commonNameAttribute = attrs.get("cn");
```

如果要枚举所有属性，也可以使用`NamingEnumeration`类。该类的设计者认为他们可以改进标准的Java迭代协议。他们提供了以下使用方法：

```
NamingEnumeration<? extends Attribute> attrEnum = attrs.getAll();
while (attrEnum.hasMore())
{
    Attribute attr = attrEnum.next();
    String id = attr.getID();
    ...
}
```

请注意，我们在这里使用的是`hasMore`方法，而非`hasNext`方法。

如果事先知道某个属性具有单个值，那么就可以调用`get`方法获取这个值：

```
String commonName = (String) commonNameAttribute.get();
```

如果一个属性具有多个值，那么可以使用另一个`NamingEnumeration`对象来列举所有的值：

```
NamingEnumeration<?> valueEnum = attr.getAll();
while (valueEnum.hasMore())
{
    Object value = valueEnum.next();
    ...
}
```

 **注意：**从Java SE 5.0开始，`NamingEnumeration`类已经成为通用类型。类型边界`<? extends Attribute>`指的是列举操作产生的对象属于`Attribute`的子类型。因此，无需对`next`方法返回的`value`对象进行类型转换。该对象属于`Attribute`类型。但是，`NamingEnumeration<?>`意味着它不知道所要列举的内容属于何种类型。此时，`next`方法将返回一个`Object`对象。

至此，我们已经介绍了如何查询目录以获取用户数据。接下来，我们将继续介绍如何修改目录的内容。

如果要增加新条目，可以将属性收集在一个`BasicAttributes`对象中。（`BasicAttributes`类实现了`Attributes`接口。）

```
Attributes attrs = new BasicAttributes();
attrs.put("uid", "alee");
attrs.put("sn", "Lee");
```

```
 attrs.put("cn", "Amy Lee");
 attrs.put("telephoneNumber", "+1 408 555 0033");
 String password = "woozle";
 attrs.put("userPassword", password.getBytes());
 // the following attribute has two values
 Attribute objclass = new BasicAttribute("objectClass");
 objclass.add("uidObject");
 objclass.add("person");
 attrs.put(objclass);
```

然后调用`createSubcontext`方法。向方法中传入新条目的专有名称和属性集。

```
context.createSubcontext("uid=alee,ou=people,dc=mycompany,dc=com", attrs);
```

**X 警告：**在装配属性时，请记住必须检查属性是否符合结构要求。不要提供未知的属性，并且要确保所有属性都是对象类所需要的。例如，如果你省略了`person`中的`sn`属性，那么`createSubcontext`方法将会失败。

使用`destroySubcontext`方法可以删除一个条目：

```
context.destroySubcontext("uid=alee,ou=people,dc=mycompany,dc=com");
```

最后，还可以调用以下方法编辑一个现有的条目：

```
context.modifyAttributes(distinguishedName, flag, attrs);
```

其中，`flag`参数的值为`DirContext.ADD_ATTRIBUTE`、`DirContext.REMOVE_ATTRIBUTE`和`DirContext.REPLACE_ATTRIBUTE`这几个常量之一。`attrs`参数包含了一组等待添加、删除或替换的属性集合。

使用`BasicAttributes(String, Object)`构造器可以方便地创建一个包含单个属性的属性集合。例如，

```
context.modifyAttributes("uid=alee,ou=people,dc=mycompany,dc=com",
 DirContext.ADD_ATTRIBUTE,
 new BasicAttributes("title", "CTO"));

context.modifyAttributes("uid=alee,ou=people,dc=mycompany,dc=com",
 DirContext.REMOVE_ATTRIBUTE,
 new BasicAttributes("telephoneNumber", "+1 408 555 0033"));

context.modifyAttributes("uid=alee,ou=people,dc=mycompany,dc=com",
 DirContext.REPLACE_ATTRIBUTE,
 new BasicAttributes("userPassword", password.getBytes()));
```

最后，使用完`context`对象，必须关闭它：

```
context.close();
```

程序清单4-6的程序演示了如何通过LDAP访问一个层次数据库。利用该程序，我们可以查看、修改和删除数据库中的信息。该数据库采用了程序清单4-5中的所包含的示例数据。

在文本区中输入一个`uid`，然后点击`Find`按钮查找条目。编辑条目之后点击`Save`按钮，就可以保存所有修改。通过编辑`uid`字段，可以创建一个新的条目。否则，将更新现有条目。也可以通过点击`Delete`按钮来删除条目（参见图4-11）。

以下步骤简要说明了程序的运行过程。

- `ldapserver.properties`文件包含了LDAP服务器的设置信息。该文件定义了类似如下设置的服务器URL、用户名和密码：

```
1dap.username=cn=Manager,dc=mycompany,dc=com
1dap.password=secret
1dap.url=ldap://localhost:389
```

`getContext`方法用于读取`ldapserver.properties`文件，并获得目录上下文。

- 当用户点击Find按钮时，`findEntry`方法就负责根据给定的uid获取其对应条目的属性集合。程序利用属性集合构造一个新的DataPanel对象。
- DataPanel构造器迭代遍历属性集合，并为每一对ID/值对添加一个标签和文本域。
- 当用户点击Delete按钮时，`deleteEntry`方法就负责根据给定的uid删除其对应的条目，并弃用它的数据面板。
- 当用户点击Save按钮时，DataPanel对象就创建一个BasicAttributes对象，用以存放当前文本域中的内容。`saveEntry`方法负责检查uid是否已经被修改。如果用户编辑了uid，那么该方法就创建一个新的条目。否则，将更新修改后的属性。负责修改属性的代码非常简单，因为我们只有一个属性具有多个值，即objectClass。一般情况下，在涉及每个属性时，我们都必须尽力处理它所具有的多个值。
- 与程序清单4-4中的程序类似，关闭程序窗口时，我们将同时关闭目录上下文。

至此，你已经了解了关于目录操作方面足够多的知识。现在，你就可以完成那些经常会遇到的LDAP目录方面的任务了。关于更多的高级信息可以参见网址<http://java.sun.com/products/jndi/tutorial>上提供的JNDI指南。

#### 程序清单4-6 LDAPTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.io.*;
4. import java.util.*;
5. import javax.naming.*;
6. import javax.naming.directory.*;
7. import javax.swing.*;

8.
9. /**
10. * This program demonstrates access to a hierarchical database through LDAP
11. * @version 1.01 2007-06-28
12. * @author Cay Horstmann
13. */
14. public class LDAPTest
15. {
16.     public static void main(String[] args)
17.     {
18.         EventQueue.invokeLater(new Runnable()
19.         {
20.             public void run()
21.             {
```

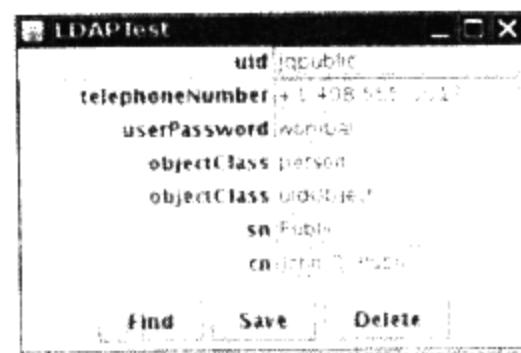


图4-11 访问一个分层数据库

```
22.         JFrame frame = new LDAPFrame();
23.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24.         frame.setVisible(true);
25.     }
26. );
27. }
28. }
29.
30 /**
31 * The frame that holds the data panel and the navigation buttons.
32 */
33 class LDAPFrame extends JFrame
34 {
35     public LDAPFrame()
36     {
37         setTitle("LDAPTest");
38         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
39
40         JPanel northPanel = new JPanel();
41         northPanel.setLayout(new java.awt.GridLayout(1, 2, 3, 1));
42         northPanel.add(new JLabel("uid", SwingConstants.RIGHT));
43         uidField = new JTextField();
44         northPanel.add(uidField);
45         add(northPanel, BorderLayout.NORTH);
46
47         JPanel buttonPanel = new JPanel();
48         add(buttonPanel, BorderLayout.SOUTH);
49
50         findButton = new JButton("Find");
51         findButton.addActionListener(new ActionListener()
52         {
53             public void actionPerformed(ActionEvent event)
54             {
55                 findEntry();
56             }
57         });
58         buttonPanel.add(findButton);
59
60         saveButton = new JButton("Save");
61         saveButton.addActionListener(new ActionListener()
62         {
63             public void actionPerformed(ActionEvent event)
64             {
65                 saveEntry();
66             }
67         });
68         buttonPanel.add(saveButton);
69
70         deleteButton = new JButton("Delete");
71         deleteButton.addActionListener(new ActionListener()
72         {
73             public void actionPerformed(ActionEvent event)
74             {
75                 deleteEntry();
76             }
77         });
78         buttonPanel.add(deleteButton);
```

```
79.
80.     addWindowListener(new WindowAdapter()
81.     {
82.         public void windowClosing(WindowEvent event)
83.         {
84.             try
85.             {
86.                 if (context != null) context.close();
87.             }
88.             catch (NamingException e)
89.             {
90.                 e.printStackTrace();
91.             }
92.         }
93.     });
94. }
95.
96. /**
97. * Finds the entry for the uid in the text field.
98. */
99. public void findEntry()
100. {
101.     try
102.     {
103.         if (scrollPane != null) remove(scrollPane);
104.         String dn = "uid=" + uidField.getText() + ",ou=people,dc=mycompany,dc=com";
105.         if (context == null) context = getContext();
106.         attrs = context.getAttributes(dn);
107.         dataPanel = new DataPanel(attrs);
108.         scrollPane = new JScrollPane(dataPanel);
109.         add(scrollPane, BorderLayout.CENTER);
110.         validate();
111.         uid = uidField.getText();
112.     }
113.     catch (NamingException e)
114.     {
115.         JOptionPane.showMessageDialog(this, e);
116.     }
117.     catch (IOException e)
118.     {
119.         JOptionPane.showMessageDialog(this, e);
120.     }
121. }
122.
123. /**
124. * Saves the changes that the user made.
125. */
126. public void saveEntry()
127. {
128.     try
129.     {
130.         if (dataPanel == null) return;
131.         if (context == null) context = getContext();
132.         if (uidField.getText().equals(uid)) // update existing entry
133.         {
134.             String dn = "uid=" + uidField.getText() + ",ou=people,dc=mycompany,dc=com";
135.             Attributes editedAttrs = dataPanel.getEditedAttributes();
```

```
136.         NamingEnumeration<? extends Attribute> attrEnum = attrs.getAll();
137.         while (attrEnum.hasMore())
138.         {
139.             Attribute attr = attrEnum.next();
140.             String id = attr.getID();
141.             Attribute editedAttr = editedAttrs.get(id);
142.             if (editedAttr != null && !attr.get().equals(editedAttr.get())) context
143.                 .modifyAttributes(dn, DirContext.REPLACE_ATTRIBUTE,
144.                     new BasicAttributes(id, editedAttr.get()));
145.         }
146.     }
147.     else
148. // create new entry
149.     {
150.         String dn = "uid=" + uidField.getText() + ",ou=people,dc=mycompany,dc=com";
151.         attrs = dataPanel.getEditedAttributes();
152.         Attribute objclass = new BasicAttribute("objectClass");
153.         objclass.add("uidObject");
154.         objclass.add("person");
155.         attrs.put(objclass);
156.         attrs.put("uid", uidField.getText());
157.         context.createSubcontext(dn, attrs);
158.     }
159.
160.     findEntry();
161. }
162. catch (NamingException e)
163. {
164.     JOptionPane.showMessageDialog(LDAPFrame.this, e);
165.     e.printStackTrace();
166. }
167. catch (IOException e)
168. {
169.     JOptionPane.showMessageDialog(LDAPFrame.this, e);
170.     e.printStackTrace();
171. }
172. }
173.
174. /**
175. * Deletes the entry for the uid in the text field.
176. */
177. public void deleteEntry()
178. {
179.     try
180.     {
181.         String dn = "uid=" + uidField.getText() + ",ou=people,dc=mycompany,dc=com";
182.         if (context == null) context = getContext();
183.         context.destroySubcontext(dn);
184.         uidField.setText("");
185.         remove(scrollPane);
186.         scrollPane = null;
187.         repaint();
188.     }
189.     catch (NamingException e)
190.     {
191.         JOptionPane.showMessageDialog(LDAPFrame.this, e);
```

```
192         e.printStackTrace();
193     }
194     catch (IOException e)
195     {
196         JOptionPane.showMessageDialog(LDAPFrame.this, e);
197         e.printStackTrace();
198     }
199 }
200
201 /**
202 * Gets a context from the properties specified in the file ldapserver.properties
203 * @return the directory context
204 */
205 public static DirContext getContext() throws NamingException, IOException
206 {
207     Properties props = new Properties();
208     FileInputStream in = new FileInputStream("ldapserver.properties");
209     props.load(in);
210     in.close();
211
212     String url = props.getProperty("ldap.url");
213     String username = props.getProperty("ldap.username");
214     String password = props.getProperty("ldap.password");
215
216     Hashtable<String, String> env = new Hashtable<String, String>();
217     env.put(Context.SECURITY_PRINCIPAL, username);
218     env.put(Context.SECURITY_CREDENTIALS, password);
219     DirContext initial = new InitialDirContext(env);
220     DirContext context = (DirContext) initial.lookup(url);
221
222     return context;
223 }
224
225 public static final int DEFAULT_WIDTH = 300;
226 public static final int DEFAULT_HEIGHT = 200;
227
228 private JButton findButton;
229 private JButton saveButton;
230 private JButton deleteButton;
231
232 private JTextField uidField;
233 private DataPanel dataPanel;
234 private Component scrollPane;
235
236 private DirContext context;
237 private String uid;
238 private Attributes attrs;
239 }
240
241 /**
242 * This panel displays the contents of a result set.
243 */
244 class DataPanel extends JPanel
245 {
246     /**
247      * Constructs the data panel.
```

```
248     * @param attributes the attributes of the given entry
249     */
250    public DataPanel(Attributes attrs) throws NamingException
251    {
252        setLayout(new java.awt.GridLayout(0, 2, 3, 1));
253
254        NamingEnumeration<? extends Attribute> attrEnum = attrs.getAll();
255        while (attrEnum.hasMore())
256        {
257            Attribute attr = attrEnum.next();
258            String id = attr.getID();
259
260            NamingEnumeration<?> valueEnum = attr.getAll();
261            while (valueEnum.hasMore())
262            {
263                Object value = valueEnum.next();
264                if (id.equals("userPassword")) value = new String((byte[]) value);
265
266                JLabel idLabel = new JLabel(id, SwingConstants.RIGHT);
267                JTextField valueField = new JTextField(" " + value);
268                if (id.equals("objectClass")) valueField.setEditable(false);
269                if (!id.equals("uid"))
270                {
271                    add(idLabel);
272                    add(valueField);
273                }
274            }
275        }
276    }
277
278    public Attributes getEditedAttributes()
279    {
280        Attributes attrs = new BasicAttributes();
281        for (int i = 0; i < getComponentCount(); i += 2)
282        {
283            JLabel idLabel = (JLabel) getComponent(i);
284            JTextField valueField = (JTextField) getComponent(i + 1);
285            String id = idLabel.getText();
286            String value = valueField.getText();
287            if (id.equals("userPassword")) attrs.put("userPassword", value.getBytes());
288            else if (!id.equals("") && !id.equals("objectClass")) attrs.put(id, value);
289        }
290        return attrs;
291    }
292 }
```



### javax.naming.directory.InitialDirContext 1.3

- InitialDirContext(Hashtable env)

使用给定的环境设置创建一个目录上下文。散列表包含了Context.SECURITY\_PRINCIPAL、Context.SECURITY\_CREDENTIALS以及其他键的相关信息。详细信息请参见javax.naming.Context接口的API文档。

**API** **javax.naming.Context 1.3**

- `Object lookup(String name)`

使用给定的名称查找对象。根据该上下文对象的属性不同，其返回值也不尽相同。返回值通常为一棵子树或一个叶对象。

- `Context createSubcontext(String name)`

使用给定的名称创建一个子上下文（subcontext）。它将成为该上下文的孩子。所有名称的路径构件，除了最后一个之外都必须存在。

- `void destroySubcontext(String name)`

根据给定的名称删除其对应的子上下文。所有名称的路径构件，除了最后一个之外都必须存在。

- `void close()`

关闭该上下文。

**API** **javax.naming.directory.DirContext 1.3**

- `Attributes getAttributes(String name)`

根据给定的名称，得到其对应条目的属性。

- `void modifyAttributes(String name, int flag, Attributes modes)`

根据给定的名称，修改其对应条目的属性。其中，`flag`值为以下常量之一：`DirContext.ADD_ATTRIBUTE`、`DirContext.REMOVE_ATTRIBUTE`或`DirContext.REPLACE_ATTRIBUTE`。

**API** **javax.naming.directory.Attributes 1.3**

- `Attribute get(String id)`

根据给定的ID，得到其对应的属性。

- `NamingEnumeration<? extends Attribute> getAll()`

返回一个枚举对象，用于迭代遍历该属性的所有值。

- `Attribute put(Attribute attr)`

- `Attribute put(String id, Object value)`

将一个属性添加到属性集合中。

**API** **javax.naming.directory.BasicAttributes 1.3**

- `BasicAttributes(String id, Object value)`

使用给定的ID和值，构造一个属性集合，该集合只包含了单个属性。

**API** **javax.naming.directory.Attribute 1.3**

- `String getID()`

获取该属性的ID。

- `Object get()`

如果值已排好序，则获取该属性的第一个值。如果未排序，则返回其中任意一个值。

- `NamingEnumeration<?> getAll()`

返回一个枚举对象，用于迭代遍历该属性的所有值。

 **javax.naming.NamingEnumeration<T>** 1.3

- `boolean hasMore()`

如果该对象还包含其他元素，则返回`true`。

- `T next()`

返回下一个元素。

在本章中，你学习了如何在Java中操作关系型数据库，并且我们还向你介绍了层次型数据库。下一章将讨论有关国际化的重要主题，并向你展示如何使你的软件对全世界的用户来说都是可用的。

# 第5章 国际化

- ▲ Locales
- ▲ 数字格式
- ▲ 日期和时间
- ▲ 排序

- ▲ 消息格式
- ▲ 文本文件和字符集
- ▲ 资源包
- ▲ 一个完整的例子

世界丰富多彩，我们希望大部分居民都能对你的软件感兴趣。一方面，因特网早已为我们打破了国家之间的界限。另一方面，如果你不去关注国际用户，你的产品的应用情况就会受到限制。

Java编程语言是第一个设计成为全面支持国际化的语言。从一开始，它就具备了进行有效的国际化所必需的一个重要特性：使用Unicode来处理所有字符串。支持Unicode使得在Java编程语言中，编写程序来操作多种语言的字符串变得异常方便。

多数程序员相信将他们的程序进行国际化需要做的所有事情就是支持Unicode并在用户接口中对消息进行翻译。但是，在本章你将会看到，国际化一个程序所要做的事情绝不仅仅是提供Unicode支持。在世界的不同地方，日期、时间、货币甚至数字的格式都不相同。你需要用一种简单的方法来为不同的语言配置菜单和按钮的名字、消息字符串和快捷键。

在本章中，我们将演示如何编写国际化的Java应用程序和Java Applet以及如何将日期、时间、数字、文本和图形用户界面本地化，还将演示Java提供的编写国际化程序的工具。最后以一个完整的例子来作为本章的结束，它是一个退休金计算器Applet，可以根据下载它的计算机的位置来改变计算结果的显示方式。



**注意：**关于国际化的更多信息，请查看网站<http://www.joconner.com/javai18n>和Sun的官方网站<http://java.sun.com/javase/technologies/core/basic/intl/>。

## 5.1 Locales

当你看到一个面向国际市场的应用软件时，它与其他软件最明显的区别就是语言。其实如果以这种外在的不同来判断是不是真正的国际化就太片面了：不同的国家可以使用相同的语言，但是为了使两个国家的用户都满意，你还有很多工作要做。<sup>⊖</sup>

不管怎样，菜单、按钮标签和程序的消息需要转换成本地语言；有时候还需要用不同的脚本来润色。这种差别很微妙；比如，数字在英语和德语中格式很不相同。对于德国用户，数字

123,456.78

---

<sup>⊖</sup> “我们现在真的是每件东西都和美国一样，当然，语言除外”，Oscar Wilde。

应该显示为

123.456,78

小数点和十进制数的逗号分隔符是相反的！在日期的显示上也有相似的变化。在美国，日期显示为月/日/年，这有些不合理。德国使用的是更合理的顺序，即日/月/年，而在中国，则使用年/月/日。因此，对于德国用户，日期

3/22/61

应该被表示为

22.03.1961

当然，如果月份的名称被显式地写了出来，那么语言之间的不同就显而易见了。英语

March 22, 1961

在德国应该被表示成

22. März 1961

在中国则是

1961年3月22日

有专门负责格式处理的类。为了对格式化进行控制，可以使用Locale类。它描述了

- 一种语言。

- 一个位置（可选）。

- 一个变量（可选）。

比如，在美国，locale对象包含

language=English, location=United States.

在德国，则是

language=German, location=Germany.

瑞士有四种官方语言（德语、法语、意大利语和里托罗曼斯语）。一个说德语的瑞士人使用的locale是：

language=German, location=Switzerland

这个locale的格式化方式和在德国使用的locale很相似；但货币值将被表示成瑞士法郎而不是德国马克。

如果只设定了语言，比如

language=German

那么locale就不能处理和国家相关的问题，比如货币。

幸运的是，变量只在处理异常的或系统相关的情况时才会被用到。比如，挪威人花了很多精力才在他们语言（丹麦语的一种派生）的拼写上达成了一致。他们使用两个拼写规则集：一种是传统的，称为Bokmål；另一种是新的，称为Nynorsk。传统的拼写方法将会以变量来表示。

language=Norwegian, location=Norway, variant=Bokmål

为了以一种简练而标准的方式来表达语言和位置，Java语言使用的是由国际标准化组织（International Organization for Standardization, ISO）所定义的编码。本地语言由小写的两个字母的代码来表示，它遵循ISO-639-1。国家代码由大写的两个字母的代码来表示，它遵循

ISO-3166-1。表5-1和5-2给出了一些常用的代码。

表5-1 常用的ISO-639-1语言代码

| 语 言        | 代 码 |
|------------|-----|
| Chinese    | zh  |
| Danish     | da  |
| Dutch      | nl  |
| English    | en  |
| French     | fr  |
| Finnish    | fi  |
| German     | de  |
| Greek      | el  |
| Italian    | it  |
| Japanese   | ja  |
| Korean     | ko  |
| Norwegian  | no  |
| Portuguese | pt  |
| Spanish    | sp  |
| Swedish    | sv  |
| Turkish    | tr  |

表5-2 常用的ISO-3166-1国家代码

| 国 家             | 代 码 |
|-----------------|-----|
| Austria         | AT  |
| Belgium         | BE  |
| Canada          | CA  |
| China           | CN  |
| Denmark         | DK  |
| Finland         | FI  |
| Germany         | DE  |
| Great Britain   | GB  |
| Greece          | GR  |
| Ireland         | IE  |
| Italy           | IT  |
| Japan           | JP  |
| Korea           | KR  |
| The Netherlands | NL  |
| Norway          | NO  |
| Portugal        | PT  |
| Spain           | ES  |
| Sweden          | SE  |
| Switzerland     | CH  |
| Taiwan          | TW  |
| Turkey          | TR  |
| United States   | US  |

 注意：ISO-639-1的完整编码可以参考[http://www.loc.gov/standards/iso639-2/php/code\\_list.php](http://www.loc.gov/standards/iso639-2/php/code_list.php)，可以在<http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/index.html>上找到ISO-3166-1的完整编码列表。

这些代码看起来有些乱，特别是其中一些是从当地的语言中来的 (German = Deutsch = de, Chinese = zhongwen = zh)，但至少它们是标准的。

为了描述locale，你要参考语言、国家代码和变量（如果有的话）并把这些字符串发送给Locale类的构造器。

```
Locale german = new Locale("de");
Locale germanGermany = new Locale("de", "DE");
Locale germanSwitzerland = new Locale("de", "CH");
Locale norwegianNorwayBokmål = new Locale("no", "NO", "B");
```

为了方便起见，Java SE预定义了大量的locale对象：

```
Locale.CANADA
Locale.CANADA_FRENCH
Locale.CHINA
Locale.FRANCE
Locale.GERMANY
Locale.ITALY
```

```
Locale.JAPAN  
Locale.KOREA  
Locale.PRC  
Locale.TAIWAN  
Locale.UK  
Locale.US
```

Java SE还预定义了大量的语言locale，它们只设定了语言而没有位置：

```
Locale.CHINESE  
Locale.ENGLISH  
Locale.FRENCH  
Locale.GERMAN  
Locale.ITALIAN  
Locale.JAPANESE  
Locale.KOREAN  
Locale.SIMPLIFIED_CHINESE  
Locale.TRADITIONAL_CHINESE
```

除了构建一个locale或使用预定义的locale外，还可以有两种方法来获得一个locale对象。

Locale类的静态getdefault方法可以获得作为本地操作系统的一部分而存放的默认locale。可以调用setDefault来改变默认的Java locale；但是，这种改变只对你的程序有效，不会对操作系统产生影响。类似地，在Applet中，getLocale方法返回用户浏览Applet时的locale对象。

最后，对于所有依赖locale的类，可以返回一个它们所支持的locale数组。比如，

```
Locale[] supportedLocales = DateFormat.getAvailableLocales();
```

将返回所有DateFormat类所能够处理的locale。

**!** 提示：为了测试，你也许希望改变你的程序的默认locale，可以在启动程序时提供语言和地域特性。比如，下面的语句将默认的locale设为German(Switzerland)：

```
java -Duser.language=de -Duser.region=CH Program
```

一旦有了一个locale，你能用它做什么呢？答案是它所能做的事情很有限。Locale类中惟一有用的是那些定义语言和国家代码的方法，其中最重要的一个getdisplayName，它返回一个描述locale的字符串。这个字符串并不包含前面所说的由两个字母组成的代码，而是以一种面向用户的形式来表现，比如

```
German (Switzerland)
```

事实上，这里有一个问题——显示的名字是以默认的locale来表示的，这可能不太恰当。如果你的用户已经选择了德语作为首选的语言，那么你可能希望将字符串显示成德语。通过给German locale一个参数就可以做到这一点：代码

```
Locale loc = new Locale("de", "CH");  
System.out.println(loc.getDisplayName(Locale.GERMAN));
```

将打印出

```
Deutsch (Schweiz)
```

这个例子说明了为什么需要Locale对象。你把它传给依赖locale的那些方法，这些方法将根据不同的地域产生不同形式的文本。在下一节中你可以见到大量的例子。

**API** **java.util.Locale 1.1**

- `Locale(String language)`
- `Locale(String language, String country)`
- `Locale(String language, String country, String variant)`  
用给定的语言、国家和变量创建一个locale。
- `static Locale getDefault()`  
返回默认的locale。
- `static voidsetDefault(Locale loc)`  
设定默认的locale。
- `String getDisplayName()`  
返回一个在当前的locale中所表示的用来描述locale的名字。
- `String getDisplayName(Locale loc)`  
返回一个在给定的locale中所表示的用来描述locale的名字。
- `String getLanguage()`  
返回语言代码，它是两个小写字母组成的ISO-639代码。
- `String getDisplayLanguage()`  
返回在当前locale中所表示的语言名称。
- `String getDisplayLanguage(Locale loc)`  
返回在给定locale中所表示的语言名称。
- `String getCountry()`  
返回国家代码，它是由两个大写字母组成的ISO-3166代码。
- `String getDisplayCountry()`  
返回在当前locale中所表示的国家名。
- `String getDisplayCountry(Locale loc)`  
返回在当前locale中所表示的国家名。
- `String getVariant()`  
返回locale中变量的字符串。
- `String getDisplayVariant()`  
返回在当前locale中所表示的变量名称。
- `String getDisplayVariant(Locale loc)`  
返回在给定locale中所表示的变量名称。
- `String toString()`  
返回locale的描述，包括语言、国家和变量，用下划线分隔（比如，“de\_CH”）。

**API** **java.awt.Applet 1.0**

- `Locale getLocale() [1.1]`

得到Applet的locale对象。

## 5.2 数字格式

我们已经提到了数字和货币的格式是高度依赖于locale的。Java类库提供了一个格式器(formatter)对象的集合，可以对java.text包中的数字值进行格式化和解析。你可以通过下面的步骤对特定locale的数字进行格式化。

1. 使用上一节的方法，得到locale对象。
2. 使用一个“工厂方法”得到一个格式器对象。
3. 使用这个格式器对象来完成格式化和解析工作。

工厂方法是NumberFormat类的一个静态方法，它接受一个Locale类型的参数。总共有3个工厂方法：getNumberInstance、getCurrencyInstance和getPercentInstance，这些方法返回的对象可以分别对数字、货币量和百分比进行格式化和解析。例如，下面显示了如何对德语中的货币值进行格式化。

```
Locale loc = new Locale("de", "DE");
NumberFormat currFmt = NumberFormat.getCurrencyInstance(loc);
double amt = 123456.78;
String result = currFmt.format(amt);
```

结果是

123.456,78€

请注意，当前符号是€，而且位于字符串的最后。同时还要注意到小数点和十进制分隔符与其他语言中的情况是相反的。

相反地，如果要想读取一个按照某个locale的规定而输入或存储的数字，那么就需要使用parse方法。比如，下面的代码解析了用户输入到文本框中的值。parse方法能够处理小数点和分隔符以及其他语言中的数字。

```
TextField inputField;
...
NumberFormat fmt = NumberFormat.getNumberInstance();
// get number formatter for default locale
Number input = fmt.parse(inputField.getText().trim());
double x = input.doubleValue();
```

parse的返回类型是抽象类型的Number。返回的对象是一个Double或Long的包装器类，这取决于被解析的数字是否是浮点数。如果两者都可以接受，可以使用Number类中的doubleValue方法来读取被包装的数字。

 **警告：**Number类型的对象并不能自动转换成相关的基本类型，因此，不能直接将一个Number对象赋给一个基本类型，而应该使用doubleValue或intValue方法。

如果数字文本的格式不正确，该方法会抛出一个ParseException异常。例如，字符串以空格开头是不允许的（可以调用trim方法来去掉空格）。但是，任何跟在数字之后的字符都将被忽略，所以这些跟在后面的字符是不会引起异常的。

请注意，由getXXXInstance工厂方法返回的类并非是NumberFormat类型的。NumberFormat类型是一个抽象类，而我们实际上得到的格式器是它的一个子类。工厂方法只知

道如何定位属于特定locale的对象。

可以用静态的getAvailableLocales方法得到一个当前所支持的locale对象列表。这个方法返回一个locale对象数组，从中可以获得针对它们的数字格式器对象。

本节的示例程序让你体会到了数字格式器的用法（参见图5-1）。图上方的组合框包含所有带数字格式器的locale，可以在数字、货币和百分率格式器之间进行选择。每次你改变选择，在文本框中的数字就会被重新格式化。在尝试了几种locale后，你就会对有这么多种方式来格式化数字和货币值而感到吃惊。也可以输入不同的数字并点击Parse按钮来调用parse方法，这个方法会试图解析你输入的内容。如果解析成功，format方法就会将结果显示出来。如果解析失败，文本框中会显示“Parse error”消息。

程序清单5-1是它的代码，显得非常直观。在构造器中，我们调用NumberFormat.getAvailableLocales。对每一个locale，我们调用getDisplayName，并把返回的结果字符串输入组合框（字符串没有被排序，在从第5.4节中，我们将深入研究排序问题）。一旦用户选择了另一个locale或点击了单选按钮，我们就创建一个新的格式器对象并更新文本框。当用户点击Parse按钮后，我们调用parse方法来基于选中的locale进行实际的解析操作。

### 程序清单5-1 NumberFormatTest.java

```

1. import java.awt.*;
2. import java.awt.event.*;
3. import java.text.*;
4. import java.util.*;
5.
6. import javax.swing.*;
7.
8. /**
9. * This program demonstrates formatting numbers under various locales.
10. * @version 1.13 2007-07-25
11. * @author Cay Horstmann
12. */
13. public class NumberFormatTest
14. {
15.     public static void main(String[] args)
16.     {
17.         EventQueue.invokeLater(new Runnable()
18.         {
19.             public void run()
20.             {
21.                 JFrame frame = new NumberFormatFrame();
22.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23.                 frame.setVisible(true);
24.             }
25.         });
26.     }
27. }
28.
29. /**

```

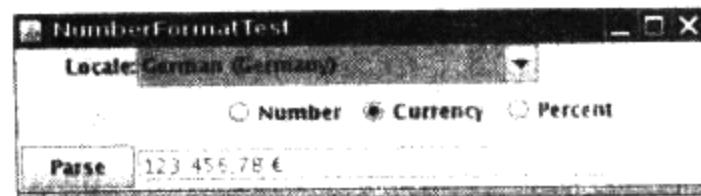


图5-1 NumberFormatTest程序

```
31. * a text field to display a formatted number, and a button to parse the text field contents.
32. */
33. class NumberFormatFrame extends JFrame
34. {
35.     public NumberFormatFrame()
36.     {
37.         setLayout(new GridBagLayout());
38.
39.         ActionListener listener = new ActionListener()
40.         {
41.             setTitle("NumberFormatTest");
42.             public void actionPerformed(ActionEvent event)
43.             {
44.                 updateDisplay();
45.             }
46.         };
47.
48.         JPanel p = new JPanel();
49.         addRadioButton(p, numberRadioButton, rbGroup, listener);
50.         addRadioButton(p, currencyRadioButton, rbGroup, listener);
51.         addRadioButton(p, percentRadioButton, rbGroup, listener);
52.
53.         add(new JLabel("Locale:"), new GBC(0, 0).setAnchor(GBC.EAST));
54.         add(p, new GBC(1, 1));
55.         add(parseButton, new GBC(0, 2).setInsets(2));
56.         add(localeCombo, new GBC(1, 0).setAnchor(GBC.WEST));
57.         add(numberText, new GBC(1, 2).setFill(GBC.HORIZONTAL));
58.         locales = (Locale[]) NumberFormat.getAvailableLocales().clone();
59.         Arrays.sort(locales, new Comparator<Locale>()
60.         {
61.             public int compare(Locale l1, Locale l2)
62.             {
63.                 return l1.getDisplayName().compareTo(l2.getDisplayName());
64.             }
65.         });
66.         for (Locale loc : locales)
67.             localeCombo.addItem(loc.getDisplayName());
68.         localeCombo.setSelectedItem(Locale.getDefault().getDisplayName());
69.         currentNumber = 123456.78;
70.         updateDisplay();
71.
72.         localeCombo.addActionListener(listener);
73.
74.         parseButton.addActionListener(new ActionListener()
75.         {
76.             public void actionPerformed(ActionEvent event)
77.             {
78.                 String s = numberText.getText().trim();
79.                 try
80.                 {
81.                     Number n = currentNumberFormat.parse(s);
82.                     if (n != null)
83.                     {
84.                         currentNumber = n.doubleValue();
85.                         updateDisplay();
86.                     }
87.                 }
88.             }
89.         });
90.     }
91. }
```

```
87.         else
88.         {
89.             numberText.setText("Parse error: " + s);
90.         }
91.     }
92.     catch (ParseException e)
93.     {
94.         numberText.setText("Parse error: " + s);
95.     }
96. }
97. });
98. pack();
99. }
100.
101. /**
102. * Adds a radio button to a container.
103. * @param p the container into which to place the button
104. * @param b the button
105. * @param g the button group
106. * @param listener the button listener
107. */
108. public void addRadioButton(Container p, JRadioButton b, ButtonGroup g,
109.                             ActionListener listener)
110. {
111.     b.setSelected(g.getButtonCount() == 0);
112.     b.addActionListener(listener);
113.     g.add(b);
114.     p.add(b);
115. }
116.
117. /**
118. * Updates the display and formats the number according to the user settings.
119. */
120. public void updateDisplay()
121. {
122.     Locale currentLocale = locales[localeCombo.getSelectedIndex()];
123.     currentNumberFormat = null;
124.     if (numberRadioButton.isSelected()) currentNumberFormat = NumberFormat
125.         .getNumberInstance(currentLocale);
126.     else if (currencyRadioButton.isSelected()) currentNumberFormat = NumberFormat
127.         .getCurrencyInstance(currentLocale);
128.     else if (percentRadioButton.isSelected()) currentNumberFormat = NumberFormat
129.         .getPercentInstance(currentLocale);
130.     String n = currentNumberFormat.format(currentNumber);
131.     numberText.setText(n);
132. }
133.
134. private Locale[] locales;
135. private double currentNumber;
136. private JComboBox localeCombo = new JComboBox();
137. private JButton parseButton = new JButton("Parse");
138. private JTextField numberText = new JTextField(30);
139. private JRadioButton numberRadioButton = new JRadioButton("Number");
140. private JRadioButton currencyRadioButton = new JRadioButton("Currency");
141. private JRadioButton percentRadioButton = new JRadioButton("Percent");
142. private ButtonGroup rbGroup = new ButtonGroup();
```

```
143.     private NumberFormat currentNumberFormat;  
144 }
```

**API** **java.text.NumberFormat 1.1**

- static Locale[] getAvailableLocales()

返回一个Locale对象的数组，其成员包含有NumberFormat 格式器。

- static NumberFormat getInstance()

- static NumberFormat getInstance(Locale l)

- static NumberFormat getCurrencyInstance()

- static NumberFormat getCurrencyInstance(Locale l)

- static NumberFormat getPercentInstance()

- static NumberFormat getPercentInstance(Locale l)

为当前的或给定的locale提供处理数字、货币值或百分比的格式器。

- String format(double x)

- String format(long x)

对给定的浮点数或整数进行格式化并以字符串的形式返回结果。

- Number parse(String s)

解析给定的字符串并返回数字值，如果输入字符串描述了一个浮点数，返回类型就是Double，否则返回类型就是Long。字符串必须以一个数字开头；以空格开头是不允许的。数字之后可以跟随其他字符，但它们都将被忽略。解析失败时抛出ParseException异常。

- void setParseIntegerOnly(boolean b)

- boolean isParseIntegerOnly()

设置或获取一个标识，该标识指示这个格式器是否应该只解析整数值。

- void setGroupingUsed(boolean b)

- boolean isGroupingUsed()

设置或获取一个标识，该标识指示这个格式器是否会添加十进制分隔符（比如，100 000）。

- void setMinimumIntegerDigits(int n)

- int getMinimumIntegerDigits()

- void setMaximumIntegerDigits(int n)

- int getMaximumIntegerDigits()

- void setMinimumFractionDigits(int n)

- int getMinimumFractionDigits()

- void setMaximumFractionDigits(int n)

- int getMaximumFractionDigits()

设置或获取整数或小数部分所允许的最大或最小位数。

## 货币

为了格式化货币值，可以使用`NumberFormat.getCurrencyInstance`方法。但是，这个方法的灵活性不好，它返回一个只针对一种货币的格式器。假设你为一个美国客户准备了一张货单，货单中有些货物的金额是用美元表示的，有些是用欧元表示的，但你不能使用两种格式器。

```
NumberFormat dollarFormatter = NumberFormat.getCurrencyInstance(Locale.US);
NumberFormat euroFormatter = NumberFormat.getCurrencyInstance(Locale.GERMANY);
```

你的货单看起来非常奇怪，有些金额的格式像\$100 000，另一些则像100.000€（注意到欧元值使用小数点而不是逗号作为分隔符）。

处理这样的情况，应该使用`Currency`类来控制被格式器所处理的货币。可以通过将一个货币标识符传给静态的`Currency.getInstance`方法来得到一个`Currency`对象，然后对每一个格式器都调用`setCurrency`方法。下面展示如何为你的美国客户设置欧元的格式：

```
NumberFormat euroFormatter = NumberFormat.getCurrencyInstance(Locale.US);
euroFormatter.setCurrency(Currency.getInstance("EUR"));
```

货币标识符由ISO 4217定义，可参考<http://www.iso.org/iso/en/prods-services/popstds/currencycodeslist.html>中的列表。表5-3提供了其中的一部分。

表5-3 货币标识符

| 货币值           | 标识符 | 货币值                     | 标识符 |
|---------------|-----|-------------------------|-----|
| U. S. Dollar  | USD | Chinese Renminbi (Yuan) | CNY |
| Euro          | EUR | Indian Rupee            | INR |
| British Pound | GBP | Russian Ruble           | RUB |
| Japanese Yen  | JPY |                         |     |

### API `java.util.Currency 1.4`

- static `Currency getInstance(String currencyCode)`
- static `Currency getInstance(Locale locale)`

返回与给定的ISO 4217货币代号匹配的`Currency`实例或返回与给定`locale`对应的国家。

- `String toString()`
- `String getCurrencyCode()`  
获取货币的ISO 4217代码。
- `String getSymbol()`
- `String getSymbol(Locale locale)`

根据默认或给定的`locale`得到货币的格式化符号。比如美元的格式化符号可能是“\$”或“US\$”，具体是哪种形式依赖于`locale`。

- `int getDefaultFractionDigits()`  
获取货币值小数点后的默认位数。

### 5.3 日期和时间

当格式化日期和时间时，需要考虑4个与locale相关的问题：

- 月份和星期应该用本地语言来表示。
- 年月日的顺序要符合本地习惯。
- 公历可能不是本地首选的日期表示方法。
- 必须要考虑本地的时区。

Java的DateFormat类可以处理这些问题。它和NumberFormat类很相似，用起来很容易。首先，得到一个locale，可以使用默认的locale或调用静态的getAvailableLocales方法来得到一个支持日期格式化的locale数组。然后，调用下面三个工厂方法之一：

```
fmt = DateFormat.getDateInstance(dateStyle, loc);
fmt = DateFormat.getTimeInstance(timeStyle, loc);
fmt = DateFormat.getDateTimeInstance(dateStyle, timeStyle, loc);
```

为了设定想要的风格，这些工厂方法有一个参数，它可以是以下某个值：

DateFormat.DEFAULT

DateFormat.FULL (例如，按照美国locale，日期格式为 Wednesday, September 12, 2007 8:51:03 PM PDT)

DateFormat.LONG (例如，按照美国locale，日期格式为September 12, 2007 8:51:03 PM PDT)

DateFormat.MEDIUM (例如，按照美国locale，日期格式为Sep 12, 2007 8:51:03 PM)

DateFormat.SHORT (例如，按照美国locale，日期格式为9/12/07 8:51 PM)

工厂方法返回一个格式化对象，你可以用它来格式化日期。

```
Date now = new Date();
String s = fmt.format(now);
```

和NumberFormat类一样，你可以使用parse方法来解析用户输入的一个日期。比如，以下代码使用默认的locale解析用户输入到文本框中的值。

```
TextField inputField;
...
DateFormat fmt = DateFormat.getDateInstance(DateFormat.MEDIUM);
Date input = fmt.parse(inputField.getText().trim());
```

但是，用户必须按预定的格式输入日期。比如，如果美国locale中的日期格式被设置为MEDIUM，那么输入的日期格式应该是：

Sep 12, 2007

如果用户输入

Sep 12 2007

(没有逗号分隔) 或用缩写形式

9/12/07

就会导致解析错误。

lenient标志用于对日期进行宽松的解释。比如，February 30, 2007会被自动转换成March 2, 2007。这似乎很危险，但是，这正是默认的行为。你也许需要关掉这个特性，这时对被解析日期进行解释的日历对象在用户输入一个不合法的日/月/年组合时将抛出IllegalArgumentException异常。

程序清单5-2展示了如何在实际中使用DateFormat类，可以选择一个locale并看看日期和时间在世界上的不同地区是如何被格式化的。如果在输出中看到乱码，那说明没有安装显示这种语言的字体。比如，如果选择中文locale，日期可能会表示成

2007年9月12日

图5-2显示了程序（已安装中文字体）。就像你看到的那样，输出能够正确显示。

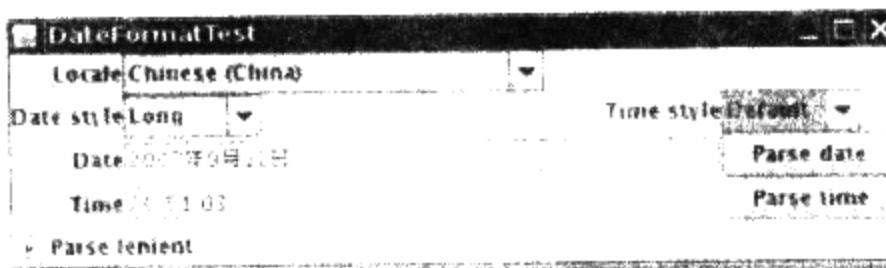


图5-2 DateFormatTest 程序

也可以对解析作试验。输入一个日期或时间，点击Parse lenient复选框，然后点击Parse date或Parse time按钮。

我们使用辅助类EnumCombo来解决一个技术问题（参见程序清单5-3）。我们想用Short、Medium和Long等值来填充一个组合框（combo），然后自动将用户的选择转换成整数值DateFormat.SHORT、DateFormat.MEDIUM和DateFormat.LONG。我们并没有编写重复的代码，而是使用了反射：我们将用户的选择转换成大写字母，所有空格都用下划线替换，然后找到使用这个名字的静态域的值。（更多关于反射的内容参见第I卷第5章。）

**提示：**为了计算不同时区的时间，需要使用TimeZone类，在<http://java.sun.com/developer/JDCTechTips/2003/tt1104.html#2>有一个使用指南。

### 程序清单5-2 DateFormatTest.java

```

1 import java.awt.*;
2 import java.awt.event.*;
3 import java.text.*;
4 import java.util.*;
5
6 import javax.swing.*;
7
8 /**
9 * This program demonstrates formatting dates under various locales.
10 * @version 1.13 2007-07-25
11 * @author Cay Horstmann
12 */
13 public class DateFormatTest
14 {
15     public static void main(String[] args)
16     {
17         EventQueue.invokeLater(new Runnable()
18         {
19             public void run()
20             {
21                 JFrame frame = new DateFormatFrame();
22                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23             }
24         });
25     }
26 }
```

```
23.         frame.setVisible(true);
24.     }
25.   });
26. }
27. */
28. /**
29. * This frame contains combo boxes to pick a locale, date and time formats, text fields
30. * to display formatted date and time, buttons to parse the text field contents, and a
31. * "lenient" checkbox.
32. */
33. class DateFormatFrame extends JFrame
34. {
35.     public DateFormatFrame()
36.     {
37.         setTitle("DateFormatTest");
38.
39.         setLayout(new GridBagLayout());
40.         add(new JLabel("Locale"), new GBC(0, 0).setAnchor(GBC.EAST));
41.         add(new JLabel("Date style"), new GBC(0, 1).setAnchor(GBC.EAST));
42.         add(new JLabel("Time style"), new GBC(2, 1).setAnchor(GBC.EAST));
43.         add(new JLabel("Date"), new GBC(0, 2).setAnchor(GBC.EAST));
44.         add(new JLabel("Time"), new GBC(0, 3).setAnchor(GBC.EAST));
45.         add(localeCombo, new GBC(1, 0, 2, 1).setAnchor(GBC.WEST));
46.         add(dateStyleCombo, new GBC(1, 1).setAnchor(GBC.WEST));
47.         add(timeStyleCombo, new GBC(3, 1).setAnchor(GBC.WEST));
48.         add(dateParseButton, new GBC(3, 2).setAnchor(GBC.WEST));
49.         add(timeParseButton, new GBC(3, 3).setAnchor(GBC.WEST));
50.         add(lenientCheckbox, new GBC(0, 4, 2, 1).setAnchor(GBC.WEST));
51.         add(dateText, new GBC(1, 2, 2, 1).setFill(GBC.HORIZONTAL));
52.         add(timeText, new GBC(1, 3, 2, 1).setFill(GBC.HORIZONTAL));
53.
54.
55.         locales = (Locale[]) DateFormat.getAvailableLocales().clone();
56.         Arrays.sort(locales, new Comparator<Locale>()
57.         {
58.             public int compare(Locale l1, Locale l2)
59.             {
60.                 return l1.getDisplayName().compareTo(l2.getDisplayName());
61.             }
62.         });
63.         for (Locale loc : locales)
64.             localeCombo.addItem(loc.getDisplayName());
65.         localeCombo.setSelectedItem(Locale.getDefault().getDisplayName());
66.         currentDate = new Date();
67.         currentTime = new Date();
68.         updateDisplay();
69.
70.         ActionListener listener = new ActionListener()
71.         {
72.             public void actionPerformed(ActionEvent event)
73.             {
74.                 updateDisplay();
75.             }
76.         };
77.
78.         localeCombo.addActionListener(listener);
79.         dateStyleCombo.addActionListener(listener);
```

```
80.     timeStyleCombo.addActionListener(listener);
81.
82.     dateParseButton.addActionListener(new ActionListener()
83.     {
84.         public void actionPerformed(ActionEvent event)
85.         {
86.             String d = dateText.getText().trim();
87.             try
88.             {
89.                 currentDateFormat.setLenient(lenientCheckbox.isSelected());
90.                 Date date = currentDateFormat.parse(d);
91.                 currentDate = date;
92.                 updateDisplay();
93.             }
94.             catch (ParseException e)
95.             {
96.                 dateText.setText("Parse error: " + d);
97.             }
98.             catch (IllegalArgumentException e)
99.             {
100.                 dateText.setText("Argument error: " + d);
101.             }
102.         }
103.     });
104.
105.    timeParseButton.addActionListener(new ActionListener()
106.    {
107.        public void actionPerformed(ActionEvent event)
108.        {
109.            String t = timeText.getText().trim();
110.            try
111.            {
112.                currentDateFormat.setLenient(lenientCheckbox.isSelected());
113.                Date date = currentTimeFormat.parse(t);
114.                currentTime = date;
115.                updateDisplay();
116.            }
117.            catch (ParseException e)
118.            {
119.                timeText.setText("Parse error: " + t);
120.            }
121.            catch (IllegalArgumentException e)
122.            {
123.                timeText.setText("Argument error: " + t);
124.            }
125.        }
126.    });
127.    pack();
128. }
129.
130. /**
131. * Updates the display and formats the date according to the user settings.
132. */
133. public void updateDisplay()
134. {
135.     Locale currentLocale = locales[localeCombo.getSelectedIndex()];
```

```
136.     int dateStyle = dateStyleCombo.getValue();
137.     currentDateFormat = DateFormat.getDateInstance(dateStyle, currentLocale);
138.     String d = currentDateFormat.format(currentDate);
139.     dateText.setText(d);
140.     int timeStyle = timeStyleCombo.getValue();
141.     currentTimeFormat = DateFormat.getTimeInstance(timeStyle, currentLocale);
142.     String t = currentTimeFormat.format(currentTime);
143.     timeText.setText(t);
144. }
145.
146. private Locale[] locales;
147. private Date currentDate;
148. private Date currentTime;
149. private DateFormat currentDateFormat;
150. private DateFormat currentTimeFormat;
151. private JComboBox localeCombo = new JComboBox();
152. private EnumCombo dateStyleCombo = new EnumCombo(DateFormat.class, new String[] { "Default",
153.     "Full", "Long", "Medium", "Short" });
154. private EnumCombo timeStyleCombo = new EnumCombo(DateFormat.class, new String[] { "Default",
155.     "Full", "Long", "Medium", "Short" });
156. private JButton dateParseButton = new JButton("Parse date");
157. private JButton timeParseButton = new JButton("Parse time");
158. private JTextField dateText = new JTextField(30);
159. private JTextField timeText = new JTextField(30);
160. private JCheckBox lenientCheckbox = new JCheckBox("Parse lenient", true);
161. }
```

### 程序清单5-3    EnumCombo.java

```
1. import java.util.*;
2. import javax.swing.*;
3.
4. /**
5.  * A combo box that lets users choose from among static field
6.  * values whose names are given in the constructor.
7.  * @version 1.13 2007-07-25
8.  * @author Cay Horstmann
9. */
10. public class EnumCombo extends JComboBox
11. {
12.     /**
13.      Constructs an EnumCombo.
14.      @param cl a class
15.      @param labels an array of static field names of cl
16.     */
17.     public EnumCombo(Class<?> cl, String[] labels)
18.     {
19.         for (String label : labels)
20.         {
21.             String name = label.toUpperCase().replace(' ', '_');
22.             int value = 0;
23.             try
24.             {
25.                 java.lang.reflect.Field f = cl.getField(name);
26.                 value = f.getInt(cl);
27.             }
```

```

28.         catch (Exception e)
29.     {
30.         label = "(" + label + ")";
31.     }
32.     table.put(label, value);
33.     addItem(label);
34.   }
35.   setSelectedItem(labels[0]);
36. }
37.
38. /**
39.  * Returns the value of the field that the user selected.
40.  * @return the static field value
41. */
42. public int getValue()
43. {
44.     return table.get(getSelectedItem());
45. }
46.
47. private Map<String, Integer> table = new TreeMap<String, Integer>();
48. }

```

### **API** java.text.DateFormat 1.1

- static Locale[] getAvailableLocales()

返回一个Locale对象的数组，可以对这些对象使用DateFormat格式器。

- static DateFormat getDateInstance(int dateStyle)

- static DateFormat getDateInstance(int dateStyle, Locale l)

- static DateFormat getTimeInstance(int timeStyle)

- static DateFormat getTimeInstance(int timeStyle, Locale l)

- static DateFormat getDateTimeInstance(int dateStyle, int timeStyle)

- static DateFormat getDateTimeInstance(int dateStyle, int timeStyle, Locale l)

为默认或指定的locale返回一个日期格式器、时间格式器或日期和时间的格式器。

参数：dateStyle, timeStyle DEFAULT、FULL、LONG、MEDIUM、SHORT之一

- String format(Date d)

对给定的日期/时间进行格式化，并返回结果。

- Date parse(String s)

解析给定的字符串，返回其中描述的日期/时间。字符串必须以日期或时间开头；以空格开头是不允许的。日期后可以跟其他字符，但它们会被忽略。如果解析没有成功，则抛出ParseException异常。

- void setLenient(boolean b)

- boolean isLenient()

设置或获得一个标志，它标记解析时是采用宽松的还是严格的方式。在宽松模式下，像 February 30, 1999 这样的日期会被自动转换成 March 2, 1999。它是默认的模式。

- void setCalendar(Calendar cal)

- `Calendar getCalendar()`

设置或获得Calendar对象，可以用它来从Date对象中获取年、月、日、小时、分钟和秒。如果不使用该locale的默认日历（一般是公历），就使用这个方法。

- `void setTimeZone(TimeZone tz)`

- `TimeZone getTimeZone()`

设置或获得格式化时间的时区对象。如果不让locale使用默认时区，可以使用这个方法。默认时区是默认locale的时区，可以从操作系统中得到它。对于其他locale，应该使用和地理位置相对应的时区。

- `void setNumberFormat(NumberFormat f)`

- `NumberFormat getNumberFormat()`

设置或获得数字格式，用它来格式化那些表示年、月、日、小时、分、秒的时间。



### `java.util.TimeZone 1.1`

- `static String[] getAvailableIDs()`

获取所有支持的时区ID。

- `static TimeZone getDefault()`

获取计算机的默认时区。

- `static TimeZone getTimeZone(String timeZoneId)`

获取给定ID的时区。

- `StringgetID()`

获取时区的ID。

- `String getDisplayName()`

- `String getDisplayName(Locale locale)`

- `String getDisplayName(boolean daylight, int style)`

- `String getDisplayName(boolean daylight, int style, Locale locale)`

获取一个时区在默认或给定的locale中的显示名称。如果daylight参数是true，则返回夏令时的名字。参数Style的值可以是SHORT或LONG。

- `boolean useDaylightTime()`

如果时区使用夏令时，则返回true。

- `boolean inDaylightTime(Date date)`

如果给定的日期在该时区中是夏令时，则返回true。

## 5.4 排序

大多数程序员都知道如何使用String类中的`compareTo`方法对字符串进行比较。如果a的字典序小于b，则`a.compareTo(b)`返回一个负数；如果相同则返回0，否则返回正值。

但是，除非你的所有字母都是大写的英语ASCII字母，否则这个方法是没有用处的。它的问题在于Java语言中的`compareTo`方法是用Unicode字符来决定顺序的。比如，小写字母的

Unicode值比大写的大，有重音符的字母的值甚至更大，这将使结果失去意义。比如，我们使用`compareTo`方法来对下面的5个字符串进行排序：

```
America
Zulu
able
zebra
Ångström
```

按照字典中的顺序，你希望将大写和小写看作是等价的。对于一个说英语的读者来说，期望的排序的结果应该是：

```
able
America
Ångström
zebra
Zulu
```

但是，这种顺序对于瑞典用户是不可接受的。在瑞典语中，字母Å和字母A是不同的，它应该排在字母Z之后！就是说，瑞典用户希望排序的结果是：

```
able
America
zebra
Zulu
Ångström
```

但是，一旦注意到这个问题，整理起来就很简单了。一般先得到一个`Locale`对象，然后，调用`getInstance`工厂方法来得到一个`Collator`对象。最后，当希望对字符串进行排序时，使用这个排序器的`compare`方法，而不是`String`类的`compareTo`方法。

```
Locale loc = . . .;
Collator coll = Collator.getInstance(loc);
if (coll.compare(a, b) < 0) // a comes before b . . .;
```

最重要的是，`Collator`类实现了`Comparator`接口。因此，可以传一个排序器对象到`Collections.sort`方法中来对一组字符串进行排序：

```
Collections.sort(strings, coll);
```

#### 5.4.1 排序强度

你可以设置排序器的强度来选择不同的排序行为。字符间的差别可以被分为首要的(primary)、其次的(secondary)和再次的(tertiary)。比如，在英语中，“A”和“Z”之间的差别被归为首要的，而“A”和“Å”之间的差别是其次的，“A”和“a”之间是再次的。

如果将排序器的强度设置成`Collator.PRIMARY`，那么排序器将只关注primary级的差别。如果设置成`Collator.SECONDARY`，排序器将把secondary级的差别也考虑进去。就是说，两个字符串在“secondary”或“tertiary”强度下更容易被区分开来，如表5-4所示。

如果强度被设置为`Collator.IDENTICAL`，则不允许有任何差异。这种设置在与排序器的第二种相当技术性的设置，即分解模式(decomposition mode)，联合使用时，就会显得非常有用。我们将在下一节讨论分解模式。

表5-4 不同的强度下的排序 (英语Locale)

| 首 要                 | 其 次                 | 再 次                 |
|---------------------|---------------------|---------------------|
| Angstrom = Ångström | Angstrom ≠ Ångström | Angstrom ≠ Ångström |
| Able = able         | Able = able         | Able ≠ able         |

### 5.4.2 分解

偶尔我们会碰到一个字符或字符序列在描述成Unicode时，可以有多种方式。例如，“Å”可以是Unicode字符U+00C5，或者可以表示成普通的A（U+0065）后跟“（上方组合环”，U+030A）。也许会让你更吃惊的是，字母序列“ffi”可以用代码U+FB03描述成单个字符“拉丁小连字ffi”。（有人会说这是表示方法的不同，不应该因此产生不同的Unicode字符，但我们不会作这样的规定。）

Unicode标准对字符串定义了四种规范化形式 (normalization form)：D、KD、C和KC，请查看<http://www.unicode.org/unicode/reports/tr15/tr15-23.html>以了解详细信息。这其中的两种都是用于排序的。在规范化形式D中，重音字符被分解为基字符和组合重音符。例如，Å就被转换成由字母A和上方组合环°构成的序列。规范化形式KD更进一步地将兼容性字符 (compatibility character) 也进行了分解，例如ffi连字或商标符号™。

我们可以选择排序器所使用的规范化程度：`Collator.NO_DECOMPOSITION`表示不对字符串做任何规范化，这个选项处理速度较快，但是对于以多种形式表示字符的文本就显得不适用了；默认值`Collator.CANONICAL_DECOMPOSITION`使用规范化形式D，这对于包含重音但不包含连字的文本是非常有用的形式；最后是使用规范化形式KD的“完全分解”。请参见表5-5中的示例：

让排序器去多次分解一个字符串是很浪费的。如果一个字符串要和其他字符串进行多次比较，可以将分解的结果保存在一个排序键对象中。`getCollationKey`方法返回一个`CollationKey`对象，可以用它来进行更深入的、更快速的比较操作。下面是一个例子：

```
String a = ...;
CollationKey aKey = coll.getCollationKey(a);
if(aKey.compareTo(coll.getCollationKey(b)) == 0) // fast comparison
    ...

```

最后，有可能在你不需要进行排序时，也希望将字符串转换成它们的规范化形式。例如，在将字符串存储到数据库中，或与其他程序进行通信时。从Java SE 6.0开始，`java.text.Normalizer`类实现了对规范化的处理。例如：

```
String name = "Ångström";
String normalized = Normalizer.normalize(name, Normalizer.Form.NFD); // uses normalization form D
```

上面的字符串规范化后包含10个字符，其中“Å”和“ö”替换成了“A°”和“o”序列。但是，这通常并非用于存储或传输的最佳形式。规范化形式C首先进行分解，然后将重音按照标准化的顺序组合在后面。根据W3C的标准，这是用于在因特网上进行数据传输的推荐模型。

表5-5 分解模式之间的差异

| 不 分 解  | 规 范 分 解 | 完 全 分 解 |
|--------|---------|---------|
| Å ≠ A° | Å = A°  | Å = A°  |
| ™ ≠ TM | ™ ≠ TM  | ™ = TM  |

程序清单5-4中的程序让你体验了一下collation排序。向文本框中输入一个词然后点击Add按钮把它添加到一个单词列表中。每次添加一个单词，或选择locale、强度或分解模式，列表中的单词就会被重新排列。=号表示这两个词被认为是等同的（参见图5-3）。

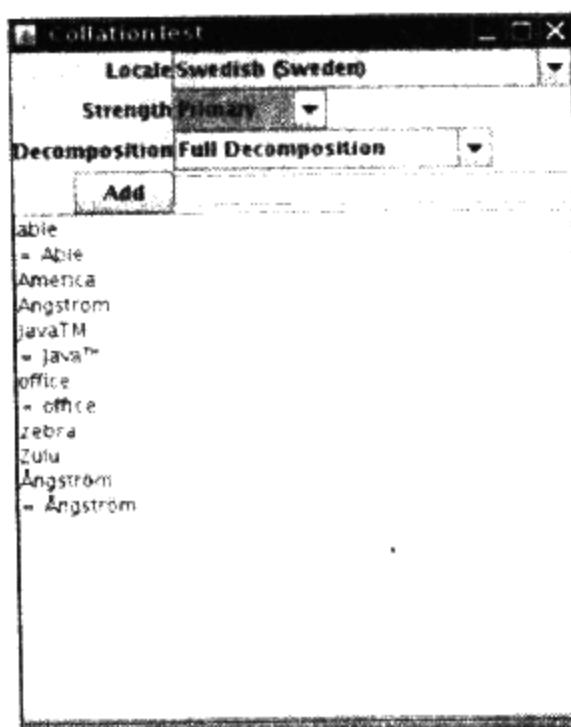


图5-3 CollationTest程序

在组合框中的locale名的显示顺序，是用默认locale的排序器进行排序而产生的顺序。如果用美国英语locale运行这个程序，即使逗号的Unicode值比后括号的Unicode值大，“Norwegian (Norway,Nynorsk)”也会显示在“Norwegian (Norway)”的前面。

#### 程序清单5-4 CollationTest.java

```
1. import java.awt.*;  
2. import java.awt.event.*;  
3. import java.text.*;  
4. import java.util.*;  
5. import java.util.List;  
6.  
7. import javax.swing.*;  
8.  
9. /**  
10. * This program demonstrates collating strings under various locales.  
11. * @version 1.13 2007-07-25  
12. * @author Cay Horstmann  
13. */  
14. public class CollationTest  
15. {  
16.     public static void main(String[] args)  
17.     {  
18.         EventQueue.invokeLater(new Runnable()  
19.         {  
20.             public void run()  
21.             {  
22.                 JFrame frame = new CollationFrame();  
23.             }  
24.         }  
25.     }  
26. }
```

```
24.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
25.         frame.setVisible(true);
26.     }
27. );
28. }
29. }
30.
31. /**
32. * This frame contains combo boxes to pick a locale, collation strength and decomposition
33. * rules, a text field and button to add new strings, and a text area to list the
34. * collated strings.
35. */
36. class CollationFrame extends JFrame
37. {
38.     public CollationFrame()
39.     {
40.         setTitle("CollationTest");
41.
42.         setLayout(new GridBagLayout());
43.         add(new JLabel("Locale"), new GBC(0, 0).setAnchor(GBC.EAST));
44.         add(new JLabel("Strength"), new GBC(0, 1).setAnchor(GBC.EAST));
45.         add(new JLabel("Decomposition"), new GBC(0, 2).setAnchor(GBC.EAST));
46.         add(addButton, new GBC(0, 3).setAnchor(GBC.EAST));
47.         add(localeCombo, new GBC(1, 0).setAnchor(GBC.WEST));
48.         add(strengthCombo, new GBC(1, 1).setAnchor(GBC.WEST));
49.         add(decompositionCombo, new GBC(1, 2).setAnchor(GBC.WEST));
50.         add(newWord, new GBC(1, 3).setFill(GBC.HORIZONTAL));
51.         add(new JScrollPane(sortedWords), new GBC(0, 4, 2, 1).setFill(GBC.BOTH));
52.
53.         locales = (Locale[]) Collator.getAvailableLocales().clone();
54.         Arrays.sort(locales, new Comparator<Locale>()
55.         {
56.             private Collator collator = Collator.getInstance(Locale.getDefault());
57.
58.             public int compare(Locale l1, Locale l2)
59.             {
60.                 return collator.compare(l1.getDisplayName(), l2.getDisplayName());
61.             }
62.         });
63.         for (Locale loc : locales)
64.             localeCombo.addItem(loc.getDisplayName());
65.         localeCombo.setSelectedItem(Locale.getDefault().getDisplayName());
66.
67.         strings.add("America");
68.         strings.add("able");
69.         strings.add("Zulu");
70.         strings.add("zebra");
71.         strings.add("\u00C5ngstr\u00F6m");
72.         strings.add("A\u030angstro\u0308m");
73.         strings.add("Angstrom");
74.         strings.add("Able");
75.         strings.add("office");
76.         strings.add("o\uFB03ce");
77.         strings.add("Java\u2122");
78.         strings.add("JavaTM");
79.         updateDisplay();
```

```
80
81     addButton.addActionListener(new ActionListener()
82     {
83         public void actionPerformed(ActionEvent event)
84         {
85             strings.add(newWord.getText());
86             updateDisplay();
87         }
88     });
89
90     ActionListener listener = new ActionListener()
91     {
92         public void actionPerformed(ActionEvent event)
93         {
94             updateDisplay();
95         }
96     };
97
98     localeCombo.addActionListener(listener);
99     strengthCombo.addActionListener(listener);
100    decompositionCombo.addActionListener(listener);
101    pack();
102 }
103
104 /**
105 * Updates the display and collates the strings according to the user settings.
106 */
107 public void updateDisplay()
108 {
109     Locale currentLocale = locales[localeCombo.getSelectedIndex()];
110     localeCombo.setLocale(currentLocale);
111
112     currentCollator = Collator.getInstance(currentLocale);
113     currentCollator.setStrength(strengthCombo.getValue());
114     currentCollator.setDecomposition(decompositionCombo.getValue());
115
116     Collections.sort(strings, currentCollator);
117
118     sortedWords.setText("");
119     for (int i = 0; i < strings.size(); i++)
120     {
121         String s = strings.get(i);
122         if (i > 0 && currentCollator.compare(s, strings.get(i - 1)) == 0) sortedWords
123             .append("=");
124         sortedWords.append(s + "\n");
125     }
126     pack();
127 }
128
129 private List<String> strings = new ArrayList<String>();
130 private Collator currentCollator;
131 private Locale[] locales;
132 private JComboBox localeCombo = new JComboBox();
133
134 private EnumCombo strengthCombo = new EnumCombo(Collator.class, new String[] { "Primary",
135     "Secondary", "Tertiary", "Identical" });
```

```

136.     private EnumCombo decompositionCombo = new EnumCombo(Collator.class, new String[] {
137.         "Canonical Decomposition", "Full Decomposition", "No Decomposition" });
138.     private JTextField newWord = new JTextField(20);
139.     private JTextArea sortedWords = new JTextArea(20, 20);
140.     private JButton addButton = new JButton("Add");
141. }
```

**java.text.Collator 1.1**

- static Locale[] getAvailableLocales()

返回Locale对象的一个数组，通过它可以得到Collator对象。

- static Collator getInstance()

- static Collator getInstance(Locale l)

为默认或给定的locale返回一个排序器。

- int compare(String a, String b)

如果a在b之前，则返回负值；如果它们等价，则返回0，否则返回正值。

- boolean equals(String a, String b)

如果它们等价，则返回true，否则返回false。

- void setStrength(int strength)

- int getStrength()

设置或获取排序器的强度。更强的排序器可以区分更多的词。强度的值可以是Collator.PRIMARY、Collator.SECONDARY和Collator.TERTIARY。

- void setDecomposition(int decomp)

- int getDecompositon()

设置或获取排序器的分解模式。分解越细，判断两个字符串是否相等时就越严格。分解的等级值可以是Collator.NO\_DECOMPOSITION、Collator.CANONICAL\_DECOMPOSITION和Collator.FULL\_DECOMPOSITION。

- CollationKey getCollationKey(String a)

返回一个排序器键，这个键包含一个按特定格式分解的字符，可以快速地和其他排序器键进行比较。

**java.text.CollationKey 1.1**

- int compareTo(CollationKey b)

如果这个键在b之前，则返回一个负值；如果两者等价，则返回0，否则返回正值。

**java.text.Normalizer 6**

- static String normalize(CharSequence str, Normalizer.Form form)

返回str的范化形式，form的值是ND、NKD、NC或NKC之一。

## 5.5 消息格式化

Java类库中有一个MessageFormat类，用来格式化带变量的文本，就像这样：

"On {2}, a {0} destroyed {1} houses and caused {3} of damage."

括号中的数字是一个占位符，可以用实际的名字和值来替换它。使用静态方法 `MessageFormat.format` 可以用实际的值来替换这些占位符。从JDK 5.0开始，它已经是一个“varargs”方法了，所以你可以通过下面的方法提供参数：

```
String msg = MessageFormat.format("On {2}, a {0} destroyed {1} houses and caused {3} of damage.",  
    "hurricane", 99, new GregorianCalendar(1999, 0, 1).getTime(), 10.0E8);
```

在这个例子中，占位符{0}被“hurricane”替换，{1}被99替换，等等。

上述例子的结果是下面的字符串

```
On 1/1/99 12:00 AM, a hurricane destroyed 99 houses and caused 100,000,000 of damage.
```

这只是开始，离完美还有距离。我们不想将时间显示为“12:00 AM”，而且我们想用当地的流通货币值来表示造成的损害量。通过为占位符提供一个可选的格式，就可以做到这一点：

```
"On {2,date,long}, a {0} destroyed {1} houses and caused {3,number,currency} of damage."
```

这段示例代码将打印出：

```
On January 1, 1999, a hurricane destroyed 99 houses and caused $100,000,000 of damage.
```

一般来说，占位符标记后面可以跟一个类型（type）和一个风格（style），它们之间用逗号隔开。类型可以是：

```
number  
time  
date  
choice
```

如果类型是number，那么风格可以是

```
integer  
currency  
percent
```

或者可以是数字格式模式，就像\$、##0。（关于格式的更多信息，参见`DecimalFormat`类的文档。）

如果类型是time或date，那么风格可以是

```
short  
medium  
long  
full
```

或者一个日期格式模版像yyyy-MM-dd。（关于格式的更多信息，见`SimpleDateFormat`类的文档。）

选择格式更复杂，我们将在下一节中讨论它们。



**警告：**静态的`MessageFormat.format`方法使用当前的`locale`对值进行格式化。要想用任意的`locale`进行格式化，还有一些工作要做，因为这个类还没有提供任何可以使用的“varargs”方法。你需要把将要格式化的值置于`Object[]`数组中，就像下面这样：

```
MessageFormat mf = new MessageFormat(pattern, loc);  
String msg = mf.format(new Object[] { values });
```

**API** `java.text.MessageFormat 1.1`

- `MessageFormat(String pattern)`
- `MessageFormat(String pattern, Locale loc)`  
用给定的模式和locale构建一个消息格式对象。
- `void applyPattern(String pattern)`  
给消息格式对象设置特定的模式。
- `void setLocale(Locale loc)`
- `Locale getLocale()`  
设置或获取消息中占位符所使用的locale。这个locale仅仅被通过调用`applyPattern`方法所设置的后续模式所使用。
- `static String format(String pattern, Object... args)`  
通过使用`args[i]`作为站位符`{i}`的输入来格式化`pattern`字符串。
- `StringBuffer format(Object args, StringBuffer result, FieldPosition pos)`  
格式化`MessageFormat`的模式。`args`参数必须是一个对象数组。被格式化的字符串会被附加到`result`末尾，并返回`result`。如果`pos`等于`new FieldPosition(MessageFormat.Field.ARGUMENT)`，就用它的`beginIndex`和`endIndex`属性值来设置替换占位符`{i}`的文本位置。如果不关心位置信息，可以将它设为`null`。

**API** `java.text.Format 1.1`

- `String format(Object obj)`

按照格式器的规则格式化给定的对象，这个方法将调用`format(obj,new StringBuffer(),new FieldPosition(1)).toString()`。

## 选择格式

让我们仔细地看看前面一节所提到的模式：

"On {2}, a {0} destroyed {1} houses and caused {3} of damage."

如果我们用“earthquake”来替换代表灾难的占位符`{0}`，那么，在英语中，这句话的语法就不正确了。

On January 1, 1999, a earthquake destroyed . . .

这说明，我们真正希望的是将冠词“a”集成到占位符中去：

"On {2}, {0} destroyed {1} houses and caused {3} of damage."

这样我们就应该用“a hurricane”或“an earthquake”来替换`{0}`。当消息需要被翻译成某种语言，而该语言中的词会随词性的变化而变化时，这种替换方式就特别适用。比如，在德语中，模式可能会是：

"{0} zerstörte am {2} {1} Häuser und richtete einen Schaden von {3} an."

这样，占位符将被正确地替换成冠词和名词的组合，比如“Ein Wirbelsturm”，“Eine Naturkatastrophe”。

让我们来看看参数{1}。如果灾难的后果不严重，{1}的替换值可能是数字1，消息就变成：

```
On January 1, 1999, a mudslide destroyed 1 houses and . . .
```

我们当然希望消息能够随占位符的值而变化，这样就能根据具体的值形成：

```
no houses
one house
2 houses
. . .
```

Choice格式化选项就是为了这个目的而设计的。

一个选择格式是由一个序列对，每一个对包括

- 一个下限 (lower limit)
- 一个格式字符串 (format string)

下限和格式字符串由一个#符号分隔，对与对之间由符号|分隔。

例如，

```
{1,choice,0#no houses|1#one house|2#{1} houses}
```

表5-6显示了格式字符串对{1}的不同值产生的作用。

为什么在格式化字符串中两次用到了{1}？当消息格式将选择的格式应用于占位符{1}而且替换值是\$2时，那么选择格式返回“{1} houses”。这个消息由消息格式再次格式化，将这次的结果和上一次的叠加。

表5-6 由选择格式进行格式化的字符串

| {1} | 结 果          |
|-----|--------------|
| 0   | "no houses"  |
| 1   | "one houses" |
| 3   | "3 houses"   |
| -1  | "no houses"  |

 注意：这个例子说明选择格式的设计者有些糊涂了。如果你有3个格式字符串，你就需要两个下限来分隔它们。一般来说，你需要的下限数目比格式字符串数目少1。就像你在表5-4中见到的，MessageFormat类将忽略第一个下限。

如果这个类的设计者意识到下限只在两个选择之间出现，那么语法就要清楚得多，比如，

```
no houses|1|one house|2|{1} houses // not the actual format
```

可以使用<符号来表示如果替换值严格小于下限，则选中这个选择项。

也可以使用≤（unicode中的代码是\u2264）来实现和#相同的效果。如果愿意的话，甚至可以将第一个下限的值定义为-∞（unicode代码是-\u221E）。

例如，

```
-∞<no houses|0<one house|2≤{1} houses
```

或者使用Unicode转义字符，

```
-\u221E<no houses|0<one house|2\u2264{1} houses
```

让我们来结束自然灾害的场景。如果我们将选择字符串放到原始消息字符串中，那么我们得到下面的格式化指令：

```
String pattern = "On {2,date,long}, {0} destroyed {1,choice,0#no houses|1#one house|2#{1} houses}" + "and caused {3,number,currency} of damage.;"
```

在德语中，即

```
String pattern = "{0} zerstörte am {2,date,long} {1,choice,0#kein Haus|1#ein Haus|2#{1} Häuser}"
    + "und richtete einen Schaden von {3,number,currency} an.";
```

请注意，在德语中词的顺序和英语中是不同的，但是你传给format方法的对象数组是相同的。可以用格式字符串中占位符的顺序来处理单词顺序的改变。

## 5.6 文本文件和字符集

众所周知，Java是完全基于Unicode的。但是，操作系统一般有它们自己的字符编码，比如在美国是ISO-8859-1（8位代码，有时候也称为“ANSI”代码），在台湾是Big5。

当把数据保存到一个文本文件中时，应该照顾到本地的字符编码，这样程序的用户就可以用他们的其他程序打开这个文本文件。字符编码是在FileWriter的构造器中指定的：

```
out = new FileWriter(filename, "ISO-8859-1");
```

可以在第 I 卷第12章中找到所支持的编码方式的完整列表。

遗憾的是，目前，locale和字符编码之间没有任何联系。比如，如果你的用户选择的是台湾的locale zh\_TW，但是在Java中并没有提供任何方法来告诉你应该用Big5字符编码。

### 源文件的字符编码

作为程序员，要牢记你需要与Java编译器交互，这种交互需要通过本地系统的工具来完成。例如，可以使用中文版的记事本来写你的Java源代码文件。但这样写出来的源码是不能随处可用的，因为它们使用的是本地的字符编码（GB或Big5，这取决于你使用的是哪种中文操作系统）。只有编译后的class文件才能随处使用，因为它们会自动地使用“modified UTF-8”编码来处理标识符和字符串。这意味着即使在程序编译和运行时，依然有3种字符编码参与其中：

- 源文件：本地编码
- 类文件：modified UTF-8
- 虚拟机：UTF-16

关于modified UTF-8和UTF-16格式的定义，参见第 I 卷第12章。



**提示：**你可以用-encoding标记来设定你的源文件的字符编码，例如

```
javac -encoding Big5 Myfile.java
```

为了使你的源文件能够到处使用，必须使用通用的ASCII编码。就是说，你需要将所有非ASCII字符转换成等价的Unicode编码。比如，不要使用字符串“Häuser”，应该使用“H\u0084user”。JDK包含一个工具——native2ascii，可以用它来将本地字符编码转换成通用的ASCII。这个工具直接将输入中的每一个非ASCII字符替换为一个\u加四位十六进制数字的Unicode值。使用native2ascii时，需要提供输入和输出文件的名字。

```
native2ascii Myfile.java Myfile.temp
```

可以用-reverse选项来进行逆向转换：

```
native2ascii -reverse Myfile.temp Myfile.java
```

可以用-encoding选项指定另一个编码。编码的名字必须是编码表所列出来的名字之一，

这个表可以在第I卷第12章中找到。

```
native2ascii -encoding Big5 Myfile.java Myfile.temp
```

**!** 提示：限制自己只使用通用ASCII的类名是一个好主意。因为类的名字会成为类文件的名字，对所有由非ASCII代码表示的名字的处理都将由本地文件系统负责。但令人沮丧的是，Windows 95使用被称为“Code Page 437”或“最初的PC编码”方式来处理文件名。如果你编译得到了一个Bär类，并试图在Windows 95上运行它，就会得到一个错误消息“不能找到类BΣr”。

## 5.7 资源包

当本地化一个应用时，可能会有大量的消息字符串、按钮标签和其他的东西需要被翻译。为了能比较轻松地完成这项任务，你会希望在外部定义消息字符串，通常称之为资源(resource)。翻译人员不需要接触程序源代码就可以很容易地编辑资源文件。

在Java中，你要使用属性文件来设定字符串资源，并为其他类型的资源实现相应的类。

- 注意：Java技术资源和Windows和Macintosh资源不同。Macintosh或Windows可执行文件在程序代码以外的地方存储类似菜单、对话框、图标和消息这样的资源。一个资源编辑器能够在不影响程序代码的情况下检查并更新这些资源。
- 注意：第I卷第10章描述了JAR文件资源的概念，以及为何数据文件、声音和图片可以被存放在JAR文件中。Class类的getResource方法找到相应的文件，打开它并返回资源的URL。通过将文件放置到JAR文件中，你就将查找这些资源文件的工作留给了类的加载器去处理，加载器知道如何定位JAR文件中的项目。但是，这个机制不支持locale。

### 5.7.1 定位资源包

当本地化一个应用时，会制造出很多资源包(resource bundle)。每一个包都是一个属性文件或者是一个描述了与locale相关的项的类(比如消息、标签等)。对于每一个包，都要为所有你想要支持的locale提供相应的版本。

需要对这些包使用一种统一的命名规则。例如，为德国定义的资源放在一个名为“包名\_de\_DE”的文件中，为所有说德语的国家所共享的资源放在名为“包名\_de”的文件中。一般来说，使用

包名\_语言\_国家

来命名所有和国家相关的资源，使用

包名\_语言

来命名所有和语言相关的资源。最后，作为后备，可以把默认资源放到一个没有后缀的文件中。

可以用下面的命令加载一个包

```
ResourceBundle currentResources = ResourceBundle.getBundle(bundleName, currentLocale);
```

getBundle方法试图加载符合当前locale定义的语言、国家和变量的包。如果失败，通过依次

放弃语言、变量和国家来继续进行查找。同样的查找被应用于默认的locale，最后，如果还不行的话就去查看默认的包文件，如果这也失败了，方法抛出一个MissingResourceException异常。

这就是说，`getBundle`方法试图加载以下的包。

包名\_当前Locale的语言\_当前Locale的国家\_当前Locale的变量

包名\_当前Locale的语言\_当前Locale的国家

包名\_当前Locale的语言

包名\_默认Locale的语言\_默认Locale的国家\_默认Locale的变量

包名\_默认Locale的语言\_默认Locale的国家

包名\_默认Locale的语言

包名

一旦`getBundle`方法定位了一个包，比如，包名`_de_DE`，它还会继续查找包名`_de`和包名。如果这些包也存在，它们在资源层次中就成为了包名`_de_DE`的父包。以后，当查找一个资源时，如果在当前包中没有找到，就去查找其父包。就是说，如果一个特定的资源在当前包中没有被找到。比如，某个特定资源在包名`_de_DE`中没有找到，那么就会去查询包名`_de`和包名。

这是一项非常有用的服务，如果手工来编写将会非常麻烦。Java编程语言的资源包机制会自动定位与给定的locale匹配得最好的项。可以很容易地把越来越多的本地化信息加到现存的程序中：你需要做的只是增加额外的资源包。

**!** 提示：不需要把你的程序的所有资源都放到同一个包中。可以用一个包来存放按钮标签、用另一个包存放错误消息等。

### 5.7.2 属性文件

对字符串进行国际化是很直接的。你把所有字符串放到一个属性文件中，比如`MyProgramStrings.properties`。这是一个每行存放一个键-值对的文本文件。典型的属性文件看起来就像这样：

```
computeButton=Rechnen  
colorName=black  
defaultPaperSize=210x297
```

然后你就像上一节描述的那样命名你的属性文件，例如，

```
MyProgramStrings.properties  
MyProgramStrings_en.properties  
MyProgramStrings_de_DE.properties
```

你可以直接加载包，如

```
 ResourceBundle bundle = ResourceBundle.getBundle("MyProgramStrings", locale);
```

要查找一个特定的字符串，可以调用

```
String computeButtonLabel = bundle.getString("computeButton");
```

**X** 警告：存储属性的文件都是ASCII文件。如果你需要将Unicode字符放到属性文件中，那么请用\uxxxx编码方式对它们进行编码。比如，设定“colorName=Grün”，使用

```
colorName=Gr\u00fcn
```

你可以使用native2ascii工具来产生这些文件。

### 5.7.3 包类

为了提供字符串以外的资源，需要定义类，它扩展自**ResourceBundle**类。应该使用标准的命名规则来命名你的类，比如

```
MyProgramResources.java  
MyProgramResources_en.java  
MyProgramResources_de_DE.java
```

你可以使用与加载属性文件相同的**getBundle**方法来加载这个类：

```
ResourceBundle bundle = ResourceBundle.getBundle("MyProgramResources", locale);
```

**X** 警告：当搜索包时，如果在类中的包和在属性文件中的包中都存在匹配，优先选择类中的包。

每一个资源绑定类都实现了一个查询表。你需要为每一个你想定位的设置都提供一个关键字字符串，使用这个字符串来提取相应的设置。例如，

```
Color backgroundColor = (Color) bundle.getObject("backgroundColor");  
double[] paperSize = (double[]) bundle.getObject("defaultPaperSize");
```

实现资源绑定类的最简单方法就是继承**ListResourceBundle**类。**ListResourceBundle**让你把所有资源都放到一个对象数组并提供查找功能。要遵循以下的代码框架：

```
public class bundleName_language_country extends ListResourceBundle  
{  
    public Object[][] getContents() { return contents; }  
    private static final Object[][] contents =  
    {  
        { key1, value1 },  
        { key2, value2 },  
        ...  
    }  
}
```

例如，

```
public class ProgramResources_de extends ListResourceBundle  
{  
    public Object[][] getContents() { return contents; }  
    private static final Object[][] contents =  
    {  
        { "backgroundColor", Color.black },  
        { "defaultPaperSize", new double[] { 210, 297 } }  
    }  
}  
  
public class ProgramResources_en_US extends ListResourceBundle  
{  
    public Object[][] getContents() { return contents; }  
    private static final Object[][] contents =  
    {  
        { "backgroundColor", Color.blue },  
        { "defaultPaperSize", new double[] { 216, 279 } }  
    }  
}
```

 注意：纸的尺寸是以毫米为单位给出的。在地球上，除了加拿大和美国，其他地区的人们都使用ISO 216规格的纸。更多信息见<http://www.cl.cam.ac.uk/~mgk25/iso-paper.html>。根据美国公制联合会(<http://lamar.colostate.edu/~hillger>)的研究，世界上只有三个国家没有正式采用公制系统，分别是利比亚、缅甸和美国。

或者，你的资源绑定类可以扩展**ResourceBundle**类。然后需要实现两个方法，一是列出所有键，二是用给定的键查找相应的值：

```
Enumeration<String> getKeys()  
Object handleGetObject(String key)
```

**ResourceBundle**类的**getObjet**方法会调用你提供的**handleGetObject**方法。

 注意：从Java SE6开始，你可以选择存储资源的其他机制。例如，你可以定制从XML文件和数据库中获取资源的资源加载机制。详细信息请查看[http://java.sun.com/developer/technicalArticles/javase/i18n\\_enhance](http://java.sun.com/developer/technicalArticles/javase/i18n_enhance)。

#### **java.util.ResourceBundle 1.1**

- static ResourceBundle getBundle(String baseName, Locale loc)
- static ResourceBundle getBundle(String baseName)

在给定的**locale**或默认的**locale**下以给定的名字加载资源绑定类和它的父类。如果资源包类位于一个Java包中，那么类的名字必须包含完整的包名，例如“**intl.ProgramResources**”。资源包类必须是**public**的，这样**getBundle**方法才能访问它们。

- Object getObject(String name)

从资源包或它的父包中查找一个对象。

- String getString(String name)

从资源包或它的父包中查找一个对象并把它转型成字符串。

- String[] getStringArray(String name)

从资源包或它的父包中查找一个对象并把它转型成字符串数组。

- Enumeration<String> getKeys()

返回一个枚举对象，枚举出资源包中的所有键，也包括父包中的键。

- Object handleGetObject(String key)

如果你定义了自己的资源查找机制，这个方法需要被覆写，用来查找与给定的键相关联的资源的值。

## 5.8 一个完整的例子

在这一节中，我们使用本章中的内容来对退休金计算器小程序进行本地化，这个小程序计算你是否为退休存够了钱，你需要输入年龄，每个月存多少钱等信息（参见图5-4）。

文本区和图表显示每年退休基金中的余额。如果你后半生的退休金余额变成负数，并且表中的数据条在x轴以下，你就需要做些什么了；例如，存更多的钱、推迟退休、早死或变得更年轻。

这个退休计算器可以在三种**locale**下工作（英语、德语和中文）。下面是进行本地化时的一些要点：

- 标签、按钮和消息被翻译成德语和中文。你可以在RetireResources\_de, RetireResources\_zh中找到它们。英语作为后备，见RetireResources文件。为了产生中文消息，我们首先用中文Windows上的记事本来编写文件，然后用native2ascii来将字符转换成Unicode。
- 当locale改变时，我们重置标签并格式化文本框中的内容。
- 文本域在本地格式下处理数字、货币值和百分数。
- 计算域使用MessageFormat。格式字符串被存储在每种语言的资源绑定中。
- 为了说明的确可行，我们按照用户选择的语言为条柱图使用不同的颜色。

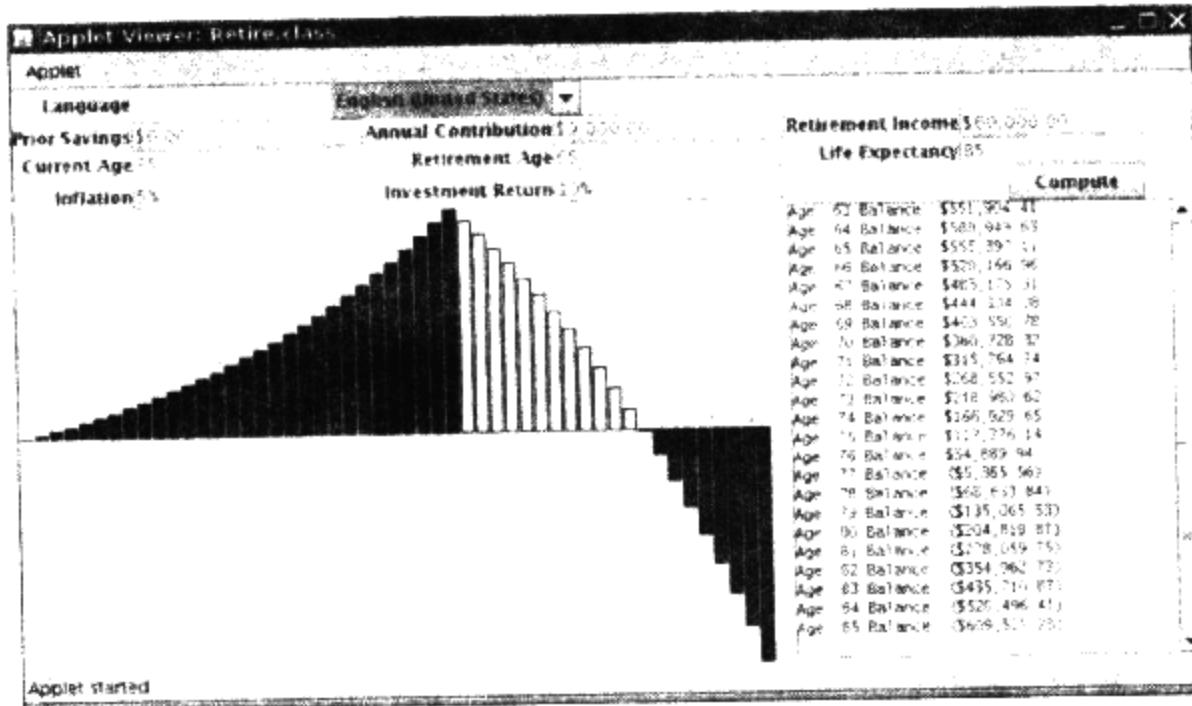


图5-4 使用英语的退休金计算器

程序清单5-5到程序清单5-8显示了代码，而程序清单5-9到程序清单5-11是本地化的字符串的属性文件。图5-5和图5-6分别显示了在德语和中文下的输出。为了显示中文，请确认你已经安装并在Java运行环境中配置了中文字体。否则，所有的中文字符将会显示“missing character”图标。

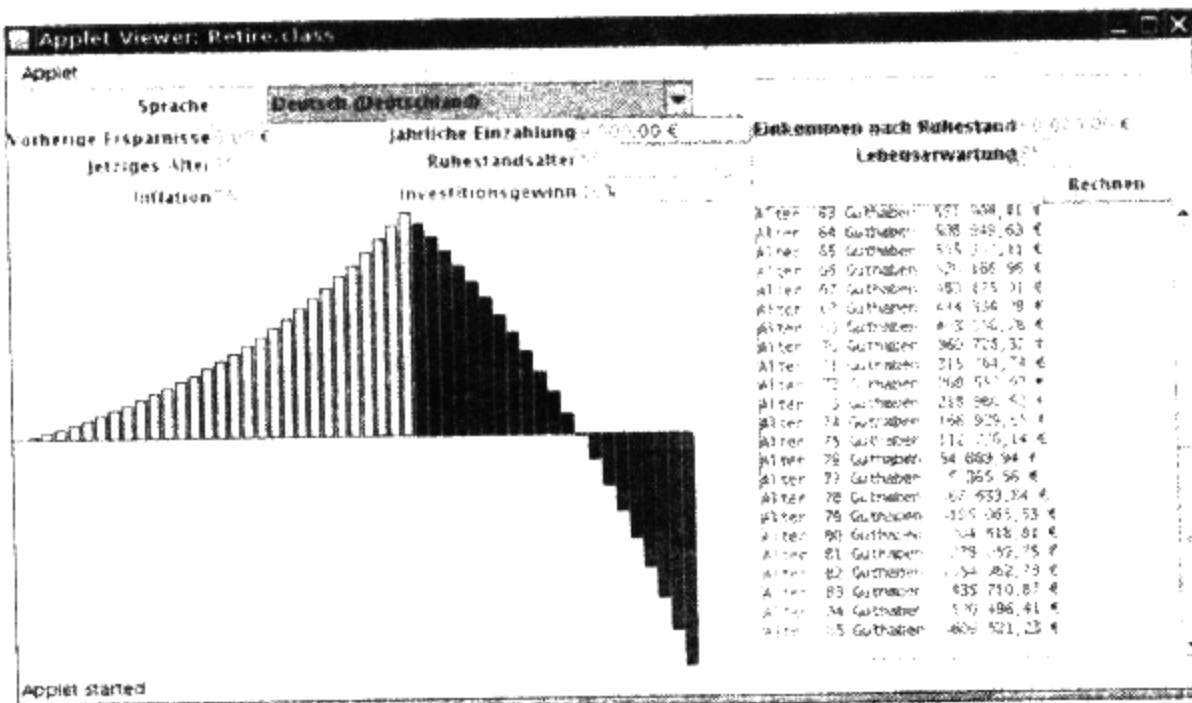


图5-5 使用德语的退休金计算器

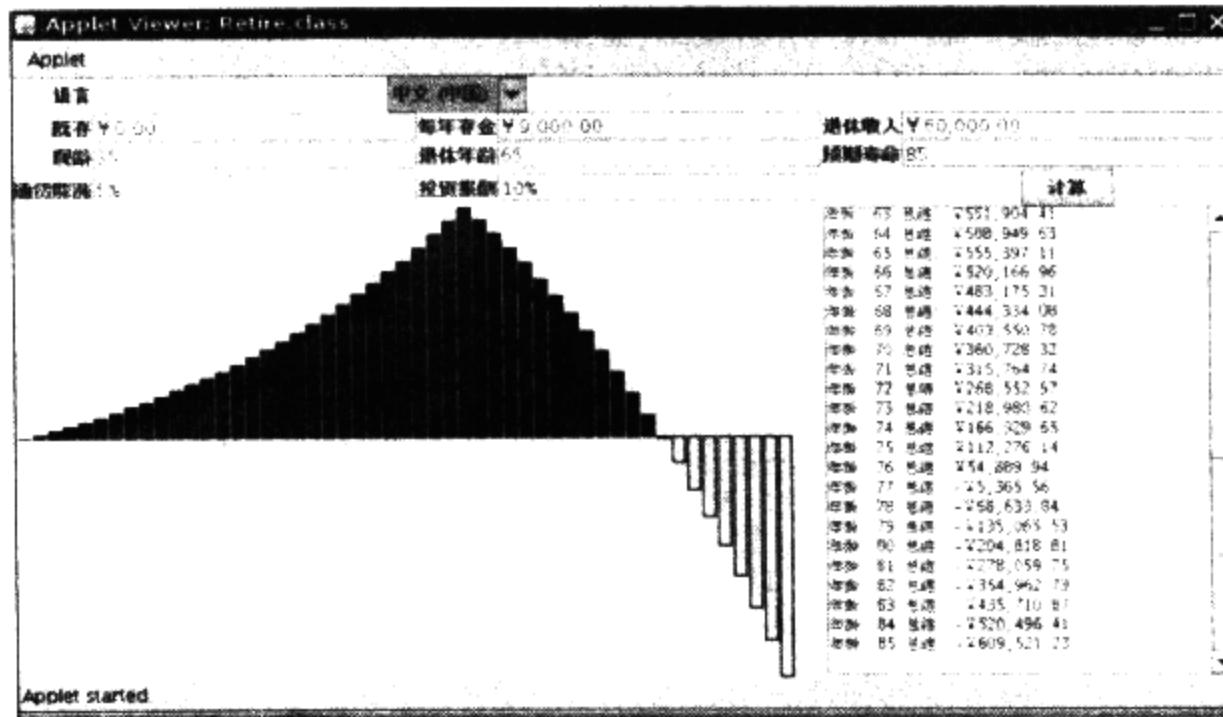


图5-6 使用中文的退休金计算器

## 程序清单5-5 Retire.java

```

1. import java.awt.*;
2. import java.awt.event.*;
3. import java.awt.geom.*;
4. import java.util.*;
5. import java.text.*;
6. import javax.swing.*;

7.
8. /**
9. * This applet shows a retirement calculator. The UI is displayed in English, German,
10. * and Chinese.
11. * @version 1.22 2007-07-25
12. * @author Cay Horstmann
13. */
14. public class Retire extends JApplet
15. {
16.     public void init()
17.     {
18.         EventQueue.invokeLater(new Runnable()
19.         {
20.             public void run()
21.             {
22.                 initUI();
23.             }
24.         });
25.     }
26.
27.     public void initUI()
28.     {
29.         setLayout(new GridBagLayout());
30.         add(languageLabel, new GBC(0, 0).setAnchor(GBC.EAST));
31.         add(savingsLabel, new GBC(0, 1).setAnchor(GBC.EAST));
32.         add(contribLabel, new GBC(2, 1).setAnchor(GBC.EAST));
33.         add(incomeLabel, new GBC(4, 1).setAnchor(GBC.EAST));

```

```
34.     add(currentAgeLabel, new GBC(0, 2).setAnchor(GBC.EAST));
35.     add(retireAgeLabel, new GBC(2, 2).setAnchor(GBC.EAST));
36.     add(deathAgeLabel, new GBC(4, 2).setAnchor(GBC.EAST));
37.     add(inflationPercentLabel, new GBC(0, 3).setAnchor(GBC.EAST));
38.     add(investPercentLabel, new GBC(2, 3).setAnchor(GBC.EAST));
39.     add(localeCombo, new GBC(1, 0, 3, 1));
40.     add(savingsField, new GBC(1, 1).setWeight(100, 0).setFill(GBC.HORIZONTAL));
41.     add(contribField, new GBC(3, 1).setWeight(100, 0).setFill(GBC.HORIZONTAL));
42.     add(incomeField, new GBC(5, 1).setWeight(100, 0).setFill(GBC.HORIZONTAL));
43.     add(currentAgeField, new GBC(1, 2).setWeight(100, 0).setFill(GBC.HORIZONTAL));
44.     add(retireAgeField, new GBC(3, 2).setWeight(100, 0).setFill(GBC.HORIZONTAL));
45.     add(deathAgeField, new GBC(5, 2).setWeight(100, 0).setFill(GBC.HORIZONTAL));
46.     add(inflationPercentField, new GBC(1, 3).setWeight(100, 0).setFill(GBC.HORIZONTAL));
47.     add(investPercentField, new GBC(3, 3).setWeight(100, 0).setFill(GBC.HORIZONTAL));
48.     add(retireCanvas, new GBC(0, 4, 4, 1).setWeight(100, 100).setFill(GBC.BOTH));
49.     add(new JScrollPane(retireText), new GBC(4, 4, 2, 1).setWeight(0, 100).setFill(GBC.BOTH));
50.
51.     computeButton.setName("computeButton");
52.     computeButton.addActionListener(new ActionListener()
53.     {
54.         public void actionPerformed(ActionEvent event)
55.         {
56.             getInfo();
57.             updateData();
58.             updateGraph();
59.         }
60.     });
61.     add(computeButton, new GBC(5, 3));
62.
63.     retireText.setEditable(false);
64.     retireText.setFont(new Font("Monospaced", Font.PLAIN, 10));
65.
66.     info.setSavings(0);
67.     info.setContrib(9000);
68.     info.setIncome(60000);
69.     info.setCurrentAge(35);
70.     info.setRetireAge(65);
71.     info.setDeathAge(85);
72.     info.setInvestPercent(0.1);
73.     info.setInflationPercent(0.05);
74.
75.     int localeIndex = 0; // US locale is default selection
76.     for (int i = 0; i < locales.length; i++)
77.         // if current locale one of the choices, select it
78.         if (getLocale().equals(locales[i])) localeIndex = i;
79.     setCurrentLocale(locales[localeIndex]);
80.
81.     localeCombo.addActionListener(new ActionListener()
82.     {
83.         public void actionPerformed(ActionEvent event)
84.         {
85.             setCurrentLocale((Locale) localeCombo.getSelectedItem());
86.             validate();
87.         }
88.     });
89. }
90.
```

```
91  /**
92   * Sets the current locale.
93   * @param locale the desired locale
94   */
95  public void setCurrentLocale(Locale locale)
96  {
97      currentLocale = locale;
98      localeCombo.setSelectedItem(currentLocale);
99      localeCombo.setLocale(currentLocale);
100 }
101 res = ResourceBundle.getBundle("RetireResources", currentLocale);
102 resStrings = ResourceBundle.getBundle("RetireStrings", currentLocale);
103 currencyFmt = NumberFormat.getCurrencyInstance(currentLocale);
104 numberFmt = NumberFormat.getNumberInstance(currentLocale);
105 percentFmt = NumberFormat.getPercentInstance(currentLocale);
106
107 updateDisplay();
108 updateInfo();
109 updateData();
110 updateGraph();
111 }
112
113 /**
114  * Updates all labels in the display.
115  */
116 public void updateDisplay()
117 {
118     languageLabel.setText(resStrings.getString("language"));
119     savingsLabel.setText(resStrings.getString("savings"));
120     contribLabel.setText(resStrings.getString("contrib"));
121     incomeLabel.setText(resStrings.getString("income"));
122     currentAgeLabel.setText(resStrings.getString("currentAge"));
123     retireAgeLabel.setText(resStrings.getString("retireAge"));
124     deathAgeLabel.setText(resStrings.getString("deathAge"));
125     inflationPercentLabel.setText(resStrings.getString("inflationPercent"));
126     investPercentLabel.setText(resStrings.getString("investPercent"));
127     computeButton.setText(resStrings.getString("computeButton"));
128 }
129
130 /**
131  * Updates the information in the text fields.
132  */
133 public void updateInfo()
134 {
135     savingsField.setText(currencyFmt.format(info.getSavings()));
136     contribField.setText(currencyFmt.format(info.getContrib()));
137     incomeField.setText(currencyFmt.format(info.getIncome()));
138     currentAgeField.setText(numberFmt.format(info.getCurrentAge()));
139     retireAgeField.setText(numberFmt.format(info.getRetireAge()));
140     deathAgeField.setText(numberFmt.format(info.getDeathAge()));
141     investPercentField.setText(percentFmt.format(info.getInvestPercent()));
142     inflationPercentField.setText(percentFmt.format(info.getInflationPercent()));
143 }
144
145 /**
146  * Updates the data displayed in the text area.
```

```
147     */
148     public void updateData()
149     {
150         retireText.setText("");
151         MessageFormat retireMsg = new MessageFormat("");
152         retireMsg.setLocale(currentLocale);
153         retireMsg.applyPattern(resStrings.getString("retire"));
154
155         for (int i = info.getCurrentAge(); i <= info.getDeathAge(); i++)
156         {
157             Object[] args = { i, info.getBalance(i) };
158             retireText.append(retireMsg.format(args) + "\n");
159         }
160     }
161
162     /**
163      * Updates the graph.
164      */
165     public void updateGraph()
166     {
167         retireCanvas.setColorPre((Color) res.getObject("colorPre"));
168         retireCanvas.setColorGain((Color) res.getObject("colorGain"));
169         retireCanvas.setColorLoss((Color) res.getObject("colorLoss"));
170         retireCanvas.setInfo(info);
171         repaint();
172     }
173
174     /**
175      * Reads the user input from the text fields.
176      */
177     public void getInfo()
178     {
179         try
180         {
181             info.setSavings(currencyFmt.parse(savingsField.getText()).doubleValue());
182             info.setContrib(currencyFmt.parse(contribField.getText()).doubleValue());
183             info.setIncome(currencyFmt.parse(incomeField.getText()).doubleValue());
184             info.setCurrentAge(numberFmt.parse(currentAgeField.getText()).intValue());
185             info.setRetireAge(numberFmt.parse(retireAgeField.getText()).intValue());
186             info.setDeathAge(numberFmt.parse(deathAgeField.getText()).intValue());
187             info.setInvestPercent(percentFmt.parse(investPercentField.getText()).doubleValue());
188             info.setInflationPercent(percentFmt.parse(
189                 inflationPercentField.getText()).doubleValue());
190         }
191         catch (ParseException e)
192         {
193         }
194     }
195
196     private JTextField savingsField = new JTextField(10);
197     private JTextField contribField = new JTextField(10);
198     private JTextField incomeField = new JTextField(10);
199     private JTextField currentAgeField = new JTextField(4);
200     private JTextField retireAgeField = new JTextField(4);
201     private JTextField deathAgeField = new JTextField(4);
202     private JTextField inflationPercentField = new JTextField(6);
203     private JTextField investPercentField = new JTextField(6);
```

```
204.     private JTextArea retireText = new JTextArea(10, 25);
205.     private RetireCanvas retireCanvas = new RetireCanvas();
206.     private JButton computeButton = new JButton();
207.     private JLabel languageLabel = new JLabel();
208.     private JLabel savingsLabel = new JLabel();
209.     private JLabel contribLabel = new JLabel();
210.     private JLabel incomeLabel = new JLabel();
211.     private JLabel currentAgeLabel = new JLabel();
212.     private JLabel retireAgeLabel = new JLabel();
213.     private JLabel deathAgeLabel = new JLabel();
214.     private JLabel inflationPercentLabel = new JLabel();
215.     private JLabel investPercentLabel = new JLabel();
216.
217.     private RetireInfo info = new RetireInfo();
218.
219.     private Locale[] locales = { Locale.US, Locale.CHINA, Locale.GERMANY };
220.     private Locale currentLocale;
221.     private JComboBox localeCombo = new LocaleCombo(locales);
222.     private ResourceBundle res;
223.     private ResourceBundle resStrings;
224.     private NumberFormat currencyFmt;
225.     private NumberFormat numberFmt;
226.     private NumberFormat percentFmt;
227. }
228.
229. /**
230. * The information required to compute retirement income data.
231. */
232. class RetireInfo
233. {
234.     /**
235.      * Gets the available balance for a given year.
236.      * @param year the year for which to compute the balance
237.      * @return the amount of money available (or required) in that year
238.     */
239.     public double getBalance(int year)
240.     {
241.         if (year < currentAge) return 0;
242.         else if (year == currentAge)
243.         {
244.             age = year;
245.             balance = savings;
246.             return balance;
247.         }
248.         else if (year == age) return balance;
249.         if (year != age + 1) getBalance(year - 1);
250.         age = year;
251.         if (age < retireAge) balance += contrib;
252.         else balance -= income;
253.         balance = balance * (1 + (investPercent - inflationPercent));
254.         return balance;
255.     }
256.
257.     /**
258.      * Gets the amount of prior savings.
259.      * @return the savings amount
```

```
260.     */
261.     public double getSavings()
262.     {
263.         return savings;
264.     }
265.
266.     /**
267.      * Sets the amount of prior savings.
268.      * @param newValue the savings amount
269.      */
270.     public void setSavings(double newValue)
271.     {
272.         savings = newValue;
273.     }
274.
275.     /**
276.      * Gets the annual contribution to the retirement account.
277.      * @return the contribution amount
278.      */
279.     public double getContrib()
280.     {
281.         return contrib;
282.     }
283.
284.     /**
285.      * Sets the annual contribution to the retirement account.
286.      * @param newValue the contribution amount
287.      */
288.     public void setContrib(double newValue)
289.     {
290.         contrib = newValue;
291.     }
292.
293.     /**
294.      * Gets the annual income.
295.      * @return the income amount
296.      */
297.     public double getIncome()
298.     {
299.         return income;
300.     }
301.
302.     /**
303.      * Sets the annual income.
304.      * @param newValue the income amount
305.      */
306.     public void setIncome(double newValue)
307.     {
308.         income = newValue;
309.     }
310.
311.     /**
312.      * Gets the current age.
313.      * @return the age
314.      */
315.     public int getCurrentAge()
316.     {
```

```
317.     return currentAge;
318. }
319.
320. /**
321. * Sets the current age.
322. * @param newValue the age
323. */
324. public void setCurrentAge(int newValue)
325. {
326.     currentAge = newValue;
327. }
328.
329. /**
330. * Gets the desired retirement age.
331. * @return the age
332. */
333. public int getRetireAge()
334. {
335.     return retireAge;
336. }
337.
338. /**
339. * Sets the desired retirement age.
340. * @param newValue the age
341. */
342. public void setRetireAge(int newValue)
343. {
344.     retireAge = newValue;
345. }
346.
347. /**
348. * Gets the expected age of death.
349. * @return the age
350. */
351. public int getDeathAge()
352. {
353.     return deathAge;
354. }
355.
356. /**
357. * Sets the expected age of death.
358. * @param newValue the age
359. */
360. public void setDeathAge(int newValue)
361. {
362.     deathAge = newValue;
363. }
364.
365. /**
366. * Gets the estimated percentage of inflation.
367. * @return the percentage
368. */
369. public double getInflationPercent()
370. {
371.     return inflationPercent;
372. }
```

```
373.  
374.     /**  
375.      * Sets the estimated percentage of inflation.  
376.      * @param newValue the percentage  
377.      */  
378.     public void setInflationPercent(double newValue)  
379.     {  
380.         inflationPercent = newValue;  
381.     }  
382.  
383.     /**  
384.      * Gets the estimated yield of the investment.  
385.      * @return the percentage  
386.      */  
387.     public double getInvestPercent()  
388.     {  
389.         return investPercent;  
390.     }  
391.  
392.     /**  
393.      * Sets the estimated yield of the investment.  
394.      * @param newValue the percentage  
395.      */  
396.     public void setInvestPercent(double newValue)  
397.     {  
398.         investPercent = newValue;  
399.     }  
400.  
401.     private double savings;  
402.     private double contrib;  
403.     private double income;  
404.     private int currentAge;  
405.     private int retireAge;  
406.     private int deathAge;  
407.     private double inflationPercent;  
408.     private double investPercent;  
409.  
410.     private int age;  
411.     private double balance;  
412. }  
413.  
414. /**  
415.      * This panel draws a graph of the investment result.  
416.      */  
417. class RetireCanvas extends JPanel  
418. {  
419.     public RetireCanvas()  
420.     {  
421.         setSize(PANEL_WIDTH, PANEL_HEIGHT);  
422.     }  
423.  
424.     /**  
425.      * Sets the retirement information to be plotted.  
426.      * @param newInfo the new retirement info.  
427.      */  
428.     public void setInfo(RetireInfo newInfo)  
429.     {
```

```
430.     info = newInfo;
431.     repaint();
432. }
433.
434. public void paintComponent(Graphics g)
435. {
436.     Graphics2D g2 = (Graphics2D) g;
437.     if (info == null) return;
438.
439.     double minValue = 0;
440.     double maxValue = 0;
441.     int i;
442.     for (i = info.getCurrentAge(); i <= info.getDeathAge(); i++)
443.     {
444.         double v = info.getBalance(i);
445.         if (minValue > v) minValue = v;
446.         if (maxValue < v) maxValue = v;
447.     }
448.     if (maxValue == minValue) return;
449.
450.     int barWidth = getWidth() / (info.getDeathAge() - info.getCurrentAge() + 1);
451.     double scale = getHeight() / (maxValue - minValue);
452.
453.     for (i = info.getCurrentAge(); i <= info.getDeathAge(); i++)
454.     {
455.         int x1 = (i - info.getCurrentAge()) * barWidth + 1;
456.         int y1;
457.         double v = info.getBalance(i);
458.         int height;
459.         int yOrigin = (int) (maxValue * scale);
460.
461.         if (v >= 0)
462.         {
463.             y1 = (int) ((maxValue - v) * scale);
464.             height = yOrigin - y1;
465.         }
466.         else
467.         {
468.             y1 = yOrigin;
469.             height = (int) (-v * scale);
470.         }
471.
472.         if (i < info.getRetireAge()) g2.setPaint(colorPre);
473.         else if (v >= 0) g2.setPaint(colorGain);
474.         else g2.setPaint(colorLoss);
475.         Rectangle2D bar = new Rectangle2D.Double(x1, y1, barWidth - 2, height);
476.         g2.fill(bar);
477.         g2.setPaint(Color.black);
478.         g2.draw(bar);
479.     }
480. }
481.
482. /**
483. * Sets the color to be used before retirement.
484. * @param color the desired color
485. */
```

```
486.     public void setColorPre(Color color)
487.     {
488.         colorPre = color;
489.         repaint();
490.     }
491.
492.     /**
493.      * Sets the color to be used after retirement while the account balance is positive.
494.      * @param color the desired color
495.     */
496.     public void setColorGain(Color color)
497.     {
498.         colorGain = color;
499.         repaint();
500.     }
501.
502.     /**
503.      * Sets the color to be used after retirement when the account balance is negative.
504.      * @param color the desired color
505.     */
506.     public void setColorLoss(Color color)
507.     {
508.         colorLoss = color;
509.         repaint();
510.     }
511.
512.     private RetireInfo info = null;
513.     private Color colorPre;
514.     private Color colorGain;
515.     private Color colorLoss;
516.     private static final int PANEL_WIDTH = 400;
517.     private static final int PANEL_HEIGHT = 200;
518. }
```

### 程序清单5-6 RetireResources.java

```
1. import java.awt.*;
2.
3. /**
4.  * These are the English non-string resources for the retirement calculator.
5.  * @version 1.21 2001-08-27
6.  * @author Cay Horstmann
7. */
8. public class RetireResources extends java.util.ListResourceBundle
9. {
10.     public Object[][][] getContents()
11.     {
12.         return contents;
13.     }
14.
15.     static final Object[][][] contents = {
16.         // BEGIN LOCALIZE
17.         { "colorPre", Color.blue }, { "colorGain", Color.white }, { "colorLoss", Color.red }
18.         // END LOCALIZE
19.     };
20. }
```

**程序清单5-7 RetireResources\_de.java**

```
1. import java.awt.*;
2.
3. /**
4. * These are the German non-string resources for the retirement calculator.
5. * @version 1.21 2001-08-27
6. * @author Cay Horstmann
7. */
8. public class RetireResources_de extends java.util.ListResourceBundle
9. {
10.     public Object[][] getContents()
11.     {
12.         return contents;
13.     }
14.
15.     static final Object[][] contents = {
16.         // BEGIN LOCALIZE
17.         { "colorPre", Color.yellow }, { "colorGain", Color.black }, { "colorLoss", Color.red }
18.         // END LOCALIZE
19.     };
20. }
```

**程序清单5-8 RetireResources\_zh.java**

```
1. import java.awt.*;
2.
3. /**
4. * These are the Chinese non-string resources for the retirement calculator.
5. * @version 1.21 2001-08-27
6. * @author Cay Horstmann
7. */
8. public class RetireResources_zh extends java.util.ListResourceBundle
9. {
10.     public Object[][] getContents()
11.     {
12.         return contents;
13.     }
14.
15.     static final Object[][] contents = {
16.         // BEGIN LOCALIZE
17.         { "colorPre", Color.red }, { "colorGain", Color.blue }, { "colorLoss", Color.yellow }
18.         // END LOCALIZE
19.     };
20. }
```

**程序清单5-9 RetireStrings.properties**

```
1. language=Language
2. computeButton=Compute
3. savings=Prior Savings
4. contrib=Annual Contribution
5. income=Retirement Income
6. currentAge=Current Age
7. retireAge=Retirement Age
8. deathAge=Life Expectancy
```

---

```

9. inflationPercent=Inflation
10. investPercent=Investment Return
11. retire=Age: {0,number} Balance: {1,number,currency}

```

---

### 程序清单5-10 RetireStrings\_de.properties

```

1. language=Sprache
2. computeButton=Rechnen
3. savings=Vorherige Ersparnisse
4. contrib=J\u00e4hrliche Einzahlung
5. income=Einkommen nach Ruhestand
6. currentAge=Jetziges Alter
7. retireAge=Ruhestandsalter
8. deathAge=Lebenserwartung
9. inflationPercent=Inflation
10. investPercent=Investitionsgewinn
11. retire=Alter: {0,number} Guthaben: {1,number,currency}

```

---

### 程序清单5-11 RetireStrings\_zh.properties

```

1. language=\u8bed\u8a00
2. computeButton=\u8ba1\u7b97
3. savings=\u65e2\u5b58
4. contrib=\u6bcf\u5e74\u5b58\u91d1
5. income=\u9000\u4f11\u6536\u5165
6. currentAge=\u73b0\u9f84
7. retireAge=\u9000\u4f11\u5e74\u9f84
8. deathAge=\u9884\u671f\u5bff\u547d
9. inflationPercent=\u901a\u8d27\u81a8\u6da8
10. investPercent=\u6295\u8d44\u62a5\u916c
11. retire=\u5e74\u9f84: {0,number} \u603b\u7ed3: {1,number,currency}

```

---

### API java.applet.Applet 1.0

- Locale getLocale() 1.1

得到Applet的当前locale。当前locale取决于执行这个Applet的客户计算机。

本章进述了如何使用Java语言的国际化特征。你可以使用提供转换成多种语言的资源包，也可以使格式器和排序器来处理特定locale的文本。

下一章将讲述高级Swing编程。

# 第6章 高级Swing

- ▲ 列表
- ▲ 表格
- ▲ 树

- ▲ 文本构件
- ▲ 进度指示器
- ▲ 构件组织器

在本章中，我们继续对第 I 卷的 Swing 用户界面工具包进行讨论。Swing 是功能丰富的工具包，而本书第 I 卷仅仅涉及了若干简单而常用的构件。所以本章的大部分内容，将研究余下三个最为丰富复杂的构件：列表、树和表格。然后我们转向文本构件，讨论那些超越第 I 卷中看到的简单的文本框和文本域的特性，我们将展示如何在文本框上添加校验和微调框，以及如何显示诸如 HTML 这样的结构化文本。接下来，你还将看到大量用于显示耗时行为的进度的构件。在本章的最后，我们将介绍一些构件组织器，比如标签面板和带有内部框架的桌面面板。

## 6.1 列表

如果你想向用户提供一个选项集，而单选按钮或复选框又显得占用了太多的空间，那么就可以使用组合框或列表。组合框相对简单，已经在卷 I 中介绍过。JList 构件的功能更加丰富，而且它的设计与树形构件和表格构件都很相似。所以，对于复杂 Swing 构件的讨论，我们将从 JList 开始。

当然，可以使用字符串列表，但也可以使用任意对象的列表，这样能够完全控制它们的外在显示形式。正是列表控件的这种内部结构，使它不仅具备极强的通用性，而且也更精巧。遗憾的是，Sun 公司的设计人员认为他们更应该炫耀这种精巧，而不是将它对那些只是想使用这些构件的程序员隐藏起来。大家很快就会发现，在通常情况下，这种列表控制有点不太灵活，因为需要一些可以在通常条件下使用的操作机制。我们先介绍最简单、最常用的情况，即字符串列表框，然后给出一个更复杂的例子来展示列表构件的灵活性。

### 6.1.1 JList 构件

JList 可以将多个选项放置在单个框中，图 6-1 是一个大家公认的不合理的例子。用户可以选择狐狸的属性，比如“quick (敏捷的)”、“brown (棕色的)”、“hungry (饥饿的)”、“wild (野生的)”，并且由于属性已经完全选择了，所以它是“static (静态的)”、“private (私有的)”和“final (最终的)”。结果，你可以让这只 static 的、final 的狐狸从那只懒狗身上

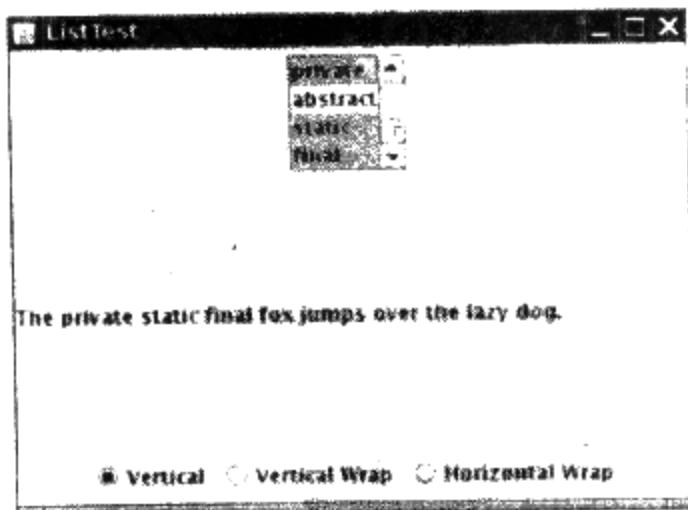


图 6-1 一个列表框

跳过去了。

为了构建这个列表框，首先需要创建一个字符串数组，然后将这个数组传递给JList构造器。

```
String[] words = { "quick", "brown", "hungry", "wild", ... };
JList wordList = new JList(words);
```

或者，还可以使用一个匿名数组：

```
JList wordList = new JList(new String[] {"quick", "brown", "hungry", "wild", ...});
```

列表框不能自动滚动，要想为列表框加上滚动条，必须将它插入到一个滚动面板中：

```
JScrollPane scrollPane = new JScrollPane(wordList);
```

然后应该把滚动面板而不是列表框，插入到外围面板上。

我们必须承认，从理论上讲，把列表框的显示和滚动机制隔离开来是优雅的设计，但是在实际应用中却令人苦不堪言，其实我们遇到的所有列表框基本上都需要滚动功能。强制程序员在默认情况下每次都去作这种麻烦事，以使他们赞赏这种优雅设计，确实有点残忍。

默认情况下，列表框构件可以显示8个选项，可以使用setVisibleRowCount方法改变这个值：

```
wordList.setVisibleRowCount(4); // display 4 items
```

还可以使用以下三个值中的任意一个来设置列表框摆放的方向：

- `JList.VERTICAL`（默认值）：垂直摆放所有选项。
- `JList.VERTICAL_WRAP`：如果选项数超过了可视行数，就开始新的一列（参见图6-2）。

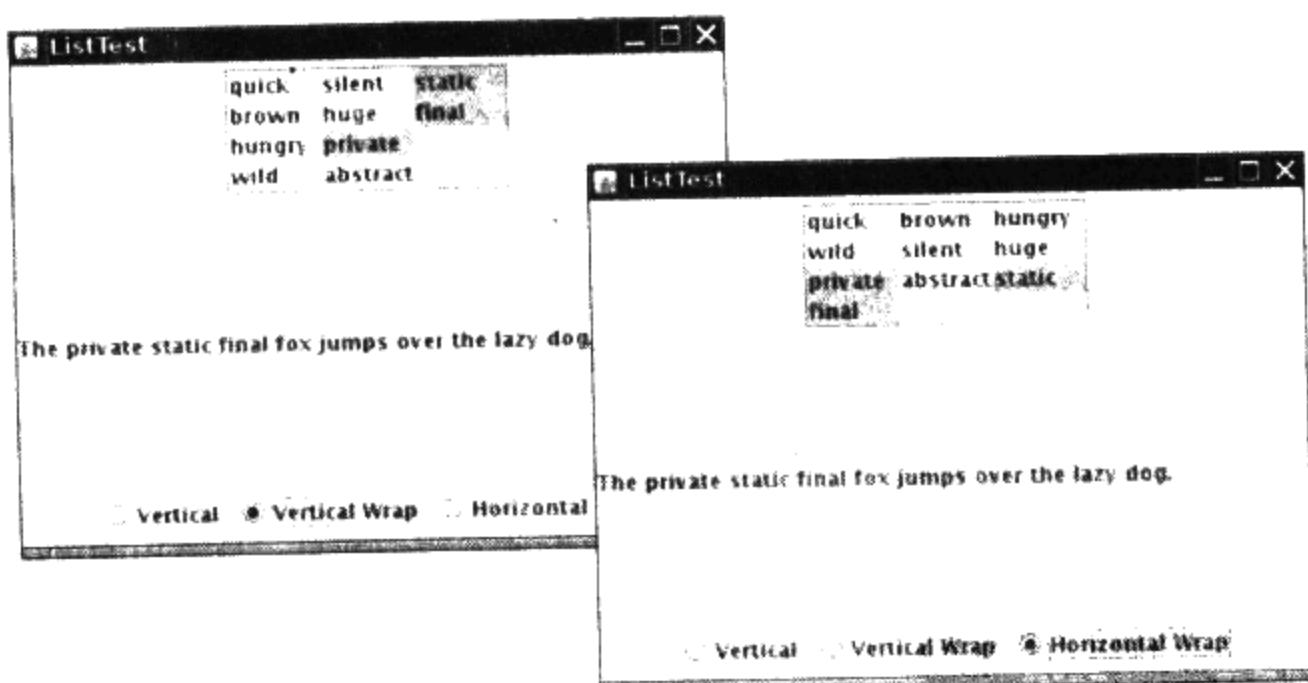


图6-2 带有垂直和水平换行的列表框

- `JList.HORIZONTAL_WRAP`：如果选项数超过了可视行数，就开始新的一行，并且按照水平方向进行填充。请观察图6-2中单词“quick”，“brown”和“hungry”的位置，以弄清楚垂直换行和水平换行的不同。

在默认情况下，用户可以选择多个选项。为了添加多个选项，只需按住CTRL键，然后在要选择的选项上单击。要选择处于连续范围内的选项，首先选择第一个选项，然后按住SHIFT键，并在最后一个选项上单击即可。

使用`setSelectionMode`方法，还可以对用户的选择模式加以限制：

```
wordList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    // select one item at a time
wordList.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
    // select one item or one range of items
```

也许你还记得，在卷I中提到，当用户激活了基本的用户界面构件时，它们将发出动作事件。列表框使用另一种不同的事件通知机制。不需要监听动作事件，而是监听列表选择事件。可以向列表构件添加一个列表选择监听器，然后在监听器中实现下面这个方法：

```
public void valueChanged(ListSelectionEvent evt)
```

在用户选择了若干个选项的同时，将产生一系列列表选择事件。假如用户在一个新选项上单击。当鼠标按下的时候，就会有一个事件来报告选项的改变。这是一种过渡型事件，在调用

```
event.isAdjusting()
```

时，如果该选择仍未最终结束则返回true。然后，当松开鼠标时，就产生另一事件，此时isAdjusting返回false。如果你对这种过渡型事件不感兴趣，那么可以等待isAdjusting调用返回false的事件。不过，如果希望只要点击鼠标就给用户一个即时反馈，那么就需要处理所有的事件。

一旦被告知一个事件已经发生，那么就需要弄清楚当前选择了哪些选项。方法getSelectedValues返回一个包含所有选中选项的对象数组。然后，应该将每个数组元素转型成一个字符串。

```
Object[] values = list.getSelectedValues();
for (Object value : values)
    do something with (String) value;
```



**警告：**你无法将getSelectedValues的返回值从一个Object[]数组转型成一个String[]数组。因为，这些返回值不是作为字符串数组而创建的，而是作为一个对象数组而创建的，只不过每个对象恰巧是一个字符串而已。如果是想要将这些返回值作为一个字符串数组来处理，你可以使用如下代码实现：

```
int length = values.length;
String[] words = new String[length];
System.arraycopy(values, 0, words, 0, length);
```

如果你的列表不允许选择多项，那么可以调用最方便的方法getSelectedItem。它仅返回第一个选择值（如果不允许多项选择，那么应该知道它就是惟一值）。

```
String value = (String) list.getSelectedItem();
```



**注意：**列表构件不响应鼠标的双击事件。正如Swing设计者所构想的那样，使用列表选择一个选项，然后点击某个按钮执行某个动作。但是，某些用户界面允许用户在一个列表选项上双击鼠标，作为选择一个选项并且接受一个默认动作的快捷方式。如果想实现这个动作，那么必须对这个列表框添加一个鼠标监听器，然后按照下面这样捕获鼠标事件：

```
public void mouseClicked(MouseEvent evt)
{
    if (evt.getClickCount() == 2)
```

```
    }
    JList source = (JList) evt.getSource();
    Object[] selection = source.getSelectedValues();
    doAction(selection);
}
}
```

程序清单6-1展示了一个填入了字符串的列表框。请注意valueChanged方法是怎样根据被选项来创建消息字符的。

### 程序清单6-1 ListTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.swing.*;
4. import javax.swing.event.*;
5.
6. /**
7. * This program demonstrates a simple fixed list of strings.
8. * @version 1.23 2007-08-01
9. * @author Cay Horstmann
10.*/
11. public class ListTest
12. {
13.     public static void main(String[] args)
14.     {
15.         EventQueue.invokeLater(new Runnable()
16.         {
17.             public void run()
18.             {
19.                 JFrame frame = new ListFrame();
20.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21.                 frame.setVisible(true);
22.             }
23.         });
24.     }
25. }
26.
27. /**
28. * This frame contains a word list and a label that shows a sentence made up from the chosen
29. * words. Note that you can select multiple words with Ctrl+click and Shift+click.
30.*/
31. class ListFrame extends JFrame
32. {
33.     public ListFrame()
34.     {
35.         setTitle("ListTest");
36.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
37.
38.         String[] words = { "quick", "brown", "hungry", "wild", "silent", "huge", "private",
39.             "abstract", "static", "final" };
40.
41.         wordList = new JList(words);
42.         wordList.setVisibleRowCount(4);
43.         JScrollPane scrollPane = new JScrollPane(wordList);
44. }
```

```
45.     listPanel = new JPanel();
46.     listPanel.add(scrollPane);
47.     wordList.addListSelectionListener(new ListSelectionListener()
48.     {
49.         public void valueChanged(ListSelectionEvent event)
50.         {
51.             Object[] values = wordList.getSelectedValues();
52.
53.             StringBuilder text = new StringBuilder(prefix);
54.             for (int i = 0; i < values.length; i++)
55.             {
56.                 String word = (String) values[i];
57.                 text.append(word);
58.                 text.append(" ");
59.             }
60.             text.append(suffix);
61.
62.             label.setText(text.toString());
63.         }
64.     });
65.
66.     buttonPanel = new JPanel();
67.     group = new ButtonGroup();
68.     makeButton("Vertical", JList.VERTICAL);
69.     makeButton("Vertical Wrap", JList.VERTICAL_WRAP);
70.     makeButton("Horizontal Wrap", JList.HORIZONTAL_WRAP);
71.
72.     add(listPanel, BorderLayout.NORTH);
73.     label = new JLabel(prefix + suffix);
74.     add(label, BorderLayout.CENTER);
75.     add(buttonPanel, BorderLayout.SOUTH);
76. }
77.
78. /**
79. * Makes a radio button to set the layout orientation.
80. * @param label the button label
81. * @param orientation the orientation for the list
82. */
83. private void makeButton(String label, final int orientation)
84. {
85.     JRadioButton button = new JRadioButton(label);
86.     buttonPanel.add(button);
87.     if (group.getButtonCount() == 0) button.setSelected(true);
88.     group.add(button);
89.     button.addActionListener(new ActionListener()
90.     {
91.         public void actionPerformed(ActionEvent event)
92.         {
93.             wordList.setLayoutOrientation(orientation);
94.             listPanel.revalidate();
95.         }
96.     });
97. }
98.
99. private static final int DEFAULT_WIDTH = 400;
100. private static final int DEFAULT_HEIGHT = 300;
```

```

101. private JPanel listPanel;
102. private JList wordList;
103. private JLabel label;
104. private JPanel buttonPanel;
105. private ButtonGroup group;
106. private String prefix = "The ";
107. private String suffix = "fox jumps over the lazy dog.";
108 }

```

### **API** javax.swing.JList 1.2

- **JList(Object[] items)**

构建一个显示这些选项 (item) 的列表。

- **int getVisibleRowCount()**

- **void setVisibleRowCount(int c)**

获取或设置列表在没有滚动条时显示的默认行数。

- **int getLayoutOrientation() 1.4**

- **void setLayoutOrientation(int orientation) 1.4**

获取或设置方向布局

参数: orientation VERTICAL、VERTICAL\_WRAP、HORIZONTAL\_WRAP其中之一

- **int getSelectionMode()**

- **void setSelectionMode(int mode)**

获取或设置选择方式是单个选项或多个选项。

参数: mode SINGLE\_SELECTION、SINGLE\_INTERVAL\_SELECTION、MULTIPLE\_INTERVAL\_SELECTION其中之一

- **void addListSelectionListener(ListSelectionListener listener)**

向列表添加一个在每次选择结果发生变化时会被告知的监听器。

- **Object[] getSelectedValues()**

返回所有的选定值, 如果选择结果为空, 则返回一个空的数组。

- **Object getSelectedValue()**

返回第一个选定值, 如果选择结果为空, 则返回null。

### **API** javax.swing.event.ListSelectionListener 1.2

- **void valueChanged(ListSelectionEvent e)**

在任何时刻, 只要选择结果发生了改变, 该方法就会被调用。

## 6.1.2 列表模式

通过前一节, 我们已经对列表构件的一些最常用的方法有了一定的了解:

1. 指定一组在列表中显示的固定字符串。

2. 将列表放置到一个滚动面板中。

3. 捕获列表选择事件。

在有关列表的章节的余下部分，我们将介绍一些需要一点技巧才能处理的复杂情形：

- 很长的列表。
- 内容会发生变化的列表。
- 不包含字符串的列表。

在第一个例子中，我们构建的那个JList构件包含固定不变的字符串集合。不过，列表框中的选项并非只能固定不变。那么我们应该怎样添加或删除列表框中的选项呢？令人有点吃惊的是，JList类并未提供关于方面的任何实现。相反地，需要进一步了解列表构件的内部设计。列表构件使用了模型—视图—控制器这种设计模式，将可视化外观（以某种方式呈现的一列选项）和底层数据（一个对象集合）进行了分离。

JList类负责数据的可视化外观。实际上，它对这些数据是怎样存储的知之甚少，它只知道可以通过某个实现了ListModel接口的对象来获取这些数据：

```
public interface ListModel
{
    int getSize();
    Object getElementAt(int i);
    void addListDataListener(ListDataListener l);
    void removeListDataListener(ListDataListener l);
}
```

通过这个接口，JList就可以获得元素的个数，并且能够获取每一个元素。另外，JList对象可以将其自身添加为一个ListDataListener。在这种方式下，一旦元素集合发生了变化，就会通知JList，从而使它能够重新绘制列表。

为什么这种通用性非常有用呢？为什么JList对象不直接存储一个对象数组呢？

请注意，这个接口并未指定这些对象是怎样存储的。尤其是，它根本就没有强制要求这些对象一定要被存储！无论何时调用getElementAt方法，它都会对每个值进行重新计算。如果想显示一个极大的集合，而且又不想存储这些值，那么这个方法可能会有所帮助。

这里举一个有点无聊的示例：允许用户在列表框中所有三个字母的单词当中进行选择（参见图6-3）。

三个字母的组合一共有 $26 \times 26 \times 26 = 17\,576$ 个。我们不希望将所有这些组合都存储起来，而是想在用户滚动这些单词的时候，依照请求对它们重新计算。

事实证明，这实现起来很容易。其中比较麻烦的部分，即添加和删除监听器，在我们所继承的AbstractListModel类中已经为我们实现了。我们只需要提供getSize和getElementAt方法便可：

```
class WordListModel extends AbstractListModel
{
```

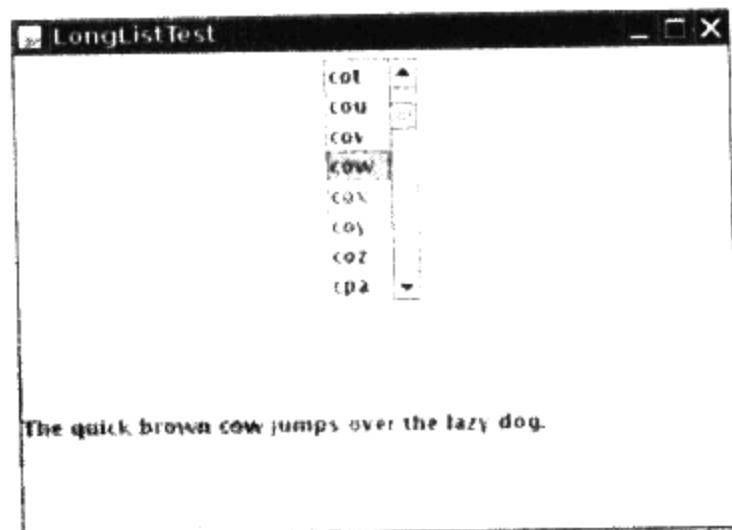


图6-3 从相当长的选项列表中选择

```

public WordListModel(int n) { length = n; }
public int getSize() { return (int) Math.pow(26, length); }
public Object getElementAt(int n)
{
    // compute nth string
    ...
}
...
}

```

对第n个字符串的计算需要一点技巧，在程序清单6-2中将看到具体实现。

既然我们已经提供了一个模型，那么，接下来我们就可以构建一个列表，让用户滚动选择该模型所提供的所有元素：

```

JList wordList = new JList(new WordListModel(3));
wordList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
JScrollPane scrollPane = new JScrollPane(wordList);

```

这里的关键是这些字符串从来没有被存储过，而只有那些用户实际要求查看的字符串才会被生成。

我们还必须进行另一项设置。那就是，我们必须告诉列表构件，所有的选项都有一个固定的宽度和高度。最简单的方法就是通过设置单元格的尺寸大小（cell dimension）来设定原型单元格的值（prototype cell value）：

```
wordList.setPrototypeCellValue("www");
```

原型单元格的值通常用来确定所有单元格的尺寸（我们使用字符串“www”是因为“w”在大多数字体中都是最宽的小写字母）。另外，可以像下面这样设置一个固定不变的单元格尺寸：

```

wordList.setFixedCellWidth(50);
wordList.setFixedCellHeight(15);

```

如果你既没有设置原型值也没有设置固定的单元格尺寸，那么列表构件就必须计算每个选项的宽度和高度。这可能需要花费更长时间。

从实际情况来看，这种很长的列表没有什么实用价值。让用户滚动浏览一个巨大的选项列表会显得相当得笨重和不便。正因为如此，我们认为这种列表控制设计得有点过火。用户能够在屏幕上舒服操作的选项列表肯定足够小，因此，可以直接存储到列表构件中。这种做法可以将编程人员解脱出来，使他们不必把列表模型作为一个单独实体进行处理。另一方面，JList类与Jtree、JTable这两个类也保持了一致，而在那里，通用性往往显得很有用。

### 程序清单6-2 LongListTest.java

```

1. import java.awt.*;
2.
3. import javax.swing.*;
4. import javax.swing.event.*;
5.
6. /**
7. * This program demonstrates a list that dynamically computes list entries.
8. * @version 1.23 2007-08-01
9. * @author Cay Horstmann
10. */

```

```
11. public class LongListTest
12. {
13.     public static void main(String[] args)
14.     {
15.         EventQueue.invokeLater(new Runnable()
16.         {
17.             public void run()
18.             {
19.                 JFrame frame = new LongListFrame();
20.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21.                 frame.setVisible(true);
22.             }
23.         });
24.     }
25. }
26.
27. /**
28. * This frame contains a long word list and a label that shows a sentence made up from
29. * the chosen word.
30. */
31. class LongListFrame extends JFrame
32. {
33.     public LongListFrame()
34.     {
35.         setTitle("LongListTest");
36.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
37.
38.         wordList = new JList(new WordListModel(3));
39.         wordList.setSelectionMode(ListSelectionMode.SINGLE_SELECTION);
40.         wordList.setPrototypeCellValue("www");
41.         JScrollPane scrollPane = new JScrollPane(wordList);
42.
43.         JPanel p = new JPanel();
44.         p.add(scrollPane);
45.         wordList.addListSelectionListener(new ListSelectionListener()
46.         {
47.             public void valueChanged(ListSelectionEvent evt)
48.             {
49.                 StringBuilder word = (StringBuilder) wordList.getSelectedValue();
50.                 setSubject(word.toString());
51.             }
52.
53.         });
54.
55.         Container contentPane = getContentPane();
56.         contentPane.add(p, BorderLayout.NORTH);
57.         label = new JLabel(prefix + suffix);
58.         contentPane.add(label, BorderLayout.CENTER);
59.         setSubject("fox");
60.     }
61.
62. /**
63. * Sets the subject in the label.
64. * @param word the new subject that jumps over the lazy dog
65. */
66. public void setSubject(String word)
67. {
```

```
68.     StringBuilder text = new StringBuilder(prefix);
69.     text.append(word);
70.     text.append(suffix);
71.     label.setText(text.toString());
72. }
73.
74. private static final int DEFAULT_WIDTH = 400;
75. private static final int DEFAULT_HEIGHT = 300;
76. private JList wordList;
77. private JLabel label;
78. private String prefix = "The quick brown ";
79. private String suffix = " jumps over the lazy dog.";
80. }
81.
82. /**
83. * A model that dynamically generates n-letter words.
84. */
85. class WordListModel extends AbstractListModel
86. {
87. /**
88. * Constructs the model.
89. * @param n the word length
90. */
91. public WordListModel(int n)
92. {
93.     length = n;
94. }
95.
96. public int getSize()
97. {
98.     return (int) Math.pow(LAST - FIRST + 1, length);
99. }
100.
101. public Object getElementAt(int n)
102. {
103.     StringBuilder r = new StringBuilder();
104.     ;
105.     for (int i = 0; i < length; i++)
106.     {
107.         char c = (char) (FIRST + n % (LAST - FIRST + 1));
108.         r.insert(0, c);
109.         n = n / (LAST - FIRST + 1);
110.     }
111.     return r;
112. }
113.
114. private int length;
115. public static final char FIRST = 'a';
116. public static final char LAST = 'z';
117. }
```



## Javax.swing.JList 1.2

- **JList(ListModel dataModel)**

构建一个以指定模式显示其元素的列表。

- void setPrototypeCellValue(Object newValue)
- Object getPrototypeCellValue()

设置或获取用于设定列表中每一个单元格宽度和高度的原型单元格值。默认值为null，它将强制对每一个单元格的尺寸进行测量。

- void setFixedCellWidth(int width)

如果width大于0，则设定列表中每一个单元格的宽度（单位为像素）。默认值为-1，它将强制对每一个单元格的尺寸进行测量。

- void setFixedCellHeight(int height)

如果height大于0，则设定列表中每一个单元格的高度（单位为像素）。默认值为-1，它将强制对每一个单元格的尺寸进行测量。



### javax.swingListModel 1.2

- int getSize()

返回该模型中的元素个数。

- Object getElementAt(int position)

返回该模型中给定位置上的一个元素。

### 6.1.3 插入和移除值

不能直接编辑列表值的集合。相反地，必须先访问模型，然后再添加或移除元素。不过，说起来容易做起来难。假设想要向列表中添加更多的选项值。首先可以通过下面的语句获得对该模型的一个引用：

```
ListModel model = list.getModel();
```

但是，正如在前一小节中看到的那样，这样做并不能带来任何好处，因为ListModel接口并未提供任何插入或移除元素的方法。毕竟，列表模型的整个重点是它不需要存储任何元素。

让我们试试另一种方法吧。JList有一个构造器可以接受一个对象向量作为参数：

```
Vector<String> values = new Vector<String>();
values.addElement("quick");
values.addElement("brown");
...
JList list = new JList(values);
```

现在，就可以通过编辑这个向量来添加或移除元素了，不过列表并不知道正在发生的事情，因此也就无法对这种变化做出响应。尤其是，当你向列表中添加元素时，列表无法及时更新它的显示视图。所以这个构造器也不太实用。

取而代之的是，应该构建一个DefaultListModel对象，填入初始值，然后将它与一个列表关联起来。DefaultListModel类实现了ListModel接口，并管理着一个对象集合。

```
DefaultListModel model = new DefaultListModel();
model.addElement("quick");
model.addElement("brown");
...
JList list = new JList(model);
```

现在，就可以从model对象中添加或移除元素值。然后，model对象会告知列表发生了哪些变化，接着，列表会对自身进行重新绘制。

```
model.removeElement("quick");
model.addElement("slow");
```

由于历史遗留的问题，DefaultListModel类使用的方法名和集合类的方法名并不相同。默认的列表模型在内部是使用一个向量来存储元素值的。

**X 警告：**JList存在着多种构造器方法，可以用一个对象或字符串数组或向量来构建列表。你可能会认为这些构造器是使用一个DefaultListModel来存储这些元素值的。但情况并非如此，这些构造器构建了一个普通而简单的模型，它可以访问元素值，但是如果内容发生了改变，它并不提供任何通知机制。例如，下面这段代码是使用一个Vector来构造JList的构造器的代码：

```
public JList(final Vector<?> listData)
{
    this (new AbstractListModel()
    {
        public int getSize() { return listData.size(); }
        public Object getElementAt(int i) { return listData.elementAt(i); }
    });
}
```

这意味着，在列表被创建之后，如果要修改向量里面的内容，那么这个列表在被完全重新绘制之前，会将旧值和新值混在一起，杂乱无章地显示出来。（上面构造器中的关键字final并不能阻止你在其他地方对这个向量进行修改，它仅仅表示构造器本身不能修改listData的引用值；一定要有这个关键字是因为listData对象是在内部类中使用的。）

#### **API** javax.swing.JList 1.2

- `ListModel getModel()`

获取列表的模型。

#### **API** javax.swing.DefaultListModel 1.2

- `void addElement(Object obj)`
- 向该模型的末端添加一个对象。
- `boolean removeElement(Object obj)`

从模型中移除第一次出现的给定对象。如果该模型中包含此对象，则返回true，否则返回false。

### 6.1.4 值的绘制

到目前为止，我们在本章看到的列表都包含字符串。实际上只需传递一个用Icon对象填充的数组或向量，便可以很容易地显示一个图标列表。更有意思的是，可以很容易地用任何图形来表示你的列表值。

尽管JList类可以自动地显示字符串和图标，但是仍然需要在JList对象中安装一个用于所

有自定义图形的列表单元格绘制器。列表单元格绘制器可以是任何一个实现了下面接口的类：

```
interface ListCellRenderer
{
    Component getListCellRendererComponent(JList list, Object value, int index,
        boolean isSelected, boolean cellHasFocus);
}
```

这个方法可以供每个单元格调用，它返回一个用于绘制单元格内容的构件。无论何时，只要某个单元格需要被绘制，该构件就会被置于合适的位置。

实现单元格绘制器的一种方法是创建一个扩展了JComponent的类，如下所示：

```
class MyCellRenderer extends JComponent implements ListCellRenderer
{
    public Component getListCellRendererComponent(JList list, Object value, int index,
        boolean isSelected, boolean cellHasFocus)
    {
        // stash away information that is needed for painting and size measurement
        return this;
    }
    public void paintComponent(Graphics g)
    {
        // paint code goes here
    }
    public Dimension getPreferredSize()
    {
        // size measurement code goes here
    }
    // instance fields
}
```

在程序清单6-3中，我们按照字体的实际外观显示这些选择字体（参见图6-4）。在paintComponent方法内部，我们用该字体显示其自身的名称。我们也要确保JList类的外观与通常颜色相匹配。通过调用JList类中的getForeground/getBackground和getSelectionForeground/getSelectionBackground方法可以获取这些颜色。在getPreferredSize方法中，我们需要使用在卷I第7章中介绍的技术来测量字符串的大小。

如果要安装单元格绘制器，只需调用setCellRenderer方法即可：

```
fontList.setCellRenderer(new FontCellRenderer());
```

现在，列表中的所有单元格都是按照自定义的方式绘制的了。

实际上，可以用一种更简单的方法来编写在大多情况下都能运行的自定义绘制器。如果绘制的图像仅仅包含文本、图标或者变化颜色，那么通过配置一个JLabel就可以得到这样的一个绘制器。例如，为了用每种字体显示该字体自身的名称，我们可以使用下面的绘制器：

```
class FontCellRenderer extends JLabel implements ListCellRenderer
{
    public Component getListCellRendererComponent(JList list, Object value, int index,
```

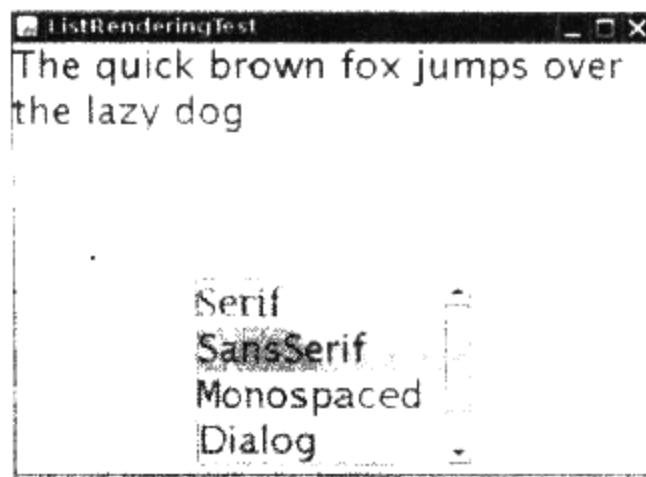


图6-4 具有绘画单元格的列表框

```

        boolean isSelected, boolean cellHasFocus)
    {
        JLabel label = new JLabel();
        Font font = (Font) value;
        setText(font.getFamily());
        setFont(font);
        setOpaque(true);
        setBackground(isSelected ? list.getSelectionBackground() : list.getBackground());
        setForeground(isSelected ? list.getSelectionForeground() : list.getForeground());
        return this;
    }
}

```

注意，这里没有编写任何paintComponent或getPreferredSize方法；JLabel类早已实现了这些方法，完全能够满足我们的要求。我们要做的全部工作就是通过设置文本、字体以及颜色来恰当地配置标签。

这段代码在某些情形下确实是一个很便利的捷径，因为在这些情形中，有现成的构件——JLabel，它已经提供了绘制单元格值所需的全部功能。

**X 警告：**在每一个getListCellRendererComponent调用中都构建一个新的构件并不是一个好主意。因为如果用户滚动了许多个列表项，那么每一次都需要构建一个新构件。而对已有构件进行重构则显得更安全更高效。

### 程序清单6-3 ListRenderingTest.java

```

1. import java.util.*;
2. import java.awt.*;
3.
4. import javax.swing.*;
5. import javax.swing.event.*;
6.
7. /**
8. * This program demonstrates the use of cell renderers in a list box.
9. * @version 1.23 2007-08-01
10. * @author Cay Horstmann
11. */
12. public class ListRenderingTest
13. {
14.     public static void main(String[] args)
15.     {
16.         EventQueue.invokeLater(new Runnable()
17.         {
18.             public void run()
19.             {
20.                 JFrame frame = new ListRenderingFrame();
21.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22.                 frame.setVisible(true);
23.             }
24.         });
25.     }
26. }
27.
28. /**

```

```
29. * This frame contains a list with a set of fonts and a text area that is set to the
30. * selected font.
31. */
32. class ListRenderingFrame extends JFrame
33. {
34.     public ListRenderingFrame()
35.     {
36.         setTitle("ListRenderingTest");
37.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
38.
39.         ArrayList<Font> fonts = new ArrayList<Font>();
40.         final int SIZE = 24;
41.         fonts.add(new Font("Serif", Font.PLAIN, SIZE));
42.         fonts.add(new Font("SansSerif", Font.PLAIN, SIZE));
43.         fonts.add(new Font("Monospaced", Font.PLAIN, SIZE));
44.         fonts.add(new Font("Dialog", Font.PLAIN, SIZE));
45.         fonts.add(new Font("DialogInput", Font.PLAIN, SIZE));
46.         fontList = new JList(fonts.toArray());
47.         fontList.setVisibleRowCount(4);
48.         fontList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
49.         fontList.setCellRenderer(new FontCellRenderer());
50.         JScrollPane scrollPane = new JScrollPane(fontList);
51.
52.         JPanel p = new JPanel();
53.         p.add(scrollPane);
54.         fontList.addListSelectionListener(new ListSelectionListener()
55.         {
56.             public void valueChanged(ListSelectionEvent evt)
57.             {
58.                 Font font = (Font) fontList.getSelectedValue();
59.                 text.setFont(font);
60.             }
61.
62.         });
63.
64.         Container contentPane = getContentPane();
65.         contentPane.add(p, BorderLayout.SOUTH);
66.         text = new JTextArea("The quick brown fox jumps over the lazy dog");
67.         text.setFont((Font) fonts.get(0));
68.         text.setLineWrap(true);
69.         text.setWrapStyleWord(true);
70.         contentPane.add(text, BorderLayout.CENTER);
71.     }
72.
73.     private JTextArea text;
74.     private JList fontList;
75.     private static final int DEFAULT_WIDTH = 400;
76.     private static final int DEFAULT_HEIGHT = 300;
77. }
78.
79. /**
80. * A cell renderer for Font objects that renders the font name in its own font.
81. */
82. class FontCellRenderer extends JComponent implements ListCellRenderer
83. {
84.     public Component getListCellRendererComponent(JList list, Object value, int index,
```

```

85.     boolean isSelected, boolean cellHasFocus)
86.     {
87.         font = (Font) value;
88.         background = isSelected ? list.getSelectionBackground() : list.getBackground();
89.         foreground = isSelected ? list.getSelectionForeground() : list.getForeground();
90.         return this;
91.     }
92.
93.     public void paintComponent(Graphics g)
94.     {
95.         String text = font.getFamily();
96.         FontMetrics fm = g.getFontMetrics(font);
97.         g.setColor(background);
98.         g.fillRect(0, 0, getWidth(), getHeight());
99.         g.setColor(foreground);
100.        g.setFont(font);
101.        g.drawString(text, 0, fm.getAscent());
102.    }
103.
104.    public Dimension getPreferredSize()
105.    {
106.        String text = font.getFamily();
107.        Graphics g = getGraphics();
108.        FontMetrics fm = g.getFontMetrics(font);
109.        return new Dimension(fm.stringWidth(text), fm.getHeight());
110.    }
111.
112.    private Font font;
113.    private Color background;
114.    private Color foreground;
115. }
```



### **javax.swing.JList 1.2**

- **Color getBackground()**  
返回未选定单元格的背景颜色。
- **Color getSelectionBackground()**  
返回选定单元格的背景颜色。
- **Color getForeground()**  
返回未选定单元格的前景颜色。
- **Color getSelectionForeground()**  
返回选定单元格的前景颜色。
- **void setCellRenderer(ListCellRenderer cellRenderer)**  
设置用于绘制列表中单元格的绘制器。



### **javax.swing.ListCellRenderer 1.2**

- **Component getListCellRendererComponent(JList list, Object item, int index, boolean isSelected, boolean hasFocus)**  
返回一个其paint方法用于绘制单元格内容的构件，如果列表的单元格尺寸没有固定，那

么该构件还必须实现`getPreferredSize`。

|            |                       |
|------------|-----------------------|
| 参数: list   | 单元格正在被绘制的列表           |
| item       | 要绘制的选项                |
| index      | 存储在模型中的选项索引           |
| isSelected | 如果指定的单元格被选定, 则返回true  |
| hasFocus   | 如果焦点在指定的单元格上, 则返回true |

## 6.2 表格

`JTable`构件用于显示二维对象表格。当然, 表格在用户界面中很常见。Swing开发小组将大量的精力投入到了表格控制方面。表格本身比较复杂, 但是它可能比其他Swing类更为成功, 因为`JTable`构件隐藏了更多的复杂性。只需编写几行代码就能够产生具有完全功能化的、行为丰富的表格。当然, 可以编写更多的代码, 为你的具体应用定制显示外观和运行特性。

在本节中, 我们将着重讲解怎样产生简单表格, 用户怎样与它们交互, 以及怎样进行一些最常见的调整操作。与其他一些复杂的Swing构件一样, 我们不可能覆盖所有的细节。如果想获得详细信息, 请查阅David M. Geary撰写的《Graphic Java 2: Mastering the JFC, Volume II: Swing》(第3版) 或Kim Topley撰写的《Core Java Foundation Classes》。

### 6.2.1 简单表格

与`JList`构件类似, `JTable`并不存储它自己的数据, 而是从一个表格模型中获取它的数据。`JTable`类有一个构造器能够将一个二维对象数组包装进一个默认的模型。这也正是我们第一个示例程序要用到的策略。在本章的后续部分, 我们将转向介绍表格模型。

图6-5展示了一个典型的表格, 用于描述太阳系各个行星的属性。(如果一个行星主要由氢气和氦气组成, 那么它就是气态行星。对于“Color”项, 你应该了解其巨大的作用, 我们之所以将它添加为一列是因为在后面的示例代码中, 它很有用的。)

正如你在程序清单6-4中见到的那样, 表格中的数据是以`Object`值的二维数组的形式来存储的:

```
Object[][] cells =
{
    { "Mercury", 2440.0, 0, false, Color.YELLOW },
    { "Venus", 6052.0, 0, false, Color.YELLOW },
    ...
}
```

图6-5 简单表格

注意: 这里, 我们充分利用了自动封装。第二列、第三列、第四列中的项会自动转换成类型为`Double`、`Integer`和`Boolean`的对象。

该表格直接调用每个对象上的`toString`方法来显示它们，这也是为什么颜色显示成为`java.awt.Color[r=...,g=...,b=...]`的原因所在。

可以用一个单独的字符串数组来提供列名：

```
String[] columnNames = { "Planet", "Radius", "Moons", "Gaseous", "Color" };
```

接着，就可以从单元格和列名数组中构建一个表格。最后，通过将表格包装到一个`JScrollPane`中这个常用方法来添加滚动条。

```
JTable table = new JTable(cells, columnNames);
JScrollPane pane = new JScrollPane(table);
```

这样产生的表格已经具有令人吃惊的丰富行为特性了。可以垂直调整表格的尺寸大小直到滚动条显现出来，然后滚动表格。请注意，列表头并不会滑出视图的外面。

接着，单击一个列表头，并且向左或右拖拉。看看整个列是怎样移开的（参见图6-6），你可以将它放到别的位置上。这种列的重新排列只是视图上的重新排列，对数据模型没有任何影响。

如果要调整列的尺寸大小，只需将鼠标移到两列之间，直到鼠标的形状变成箭头为止，然后将列的边界拖移到你期望的位置上（参见图6-7）。

图6-6 移动表格中的一列

图6-7 调整列的尺寸大小

用户可以通过点击行中任何一个地方来选中一行，而选中的行会加亮显示，后面将会介绍怎样获取这些选择事件。通过单击一个单元格并键入数据，用户还可以编辑表格中的各个项。不过，在这个代码示例中，这些编辑并没有改变底层的数据。在程序中，你应该要么使这些单元格不可编辑，要么处理单元格编辑事件并更新你的模型。我们将会在本节的后面对这些问题进行讨论。

最后，点击列的头，行就会自动排序。如果再次点击，排序顺序就会反过来。这个行为是通过下面的调用激活的：

```
table.setAutoCreateRowSorter(true);
```

可以使用下面的调用对表格进行打印：

```
table.print();
```

此时会出现一个打印对话框，并将表格传送给打印机。我们将在第7章讨论定制打印选项。



**注意：**如果调整`TableTest`框架的尺寸，使它的高度超过了表的高度，那么就会看到表的下方有一块灰色区域。与`JList`和`JTree`构件不同，表没有填充滚动面板视图。当希望支持拖拽时，这可能会成为一个问题（关于拖拽的更多信息，请查看第7章）。在这种情况下，可以调用

```
table.setFillViewport(true);
```

**程序清单6-4 PlanetTable.java**

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.swing.*;
4.
5. /**
6. * This program demonstrates how to show a simple table
7. * @version 1.11 2007-08-01
8. * @author Cay Horstmann
9. */
10. public class PlanetTable
11. {
12.     public static void main(String[] args)
13.     {
14.         EventQueue.invokeLater(new Runnable()
15.         {
16.             public void run()
17.             {
18.                 JFrame frame = new PlanetTableFrame();
19.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20.                 frame.setVisible(true);
21.             }
22.         });
23.     }
24. }
25.
26. /**
27. * This frame contains a table of planet data.
28. */
29. class PlanetTableFrame extends JFrame
30. {
31.     public PlanetTableFrame()
32.     {
33.         setTitle("PlanetTable");
34.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
35.         final JTable table = new JTable(cells, columnNames);
36.         table.setAutoCreateRowSorter(true);
37.         add(new JScrollPane(table), BorderLayout.CENTER);
38.         JButton printButton = new JButton("Print");
39.         printButton.addActionListener(new ActionListener()
40.         {
41.             public void actionPerformed(ActionEvent event)
42.             {
43.                 try
44.                 {
45.                     table.print();
46.                 }
47.                 catch (java.awt.print.PrinterException e)
48.                 {
49.                     e.printStackTrace();
50.                 }
51.             }
52.         });
53.         JPanel buttonPanel = new JPanel();
54.         buttonPanel.add(printButton);
55.         add(buttonPanel, BorderLayout.SOUTH);
```

```

56. }
57.
58. private Object[][] cells = { { "Mercury", 2440.0, 0, false, Color.yellow },
59.     { "Venus", 6052.0, 0, false, Color.yellow }, { "Earth", 6378.0, 1, false, Color.blue },
60.     { "Mars", 3397.0, 2, false, Color.red }, { "Jupiter", 71492.0, 16, true, Color.orange },
61.     { "Saturn", 60268.0, 18, true, Color.orange }, { "Uranus", 25559.0, 17, true, Color.blue }, { "Neptune", 24766.0, 8, true, Color.blue },
62.     { "Pluto", 1137.0, 1, false, Color.black } };
63.
64.
65. private String[] columnNames = { "Planet", "Radius", "Moons", "Gaseous", "Color" };
66.
67. private static final int DEFAULT_WIDTH = 400;
68. private static final int DEFAULT_HEIGHT = 200;
69. }

```



## javax.swing.JTable 1.2

- `JTable(Object[][] entries, Object[] columnNames)`

用默认的表格模型构建一个表格。

- `void print() 5.0`

显示打印对话框，并打印该表格。

- `boolean getAutoCreateRowSorter() 6`

- `void setAutoCreateRowSorter(boolean newValue) 6`

获取或设置autoCreateRowSorter属性，默认值为false。如果进行了设置，只要模型发生变化，就会自动设置一个默认的行排序器。

- `boolean getFillsViewportHeight() 6`

- `void setFillsViewportHeight(boolean newValue) 6`

获取或设置fillsViewportHeight 属性，默认值为false。如果进行了设置，该表格就总是会填充一个封装在外边的视图。

### 6.2.2 表格模型

在上一个示例中，表格数据是存储在一个二维数组中的。不过，通常不应该在自己的代码中使用这种策略。如果你发现自己在将数据装入一个数组中，然后作为一个表格显示出来，那么就应该考虑实现自己的表格模型了。

表格模型实现起来特别简单，因为你可以充分利用AbstractTableModel类，它实现了大部分必需的方法。你仅仅需要提供下面三个方法便可：

```

public int getRowCount();
public int getColumnCount();
public Object getValueAt(int row, int column);

```

实现getValueAt方法有多种途径。例如，如果你想显示包含数据库查询结果的RowSet的内容，只需提供下面的方法：

```

public Object getValueAt(int r, int c)
{
    try

```

```

    {
        rowSet.absolute(r + 1);
        return rowSet.getObject(c + 1);
    }
    catch (SQLException e)
    {
        e.printStackTrace();
        return null;
    }
}

```

我们的示例程序相当简单，我们构建了一个只是用来显示某些计算结果的表格，这些计算结果也就是在不同利率条件下的投资增长额（参见图6-8）。

|    | 5%       | 6%        | 7%        | 8%        | 9%        | 10%       |
|----|----------|-----------|-----------|-----------|-----------|-----------|
| 1  | 10000.00 | 106180.00 | 112950.00 | 119710.00 | 126550.00 | 133490.00 |
| 2  | 10500.00 | 112360.00 | 119130.00 | 125900.00 | 132700.00 | 140500.00 |
| 3  | 11000.00 | 118750.00 | 124430.00 | 130110.00 | 136810.00 | 143610.00 |
| 4  | 11500.00 | 125340.00 | 130510.00 | 136090.00 | 143210.00 | 150610.00 |
| 5  | 12000.00 | 132140.00 | 136700.00 | 142280.00 | 150590.00 | 157900.00 |
| 6  | 12500.00 | 139140.00 | 143000.00 | 148670.00 | 158100.00 | 165400.00 |
| 7  | 13000.00 | 146340.00 | 149400.00 | 155260.00 | 165810.00 | 172900.00 |
| 8  | 13500.00 | 153740.00 | 155900.00 | 162040.00 | 173630.00 | 180400.00 |
| 9  | 14000.00 | 161340.00 | 162500.00 | 168930.00 | 181550.00 | 188000.00 |
| 10 | 14500.00 | 169140.00 | 169200.00 | 175930.00 | 189580.00 | 195500.00 |
| 11 | 15000.00 | 177140.00 | 176000.00 | 183040.00 | 197710.00 | 203000.00 |
| 12 | 15500.00 | 185340.00 | 182900.00 | 190260.00 | 205950.00 | 210200.00 |
| 13 | 16000.00 | 193740.00 | 189800.00 | 197600.00 | 214300.00 | 217200.00 |
| 14 | 16500.00 | 202340.00 | 196700.00 | 205040.00 | 223750.00 | 224500.00 |
| 15 | 17000.00 | 211140.00 | 203600.00 | 212590.00 | 233300.00 | 234870.00 |
| 16 | 17500.00 | 220140.00 | 210500.00 | 220240.00 | 243950.00 | 245700.00 |
| 17 | 18000.00 | 229340.00 | 217400.00 | 228090.00 | 254700.00 | 257500.00 |
| 18 | 18500.00 | 238740.00 | 224300.00 | 236040.00 | 265550.00 | 269300.00 |
| 19 | 19000.00 | 248340.00 | 231200.00 | 244100.00 | 276450.00 | 273100.00 |
| 20 | 19500.00 | 258140.00 | 238100.00 | 252260.00 | 287450.00 | 280900.00 |
| 21 | 20000.00 | 268140.00 | 245000.00 | 260510.00 | 298550.00 | 288700.00 |
| 22 | 20500.00 | 278340.00 | 251900.00 | 268860.00 | 309750.00 | 296500.00 |
| 23 | 21000.00 | 288740.00 | 258800.00 | 277310.00 | 321050.00 | 304300.00 |
| 24 | 21500.00 | 299340.00 | 265600.00 | 285860.00 | 332450.00 | 312100.00 |
| 25 | 22000.00 | 310140.00 | 272400.00 | 294510.00 | 344050.00 | 320000.00 |
| 26 | 22500.00 | 321140.00 | 279200.00 | 303260.00 | 355750.00 | 327900.00 |
| 27 | 23000.00 | 332340.00 | 286000.00 | 312110.00 | 367650.00 | 335800.00 |
| 28 | 23500.00 | 343740.00 | 292800.00 | 321060.00 | 379650.00 | 343700.00 |
| 29 | 24000.00 | 355340.00 | 299600.00 | 330110.00 | 391750.00 | 351600.00 |
| 30 | 24500.00 | 367140.00 | 306400.00 | 339260.00 | 404050.00 | 359500.00 |

图6-8 一个投资增长额表格

`getValueAt`方法计算出正确值，并将其格式化：

```

public Object getValueAt(int r, int c)
{
    double rate = (c + minRate) / 100.0;
    int nperiods = r;
    double futureBalance = INITIAL_BALANCE * Math.pow(1 + rate, nperiods);
    return String.format("%.2f", futureBalance);
}

```

`getRowCount`和`getColumnCount`方法只是返回行数和列数。

```

public int getRowCount() { return years; }
public int getColumnCount() { return maxRate - minRate + 1; }

```

如果不提供列名，那么`AbstractTableModel`的`getColumnName`方法会将列命名为A、B、C等。如果要改变列名，请覆盖`getColumnName`方法。通常需要覆盖默认的行为。在这个示例中，我们只是将每列用利率标识了出来。

```
public String getColumnName(int c) { return (c + minRate) + "%"; }
```

程序清单6-5中显示了完整的源代码。

### 程序清单6-5 InvestmentTable.java

```

1 import java.awt.*;
2
3 import javax.swing.*;

```

```
4 import javax.swing.table.*;
5
6 /**
7 * This program shows how to build a table from a table model.
8 * @version 1.02 2007-08-01
9 * @author Cay Horstmann
10 */
11 public class InvestmentTable
12 {
13     public static void main(String[] args)
14     {
15         EventQueue.invokeLater(new Runnable()
16         {
17             public void run()
18             {
19                 JFrame frame = new InvestmentTableFrame();
20                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21                 frame.setVisible(true);
22             }
23         });
24     }
25 }
26
27 /**
28 * This frame contains the investment table.
29 */
30 class InvestmentTableFrame extends JFrame
31 {
32     public InvestmentTableFrame()
33     {
34         setTitle("InvestmentTable");
35         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
36
37         TableModel model = new InvestmentTableModel(30, 5, 10);
38         JTable table = new JTable(model);
39         add(new JScrollPane(table));
40     }
41
42     private static final int DEFAULT_WIDTH = 600;
43     private static final int DEFAULT_HEIGHT = 300;
44 }
45
46 /**
47 * This table model computes the cell entries each time they are requested. The table contents
48 * shows the growth of an investment for a number of years under different interest rates.
49 */
50 class InvestmentTableModel extends AbstractTableModel
51 {
52     /**
53      * Constructs an investment table model.
54      * @param y the number of years
55      * @param r1 the lowest interest rate to tabulate
56      * @param r2 the highest interest rate to tabulate
57      */
58     public InvestmentTableModel(int y, int r1, int r2)
59     {
```

```
60.     years = y;
61.     minRate = r1;
62.     maxRate = r2;
63. }
64.
65. public int getRowCount()
66. {
67.     return years;
68. }
69.
70. public int getColumnCount()
71. {
72.     return maxRate - minRate + 1;
73. }
74.
75. public Object getValueAt(int r, int c)
76. {
77.     double rate = (c + minRate) / 100.0;
78.     int nperiods = r;
79.     double futureBalance = INITIAL_BALANCE * Math.pow(1 + rate, nperiods);
80.     return String.format("%.2f", futureBalance);
81. }
82.
83. public String getColumnName(int c)
84. {
85.     return (c + minRate) + "%";
86. }
87.
88. private int years;
89. private int minRate;
90. private int maxRate;
91.
92. private static double INITIAL_BALANCE = 100000.0;
93. }
```

### API javax.swing.table.TableModel 1.2

- `int getCount()`  
获取表模型中的行和列的数量。
- `Object getValueAt(int row, int column)`  
获取在给定的行和列所确定的位置处的值。
- `void setValueAt(Object newValue, int row, int column)`  
设置在给定的行和列所确定的位置处的值。
- `boolean isCellEditable(int row, int column)`  
如果在给定的行和列所确定的位置处的值是可编辑的，则返回`true`。
- `String getColumnName(int column)`  
获取列的名字。

### 6.2.3 对行和列的操作

在本小节中，你会看到怎样操作一个表格中的行和列。在你阅读本材料的整个过程中，要牢记Swing中的表格是相当不对称的，也就是你可以实施的行操作和列操作会有所不同。表格构件已经被优化过，以便能够显示具有相同结构的行信息，例如一次数据库查询的结果，而不是任意的二维对象表格。你将会看到，这种不对称性贯穿于本小节。

#### 各种列类

在下一个示例中，我们将再次展示行星数据，不过这次我们会赋予表格更多的有关列类型的信息。这是通过定义下面这个方法来实现的：

```
Class<?> getColumnClass(int columnIndex)
```

这个方法可以返回一个描述列类型的类。

JTable类会为该类选取合适的绘制器。表6-1显示了JTable类是怎样在默认情况下绘制类型的。

可以在图6-33中看到复选框和图像。（感谢Jim Evins提供了这些行星图像，网址为：<http://www.snaught.com/JimsCoolIcons/Planets。>）

表6-1 默认的绘制操作

| 类 型     | 绘 制 结 果 |
|---------|---------|
| Boolean | 复选框     |
| Icon    | 图像      |
| object  | 字符串     |

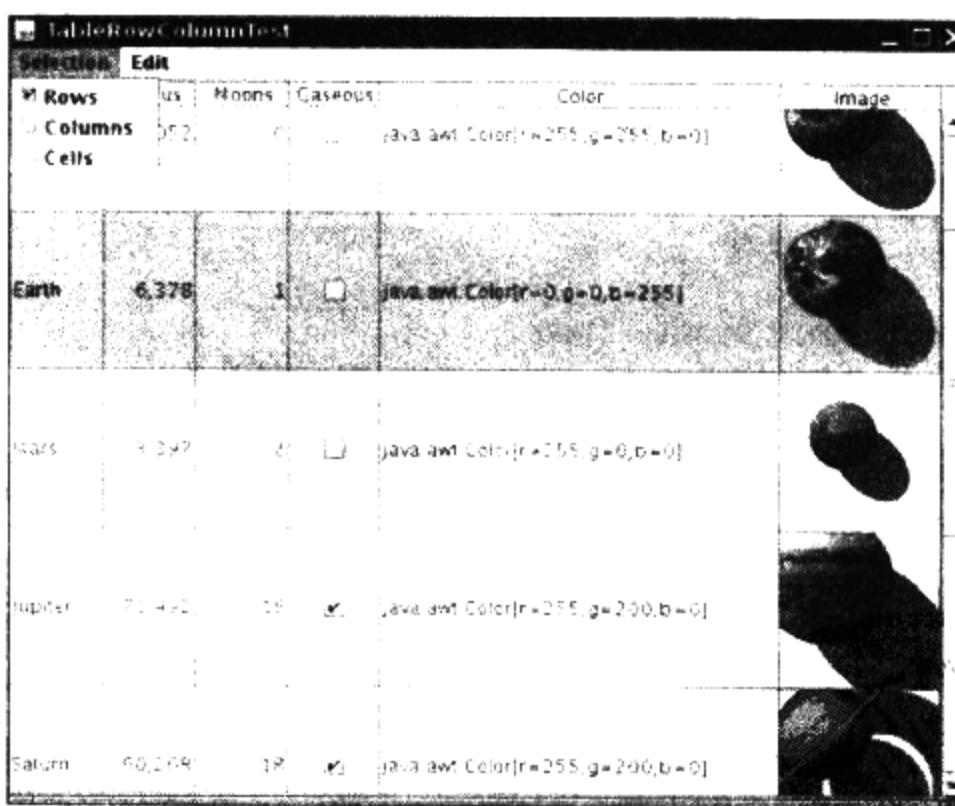


图6-9 具有单元格绘制器的表格

要绘制其他类型，需要安装定制的绘制器，请参见第6.2.4节。

#### 访问表格列

JTable类将有关表格列的信息存放在类型为TableColumn的对象中。由一个TableColumnModel对象负责管理这些列。（图6-10展示了最重要的表格类之间的关系。）如果不想动态地插入或删除这些列，那么最好不要过多地使用表格列模型。列模型的最常见的用法是直接获取一个TableColumn对象：

```
int columnIndex = ...;
TableColumn column = table.getColumnModel().getColumn(columnIndex);
```

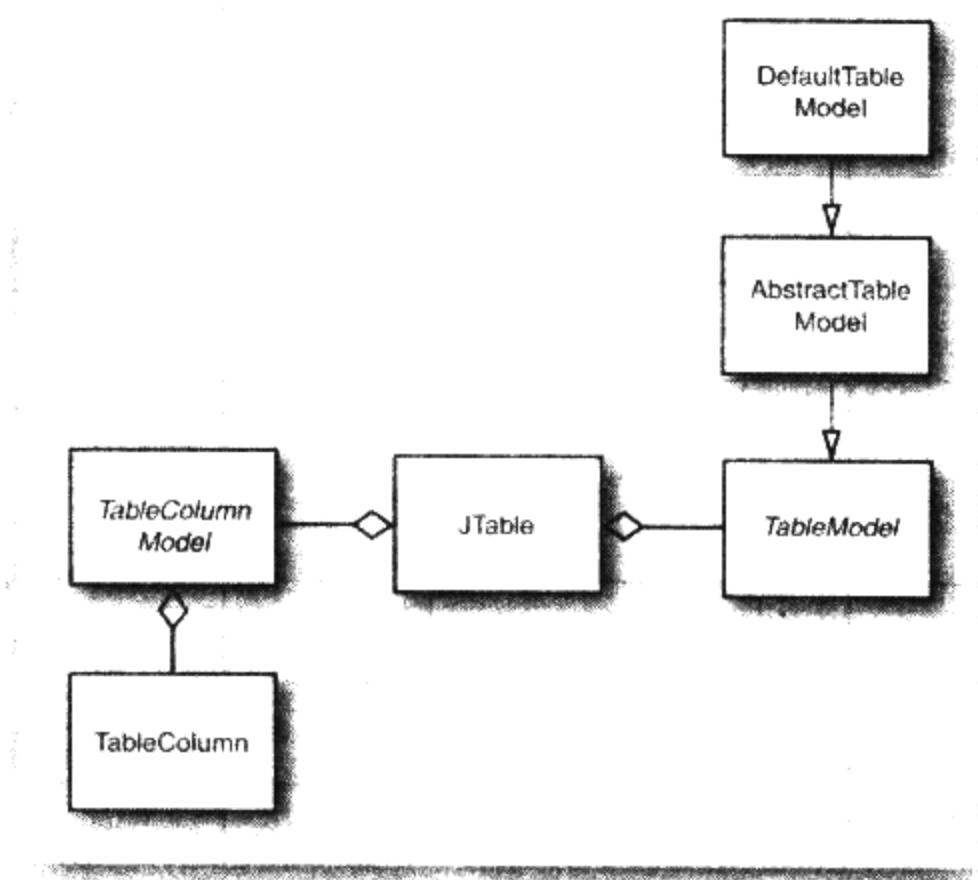


图6-10 表格类之间的关系图

### 改变列的大小

TableColumn类可以控制更改列的大小的行为。使用下面这些方法，可以设置首选的、最小的以及最大的宽度：

```
void setPreferredSize(int width)
void setMinWidth(int width)
void setMaxWidth(int width)
```

这些信息将提供给表格构件，以便对列进行布局。

### 使用方法

```
void setResizable(boolean resizable)
```

可以控制是否允许用户改变列的大小。

可以使用下面这个方法在程序中改变列的大小：

```
void setWidth(int width)
```

当调整了一个列的大小时，默认情况下表格的总体大小会保持不变。当然，更改过大小的列的宽度的增加值或减小值会分摊到其他列上。默认方式是更改那些在被改变大小列右边的所有列的大小。这是一种很好的默认方式，因为这样使得用户可以通过将所有列从左到右移动，将它们调整为自己所期望的宽度。

使用下面这个方法，可以设置表6-2中列出的其他行为：

```
void setAutoResizeMode(int mode)
```

表6-2 变更列大小的模式

| 模 式                            | 运 行 方 式   |
|--------------------------------|---|
| AUTO_RESIZE_OFF                | 不更改其他列的大小，而是更改表格的大小                                 |
| AUTO_RESIZE_NEXT_COLUMN        | 只更改下一列的大小   |
| AUTO_RESIZE_SUBSEQUENT_COLUMNS | 均匀地更改随后列的大小，这是默认的运行方式                               |
| AUTO_RESIZE_LAST_COLUMN        | 只更改最后一列的大小  |
| AUTO_RESIZE_ALL_COLUMNS        | 更改表格中的所有列的大小，这并不是一种很明智的选择，因为这需要用户对多个列进行调整以达到自己期望的大小 |

### 改变行的大小

行的高度是直接由JTable类管理的。如果单元格比默认值高，那么可以像下面这样设置行的高度：

```
table.setRowHeight(height);
```

默认情况下，表格中的所有行都具有相同的高度，可以用下面的调用来为每一行单独设置高度：

```
table.setRowHeight(row, height);
```

实际的行高度等于用这些方法设置的行高度减去行边距，其中行边距的默认值是1个像素，但是可以通过下面的调用来修改它：

```
table.setRowMargin(margin);
```

### 选择行、列和单元格

利用选择模式，用户可以选择表格中的行、列或者单独的单元格。默认情况下，行选择是可用的。点击一个单元格的内部以选择整行（参见图6-9）。调用

```
table.setRowSelectionAllowed(false)
```

可以禁用行选择。

当行选择功能可用时，可以控制用户是否可以选择单一行、连续几行或者任何几行。需要获取选择模式，然后调用它的setSelectionMode方法：

```
table.getSelectionModel().setSelectionMode(mode);
```

在这里，`mode`是下面三个值的其中一个：

```
ListSelectionModel.SINGLE_SELECTION  
ListSelectionModel.SINGLE_INTERVAL_SELECTION  
ListSelectionModel.MULTIPLE_INTERVAL_SELECTION
```

默认情况下，列选择是被禁用的。不过可以调用下面这个方法启用列选择：

```
table.setColumnSelectionAllowed(true)
```

同时使行选择和列选择可用等价于使单元格选择可用，这样用户就可以选择一定范围内的单元格（参见图6-11）。也可以使用下面的调用完成这项设置：

```
table.setCellSelectionEnabled(true)
```

可以运行程序清单6-6中的程序，观察一下单元格选择的运行情况。使Selection菜单中的行、列或单元格选项可用，然后观察选择行为是如何改变的。

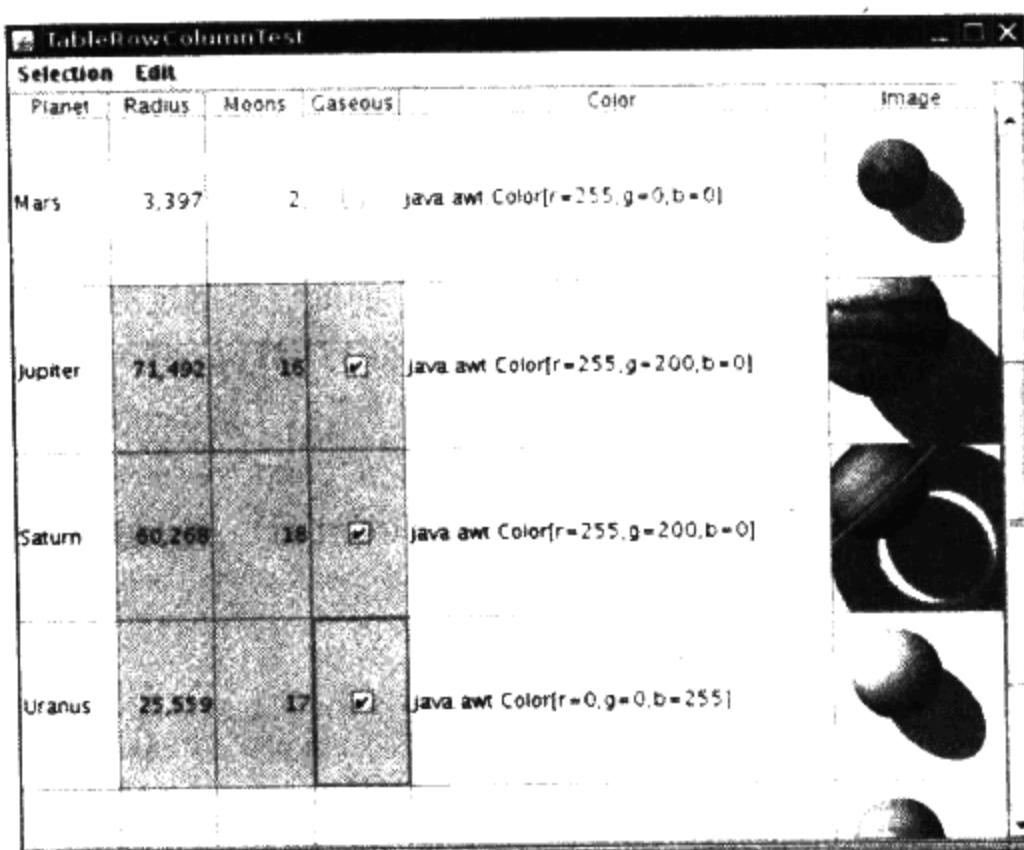


图6-11 选择一个单元格范围

可以通过调用`getSelectedRows`方法和`getSelectedColumns`方法来查看选中了哪些行及哪些列。这两个方法都返回一个由被选定项的索引构成的`int[]`数组。注意，这些索引值是表格视图中的索引值，而不是底层表格模型中的索引值。尝试着选择一些行和列，然后将列拖拽到不同的位置，并通过点击列头来对这些行进行排序。使用Print Selection菜单项来查看它会报告哪些行和列被选中。

#### 对行排序

正如在第一个表格示例中看到的那样，向`JTable`中添加行排序机制是很容易的，只需调用`setAutoCreateRowSorter`方法。但是，要对排序行为进行细粒度的控制，就必须像`JTable`中安装一个`TableRowSorter<M>`对象，并对其进行定制化。类型参数`M`表示表格模型，它必须是`TableModel`接口的子类型。

```
TableRowSorter<TableModel> sorter = new TableRowSorter<TableModel>(model);
table.setRowSorter(sorter);
```

某些列是不应该排序的，例如在我们的行星数据中的图像列，可以通过下面的调用来关闭排序机制：

```
sorter.setSortable(IMAGE_COLUMN, false);
```

可以对每个列都安装一个定制的比较器。在我们的示例中，我们将对`Color`列中的颜色进行排序，因为我们相对于红色来说，更喜欢蓝色和绿色。当你点击`Color`列时，将会看到蓝色行星出现在表格底部，这是通过下面的调用完成的：

```
sorter.setComparator(COLOR_COLUMN, new Comparator<Color>()
{
    public int compare(Color c1, Color c2)
    {
        int d = c1.getBlue() - c2.getBlue();
    }
});
```

```

        if (d != 0) return d;
        d = c1.getGreen() - c2.getGreen();
        if (d != 0) return d;
        return c1.getRed() - c2.getRed();
    }
});

```

如果不指定列的比较器，那么排序顺序就是按照下面的原则确定的：

1. 如果列所属的类是String，就使用Collator.getInstance()方法返回的默认比较器。它按照适用于当前locale的方式对字符串排序。（参加第5章以了解locale和比较器的更多信息）。
2. 如果列所属的类型实现了Comparable，则使用它的compareTo方法。
3. 如果已经为比较器设置过TableStringConverter，就用默认比较器对转换器的toString方法返回的字符串进行排序。

```

sorter.setStringConverter(new TableStringConverter()
{
    public String toString(TableModel model, int row, int column)
    {
        Object value = model.getValueAt(row, column);
        convert value to a string and return it
    }
});

```

4. 否则，在单元格的值上调用toString方法，然后用默认比较器对它们进行比较。

### 过滤行

除了可以对行排序之外，TableRowSorter还可以有选择性地隐藏行，这种处理称为过滤(filter)。要想激活过滤机制，需要设置RowFilter。例如，要包含所有至少有一个卫星的所有行星行，可以调用：

```
sorter.setRowFilter(RowFilter.numberFilter(ComparisonType.NOT_EQUAL, 0, MOONS_COLUMN));
```

这里我们使用了预定义的过滤器，即数字过滤器。要构建数字过滤器，需要提供：

- 比较类型 (EQUAL、NOT\_EQUAL、AFTER和BEFORE之一)。
- Number的某个子类的一个对象（例如Integer和Double），只有与给定的Number对象属于相同的类的对象才在考虑的范围内。
- 0或多列的索引值，如果不提供任何索引值，那么所有的列都被搜索。

静态的RowFilter.dataFilter方法以相同的方式构建了日期过滤器，这里需要提供Date对象而不是Number对象。

最后，静态的RowFilter.regexFilter方法构建的过滤器可以查找匹配某个正则表达式的字符串。例如：

```
sorter.setRowFilter(RowFilter.regexFilter(".*[As]$", PLANET_COLUMN));
```

将只显示那些名字以“s”结尾的行星（参见第1章以了解有关正则表达式的更新信息）。

还可以用andFilter、orFilter和notFilter方法来组合过滤器，例如，要过滤掉名字不是以“s”结尾，并且至少有一颗卫星的行星，可以使用下面的过滤器组合：

```
sorter.setRowFilter(RowFilter.andFilter(Arrays.asList(
    RowFilter.regexFilter(".*[As]$", PLANET_COLUMN),
    RowFilter.numberFilter(ComparisonType.NOT_EQUAL, 0, MOONS_COLUMN))));
```

**X** 警告：令人恼火的是，`andFilter`和`orFilter`方法不使用可变参数，而是单个的类型为`Iterable`的参数。

要实现自己的过滤器，需要提供`RowFilter`的一个子类，并实现`include`方法来表示哪些列应该显示。这很容易实现，但是`RowFilter`类卓越的普适性令它有点可怕。

`RowFilter<M, I>`类有两个类型参数：模型的类型和行标识符的类型。在处理表格时，模型总是`TableModel`的某个子类型，而标识符类型总是`Integer`。（在将来的某个时刻，其他构件可能也会支持行过滤机制。例如，要过滤`JTree`中的行，就可能可以使用`RowFilter<TreeModel, TreePath>`了。）

行过滤器必须实现下面的方法：

```
public boolean include(RowFilter.Entry<? extends M, ? extends I> entry)
```

`RowFilter.Entry`类提供了获取模型、行标识符和给定索引出的值等内容的方法，因此，按照行标识符和行的内容都可以进行过滤。

例如，下面的过滤器将隔行显示：

```
RowFilter<TableModel, Integer> filter = new RowFilter<TableModel, Integer>()
{
    public boolean include(Entry<? extends TableModel, ? extends Integer> entry)
    {
        return entry.getIdentifier() % 2 == 0;
    }
};
```

如果想要只包含那些具有偶数个卫星的行星，可以将上面的测试条件替换为下面的内容：

```
((Integer) entry.getValue(MOONS_COLUMN)) % 2 == 0
```

在我们的示例程序中，允许用户隐藏任意多行，我们在一个`set`中存储了所有隐藏的行的索引。而其中的行过滤器将包含那些索引不在这个`set`中的所有行。

过滤机制并不是为那些过滤标准在不时地发生变化的过滤器而设计的。因此，在我们的示例程序中，只要隐藏行的`set`发生了变化，我们就会调用下面的语句：

```
sorter.setRowFilter(filter);
```

设置过滤器会导致它立即得到应用。

#### 隐藏和显示列

正如在前一节中看到的，可以根据内容或标识符来过滤表格行，而隐藏表格列使用的完全不同的机制。

`JTable`类的`removeColumn`方法可以将一列从表格视图中移除。该列的数据实际上并没有从模型中移除，只是将它从视图中隐藏了起来。`removeColumn`方法接收一个 `TableColumn`参数。如果拥有了一个列号（比如来自于`getSelectedColumns`的调用结果），就必须向表格模型请求实际的列对象：

```
TableColumnModel columnModel = table.getColumnModel();
TableColumn column = columnModel.getColumn(i);
table.removeColumn(column);
```

如果你记得住该列，那么将来就可以再把它添加回去：

```
table.addColumn(column);
```

该方法将该列添加到表格的最后面。如果想让它出现在表格中的其他任何地方，那么可以调用moveColumn方法。

通过添加一个新的TableColumn对象，还可以添加一个对应于表格模型中的一个列索引的新列：

```
table.addColumn(new TableColumn(modelColumnIndex));
```

可以让多个表格列展示模型中的同一列。

程序清单6-6中的程序展示了如何选择和过滤行与列。

### 程序清单6-6 TableSelectionTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.util.*;
4. import javax.swing.*;
5. import javax.swing.table.*;
6.
7. /**
8. * This program demonstrates selection, addition, and removal of rows and columns.
9. * @version 1.03 2007-08-01
10. * @author Cay Horstmann
11. */
12. public class TableSelectionTest
13. {
14.     public static void main(String[] args)
15.     {
16.         EventQueue.invokeLater(new Runnable()
17.         {
18.             public void run()
19.             {
20.                 JFrame frame = new TableSelectionFrame();
21.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22.                 frame.setVisible(true);
23.             }
24.         });
25.     }
26. }
27.
28. /**
29. * This frame shows a multiplication table and has menus for setting the row/column/cell
30. * selection modes, and for adding and removing rows and columns.
31. */
32. class TableSelectionFrame extends JFrame
33. {
34.     public TableSelectionFrame()
35.     {
36.         setTitle("TableSelectionTest");
37.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
38.
39.         // set up multiplication table
40.
41.         model = new DefaultTableModel(10, 10);
42.
```

```
43.     for (int i = 0; i < model.getRowCount(); i++)
44.         for (int j = 0; j < model.getColumnCount(); j++)
45.             model.setValueAt((i + 1) * (j + 1), i, j);
46.
47.     table = new JTable(model);
48.
49.     add(new JScrollPane(table), "Center");
50.
51.     removedColumns = new ArrayList<TableColumn>();
52.
53.     // create menu
54.
55.     JMenuBar menuBar = new JMenuBar();
56.     setJMenuBar(menuBar);
57.
58.     JMenu selectionMenu = new JMenu("Selection");
59.     menuBar.add(selectionMenu);
60.
61.     final JCheckBoxMenuItem rowsItem = new JCheckBoxMenuItem("Rows");
62.     final JCheckBoxMenuItem columnsItem = new JCheckBoxMenuItem("Columns");
63.     final JCheckBoxMenuItem cellsItem = new JCheckBoxMenuItem("Cells");
64.
65.     rowsItem.setSelected(table.getRowSelectionAllowed());
66.     columnsItem.setSelected(table.getColumnSelectionAllowed());
67.     cellsItem.setSelected(table.getCellSelectionEnabled());
68.
69.     rowsItem.addActionListener(new ActionListener()
70.     {
71.         public void actionPerformed(ActionEvent event)
72.         {
73.             table.clearSelection();
74.             table.setRowSelectionAllowed(rowsItem.isSelected());
75.             cellsItem.setSelected(table.getCellSelectionEnabled());
76.         }
77.     });
78.     selectionMenu.add(rowsItem);
79.
80.     columnsItem.addActionListener(new ActionListener()
81.     {
82.         public void actionPerformed(ActionEvent event)
83.         {
84.             table.clearSelection();
85.             table.setColumnSelectionAllowed(columnsItem.isSelected());
86.             cellsItem.setSelected(table.getCellSelectionEnabled());
87.         }
88.     });
89.     selectionMenu.add(columnsItem);
90.
91.     cellsItem.addActionListener(new ActionListener()
92.     {
93.         public void actionPerformed(ActionEvent event)
94.         {
95.             table.clearSelection();
96.             table.setCellSelectionEnabled(cellsItem.isSelected());
97.             rowsItem.setSelected(table.getRowSelectionAllowed());
98.             columnsItem.setSelected(table.getColumnSelectionAllowed());
99.         }
99.
```

```
100.    });
101.    selectionMenu.add(cellsItem);
102.
103.    JMenu tableMenu = new JMenu("Edit");
104.    menuBar.add(tableMenu);
105.
106.    JMenuItem hideColumnsItem = new JMenuItem("Hide Columns");
107.    hideColumnsItem.addActionListener(new ActionListener()
108.    {
109.        public void actionPerformed(ActionEvent event)
110.        {
111.            int[] selected = table.getSelectedColumns();
112.            TableColumnModel columnModel = table.getColumnModel();
113.
114.            // remove columns from view, starting at the last
115.            // index so that column numbers aren't affected
116.
117.            for (int i = selected.length - 1; i >= 0; i--)
118.            {
119.                TableColumn column = columnModel.getColumn(selected[i]);
120.                table.removeColumn(column);
121.
122.                // store removed columns for "show columns" command
123.
124.                removedColumns.add(column);
125.            }
126.        }
127.    });
128.    tableMenu.add(hideColumnsItem);
129.
130.    JMenuItem showColumnsItem = new JMenuItem("Show Columns");
131.    showColumnsItem.addActionListener(new ActionListener()
132.    {
133.        public void actionPerformed(ActionEvent event)
134.        {
135.            // restore all removed columns
136.            for (TableColumn tc : removedColumns)
137.                table.addColumn(tc);
138.            removedColumns.clear();
139.        }
140.    });
141.    tableMenu.add(showColumnsItem);
142.
143.    JMenuItem addRowItem = new JMenuItem("Add Row");
144.    addRowItem.addActionListener(new ActionListener()
145.    {
146.        public void actionPerformed(ActionEvent event)
147.        {
148.            // add a new row to the multiplication table in
149.            // the model
150.
151.            Integer[] newCells = new Integer[model.getColumnCount()];
152.            for (int i = 0; i < newCells.length; i++)
153.                newCells[i] = (i + 1) * (model.getRowCount() + 1);
154.            model.addRow(newCells);
155.        }
156.
```

```
156.        });
157.        tableMenu.add(addRowItem);
158.
159.        JMenuItem removeRowsItem = new JMenuItem("Remove Rows");
160.        removeRowsItem.addActionListener(new ActionListener()
161.        {
162.            public void actionPerformed(ActionEvent event)
163.            {
164.                int[] selected = table.getSelectedRows();
165.
166.                for (int i = selected.length - 1; i >= 0; i--)
167.                    model.removeRow(selected[i]);
168.            }
169.        });
170.        tableMenu.add(removeRowsItem);
171.
172.        JMenuItem clearCellsItem = new JMenuItem("Clear Cells");
173.        clearCellsItem.addActionListener(new ActionListener()
174.        {
175.            public void actionPerformed(ActionEvent event)
176.            {
177.                for (int i = 0; i < table.getRowCount(); i++)
178.                    for (int j = 0; j < table.getColumnCount(); j++)
179.                        if (table.isCellSelected(i, j)) table.setValueAt("", i, j);
180.            }
181.        });
182.        tableMenu.add(clearCellsItem);
183.    }
184.
185.    private DefaultTableModel model;
186.    private JTable table;
187.    private ArrayList<TableColumn> removedColumns;
188.
189.    private static final int DEFAULT_WIDTH = 400;
190.    private static final int DEFAULT_HEIGHT = 300;
191.}
```



### javax.swing.JTable.TableModel 1.2

- Class getColumnClass(int columnIndex)

获取该列中的值的类。该信息用于绘制或分类。



### javax.swing.JTable 1.2

- TableColumnModel getColumnModel()

获取描述表格列布局安排的“列模式”。

- void setAutoResizeMode(int mode)

设置自动更改表格列大小的模式。

参数: mode AUTO\_RESIZE\_OFF、AUTO\_RESIZE\_NEXT\_COLUMN、  
AUTO\_RESIZE\_SUBSEQUENT\_COLUMNS、AUTO\_RESIZE\_LAST\_COLUMN以及  
AUTO\_RESIZE\_ALL\_COLUMNS其中之一

- `int getRowMargin()`
- `void setRowMargin(int margin)`  
获取和设置相邻行中单元格之间的间隔大小。
- `int getRowHeight()`
- `void setRowHeight(int height)`  
获取和设置表格中所有行的默认高度。
- `int getRowHeight(int row)`
- `void setRowHeight(int row, int height)`  
获取和设置表格中给定行的高度。
- `ListSelectionModel getSelectionModel()`  
返回列表的选择模式。你需要该模式以便在行、列以及单元格之间进行选择。
- `boolean getRowSelectionAllowed()`
- `void setRowSelectionAllowed(boolean b)`  
获取和设置`rowSelectionAllowed`属性。如果为true，那么当用户点击单元格的时候，可以选定行。
- `boolean getColumnSelectionAllowed()`
- `void setColumnSelectionAllowed(boolean b)`  
获取和设置`columnSelectionAllowed`属性。如果为true，那么当用户点击单元格的时候，可以选定列。
- `boolean getCellSelectionEnabled()`  
如果既允许选定行又允许选定列，则返回true。
- `void setCellSelectionEnabled(boolean b)`  
同时将`rowSelectionAllowed`和`columnSelectionAllowed`设置为b。
- `voidaddColumn(TableColumn column)`  
向表格视图中添加一列。
- `void moveColumn(int from, int to)`  
移动表格from索引位置中的列，使它的索引变成to。该操作仅仅影响到视图。
- `void removeColumn(TableColumn column)`  
将给定的列从视图中移除。
- `int convertRowIndexToModel(int index) 6`
- `int convertColumnIndexToModel(int index)`  
返回具有给定索引的行或列的模型索引，这个值与行被排序和过滤，以及列被移动和移除时的索引不同。
- `void setRowSorter(RowSorter<? extends TableModel> sorter)`  
设置行排序器。

**API** **javax.swing.table.TableColumnModel 1.2**

- `TableColumn getColumn(int index)`

获取表格的列对象，用于描述给定索引的列。

**API** **javax.swing.table.TableColumn 1.2**

- `TableColumn(int modelColumnIndex)`

构建一个表格列，用以显示给定索引位置上的模型列。

- `void setPreferredWidth(int width)`

- `void setMinWidth(int width)`

- `void setMaxWidth(int width)`

将表格的首选宽度、最小宽度以及最大宽度设置为width。

- `void setWidth(int width)`

设置该列的实际宽度为width。

- `void setResizable(boolean b)`

如果b为true，那么该列可以更改大小。

**API** **javax.swing.ListSelectionModel 1.2**

- `void setSelectionMode(int mode)`

参数：mode `SINGLE_SELECTION`、`SINGLE_INTERVAL_SELECTION`以及  
`MULTIPLE_INTERVAL_SELECTION`其中之一

**API** **javax.swing.DefaultRowSorter<M, I> 6**

- `void setComparator(int column, Comparator<?> comparator)`

设置用于给定列的比较器。

- `void setSortable(int column, boolean enabled)`

使对给定列的排序可用或禁用。

- `void setRowFilter(RowFilter<? super M, ? super I> filter)`

设置行过滤器。

**API** **javax.swing.table.TableRowSorter<M extends TableModel> 6**

- `void setStringConverter(TableModelStringConverter stringConverter)`

设置用于排序和过滤的字符串转换器。

**API** **javax.swing.table.TableStringConverter<M extends TableModel> 6**

- `abstract String toString(TableModel model, int row, int column)`

覆盖这个方法，将给定位置的模型值转换为字符串。

**API** **javax.swing.RowFilter<M, I> 6**

- `boolean include(RowFilter.Entry<? extends M, ? extends I> entry)`

覆盖这个方法为指定要保留的行。

- static <M,I> RowFilter<M,I> numberFilter(RowFilter.ComparisonType type, Number number, int... indices)
  - static <M,I> RowFilter<M,I> dateFilter(RowFilter.ComparisonType type, Date date, int... indices)
- 返回一个过滤器，它包含的行是那些与给定的数字或日期进行给定比较后匹配的行。比较类型是EQUAL、NOT\_EQUAL、AFTER或BEFORE之一。如果给定了列模型索引，则只搜索这些列。否则，将搜索所有列。对于数字过滤器，单元格的值所属的类必须与给定数字的类匹配。
- static <M,I> RowFilter<M,I> regexFilter(String regex, int... indices)
- 返回一个过滤器，它包含行含有与给定的正则表达式匹配的字符串。如果给定了列模型索引，则只搜索这些列。否则，将搜索所有列。注意，RowFilter.Entry的getStringValue方法返回的字符串是匹配的。
- static <M,I> RowFilter<M,I> andFilter(Iterable<? extends RowFilter<? super M,> filters)
  - static <M,I> RowFilter<M,I> orFilter(Iterable<? extends RowFilter<? super M,> filters)
- 返回一个过滤器，它包含的项是那些包含在所有的过滤器或至少包含在一个过滤器中的项。
- static <M,I> RowFilter<M,I> notFilter(RowFilter<M,I> filter)
- 返回一个过滤器，它包含的项是那些不包含在给定过滤器中的项。



#### javax.swing.RowFilter.Entry<M, I>

- I getIdentifier()
- 返回这个行项的标识符。
- M getModel()
- 返回这个行项的模型。
- Object getValue(int index)
- 返回在这个行的给定索引处存储的值。
- int getValueCount()
- 返回在这个行中存储的值的数量。
- String getStringValue()
- 返回在这个行的给定索引处存储的值转换成的字符串。由TableRowSorter产生的项的getStringValue方法会调用排序器的字符串转换器。

#### 6.2.4 单元格的绘制和编辑

正如在第6.2.3节中看到的，列的类型确定了单元格应该如何绘制。Boolean和Icon类型有默认的绘制器，它们将绘制复选框或图标，而对于其他所有类型，都需要安装定制的绘制器。

表格的单元格绘制器与你在前面看到的列表单元格绘制器类似。它们都实现了TableCellRenderer接口，并只有一个方法

```
Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected,
```

```
boolean hasFocus, int row, int column)
```

该方法在表格需要绘制一个单元格的时候被调用。它会返回一个构件，接着该构件的paint方法会被调用，以填充单元格区域。

在图6-12中的表格包含类型为Color的单元格，绘制器直接返回一个面板，其背景颜色设置为存储在该单元格中的颜色对象，该颜色是作为value参数传递的。

```
class ColorTableCellRenderer extends JPanel implements TableCellRenderer
{
    public Component getTableCellRendererComponent(JTable table, Object value,
                                                   boolean isSelected,
                                                   boolean hasFocus, int row, int column)
    {
        setBackground((Color) value);
        if (hasFocus)
            setBorder(UIManager.getBorder("Table.focusCellHighlightBorder"));
        else
            setBorder(null);
    }
    return this;
}
```

| Planet  | Radius | Gasous | Color      | Image |
|---------|--------|--------|------------|-------|
| Mars    | 3,390  | Yes    | Black      |       |
| Jupiter | 74,492 | No     | Light Gray |       |
| Saturn  | 60,268 | Yes    | Dark Gray  |       |

图6-12 具有单元格绘制器的表格

正如你看到的那样，当该单元格获得焦点的时候，绘制器会安装一个边框。（我们可以向UIManager寻求合适的边框。为了发现查找的关键所在，我们可以深入DefaultTableCellRenderer类的源码内部看个究竟。）

通常情况下，你可能还想设置单元格的背景颜色，以指示当前是否选中了它。这里我们跳过这步，因为这会干扰我们现在讨论的被显示颜色。程序清单6-3中的ListRenderingTest示例展示了怎样在一个绘制器中指示选择状态。

**!** 提示：如果你的绘制器只是绘制一个文本字符串或者一个图标，那么可以继承DefaultTableCellRenderer这个类。该类会负责绘制焦点和选择状态。

你必须告诉表格要使用这个绘制器去绘制所有类型为Color的对象。JTable类的

`setDefaultRenderer`方法可以让你建立这种联系。你需要提供一个Class对象和绘制器。

```
table.setDefaultRenderer(Color.class, new ColorTableCellRenderer());
```

现在这个绘制器就可以用于具有给定类型的所有对象了。

如果想要基于其他标准选择绘制器，则需要从`JTable`类中扩展子类，并覆盖`getCellRenderer`方法。

#### 绘制表头

为了在表头中显示图标，如下设置

```
moonColumn.setHeaderValue(new ImageIcon("Moons.gif"));
```

然而，表头不够智能为表头值选择一个合适的绘制器。绘制器需要手工安装。例如，在列头显示图像图标，调用如下：

```
moonColumn.setHeaderRenderer(table.getDefaultRenderer(ImageIcon.class));
```

#### 单元格编辑

为了使单元格可编辑，表格模型必须通过定义`isCellEditable`方法来指明哪些单元格是可编辑的。最常见的情况是，你可能想使某几列可编辑。在这个示例程序中，我们允许对表格中的四列进行编辑。

```
public boolean isCellEditable(int r, int c)
{
    return c == PLANET_COLUMN || c == MOONS_COLUMN || c == GASEOUS_COLUMN || c == COLOR_COLUMN;
}
```



**注意：**`AbstractTableModel`定义的`isCellEditable`方法总是返回`false`。`DefaultTableModel`覆盖了该方法以便总是返回`true`。

运行一下程序清单6-7的程序就会注意到，可以点击Gaseous列中的复选框，并能打开或关闭复选标记。如果点击Moons列中的某个单元格，就会出现一个组合框（参见图6-13）。你很快就会看到怎样将这样一个组合框作为一个单元格编辑器安装到表格上。

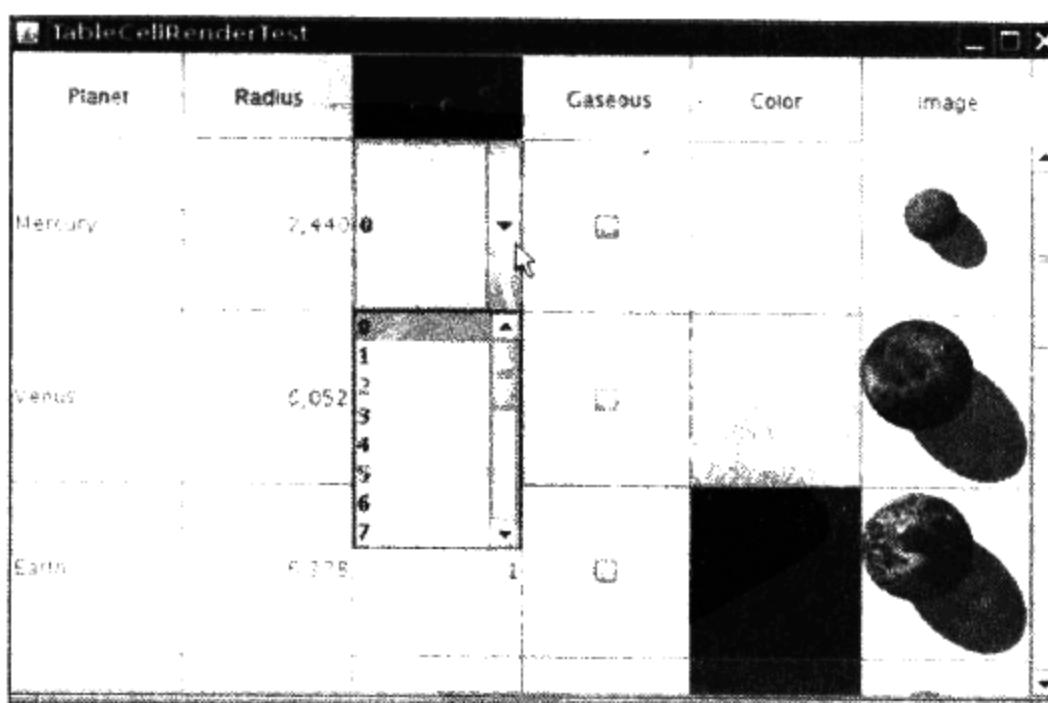


图6-13 单元格编辑器

最后，点击第一列中的某个单元格，该单元格就会获取焦点。你就可以开始键入数据，而该单元格的内容也会随之更改。

你刚刚看到的只是DefaultCellEditor类的三种变型。DefaultCellEditor可以用 JTextField、JCheckBox或者 JComboBox来构造。JTable类会自动为Boolean类型的单元格安装一个复选框编辑器，并为所有可编辑的、但未提供它们自己的绘制器的单元格安装一个文本编辑器。文本框可以让用户去编辑那些对表格模型getValueAt方法的返回值执行toString操作而产生的字符串。

一旦编辑完成，通过调用编辑器的getCellEditorValue方法就可以读取编辑过的值。该方法应该返回一个正确类型的值（也就是模型的getColumnType方法返回的类型）。

为了获得一个组合框编辑器，你需要手动设置单元格编辑器，因为JTable构件并不知道什么样的值对某一特殊类型来说是适合的。对于Moons列来说，我们希望可以让用户选择0到20之间的任何值。下面是对组合框进行初始化的代码。

```
JComboBox moonCombo = new JComboBox();
for (int i = 0; i <= 20; i++)
    moonCombo.addItem(i);
```

为了构造一个DefaultCellEditor，需要在该构造器中提供一个组合框。

```
TableCellEditor moonEditor = new DefaultCellEditor(moonCombo);
```

接下来，我们需要安装这个编辑器。与颜色单元格绘制器不同，这个编辑器不依赖于对象类型，我们未必想要把它作用于类型为Integer的所有对象上。相反地，我们需要把它安装到一个特定列中：

```
moonColumn.setCellEditor(moonEditor);
```

#### 定制编辑器

再次运行一下示例程序并点击一种颜色。这时会弹出一个颜色选择器让你为行星选择一种新颜色。选中一种颜色，然后点击OK。单元格颜色就会随之更新（参见图6-14）。

颜色单元格编辑器并不是一种标准的表格单元格编辑器，而是一种定制实现的编辑器。为了创建一个定制的单元格编辑器，需要实现TableCellEditor接口。这个接口有点拖沓冗长，从Java SE 1.3开始，提供了AbstractCellEditor类，用于负责事件处理的细节。

TableCellEditor接口的getTableCellEditorComponent方法请求某个构件去绘制单元格。除了没有focus参数之外，它和TableCellRenderer接口的getTableCellRendererComponent方法极为相似。因为我们假设要编辑单元格，所以我们假设它具有焦点。在编辑过程中，编辑器构件暂时取代绘制器。在我们的示例中，我们返回的是一个没有颜色的面板。这只是告诉用户该单元格正在被编辑。

接下来，当用户点击单元格时，你希望能弹出你自己的编辑器。

JTable类用一个事件（例如鼠标点击）去调用你的编辑器，以便确定该事件是否可以被接收去启动编辑过程。AbstractCellEditor将该方法定义为能够接收所有的事件类型。

```
public boolean isCellEditable(EventObject anEvent)
{
    return true;
}
```

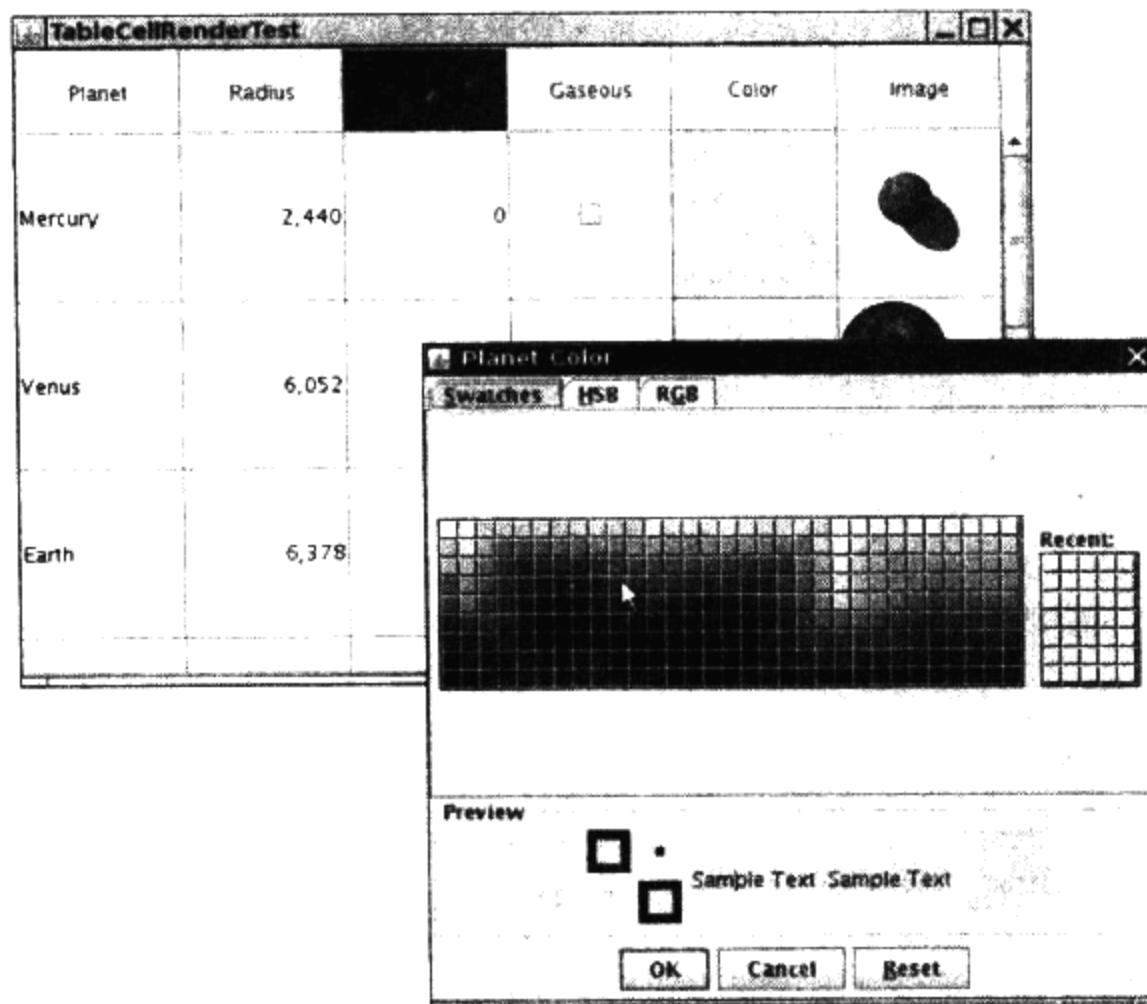


图6-14 使用颜色选择器对单元格的颜色进行编辑

然而，如果你将该方法覆盖成`false`，那么表格模型就不会遇到插入编辑器构件这样的麻烦了。

一旦安装了编辑器构件，假设我们使用的是相同的事件，那么`shouldSelectCell`方法就会被调用。应该在这个方法中启动编辑过程，例如，弹出一个外部的编辑对话框。

```
public boolean shouldSelectCell(EventObject anEvent)
{
    colorDialog.setVisible(true);
    return true;
}
```

如果用户取消编辑，表格会调用`cancelCellEditing`方法。如果用户已经点击了另一个表格单元，那么表格会调用`stopCellEditing`方法。在这两种情况中，你都应该将对话框隐藏起来。当`stopCellEditing`方法被调用时，表格可能会使用部分被编辑过的值。如果当前值有效，那么应该返回`true`。在颜色选择器中，任何值都是有效的。但是如果编辑的是其他数据，那么应该保证只有有效的数据才能从编辑器中读取出来。

另外，应该调用超类的方法，以便进行事件的触发，否则，编辑事件就无法正确地撤销。

```
public void cancelCellEditing()
{
    colorDialog.setVisible(false);
    super.cancelCellEditing();
}
```

最后，必须提供一个方法，以便产生用户在编辑过程中所提供的值。

```
public Object getCellEditorValue()
{
    return colorChooser.getColor();
}
```

总结一下，你的定制编辑器应该遵循下面几点：

1. 继承AbstractCellEditor类，并实现TableCellEditor接口。
2. 定义getTableCellEditorComponent方法以提供一个构件。它可以是一个哑构件（如果你弹出一个对话框）或者是适当的编辑构件，例如复选框或文本框。
3. 定义shouldSelectCell、stopCellEditing及cancelCellEditing方法，来处理编辑过程的开始、完成以及撤销。stopCellEditing和cancelCellEditing方法应该调用超类方法以保证监听器能够接收到通知。
4. 定义getCellEditorValue方法返回编辑结果的值。

最后，通过调用stopCellEditing和cancelCellEditing方法，以表明用户什么时间完成了编辑操作。在构建颜色对话框的时候，我们安装了接收和取消的回调，用于触发这些事件。

```
colorDialog = JColorChooser.createDialog(null, "Planet Color", false, colorChooser,
    new
        ActionListener() // OK button listener
    {
        public void actionPerformed(ActionEvent event)
        {
            stopCellEditing();
        }
    },
    new
        ActionListener() // Cancel button listener
    {
        public void actionPerformed(ActionEvent event)
        {
            cancelCellEditing();
        }
    });
});
```

另外，当用户关闭对话框的时候，编辑应该随之取消。这是通过安装窗体监听器来实现的：

```
colorDialog.addWindowListener(new
    WindowAdapter()
{
    public void windowClosing(WindowEvent event)
    {
        cancelCellEditing();
    }
});
```

这样就完成了定制编辑器的实现过程。

你现在已经知道了怎样使一个单元格可编辑，以及怎样安装一个编辑器。还剩下一个问题，即怎样使用用户编辑过的值来更新表格模型。当编辑完成的时候，JTable类会调用表格模型的下面这个方法：

```
void setValueAt(Object value, int r, int c)
```

需要将这个方法覆盖掉以便存储新值。value参数是单元格编辑器返回的对象。如果实现

了单元格编辑器，那么你就知道从getCellEditorValue方法返回的是什么类型的对象。在DefaultCellEditor这种情况下，这个值有三种可能：如果单元格编辑器是复选框，那么它就是Boolean值；如果是一个文本框，那么它就是一个字符串；如果这个值来源于组合框，那么就是用户选定的对象。

如果value对象不具有合适的类型，那么需要对它进行转换。例如在一个文本框中编辑一个数字，这种情况最常发生。在我们的示例中，我们是在组合框安装了Integer对象，所以不需要任何转换。

### 程序清单6-7 TableCellRenderTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.util.*;
4. import javax.swing.*;
5. import javax.swing.table.*;
6.
7. /**
8. * This program demonstrates cell rendering and editing in a table.
9. * @version 1.02 2007-08-01
10. * @author Cay Horstmann
11. */
12. public class TableCellRenderTest
13. {
14.     public static void main(String[] args)
15.     {
16.         EventQueue.invokeLater(new Runnable()
17.         {
18.             public void run()
19.             {
20.
21.                 JFrame frame = new TableCellRenderFrame();
22.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23.                 frame.setVisible(true);
24.             }
25.         });
26.     }
27. }
28.
29. /**
30. * This frame contains a table of planet data.
31. */
32. class TableCellRenderFrame extends JFrame
33. {
34.     public TableCellRenderFrame()
35.     {
36.         setTitle("TableCellRenderTest");
37.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
38.
39.         TableModel model = new PlanetTableModel();
40.         JTable table = new JTable(model);
41.         table.setRowSelectionAllowed(false);
42.
43.         // set up renderers and editors
```

```
44.  
45.     table.setDefaultRenderer(Color.class, new ColorTableCellRenderer());  
46.     table.setDefaultEditor(Color.class, new ColorTableCellEditor());  
47.  
48.     JComboBox moonCombo = new JComboBox();  
49.     for (int i = 0; i <= 20; i++)  
50.         moonCombo.addItem(i);  
51.  
52.     TableColumnModel columnModel = table.getColumnModel();  
53.     TableColumn moonColumn = columnModel.getColumn(PlanetTableModel.MOONS_COLUMN);  
54.     moonColumn.setCellEditor(new DefaultCellEditor(moonCombo));  
55.     moonColumn.setHeaderRenderer(table.getDefaultRenderer(ImageIcon.class));  
56.     moonColumn.setHeaderValue(new ImageIcon("Moons.gif"));  
57.  
58.     // show table  
59.  
60.     table.setRowHeight(100);  
61.     add(new JScrollPane(table), BorderLayout.CENTER);  
62. }  
63.  
64. private static final int DEFAULT_WIDTH = 600;  
65. private static final int DEFAULT_HEIGHT = 400;  
66. }  
67.  
68. /**  
69. * The planet table model specifies the values, rendering and editing properties for the  
70. * planet data.  
71. */  
72. class PlanetTableModel extends AbstractTableModel  
73. {  
74.     public String getColumnName(int c)  
75.     {  
76.         return columnNames[c];  
77.     }  
78.  
79.     public Class<?> getColumnClass(int c)  
80.     {  
81.         return cells[0][c].getClass();  
82.     }  
83.  
84.     public int getColumnCount()  
85.     {  
86.         return cells[0].length;  
87.     }  
88.  
89.     public int getRowCount()  
90.     {  
91.         return cells.length;  
92.     }  
93.  
94.     public Object getValueAt(int r, int c)  
95.     {  
96.         return cells[r][c];  
97.     }  
98.  
99.     public void setValueAt(Object obj, int r, int c)  
100.    {
```

```
101.     cells[r][c] = obj;
102. }
103.
104. public boolean isCellEditable(int r, int c)
105. {
106.     return c == PLANET_COLUMN || c == MOONS_COLUMN || c == GASEOUS_COLUMN ||
107.           c == COLOR_COLUMN;
108. }
109.
110. public static final int PLANET_COLUMN = 0;
111. public static final int MOONS_COLUMN = 2;
112. public static final int GASEOUS_COLUMN = 3;
113. public static final int COLOR_COLUMN = 4;
114.
115. private Object[][] cells = {
116.     { "Mercury", 2440.0, 0, false, Color.YELLOW, new ImageIcon("Mercury.gif") },
117.     { "Venus", 6052.0, 0, false, Color.YELLOW, new ImageIcon("Venus.gif") },
118.     { "Earth", 6378.0, 1, false, Color.BLUE, new ImageIcon("Earth.gif") },
119.     { "Mars", 3397.0, 2, false, Color.RED, new ImageIcon("Mars.gif") },
120.     { "Jupiter", 71492.0, 16, true, Color.ORANGE, new ImageIcon("Jupiter.gif") },
121.     { "Saturn", 60268.0, 18, true, Color.ORANGE, new ImageIcon("Saturn.gif") },
122.     { "Uranus", 25559.0, 17, true, Color.BLUE, new ImageIcon("Uranus.gif") },
123.     { "Neptune", 24766.0, 8, true, Color.BLUE, new ImageIcon("Neptune.gif") },
124.     { "Pluto", 1137.0, 1, false, Color.BLACK, new ImageIcon("Pluto.gif") } };
125.
126. private String[] columnNames = { "Planet", "Radius", "Moons", "Gaseous", "Color",
127.                                 "Image" };
128. }
129.
130. /**
131. * This renderer renders a color value as a panel with the given color.
132. */
133. class ColorTableCellRenderer extends JPanel implements TableCellRenderer
134. {
135.     public Component getTableCellRendererComponent(JTable table, Object value,
136.                                                   boolean isSelected, boolean hasFocus, int row, int column)
137.     {
138.         setBackground((Color) value);
139.         if (hasFocus) setBorder(UIManager.getBorder("Table.focusCellHighlightBorder"));
140.         else setBorder(null);
141.         return this;
142.     }
143. }
144.
145. /**
146. * This editor pops up a color dialog to edit a cell value
147. */
148. class ColorTableCellEditor extends AbstractCellEditor implements TableCellEditor
149. {
150.     public ColorTableCellEditor()
151.     {
152.         panel = new JPanel();
153.         // prepare color dialog
154.
155.         colorChooser = new JColorChooser();
156.         colorDialog = JColorChooser.createDialog(null, "Planet Color", false, colorChooser,
157.                                               new ActionListener() // OK button listener
```

```
158.         {
159.             public void actionPerformed(ActionEvent event)
160.             {
161.                 stopCellEditing();
162.             }
163.             }, new ActionListener() // Cancel button listener
164.             {
165.                 public void actionPerformed(ActionEvent event)
166.                 {
167.                     cancelCellEditing();
168.                 }
169.             });
170.             colorDialog.addWindowListener(new WindowAdapter()
171.             {
172.                 public void windowClosing(WindowEvent event)
173.                 {
174.                     cancelCellEditing();
175.                 }
176.             });
177.         }
178.
179.         public Component getTableCellEditorComponent(JTable table, Object value,
180.             boolean isSelected, int row, int column)
181.         {
182.             // this is where we get the current Color value. We store it in the dialog in case
183.             // the user starts editing
184.             colorChooser.setColor((Color) value);
185.             return panel;
186.         }
187.
188.         public boolean shouldSelectCell(EventObject anEvent)
189.         {
190.             // start editing
191.             colorDialog.setVisible(true);
192.
193.             // tell caller it is ok to select this cell
194.             return true;
195.         }
196.
197.         public void cancelCellEditing()
198.         {
199.             // editing is canceled--hide dialog
200.             colorDialog.setVisible(false);
201.             super.cancelCellEditing();
202.         }
203.
204.         public boolean stopCellEditing()
205.         {
206.             // editing is complete--hide dialog
207.             colorDialog.setVisible(false);
208.             super.stopCellEditing();
209.
210.             // tell caller is is ok to use color value
211.             return true;
212.         }
213.
214.         public Object getCellEditorValue()
```

```

215.    {
216.        return colorChooser.getColor();
217.    }
218.
219.    private JColorChooser colorChooser;
220.    private JDialog colorDialog;
221.    private JPanel panel;
222. }
```

**API** **javax.swing.JTable 1.2**

- TableCellRenderer getDefaultRenderer(Class<?> type)

获取给定类型的默认绘制器。

- TableCellEditor getDefaultEditor(Class<?> type)

获取给定类型的默认编辑器。

**API** **javax.swing.JTable.TableCellRender 1.2**

- Component getTableCellRendererComponent(JTable table, Object value, boolean selected, boolean hasFocus, int row, int column)

返回一个构件，它的paint方法将被调用以便绘制一个表格单元格。

参数：table 该表格包含要绘制的单元格。

value 要绘制的单元格

selected 如果该单元格当前已被选中，则返回true

hasFocus 如果该单元格当前具有焦点，则返回true

row, column 单元格的行及列

**API** **javax.swing.table.TableColumn 1.2**

- void setCellEditor(TableCellEditor editor)

- void setCellRenderer(TableCellRenderer renderer)

为该列中的所有单元格设置单元格编辑器或绘制器。

- void setHeaderRenderer(TableCellRenderer renderer)

为该列中的所有表头单元格设置单元格绘制器。

- void setHeaderValue(Object value)

为该列中的表头设置用于显示的值。

**API** **javax.swing.DefaultCellEditor 1.2**

- DefaultCellEditor(JComboBox comboBox)

构建一个单元格编辑器，并以一个组合框的形式显示出来，用于选择单元格的值。

**API** **javax.swing.table.TableCellEditor 1.2**

- Component getTableCellEditorComponent(JTable table, Object value, boolean selected, int row, int column)

返回一个构件，它的paint方法用于绘制表格的单元格。

|             |                      |
|-------------|----------------------|
| 参数: table   | 包含要绘制的单元格的表格         |
| value       | 要绘制的单元格              |
| selected    | 如果该单元格已被当前选中，则返回true |
| row, column | 单元格的行及列              |

### API javax.swing.CellEditor 1.2

- boolean isCellEditable(EventObject event)

如果该事件能够启动对该单元格的编辑过程，那么返回true。

- boolean shouldSelectCell(EventObject anEvent)

启动编辑过程。如果可以选中编辑过的单元格，则返回true。通常情况下，你希望返回的是true，不过，如果你不希望在编辑过程中改变单元格的选择，那么你可以返回false。

- void cancelCellEditing()

取消编辑过程。你可以放弃部分的编辑操作。

- boolean stopCellEditing()

处于使用编辑结果的目的，停止编辑过程。如果编辑过的值对读取来说处于适合的状态，则返回true。

- Object getCellEditorValue()

返回编辑结果。

- void addCellEditorListener(CellEditorListener l)

- void removeCellEditorListener(CellEditorListener l)

添加或移除必须的单元格编辑器的监听器。

## 6.3 树

每个使用过分层结构的文件系统的计算机用户都见过树状显示。当然，目录和文件形式仅仅是树状组织结构中的一种。日常生活中还有很多这样的树结构，例如国家、州以及城市之间的层次结构，如图6-15所示。

作为一名编程人员，我们经常需要显示这些树型结构。幸运的是，Swing类库中有一个正是用于此目的的JTree类。JTree类（以及它的辅助类）负责布局树状结构，按照用户请求展开或折叠树的节点。在本节中，我们将介绍怎样使用JTree类。

与其他复杂的Swing构件一样，我们必须集中介绍一些常用方法，无法涉及所有的细节。如果读者想获得与众不同的效果，我们推荐你参考David M. Geary撰写的《Graphic Java 2: Mastering the JFC, Volume II: Swing》（第3版），Prentice-Hall出版社1999年出版、Kim Topley编写的《Core Java Foundation Classes》，Prentice-Hall出版社1998年出版或者Kim Topley编写的《Core Swing: Advanced Programming》（Prentice-Hall出版社1999年出版）。

在我们深入展开之前，先介绍一些术语（参见图6-16）。一棵树由一些节点（node）组成。每个节点要么是叶节点（leaf）要么是有孩子节点（child node）的节点。除了根节点（root

node)，每一个节点都有一个惟一的父节点 (parent node)。一棵树只有一个根节点。有时，你可能有一个树的集合，其中每棵树都有自己的根节点。这样的集合称作森林 (forest)。

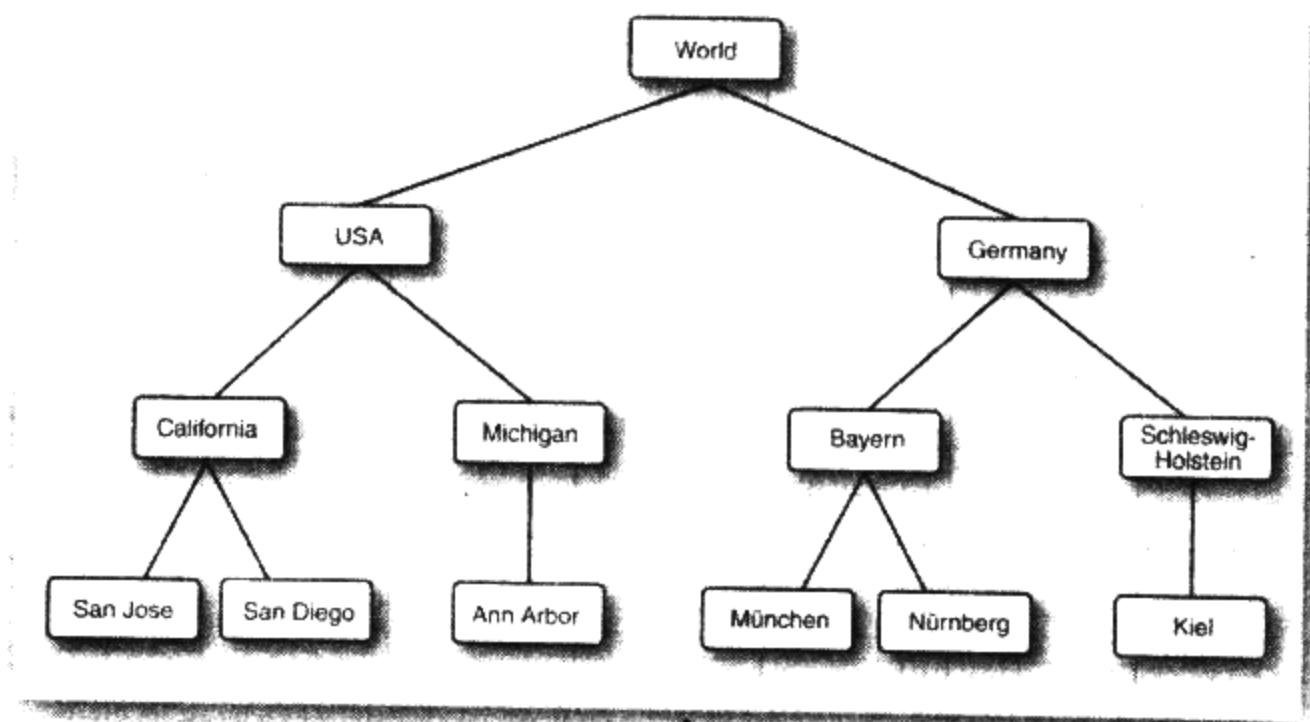


图6-15 国家、州及城市的层次结构

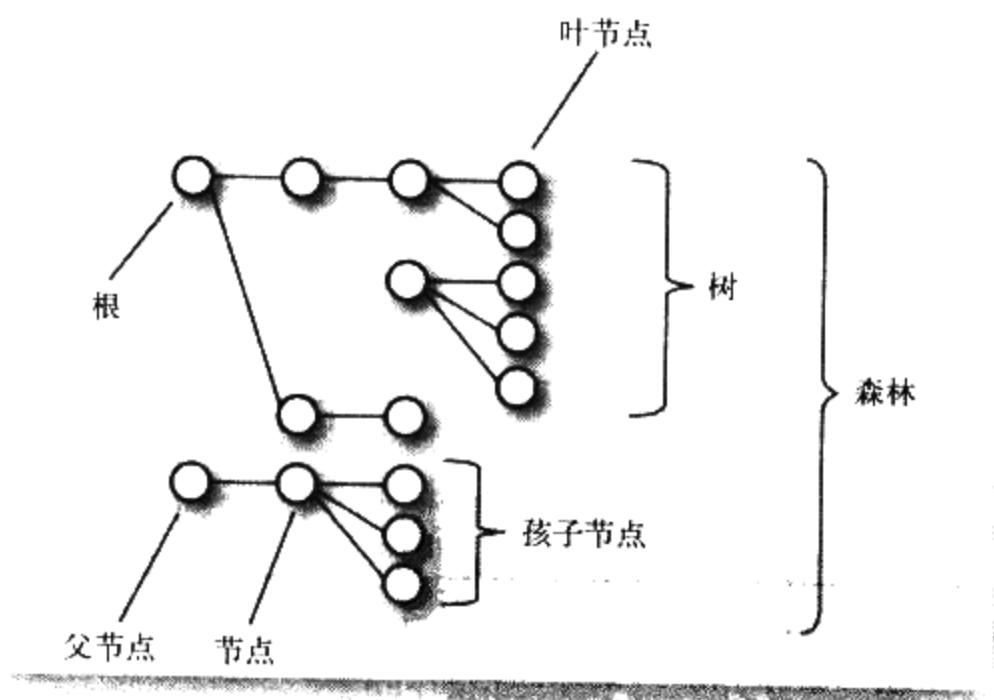


图6-16 树中的术语

### 6.3.1 简单的树

在第一个示例程序中，我们仅仅展示了一个具有几个节点的树（参见图6-18）。如同大多数Swing构件一样，只要提供一个数据模型，构件就可以将它显示出来。为了构建JTree，需要在构造器中提供这样一个树模型：

```
TreeModel model = . . .;
JTree tree = new JTree(model);
```

注意：还有一些构造器可以用一些元素的集合来构建树。

```
JTree(Object[] nodes)  
JTree(Vector<?> nodes)  
JTree(Hashtable<?, ?> nodes) // the values become the nodes
```

这些构造器不是特别有用。它们仅仅是创建出一个包含了若干棵树的森林，其中每棵树只有一个节点。第三个构造器显得特别没用，因为这些节点实际的显示次序是由散列编码的键值所提供的。

怎样才能获得一个树模型呢？可以通过创建一个实现了TreeModel接口的类来构建自己的树模型。在本章的后面部分，将会介绍应该如何实现。现在，我们仍坚持使用Swing类库提供的DefaultTreeModel模型。

为了构建一个默认的树模型，必须提供一个根节点。

```
TreeNode root = . . .;  
DefaultTreeModel model = new DefaultTreeModel(root);
```

TreeNode是另外一个接口。可以将任何实现了这个接口的类的对象填入到默认的树模型中。这里，我们使用的是Swing提供的实体节点类，叫做DefaultMutableTreeNode。这个类实现了MutableTreeNode接口，该接口是TreeNode的一个子接口（参见图6-17）。

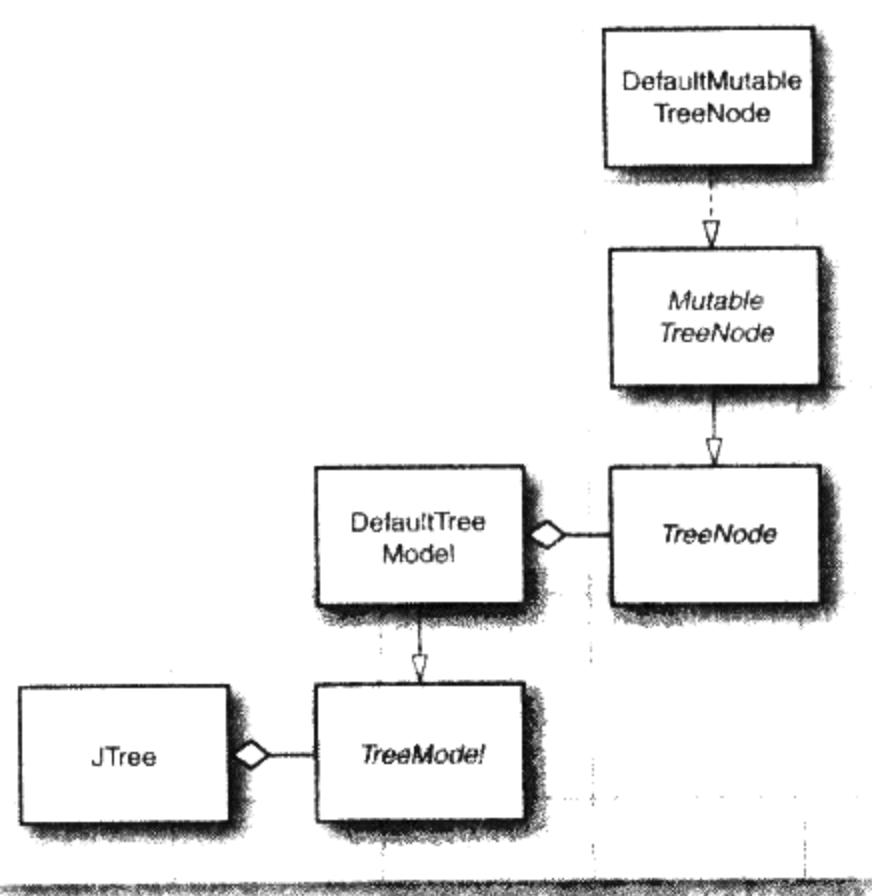


图6-17 有关树的类

任何一个默认的可变树节点都存放一个对象，即用户对象（user object）。树会为所有的节点绘制这些用户对象。除非指定一个绘制器，否则树将直接显示执行完toString方法之后的结果字符串。

在第一个示例程序中，我们使用了字符串作为用户对象。实际应用中，通常会向树中填入

更具表现力的用户对象。例如，当显示一个目录树时，将File对象用于节点将具有实际意义。可以在构造器中设定用户对象，也可以稍后在setUserObject方法中设定用户对象。

```
DefaultMutableTreeNode node = new DefaultMutableTreeNode("Texas");
...
node.setUserObject("California");
```

接下来，可以建立节点之间的父/子关系。从根节点开始，使用add方法来添加子节点：

```
DefaultMutableTreeNode root = new DefaultMutableTreeNode("World");
DefaultMutableTreeNode country = new DefaultMutableTreeNode("USA");
root.add(country);
DefaultMutableTreeNode state = new DefaultMutableTreeNode("California");
country.add(state);
```

图6-18显示了这棵树的外观。

按照这种方式将所有的节点贯穿起来。然后用根节点构建一个DefaultTreeModel。最后，用这个树模型构建一个Jtree。

```
DefaultTreeModel treeModel = new DefaultTreeModel(root);
JTree tree = new JTree(treeModel);
```

或者，使用快捷方式，直接将根节点传递给Jtree构造器。那么这棵树就会自动构建一个默认的树模型：

```
JTree tree = new JTree(root);
```

程序清单6-8给出了完整的代码。

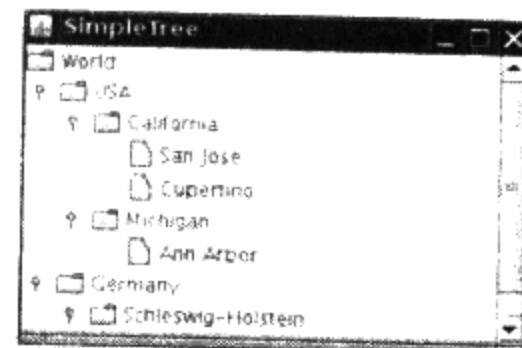


图6-18 一棵简单的树

### 程序清单6-8 SimpleTree.java

```
1. import java.awt.*;
2.
3. import javax.swing.*;
4. import javax.swing.tree.*;
5.
6. /**
7. * This program shows a simple tree.
8. * @version 1.02 2007-08-01
9. * @author Cay Horstmann
10. */
11. public class SimpleTree
12 {
13     public static void main(String[] args)
14     {
15         EventQueue.invokeLater(new Runnable()
16         {
17             public void run()
18             {
19                 JFrame frame = new SimpleTreeFrame();
20                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21                 frame.setVisible(true);
22             }
23         });
24     }
25 }
```

```
26.  
27. /**  
28. * This frame contains a simple tree that displays a manually constructed tree model.  
29. */  
30. class SimpleTreeFrame extends JFrame  
31. {  
32.     public SimpleTreeFrame()  
33.     {  
34.         setTitle("SimpleTree");  
35.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);  
36.         // set up tree model data  
37.  
38.         DefaultMutableTreeNode root = new DefaultMutableTreeNode("World");  
39.         DefaultMutableTreeNode country = new DefaultMutableTreeNode("USA");  
40.         root.add(country);  
41.         DefaultMutableTreeNode state = new DefaultMutableTreeNode("California");  
42.         country.add(state);  
43.         DefaultMutableTreeNode city = new DefaultMutableTreeNode("San Jose");  
44.         state.add(city);  
45.         city = new DefaultMutableTreeNode("Cupertino");  
46.         state.add(city);  
47.         state = new DefaultMutableTreeNode("Michigan");  
48.         country.add(state);  
49.         city = new DefaultMutableTreeNode("Ann Arbor");  
50.         state.add(city);  
51.         country = new DefaultMutableTreeNode("Germany");  
52.         root.add(country);  
53.         state = new DefaultMutableTreeNode("Schleswig-Holstein");  
54.         country.add(state);  
55.         city = new DefaultMutableTreeNode("Kiel");  
56.         state.add(city);  
57.         state.add(city);  
58.  
59.         // construct tree and put it in a scroll pane  
60.  
61.         JTree tree = new JTree(root);  
62.         add(new JScrollPane(tree));  
63.     }  
64.  
65.     private static final int DEFAULT_WIDTH = 300;  
66.     private static final int DEFAULT_HEIGHT = 200;  
67. }
```

运行这段程序代码时，最初的树外观如图6-19所示。只有根节点和它的子节点可见。点击圆形图标（把手）展开子树。当子树折叠起来时，把手图标的线伸出指向右边，当子树展开时，把手图标的线伸出指向下方（参见图6-20）。虽然我们无法得知Metal外观的设计者当时是如何构想的，但是我们可以将这个图标看作一个门把手。按下把手就可以打开子树。



**注意：**当然，树的显示还依赖于所选择的外观模式。我们这里只讨论Metal这种外观模式。在Windows或Motif外观模式中，把手则具有我们更熟悉的外观，即带有“-”或“+”框结构（参见图6-21）。

可以使用下面这句神奇的代码撤销父子节点之间的连接线（参见图6-22）：

```
tree.putClientProperty("JTree.lineStyle", "None");
```

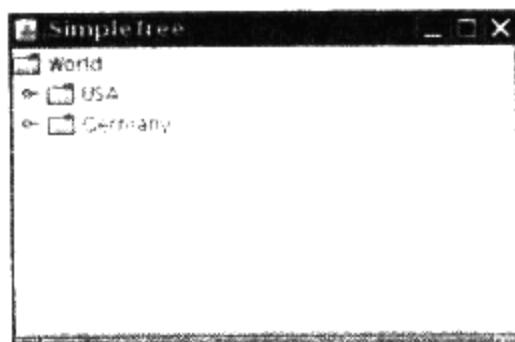


图6-19 最初的树的显示

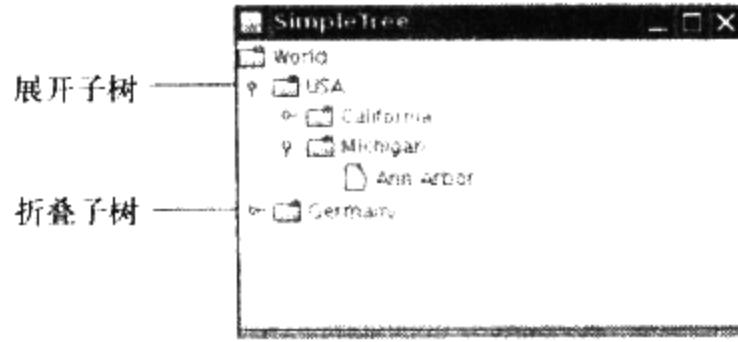


图6-20 折叠和展开后的子树

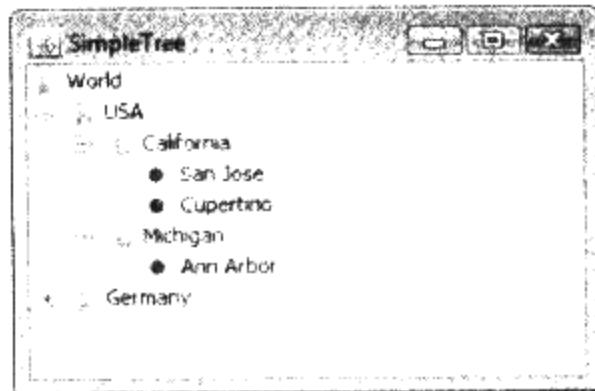


图6-21 一棵具有Windows外观的树

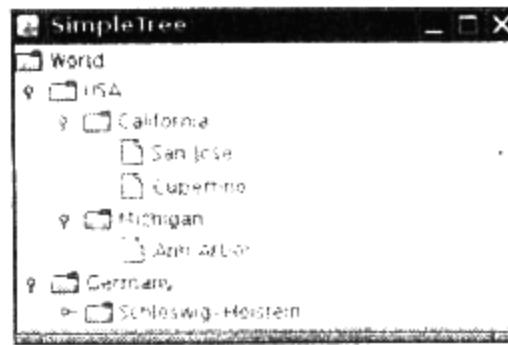


图6-22 不带连接线的树

相反地，如果要确保显示这些线条，则可以使用：

```
tree.putClientProperty("JTree.lineStyle", "Angled");
```

另一种线条样式，“水平型的”，如图6-23所示。这棵树显示有水平线，而这些水平线只是用来将根节点和孩子节点分离开来。我们很难确定这样做是好还是不好。

默认情况下，这种树中的根节点没有用于折叠的手柄。如果需要的话，可以通过下面的调用添加一个把手：

```
tree.setShowsRootHandles(true);
```

图6-24显示了调用后的结果。现在你就可以将整棵树折叠到根节点中了。

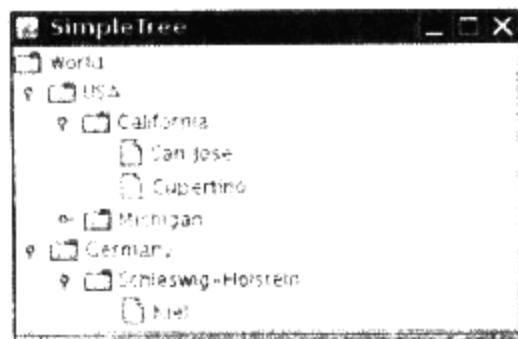


图6-23 具有水平线样式的树



图6-24 具有一个根把手的树

相反地，也可以将根节点完全隐藏起来。这样做只是为了显示一个森林，即一个树集，每棵树都有它自己的根节点。但是仍然必须将森林中的所有树都放到一个公共节点下。因此，可以使用下面这条指令将根节点隐藏起来。

```
tree.setRootVisible(false);
```

请观察图6-25。它看起来似乎有两个根节点，分别用“USA”和“Germany”标识了出来，而实际上将二者合并起来的根节点是不可见的。

让我们将注意力从树的根节点转移到叶节点。注意，这些叶节点的图标和其他节点的图标是不同的（参见图6-26）。

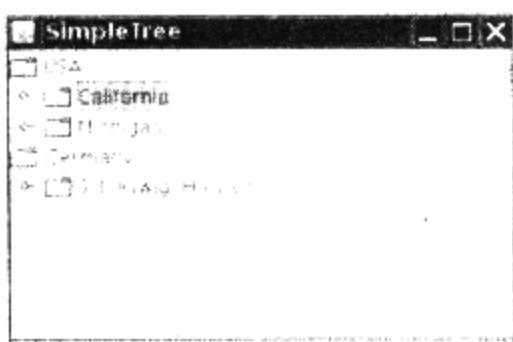


图6-25 一个森林

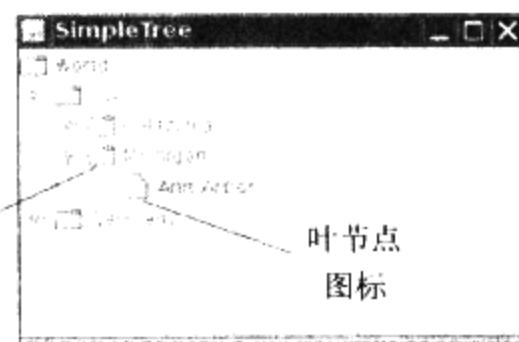


图6-26 叶节点和折叠节点的图标

在显示这棵树的时候，每个节点都绘有一个图标。实际上一共有三种图标：叶节点图标、展开的非叶节点图标以及闭合的非叶节点图标。为了简化起见，我们将后面两种图标称为文件夹图标。

节点绘制器必须知道每个节点要使用什么样的图标。默认情况下，这个决策过程是这样的：如果某个节点的`isLeaf`方法返回的是`true`，那么就使用叶节点图标，否则，使用文件夹图标。

如果某个节点没有任何儿子节点，那么`DefaultMutableTreeNode`类的`isLeaf`方法将返回`true`。因此，具有儿子节点的节点使用文件夹图标，没有儿子节点的节点使用叶节点图标。

有时，这种做法并不合适。假设我们要向我们那棵简单的树中添加一个“Montana”节点，但是我们还不知道要添加什么城市。我们并不希望一个州节点使用叶节点图标，因为从概念上来说，只有城市才使用叶节点。

`JTree`类无法知道哪些节点是叶节点，它要询问树模型。如果一个没有任何子节点的节点不应该自动地被设置为概念上的叶节点，那么可以让树模型对这些叶节点使用一个不同的标准，即可以查询“允许子节点”的节点属性。

对于那些不应该有子节点的节点，调用

```
node.setAllowsChildren(false);
```

然后，告诉树模型去查询“允许子节点”的属性值以确定一个节点是否应该显示成叶子图标。你可以使用`DefaultTreeModel`类中的方法`setAsksAllowsChildren`设定此动作：

```
model.setAsksAllowsChildren(true);
```

有了这个判定规则，允许有子节点的节点就可以获得文件夹图标，而不允许有子节点的节点将获得叶子图标。

另外，如果你是通过提供根节点来构建一棵树的，那么请在构造器中直接提供“询问允许子节点”属性值的设置。

```
JTree tree = new JTree(root, true); // nodes that don't allow children get leaf icons
```



`javax.swing.JTree 1.2`

- `Jtree(TreeModel model)`

根据一个树模型构造一棵树。

- `Jtree(TreeNode root)`
- `Jtree(TreeNode root, boolean asksAllowChildren)`

使用默认的树模型构造一棵树，显示根节点和它的子节点。

参数：root 根节点  
asksAllowChildren 如果设置为true，则使用“允许子节点”的节点属性来确定一个节点是否是叶节点

- `void setShowsRootHandles(boolean b)`  
如果b为true，则根节点具有折叠或展开它的子节点的把手图标。
- `void setRootVisible(boolean b)`  
如果b为true，则显示根节点，否则隐藏根节点。

#### API `javax.swing.tree.TreeNode` 1.2

- `boolean isLeaf()`  
如果该节点是一个概念上的叶节点，则返回true。
- `boolean getAllowsChildren()`  
如果该节点可以拥有子节点，则返回true。

#### API `javax.swing.tree.MutableTreeNode` 1.2

- `void setUserObject(Object userObject)`  
设置树节点用于绘制的“用户对象”。

#### API `javax.swing.tree.TreeNode` 1.2

- `boolean isLeaf(object node)`  
如果该节点应该以叶节点的形式显示，则返回true。

#### API `javax.swing.tree.DefaultTreeModel` 1.2

- `void setAsksAllowsChildren(boolean b)`  
如果b为true，那么当节点的getAllowsChildren方法返回false时，这些节点显示为叶节点。否则，当节点的isLeaf方法返回true时，它们显示为叶节点。

#### API `javax.swing.tree.DefaultMutableTreeNode` 1.2

- `DefaultMutableTreeNode (object userObject)`  
用给定的用户对象构建一个可变树节点。
- `void add(MutableTreeNode child)`  
向一个节点添加为该节点最后一个子节点。
- `void setAllowsChildren(boolean b)`  
如果b为true，则可以向该节点添加子节点。



### javax.swing.JComponent 1.2

- void putClientProperty(Object key, Object value)

将一个键/值对添加到一个小表格中，每一个构件都管理着这样的一个小表格。这是一种“紧急逃生”机制，很多Swing构件用它来存放具体的外观属性。

#### 编辑树和树的路径

在下面的一个示例程序中，将会看到怎样编辑一棵树。图6-27显示了用户界面。如果点击“Add Sibling”（添加兄弟节点）或“Add Child”（添加子节点）按钮，该程序将向树中添加一个新节点（带有“New”标题）。如果你点击“Delete”（删除）按钮，该程序将删除当前选中的节点。

为了实现这种行为，需要弄清楚当前选定的是哪个节点。JTree类用的是一种令人惊讶的方式来标识树中的节点。它并不处理树的节点，而是处理对象路径（称为树路径）。一个树路径从根节点开始，由一个子节点序列构成，参见图6-28。

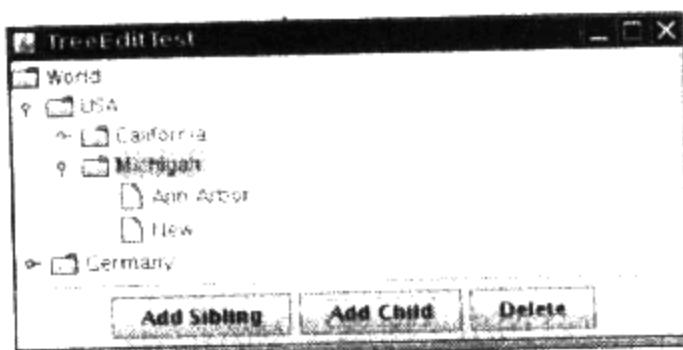


图6-27 编辑一棵树

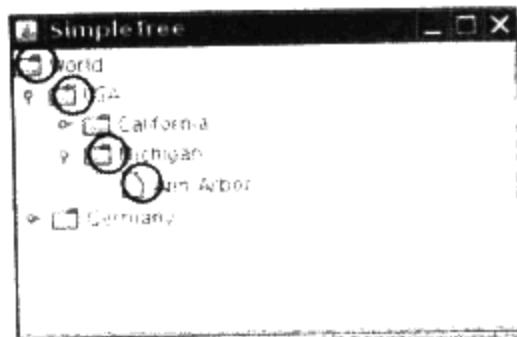


图6-28 一个树路径

你可能要怀疑JTree类为什么需要整个路径。它不能只获得一个TreeNode，然后不断调用getParent方法吗？实际上，JTree类一点都不清楚TreeNode接口的情况。该接口从来没有被TreeModel接口用到过，它只被DefaultTreeModel的实现用到了。你完全可以拥有其他的树模型，这些树模型中的节点可能根本就没有实现TreeNode接口。如果你使用的是一个管理其他类型对象的树模型，那么这些对象有可能根本就没有getParent和getChild方法。它们彼此之间当然会有其他某种连接。将其他节点连接起来这是树模型的职责，JTree类本身并没有节点之间连接属性的任何线索。因此，JTree类总是需要用完整的路径来运作。

TreePath类管理着一个Object（不是TreeNode！）引用序列。有很多JTree的方法都可以返回TreePath对象。当拥有一个树路径时，通常只需要知道其终端节点，该节点可以通过getLastPathComponent方法得到。例如，如果要查找一棵树中当前选定的节点，可以使用JTree类中的getSelectionPath方法。它将返回一个对象，根据这个对象就可以检索实际节点。

```
TreePath selectionPath = tree.getSelectionPath();
DefaultMutableTreeNode selectedNode
    = (DefaultMutableTreeNode) selectionPath.getLastPathComponent();
```

实际上，由于这种特殊查询经常被使用到，因此还提供了一个更方便的方法，它能够立即给出选定的节点。

```
DefaultMutableTreeNode selectedNode
    = (DefaultMutableTreeNode) tree.getLastSelectedPathComponent();
```

该方法之所以没有被称为getSelectedNode，是因为这棵树并不了解它包含的节点，它的树模型只处理对象的路径。

 **注意：**树路径是JTree类描述节点的两种方式之一。JTree有许多方法可以接收或返回一个整数索引——行的位置。行的位置仅仅是节点在树中显示的一个行号（从0开始）。只有那些可视节点才有行号，并且如果一个节点之前的其他节点展开、折叠或者被修改过，这个节点的行号也会随之改变。因此，你应该避免使用行的位置。相反地，所有使用行的JTree方法都有一个与之等价的使用树路径的方法。

一旦你持有一个选定了的节点，那么就可以对它进行编辑了。不过，不能直接向树节点添加子节点：

```
selectedNode.addNode(newNode); // NO!
```

如果你改变了节点的结构，那么改变的只是树模型，而相关的视图却没有被通知到。可以自己发送一个通知消息，但是如果使用DefaultTreeModel类的insertNodeInto方法，那么该模型类会全权负责这件事情。例如，下面的调用可以将一个新节点作为选定节点的最后子节点添加到树中，并通知树的视图。

```
model.insertNodeInto(newNode, selectedNode, selectedNode.getChildCount());
```

类似的调用removeNodeFromParent可以移除一个节点并通知树的视图：

```
model.removeNodeFromParent(selectedNode);
```

如果想保持节点结构，但是要改变用户对象，那么可以调用下面这个方法：

```
model.nodeChanged(changedNode);
```

自动通知是使用DefaultTreeModel的主要优势。如果你提供自己的树模型，那么必须自己动手实现这种自动通知。（详见Kim Topley撰写的《Core Java Foundation Classes》。）

 **警告：**DefaultTreeModel有一个reload方法能够将整个模型重新载入。但是，不要在进行了少数几个修改之后，只是为了更新树而调用reload方法。在重建一棵树的时候，根节点的子节点之后的所有节点将全部再次折叠起来。如果你的用户在每次修改之后都要不断地展开整棵树，这确实是一件令人很不安的事。

当视图接收到节点结构被改变的通知时，它会更新显示树的视图，但是不会自动展开某个节点以展现新添加的子节点。特别是在我们上面那个示例程序中，如果用户将一个新节点添加到其子节点正处于折叠状态的节点上，那么这个新添加的节点就被悄无声息地添加到了一个处于折叠状态的子树中。没有给用户提供任何反馈信息以告诉用户已经实施了该命令。在这种情况下，你可能需要特别费劲地展开所有的父节点，以便让新添加的节点成为可视节点。可以使用类JTree中的方法makeVisible实现这个目的。makeVisible方法通过一个树路径让某个节点变成可见的。

因此，你需要构建一个从根节点到新添加节点的树路径。为了获得一个这样的树路径，首先要调用DefaultTreeModel类中的getPathToRoot方法，它返回一个包含了某一节点到根节点之间所有节点的数组TreeNode[]。可以将这个数组传递给一个TreePath构造器。

例如，下面展示了怎样将一个新节点变成可见的：

```
TreeNode[] nodes = model.getPathToRoot(newNode);
TreePath path = new TreePath(nodes);
tree.makeVisible(path);
```

 **注意：**令人惊奇的是，DefaultTreeModel类好像完全忽视了TreePath类，尽管它的职责是与一个JTree通信。JTree类大量地使用到了树路径，而它从不使用节点对象数组。

但是，现在假设你的树是放在一个滚动面板里面。展开树节点之后，新节点仍是不可见的，因为它落在视图之外。为了克服这个问题，请调用

```
tree.scrollPathToVisible(path);
```

而不是调用makeVisible。这个调用将展开路径中的所有节点，并告诉外围的滚动面板将路径末端的节点滚动到视图中（参见图6-29）。

默认情况下，这些树节点是不可编辑的。不过，如果调用

```
tree.setEditable(true);
```

那么，用户就可以编辑某一节点了。可以先双击该节点，然后编辑字符串，最后按下回车键。双击操作会调用默认单元格编辑器，它实现了DefaultCellEditor类（参见图6-30）。也可以安装其他一些单元格编辑器，其过程与表格单元格编辑器中讨论的过程一样。

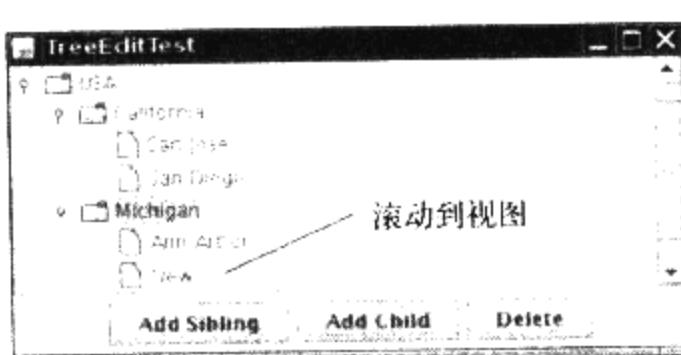


图6-29 滚动以显示新节点的滚动面板

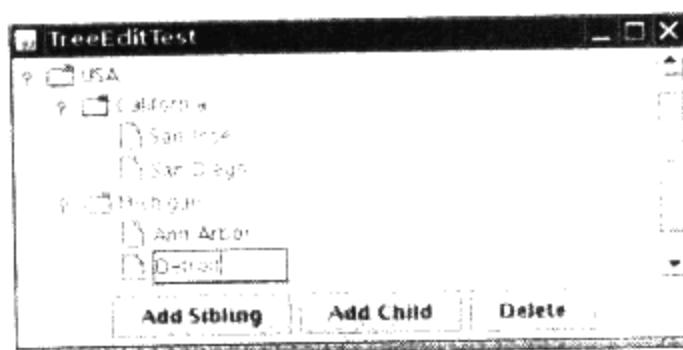


图6-30 默认的单元格编辑器

程序清单6-9展示了树编辑程序的完整源代码。运行该程序，添加几个新节点，然后通过双击它们进行编辑操作。请观察折叠的节点是怎样展开以显现添加的子节点的，以及滚动面板是怎样让添加的节点保持在视图中的。

#### 程序清单6-9 TreeEditTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.swing.*;
4. import javax.swing.tree.*;

5.
6. /**
7. * This program demonstrates tree editing.
8. * @version 1.03 2007-08-01
9. * @author Cay Horstmann
10.*/
11. public class TreeEditTest
12. {
13.     public static void main(String[] args)
```

```
14  {
15      EventQueue.invokeLater(new Runnable()
16      {
17          public void run()
18          {
19              JFrame frame = new TreeEditFrame();
20              frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21              frame.setVisible(true);
22          }
23      });
24  }
25 }
26
27 /**
28 * A frame with a tree and buttons to edit the tree.
29 */
30 class TreeEditFrame extends JFrame
31 {
32     public TreeEditFrame()
33     {
34         setTitle("TreeEditTest");
35         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
36
37         // construct tree
38
39         TreeNode root = makeSampleTree();
40         model = new DefaultTreeModel(root);
41         tree = new JTree(model);
42         tree.setEditable(true);
43
44         // add scroll pane with tree
45
46         JScrollPane scrollPane = new JScrollPane(tree);
47         add(scrollPane, BorderLayout.CENTER);
48
49         makeButtons();
50     }
51
52     public TreeNode makeSampleTree()
53     {
54         DefaultMutableTreeNode root = new DefaultMutableTreeNode("World");
55         DefaultMutableTreeNode country = new DefaultMutableTreeNode("USA");
56         root.add(country);
57         DefaultMutableTreeNode state = new DefaultMutableTreeNode("California");
58         country.add(state);
59         DefaultMutableTreeNode city = new DefaultMutableTreeNode("San Jose");
60         state.add(city);
61         city = new DefaultMutableTreeNode("San Diego");
62         state.add(city);
63         state = new DefaultMutableTreeNode("Michigan");
64         country.add(state);
65         city = new DefaultMutableTreeNode("Ann Arbor");
66         state.add(city);
67         country = new DefaultMutableTreeNode("Germany");
68         root.add(country);
69         state = new DefaultMutableTreeNode("Schleswig-Holstein");
```

```
70     country.add(state);
71     city = new DefaultMutableTreeNode("Kiel");
72     state.add(city);
73     return root;
74 }
75
76 /**
77 * Makes the buttons to add a sibling, add a child, and delete a node.
78 */
79 public void makeButtons()
80 {
81     JPanel panel = new JPanel();
82     JButton addSiblingButton = new JButton("Add Sibling");
83     addSiblingButton.addActionListener(new ActionListener()
84     {
85         public void actionPerformed(ActionEvent event)
86         {
87             DefaultMutableTreeNode selectedNode = (DefaultMutableTreeNode) tree
88                 .getLastSelectedPathComponent();
89
90             if (selectedNode == null) return;
91
92             DefaultMutableTreeNode parent = (DefaultMutableTreeNode)
93                 selectedNode.getParent();
94
95             if (parent == null) return;
96
97             DefaultMutableTreeNode newNode = new DefaultMutableTreeNode("New");
98
99             int selectedIndex = parent.getIndex(selectedNode);
100            model.insertNodeInto(newNode, parent, selectedIndex + 1);
101
102            // now display new node
103
104            TreeNode[] nodes = model.getPathToRoot(newNode);
105            TreePath path = new TreePath(nodes);
106            tree.scrollPathToVisible(path);
107        }
108    });
109    panel.add(addSiblingButton);
110
111    JButton addChildButton = new JButton("Add Child");
112    addChildButton.addActionListener(new ActionListener()
113    {
114        public void actionPerformed(ActionEvent event)
115        {
116            DefaultMutableTreeNode selectedNode = (DefaultMutableTreeNode) tree
117                .getLastSelectedPathComponent();
118
119            if (selectedNode == null) return;
120
121            DefaultMutableTreeNode newNode = new DefaultMutableTreeNode("New");
122            model.insertNodeInto(newNode, selectedNode, selectedNode.getChildCount());
123
124            // now display new node
125        }
```

```

126.         TreeNode[] nodes = model.getPathToRoot(newNode);
127.         TreePath path = new TreePath(nodes);
128.         tree.scrollPathToVisible(path);
129.     }
130. });
131. panel.add(addChildButton);
132.
133. JButton deleteButton = new JButton("Delete");
134. deleteButton.addActionListener(new ActionListener()
135. {
136.     public void actionPerformed(ActionEvent event)
137.     {
138.         DefaultMutableTreeNode selectedNode = (DefaultMutableTreeNode) tree
139.             .getLastSelectedPathComponent();
140.
141.         if (selectedNode != null && selectedNode.getParent() != null) model
142.             .removeNodeFromParent(selectedNode);
143.     }
144. });
145. panel.add(deleteButton);
146. add(panel, BorderLayout.SOUTH);
147. }
148.
149. private DefaultTreeModel model;
150. private JTree tree;
151. private static final int DEFAULT_WIDTH = 400;
152. private static final int DEFAULT_HEIGHT = 200;
153. }

```

### **API** javax.swing.JTree 1.2

- **TreePath getSelectionPath()**

获取到当前选定节点的路径，如果选定多个节点，则获取到第一个选定节点的路径。如果没有选定任何节点，则返回null。

- **Object getLastSelectedPathComponent()**

获取表示当前选定节点的节点对象，如果选定多个节点，则获取第一个选定的节点。如果没有选定任何节点，则返回null。

- **void makeVisible(TreePath path)**

展开该路径中的所有节点。

- **void scrollPathToVisible(TreePath path)**

展开该路径中的所有节点，如果这棵树是置于滚动面板中的，则滚动以确保该路径中的最后一个节点是可见的。

### **API** javax.swing.tree.TreePath 1.2

- **Object getLastPathComponent()**

获取该路径中最后一个节点，也就该路径代表的节点对象。

**API** **javax.swing.tree.TreeNode 1.2**

- **TreeNode getParent()**  
返回该节点的父节点。
- **TreeNode getChildAt(int index)**  
查找给定索引号上的子节点。该索引号必须在0和getChildCount() - 1之间。
- **int getChildCount()**  
返回该节点的子节点个数。
- **Enumeration children()**  
返回一个枚举对象，可以迭代遍历该节点的所有子节点。

**API** **javax.swing.tree.DefaultTreeNode 1.2**

- **void insertNodeInto(MutableTreeNode newChild, MutableTreeNode parent, int index)**  
将newChild作为parent的新子节点添加到给定的索引位置上，并通知树模型的监听器。
- **void removeNodeFromParent(MutableTreeNode node)**  
将节点node从该模型中删除，并通知树模型的监听器。
- **void nodesChanged(TreeNode node)**  
通知树模型的监听器：节点node发生了改变。
- **void nodesChanged(TreeNode parent, int[] changedChildIndexes)**  
通知树模型的监听器：节点parent所有在给定索引位置上的子节点发生了改变。
- **void reload()**  
将所有节点重新载入到树模型中。这是一项动作剧烈的操作，只有当由于一些外部作用，导致树的节点完全改变时，才应该使用该方法。

### 6.3.2 节点枚举

有时为了查找树中一个节点，必须从根节点开始，遍历所有子节点直到找到相匹配的节点。**DefaultMutableTreeNode**类有几个很方便的方法用于迭代遍历所有节点。

使用广度优先或深度优先的遍历方式，**breadthFirstEnumeration**方法和**depthFirstEnumeration**方法可以返回枚举对象，它们的**nextElement**方法能够访问当前节点的所有子节点。图6-31显示了对示例树进行遍历的情况，节点标签则指示遍历节点时的先后次序。

按照广度优先的方式进行枚举是最容易想到的。树是以层的形式遍历的，首先访问根节点，然后是它的所有子节点，接着是它的孙子节点，依此类推。

为了形象地说明深度优先的枚举，让我们想像一只老鼠陷入一个树状陷阱的情形。它沿着第一条路径迅速爬行，直到它到达一个叶节点位置。然后，原路返回并转入下一条路径，依此类推。

计算机科学家也将其称为后序遍历（postorder traversal），因为整个查找过程是先访问到子节点，然后才访问到父节点。**postOrderTraversal**方法是**depthFirstTraversal**的同义语。为

为了完整性，还存在一个`preOrderTraversal`方法，它是一种深度优先搜索方法，首先枚举父节点，然后是子节点。

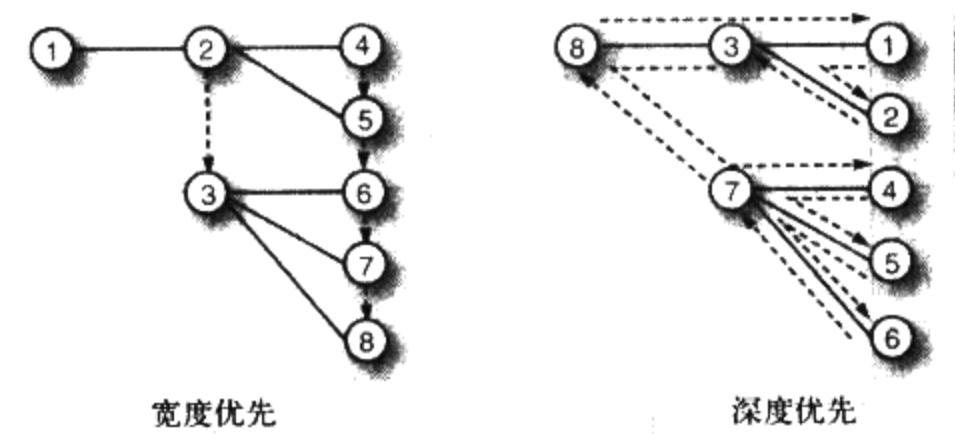


图6-31 树的遍历顺序

下面是一个典型的使用模式：

```
Enumeration breadthFirst = node.breadthFirstEnumeration();
while (breadthFirst.hasMoreElements())
    do something with breadthFirst.nextElement();
```

最后，还有一个相关方法`pathFromAncestor Enumeration`，用于查找一条从祖先节点到给定节点之间的路径，然后枚举出该路径中的所有节点。整个过程并不需要大量的处理操作，只需要不断调用`getParent`直到发现祖先节点，然后将该路径倒置过来存放即可。

在我们的下个示例程序中，将运用到节点枚举。该程序显示了类之间的继承树。向窗体最下面的文本框中输入一个类名，该类以及它的子类就会添加到树中（参见图6-32）。

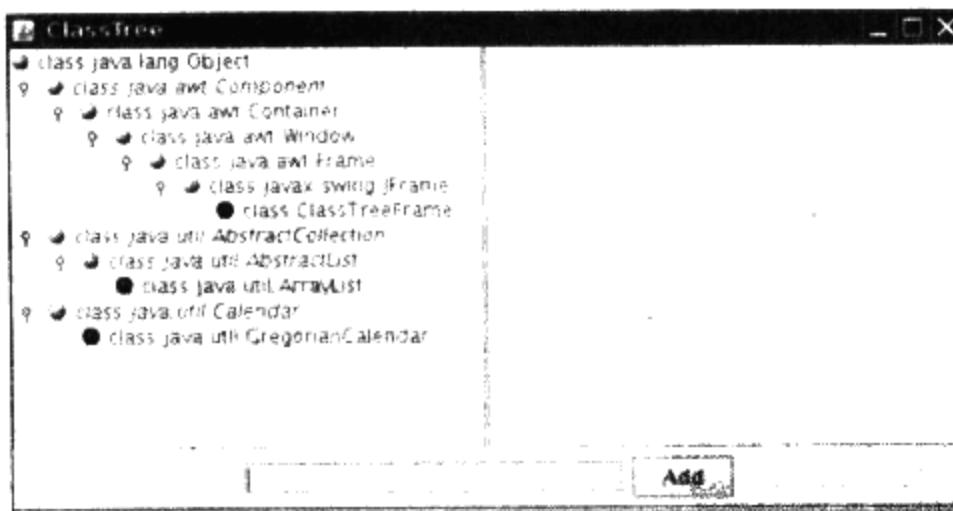


图6-32 一棵继承树

在这个示例中，我们充分利用了这个事实，即树节点的用户对象可以是任何类型的对象。因为我们这里的节点是用来描述类的，因此我们在这些节点中存储的是`Class`对象。

当然，我们不想对同一个类对象添加两次，因此我们必须检查一个类是否已经存在于树中。如果在树中存在给定用户对象的节点，那么下面这个方法就可以用来查找该节点。

```
public DefaultMutableTreeNode findUserObject(Object obj)
{
```

```
Enumeration e = root.breadthFirstEnumeration();
while (e.hasMoreElements())
{
    DefaultMutableTreeNode node = (DefaultMutableTreeNode) e.nextElement();
    if (node.getUserObject().equals(obj))
        return node;
}
return null;
}
```

### 6.3.3 绘制节点

在应用中可能会经常需要改变树构件绘制节点的方式。最常用到的改变当然是为节点和叶节点选取不同的图标。其他一些改变可能涉及节点标签的字体或节点上的图像绘制等方面。所有这些改变都可以通过向树中安装一个新的树单元格绘制器来实现。在默认情况下，`JTree`类使用`DefaultTreeCellRenderer`对象来绘制每个节点。`DefaultTreeCellRenderer`类继承自`JLabel`类。该标签包含节点图标和节点标签。

 **注意：**单元格绘制器并不能绘制用于展开或折叠子树的“把手”图标。这些把手是外观模式的一部分，建议最好不要试图改变它们。

可以通过以下三种方式定制显示外观：

1. 可以使用`DefaultTreeCellRenderer`改变图标、字体以及背景颜色。这些设置适用于树中所有节点。
2. 可以安装一个继承了`DefaultTreeCellRenderer`类的绘制器，用于改变每个节点的图标、字体以及背景颜色。
3. 可以安装一个实现了`treeCellRenderer`接口的绘制器，为每个节点绘制自定义的图像。

让我们逐个研究这几种可能。最简单的定制方法是构建一个`DefaultTreeCellRenderer`对象，改变图标，然后将它安装到树中：

```
DefaultTreeCellRenderer renderer = new DefaultTreeCellRenderer();
renderer.setLeafIcon(new ImageIcon("blue-ball.gif")); // used for leaf nodes
renderer.setClosedIcon(new ImageIcon("red-ball.gif")); // used for collapsed nodes
renderer.setOpenIcon(new ImageIcon("yellow-ball.gif")); // used for expanded nodes
tree.setCellRenderer(renderer);
```

可以在图6-32中看到运行效果。我们只是使用“球”图标作为占位符，这里假设你的用户界面设计者会为你的应用提供合适的图标。

我们不建议改变整棵树中的字体或背景颜色，因为这实际上是外观设置的职责所在。

不过，改变树中个别节点的字体，以突显某些节点还是很有用的。如果仔细观察图6-32，你会看到抽象类是设成斜体字的。

为了改变单个节点的外观，须要安装一个树单元格绘制器。树单元格绘制器与我们在本章前一节讨论的列表单元格绘制器很相似。`TreeCellRenderer`接口只有下面这个单一方法：

```
Component getTreeCellRendererComponent(JTree tree, Object value, boolean selected,
    boolean expanded, boolean leaf, int row, boolean hasFocus)
```

`DefaultTreeCellRenderer`类的`getTreeCellRendererComponent`方法返回的是`this`，换句

话说，就是一个标签。（`DefaultTreeCellRenderer`类继承了`JLabel`类。）如果要定制一个构件，需要继承`DefaultTreeCellRenderer`类。按照以下方式覆盖`getTreeCellRendererComponent`方法：调用超类中的方法，以便准备标签的数据。然后定制标签属性，最后返回`this`。

```
class MyTreeCellRenderer extends DefaultTreeCellRenderer
{
    public Component getTreeCellRendererComponent(JTree tree, Object value, boolean selected,
        boolean expanded, boolean leaf, int row, boolean hasFocus)
    {
        super.getTreeCellRendererComponent(tree, value, selected, expanded, leaf, row, hasFocus);
        DefaultMutableTreeNode node = (DefaultMutableTreeNode) value;
        lookAtNode.getUserObject();
        Font font = appropriateFont;
        setFont(font);
        return this;
    }
};
```

**X** 警告：`getTreeCellRendererComponent`方法的`value`参数是节点对象，而不是用户对象！请记住，用户对象是`DefaultMutableTreeNode`的一个特性，而`JTree`可以包含任意类型的节点。如果树使用的是`DefaultMutableTreeNode`节点，那么必须在第二个步骤中获取这个用户对象，正如我们在上一个代码示例中所做的那样。

**X** 警告：`DefaultTreeCellRenderer`为所有节点使用的是相同的标签对象，仅仅是为每个节点改变标签文本而已。如果想为某个特定节点更改字体，那么必须在该方法再次调用的时候将它设置回默认值。否则，随后的所有节点都会以更改过的字体进行绘制！见程序清单6-10中的程序代码，看看它是怎样将字体恢复到其默认值的。

我们没有给出有关用来绘制任意图形的树单元格绘制器的示例。如果你需要这个功能，可以参考程序清单6-3中的列表单元格绘制器；它们用到的技术完全相似。

根据`Class`对象的`ABSTRACT`修饰符，程序清单6-10中的`ClassNameTreeCellRenderer`将类名设置为标准字体或斜体字体。我们不想设置成特殊的字体，因为我们不想改变通常用于显示标签的任何字体外观。因此，我们使用来自于标签本身的字体以及从它衍生而来的一个斜体字体。请回忆一下，全部的调用只返回一个共享的单一的`JLabel`对象。因此，我们需要保存初始字体，并在下一次调用`getTreeCellRendererComponent`方法时将其恢复为初始值。

同时，注意一下我们是如何改变`ClassTreeFrame`构造器中的节点图标的。

#### API `javax.swing.tree.DefaultMutableTreeNode 1.2`

- `Enumeration breadthFirstEnumeration()`
- `Enumeration depthFirstEnumeration()`
- `Enumeration preOrderEnumeration()`
- `Enumeration postOrderEnumeration()`

返回枚举对象，用于按照某种特定顺序访问树模型中所有节点的。在广度优先遍历中，先访问离根节点更近的子节点，再访问那些离根节点远的节点。在深度优先遍历中，先

访问一个节点的所有子节点，然后再访问它的兄弟节点。`postOrderEnumeration`方法与`depthFirstEnumeration`基本上相似。除了先访问父节点，后访问子节点之外，先序遍历和后序遍历基本上一样。

#### **API** javax.swing.tree.TreeCellRenderer 1.2

- `Component getTreeCellRendererComponent(JTree tree, Object value, boolean selected, boolean expanded, boolean leaf, int row, boolean hasFocus)`  
返回一个paint方法被调用的构件，以便绘制树的一个单元格。

参数：`tree` 包含要绘制节点的树

`value` 要绘制的节点

`selected` 如果该节点是当前选定的节点，则为true

`expanded` 如果该节点的子节点可见，则为true

`leaf` 如果该节点应该显示为叶节点，则为true

`row` 显示包含该节点的那行

`hasFocus` 如果当前选定的节点拥有输入焦点，则为true

#### **API** javax.swing.tree.DefaultTreeCellRenderer 1.2

- `void setLeafIcon(Icon icon)`
- `void setOpenIcon(Icon icon)`
- `void setClosedIcon(Icon icon)`

设置叶节点、展开节点以及折叠节点的显示图标。

### 6.3.4 监听树事件

通常情况下，一个树构件成对伴随着其他某个构件。当用户选定了一些树节点时，某些信息就会在其他窗口中显示出来。参见图6-33的示例。当用户选定一个类时，这个类的实例及静态变量信息就会在右边的文本区显示出来。

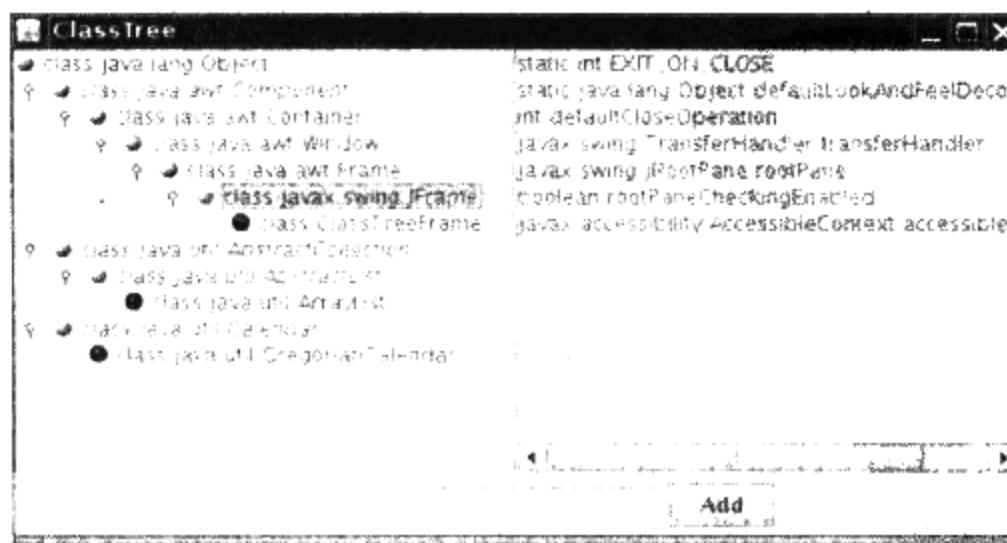


图6-33 一个类浏览器

为了获得这项功能，你可以安装一个树选择监听器。该监听器必须实现`treeSelectionListener`

接口，这是一个只有下面这个单一方法的接口：

```
void valueChanged(TreeSelectionEvent event)
```

每当用户选定或者撤销选定树节点的时候，这个方法就会被调用。

可以按照下面这种通常方式向树中添加监听器：

```
tree.addTreeSelectionListener(listener);
```

可以设定是否允许用户选定一个单一的节点、连续区间内的节点或者一个任意的、可能不连续的节点集。JTree类使用TreeSelectionModel来管理节点的选择。必须检索整个模型，以便将选择状态设置为SINGLE\_TREE\_SELECTION、CONTIGUOUS\_TREE\_SELECTION或DISCONTIGUOUS\_TREE\_SELECTION三种状态之一。（在默认情况下是非连续的选择模式。）例如，在我们的类浏览器中，我们希望只允许选择单个类：

```
int mode = TreeSelectionModel.SINGLE_TREE_SELECTION;  
tree.getSelectionModel().setSelectionMode(mode);
```

除了设置选择模式之外，你并不需要担心树的选择模型。

 **注意：** 用户怎样选定多个选项则依赖于外观。在Metal外观中，按下CTRL键，同时点击一个选项将它添加到选项集中，如果当前已经选定了该选项，则将其从选项集中删除。按下SHIFT键，同时点击一个选项，可以选定一个选项范围，它从先前已选定的选项延伸到新选定的选项。

要找出当前的选项集，可以用getSelectionPaths方法来查询树：

```
TreePath[] selectedPaths = tree.getSelectionPaths();
```

如果想限制用户只能做单项选择，那么可以使用便捷的getSelectionPath方法，它将返回第一个被选择的路径，或者是null（如果没有任何路径被选）。

 **警告：** TreeSelectionEvent类具有一个getPaths方法，它将返回一个treePath对象数组，但是该数组描述的是选项集的变化，而不是当前的选项集。

程序清单6-6显示了类树这个程序的完整源代码。该程序可以显示继承的层次结构，并且将抽象类指定显示为斜体字。可以在窗体下面的文本框中键入任何类名。按下“Enter”键或者点击“Add”按钮，将该类及其超类添加到树中。必须输入完整的包名，例如java.util.ArrayList。

这个程序用到了一点小小的技巧，它是通过反射机制来构建这棵类树的。这项操作包含在addClass方法内。（细节倒不那么重要，在这个例子中，我们之所以使用类树，是因为继承树不需要怎么费劲地编码就能生成一棵丰富的树。如果想在自己的应用中显示这棵树，那么你需要拥有自己的层次结构数据的来源。）该方法使用广度优先的搜索算法，通过调用我们在前一节实现的findUserObject方法，来确定当前的类是否已经存在于树中。如果这个类还不存在于树中，那么我们将超类添加到这棵树中，然后将新节点作为它的子节点，并使该节点成为可见的。

在选择树的一个节点时，右侧的文本域将填充为选中的类的域。在窗体构造器中，限制用户只能进行单个选项的选择，并添加了一个树选择监听器。当调用valueChanged方法时，我们忽略它的事件参数，只向该树询问当前的选定路径。正如通常情况那样，我们必须获得路径中

的最后一个节点，并且查看它的用户对象。然后调用`getFieldDescription`方法，该方法使用反射机制将所选类的所有属性字段组装成一个字符串。

**程序清单6-10 ClassTree.java**

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.lang.reflect.*;
4. import java.util.*;
5. import javax.swing.*;
6. import javax.swing.event.*;
7. import javax.swing.tree.*;
8.
9. /**
10. * This program demonstrates cell rendering and listening to tree selection events.
11. * @version 1.03 2007-08-01
12. * @author Cay Horstmann
13. */
14. public class ClassTree
15. {
16.     public static void main(String[] args)
17.     {
18.         EventQueue.invokeLater(new Runnable()
19.         {
20.             public void run()
21.             {
22.                 JFrame frame = new ClassTreeFrame();
23.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24.                 frame.setVisible(true);
25.             }
26.         });
27.     }
28. }
29.
30. /**
31. * This frame displays the class tree, a text field and add button to add more classes
32. * into the tree.
33. */
34. class ClassTreeFrame extends JFrame
35. {
36.     public ClassTreeFrame()
37.     {
38.         setTitle("ClassTree");
39.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
40.
41.         // the root of the class tree is Object
42.         root = new DefaultMutableTreeNode(java.lang.Object.class);
43.         model = new DefaultTreeModel(root);
44.         tree = new JTree(model);
45.
46.         // add this class to populate the tree with some data
47.         addClass(getClass());
48.
49.         // set up node icons
50.         ClassNameTreeCellRenderer renderer = new ClassNameTreeCellRenderer();
51.         renderer.setClosedIcon(new ImageIcon("red-ball.gif"));
```

```
52.     renderer.setOpenIcon(new ImageIcon("yellow-ball.gif"));
53.     renderer.setLeafIcon(new ImageIcon("blue-ball.gif"));
54.     tree.setCellRenderer(renderer);
55.
56.     // set up selection mode
57.     tree.addTreeSelectionListener(new TreeSelectionListener()
58.     {
59.         public void valueChanged(TreeSelectionEvent event)
60.         {
61.             // the user selected a different node--update description
62.             TreePath path = tree.getSelectionPath();
63.             if (path == null) return;
64.             DefaultMutableTreeNode selectedNode = (DefaultMutableTreeNode) path
65.                 .getLastPathComponent();
66.             Class<?> c = (Class<?>) selectedNode.getUserObject();
67.             String description = getFieldDescription(c);
68.             textArea.setText(description);
69.         }
70.     });
71.     int mode = TreeSelectionModel.SINGLE_TREE_SELECTION;
72.     tree.getSelectionModel().setSelectionMode(mode);
73.
74.     // this text area holds the class description
75.     textArea = new JTextArea();
76.
77.     // add tree and text area
78.     JPanel panel = new JPanel();
79.     panel.setLayout(new GridLayout(1, 2));
80.     panel.add(new JScrollPane(tree));
81.     panel.add(new JScrollPane(textArea));
82.
83.     add(panel, BorderLayout.CENTER);
84.
85.     addTextField();
86. }
87.
88. /**
89. * Add the text field and "Add" button to add a new class.
90. */
91. public void addTextField()
92. {
93.     JPanel panel = new JPanel();
94.
95.     ActionListener addListener = new ActionListener()
96.     {
97.         public void actionPerformed(ActionEvent event)
98.         {
99.             // add the class whose name is in the text field
100.             try
101.             {
102.                 String text = textField.getText();
103.                 addClass(Class.forName(text)); // clear text field to indicate success
104.                 textField.setText("");
105.             }
106.             catch (ClassNotFoundException e)
107.             {
108.                 JOptionPane.showMessageDialog(null, "Class not found");
109.             }
110.         }
111.     };
112. }
```

```
109.         }
110.     }
111. }
112.
113. // new class names are typed into this text field
114. textField = new JTextField(20);
115. textField.addActionListener(addListener);
116. panel.add(textField);
117.
118. JButton addButton = new JButton("Add");
119. addButton.addActionListener(addListener);
120. panel.add(addButton);
121.
122. add(panel, BorderLayout.SOUTH);
123. }
124.
125. /**
126. * Finds an object in the tree.
127. * @param obj the object to find
128. * @return the node containing the object or null if the object is not present in the tree
129. */
130. @SuppressWarnings("unchecked")
131. public DefaultMutableTreeNode findUserObject(Object obj)
132. {
133.     // find the node containing a user object
134.     Enumeration<TreeNode> e = (Enumeration<TreeNode>) root.breadthFirstEnumeration();
135.     while (e.hasMoreElements())
136.     {
137.         DefaultMutableTreeNode node = (DefaultMutableTreeNode) e.nextElement();
138.         if (node.getUserObject().equals(obj)) return node;
139.     }
140.     return null;
141. }
142.
143. /**
144. * Adds a new class and any parent classes that aren't yet part of the tree
145. * @param c the class to add
146. * @return the newly added node.
147. */
148. public DefaultMutableTreeNode addClass(Class<?> c)
149. {
150.     // add a new class to the tree
151.
152.     // skip non-class types
153.     if (c.isInterface() || c.isPrimitive()) return null;
154.
155.     // if the class is already in the tree, return its node
156.     DefaultMutableTreeNode node = findUserObject(c);
157.     if (node != null) return node;
158.
159.     // class isn't present--first add class parent recursively
160.
161.     Class<?> s = c.getSuperclass();
162.
163.     DefaultMutableTreeNode parent;
164.     if (s == null) parent = root;
165.     else parent = addClass(s);
```

```
166.  
167.    // add the class as a child to the parent  
168.    DefaultMutableTreeNode newNode = new DefaultMutableTreeNode(c);  
169.    model.insertNodeInto(newNode, parent, parent.getChildCount());  
170.  
171.    // make node visible  
172.    TreePath path = new TreePath(model.getPathToRoot(newNode));  
173.    tree.makeVisible(path);  
174.  
175.    return newNode;  
176. }  
177.  
178. /**  
179. * Returns a description of the fields of a class.  
180. * @param the class to be described  
181. * @return a string containing all field types and names  
182. */  
183. public static String getFieldDescription(Class<?> c)  
184. {  
185.    // use reflection to find types and names of fields  
186.    StringBuilder r = new StringBuilder();  
187.    Field[] fields = c.getDeclaredFields();  
188.    for (int i = 0; i < fields.length; i++)  
189.    {  
190.        Field f = fields[i];  
191.        if ((f.getModifiers() & Modifier.STATIC) != 0) r.append("static ");  
192.        r.append(f.getType().getName());  
193.        r.append(" ");  
194.        r.append(f.getName());  
195.        r.append("\n");  
196.    }  
197.    return r.toString();  
198. }  
199.  
200. private DefaultMutableTreeNode root;  
201. private DefaultTreeModel model;  
202. private JTree tree;  
203. private JTextField textField;  
204. private JTextArea textArea;  
205. private static final int DEFAULT_WIDTH = 400;  
206. private static final int DEFAULT_HEIGHT = 300;  
207. }  
208.  
209. /**  
210. * This class renders a class name either in plain or italic. Abstract classes are italic.  
211. */  
212. class ClassNameTreeCellRenderer extends DefaultTreeCellRenderer  
213. {  
214.    public Component getTreeCellRendererComponent(JTree tree, Object value, boolean selected,  
215.                                              boolean expanded, boolean leaf, int row, boolean hasFocus)  
216.    {  
217.        super.getTreeCellRendererComponent(tree, value, selected, expanded, leaf,  
218.                                              row, hasFocus);  
219.        // get the user object  
220.        DefaultMutableTreeNode node = (DefaultMutableTreeNode) value;  
221.        Class<?> c = (Class<?>) node.getUserObject();  
222.    }
```

```

223.     // the first time, derive italic font from plain font
224.     if (plainFont == null)
225.     {
226.         plainFont = getFont();
227.         // the tree cell renderer is sometimes called with a label that has a null font
228.         if (plainFont != null) italicFont = plainFont.deriveFont(Font.ITALIC);
229.     }
230.
231.     // set font to italic if the class is abstract, plain otherwise
232.     if ((c.getModifiers() & Modifier.ABSTRACT) == 0) setFont(plainFont);
233.     else setFont(italicFont);
234.     return this;
235. }
236.
237. private Font plainFont = null;
238. private Font italicFont = null;
239. }

```

**API** **javax.swing.JTree 1.2**

- TreePath getSelectionPath()
- TreePath[] getSelectionPaths()

返回第一个选定的路径，或者一个包含所有选定节点的数组。如果没有选定任何路径，这两个方法都返回为null。

**API** **javax.swing.event.TreeSelectionListener 1.2**

- void valueChanged(TreeSelectionEvent event)

每当选定节点或撤销选定的时候，该方法就被调用。

**API** **javax.swing.event.TreeSelectionEvent 1.2**

- TreePath getPath()
- TreePath[] getPaths()

获取在该选择事件中已经发生更改的第一个路径或所有路径。如果你想知道当前的选择路径，而不是选择路径的更改情况，那么应该调用JTree.getSelectionPaths。

### 6.3.5 定制树模型

在最后一个示例中，我们实现了一个能够查看变量内容的程序，正如调试器所做的那样（参见图6-34）。

在继续深入之前，请先编译运行这个示例程序。其中每个节点对应于一个实例变量。如果该变量是一个对象，那么可以展开该节点以便查看它的实例变量。该程序可以观察框架窗口的内容。如果你浏

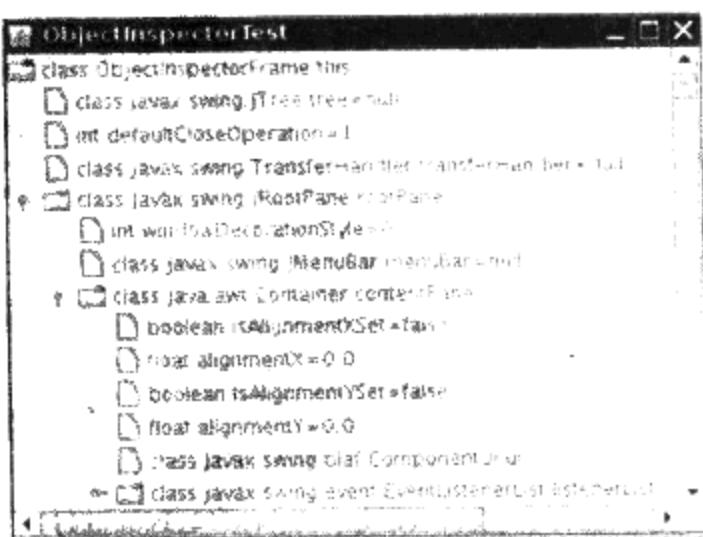


图6-34 一个对象查看树

览了好几个实例变量，那么你将会发现一些熟悉的类，还会对复杂的Swing用户界面构件有所了解。

该程序的不同之处在于它的树并没有使用DefaultTreeModel。如果你已经拥有按照层次结构组织的数据，那么你可能并不想去再创建一棵副本树，创建副本树还要担心怎样保持两棵树的一致性。这正是我们要讨论的情形：通过对象的引用，要观察的对象已经彼此连接起来了，因此在这里就不需要复制这种连接结构了。

TreeModel接口只有几个方法。第一组方法使得JTree能够按照先是根节点，后是子节点的顺序找到树中的节点。JTree类只在用户真正展开一个节点的时候才会调用这些方法。

```
Object getRoot()  
int getChildCount(Object parent)  
Object getChild(Object parent, int index)
```

这个示例显示了为什么TreeModel接口像JTree类那样，不需要很清楚地了解节点。根节点和子节点可以是任何对象，TreeModel负责告知JTree它们是怎样联系起来的。

TreeModel接口的下一个方法与getChild相反：

```
int getIndexOfChild(Object parent, Object child)
```

实际上，这个方法可以用前面的三个方法实现，参见程序清单6-11中的代码。

树模型会告诉JTree哪些节点应该显示成叶节点：

```
boolean isLeaf(Object node)
```

如果你的代码更改了树模型，那么必须告知这棵树以便它能够对自己进行重新绘制。树是将它自己作为一个TreeModelListener添加到模型中的，因此，模型必须支持通常的监听器管理方法：

```
void addTreeModelListener(TreeModelListener l)  
void removeTreeModelListener(TreeModelListener l)
```

可以在程序清单6-11中看到这些方法的具体实现。

当模型修改了树的内容时，它会调用TreeModelListener接口中下面4个方法中的某一个：

```
void treeNodesChanged(TreeModelEvent e)  
void treeNodesInserted(TreeModelEvent e)  
void treeNodesRemoved(TreeModelEvent e)  
void treeStructureChanged(TreeModelEvent e)
```

TreeModelEvent对象用于描述修改的位置。对描述插入或移除事件的树模型事件进行组装的细节是相当技术性的。如果树中确实有要添加或移除的节点，只需要考虑如何触发这些事件。在程序清单6-11中，我们展示了怎么触发一个事件：将根节点替换为一个新的对象。

**提示：**为了简化事件触发的代码，我们使用了javax.swing.EventListenerList这个使用方便、能够收集监听器的类。请参阅卷Ⅰ第8章查看这个类的更多细节信息。

最后，如果用户要编辑树节点，那么模型会随着这种修改而调用：

```
void valueForPathChanged(TreePath path, Object newValue)
```

如果不允许编辑，则永远不会调用到该方法。

如果不支持编辑功能，那么构建一个树模型就变得相当容易了。我们要实现下面3个方法：

```
Object getRoot()  
int getChildCount(Object parent)  
Object getChild(Object parent, int index)
```

这3个方法用于描述树的结构。还要提供另外5个方法的常规实现，如程序清单6-11那样，然后就可以准备显示你的树了。

现在让我们转向示例程序的具体实现，我们的树将包含类型为Variable的对象。

**注意：**一旦使用了DefaultTreeModel，我们的节点就可以具有类型为DefaultMutableTreeNode、用户对象类型为Variable的对象。

例如，假设我们查看下面这个变量

```
Employee joe;
```

该变量的类型为Employee.class，名字为joe，值为对象引用joe的值。我们定义了Variable这个类，用来描述程序中的变量：

```
Variable v = new Variable(Employee.class, "joe", joe);
```

如果该变量的类型为基本类型，必须为这个值使用对象包装器。

```
new Variable(double.class, "salary", new Double(salary));
```

如果变量的类型是一个类，那么该变量就会拥有一些域。使用反射机制可以将所有域枚举出来，并将它们收集存放到一个ArrayList中。因为Class类的getFields方法不返回超类的任何域，因此还必须调用超类中的getFields方法。读者可以在Variable构造器中找到这些代码。Variable类的getFields方法将返回包含域的一个数组。最后，Variable类的toString方法将节点格式化为标签。这个标签通常包含变量的类型和名称。如果变量不是一个类，那么该标签还将包含变量的值。

**注意：**如果类型是一个数组，那么我们不会显示数组中的元素。这并不难实现；因此我们就把它留作众所周知的“读者练习”。

让我们继续介绍树模型，最初的两个方法很简单。

```
public Object getRoot()  
{  
    return root;  
}  
  
public int getChildCount(Object parent)  
{  
    return ((Variable) parent).getFields().size();  
}
```

getChild方法返回一个新的Variable对象，用于描述给定索引位置上的域。Field类的getType方法和getName方法用于产生域的类型和名称。通过使用反射机制，你可以按照f.getValue(parentValue)这种方式读取域的值。该方法可以抛出一个异常IllegalAccessException。不过，我们可以让所有域在Variable构造器中都是可访问的。这样，在实际应用中，就不会发生这种情况。

下面是getChild方法的完整代码。

```

public Object getChild(Object parent, int index)
{
    ArrayList fields = ((Variable) parent).getFields();
    Field f = (Field) fields.get(index);
    Object parentValue = ((Variable) parent).getValue();
    try
    {
        return new Variable(f.getType(), f.getName(), f.get(parentValue));
    }
    catch (IllegalAccessException e)
    {
        return null;
    }
}

```

这3个方法展示了对象树到JTree构件之间的结构，其余的方法是一些常规方法，源代码请见程序清单6-11。

关于该树模型，有一个不同寻常之处：它实际上描述的是一棵无限树。可以通过追踪WeakReference对象来证实这一点。当你点击名字为referent的变量时，它会引导你回到初始的对象。你将获得一棵相同的子树，并且可以再次展开它的WeakReference对象，周而复始，无穷无尽。当然，你无法存储一个无限的节点集合。树模型只是在用户展开父节点时，按照需要来产生这些节点。

我们用这个示例来结束对树的讨论。我们将继续下一个复杂的Swing构件——表格构件。从表面上看，树和表格似乎没有什么相同之处，但是你会发现对于数据模型和单元格绘制，二者采用了相同的概念。

### 程序清单6-11 ObjectInspectorTest.java

```

1. import java.awt.*;
2. import java.lang.reflect.*;
3. import java.util.*;
4. import javax.swing.*;
5. import javax.swing.event.*;
6. import javax.swing.tree.*;
7.
8. /**
9. * This program demonstrates how to use a custom tree model. It displays the fields of
10. * an object.
11. * @version 1.03 2007-08-01
12. * @author Cay Horstmann
13. */
14. public class ObjectInspectorTest
15. {
16.     public static void main(String[] args)
17.     {
18.         EventQueue.invokeLater(new Runnable()
19.         {
20.             public void run()
21.             {
22.                 JFrame frame = new ObjectInspectorFrame();
23.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24.                 frame.setVisible(true);

```

```
25.         }
26.     });
27. }
28. }
29. /**
30. * This frame holds the object tree.
31. */
32. class ObjectInspectorFrame extends JFrame
33. {
34.     public ObjectInspectorFrame()
35.     {
36.         setTitle("ObjectInspectorTest");
37.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
38.
39.         // we inspect this frame object
40.
41.         Variable v = new Variable(getClass(), "this", this);
42.         ObjectTreeModel model = new ObjectTreeModel();
43.         model.setRoot(v);
44.
45.         // construct and show tree
46.
47.         tree = new JTree(model);
48.         add(new JScrollPane(tree), BorderLayout.CENTER);
49.     }
50. }
51.
52. private JTree tree;
53. private static final int DEFAULT_WIDTH = 400;
54. private static final int DEFAULT_HEIGHT = 300;
55. }
56.
57. /**
58. * This tree model describes the tree structure of a Java object. Children are the objects
59. * that are stored in instance variables.
60. */
61. class ObjectTreeModel implements TreeModel
62. {
63.     /**
64.      * Constructs an empty tree.
65.      */
66.     public ObjectTreeModel()
67.     {
68.         root = null;
69.     }
70.
71.     /**
72.      * Sets the root to a given variable.
73.      * @param v the variable that is being described by this tree
74.      */
75.     public void setRoot(Variable v)
76.     {
77.         Variable oldRoot = v;
78.         root = v;
79.         fireTreeStructureChanged(oldRoot);
80.     }
81.
```

```
82.     public Object getRoot()
83.     {
84.         return root;
85.     }
86.
87.     public int getChildCount(Object parent)
88.     {
89.         return ((Variable) parent).getFields().size();
90.     }
91.
92.     public Object getChild(Object parent, int index)
93.     {
94.         ArrayList<Field> fields = ((Variable) parent).getFields();
95.         Field f = (Field) fields.get(index);
96.         Object parentValue = ((Variable) parent).getValue();
97.         try
98.         {
99.             return new Variable(f.getType(), f.getName(), f.get(parentValue));
100.        }
101.       catch (IllegalAccessException e)
102.       {
103.           return null;
104.       }
105.     }
106.
107.    public int getIndexOfChild(Object parent, Object child)
108.    {
109.        int n = getChildCount(parent);
110.        for (int i = 0; i < n; i++)
111.            if (getChild(parent, i).equals(child)) return i;
112.        return -1;
113.    }
114.
115.    public boolean isLeaf(Object node)
116.    {
117.        return getChildCount(node) == 0;
118.    }
119.
120.    public void valueForPathChanged(TreePath path, Object newValue)
121.    {
122.    }
123.
124.    public void addTreeModelListener(TreeModelListener l)
125.    {
126.        listenerList.add(TreeModelListener.class, l);
127.    }
128.
129.    public void removeTreeModelListener(TreeModelListener l)
130.    {
131.        listenerList.remove(TreeModelListener.class, l);
132.    }
133.
134.    protected void fireTreeStructureChanged(Object oldRoot)
135.    {
136.        TreeModelEvent event = new TreeModelEvent(this, new Object[] { oldRoot });
137.        EventListener[] listeners = listenerList.getListeners(TreeModelListener.class);
138.        for (int i = 0; i < listeners.length; i++)
```

```
139.         ((TreeModelListener) listeners[i]).treeStructureChanged(event);
140.     }
141.
142.     private Variable root;
143.     private EventListenerList listenerList = new EventListenerList();
144. }
145.
146. /**
147. * A variable with a type, name, and value.
148. */
149. class Variable
150. {
151.     /**
152.      * Construct a variable
153.      * @param aType the type
154.      * @param aName the name
155.      * @param aValue the value
156.      */
157.     public Variable(Class<?> aType, String aName, Object aValue)
158.     {
159.         type = aType;
160.         name = aName;
161.         value = aValue;
162.         fields = new ArrayList<Field>();
163.
164.         // find all fields if we have a class type except we don't expand strings and null values
165.
166.         if (!type.isPrimitive() && !type.isArray() && !type.equals(String.class) && value != null)
167.         {
168.             // get fields from the class and all superclasses
169.             for (Class<?> c = value.getClass(); c != null; c = c.getSuperclass())
170.             {
171.                 Field[] fs = c.getDeclaredFields();
172.                 AccessibleObject.setAccessible(fs, true);
173.
174.                 // get all nonstatic fields
175.                 for (Field f : fs)
176.                     if ((f.getModifiers() & Modifier.STATIC) == 0) fields.add(f);
177.             }
178.         }
179.     }
180.
181. /**
182. * Gets the value of this variable.
183. * @return the value
184. */
185. public Object getValue()
186. {
187.     return value;
188. }
189.
190. /**
191. * Gets all nonstatic fields of this variable.
192. * @return an array list of variables describing the fields
193. */
194. public ArrayList<Field> getFields()
195. {
```

```

196     return fields;
197 }
198
199 public String toString()
200 {
201     String r = type + " " + name;
202     if (type.isPrimitive()) r += "=" + value;
203     else if (type.equals(String.class)) r += "=" + value;
204     else if (value == null) r += "=null";
205     return r;
206 }
207
208 private Class<?> type;
209 private String name;
210 private Object value;
211 private ArrayList<Field> fields;
212 }

```

**API** **javax.swing.tree.TreeModel 1.2**

- **Object getRoot()**  
返回根节点。
- **int getChildCount(Object parent)**  
获取parent节点的子节点个数。
- **Object getChild(Object parent, int index)**  
获取给定索引位置上parent节点的子节点。
- **int getIndexOfChild(Object parent, Object child)**  
获取parent节点的子节点child的索引位置。如果在树模型中child节点不是parent的一个子节点，则返回-1。
- **boolean isLeaf(Object node)**  
如果节点node从概念上讲是一个叶节点，则返回true。
- **void addTreeModelListener(TreeModelListener l)**
- **void removeTreeModelListener(TreeModelListener l)**  
当模型中的信息发生变化时，告知添加和移除监听器。
- **void valueForPathChanged(TreePath path, Object newValue)**  
当一个单元格编辑器修改了节点值的时候，该方法被调用。  
参数：path 到被编辑节点的树路径  
newValue 编辑器返回的修改值

**API** **javax.swing.event.TreeModelListener 1.2**

- **void treeNodesChanged(TreeModelEvent e)**
- **void treeNodesInserted(TreeModelEvent e)**
- **void treeNodesRemoved(TreeModelEvent e)**
- **void treeStructureChanged(TreeModelEvent e)**

如果树被修改过，树模型将调用该方法。

### **API** javax.swing.event.TreeModelEvent 1.2

- TreeModelEvent(Object eventSource, TreePath node)

构建一个树模型事件。

参数: eventSource 产生该事件的树模型

node 到达要修改节点的树路径

## 6.4 文本构件

图6-35展示了Swing类库中包含的所有文本构件，在第I卷第9章你已经看到过其中3个最常用的构件： JTextField、JPasswordField和JTextArea。在下面各节中，我们将介绍其余的文本构件。我们还将讨论JSpinner构件，它包含一个格式化的文本框，以及用来改变其内容的“up (上)”和“down (下)”小按钮。

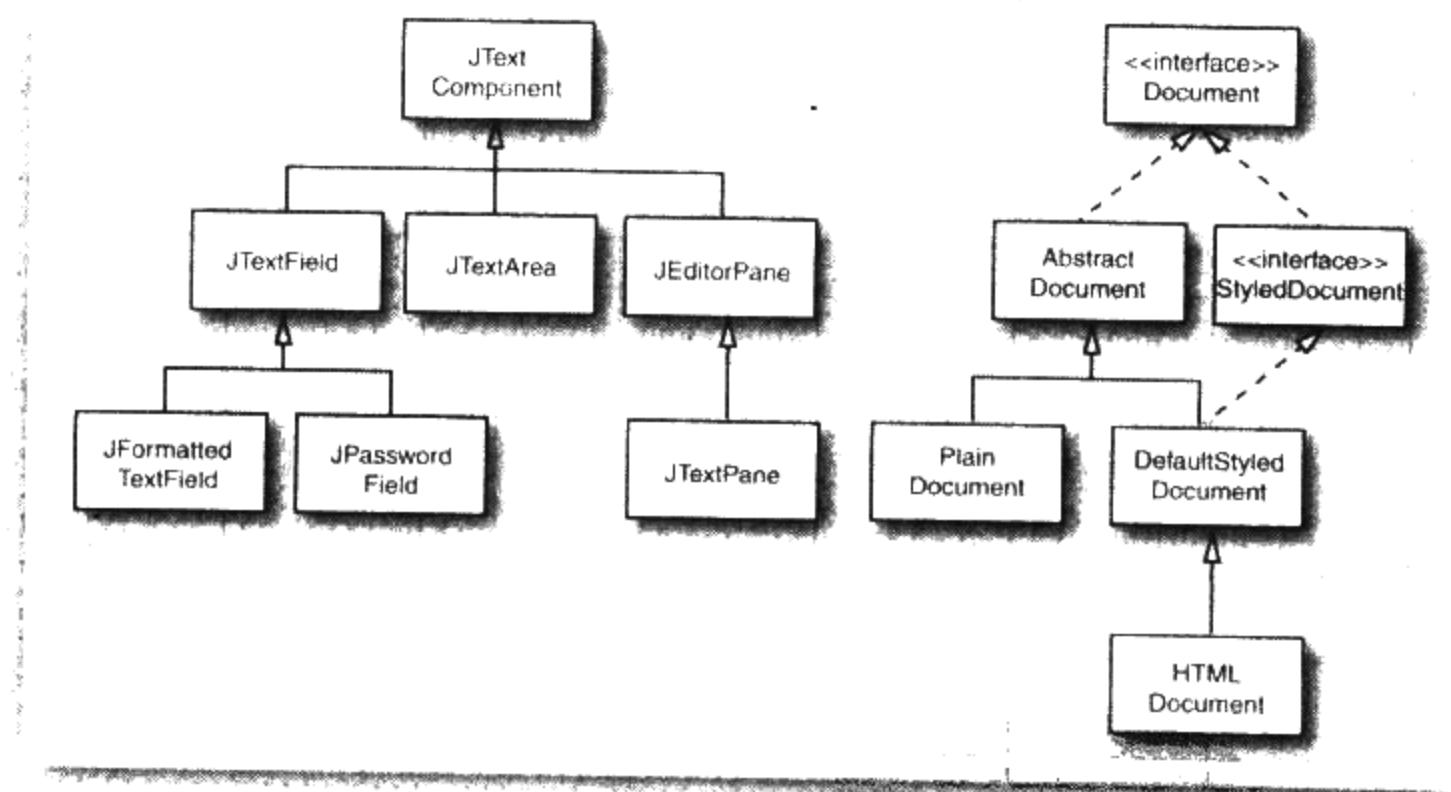


图6-35 文本构件和文档的层次结构

所有文本构件都可以绘制和编辑存储在实现了Document接口的类的模型对象中的数据。JTextField和JTextArea构件使用的是PlainDocument，该构件只存储普通文本的行序列，而不进行任何格式化。

JEditorPane可以展示和编辑各种格式的样式文本（包括字体、颜色等），特别是HTML，参见第6.4.4节，StyledDocument接口描述了对样式、字体和颜色的额外需求，而HTMLDocument类实现了这个接口。

JEditorPane的子类JTextPane可以持有样式的文本和嵌入的Swing构件。我们在本书中将不讨论过于复杂的JTextPane，但是推荐你参考Kim Toley所著的《Core Swing: Advanced

Programming》一书以了解其中关于此构件十分详细的描述。对于JTextPane类的典型用法，可以查看JDK中的StylePad演示程序。

#### 6.4.1 文本构件中的修改跟踪

只有当你希望实现自己的文本编辑器时，你才需要面对Document接口的复杂性。然而，这个接口的最常见的用法是：跟踪修改。

有时，你希望只要用户进行了文本编辑，无需等待他点击某个按钮，就马上更新部分用户界面。下面是一个简单的示例：我们显示了三个文本框，用于编辑颜色的红、蓝、绿色调。只要这些文本框的内容发生了变化，颜色就应该立即更新。图6-36展示了程序清单6-12中的程序运行起来的样子。



图6-36 跟踪文本框中的修改

首先请注意，监视键盘点击事件并非好主意，因为有些键盘点击事件并不修改文本（例如，点击方向键）。更重要的是，文本可以因鼠标的姿态变化而改变（例如在X11中的“鼠标中键粘贴”）。因此，应该让文档（document）来通知我们数据发生了变化，方法是在文档（而不是文本构件）上安装文档监听器（document listener）：

```
textField.getDocument().addDocumentListener(listener);
```

当文本发生变化时，会调用下列DocumentListener方法之一：

```
void insertUpdate(DocumentEvent event)
void removeUpdate(DocumentEvent event)
void changedUpdate(DocumentEvent event)
```

前两个方法是在插入或移除字符时被调用的，第三个方法对于文本框来说根本不会被调用，而对于更复杂的文档类型，在产生某些其他类型的变化，例如格式上的变化时，这个方法才会被调用。但是，由于没有任何单个的回调可以告诉我们文本发生了变化（通常我们也并不太关心文本发生了怎样的变化），同时也没有任何适配器类。因此，文档监听器必须实现所有这3个方法。下面是我们在示例程序中的做法：

```
DocumentListener listener = new DocumentListener()
{
    public void insertUpdate(DocumentEvent event) { setColor(); }
    public void removeUpdate(DocumentEvent event) { setColor(); }
    public void changedUpdate(DocumentEvent event) {}
}
```

setColor方法使用getText方法从文本框中获得当前的用户输入字符串，并设置其颜色。

我们的程序有一个限制：用户可以在文本框中键入非数字的畸形输入，例如“twenty”，或者使文本框保持为空。因此，我们将捕获parseInt方法抛出的NumberFormatException，并且在文本框中的内容不是数字时，不执行更新颜色的操作。在下一节，你将会看到可以如何预先防止用户键入无效的输入。

**注意：**除了监听文档事件，还可以在文本框上添加一个行为事件监听器。只要用户按下了回车键，动作监听器就会得到通知。我们不推荐这种方法，因为用户在完成数据输入后，并非总是记得按回车键。如果使用动作监听器，就应该同时安装一个焦点监听器，

这样我们可以跟踪用户何时离开该文本框。

### 程序清单6-12 ChangeTrackingTest.java

```
1. import java.awt.*;
2. import javax.swing.*;
3. import javax.swing.event.*;
4.
5. /**
6. * @version 1.40 2007-08-05
7. * @author Cay Horstmann
8. */
9. public class ChangeTrackingTest
10. {
11.     public static void main(String[] args)
12.     {
13.         EventQueue.invokeLater(new Runnable()
14.         {
15.             public void run()
16.             {
17.                 ColorFrame frame = new ColorFrame();
18.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19.                 frame.setVisible(true);
20.             }
21.         });
22.     }
23. }
24.
25. /**
26. * A frame with three text fields to set the background color.
27. */
28. class ColorFrame extends JFrame
29. {
30.     public ColorFrame()
31.     {
32.         setTitle("ChangeTrackingTest");
33.
34.         DocumentListener listener = new DocumentListener()
35.         {
36.             public void insertUpdate(DocumentEvent event)
37.             {
38.                 setColor();
39.             }
40.
41.             public void removeUpdate(DocumentEvent event)
42.             {
43.                 setColor();
44.             }
45.
46.             public void changedUpdate(DocumentEvent event)
47.             {
48.             }
49.         };
50.
51.         panel = new JPanel();
52.     }
```

```

53     panel.add(new JLabel("Red:"));
54     redField = new JTextField("255", 3);
55     panel.add(redField);
56     redField.getDocument().addDocumentListener(listener);
57
58     panel.add(new JLabel("Green:"));
59     greenField = new JTextField("255", 3);
60     panel.add(greenField);
61     greenField.getDocument().addDocumentListener(listener);
62
63     panel.add(new JLabel("Blue:"));
64     blueField = new JTextField("255", 3);
65     panel.add(blueField);
66     blueField.getDocument().addDocumentListener(listener);
67
68     add(panel);
69     pack();
70 }
71
72 /**
73 * Set the background color to the values stored in the text fields.
74 */
75 public void setColor()
76 {
77     try
78     {
79         int red = Integer.parseInt(redField.getText().trim());
80         int green = Integer.parseInt(greenField.getText().trim());
81         int blue = Integer.parseInt(blueField.getText().trim());
82         panel.setBackground(new Color(red, green, blue));
83     }
84     catch (NumberFormatException e)
85     {
86         // don't set the color if the input can't be parsed
87     }
88 }
89
90 private JPanel panel;
91 private JTextField redField;
92 private JTextField greenField;
93 private JTextField blueField;
94 }
```

**API javax.swing.JComponent 1.2**

- Dimension getPreferredSize()
- void setPreferredSize(Dimension d)

获取和设置该构件的偏好尺寸。

**API javax.swing.text.Document 1.2**

- int getLength()
- 返回文档中当前的字符数量。
- String getText(int offset, int length)

返回在文档的给定部分中所包含的文本。

参数: offset 文本的起始位置

length 希望得到的字符串的长度

- void addDocumentListener(DocumentListener listener)

注册监听器,使得在文档发生变化时,可以得到通知。



#### javax.swing.event.DocumentEvent 1.2

- Document getDocument()

获取事件来源的文档。



#### javax.swing.event.DocumentListener 1.2

- void changedUpdate(DocumentEvent event)

当某个属性或属性集发生变化时,该方法即被调用。

- void insertUpdate(DocumentEvent event)

在文档中插入时,该方法即被调用。

- void removeUpdate(DocumentEvent event)

在文档中有部分内容被移除时,该方法即被调用。

### 6.4.2 格式化的输入框

在最后一个示例程序中,我们希望程序的用户键入数字而不是任意的字符串。也就是说,只允许用户键入数字0到9以及连字符,并且如果有连字符,它必须是输入字符串的第一个字符。

表面上看,这种输入检验任务很简单。我们可以在文本框上安装一个按键监听器,然后处理掉所有不是数字和连字符的按键事件。但是,这种简单的方法在实践中并非很有效,尽管这是通常被推荐的输入检验方法。首先,并非每一种有效输入字符的组合都是一个有效的数字,例如,--3和3-3都无效,尽管它们是由有效的输入字符构成的。但是,更重要的是,有些对文本修改的方式并不涉及键入字符键。根据不同的用户界面感观,某些组合键可以用来剪切、复制和粘贴文本。例如,在金属用户界面感观中,CTRL+V组合键可以将粘贴缓冲区中的内容粘贴到文本框中。也就是说,我们还需要监视用户是否粘贴了无效的字符。很明显,试图通过过滤键盘点击来确保文本框的内容总是有效这种方法看起来已经很麻烦了,而这些任务并不应该让应用系统的程序员去关注。

有点令人惊讶的是,在Java SE 1.4之前,没有任何构件用于输入数字型的值。从《Core Java》的第1版开始,我们就提供了一个IntTextField实现,这是一个用于输入正确格式的整数的文本框。在此后的每个新版本中,我们都在修改这个实现,以利用Java在其每个新版本中不断添加的各种不太全面的校验模式。最终,在Java SE 1.4中,Swing的设计者们正视了这个问题,并且提供了通用的JFormattedTextField类,它不仅可以用于数字型的输入,而且可以用于日期型输入以及更加专用的格式化输入值,例如IP地址。

#### 整数输入

让我们从简单的情况入手:用于整数输入的文本框

```
JFormattedTextField intField = new JFormattedTextField(NumberFormat.getIntegerInstance());
```

`NumberFormat.getIntegerInstance`将使用当前的`locale`返回一个用于格式化整数的格式器对象。在美国`locale`中，逗号用十进制分隔符，从而允许用户输入像1 729这样的值。第5章详细解释了如何选择其他的`locale`。

对于任何文本框，都可以设置其位数：

```
intField.setColumns(6);
```

还可以用`setValue`方法设置其默认值，该方法接受一个`Object`类型的参数，因此我们需要将默认的`int`值包装到一个`Integer`对象中：

```
intField.setValue(new Integer(100));
```

通常，用户会在多个文本框中输入，然后点击某个按钮来读取所有这些值。当按钮被点击后，可以用`getValue`方法来获取用户提供的值，这个方法返回的是一个`Object`类型的结果，必须将它转型为恰当的类型。如果用户进行了编辑，那么`JFormattedTextField`将返回`Long`类型的对象。但是，如果用户没有进行修改，就会返回最初的`Integer`对象。因此，应该将返回值转型为它们的共同超类`Number`：

```
Number value = (Number) intField.getValue();
int v = value.intValue();
```

格式化文本框看上去可能并没什么太大的用处，但是如果你要考虑用户提供非法输入时的情况，那么它就有用处了，这正是下一节的主题。

#### 失去焦点时的行为

考虑一下当用户向文本框中输入时会发生什么。用户键入输入，并且在完成后决定离开这个文本框，因此可能会用鼠标点击其他的构件，然后这个文本框将失去焦点（lose focus），在其中不再会看到I闪烁光标，这样键盘点击都将被导向另一个不同的构件。

当格式化文本框失去焦点时，格式器会查看用户输入的文本字符串。如果格式器知道如何将这个文本字符串转换为对象，那么这个文本就是有效的，否则就是无效的。可以使用`isEditValid`方法来检查文本框的当前内容是否有效。

失去焦点的默认行为称为“提交或回复”。如果文本字符串有效，则它被提交（commit），之后格式器将其转换为对象，而该对象将成为文本框的当前值（也就是前一节中提到的`getValue`方法的返回值）。这个值然后再被转换回字符串，成为在文本框中看到的字符串。例如，整数格式器将输入的1729识别为有效，将当前值设置为`new Long(1729)`，然后将其转换回带有十进制逗号的字符串1,729。

反之，如果文本字符串无效，则当前值不发生变化，而文本框将回复到表示原有值的字符串。例如，如果用户输入了无效值，例如x1，那么当文本框失去焦点时，将恢复原有值。

 **注意：**整数格式器将以整数开头的文本字符串当作是有效的。例如，1729x是有效的字符串，它将被转换为数字1729，这个数字之后会被格式化为字符串1,729。

可以用`setFocusLostBehavior`方法来设置其他的行为。“提交”行为与默认行为有些细微的差异，如果文本字符串无效，那么文本字符串和文本框的值都将保持不变，现在它们是不同

步的。“持久化”行为更加保守，即使文本字符串是有效的，文本框和当前值都不发生变化，这时需要调用commitEdit、setValue和setText来使它们同步。最后，还有一个“回复”行为，它看起来永远都没什么用，其行为是只要失去了焦点，用户输入就会被丢弃，而文本字符串将恢复到原有值。

 **注意：**通常，“提交或回复”作为默认行为是合理的，这么做只有一个潜在可能发生的问题。假设对话框中包含用于整数值的文本框，而用户输入了字符串“1729”，其中带一个先导的空格，然后点击了OK按钮。这个先导的空格将会使数字无效，而这个文本框的值也将恢复到原有值。接着，OK按钮的动作监听器获取文本框的值，然后关闭对话框。这样用户永远都不会知道他输入的新值被拒绝了。在这种情况下，恰当的选择应该是“提交”行为，然后让OK按钮的监听器在关闭对话框之前检查所有的文本框编辑是否都有效。

### 过滤器

格式化文本框的基本功能对于大多数用户来说很直观，而且也足够用了。但是，我们还可以添加一些精化的功能，例如同时还要防止用户键入非数字字符，我们可以用文档过滤器 (document filter) 来实现这个行为。回忆一下，在模型-视图-控制器架构中，控制器将输入事件转译成了修改文本框底层文档的命令，这个底层文档也就是存储在PlainDocument对象中的文本字符串。例如，每当控制器处理的命令会导致在该文档中插入字符串时，它就会调用“插入字符串”命令。要插入的字符串可以是单个的字符，也可以是粘贴缓冲区中的内容。文档过滤器可以拦截这个命令，并修改字符串或放弃插入操作。下面是过滤器的insertString方法的代码，该方法对要插入的字符串进行分析，并只插入那些数字和负号（-）字符。（这段代码可以处理第3章中描述的补充Unicode字符，请参加第12章StringBuilder类。）

```
public void insertString(FilterBypass fb, int offset, String string, AttributeSet attr)
    throws BadLocationException
{
    StringBuilder builder = new StringBuilder(string);
    for (int i = builder.length() - 1; i >= 0; i--)
    {
        int cp = builder.codePointAt(i);
        if (!Character.isDigit(cp) && cp != '-')
        {
            builder.deleteCharAt(i);
            if (Character.isSupplementaryCodePoint(cp))
            {
                i--;
                builder.deleteCharAt(i);
            }
        }
    }
    super.insertString(fb, offset, builder.toString(), attr);
}
```

还应该覆盖DocumentFilter类的replace方法，该方法在文本被选中并被替换时调用。replace方法的实现很直观，参见程序清单6-13。

现在需要安装文档过滤器。但是，没有很直观的方法可以实现这个任务，必须覆盖某个格

式器类的getDocumentFilter方法，然后将这个格式器的一个对象传递给JFormattedTextField。整数文本框使用的是用NumberFormat.getIntegerInstance()初始化的InternationalFormatter。下面展示了如何安装格式器以产生所需的过滤器：

```
JFormattedTextField intField = new JFormattedTextField(new
    InternationalFormatter(NumberFormat.getIntegerInstance()))
{
    protected DocumentFilter getDocumentFilter()
    {
        return filter;
    }
    private DocumentFilter filter = new IntFilter();
});
```

 **注意：**Java SE文档声明DocumentFilter类被设计为禁止子类化。直到Java SE 1.3，文本框中的过滤机制才通过扩展PlainDocument类和覆盖insertString与replace方法得到了实现。现在，PlainDocument类有了可插拔的过滤器，这是一项极佳的改进。如果过滤器在格式器类中也是可插拔的，那么这项改进就更好了。唉，但是它不是，我们必须子类化格式器。

试验一下本节最后的FormatTest示例程序，其中第三个文本框就安装了一个过滤器，这样就只能插入数字和负号字符了。注意，现在你仍旧可以键入诸如“1-2-3”这样的无效字符串。通常，通过过滤机制来避免所有无效字符串是不可能的。例如，字符串“-”是无效的，但是过滤器不能拒绝它，因为它是合法字符串“-1”的前缀。即使过滤器不能进行完美的保护，但是使用它们来拒绝明显无效的输入仍旧是有意义的。

 **提示：**过滤机制的另一种用法是将一个字符串的所有字符都转为大写。这样的过滤器很容易编写，在其insertString和replace方法中，将要插入的字符串转换成大写，然后调用超类的方法即可。

### 校验器

还有一种很有用的机制，可以就无效输入对用户发出警告，这就是在任意的JComponent上附着一个校验器（verifier）。如果该构件失去了焦点，那么校验器就会被查询。如果校验器报告该构件无效，那么该构件就会立即重新获得焦点。这样，用户就被强制要求在进行其他输入之前先订正刚输入的内容。

校验器必须扩展InputVerifier类并定义verify方法，而定义检查格式化文本框的校验器非常容易。JFormattedTextField类的isEditValid方法将调用格式器，并且在格式器可以将文本字符串转换为对象时返回true。下面是一个校验器：

```
class FormattedTextFieldVerifier extends InputVerifier
{
    public boolean verify(JComponent component)
    {
        JFormattedTextField field = (JFormattedTextField) component;
        return field.isEditValid();
    }
}
```

我们可以将它附着到任何JFormattedTextField上：

```
intField.setInputVerifier(new FormattedTextFieldVerifier());
```

但是，校验器并非总是很安全。如果点击了某个按钮，而这个按钮在无效构件再次获得焦点之前通知了它的动作监听器，那么这个动作监听器就会从未通过校验的构件中得到一个无效的结果。这种行为的原因在于：用户可能希望点击Cancel按钮，而无需订正无效输入。

在示例程序中的第四个文本框就附着了一个校验器。试着在其中键入无效数字（例如x1729），然后按下TAB键，或者用鼠标点击其他文本框。注意，该文本框会立即重新得到焦点。但是，如果你点击OK按钮，动作监听器就会调用getValue，它会报告最后一个有效值。

#### 其他的标准格式器

除了整数格式器，JFormattedTextField还支持若干种其他的格式器。NumberFormat类有下列静态方法：

```
getNumberInstance  
getCurrencyInstance  
getPercentInstance
```

它们将分别产生用于浮点数字、货币值和百分比的格式器。例如，通过下面的调用可以获得用于输入货币值的文本框。

```
JFormattedTextField currencyField = new JFormattedTextField(NumberFormat.getCurrencyInstance());
```

要编辑日期和时间，可以调用DateFormat类的下列静态方法之一：

```
getDateInstance  
getTimeInstance  
getDateTimeInstance
```

例如：

```
JFormattedTextField dateField = new JFormattedTextField(DateFormat.getDateInstance());
```

所产生的文本框将用默认格式或下面的“中等长度”格式来编辑日期：

Aug 5, 2007

也可以选择使用“短”格式

8/5/07

方法是调用下面的语句：

```
DateFormat.getDateInstance(DateFormat.SHORT)
```



注意：默认情况下，日期格式器是很“宽容”的，也就是说，像2002年2月31号这样的无效日期将会滚动到下一个有效日期2002年3月3日。这种行为可能会让用户觉得意外，此时，可以在DateFormat对象上调用setLenient(false)。

对于任何类，只要它有一个接受字符串参数的构造器，以及相匹配的toString方法，那么DefaultFormatter就可以格式化它的对象。例如，URL类有一个URL(String)构造器，可以从字符串中构建URL，例如：

```
URL url = new URL("http://java.sun.com");
```

因此，我们可以用DefaultFormatter格式化URL对象。格式器会在文本框值上调用toString

方法以初始化该文本框的文本。当文本框失去焦点时，格式器将使用带有String参数的构造器来构建与当前值属于相同类的新对象。如果这个构造器抛出了异常，那么这次编辑就是无效的。你可以运行示例程序，键入并非以“`http:`”这种前缀开头的URL，然后观察其响应。

**注意：**默认情况下，`DefaultFormatter`是覆盖模式，这与其他格式器很不相同，并且不是非常有用。调用`setOverwriteMode(false)`可以关闭覆盖模式。

最后，`MaskFormatter`对于包含部分常量和部分变量字符的固定尺寸的模式是非常有用的。例如，社会保障号（例如，078-05-1120）可以用下面的格式器进行格式化：

```
new MaskFormatter("###-##-###")
```

其中#符号表示单个数字，表6-3展示了可以在掩码格式器中使用的各种符号。

我们可以通过调用`MaskFormatter`类的下列方法之一来限制可以键入到文本框中的字符：

```
setValidCharacters  
setInvalidCharacters
```

例如，要读入用字母表示的成绩（例如A+或F），可以执行下面的语句：

```
MaskFormatter formatter = new MaskFormatter("U±");  
formatter.setValidCharacters("ABCDF+- ");
```

但是，没有办法可以指定第二个字符不能是字母。

请注意，由掩码格式器格式化的字符串与掩码有严格相同的长度。如果用户在编辑时删除了某些字符，那么它们就会被占位符所替换。默认的占位符是空格，但是可以用`setPlaceholderCharacter`方法来改变它，例如：

```
formatter.setPlaceholderCharacter('0');
```

默认情况下，掩码格式器处于覆盖模式，这很直观，所以运行示例程序来观察它。同时还要注意脱字符的位置会跳过掩码中的固定字符。

掩码格式器对于像社会保障号或美国电话号码这样的严格模式来说显得非常有效。但是，请注意，掩码模式中不允许有任何变体。例如，不能将掩码格式器用于国际电话号码，因为它们的位数并不固定。

### 定制格式器

如果所有的标准格式器都不适用，那么我们可以很方便地定义自己的格式器。请考虑4字节的IP地址，例如：

130.65.86.66

我们不能使用`MaskFormatter`，因为每个字节都可以由1个、2个或3个数字表示。而且，我们希望格式器能够检查每个字节的值最大不能超过255。

要定义自己的格式器，需要扩展`DefaultFormatter`类，并覆盖下面的方法：

```
String valueToString(Object value)  
Object stringToValue(String text)
```

表6-3 MaskFormatter符号

| 符 号 | 解 释                 |
|-----|---------------------|
| #   | 一个数字                |
| ?   | 一个字母                |
| U   | 一个字母，转换为大写          |
| L   | 一个字母，转换为小写          |
| A   | 一个字母或数字             |
| H   | 一个十六进制数字[0-9A-Fa-f] |
| *   | 任何字符                |
| '   | 在模式中包含的转义字符         |

第一个方法将文本框的值转换为显示在其中的字符串；第二个方法解析用户键入的文本，并将其转换回对象。这两个方法只要发现了错误，就应该抛出ParseException。

在示例程序中，我们用长度为4的byte[]数组存储IP地址。valueToString方法将构建由这些字节构成的字符串，其中字节与字节之间由句点隔开。注意，byte值是有符号的，取值范围位于-128与127之间（例如，在IP地址130.65.86.66中，第一个八位实际上是表示-126的字节）。要想将负的字节值转换为无符号的整数值，需要加上256。

```
public String valueToString(Object value) throws ParseException
{
    if (!(value instanceof byte[]))
        throw new ParseException("Not a byte[]", 0);
    byte[] a = (byte[]) value;
    if (a.length != 4)
        throw new ParseException("Length != 4", 0);
    StringBuilder builder = new StringBuilder();
    for (int i = 0; i < 4; i++)
    {
        int b = a[i];
        if (b < 0) b += 256;
        builder.append(String.valueOf(b));
        if (i < 3) builder.append('.');
    }
    return builder.toString();
}
```

反过来，stringToValue方法解析这个字符串，并且在该字符串有效的情况下产生一个byte[]对象。如果该字符串无效，则抛出ParseException。

```
public Object stringToValue(String text) throws ParseException
{
    StringTokenizer tokenizer = new StringTokenizer(text, ".");
    byte[] a = new byte[4];
    for (int i = 0; i < 4; i++)
    {
        int b = 0;
        try
        {
            b = Integer.parseInt(tokenizer.nextToken());
        }
        catch (NumberFormatException e)
        {
            throw new ParseException("Not an integer", 0);
        }
        if (b < 0 || b >= 256)
            throw new ParseException("Byte out of range", 0);
        a[i] = (byte) b;
    }
    return a;
}
```

在示例程序中试验一下IP地址文本框，如果你键入了无效的地址，那么这个文本框就会转换到最后一个有效的地址。

程序清单6-13中的程序展示了各种格式化的文本框（参见图6-37），点击OK按钮可以从这

些文本框中获取当前的值。

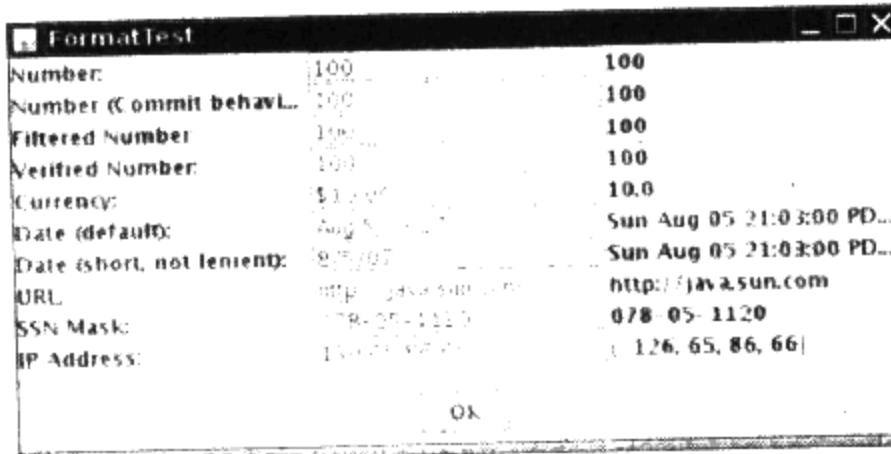


图6-37 FormatTest程序

注意：“Swing Connection”在线通讯有一篇短文描述了一个可以与任何正则表达式匹配的格式器。参加<http://java.sun.com/products/jfc/tsc/articles/reftf/>。

### 程序清单6-13 FormatTest.java

```

1 import java.awt.*;
2 import java.awt.event.*;
3 import java.net.*;
4 import java.text.*;
5 import java.util.*;
6 import javax.swing.*;
7 import javax.swing.text.*;
8
9 /**
10 * A program to test formatted text fields
11 * @version 1.02 2007-06-12
12 * @author Cay Horstmann
13 */
14 public class FormatTest
15 {
16     public static void main(String[] args)
17     {
18         EventQueue.invokeLater(new Runnable()
19         {
20             public void run()
21             {
22                 FormatTestFrame frame = new FormatTestFrame();
23                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24                 frame.setVisible(true);
25             }
26         });
27     }
28 }
29
30 /**
31 * A frame with a collection of formatted text fields and a button that displays the
32 * field values.
33 */
34 class FormatTestFrame extends JFrame
35 {

```

```
36.     public FormatTestFrame()
37.     {
38.         setTitle("FormatTest");
39.         setSize(WIDTH, HEIGHT);
40.
41.         JPanel buttonPanel = new JPanel();
42.         okButton = new JButton("Ok");
43.         buttonPanel.add(okButton);
44.         add(buttonPanel, BorderLayout.SOUTH);
45.
46.         mainPanel = new JPanel();
47.         mainPanel.setLayout(new GridLayout(0, 3));
48.         add(mainPanel, BorderLayout.CENTER);
49.
50.         JFormattedTextField intField =
51.             new JFormattedTextField(NumberFormat.getIntegerInstance());
52.         intField.setValue(new Integer(100));
53.         addRow("Number:", intField);
54.
55.         JFormattedTextField intField2 =
56.             new JFormattedTextField(NumberFormat.getIntegerInstance());
57.         intField2.setValue(new Integer(100));
58.         intField2.setFocusLostBehavior(JFormattedTextField.COMMIT);
59.         addRow("Number (Commit behavior):", intField2);
60.
61.         JFormattedTextField intField3 = new JFormattedTextField(new InternationalFormatter(
62.             NumberFormat.getIntegerInstance())
63.         {
64.             protected DocumentFilter getDocumentFilter()
65.             {
66.                 return filter;
67.             }
68.
69.             private DocumentFilter filter = new IntFilter();
70.         });
71.         intField3.setValue(new Integer(100));
72.         addRow("Filtered Number", intField3);
73.
74.         JFormattedTextField intField4 =
75.             new JFormattedTextField(NumberFormat.getIntegerInstance());
76.         intField4.setValue(new Integer(100));
77.         intField4.setInputVerifier(new FormattedTextFieldVerifier());
78.         addRow("Verified Number:", intField4);
79.
80.         JFormattedTextField currencyField = new JFormattedTextField(NumberFormat
81.             .getCurrencyInstance());
82.         currencyField.setValue(new Double(10));
83.         addRow("Currency:", currencyField);
84.
85.         JFormattedTextField dateField = new JFormattedTextField(DateFormat.getDateInstance());
86.         dateField.setValue(new Date());
87.         addRow("Date (default):", dateField);
88.
89.         DateFormat format = DateFormat.getDateInstance(DateFormat.SHORT);
90.         format.setLenient(false);
91.         JFormattedTextField dateField2 = new JFormattedTextField(format);
92.         dateField2.setValue(new Date());
```

```
93     addRow("Date (short, not lenient):", dateField2);
94
95     try
96     {
97         DefaultFormatter formatter = new DefaultFormatter();
98         formatter.setOverwriteMode(false);
99         JFormattedTextField urlField = new JFormattedTextField(formatter);
100        urlField.setValue(new URL("http://java.sun.com"));
101        addRow("URL:", urlField);
102    }
103    catch (MalformedURLException e)
104    {
105        e.printStackTrace();
106    }
107
108    try
109    {
110        MaskFormatter formatter = new MaskFormatter("###-##-####");
111        formatter.setPlaceholderCharacter('0');
112        JFormattedTextField ssnField = new JFormattedTextField(formatter);
113        ssnField.setValue("078-05-1120");
114        addRow("SSN Mask:", ssnField);
115    }
116    catch (ParseException exception)
117    {
118        exception.printStackTrace();
119    }
120
121    JFormattedTextField ipField = new JFormattedTextField(new IPAddressFormatter());
122    ipField.setValue(new byte[] { (byte) 130, 65, 86, 66 });
123    addRow("IP Address:", ipField);
124}
125
126 /**
127 * Adds a row to the main panel.
128 * @param labelText the label of the field
129 * @param field the sample field
130 */
131 public void addRow(String labelText, final JFormattedTextField field)
132 {
133     mainPanel.add(new JLabel(labelText));
134     mainPanel.add(field);
135     final JLabel valueLabel = new JLabel();
136     mainPanel.add(valueLabel);
137     okButton.addActionListener(new ActionListener()
138     {
139         public void actionPerformed(ActionEvent event)
140         {
141             Object value = field.getValue();
142             Class<?> cl = value.getClass();
143             String text = null;
144             if (cl.isArray())
145             {
146                 if (cl.getComponentType().isPrimitive())
147                 {
148                     try
149                     {
```

```
150.             text = Arrays.class.getMethod("toString", c1).invoke(null, value)
151.                     .toString();
152.         }
153.         catch (Exception ex)
154.         {
155.             // ignore reflection exceptions
156.         }
157.     }
158.     else text = Arrays.toString((Object[]) value);
159. }
160. else text = value.toString();
161. valueLabel.setText(text);
162. }
163. });
164. }
165.
166. public static final int WIDTH = 500;
167. public static final int HEIGHT = 250;
168.
169. private JButton okButton;
170. private JPanel mainPanel;
171. }
172.
173. /**
174. * A filter that restricts input to digits and a '-' sign.
175. */
176. class IntFilter extends DocumentFilter
177. {
178.     public void insertString(FilterBypass fb, int offset, String string, AttributeSet attr)
179.             throws BadLocationException
180.     {
181.         StringBuilder builder = new StringBuilder(string);
182.         for (int i = builder.length() - 1; i >= 0; i--)
183.         {
184.             int cp = builder.codePointAt(i);
185.             if (!Character.isDigit(cp) && cp != '-')
186.             {
187.                 builder.deleteCharAt(i);
188.                 if (Character.isSupplementaryCodePoint(cp))
189.                 {
190.                     i--;
191.                     builder.deleteCharAt(i);
192.                 }
193.             }
194.         }
195.         super.insertString(fb, offset, builder.toString(), attr);
196.     }
197.
198.     public void replace(FilterBypass fb, int offset, int length, String string,
199.                         AttributeSet attr)
200.             throws BadLocationException
201.     {
202.         if (string != null)
203.         {
204.             StringBuilder builder = new StringBuilder(string);
205.             for (int i = builder.length() - 1; i >= 0; i--)
```

```
206     {
207         int cp = builder.codePointAt(i);
208         if (!Character.isDigit(cp) && cp != '-')
209         {
210             builder.deleteCharAt(i);
211             if (Character.isSupplementaryCodePoint(cp))
212             {
213                 i--;
214                 builder.deleteCharAt(i);
215             }
216         }
217     }
218     string = builder.toString();
219 }
220     super.replace(fb, offset, length, string, attr);
221 }
222 }
223
224 /**
225 * A verifier that checks whether the content of a formatted text field is valid.
226 */
227 class FormattedTextFieldVerifier extends InputVerifier
228 {
229     public boolean verify(JComponent component)
230     {
231         JFormattedTextField field = (JFormattedTextField) component;
232         return field.isEditValid();
233     }
234 }
235
236 /**
237 * A formatter for 4-byte IP addresses of the form a.b.c.d
238 */
239 class IPAddressFormatter extends DefaultFormatter
240 {
241     public String valueToString(Object value) throws ParseException
242     {
243         if (!(value instanceof byte[])) throw new ParseException("Not a byte[]", 0);
244         byte[] a = (byte[]) value;
245         if (a.length != 4) throw new ParseException("Length != 4", 0);
246         StringBuilder builder = new StringBuilder();
247         for (int i = 0; i < 4; i++)
248         {
249             int b = a[i];
250             if (b < 0) b += 256;
251             builder.append(String.valueOf(b));
252             if (i < 3) builder.append('.');
253         }
254         return builder.toString();
255     }
256
257     public Object stringToValue(String text) throws ParseException
258     {
259         StringTokenizer tokenizer = new StringTokenizer(text, ".");
260         byte[] a = new byte[4];
261         for (int i = 0; i < 4; i++)
```

```
262.    {
263.        int b = 0;
264.        if (!tokenizer.hasMoreTokens()) throw new ParseException("Too few bytes", 0);
265.        try
266.        {
267.            b = Integer.parseInt(tokenizer.nextToken());
268.        }
269.        catch (NumberFormatException e)
270.        {
271.            throw new ParseException("Not an integer", 0);
272.        }
273.        if (b < 0 || b >= 256) throw new ParseException("Byte out of range", 0);
274.        a[i] = (byte) b;
275.    }
276.    if (tokenizer.hasMoreTokens()) throw new ParseException("Too many bytes", 0);
277.    return a;
278.}
279.}
```

### API **javax.swing.JFormattedTextField 1.4**

- **JFormattedTextField(Format fmt)**

构建使用指定格式的文本框。

- **JFormattedTextField(JFormattedTextField.AbstractFormatter formatter)**

构建使用指定格式器的文本框。注意，DefaultFormatter和InternationalFormatter都是JFormattedTextField.AbstractFormatter的子类。

- **Object getValue()**

返回文本框当前的有效值。注意，它可能并不对应于正在编辑的字符串。

- **void setValue(Object value)**

尝试设置给定对象的值。如果格式器不能将该对象转换为字符串，则尝试失败。

- **void commitEdit()**

尝试从编辑的字符串中设置文本框的有效值。如果格式器不能转换该字符串，则该尝试可能失败。

- **boolean isEditValid()**

检查编辑的字符串表示的是否是一个有效值。

- **int getFocusLostBehavior()**

- **void setFocusLostBehavior(int behavior)**

获取或设置“失去焦点”的行为。表示该行为的合法值是JFormattedTextField类的常量COMMIT\_OR\_REVERT、REVERT、COMMIT和PERSIST。

### API **javax.swing.JFormattedTextField.AbstractFormatter 1.4**

- **abstract String valueToString(Object value)**

将值转换为可编辑的字符串。如果值并不适用于这个格式器，则抛出ParseException。

- **abstract Object stringToValue(String s)**

将字符串转换为值。如果s格式不合适，则抛出ParseException。

- **DocumentFilter getDocumentFilter()**

覆盖该方法以提供可以限制该文本框输入的文档过滤器。null返回值表示不需要任何过滤机制。

**API** **javax.swing.text.DefaultFormatter 1.3**

- **boolean getOverwriteMode()**
- **void setOverwriteMode(boolean mode)**

获取或设置覆写模式。如果确实处于覆写模式，那么在编辑文本时，新字符会覆写现有字符。

**API** **javax.swing.text.DocumentFilter 1.4**

- **void insertString(DocumentFilter.FilterBypass bypass, int offset, String text, AttributeSet attrib)**

在字符串插入到文档中之前被调用。可以覆盖该方法并修改字符串。可以禁止插入，方法是不要调用super.insertString方法，或者调用bypass方法来修改没有过滤机制的文档。

参数：bypass 这是一个允许我们执行绕开过滤器的编辑命令的对象

offset 插入文本处的偏移量

text 待插入的字符

attrib 待插入文本的格式化属性

- **void replace(DocumentFilter.FilterBypass bypass, int offset, int length, String text, AttributeSet attrib)**

在文档的部分内容被替换为新字符串之前被调用。可以覆盖该方法并修改字符串。可以禁止替换，方法是不要调用super.replace，或者调用bypass方法来修改没有过滤机制的文档。

参数：bypass 这是一个允许我们执行绕开过滤器的编辑命令的对象

offset 插入文本处的偏移量

length 被替换部分的长度

text 待插入的字符

attrib 待插入文本的格式化属性

- **void remove(DocumentFilter.FilterBypass bypass, int offset, int length)**

在文本的部分内容被删除之前被调用。如果需要分析移除的效果，可以通过调用**bypass.getDocument()**来获取该文档。

参数：bypass 这是一个允许我们执行绕开过滤器的编辑命令的对象

offset 待移除部分的偏移量

length 待移除部分的长度

**API** javax.swing.text.MaskFormatter 1.4• **MaskFormatter(String mask)**

用给定的掩码构建掩码格式器。参见表6-3以了解掩码中的符号。

• **String getValidCharacters()**• **void setValidCharacters(String characters)**

获取或设置有效的编辑字符。对于掩码中的可变部分，只有位于给定字符串中的字符才是可接受的。

• **String getInvalidCharacters()**• **void setInvalidCharacters(String characters)**

获取或设置无效的编辑字符。在给定字符串中的任何字符都不能作为输入接受。

• **char getPlaceholderCharacter()**• **void setPlaceholderCharacter(char ch)**

获取或设置占位字符，这些字符用于用户没有提供的掩码中的可变字符。默认的占位符是空格。

• **String getPlaceholder()**• **void setPlaceholder(String s)**

获取或设置占位字符串。如果用户没有提供掩码中的所有可变字符，那么就会使用该字符串的末端。如果该字符串为null，或者比掩码短，那么占位符就会填充剩余的输入。

• **boolean getValueContainsLiteralCharacters()**• **void setValueContainsLiteralCharacters(boolean b)**

获取或设置“值包含字面常量字符”标志。如果该标志为true，那么该文本框的值就包含掩码中的字面常量（不可变）部分。如果该标志为false，那么字面常量字符将被移除。其默认值为true。

### 6.4.3 JSpinner构件

JSpinner是包含一个文本框以及两个在文本框旁边的小按钮的构件。当点击按钮时，文本框的值就会递增或递减（参见图6-38）。

微调器中的值可以是数字、日期、列表中的值，或者是更为普遍的情况，即前驱和后继可以确定的任何值序列。JSpinner类为前三种情况定义了标准的数据模型。我们可以定义自己的数据模型来描述任意的序列。

默认情况下，微调器管理着一个整数，并且两个按钮将对其进行递增和递减。可以通过调用getValue方法来获取当前值，这个方法将返回一个Object，应该将其转型为Integer并获取其中包装的值。

```
JSpinner defaultSpinner = new JSpinner();
```

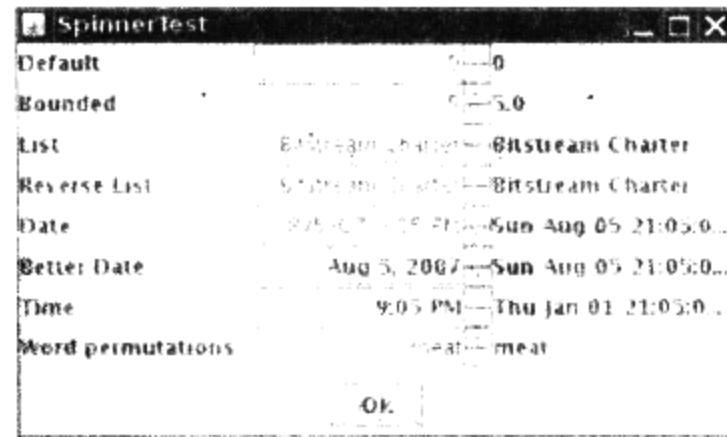


图6-38 JSpinner构件的数种变体

```
int value = (Integer) defaultSpinner.getValue();
```

我们可以将递增的值修改为1之外的值，还可以提供递增的上界和下届。下面的微调器的初始值为5，边界为0到10，每次递增0.5：

```
JSpinner boundedSpinner = new JSpinner(new SpinnerNumberModel(5, 0, 10, 0.5));
```

**SpinnerNumberModel**有两个构造器，其中一个只有int参数，而另一个有double参数。只要有参数是浮点数，就会使用第二个构造器，它会将微调器的值设置为Double对象。

微调器并未限制为只能是数字型值，我们可以用微调器迭代任何值集合，只需将一个**SpinnerListModel**传递给**JSpinner**构造器即可。我们可以从数组或实现了List接口的类（例如**ArrayList**）中构建**SpinnerListModel**。在示例程序中，我们显示了一个微调控制器，它的值是所有可用的字体名。

```
String[] fonts = GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyNames();
JSpinner listSpinner = new JSpinner(new SpinnerListModel(fonts));
```

但是，我们发现迭代的方向略有些令人疑惑，因为它与用户关于组合框的体验相关。在组合框中，较高的值在较低的值的下面，因此，我们用向下箭头来导航到较高的值。但是微调器将递增数组索引，使得向上箭头可以产生较高的值。在**SpinnerListModel**中没有用于颠倒遍历顺序的方法，但是临时创建一个匿名子类就可以产生想要的结果：

```
JSpinner reverseListSpinner = new JSpinner(
    new SpinnerListModel(fonts)
{
    public Object getNextValue()
    {
        return super.getPreviousValue();
    }

    public Object getPreviousValue()
    {
        return super.getNextValue();
    }
});
```

试运行这两个版本，看看哪一个更直观些。

微调器的另一个大显身手之处是可以让用户递增或递减的日期。用下面的调用就可以获得这样的微调器，并用当日的时间进行初始化。

```
JSpinner dateSpinner = new JSpinner(new SpinnerDateModel());
```

但是，如果你仔细查看图6-38，就会发现微调器文本同时显示了日期和时间，例如

8/05/07 7:23 PM

时间对于日期选择器来说没有任何意义，而让微调器只显示日期被证明有些困难，下面就是这样的“魔咒”：

```
JSpinner betterDateSpinner = new JSpinner(new SpinnerDateModel());
String pattern = ((SimpleDateFormat) DateFormat.getDateInstance()).toPattern();
betterDateSpinner.setEditor(new JSpinner.DateEditor(betterDateSpinner, pattern));
```

使用相同的方法，还可以创建一个时间选择器

```
JSpinner timeSpinner = new JSpinner(new SpinnerDateModel());
pattern = ((SimpleDateFormat) DateFormat.getTimeInstance(DateFormat.SHORT)).toPattern();
timeSpinner.setEditor(new JSpinner.DateEditor(timeSpinner, pattern));
```

通过定义自己的微调器模型，你可以在微调器中显示任意的序列。在示例程序中，我们用一个微调器迭代了字符串“meat”的所有排列。你可以通过点击微调器按钮来获取“mate”、“meta”、“team”以及其他20种排列。

在定义自己的模型时，需要扩展AbstractSpinnerModel类并定义下面的4个方法：

```
Object getValue()
void setValue(Object value)
Object getNextValue()
Object getPreviousValue()
```

getValue方法将返回模型存储的值，而setValue方法则把这个值设置为新值，如果新值并不适合用于设置，则该方法会抛出IllegalArgumentException。

**X** **警告：** setValue方法必须在设置新值之后调用fireStateChanged方法，否则，微调器文本框并不会更新。

getNextValue和getPreviousValue方法将分别位于返回当前值之后和之前的值，或者在到达遍历的终点时返回null。

**X** **警告：** getNextValue和getPreviousValue方法不应该改变当前值。当用户点击微调器的向上箭头时，getNextValue方法就会被调用。如果其返回值不是null，微调器的值会通过一个对setValue的调用进行设置。

在示例程序中，我们使用了标准的算法来确定下一个和前一个排列，而这个算法的细节並不重要。

程序清单6-14展示了如何生成各种不同的微调器类型，请点击Ok按钮以观察微调器的值。

#### 程序清单6-14 SpinnerTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.text.*;
4. import java.util.*;
5. import javax.swing.*;
6.
7. /**
8.  * A program to test spinners.
9. */
10. public class SpinnerTest
11 {
12     public static void main(String[] args)
13     {
14         SpinnerFrame frame = new SpinnerFrame();
15         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16         frame.setVisible(true);
17     }
18 }
19.
```

```
20. /**
21.  A frame with a panel that contains several spinners and
22.  a button that displays the spinner values.
23. */
24. class SpinnerFrame extends JFrame
25. {
26.     public SpinnerFrame()
27.     {
28.         setTitle("SpinnerTest");
29.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
30.         JPanel buttonPanel = new JPanel();
31.         okButton = new JButton("Ok");
32.         buttonPanel.add(okButton);
33.         add(buttonPanel, BorderLayout.SOUTH);
34.
35.         mainPanel = new JPanel();
36.         mainPanel.setLayout(new GridLayout(0, 3));
37.         add(mainPanel, BorderLayout.CENTER);
38.
39.         JSpinner defaultSpinner = new JSpinner();
40.         addRow("Default", defaultSpinner);
41.
42.         JSpinner boundedSpinner = new JSpinner(new SpinnerNumberModel(5, 0, 10, 0.5));
43.         addRow("Bounded", boundedSpinner);
44.
45.         String[] fonts = GraphicsEnvironment
46.             .getLocalGraphicsEnvironment()
47.             .getAvailableFontFamilyNames();
48.
49.         JSpinner listSpinner = new JSpinner(new SpinnerListModel(fonts));
50.         addRow("List", listSpinner);
51.
52.         JSpinner reverseListSpinner = new JSpinner(
53.             new
54.                 SpinnerListModel(fonts)
55.             {
56.                 public Object getNextValue()
57.                 {
58.                     return super.getPreviousValue();
59.                 }
60.                 public Object getPreviousValue()
61.                 {
62.                     return super.getNextValue();
63.                 }
64.             });
65.         addRow("Reverse List", reverseListSpinner);
66.
67.         JSpinner dateSpinner = new JSpinner(new SpinnerDateModel());
68.         addRow("Date", dateSpinner);
69.
70.         JSpinner betterDateSpinner = new JSpinner(new SpinnerDateModel());
71.         String pattern = ((SimpleDateFormat) DateFormat.getDateInstance()).toPattern();
72.         betterDateSpinner.setEditor(new JSpinner.DateEditor(betterDateSpinner, pattern));
73.         addRow("Better Date", betterDateSpinner);
74.
75.         JSpinner timeSpinner = new JSpinner(
76.             new SpinnerDateModel()
```

```
77.         new GregorianCalendar(2000, Calendar.JANUARY, 1, 12, 0, 0).getTime(),
78.             null, null, Calendar.HOUR));
79.     addRow("Time", timeSpinner);
80.
81.     JSpinner permSpinner = new JSpinner(new PermutationSpinnerModel("meat"));
82.     addRow("Word permutations", permSpinner);
83. }
84.
85. /**
86.  * Adds a row to the main panel.
87.  * @param labelText the label of the spinner
88.  * @param spinner the sample spinner
89. */
90. public void addRow(String labelText, final JSpinner spinner)
91. {
92.     mainPanel.add(new JLabel(labelText));
93.     mainPanel.add(spinner);
94.     final JLabel valueLabel = new JLabel();
95.     mainPanel.add(valueLabel);
96.     okButton.addActionListener(new
97.         ActionListener()
98.     {
99.         public void actionPerformed(ActionEvent event)
100.         {
101.             Object value = spinner.getValue();
102.             valueLabel.setText(value.toString());
103.         }
104.     });
105. }
106.
107. public static final int DEFAULT_WIDTH = 400;
108. public static final int DEFAULT_HEIGHT = 250;
109.
110. private JPanel mainPanel;
111. private JButton okButton;
112. }
113.
114. /**
115.  * A model that dynamically generates word permutations
116. */
117. class PermutationSpinnerModel extends AbstractSpinnerModel
118. {
119.     /**
120.      * Constructs the model.
121.      * @param w the word to permute
122.     */
123.     public PermutationSpinnerModel(String w)
124.     {
125.         word = w;
126.     }
127.
128.     public Object getValue()
129.     {
130.         return word;
131.     }
132.
```

```
133.     public void setValue(Object value)
134.     {
135.         if (!(value instanceof String))
136.             throw new IllegalArgumentException();
137.         word = (String) value;
138.         fireStateChanged();
139.     }
140.
141.     public Object getNextValue()
142.     {
143.         int[] codePoints = toCodePointArray(word);
144.         for (int i = codePoints.length - 1; i > 0; i--)
145.         {
146.             if (codePoints[i - 1] < codePoints[i])
147.             {
148.                 int j = codePoints.length - 1;
149.                 while (codePoints[i - 1] > codePoints[j]) j--;
150.                 swap(codePoints, i - 1, j);
151.                 reverse(codePoints, i, codePoints.length - 1);
152.                 return new String(codePoints, 0, codePoints.length);
153.             }
154.         }
155.         reverse(codePoints, 0, codePoints.length - 1);
156.         return new String(codePoints, 0, codePoints.length);
157.     }
158.
159.     public Object getPreviousValue()
160.     {
161.         int[] codePoints = toCodePointArray(word);
162.         for (int i = codePoints.length - 1; i > 0; i--)
163.         {
164.             if (codePoints[i - 1] > codePoints[i])
165.             {
166.                 int j = codePoints.length - 1;
167.                 while (codePoints[i - 1] < codePoints[j]) j--;
168.                 swap(codePoints, i - 1, j);
169.                 reverse(codePoints, i, codePoints.length - 1);
170.                 return new String(codePoints, 0, codePoints.length);
171.             }
172.         }
173.         reverse(codePoints, 0, codePoints.length - 1);
174.         return new String(codePoints, 0, codePoints.length);
175.     }
176.
177.     private static int[] toCodePointArray(String str)
178.     {
179.         int[] codePoints = new int[str.codePointCount(0, str.length())];
180.         for (int i = 0, j = 0; i < str.length(); i++, j++)
181.         {
182.             int cp = str.codePointAt(i);
183.             if (Character.isSupplementaryCodePoint(cp)) i++;
184.             codePoints[j] = cp;
185.         }
186.         return codePoints;
187.     }
188.
189.     private static void swap(int[] a, int i, int j)
```

```
190. {
191.     int temp = a[i];
192.     a[i] = a[j];
193.     a[j] = temp;
194. }
195.
196. private static void reverse(int[] a, int i, int j)
197. {
198.     while (i < j) { swap(a, i, j); i++; j--; }
199. }
200.
201. private String word;
202. }
```

## API **javax.swing.JSpinner 1.4**

- **JSpinner()**

构建一个微调器，它可以编辑从0开始、每次递增1，并且没有边界的整数值。

- **JSpinner(SpinnerModel model)**

构建一个微调器，它将使用给定的数据模型。

- **Object getValue()**

获取微调器的当前值。

- **void setValue(Object value)**

尝试着设置微调器的值，如果模型不接受这个值，将抛出IllegalArgumentException。

- **void setEditor(JComponent editor)**

设置用于编辑微调器值的构件。

## API **javax.swing.SpinnerNumberModel 1.4**

- **SpinnerNumberModel(int initval, int minimum, int maximum, int stepSize)**

- **SpinnerNumberModel(double initval, double minimum, double maximum, double stepSize)**

这些构造器将产生一个管理Integer或Double类型值的数字模型。可以用Integer或Double类的MIN\_VALUE和MAX\_VALUE常量来表示不受边界限制的值。

参数：initval 值的间距

minimum 最小值

maximum 最大值

stepSize 每次微调的递增或递减量

## API **javax.swing.SpinnerListModel 1.4**

- **SpinnerListModel(Object[] values)**

- **SpinnerListModel(List values)**

这些构造器将产生从给定的值中选择一个值的模型。

**API** javax.swing.SpinnerDateModel 1.4

- `SpinnerDateModel()`

用当日的日期作为初始值构建一个日期模型，在该模型中没有上界和下界，其递增量为`Calendar.DAY_OF_MONTH`。

- `SpinnerDateModel(Date initval, Comparable minimum, Comparable maximum, int step)`

参数：`initval` 初始值

`minimum` 最小值，在不希望有下界时为null

`maximum` 最大值，在不希望有上界时为null

`step` 每次微调递增或递减的日期，它的值是`Calendar`类的常量ERA、YEAR、MONTH、WEEK\_OF\_YEAR、WEEK\_OF\_MONTH、DAY\_OF\_MONTH、DAY\_OF\_YEAR、DAY\_OF\_WEEK、DAY\_OF\_WEEK\_IN\_MONTH、AM\_PM、HOUR、HOUR\_OF\_DAY、MINUTE、SECOND或MILLISECOND之一

**API** java.text.SimpleDateFormat 1.1

- `String toPattern() 1.2`

获取用于这个日期格式器的编辑模式。典型的模式为“yyyy-MM-dd”，参见Java SE文档以了解关于该模式的详细信息。

**API** javax.swing.JSpinner.DateEditor 1.4

- `DateEditor(JSpinner spinner, String pattern)`

构建一个用于微调器的日期编辑器。

参数：`spinner` 该编辑器所属的微调器

`pattern` 用于相关联的`SimpleDateFormat`的格式化模式

**API** javax.swing.AbstractSpinnerModel 1.4

- `Object getValue()`

获取该模型的当前值。

- `void setValue(Object value)`

尝试着设置用于该模型的新值。如果这个值不可接受，则抛出`IllegalArgumentException`。

当覆盖该方法时，应该在设置新值之后调用`fireStateChanged`。

- `Object getNextValue()`

- `Object getPreviousValue()`

计算（但不是设置）该模型所定义的序列中的下一个和前一个值。

#### 6.4.4 用JEditorPane显示HTML

与之前我们讨论的文本构件不同，`JEditorPane`能够以HTML和RTF的格式显示和编辑文本。（RTF即“富文本格式”，是许多微软应用进行文档交换的格式。它是一种弱文档格式，即

使在微软自己的应用之间也无法很好地运行。在本书中我们将不介绍RTF的应用。)

坦白地说，JEditorPane的功能还不尽如人意。HTML绘制器只能显示简单的文件，但是对于在Web上经常出现的复杂页面，它往往难于处理。HTML编辑器不仅功能有限，而且还不稳定。

JEditorPane看似合理的一种应用就是以HTML的形式显示程序的帮助文档。因为你有权控制你提供的帮助文件，所以可以避开JEditorPane不能很好显示的特性。

 **注意：**如果想获得有关业界强度的帮助系统的更多信息，请到网站<http://java.sun.com/products/javahelp/index.html>上查看JavaHelp。

程序清单6-15中的程序代码包含一个编辑器面板，用于显示HTML页面的内容。在文本框中键入一个URL，该URL必须以http:或file:开头，接着点击Load按钮，选定的HTML页面就会显示到编辑器面板中（参见图6-39）。

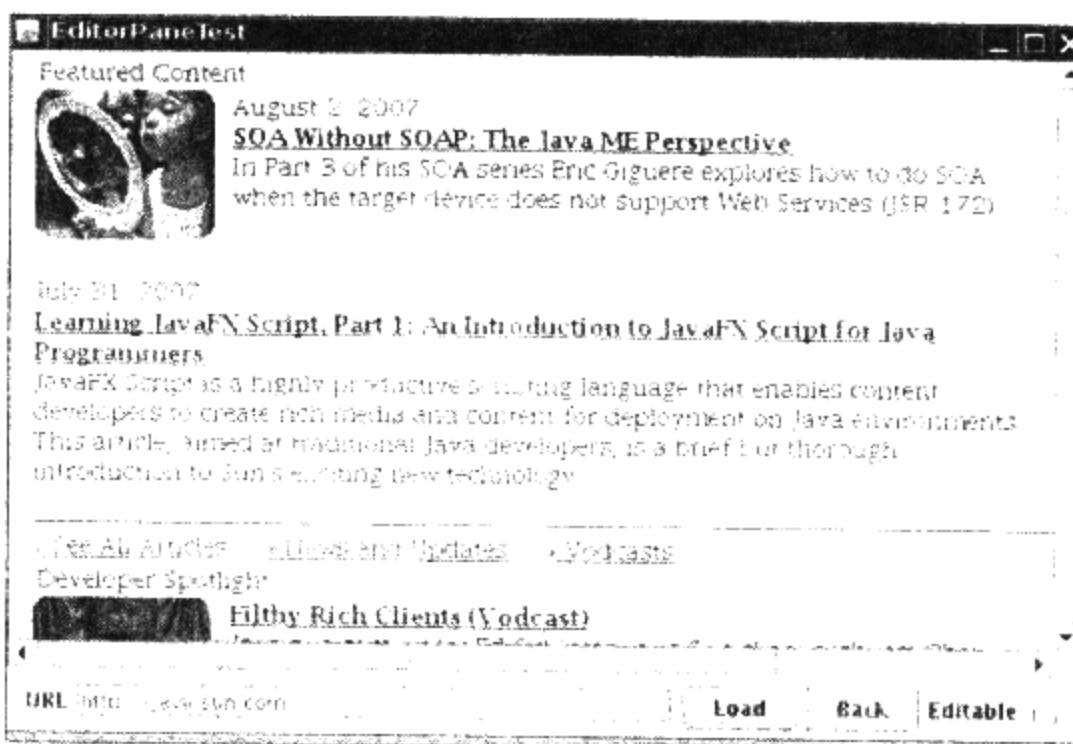


图6-39 显示一个HTML页面的编辑器面板

该超链接是活动的：如果你点击一个链接，该应用程序就将其载入。Back按钮可以返回前一页面。

这个程序实际上是一个非常简单的浏览器。当然，它并不具有你期望从商业浏览器可获得的任何舒适特性，例如页面缓冲或者书签列表等。该编辑器面板甚至不能显示Applet。

如果你点击Editable复选框，那么编辑器面板就会成为可编辑的。你可以键入文本，并且可以使用BACKSPACE删除文本。该构件还能够理解用于剪切、复制以及粘贴的CTRL+X、CTRL+C以及CTRL+V快捷键。不过，还必须进行一些编程来添加对字体和格式的支持。

当该构件变成可编辑的之后，超链接就不是活动的了。另外，对于一些Web页面，在启动编辑模式的时候（参见图6-40），你可以看到JavaScript命令、注释以及其他一些标签。这个示例程序可以让你查看到编辑的特性，但是我们建议在程序中忽略这些特性。

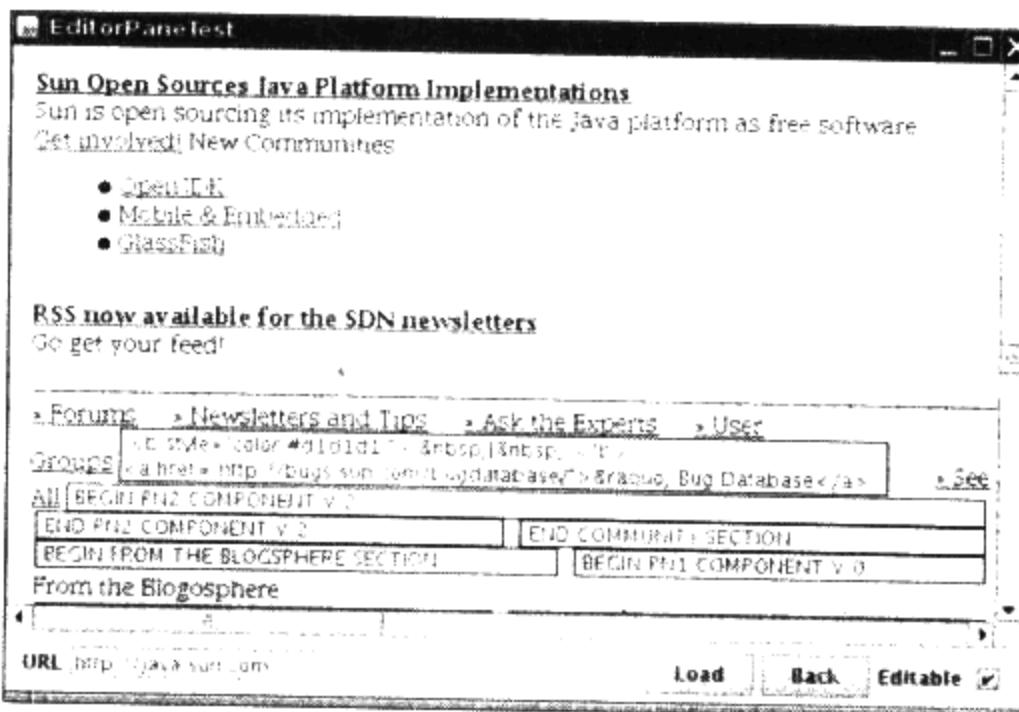


图6-40 处于编辑模式的编辑器面板

**!** 提示：在默认情况下，JEditorPane是处于编辑模式的。可以调用editorPane.setEditable(false)将其关闭。

在该示例程序中所看到的编辑器面板的一些特性是很容易使用的，可以使用 setPage方法载入一个新文档。例如，

```
JEditorPane editorPane = new JEditorPane();
editorPane.setPage(url);
```

其参数要么是一个字符串，要么是一个URL对象。JEditorPane类继承了JTextComponent类。因此，也可以调用只能显示纯文本的setText方法。

**!** 提示：关于 setPage是否是在一个单独的线程中载入一个新的文档，它的API文档写得也不是很清楚（这也是通常认为JEditorPane不成功的原因所在）。不过，可以使用下面几条语句在一个单独线程中强行载入：

```
AbstractDocument doc = (AbstractDocument) editorPane.getDocument();
doc.setAsynchronousLoadPriority(0);
```

为了监听超链接的点击事件，需要添加一个HyperlinkListener。HyperlinkListener接口只有一个单一方法hyperlinkUpdate，当用户移到或点击一个超链接的时候，该方法就会被调用。该方法接收一个类型为HyperlinkEvent的数据作为参数。

需要调用getEventType方法以确定发生了什么类型的事件。下面是三种可能的返回值：

```
HyperlinkEvent.EventType.ACTIVATED
HyperlinkEvent.EventType.ENTERED
HyperlinkEvent.EventType.EXITED
```

第一个值表明用户点击了该超链接。在这种情况下，通常希望打开一个新的链接，可以使用第二个值和第三个值提供可视化的反馈信息，例如当鼠标停留在一个链接上面，提供一个工具提示。

注意：至于为什么在HyperlinkListener接口里面不用3个独立的方法来处理启动、进入和退出，完全是一件神秘的事情。

HyperlinkEvent类的getURL方法返回超链接的URL。例如，下面展示了怎样安装一个超链接监听器追踪用户激活的链接：

```
editorPane.addHyperlinkListener(new
    HyperlinkListener()
{
    public void hyperlinkUpdate(HyperlinkEvent event)
    {
        if (event.getEventType() == HyperlinkEvent.EventType.ACTIVATED)
        {
            try
            {
                editorPane.setPage(event.getURL());
            }
            catch (IOException e)
            {
                editorPane.setText("Exception: " + e);
            }
        }
    }
});
```

事件处理器仅仅获得URL，并更新编辑器面板。 setPage方法可以抛出一个IOException异常，在这种情况下，我们将一条错误消息作为纯文本进行显示。

程序清单6-15展示了构建一个HTML帮助系统所需的全部特性。从本质上讲，JEditorPane比树和表格构件都要复杂。不过，如果不需要编写定制文本格式的文本编辑器或者绘制器，这些复杂性就会自动对你隐藏起来了。

#### 程序清单6-15 EditorPaneTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.io.*;
4. import java.util.*;
5. import javax.swing.*;
6. import javax.swing.event.*;

7.
8. /**
9. * This program demonstrates how to display HTML documents in an editor pane.
10 * @version 1.03 2007-08-01
11 * @author Cay Horstmann
12 */
13. public class EditorPaneTest
14. {
15.     public static void main(String[] args)
16.     {
17.         EventQueue.invokeLater(new Runnable()
18.         {
19.             public void run()
20.             {
21.                 JFrame frame = new EditorPaneFrame();
```

```
22         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23         frame.setVisible(true);
24     }
25   });
26 }
27 }
28
29 /**
30 * This frame contains an editor pane, a text field and button to enter a URL and load
31 * a document, and a Back button to return to a previously loaded document.
32 */
33 class EditorPaneFrame extends JFrame
34 {
35     public EditorPaneFrame()
36     {
37         setTitle("EditorPaneTest");
38         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
39
40         final Stack<String> urlStack = new Stack<String>();
41         final JEditorPane editorPane = new JEditorPane();
42         final JTextField url = new JTextField(30);
43
44         // set up hyperlink listener
45
46         editorPane.setEditable(false);
47         editorPane.addHyperlinkListener(new HyperlinkListener()
48         {
49             public void hyperlinkUpdate(HyperlinkEvent event)
50             {
51                 if (event.getEventType() == HyperlinkEvent.EventType.ACTIVATED)
52                 {
53                     try
54                     {
55                         // remember URL for back button
56                         urlStack.push(event.getURL().toString());
57                         // show URL in text field
58                         url.setText(event.getURL().toString());
59                         editorPane.setPage(event.getURL());
60                     }
61                     catch (IOException e)
62                     {
63                         editorPane.setText("Exception: " + e);
64                     }
65                 }
66             }
67         });
68         // set up checkbox for toggling edit mode
69
70         final JCheckBox editable = new JCheckBox();
71         editable.addActionListener(new ActionListener()
72         {
73             public void actionPerformed(ActionEvent event)
74             {
75                 editorPane.setEditable(editable.isSelected());
76             }
77         });
78 }
```

```
78.  
79.    // set up load button for loading URL  
80.  
81.    ActionListener listener = new ActionListener()  
82.    {  
83.        public void actionPerformed(ActionEvent event)  
84.        {  
85.            try  
86.            {  
87.                // remember URL for back button  
88.                urlStack.push(url.getText());  
89.                editorPane.setPage(url.getText());  
90.            }  
91.            catch (IOException e)  
92.            {  
93.                editorPane.setText("Exception: " + e);  
94.            }  
95.        }  
96.    };  
97.  
98. JButton loadButton = new JButton("Load");  
99. loadButton.addActionListener(listener);  
100. url.addActionListener(listener);  
101.  
102. // set up back button and button action  
103.  
104. JButton backButton = new JButton("Back");  
105. backButton.addActionListener(new ActionListener()  
106. {  
107.     public void actionPerformed(ActionEvent event)  
108.     {  
109.         if (urlStack.size() <= 1) return;  
110.         try  
111.         {  
112.             // get URL from back button  
113.             urlStack.pop();  
114.             // show URL in text field  
115.             String urlString = urlStack.peek();  
116.             url.setText(urlString);  
117.             editorPane.setPage(urlString);  
118.         }  
119.         catch (IOException e)  
120.         {  
121.             editorPane.setText("Exception: " + e);  
122.         }  
123.     }  
124. });
125.  
126. add(new JScrollPane(editorPane), BorderLayout.CENTER);
127.  
128. // put all control components in a panel  
129.  
130. JPanel panel = new JPanel();
131. panel.add(new JLabel("URL"));
132. panel.add(url);
133. panel.add(loadButton);
134. panel.add(backButton);
```

```

135.     panel.add(new JLabel("Editable"));
136.     panel.add(editable);
137.
138.     add(panel, BorderLayout.SOUTH);
139. }
140.
141. private static final int DEFAULT_WIDTH = 600;
142. private static final int DEFAULT_HEIGHT = 400;
143. }

```

### API **javax.swing.JEditorPane 1.2**

- **void setPage(URL url)**  
将来自于url的页面导入到编辑器面板中。
- **void addHyperlinkListener(HyperLinkListener listener)**  
为该编辑器面板添加一个超链接监听器。

### API **javax.swing.event.HyperlinkListener 1.2**

- **void hyperlinkUpdate(HyperlinkEvent event)**  
无论何时，只要选定了一个超链接，该方法就会被调用。

### API **javax.swing.HyperlinkEvent 1.2**

- **URL getURL()**  
返回所选超链接的URL。

## 6.5 进度指示器

在随后的几节中，我们将讨论用于指示缓慢活动进度的三个类。JProgressBar是一个用于指示进度的Swing构件；ProgressMonitor是一个包含进度条的对话框；在读取流的时候，ProgressMonitorInputStream用于显示进度监视器对话框。

### 6.5.1 进度条

进度条只不过是一个矩形构件，它被部分地填充了颜色以指示一个操作的进度。默认情况下，进度是用字符串“n%”来指示的。在图6-41右下方，你可以看到一个进度条。

通过提供最大值和最小值以及一个可供选择的定位方向，可以像构建一个滑动条那样构建一个进度条：

```

progressBar = new JProgressBar(0, 1000);
progressBar = new JProgressBar(SwingConstants.VERTICAL, 0, 1000);

```

也可以使用setMinimum和setMaximum方法来设置最大值和最小值。

和滑动条不同的是，进度条不能让用户自行调节。你的程序必须调用setValue才能对它进

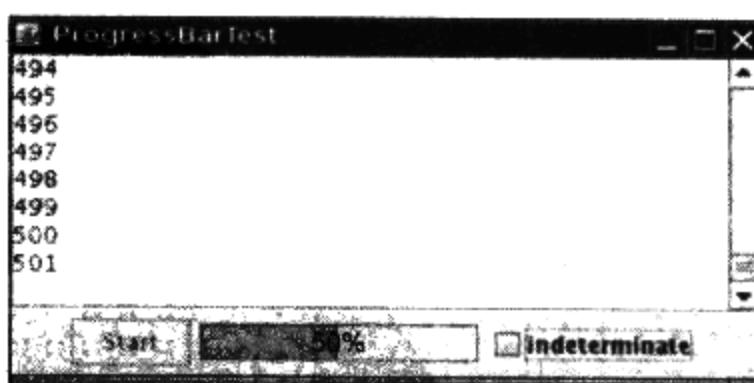


图6-41 进度条

行更新。

如果调用

```
progressBar.setStringPainted(true);
```

那么进度条会计算出某项操作完成的百分比，然后以一个“n%”形式的字符串将它显示出来。如果你想以不同形式的字符串将它显示出来，可以用setString方法提供该字符串：

```
if (progressBar.getValue() > 900)  
    progressBar.setString("Almost Done");
```

程序清单6-16展示了一个进度条，用于监视一个耗时的模拟活动。

SimulatedActivity类将值current每秒钟增加10倍。每当它达到目标值的时候，该任务就结束。我们使用SwingWorker类实现了这项任务并在process方法中更新了进度条，而SwingWorker是在事件分发线程中调用方法的，这样它就可以安全地更新进度条了。（有关Swing中线程安全的更多信息请参见第I卷第14章。）

Java SE 1.4增加了对不确定进度条的支持，这种进度条能够动画显示某种类型的进度，而不具体显示完成情况的百分比。可以在你的浏览器中看到这种类型的进度条，它指示浏览器正在等待服务器，但是无法知道到底可能要等待多久。如果要动画显示“不确定等待”，请调用setIndeterminate方法。

程序清单6-16展示了这个程序的完整代码。

### 程序清单6-16 ProgressBarTest.java

```
1. import java.awt.*;  
2. import java.awt.event.*;  
3. import java.util.List;  
4.  
5. import javax.swing.*;  
6.  
7. /**  
8. * This program demonstrates the use of a progress bar to monitor the progress of a thread.  
9. * @version 1.04 2007-08-01  
10. * @author Cay Horstmann  
11. */  
12. public class ProgressBarTest  
13. {  
14.     public static void main(String[] args)  
15.     {  
16.         EventQueue.invokeLater(new Runnable()  
17.         {  
18.             public void run()  
19.             {  
20.                 JFrame frame = new ProgressBarFrame();  
21.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
22.                 frame.setVisible(true);  
23.             }  
24.         });  
25.     }  
26. }  
27.  
28. /**  
29. * A frame that contains a button to launch a simulated activity, a progress bar, and a
```

```
30. * text area for the activity output.  
31. */  
32. class ProgressBarFrame extends JFrame  
33. {  
34.     public ProgressBarFrame()  
35.     {  
36.         setTitle("ProgressBarTest");  
37.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);  
38.         // this text area holds the activity output  
39.         textArea = new JTextArea();  
40.         // set up panel with button and progress bar  
41.         final int MAX = 1000;  
42.         JPanel panel = new JPanel();  
43.         startButton = new JButton("Start");  
44.         progressBar = new JProgressBar(0, MAX);  
45.         progressBar.setStringPainted(true);  
46.         panel.add(startButton);  
47.         panel.add(progressBar);  
48.         checkBox = new JCheckBox("indeterminate");  
49.         checkBox.addActionListener(new ActionListener()  
50.         {  
51.             public void actionPerformed(ActionEvent event)  
52.             {  
53.                 progressBar.setIndeterminate(checkBox.isSelected());  
54.                 progressBar.setStringPainted(!progressBar.isIndeterminate());  
55.             }  
56.         });  
57.         panel.add(checkBox);  
58.         add(new JScrollPane(textArea), BorderLayout.CENTER);  
59.         add(panel, BorderLayout.SOUTH);  
60.         // set up the button action  
61.         startButton.addActionListener(new ActionListener()  
62.         {  
63.             public void actionPerformed(ActionEvent event)  
64.             {  
65.                 startButton.setEnabled(false);  
66.                 activity = new SimulatedActivity(MAX);  
67.                 activity.execute();  
68.             }  
69.         });  
70.     }  
71.     private JButton startButton;  
72.     private JProgressBar progressBar;  
73.     private JCheckBox checkBox;  
74.     private JTextArea textArea;  
75.     private SimulatedActivity activity;  
76.     public static final int DEFAULT_WIDTH = 400;  
77.     public static final int DEFAULT_HEIGHT = 200;  
78. }
```

```
87. class SimulatedActivity extends SwingWorker<Void, Integer>
88. {
89.     /**
90.      * Constructs the simulated activity that increments a counter from 0 to a
91.      * given target.
92.      * @param t the target value of the counter.
93.      */
94.     public SimulatedActivity(int t)
95.     {
96.         current = 0;
97.         target = t;
98.     }
99.
100.    protected Void doInBackground() throws Exception
101.    {
102.        try
103.        {
104.            while (current < target)
105.            {
106.                Thread.sleep(100);
107.                current++;
108.                publish(current);
109.            }
110.        }
111.        catch (InterruptedException e)
112.        {
113.        }
114.        return null;
115.    }
116.
117.    protected void process(List<Integer> chunks)
118.    {
119.        for (Integer chunk : chunks)
120.        {
121.            textArea.append(chunk + "\n");
122.            progressBar.setValue(chunk);
123.        }
124.    }
125.
126.    protected void done()
127.    {
128.        startButton.setEnabled(true);
129.    }
130.
131.    private int current;
132.    private int target;
133. }
134. }
```

### 6.5.2 进度监视器

进度条是一个很简单的构件，可以放在一个窗体中。相比之下，**ProgressMonitor**是一个完整的包含进度条的对话框（参见图6-42）。这个对话框还包含一个Cancel按钮。如果点击该按钮，那么将会关闭监视器对话框。另外，程序还可以查询用户是否已经取消对话框并终止了监

视活动。(注意：这个类的类名并不是以“J”开头的。)

通过提供下面这些信息，就可以构建一个进度监视器：

- 在其上弹出对话框的父构件。
- 在对话框上显示的一个对象（可能是一个字符串、图标或者是一个构件）。
- 在对象下面显示的一个可选注释。
- 最大值以及最小值。

不过，进度监视器无法自己测量进度或者取消活动。因此，仍须定时调用`setProgress`方法设置进度值。（该方法等价于`JProgressBar`类的`setValue`方法。）在取消监视器活动的时候，请调用`close`方法来撤销对话框。还可以再次调用`start`重新使用该对话框。

使用进度监视器的最大问题在于处理取消请求，因为我们不能将一个事件处理器附加到`Cancel`按钮上，而是应该周期性地调用`isCancel`方法来观察程序用户是否按下了`Cancel`按钮。

如果工作线程可以无限地阻塞下去（例如，在从网络连接中读取输入时），那么它就不能监视`Cancel`按钮。在我们的示例程序中，我们展示了如何使用定时器来达到此目的，另外，我们还让定时器负责更新对进度的度量。

如果运行一下程序清单6-17中的程序，你会观察到进度监视器对话框有一个很有趣的特性。该对话框不会立即出现，相反地，它会等待一小段时间看看活动是否已经完成，或者是否可能在比对话框出现所需时间更短的时间内完成。

可以按照下面的叙述来控制定时。使用`setMillisToDecideToPopup`方法设置在构建对话框对象和确定是否弹出显示对话框之间需要等待的毫秒数。默认值是500毫秒。`setMillisToPopup`是你估计对话框弹出所需的时间。Swing设计者将这个值默认设置为2秒。很显然，他们考虑了这个事实，即Swing对话框不总是按照我们希望的那样立即显示出来。最好不要修改这个值。

### 程序清单6-17 ProgressMonitorTest.java

```

1. import java.awt.*;
2. import java.awt.event.*;
3.
4. import javax.swing.*;
5.
6. /**
7. * A program to test a progress monitor dialog.
8. * @version 1.04 2007-08-01
9. * @author Cay Horstmann
10.*/
11. public class ProgressMonitorTest
12. {
13.     public static void main(String[] args)
14.     {
15.         EventQueue.invokeLater(new Runnable()
16.         {
17.             public void run()
18.             {
19.                 JFrame frame = new ProgressMonitorFrame();

```

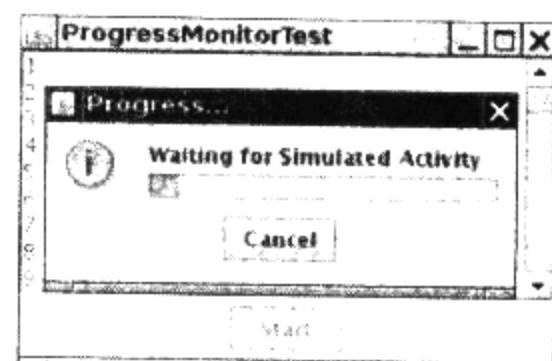


图6-42 一个进度监视器对话框

```
20.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21.         frame.setVisible(true);
22.     }
23. );
24. }
25 }
26
27 /**
28 * A frame that contains a button to launch a simulated activity and a text area for the
29 * activity output.
30 */
31 class ProgressMonitorFrame extends JFrame
32 {
33     public ProgressMonitorFrame()
34     {
35         setTitle("ProgressMonitorTest");
36         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
37
38         // this text area holds the activity output
39         textArea = new JTextArea();
40
41         // set up a button panel
42         JPanel panel = new JPanel();
43         startButton = new JButton("Start");
44         panel.add(startButton);
45
46         add(new JScrollPane(textArea), BorderLayout.CENTER);
47         add(panel, BorderLayout.SOUTH);
48
49         // set up the button action
50
51         startButton.addActionListener(new ActionListener()
52         {
53             public void actionPerformed(ActionEvent event)
54             {
55                 startButton.setEnabled(false);
56                 final int MAX = 1000;
57
58                 // start activity
59                 activity = new SimulatedActivity(MAX);
60                 activity.execute();
61
62                 // launch progress dialog
63                 progressDialog = new ProgressMonitor(ProgressMonitorFrame.this,
64                     "Waiting for Simulated Activity", null, 0, MAX);
65                 cancelMonitor.start();
66             }
67         });
68
69         // set up the timer action
70
71         cancelMonitor = new Timer(500, new ActionListener()
72         {
73             public void actionPerformed(ActionEvent event)
74             {
75                 if (progressDialog.isCanceled())
76                 {
```

```
77.         activity.cancel(true);
78.         startButton.setEnabled(true);
79.     }
80.     else if (activity.isDone())
81.     {
82.         progressDialog.close();
83.         startButton.setEnabled(true);
84.     }
85.     else
86.     {
87.         progressDialog.setProgress(activity.getProgress());
88.     }
89. }
90. });
91. }
92.
93. private Timer cancelMonitor;
94. private JButton startButton;
95. private ProgressMonitor progressDialog;
96. private JTextArea textArea;
97. private SimulatedActivity activity;
98.
99. public static final int DEFAULT_WIDTH = 300;
100. public static final int DEFAULT_HEIGHT = 200;
101.
102. class SimulatedActivity extends SwingWorker<Void, Integer>
103. {
104.     /**
105.      * Constructs the simulated activity that increments a counter from 0 to a
106.      * given target.
107.      * @param t the target value of the counter.
108.      */
109.     public SimulatedActivity(int t)
110.     {
111.         current = 0;
112.         target = t;
113.     }
114.
115.     protected Void doInBackground() throws Exception
116.     {
117.         try
118.         {
119.             while (current < target)
120.             {
121.                 Thread.sleep(100);
122.                 current++;
123.                 textArea.append(current + "\n");
124.                 setProgress(current);
125.             }
126.         }
127.         catch (InterruptedException e)
128.         {
129.         }
130.         return null;
131.     }
132.
133.     private int current;
```

```
134.     private int target;
135. }
136. }
```

### 6.5.3 监视输入流的进度

Swing类包有一个很有用的流过滤器，`ProgressMonitorInputStream`，它可以自动弹出一个对话框，监视已经读取了多少流。

这个过滤器很容易使用。可以在你要过滤的普通的流序列之间插入`ProgressMonitorInputStream`。（请参阅第1章关于流的更多详细细节。）

例如，假定你现在要从一个文件中读取文本。首先要使用一个`FileInputStream`：

```
FileInputStream in = new FileInputStream(f);
```

通常情况下，要将`in`转换成一个`InputStreamReader`：

```
InputStreamReader reader = new InputStreamReader(in);
```

不过，为了监视这个流，首先要将这个文件输入流转换成一个具有进度监视器的数据流：

```
ProgressMonitorInputStream progressIn = new ProgressMonitorInputStream(parent, caption, in);
```

你要提供一个父构件，一个标题，当然还有要监视的流。进度监视器流的`read`方法只能传输字节和更新进度对话框。

现在可以开始着手构建你的过滤序列：

```
InputStreamReader reader = new InputStreamReader(progressIn);
```

这就是我们要做的全部内容。当读取文件的时候，进度监视器会自动弹出（参见图6-43）。这是一个流过滤的极佳应用。



**警告：**进度监视器流使用`InputStream`类的`available`方法来确定流中的总字节数。不过，`available`方法只报告流中可访问的、未中断的字节数。进度监视器适用于文件以及HTTP URL，因为它们的长度都是事先可以知道的，但它并不适用于所有的流。

程序清单6-18中的程序可以计算文件中的行数。如果读取的是一个大型文件（例如附带的代码中的`gutenberg`目录中的“*The Count of Monte Cristo*”），那么将会弹出进度对话框。

如果用户点击`Cancel`按钮，输入流就会关闭。因为处理输入的代码已经知道了应该如何处理输入结束，所以处理取消请求并不需要对编程逻辑做任何修改。

注意：该程序并没有使用很高效的方式来填充文本区域。如果首先将文件读取到一个`StringBuffer`中，然后将文本区域的文本设置为字符串缓冲的内容，可能会更快一些。不过，在这个示例程序中，我们实际上喜欢这种缓慢的方式，因为它可以让你有更多的时间欣赏进度对话框。

为了避免闪烁，我们并不显示正在进行填充的文本区域。

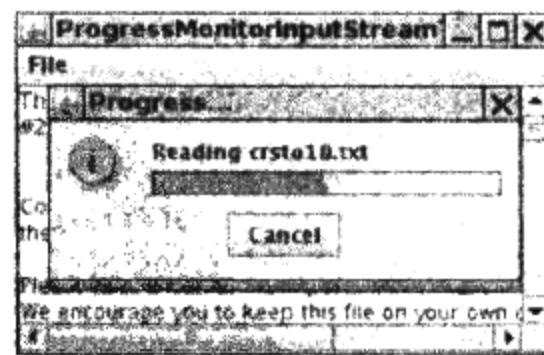


图6-43 用于输入流的进度监视器

**程序清单6-18 · ProgressMonitorInputStreamTest.java**

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.io.*;
4. import java.util.*;
5. import javax.swing.*;
6.
7. /**
8. * A program to test a progress monitor input stream.
9. * @version 1.04 2007-08-01
10. * @author Cay Horstmann
11. */
12. public class ProgressMonitorInputStreamTest
13. {
14.     public static void main(String[] args)
15.     {
16.         EventQueue.invokeLater(new Runnable()
17.         {
18.             public void run()
19.             {
20.                 JFrame frame = new TextFrame();
21.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22.                 frame.setVisible(true);
23.             }
24.         });
25.     }
26. }
27.
28. /**
29. * A frame with a menu to load a text file and a text area to display its contents. The text
30. * area is constructed when the file is loaded and set as the content pane of the frame when
31. * the loading is complete. That avoids flicker during loading.
32. */
33. class TextFrame extends JFrame
34. {
35.     public TextFrame()
36.     {
37.         setTitle("ProgressMonitorInputStreamTest");
38.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
39.
40.         textArea = new JTextArea();
41.         add(new JScrollPane(textArea));
42.
43.         chooser = new JFileChooser();
44.         chooser.setCurrentDirectory(new File("."));
45.
46.         JMenuBar menuBar = new JMenuBar();
47.         setJMenuBar(menuBar);
48.         JMenu fileMenu = new JMenu("File");
49.         menuBar.add(fileMenu);
50.         openItem = new JMenuItem("Open");
51.         openItem.addActionListener(new ActionListener()
52.         {
53.             public void actionPerformed(ActionEvent event)
54.             {
55.                 try
```

```
56.        {
57.            openFile();
58.        }
59.        catch (IOException exception)
60.        {
61.            exception.printStackTrace();
62.        }
63.    }
64.);
65.
66. fileMenu.add(openItem);
67. exitItem = new JMenuItem("Exit");
68. exitItem.addActionListener(new ActionListener()
69. {
70.     public void actionPerformed(ActionEvent event)
71.     {
72.         System.exit(0);
73.     }
74. });
75. fileMenu.add(exitItem);
76. }
77.
78. /**
79. * Prompts the user to select a file, loads the file into a text area, and sets it as
80. * the content pane of the frame.
81. */
82. public void openFile() throws IOException
83. {
84.     int r = chooser.showOpenDialog(this);
85.     if (r != JFileChooser.APPROVE_OPTION) return;
86.     final File f = chooser.getSelectedFile();
87.
88.     // set up stream and reader filter sequence
89.
90.     FileInputStream fileIn = new FileInputStream(f);
91.     ProgressMonitorInputStream progressIn = new ProgressMonitorInputStream(this,
92.         "Reading " + f.getName(), fileIn);
93.     final Scanner in = new Scanner(progressIn);
94.
95.     textArea.setText("");
96.
97.     SwingWorker<Void, Void> worker = new SwingWorker<Void, Void>()
98.     {
99.         protected Void doInBackground() throws Exception
100.         {
101.             while (in.hasNextLine())
102.             {
103.                 String line = in.nextLine();
104.                 textArea.append(line);
105.                 textArea.append("\n");
106.             }
107.             in.close();
108.             return null;
109.         }
110.     };
111.     worker.execute();
112. }
```

```
113.  
114.     private JMenuItem openItem;  
115.     private JMenuItem exitItem;  
116.     private JTextArea textArea;  
117.     private JFileChooser chooser;  
118.  
119.     public static final int DEFAULT_WIDTH = 300;  
120.     public static final int DEFAULT_HEIGHT = 200;  
121. }
```



### javax.swing.JProgressBar 1.2

- `JProgressBar()`
- `JProgressBar(int direction)`
- `JProgressBar(int min, int max)`
- `JProgressBar(int direction, int min, int max)`

按照给定的方向、最小值以及最大值构建一个滑动条。

参数: `direction` `SwingConstants.HORIZONTAL`或者`SwingConstants.VERTICAL`其中之一。默认值是水平方向  
`min, max` 进度条的最大值和最小值。默认值是0和100

- `int getMinimum()`
- `int getMaximum()`
- `void setMinimum(int value)`
- `void setMaximum(int value)`

获取并设置最小值以及最大值。

- `int getValue()`
- `void setValue(int value)`

获取并设置当前的值。

- `String getString()`
- `void setString(String s)`

获取并设置在进度条中显示的字符串。如果该字符串为`null`, 那么将会显示一个默认字符串“n%”。

- `boolean isStringPainted()`
- `void setStringPainted(boolean b)`

获取并设置“字符串绘制”属性。如果这个属性是`true`, 那么会在进度条的上面绘制出一个字符串。默认值是`false`; 不绘制任何字符串。

- `boolean isIndeterminate() 1.4`
- `void setIndeterminate(boolean b) 1.4`

获取并设置“不确定”属性。如果该属性是`true`, 那么该进度条就会变成一个前后移动的滑动块, 表明一个持续时间不可知的等待。默认值是`false`。

**javax.swing.ProgressMonitor 1.2**

- `ProgressMonitor(Component parent, Object message, String note, int min, int max)`  
构建一个进度监视器对话框。  
参数：  
    **parent** 父构件，在其上弹出对话框  
    **message** 对话框中要显示的消息对象  
    **note** 在消息下显示的可选字符串。如果该值为null，则不会为注释设置任何空间，并且随后对setNote的调用不会产生任何效果  
    **min, max** 进度条的最小值以及最大值
- `void setNote(String note)`  
更改注释文本。
- `void setProgress(int value)`  
将进度条的值设置为给定值。
- `void close()`  
关闭对话框。
- `boolean isCanceled()`  
如果用户取消了对话框，则返回true。

**javax.swing.ProgressMonitorInputStream 1.2**

- `ProgressMonitorInputStream(Component parent, Object message, InputStream in)`  
参数：  
    **parent** 父构件，在其上弹出对话框  
    **message** 在对话框中显示的消息对象  
    **in** 正被监视的输入流

## 6.6 构件组织器

我们在这里通过展示一些帮助组织其他构件的构件来结束对高级Swing特性的讨论。这些构件包括分割面板、选项卡面板以及桌面面板。分割面板是将一个区域分割成多个边界可调整的区域的一种机制。选项卡面板使用选项卡分割器，允许用户浏览多个面板。桌面面板可用来实现显示多个内部框体的应用。

### 6.6.1 分割面板

分割面板可以将一个构件分割成两部分，并且这两部分之间具有可调整的边界。图6-44显示了一个具有两个分割面板的框体。外部面板中的构件是垂直布局的，底部是一个文本区，上面是另外一个分割面板。上面这个分割面板是水平分割的，左边是一个列表，右边是一个包含图形的标签。

如果要构建一个分割面板，需要设定一个方向，其值为JSplitPane.HORIZONTAL\_SPLIT和JSplitPane.VERTICAL\_SPLIT中的之一，随后是两个构件。例如：

```
JSplitPane innerPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, planetList, planetImage);
```

这就是你要做的全部事情。如果你喜欢，你可以为分割器添加“一触即展”的图标。你可以在图6-44中的顶层面板中看到这些图标。在Metal外观模式中，它们是小箭头的形式。如果你点中它们中的一个，那么分割器将会一直沿着箭头指定的方向移动，将其中的一个面板完全展开。

如果要添加这项功能，请调用：

```
innerPane.setOneTouchExpandable(true);
```

当用户调整分割器的时候，“连续布局”的特性会一直不断地刷新这两个构件的内容。这种情形看似经典，实则运行缓慢。你可以调用下面这个方法启动该功能：

```
innerPane.setContinuousLayout(true);
```

在这个示例程序中，我们将分割器设为默认状态（非连续布局）。拖动它的时候，只能移动一个黑色的轮廓。当释放鼠标完成这项操作时，才会刷新这些构件。

在简单明了的程序清单6-19中，安装了一个具有行星数据的列表框。当用户进行一次选择的时候，行星的图片便在右边显示了出来，并且在底部的文本区显示出对它的描述。当你运行这个程序的时候，请调整一下分割器，并试试一触即展和连续布局这些特性。

### 程序清单6-19 SplitPaneTest.java

```

1. import java.awt.*;
2.
3. import javax.swing.*;
4. import javax.swing.event.*;
5.
6. /**
7. * This program demonstrates the split pane component organizer.
8. * @version 1.03 2007-08-01
9. * @author Cay Horstmann
10.*/
11. public class SplitPaneTest
12. {
13.     public static void main(String[] args)
14.     {
15.         EventQueue.invokeLater(new Runnable()
16.         {
17.             public void run()
18.             {
19.                 JFrame frame = new SplitPaneFrame();
20.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21.                 frame.setVisible(true);
22.             }
23.         });
24.     }
25. }
```

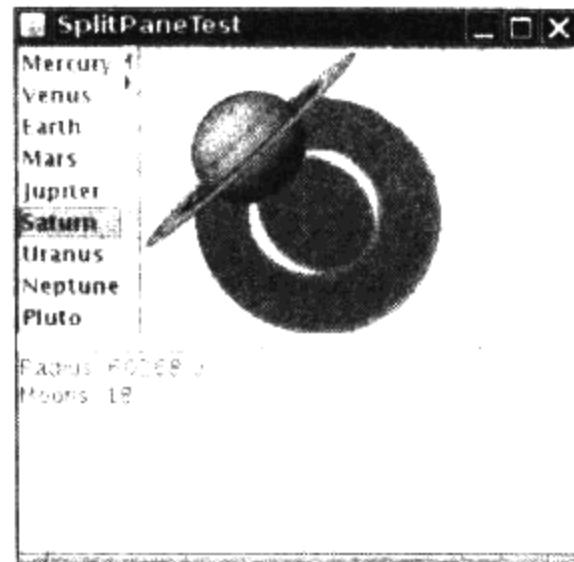


图6-44 具有两个嵌套的分割面板的框架

```
27. /**
28. * This frame consists of two nested split panes to demonstrate planet images and data.
29. */
30. class SplitPaneFrame extends JFrame
31. {
32.     public SplitPaneFrame()
33.     {
34.         setTitle("SplitPaneTest");
35.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
36.
37.         // set up components for planet names, images, descriptions
38.
39.         final JList planetList = new JList(planets);
40.         final JLabel planetImage = new JLabel();
41.         final JTextArea planetDescription = new JTextArea();
42.
43.         planetList.addListSelectionListener(new ListSelectionListener()
44.         {
45.             public void valueChanged(ListSelectionEvent event)
46.             {
47.                 Planet value = (Planet) planetList.getSelectedValue();
48.
49.                 // update image and description
50.
51.                 planetImage.setIcon(value.getImage());
52.                 planetDescription.setText(value.getDescription());
53.             }
54.         });
55.
56.         // set up split panes
57.
58.         JSplitPane innerPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, planetList,
59.             planetImage);
60.
61.         innerPane.setContinuousLayout(true);
62.         innerPane.setOneTouchExpandable(true);
63.
64.         JSplitPane outerPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT, innerPane,
65.             planetDescription);
66.
67.         add(outerPane, BorderLayout.CENTER);
68.     }
69.
70.     private Planet[] planets = { new Planet("Mercury", 2440, 0), new Planet("Venus", 6052, 0),
71.         new Planet("Earth", 6378, 1), new Planet("Mars", 3397, 2),
72.         new Planet("Jupiter", 71492, 16), new Planet("Saturn", 60268, 18),
73.         new Planet("Uranus", 25559, 17), new Planet("Neptune", 24766, 8),
74.         new Planet("Pluto", 1137, 1), };
75.     private static final int DEFAULT_WIDTH = 300;
76.     private static final int DEFAULT_HEIGHT = 300;
77. }
78.
79. /**
80. * Describes a planet.
81. */
82. class Planet
83. {
```

```
84.  /**
85.   * Constructs a planet.
86.   * @param n the planet name
87.   * @param r the planet radius
88.   * @param m the number of moons
89.   */
90.  public Planet(String n, double r, int m)
91.  {
92.      name = n;
93.      radius = r;
94.      moons = m;
95.      image = new ImageIcon(name + ".gif");
96.  }
97.
98.  public String toString()
99.  {
100.     return name;
101. }
102.
103. /**
104.  * Gets a description of the planet.
105.  * @return the description
106.  */
107. public String getDescription()
108. {
109.     return "Radius: " + radius + "\nMoons: " + moons + "\n";
110. }
111.
112. /**
113.  * Gets an image of the planet.
114.  * @return the image
115.  */
116. public ImageIcon getImage()
117. {
118.     return image;
119. }
120.
121. private String name;
122. private double radius;
123. private int moons;
124. private ImageIcon image;
125. }
```

### API javax.swing.JSplitPane 1.2

- JSplitPane()
- JSplitPane(int direction)
- JSplitPane(int direction, boolean continuousLayout)
- JSplitPane(int direction, Component first, Component second)
- JSplitPane(int direction, boolean continuousLayout, Component first, Component second)

构建一个新的分割面板。

参数: direction HORIZONTAL\_SPLIT或VERTICAL\_SPLIT  
continuousLayout 如果为true, 那么当移动分割器时, 该构件是连续更新的  
first, second 要添加的构件

- boolean isOneTouchExpandable()
- void setOneTouchExpandable(boolean b)

获取并设置“一触即展”属性。如果设置了该属性, 那么该分割器具有两个图标以完全展开一个或另一个构件。

- boolean isContinuousLayout()
- void setContinuousLayout(boolean b)

获取并设置“连续布局”属性。如果设置了该属性, 那么当移动分割器的时候, 该构件是连续更新的。

- void setLeftComponent(Component c)
- void setTopComponent(Component c)

这两个操作具有同等效果, 用于将c设置为分割面板中第一个构件。

- void setRightComponent(Component c)
- void setBottomComponent(Component c)

这两个操作具有同等效果, 用于将c设置为分割面板中第二个构件。

### 6.6.2 选项卡面板

选项卡面板是一种大家都很熟悉的用户界面设施, 可以将一个复杂的对话框分割成相关选项的子集。也可以使用选项卡让用户浏览一组文档或图像(参见图6-45)。这也是我们在示例程序中要讲解的。

为了创建一个选项卡面板, 首先要构建一个JTabbedPane对象, 然后向其中添加选项卡。

```
JTabbedPane tabbedPane = new JTabbedPane();
tabbedPane.addTab(title, icon, component);
```

addTab方法最后一个参数的类型为Component。为了向同一个选项卡中添加多了构件, 首先要将这些构件包装到一个容器中, 例如一个 JPanel。

该方法中的图标参数是一个可选项。addTab方法并非一定要有一个图标参数, 例如:

```
tabbedPane.addTab(title, component);
```

也可以使用insertTab方法, 将一个选项卡添加到选项卡集中:

```
tabbedPane.insertTab(title, icon, component, tooltip, index);
```

如果要从选项卡集中删掉一个选项卡, 请使用

```
tabbedPane.removeTabAt(index);
```

向选项集中添加一个新的选项卡时, 并不能自动将其显示出来。必须使用setSelectedIndex方法选定它。例如, 下面这段代码展示了怎样将刚刚添加到末尾的选项卡显示出来:

```
tabbedPane.setSelectedIndex(tabbedPane.getTabCount() - 1);
```

如果有很多选项卡, 那么它们会占用很多空间。从Java SE 1.4开始, 可以将选项卡以滚动

模式显示出来，在这种模式中，只显示一行面板，不过会配有一组箭头允许用户滚动显示这些选项卡（参见图6-46）。

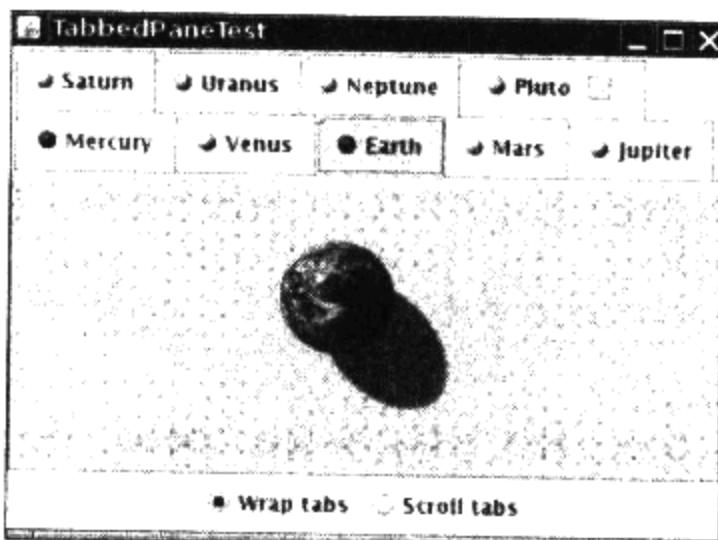


图6-45 一个选项卡面板

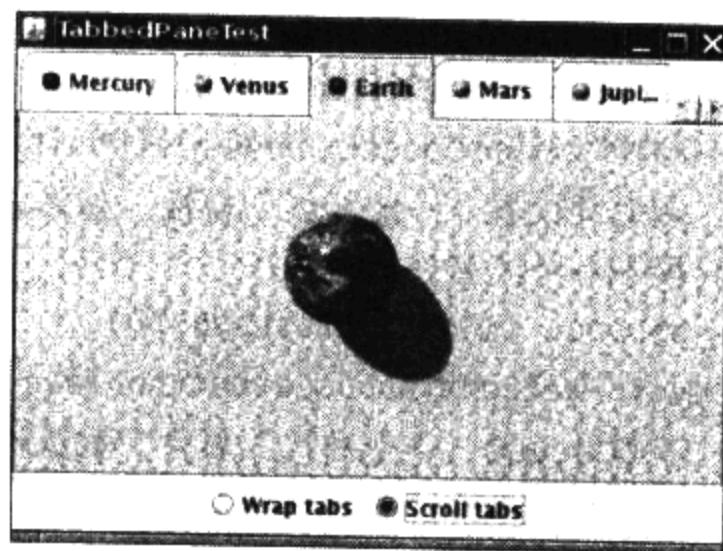


图6-46 具有滚动选项卡的选项卡面板

调用下面这个方法，就可以将选项卡布局设置为隐藏格式或者滚动模式：

```
tabbedPane.setTabLayoutPolicy(JTabbedPane.WRAP_TAB_LAYOUT);
```

或者

```
tabbedPane.setTabLayoutPolicy(JTabbedPane.SCROLL_TAB_LAYOUT);
```

选项卡标签可以有快捷键，就像菜单项一样。例如：

```
int marsIndex = tabbedPane.indexOfTab("Mars");
tabbedPane.setMnemonicAt(marsIndex, KeyEvent.VK_M);
```

之后M就会有下划线，而程序用户可以通过键入ALT+M来选择选项卡。

从Java SE 6开始，可以在选项卡标题栏中添加任何构件。首先添加选项卡，然后调用：

```
tabbedPane.setTabComponentAt(index, component);
```

在我们的示例程序中，我们向Pluto选项卡中添加了一个“关闭框”（因为毕竟有些天文学家不认为冥王星算得上一颗真正的行星）。实现这项任务的方法是将选项卡构件设置为包含两个构件的面板：具有图标和选项卡文本的标签，以及具有能够移除该选项卡的动作监听器的复选框。

这个示例程序展示了选项卡面板一个非常有用的技术。有时候需要在一个构件显示之前对其进行更新。在我们这个示例程序中，只在用户真正点击一个选项卡的时候才将行星图片载入。

为了在用户任何时候点击一个新选项卡时都能获得通知，需要为选项卡面板安装一个ChangeListener。注意，必须为选项卡面板本身添加监听器，而不是它所包含的任何一个选项卡构件。

```
tabbedPane.addChangeListener(listener);
```

当用户选定一个选项卡时，就会调用修改监听器的stateChanged方法。可以将选项卡面板作为事件源来读取，并调用getSelectedIndex方法就可以查明将要显示哪个面板。

```
public void stateChanged(ChangeEvent event)
{
    int n = tabbedPane.getSelectedIndex();
```

```
    loadTab(n);
}
```

在程序清单6-20中，我们首先将选项卡构件设置为null。当选定一个新的面板时，我们会测试它的构件是否仍为null。如果为null，我们会用一个图片替代显示。（这种情况在点击一个选项卡的那一瞬间发生。你将不会看到任何空的面板。）只是为了有趣，我们还将这个图标从黄色球更改为红色球以指示我们已经访问过的那些面板。

### 程序清单6-20 TabbedPaneTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3.
4. import javax.swing.*;
5. import javax.swing.event.*;
6.
7. /**
8. * This program demonstrates the tabbed pane component organizer.
9. * @version 1.03 2007-08-01
10. * @author Cay Horstmann
11. */
12. public class TabbedPaneTest
13. {
14.     public static void main(String[] args)
15.     {
16.         EventQueue.invokeLater(new Runnable()
17.         {
18.             public void run()
19.             {
20.
21.                 JFrame frame = new TabbedPaneFrame();
22.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23.                 frame.setVisible(true);
24.             }
25.         });
26.     }
27. }
28.
29. /**
30. * This frame shows a tabbed pane and radio buttons to switch between wrapped and scrolling
31. * tab layout.
32. */
33. class TabbedPaneFrame extends JFrame
34. {
35.     public TabbedPaneFrame()
36.     {
37.         setTitle("TabbedPaneTest");
38.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
39.
40.         tabbedPane = new JTabbedPane();
41.         // we set the components to null and delay their loading until the tab is shown
42.         // for the first time
43.
44.         ImageIcon icon = new ImageIcon("yellow-ball.gif");
45.
```

```
46.     tabbedPane.addTab("Mercury", icon, null);
47.     tabbedPane.addTab("Venus", icon, null);
48.     tabbedPane.addTab("Earth", icon, null);
49.     tabbedPane.addTab("Mars", icon, null);
50.     tabbedPane.addTab("Jupiter", icon, null);
51.     tabbedPane.addTab("Saturn", icon, null);
52.     tabbedPane.addTab("Uranus", icon, null);
53.     tabbedPane.addTab("Neptune", icon, null);
54.     tabbedPane.addTab("Pluto", null, null);
55.
56.     final int plutoIndex = tabbedPane.indexOfTab("Pluto");
57.     JPanel plutoPanel = new JPanel();
58.     plutoPanel.add(new JLabel("Pluto", icon, SwingConstants.LEADING));
59.     JToggleButton plutoCheckBox = new JCheckBox();
60.     plutoCheckBox.addActionListener(new ActionListener()
61.     {
62.         public void actionPerformed(ActionEvent e)
63.         {
64.             tabbedPane.remove(plutoIndex);
65.         }
66.     });
67.     plutoPanel.add(plutoCheckBox);
68.     tabbedPane.setTabComponentAt(plutoIndex, plutoPanel);
69.
70.     add(tabbedPane, "Center");
71.
72.     tabbedPane.addChangeListener(new ChangeListener()
73.     {
74.         public void stateChanged(ChangeEvent event)
75.         {
76.
77.             // check if this tab still has a null component
78.
79.             if (tabbedPane.getSelectedComponent() == null)
80.             {
81.                 // set the component to the image icon
82.
83.                 int n = tabbedPane.getSelectedIndex();
84.                 loadTab(n);
85.             }
86.         }
87.     });
88.
89.     loadTab(0);
90.
91.     JPanel buttonPanel = new JPanel();
92.     ButtonGroup buttonGroup = new ButtonGroup();
93.     JRadioButton wrapButton = new JRadioButton("Wrap tabs");
94.     wrapButton.addActionListener(new ActionListener()
95.     {
96.         public void actionPerformed(ActionEvent event)
97.         {
98.             tabbedPane.setTabLayoutPolicy(JTabbedPane.WRAP_TAB_LAYOUT);
99.         }
100.    });
101.   buttonPanel.add(wrapButton);
```

```
102     buttonGroup.add(wrapButton);
103     wrapButton.setSelected(true);
104     JRadioButton scrollButton = new JRadioButton("Scroll tabs");
105     scrollButton.addActionListener(new ActionListener()
106     {
107         public void actionPerformed(ActionEvent event)
108         {
109             tabbedPane.setTabLayoutPolicy(JTabbedPane.SCROLL_TAB_LAYOUT);
110         }
111     });
112     buttonPanel.add(scrollButton);
113     buttonGroup.add(scrollButton);
114     add(buttonPanel, BorderLayout.SOUTH);
115 }
116
117 /**
118 * Loads the tab with the given index.
119 * @param n the index of the tab to load
120 */
121 private void loadTab(int n)
122 {
123     String title = tabbedPane.getTitleAt(n);
124     ImageIcon planetIcon = new ImageIcon(title + ".gif");
125     tabbedPane.setComponentAt(n, new JLabel(planetIcon));
126
127     // indicate that this tab has been visited--just for fun
128
129     tabbedPane.setIconAt(n, new ImageIcon("red-ball.gif"));
130 }
131
132 private JTabbedPane tabbedPane;
133
134 private static final int DEFAULT_WIDTH = 400;
135 private static final int DEFAULT_HEIGHT = 300;
136 }
```



## javax.swing.JTabbedPane 1.2

- `JTabbedPane()`
- `JTabbedPane(int placement)`

构建一个选项卡面板。

参数: `placement` `SwingConstants.TOP`、`SwingConstants.LEFT`、  
`SwingConstants.RIGHT`或`SwingConstants.BOTTOM`其中之一

- `void addTab(String title, Component c)`
- `void addTab(String title, Icon icon, Component c)`
- `void addTab(String title, Icon icon, Component c, String tooltip)`  
向选项卡面板的末尾添加一个选项卡。
- `void insertTab(String title, Icon icon, Component c, String tooltip, int index)`  
在选项卡面板的给定索引处添加一个选项卡。
- `void removeTabAt(int index)`

移除指定索引处的选项卡。

- `void setSelectedIndex(int index)`

选定给定索引处的选项卡。

- `void getSelectedIndex()`

获取选定的选项卡的索引。

- `Component getSelectedComponent()`

返回选定的选项卡构件。

- `String getTitleAt(int index)`

- `void setTitleAt(int index, String title)`

- `Icon getIconAt(int index)`

- `void setIconAt(int index, Icon icon)`

- `Component getComponentAt(int index)`

- `void setComponentAt(int index, Component c)`

获取或设置给定索引处的标题、图标或者构件。

- `int indexOfTab(String title)`

- `int indexOfTab(Icon icon)`

- `int indexOfComponent(Component c)`

返回具有给定的标题、图标或者构件的索引。

- `int getTabCount()`

返回该选项卡面板上的选项卡总数。

- `int getTabLayoutPolicy()`

- `void setTabLayoutPolicy(int policy)` 1.4

获得或者设置选项卡布局策略。policy是`JTabbedPane.WRAP_TAB_LAYOUT`或`JTabbedPane.SCROLL_TAB_LAYOUT`其中之一。

- `int getMnemonicAt(int index)` 1.4

- `void setMnemonicAt(int index, int mnemonic)`

获得或者设置给定选项卡索引的快捷字符。这个字符是作为`KeyEvent`类的一个`VK_X`常量指定的，-1表示没有快捷方式。

- `Component getTabComponentAt(int index)` 6

- `void setTabComponentAt(int index, Component c)` 6

获得或者设置构件，用于绘制给定索引的选项卡的标题栏。如果该构件为`null`，则绘制选项卡的图标和标题，否则，在选项卡中只绘制给定的构件。

- `int indexOfTabComponent(Component c)` 6

返回具有给定标题栏构件的选项卡的索引。

- `void addChangeListener(ChangeListener listener)`

添加一个修改监听器，当用户选定了另一个选项卡的时候，会通知它。

### 6.6.3 桌面面板和内部框体

很多应用会将信息在多个窗口中显示，并且这些窗口都包含在一个大的框体中。如果将应用框体最小化，那么它当中的所有窗口会在同一时间全部隐藏起来。在Windows环境中，这种用户界面有时称作多文档界面（multiple document interface，MDI）。图6-47显示了一个使用到该界面的传统应用程序。

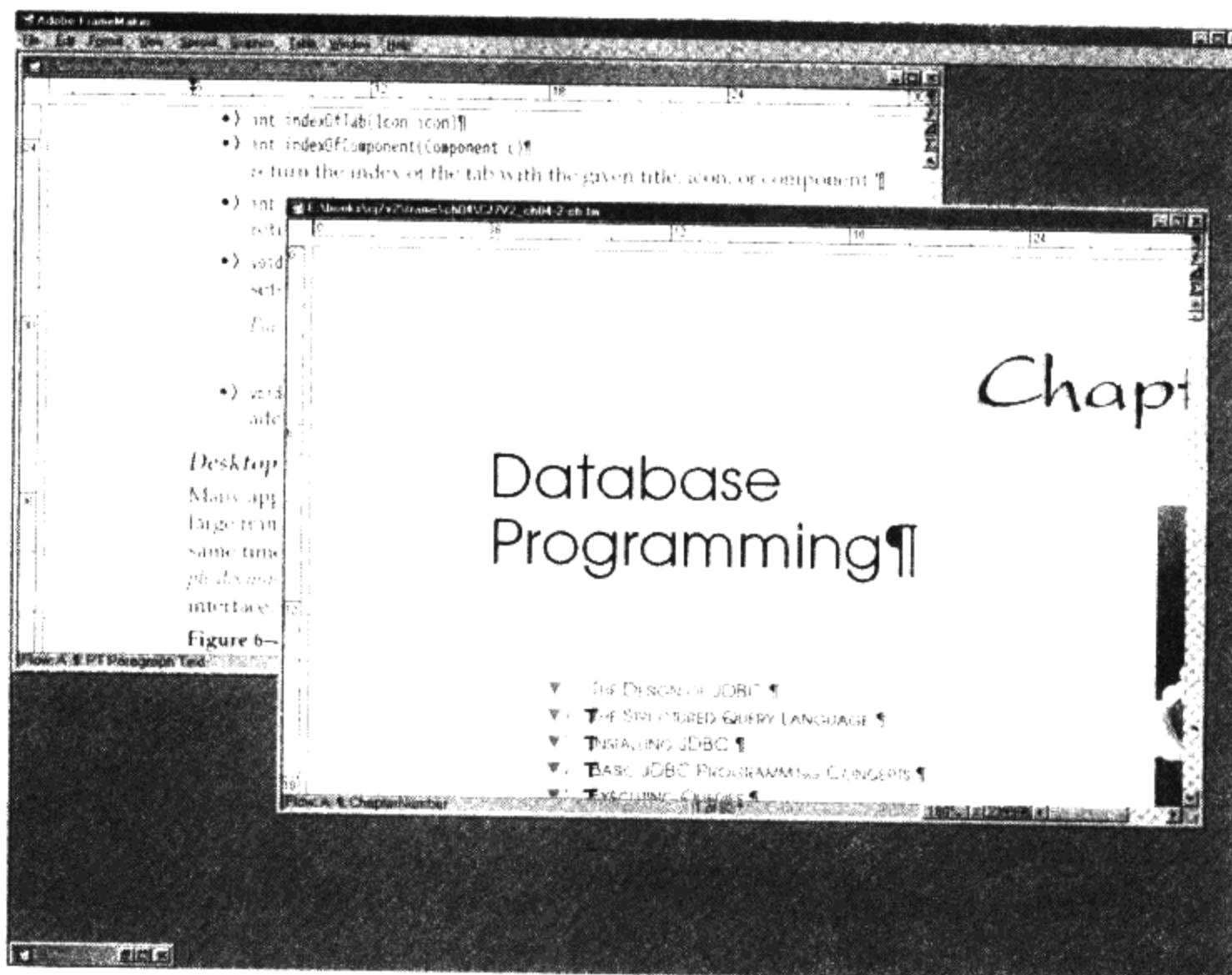


图6-47 一个多文档界面的应用

有一段时间，这种用户界面格式非常流行，不过最近几年已经变得不那么常用了。现在，很多应用为每个文档只显示一个独立的顶层框体。哪一种格式更好呢？MDI减少了窗口的混乱，但是如果拥有了独立的顶层窗口，意味着可以使用主窗口系统的按钮及热键浏览所有窗口。

在Java环境中，不能依赖于一个丰富的主机窗口系统，让你的应用管理它自己的框体还是很有必要的。

图6-48显示了一个具有三个内部框体的Java应用程序。其中的两个有一些边框装饰，用于对它们进行最大化和图标显示。第三个已经处于图标状态。

在Metal外观模式中，内部框体具有独一无二的“grabber”区域，可以让你随意移动这些框体，并可以通过拖动调整大小的角来更改窗口的大小。

为了实现这项功能，请遵循下面几步：

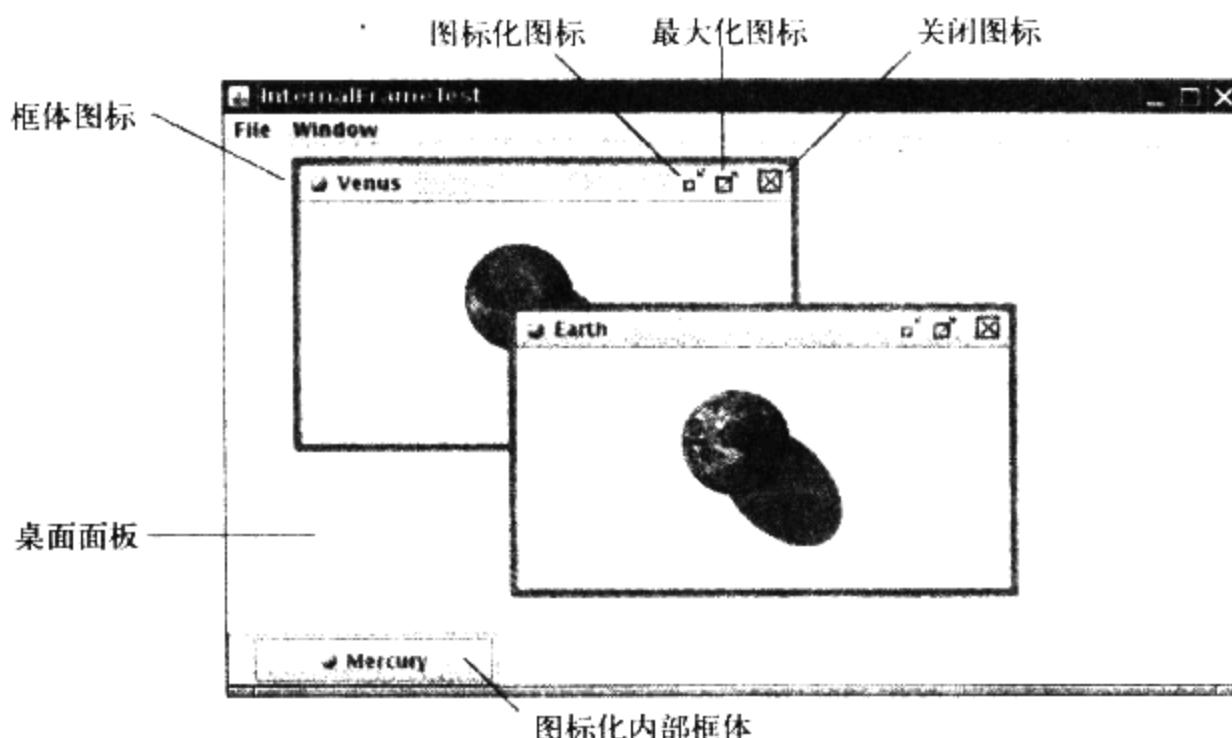


图6-48 具有三个内部框体的Java应用程序

1. 在该应用中使用常规的JFrame。
2. 向该JFrame添加JDesktopPane。

```
desktop = new JDesktopPane();
add(desktop, BorderLayout.CENTER);
```

3. 构建JInternalFrame窗口，可以设定是要更改框体的大小还是要关闭框体。通常情况下，需要添加所有的图标。

```
JInternalFrame iframe = new JInternalFrame(title,
    true, // resizable
    true, // closable
    true, // maximizable
    true); // iconifiable
```

4. 向该内部框体中添加构件。

```
iframe.add(c, BorderLayout.CENTER);
```

5. 设置该内部框体的图标。该图标会显示在框体左上角。

```
iframe setFrameIcon(icon);
```

注意：在Metal当前版本中，框体图标并不在图标化的框体中显示出来。

6. 设置内部框体的大小。和常规框体一样，内部框体初始大小为0×0个像素。因为你并不希望内部框体在另一个框体上面显示出来，因此，应该为下一个框体使用一个可变位置。使用reshape方法对框体的位置和大小进行设置：

```
iframe.reshape(nextFrameX, nextFrameY, width, height);
```

7. 和JFrames一样，需要将该框体设为可见的。

```
iframe.setVisible(true);
```

注意：在Swing的早期版本，内部框体自动是可见的，因此就不需要调用这个方法。

8. 将该框体添加到JDesktopPane:

```
desktop.add(iframe);
```

9. 你可能想使新的框体成为选定框体。对于桌面上的内部框体，只有选定的框体才能接收键盘焦点。在Metal外观模式中，选定框体具有蓝色标题栏，相反地，其他框体是灰色标题栏。可以使用setSelected方法选定一个框体。不过，这种“选定”属性可能会被否决掉，当前选定的框体可以拒绝放弃焦点。在这种情况下，setSelected方法会抛出一个PropertyVetoException异常让你处理。

```
try
{
    iframe.setSelected(true);
}
catch (PropertyVetoException e)
{
    // attempt was vetoed
}
```

10. 你可能希望下一个内部框体的位置能够向下移动，不至于覆盖已经存在的框体。框体之间的合适距离是标题栏的高度，可以通过下面的方式获得。

```
int frameDistance = iframe.getHeight() - iframe.getContentPane().getHeight()
```

11. 使用该距离确定下一个内部框体的位置。

```
nextFrameX += frameDistance;
nextFrameY += frameDistance;
if (nextFrameX + width > desktop.getWidth())
    nextFrameX = 0;
if (nextFrameY + height > desktop.getHeight())
    nextFrameY = 0;
```

#### 6.6.4 级联与平铺

在Windows环境中，有一些用于级联及平铺窗口的标准命令（参见图6-49及图6-50）。Java语言的JDesktopPane类和JInternalFrame类对这些操作未提供任何内置支持。在程序清单6-21中，我们将展示如何实现这些操作。

为了级联所有的窗口，可以将这些窗口重新绘制成同样的大小，并交错排列它们的位置。JDesktopPane类的getAllFrames方法可以返回一个所有内部框体的数组。

```
JInternalFrame[] frames = desktop.getAllFrames();
```

不过，要注意一下框体的状态。一个内部框体可以具有下面三种状态之一：

- 图标
- 可放缩
- 最大化

可以使用isIcon方法确定哪些内部框体当前是处于图标状态，因而应该跳过。不过，如果一个框体处于最大状态，那么首先要通过调用setMaximum(false)方法将它设置为可放缩状态。这是另外一个可能被否决掉的属性，因此你必须捕获PropertyVetoException异常。

下面这个循环用于级联一个桌面上的所有内部框体：

```
for (JInternalFrame frame : desktop.getAllFrames())
{
```

```

if (!frame.isIcon())
{
    try
    {
        // try to make maximized frames resizable; this might be vetoed
        frame.setMaximum(false);
        frame.reshape(x, y, width, height);
        x += frameDistance;
        y += frameDistance;
        // wrap around at the desktop edge
        if (x + width > desktop.getWidth()) x = 0;
        if (y + height > desktop.getHeight()) y = 0;
    }
    catch (PropertyVetoException e)
    {}
}
}

```

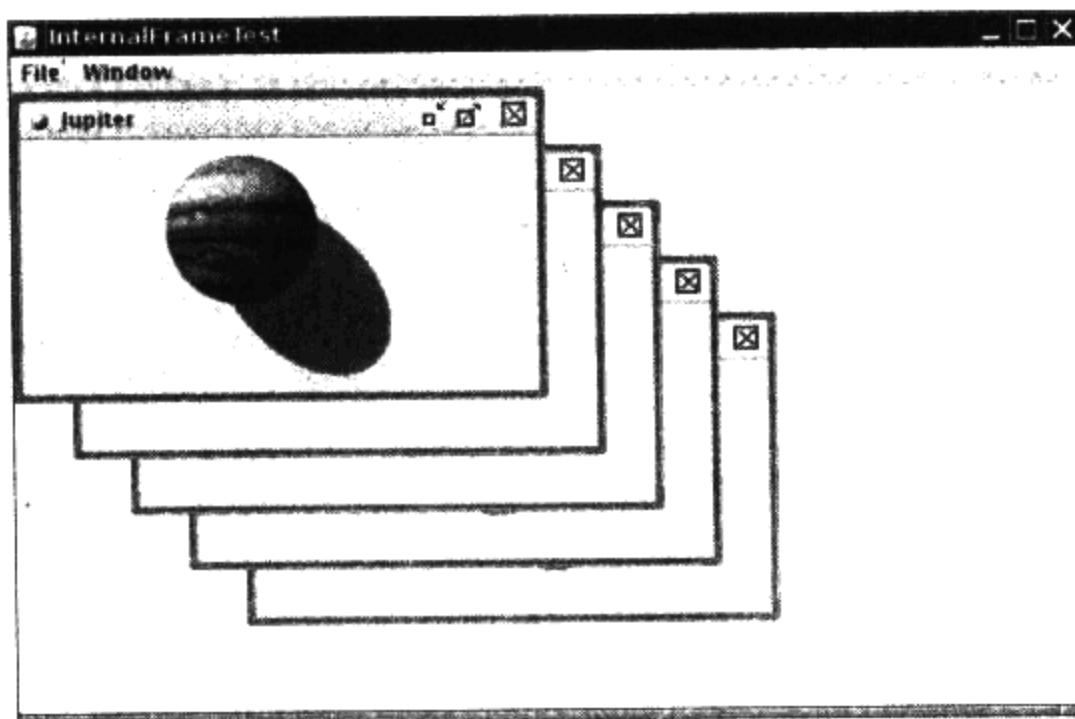


图6-49 级联的内部框体

平铺框体更具技巧性，尤其是当框体数不是一个完全平方数时。首先，计算出不是图标的框体数。然后，按照下面的方法计算行数：

```
int rows = (int) Math.sqrt(frameCount);
```

然后是计算列数：

```
int cols = frameCount / rows;
```

除了最后一列是：

```
int extra = frameCount % rows
```

其余每列的行数是rows + 1。

下面这个循环用于平铺桌面上的所有内部框体：

```

int width = desktop.getWidth() / cols;
int height = desktop.getHeight() / rows;
int r = 0;
int c = 0;

```

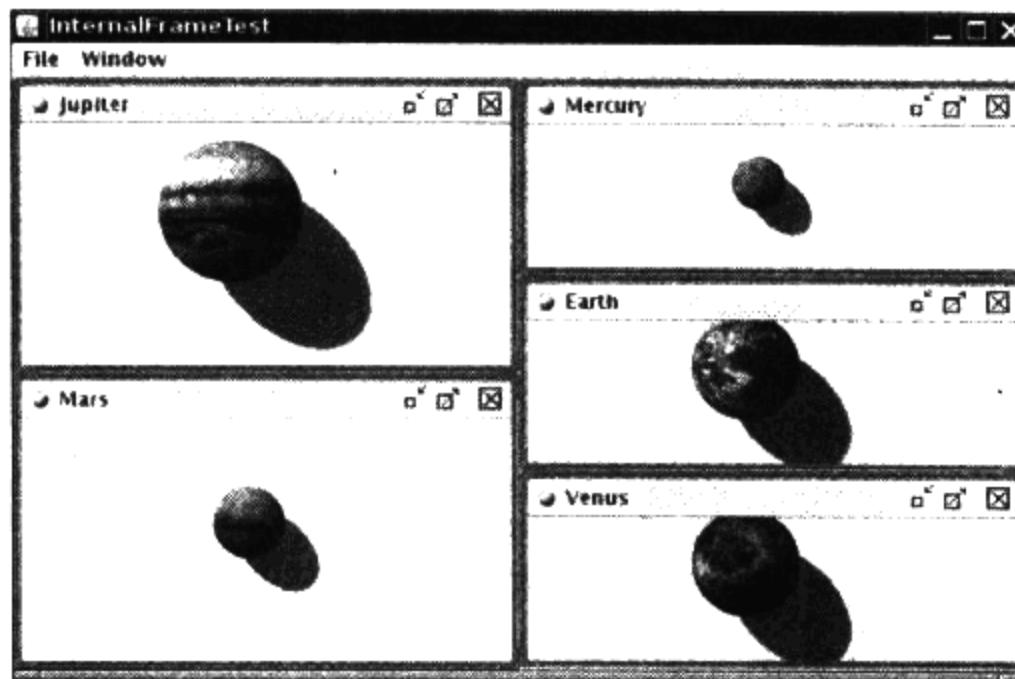


图6-50 平铺的内部框体

```
for (JInternalFrame frame : desktop.getAllFrames())
{
    if (!frame.isIcon())
    {
        try
        {
            frame.setMaximum(false);
            frame.reshape(c * width, r * height, width, height);
            r++;
            if (r == rows)
            {
                r = 0;
                c++;
                if (c == cols - extra)
                {
                    // start adding an extra row
                    rows++;
                    height = desktop.getHeight() / rows;
                }
            }
        }
        catch (PropertyVetoException e)
        {}
    }
}
```

这个示例程序演示了另一个常用的框体操作：将所选择的框体从当前框体移到下一个非图标框体。接着遍历所有的框体并调用`isSelected`方法，直到发现当前选定的框体为止。然后，查找框体序列中下一个非图标框体，进而通过如下调用选中它：

```
frames[next].setSelected(true);
```

正如前面那样，该方法会抛出一个`PropertyVetoException`异常，在这种情况下，需要一直进行监视。如果返回到原先那个框体，那么其他任何框体都无法选定，因此只有放弃。下面

是完整的循环代码：

```

JInternalFrame[] frames = desktop.getAllFrames();
for (int i = 0; i < frames.length; i++)
{
    if (frames[i].isSelected())
    {
        // find next frame that isn't an icon and can be selected
        int next = (i + 1) % frames.length;
        while (next != i)
        {
            if (!frames[next].isIcon())
            {
                try
                {
                    // all other frames are icons or veto selection
                    frames[next].setSelected(true);
                    frames[next].toFront();
                    frames[i].toBack();
                    return;
                }
                catch (PropertyVetoException e)
                {}
            }
            next = (next + 1) % frames.length;
        }
    }
}

```

### 6.6.5 否决属性设置

到现在为止，你已经看到所有这些否决异常，那么你可能会问，框体是怎样发布一个否决的。JInternalFrame类使用很普遍的JavaBean机制来监视这些属性设置。我们会在第8章详细讨论该机制。但是现在，我们只想展示框体是怎样对属性更改发送否决请求的。

框体通常并不想使用否决机制以抗议将窗口图标化或失去焦点，但是对于框体来说，检查它们是不是可以关闭则是很常见的。可以使用JInternalFrame类的setClosed方法关闭一个窗口。因为该方法是可否决的，因此在进行更改之前，它会调用所有已注册的可否决的更改监听器（vetoable change listener）。这样就赋予每个监听器抛出一个PropertyVetoException异常的机会，并且在它更改任何设置之前，终止对setClosed的调用。

在我们的示例程序中，我们建立了一个对话框，以询问用户是否可以关闭窗口（参见图6-51）。如果用户不同意关闭窗口，那么该窗口仍旧保持打开状态。

下面就说说要怎样才能实现这样一个通知机制。

- 为每个框体添加一个监听器对象。该监听器对象必须属于实现了VetoableChangeListener接口的某个类。最好是在刚构建完这个框体时就添加监听器。在我们的示例程序中，我们是使用框体类来构建内部框体的。另外一种选择是使用一个匿名内部类进行构建。

```
iframe.addVetoableChangeListener(listener);
```

- 实现vetoableChange方法，该方法是VetoableChangeListener接口惟一要求要实现的方法。该方法接收一个PropertyChangeEvent对象，使用getName方法查找将要更改的属性的

名称（例如，如果调用否决的方法是`setClosed(true)`），那么属性名就是“closed”。正如你将在第8章看到的那样，通过移除方法名的“set”前缀，并且将后面的字母变为小写，便可获得属性名。

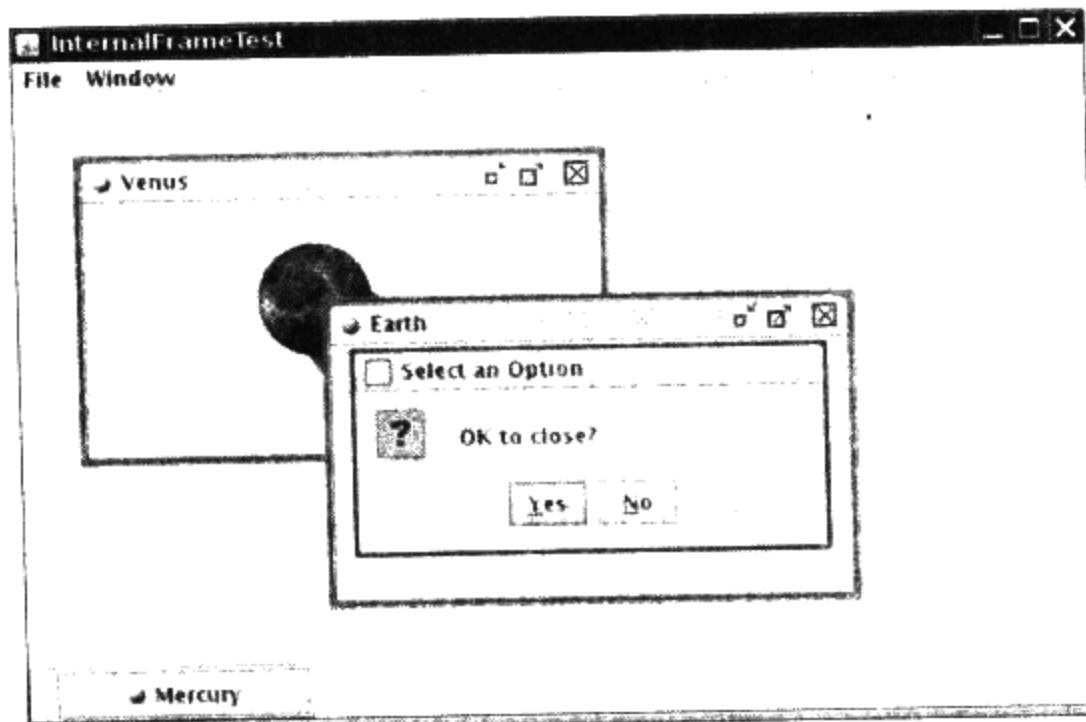


图6-51 用户可以否决关闭属性

使用`getNewValue`方法获取建议使用的新值。

```
String name = event.getPropertyName();
Object value = event.getNewValue();
if (name.equals("closed") && value.equals(true))
{
    ask user for confirmation
}
```

3. 仅仅通过抛出一个`PropertyVetoException`异常就可以阻止属性修改。如果不否决更改，则正常返回。

```
class DesktopFrame extends JFrame
    implements VetoableChangeListener
{
    ...
    public void vetoableChange(PropertyChangeEvent event)
        throws PropertyVetoException
    {
        ...
        if (not ok)
            throw new PropertyVetoException(reason, event);
        // return normally if ok
    }
}
```

#### 内部框体中的对话框

如果使用内部框体，那么不应该将`JDialog`类作为对话框。因为，这些对话框有两个缺点：

- 它们是重量级的，因为它们是在窗口系统中创建了一个新的框体。

- 窗口系统并不知道应该如何确定这些对话框与派生出它们的内部框体之间的相对位置。

相反地，对于简单的对话框，请使用JOptionPane类的showInternalXxxDialog方法。除了它们是在内部框体上放置一个轻量级窗口外，它们的运行特性和showXxxDialog方法极为相似。

对于更复杂的对话框，请使用一个JInternalFrame来构建。遗憾的是，这样你就无法使用任何对模式对话框的内置支持了。

在我们的示例程序中，我们使用了一个内部对话框，以询问用户是否可以关闭某个窗口。

```
int result = JOptionPane.showInternalConfirmDialog(  
    iframe, "OK to close?", "Select an Option", JOptionPane.YES_NO_OPTION);
```

 **注意：**如果只是想在关闭一个框体时能够被通知到，那么就应该不使用否决机制。相反地，应该安装一个InternalFrameListener监听器。内部框体监听器和WindowListener监听器运行特性极为相似。当关闭一个内部框体时，调用的是internalFrameClosing方法，而不是大家所熟悉的windowClosing方法。其他六个内部框体的通知（打开/关闭，图标化/非图标化，激活/禁用）也对应于窗口监听器的相应方法。

#### 边框拖拽

程序开发人员反对内部框体的原因之一是：它的运行特性不是很好。最缓慢的操作就是在桌面上拖拽具有复杂内容的框体。在拖动框体的过程中，桌面管理器会不断要求框体重新绘制，这样就导致其速度非常缓慢。

实际上，如果使用的Windows或者X Windows的视频驱动编写得比较差的话，你会遇到同样的问题。在大多数系统上，窗口拖动的运行速度看起来都很快，因为视频硬件支持拖动操作，在拖动过程中，可以将框体中的图像映射到屏幕别的位置上。

为了提高性能，而又不丧失用户使用的积极性，可以设置“边框拖拽”。当用户拖动框体时，只有框体的边框是连续更新的。框体里面的内容只有当用户将框体拖动到它的最终停止位置上的时候才会刷新。

为了启动边框拖动，请调用

```
desktop.setDragMode(JDesktopPane.OUTLINE_DRAG_MODE);
```

这个设置等价于JSplitPane类的“连续布局”。

 **注意：**在Swing的早期版本中，必须使用下面这句神奇的代码开启边框拖拽：

```
desktop.putClientProperty("JDesktopPane.dragMode", "outline");
```

在示例程序中，可以使用Window->Drag复选框菜单选项，开启或关闭边框拖拽。

 **注意：**桌面上的内部框体由DesktopManager类负责管理，你并不需要知道该类是怎样用于常规编程的。通过安装一个新的桌面管理器，就可以实现不同的桌面运行特性，不过我们在这里将不做介绍。

在程序清单6-21的桌面中嵌入了一个用于显示HTML页面的内部框体。执行File->Open菜单选项，会弹出一个文件对话框用于将一个本地HTML文件读取到一个新的内部框体中。如果

点击了任何一个链接，那么该链接文档便会在另外一个内部框体中显示出来。请试运行一下 Window->Cascade和Window->Tile命令。介绍完这个示例之后，我们就结束了对高级Swing特性的讨论。

### 程序清单6-21 InternalFrameTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.beans.*;
4. import javax.swing.*;
5.
6. /**
7. * This program demonstrates the use of internal frames.
8. * @version 1.11 2007-08-01
9. * @author Cay Horstmann
10.*/
11 public class InternalFrameTest
12 {
13     public static void main(String[] args)
14     {
15         EventQueue.invokeLater(new Runnable()
16         {
17             public void run()
18             {
19                 JFrame frame = new DesktopFrame();
20                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21                 frame.setVisible(true);
22             }
23         });
24     }
25 }
26
27 /**
28 * This desktop frame contains editor panes that show HTML documents.
29 */
30 class DesktopFrame extends JFrame
31 {
32     public DesktopFrame()
33     {
34         setTitle("InternalFrameTest");
35         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
36
37         desktop = new JDesktopPane();
38         add(desktop, BorderLayout.CENTER);
39
40         // set up menus
41
42         JMenuBar menuBar = new JMenuBar();
43         setJMenuBar(menuBar);
44         JMenu fileMenu = new JMenu("File");
45         menuBar.add(fileMenu);
46         JMenuItem openItem = new JMenuItem("New");
47         openItem.addActionListener(new ActionListener()
48         {
49             public void actionPerformed(ActionEvent event)
50             {
```

```
51.         createInternalFrame(new JLabel(new ImageIcon(planets[counter] + ".gif")),
52.             planets[counter]);
53.         counter = (counter + 1) % planets.length;
54.     }
55. );
56. fileMenu.add(openItem);
57. JMenuItem exitItem = new JMenuItem("Exit");
58. exitItem.addActionListener(new ActionListener()
59. {
60.     public void actionPerformed(ActionEvent event)
61.     {
62.         System.exit(0);
63.     }
64. });
65. fileMenu.add(exitItem);
66. JMenu windowMenu = new JMenu("Window");
67. menuBar.add(windowMenu);
68. JMenuItem nextItem = new JMenuItem("Next");
69. nextItem.addActionListener(new ActionListener()
70. {
71.     public void actionPerformed(ActionEvent event)
72.     {
73.         selectNextWindow();
74.     }
75. });
76. windowMenu.add(nextItem);
77. JMenuItem cascadeItem = new JMenuItem("Cascade");
78. cascadeItem.addActionListener(new ActionListener()
79. {
80.     public void actionPerformed(ActionEvent event)
81.     {
82.         cascadeWindows();
83.     }
84. });
85. windowMenu.add(cascadeItem);
86. JMenuItem tileItem = new JMenuItem("Tile");
87. tileItem.addActionListener(new ActionListener()
88. {
89.     public void actionPerformed(ActionEvent event)
90.     {
91.         tileWindows();
92.     }
93. });
94. windowMenu.add(tileItem);
95. final JCheckBoxMenuItem dragOutlineItem = new JCheckBoxMenuItem("Drag Outline");
96. dragOutlineItem.addActionListener(new ActionListener()
97. {
98.     public void actionPerformed(ActionEvent event)
99.     {
100.         desktop.setDragMode(dragOutlineItem.isSelected() ?
101.             JDesktopPane.OUTLINE_DRAG_MODE : JDesktopPane.LIVE_DRAG_MODE);
102.     }
103. });
104. windowMenu.add(dragOutlineItem);
105. }
106.
107. /**
```

```
108     * Creates an internal frame on the desktop.  
109     * @param c the component to display in the internal frame  
110     * @param t the title of the internal frame.  
111     */  
112     public void createInternalFrame(Component c, String t)  
113     {  
114         final JInternalFrame iframe = new JInternalFrame(t, true, // resizable  
115             true, // closable  
116             true, // maximizable  
117             true); // iconifiable  
118  
119         iframe.add(c, BorderLayout.CENTER);  
120         desktop.add(iframe);  
121  
122         iframe.setFrameIcon(new ImageIcon("document.gif"));  
123  
124         // add listener to confirm frame closing  
125         iframe.addVetoableChangeListener(new VetoableChangeListener()  
126         {  
127             public void vetoableChange(PropertyChangeEvent event) throws PropertyVetoException  
128             {  
129                 String name = event.getPropertyName();  
130                 Object value = event.getNewValue();  
131  
132                 // we only want to check attempts to close a frame  
133                 if (name.equals("closed") && value.equals(true))  
134                 {  
135                     // ask user if it is ok to close  
136                     int result = JOptionPane.showInternalConfirmDialog(iframe, "OK to close?",  
137                         "Select an Option", JOptionPane.YES_NO_OPTION);  
138  
139                     // if the user doesn't agree, veto the close  
140                     if (result != JOptionPane.YES_OPTION) throw new PropertyVetoException(  
141                         "User canceled close", event);  
142                 }  
143             }  
144         });  
145  
146         // position frame  
147         int width = desktop.getWidth() / 2;  
148         int height = desktop.getHeight() / 2;  
149         iframe.reshape(nextFrameX, nextFrameY, width, height);  
150  
151         iframe.show();  
152  
153         // select the frame--might be vetoed  
154         try  
155         {  
156             iframe.setSelected(true);  
157         }  
158         catch (PropertyVetoException e)  
159         {  
160         }  
161  
162         frameDistance = iframe.getHeight() - iframe.getContentPane().getHeight();  
163  
164         // compute placement for next frame
```

```
165.     nextFrameX += frameDistance;
166.     nextFrameY += frameDistance;
167.     if (nextFrameX + width > desktop.getWidth()) nextFrameX = 0;
168.     if (nextFrameY + height > desktop.getHeight()) nextFrameY = 0;
169. }
170. .
171. /**
172. * Cascades the non-iconified internal frames of the desktop.
173. */
174. public void cascadeWindows()
175. {
176.     int x = 0;
177.     int y = 0;
178.     int width = desktop.getWidth() / 2;
179.     int height = desktop.getHeight() / 2;
180.
181.     for (JInternalFrame frame : desktop.getAllFrames())
182.     {
183.         if (!frame.isIcon())
184.         {
185.             try
186.             {
187.                 // try to make maximized frames resizable; this might be vetoed
188.                 frame.setMaximum(false);
189.                 frame.reshape(x, y, width, height);
190.
191.                 x += frameDistance;
192.                 y += frameDistance;
193.                 // wrap around at the desktop edge
194.                 if (x + width > desktop.getWidth()) x = 0;
195.                 if (y + height > desktop.getHeight()) y = 0;
196.             }
197.             catch (PropertyVetoException e)
198.             {
199.             }
200.         }
201.     }
202. }
203. }
204.
205. /**
206. * Tiles the non-iconified internal frames of the desktop.
207. */
208. public void tileWindows()
209. {
210.     // count frames that aren't iconized
211.     int frameCount = 0;
212.     for (JInternalFrame frame : desktop.getAllFrames())
213.         if (!frame.isIcon()) frameCount++;
214.     if (frameCount == 0) return;
215.
216.     int rows = (int) Math.sqrt(frameCount);
217.     int cols = frameCount / rows;
218.     int extra = frameCount % rows;
219.     // number of columns with an extra row
220.
221.     int width = desktop.getWidth() / cols;
```

```
222.     int height = desktop.getHeight() / rows;
223.     int r = 0;
224.     int c = 0;
225.     for (JInternalFrame frame : desktop.getAllFrames())
226.     {
227.         if (!frame.isIcon())
228.         {
229.             try
230.             {
231.                 frame.setMaximum(false);
232.                 frame.reshape(c * width, r * height, width, height);
233.                 r++;
234.                 if (r == rows)
235.                 {
236.                     r = 0;
237.                     c++;
238.                     if (c == cols - extra)
239.                     {
240.                         // start adding an extra row
241.                         rows++;
242.                         height = desktop.getHeight() / rows;
243.                     }
244.                 }
245.             }
246.             catch (PropertyVetoException e)
247.             {
248.             }
249.         }
250.     }
251. }
252.
253. /**
254. * Brings the next non-iconified internal frame to the front.
255. */
256. public void selectNextWindow()
257. {
258.     JInternalFrame[] frames = desktop.getAllFrames();
259.     for (int i = 0; i < frames.length; i++)
260.     {
261.         if (frames[i].isSelected())
262.         {
263.             // find next frame that isn't an icon and can be selected
264.             int next = (i + 1) % frames.length;
265.             while (next != i)
266.             {
267.                 if (!frames[next].isIcon())
268.                 {
269.                     try
270.                     {
271.                         // all other frames are icons or veto selection
272.                         frames[next].setSelected(true);
273.                         frames[next].toFront();
274.                         frames[i].toBack();
275.                         return;
276.                     }
277.                     catch (PropertyVetoException e)
278.                     {
```

```

279.         }
280.     }
281.     next = (next + 1) % frames.length;
282.   }
283. }
284. }
285. }
286.
287. private JDesktopPane desktop;
288. private int nextFrameX;
289. private int nextFrameY;
290. private int frameDistance;
291. private int counter;
292. private static final String[] planets = { "Mercury", "Venus", "Earth", "Mars", "Jupiter",
293.                                         "Saturn", "Uranus", "Neptune", "Pluto", };
294.
295. private static final int DEFAULT_WIDTH = 600;
296. private static final int DEFAULT_HEIGHT = 400;
297. }

```

### javax.swing.JDesktopPane 1.2

- `JInternalFrame[] getAllFrames()`

获取该桌面中的所有内部框体。

- `void setDragMode(int mode)`

将拖动模式设置为实况拖动模式或边框拖动模式。

参数: `mode`      `JDesktopPane.LIVE_DRAG_MODE`、`JDesktopPane.OUTLINE_DRAG_MODE`  
其中之一

### javax.swing.JInternalFrame 1.2

- `JInternalFrame()`
- `JInternalFrame(String title)`
- `JInternalFrame(String title, boolean resizable)`
- `JInternalFrame(String title, boolean resizable, boolean closable)`
- `JInternalFrame(String title, boolean resizable, boolean closable, boolean maximizable)`
- `JInternalFrame(String title, boolean resizable, boolean closable, boolean maximizable, boolean iconifiable)`

构建一个新的内部框体。

参数: `title`      标题栏显示的字符串

`resizable`      如果该框体可放缩，则为true

`closable`      如果框体可以关闭，则为true

`maxmizable`      如果该框体可以最大化，则为true

`iconifiable`      如果该框体可以图标化，则为true

- `boolean isResizable()`

- `void setResizable(boolean b)`

- `boolean isClosable()`

- void setClosable(boolean b)
- boolean isMaximizable()
- void setMaximizable(boolean b)
- boolean isIconifiable()
- void setIconifiable(boolean b)

获取并设置属性resizable、closable、maximizable以及iconifiable。如果该属性为true，那么在框体的标题栏处会显示一个图标，用于缩放、关闭、最大化或者图标化该内部框体。

- boolean isIcon()
- void setIcon(boolean b)
- boolean isMaximum()
- void setMaximum(boolean b)
- boolean isClosed()
- void setClosed(boolean b)

获取或设置icon、maximum或者closed属性。如果该属性为true，那么该内部框体可以图标化、最大化以及关闭。

- boolean isSelected()
- void setSelected(boolean b)

获取或设置selected属性。如果该属性为true，那么当前的内部框体就成为桌面上被选定的框体。

- void moveToFront()
- void moveToBack()

将该内部框体移到桌面的前面或后面。

- void reshape(int x, int y, int width, int height)

移动并缩放该内部框体。

参数：x, y 框体的左上角  
width, height 框体的宽度及高度

- Container getContentPane()
- void setContentPane(Container c)

获取并设置该内部框体的内容面板。

- JDesktopPane getDesktopPane()

获取该内部框体的桌面面板。

- Icon getFrameIcon()
- void setFrameIcon(Icon anIcon)

获取并设置显示在标题栏中的框体图标。

- boolean isVisible()
- void setVisible(boolean b)

获取并设置“可见”属性。

- void show()

将该内部框体设为可视的，并将它移到前面。

#### API javax.swing.JComponent 1.2

- void addVetoableChangeListener(VetoableChangeListener listener)

添加一个可否决的更改监听器，当试图更改一个受约束属性的时候，会将更改信息通告给它。

#### API java.beans.VetoableChangeListener 1.1

- void vetoableChange(PropertyChangeEvent event)

当受约束属性的set方法通知可否决的更改监视器的时候，调用该方法。

#### API java.beans.PropertyChangeEvent 1.1

- String getPropertyName()

返回将要被更改的属性的名称。

- Object getNewValue()

返回建议用于该属性的新值。

#### API java.beans.PropertyVetoException 1.1

- PropertyVetoException(String reason, PropertyChangeEvent event)

构建一个属性否决异常。

参数: reason 否决的原因

event 被否决的事件

你已经看到了可以如何使用Swing框架提供的复杂构件。在下一章，我们将转向AWT相关的话题：复杂的绘制操作、图像处理、打印机制以及与本地窗口系统的接口机制等。

# 第7章 高级AWT

- ▲ 绘图操作流程
- ▲ 形状
- ▲ 区域
- ▲ 笔划
- ▲ 着色
- ▲ 坐标变换
- ▲ 剪切
- ▲ 透明与组合

- ▲ 绘图提示
- ▲ 图像读取器和写入器
- ▲ 图像处理
- ▲ 打印
- ▲ 剪贴版
- ▲ 拖放操作
- ▲ 平台集成

Graphics类有多种方法可以用来创建简单的图形。这些方法对于简单的applet和应用来说已经绰绰有余了，但是当你创建复杂的图形或者需要全面控制图形的外观时，它们就显得力不从心了。Java 2D API 是一个更加成熟的类库，可以用它产生高质量的图形。本章中，我们将概要地介绍一下该API。

然后我们将要讲述关于打印方面的问题，说明如何将打印功能纳入到程序之中。

最后，我们将介绍在程序间传递数据的两种方法：系统剪切板和拖放机制。可以使用这些技术在两个Java应用之间，或者在Java应用和本机程序之间传递数据。

## 7.1 绘图操作流程

在最初的JDK 1.0中，用来绘制形状的是一种非常简单的机制，即选择颜色和画图的模式，并调用Graphics 类的各种方法，比如drawRect或者filloval。而Java 2D API支持更多的功能：

- 可以很容易地绘制各式各样的形状。
- 可以控制绘制形状的笔划，即控制跟踪形状边界的绘图笔。
- 可以用单色、变化的色调和重复的模式来填充各种形状。
- 可以使用变换法，对各种形状进行移动、缩放、旋转和拉伸。
- 可以对形状进行剪切，将其限制在任意的区域内。
- 可以选择各种组合规则，来描述如何将新形状的像素与现有的像素组合起来。
- 可以提供绘制图形提示，以便在速度与绘图质量之间实现平衡。

如果要绘制一个形状，可以按照如下步骤操作：

1. 获得一个Graphics2D类的对象，该类是Graphics类的子类。自Java SE 1.2以来，paint 和paintComponent等方法就能够自动地接收一个Graphics2D类的对象，这时可以直接使用如下的转型：

```
public void paintComponent(Graphics g)
{
    Graphics2D g2 = (Graphics2D) g;
    ...
}
```

2. 使用`setRenderingHints`方法来设置绘图提示，它提供了速度与绘图质量之间的一种平衡。

```
RenderingHints hints = ...;
g2.setRenderingHints(hints);
```

3. 使用`setStroke`方法来设置笔划，笔划用于绘制形状的边框。可以选择边框的粗细和线段的虚实。

```
Stroke stroke = ...;
g2.setStroke(stroke);
```

4. 使用`setPaint`方法来设置着色法，着色法用于填充诸如笔划路径或者形状内部等区域的颜色。可以创建单色、渐变色或者平铺的填充模式。

```
Paint paint = ...;
g2.setPaint(paint);
```

5. 使用`clip`方法来设置剪切区域。

```
Shape clip = ...;
g2.clip(clip);
```

6. 使用`transform`方法设置一个从用户空间到设备空间的变换方式。如果使用变换方式比使用像素坐标更容易定义在定制坐标系统中的形状，那么就可以使用变换方式。

```
AffineTransform transform = ...;
g2.transform(transform);
```

7. 使用`setComposite`方法设置一个组合规则，用来描述如何将新像素与现有的像素组合起来。

```
Composite composite = ...;
g2.setComposite(composite);
```

8. 建立一个形状，Java 2D API 提供了用来组合各种形状的许多形状对象和方法。

```
Shape shape = ...;
```

9. 绘制或者填充该形状。如果要绘制该形状，那么它的边框就会用笔划画出来。如果要填充该形状，那么它的内部就会被着色。

```
g2.draw(shape);
g2.fill(shape);
```

当然，在许多实际的环境中，并不需要采用所有这些操作步骤。Java 2D 图形上下文中有合理的默认设置。只有当你确实想要改变设置时，再去修改这些默认设置。

在下面的几节中，我们将要介绍如何描绘形状、笔划、着色、变换及组合的规则。

各种不同的`set`方法只是用于设置2D图形上下文的状态，它们并不进行任何实际的绘图操作。同样，在构建`shape`对象时，也不进行任何绘图操作。只有在调用`draw`或者`fill`方法时，才会绘制出图形的形状，而就在此刻，这个新的图形在绘图操作流程中被计算了出来（参见图7-1）

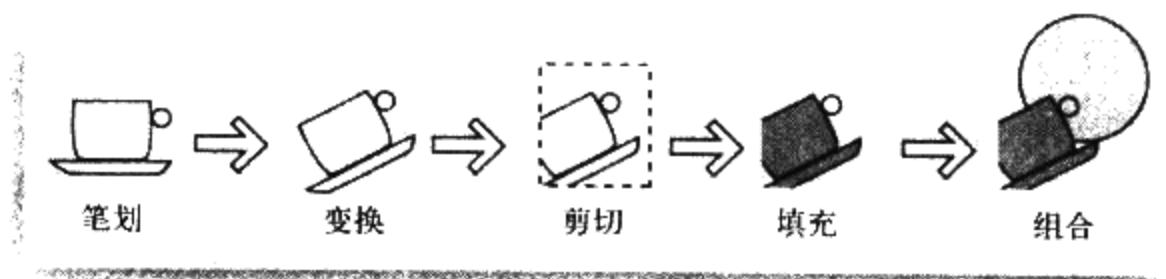


图7-1 绘图操作流程

在绘图流程中，需要以下这些操作步骤来绘制一个形状：

1. 用笔划画出形状的线条；
2. 对形状进行变换操作；
3. 对形状进行剪切。如果形状与剪切区域之间没有任何相交的地方，那么就停止本次操作过程；
4. 对剪切后的形状进行填充；
5. 把填充后的形状与已有的形状进行组合（在图7-1中，圆形是已有像素部分，杯子的形状加在了它的上面）。

在下一节中，将会讲述如何对形状进行定义。然后，我们将转而对2D图形上下文设置进行介绍。

#### API java.awt.Graphics2D 1.2

- `void draw(Shape s)`  
用当前的笔划来绘制给定形状的边框。
- `void fill(Shape s)`  
用当前的着色方案来填充给定形状的内部。

## 7.2 形状

下面是Graphics类中绘制形状的若干方法：

```
drawLine  
drawRectangle  
drawRoundRect  
draw3DRect  
drawPolygon  
drawPolyline  
drawOval  
drawArc
```

它们还有对应的fill方法，这些方法从JDK 1.0起就被纳入到Graphics类中了。Java 2D API 使用了一套完全不同的面向对象的处理方法，即不再使用方法，而是使用下面的这些类：

```
Line2D  
Rectangle2D  
RoundRectangle2D  
Ellipse2D  
Arc2D
```

QuadCurve2D  
CubicCurve2D  
GeneralPath

这些类全部都实现了Shape接口。

最后，还有一个Point2D类，它用x和y坐标来描述一个点。点对于定义形状非常有用，不过它们本身并不是形状。

如果要绘制一个形状，首先要创建一个实现了Shape接口的类的对象，然后调用Graphics2D类的draw方法。

Line2D、Rectangle2D、RoundRectangle2D、Ellipse2D和Arc2D等这些类对应于drawLine、drawRectangle、drawRoundRect、drawOval和drawArc等方法。（“3D矩形”的概念已经理所当然地过时了，因为没有类似draw3DRect的方法。）Java 2D API 提供了两个补充类，即二次曲线类和三次曲线类。我们将在本节的后面部分阐释这些形状。Java 2D API中没有任何Polygon2D类。相反，它用GeneralPath类来描述由线条、二次曲线、三次曲线构成的线条路径。可以使用GeneralPath 来描述一个多边形；我们将在本节的后面部分对它进行介绍。

下面这些类：

Rectangle2D  
RoundRectangle2D  
Ellipse2D  
Arc2D

都是从一个常用的超类RectangularShape继承而来的。诚然，椭圆形和弧形都不是矩形，但是它们都有一个矩形的边界框（参见图7-2）。

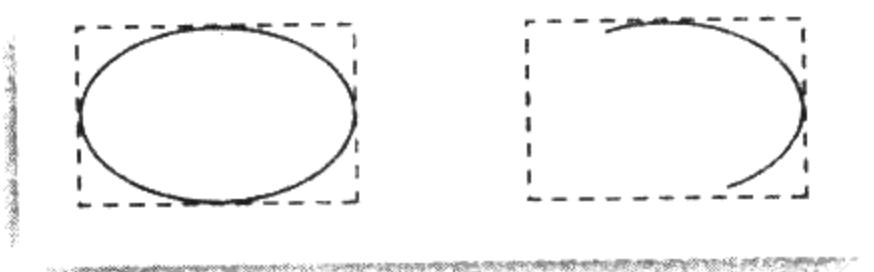


图7-2 椭圆形和弧形的矩形边界框

名字以“2D”结尾的每个类都有两个子类，用于指定坐标是float类型的还是double类型的。在本书的第I卷中，我们已经介绍了Rectangle2D.Float和Rectangle2D.Double。

其他类也同样，比如Arc2D.Float和Arc2D.Double。

从内部来讲，所有的图形类使用的都是float类型的坐标，因为float类型的数占用较少的存储空间，而且它们有足够高的几何计算精度。然而，Java编程语言使得对float类型的数的操作要稍微复杂些。由于这个原因，图形类的大多数方法使用的都是double类型的参数和返回值。只有在创建一个2D对象的时候，才需要选择究竟是使用带有float类型坐标的构造器，还是使用带有double类型坐标的构造器。例如：

```
Rectangle2D floatRect = new Rectangle2D.Float(5F, 10F, 7.5F, 15F);
Rectangle2D doubleRect = new Rectangle2D.Double(5, 10, 7.5, 15);
```

`Xxx2D.Float`和`Xxx2D.Double`两个类都是`Xxx2D`类的子类，在对象被构建之后，再记住其确切的子类型实质上已经没有任何额外的好处了，因此可以将刚被构建的对象存储为一个超类变量，正如上面代码示例中所阐释的那样。

从这些类古怪的名字中就可以判断出，`Xxx2D.Float`和`Xxx2D.Double`两个类同时也是`Xxx2D`类的内部类。这只是为了在语法上比较方便，以避免外部类的名字变得太长。

图7-3显示了各个形状类之间的关系。不过图中省略了`Double`和`Float`子类。来自以前的2D类库的遗留类用灰色的填充色标识。

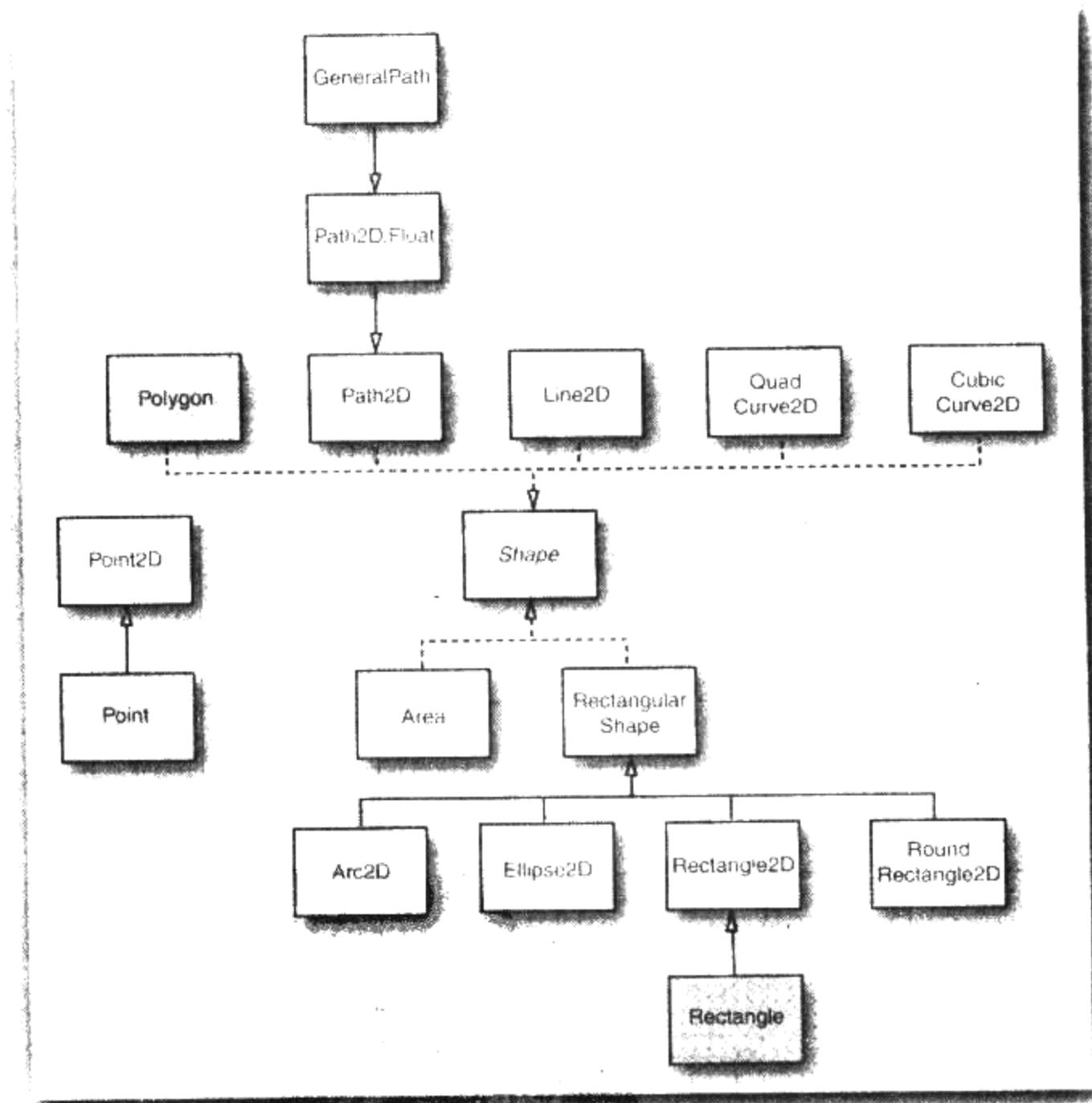


图7-3 形状类之间的关系

## 使用形状类

我们在本书的第I卷第7章中介绍了如何使用`Rectangle2D`、`Ellipse2D`和`Line2D`类的方法。本节将介绍如何建立其他的2D形状。

如果要建立一个`RoundRectangle2D`形状，应该设定左上角、宽度、高度及应该变成圆角的

边角区的x和y的坐标尺寸（参见图7-4）。例如，调用下面的方法：

```
RoundRectangle2D r = new RoundRectangle2D.Double(150, 200, 100, 50, 20, 20);
```

便产生了一个带圆角的矩形，每个角的圆半径为20。

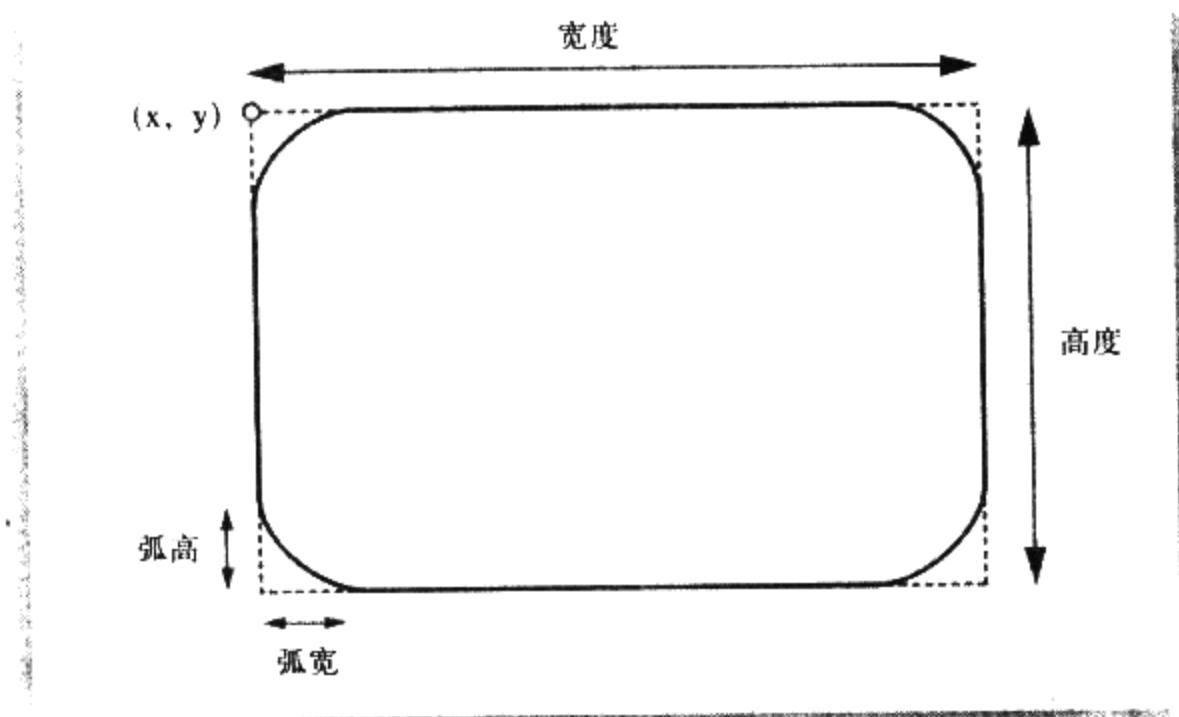


图7-4 构建一个 RoundRectangle2D

如果要建立一个弧形，首先应该设定边界框，接着设定它的起始角度和弧形跨越的角度（见图7-5），并且设定弧形闭合的类型，即Arc2D.OPEN、Arc2D.PIE或者Arc2D.CHORD这几种类型中的一个。

```
Arc2D a = new Arc2D(x, y, width, height, startAngle, arcAngle, closureType);
```

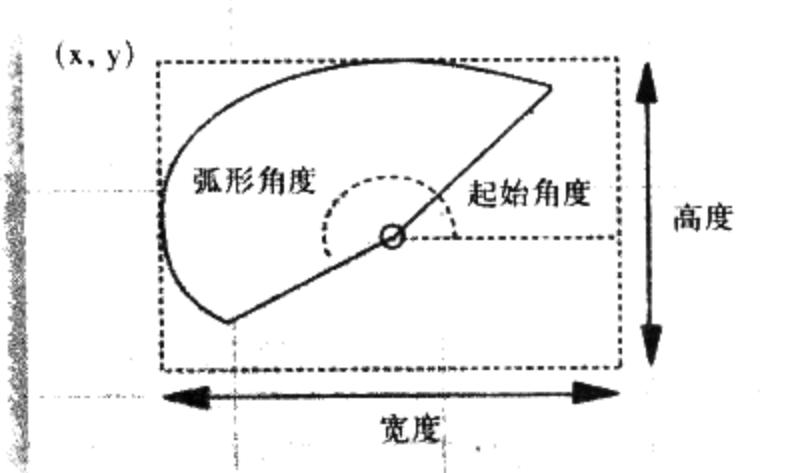


图7-5 构建一个椭圆弧形

图7-6显示了几种弧形的类型。

**X 警告：**如果弧形是椭圆的，那么弧形角就完全不是直接计算出来的。API文档中描述到：“角是相对于非正方形的矩形边框指定的，以使得45度总是落到了从椭圆中心指向矩形边

框右上角的方向上。因此，如果矩形边框的一条轴比另一条轴明显长许多，那么弧形段的起始点和终止点就会与边框中的长轴斜交。”但是，文档中并没有说明如何计算这种“斜交”。下面是其细节：

假设弧形的中心是原点，而且点  $(x,y)$  在弧形上。那么我们可以用下面的公式来获得这个斜交角：

```
skewedAngle = Math.toDegrees(Math.atan2(-y * height, x * width));
```

这个值介于 -180 到 180 之间。按照这种方式计算斜交的起始角和终止角，然后，计算两个斜交角之间的差，如果起始角或角的差是负数，则加上 360。之后，将起始角和角的差提供给弧形的构造器。如果运行本节末尾的程序，你用肉眼就能观察到这种计算所产生的用于弧形构造器的值是正确的。可参见本章图 7-9。

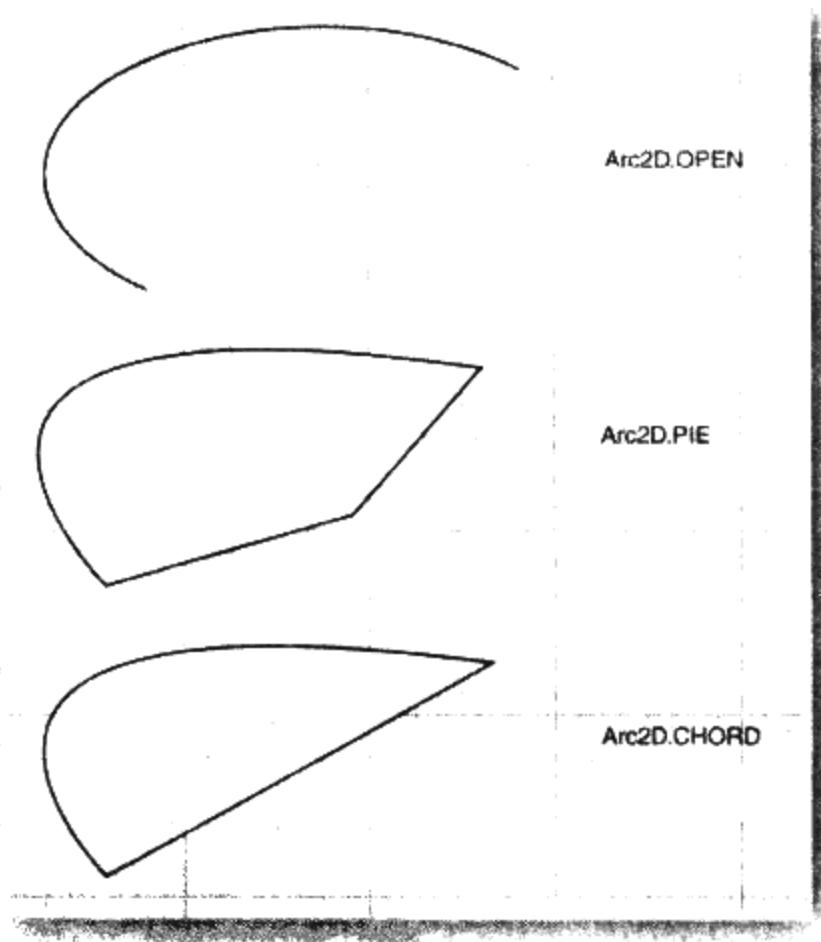


图7-6 弧形的类型

Java 2D API 提供了对二次曲线和三次曲线的支持。在本章中，我们并不会深入介绍这些曲线的数学特征。我们建议你通过运行程序清单 7-1 的代码，对曲线的形状有一个感性的认识。正如在图 7-7 和图 7-8 中看到的那样，二次曲线和三次曲线是由两个端点和一个或两个控制点来设定的。移动控制点，曲线的形状就会改变。

如果要构建二次曲线和三次曲线，需要给出两个端点和控制点的坐标。例如，

```
QuadCurve2D q = new QuadCurve2D.Double(startX, startY, controlX, controlY, endX, endY);
CubicCurve2D c = new CubicCurve2D.Double(startX, startY, control1X, control1Y,
    control2X, control2Y, endX, endY);
```

二次曲线不是非常灵活，所以实际上它并不常用。三次曲线（比如用 CubicCurve2D 类绘制

的贝济埃（Bézier）曲线）却是非常常见的。通过将三次曲线组合起来，使得连接点的各个斜率互相匹配，就能够创建复杂的、外观平滑的曲线形状。如果要了解这方面的详细信息，请参阅James D. Foley、Andries van Dam和Steven K. Feiner等人合作撰写的《Computer Graphics: Principles and Practice》<sup>⊖</sup>，Addison Wesley出版社1995年出版。

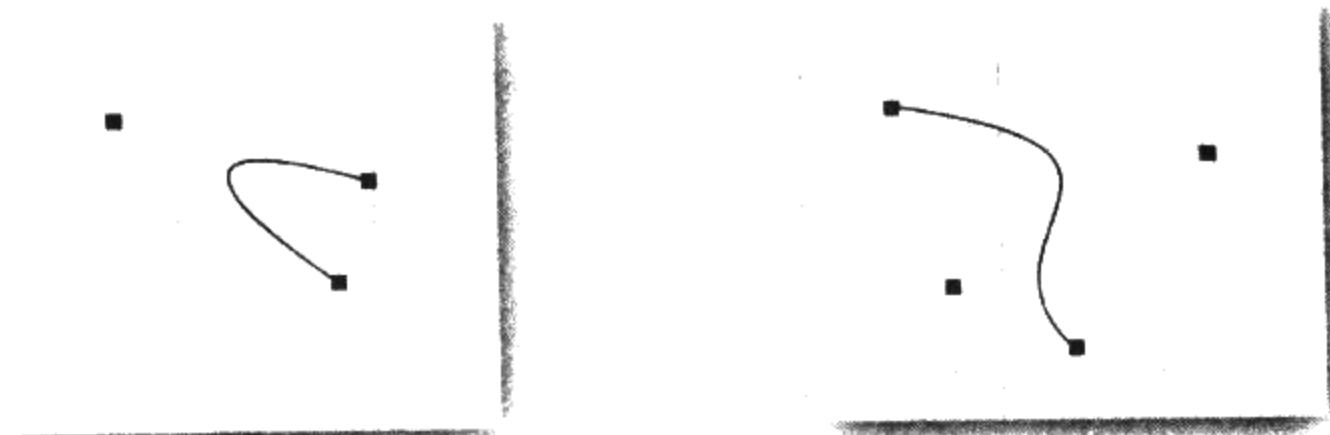


图7-7 二次曲线

图7-8 三次曲线

可以建立线段、二次曲线和三次曲线的任意序列，并把它们存放到一个GeneralPath对象中去。可以用moveTo方法来指定路径的第一个坐标，例如，

```
GeneralPath path = new GeneralPath();
path.moveTo(10, 20);
```

然后，可以通过调用lineTo、quadTo或curveTo三种方法之一来扩展路径。这些方法是通过线条、二次曲线或者三次曲线来扩展路径的。如果要调用lineTo方法，需要提供它的端点。而对两个曲线方法的调用，需要先提供控制点，然后提供端点。例如，

```
path.lineTo(20, 30);
path.curveTo(control1X, control1Y, control2X, control2Y, endX, endY);
```

可以调用closePath方法来闭合路径，它能够绘制一条回到最后一个moveTo的线条。

如果要绘制一个多边形，只需调用moveTo方法，以到达第一个拐角点，然后反复调用lineTo方法，以便到达其他的拐角点。最后调用closePath方法来闭合多边形。程序清单7-1更加详细地展示了构建多边形的方法。

普通路径没有必要一定要连接在一起，你随时都可以调用moveTo方法来建立一个新的路径段。

最后，可以使用append方法，向普通路径添加任意个Shape对象。如果新建的形状应该连接到路径的最后一个端点，那么append方法的第二个参数值就是true，如果不应该连接，那么该参数值就是false。例如，调用下面的方法：

```
Rectangle2D r = ...;
path.append(r, false);
```

可以把矩形的边框添加到该路径中，但是并不与现有的路径连接在一起。但是，下面的方

<sup>⊖</sup> 本书的中文版以及英文影印版已由机械工业出版社出版。中文版书号为：7-111-24102-7，英文影印版书号为：7-111-23916-1。——编辑注

法调用：

```
path.append(r, true);
```

则是在路径的终点和矩形的起点之间添加了一条直线，然后将矩形的边框添加到该路径中。

程序清单7-1中的程序使你能够构建许多示例路径。图7-7和图7-8显示了运行该程序的示例结果。你可以从组合框中选择一个形状绘制器。该程序包含的形状绘制器可以用来绘制：

- 直线；
- 矩形、圆角矩形和椭圆形；
- 弧形（除了显示弧形本身外，还可以显示矩形边框的线条和起始角度及结束角度）；
- 多边形（使用GeneralPath方法）；
- 二次曲线和三次曲线。

可以用鼠标来调整控制点。当你移动控制点时，形状会连续地重绘。

该程序有些复杂，因为它可以用来处理多种不同的形状，并且支持对控制点的拖拽操作。

抽象超类ShapeMaker封装了形状绘制器类的共性特征。每个形状都拥有固定数量的控制点，用户可以在控制点周围随意移动。getPointCount方法用于返回控制点数量的值。下面这个抽象方法：

```
Shape makeShape(Point2D[] points)
```

将在给定控制点的当前位置的情况下，计算实际的形状。toString方法用于返回类的名字，这样，ShapeMaker对象就能够放置到一个JComboBox中。

为了激活控制点的拖拽特征，ShapePanel类用以处理鼠标事件和鼠标移动事件。当鼠标在一个矩形上面被按下时，那么拖拽鼠标就可以移动该矩形了。

大部分形状绘制器类都很简单，它们的makeShape方法只是用于构建和返回需要的形状。然而，当使用ArcMaker类的时候，需要计算弧形的变形起始角度和结束角度。此外，为了说明这些计算确实是正确的，返回的形状应该是包含该弧本身、矩形边框和从弧形中心到角度控制点之间的线条等的GeneralPath（参见图7-9）。

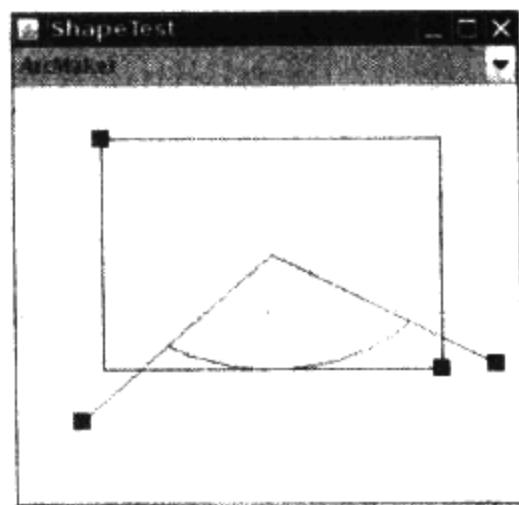


图7-9 ShapeTest 程序的运行结果

### 程序清单7-1 ShapeTest.java

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import java.awt.geom.*;
4 import java.util.*;
5 import javax.swing.*;
6
7 /**
8 * This program demonstrates the various 2D shapes.
9 * @version 1.02 2007-08-16
10 * @author Cay Horstmann
11 */
12 public class ShapeTest
13 {
```

```
14  public static void main(String[] args)
15  {
16      EventQueue.invokeLater(new Runnable()
17      {
18          public void run()
19          {
20              JFrame frame = new ShapeTestFrame();
21              frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22              frame.setVisible(true);
23          }
24      });
25  }
26 }
27
28 /**
29 * This frame contains a combo box to select a shape and a component to draw it.
30 */
31 class ShapeTestFrame extends JFrame
32 {
33     public ShapeTestFrame()
34     {
35         setTitle("ShapeTest");
36         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
37
38         final ShapeComponent comp = new ShapeComponent();
39         add(comp, BorderLayout.CENTER);
40         final JComboBox comboBox = new JComboBox();
41         comboBox.addItem(new LineMaker());
42         comboBox.addItem(new RectangleMaker());
43         comboBox.addItem(new RoundRectangleMaker());
44         comboBox.addItem(new EllipseMaker());
45         comboBox.addItem(new ArcMaker());
46         comboBox.addItem(new PolygonMaker());
47         comboBox.addItem(new QuadCurveMaker());
48         comboBox.addItem(new CubicCurveMaker());
49         comboBox.addActionListener(new ActionListener()
50         {
51             public void actionPerformed(ActionEvent event)
52             {
53                 ShapeMaker shapeMaker = (ShapeMaker) comboBox.getSelectedItem();
54                 comp.setShapeMaker(shapeMaker);
55             }
56         });
57         add(comboBox, BorderLayout.NORTH);
58         comp.setShapeMaker((ShapeMaker) comboBox.getItemAt(0));
59     }
60
61     private static final int DEFAULT_WIDTH = 300;
62     private static final int DEFAULT_HEIGHT = 300;
63 }
64
65 /**
66 * This component draws a shape and allows the user to move the points that define it.
67 */
68 class ShapeComponent extends JComponent
69 {
70     public ShapeComponent()
```

```
71.  {
72.      addMouseListener(new MouseAdapter()
73.      {
74.          public void mousePressed(MouseEvent event)
75.          {
76.              Point p = event.getPoint();
77.              for (int i = 0; i < points.length; i++)
78.              {
79.                  double x = points[i].getX() - SIZE / 2;
80.                  double y = points[i].getY() - SIZE / 2;
81.                  Rectangle2D r = new Rectangle2D.Double(x, y, SIZE, SIZE);
82.                  if (r.contains(p))
83.                  {
84.                      current = i;
85.                      return;
86.                  }
87.              }
88.          }
89.
90.          public void mouseReleased(MouseEvent event)
91.          {
92.              current = -1;
93.          }
94.      });
95.      addMouseMotionListener(new MouseMotionAdapter()
96.      {
97.          public void mouseDragged(MouseEvent event)
98.          {
99.              if (current == -1) return;
100.             points[current] = event.getPoint();
101.             repaint();
102.         }
103.     });
104.     current = -1;
105. }
106.
107. /**
108. * Set a shape maker and initialize it with a random point set.
109. * @param aShapeMaker a shape maker that defines a shape from a point set
110. */
111. public void setShapeMaker(ShapeMaker aShapeMaker)
112. {
113.     shapeMaker = aShapeMaker;
114.     int n = shapeMaker.getPointCount();
115.     points = new Point2D[n];
116.     for (int i = 0; i < n; i++)
117.     {
118.         double x = generator.nextDouble() * getWidth();
119.         double y = generator.nextDouble() * getHeight();
120.         points[i] = new Point2D.Double(x, y);
121.     }
122.     repaint();
123. }
124.
125. public void paintComponent(Graphics g)
126. {
```

```
127.     if (points == null) return;
128.     Graphics2D g2 = (Graphics2D) g;
129.     for (int i = 0; i < points.length; i++)
130.     {
131.         double x = points[i].getX() - SIZE / 2;
132.         double y = points[i].getY() - SIZE / 2;
133.         g2.fill(new Rectangle2D.Double(x, y, SIZE, SIZE));
134.     }
135.
136.     g2.draw(shapeMaker.makeShape(points));
137. }
138.
139. private Point2D[] points;
140. private static Random generator = new Random();
141. private static int SIZE = 10;
142. private int current;
143. private ShapeMaker shapeMaker;
144. }
145.
146. /**
147. * A shape maker can make a shape from a point set. Concrete subclasses must return a shape
148. * in the makeShape method.
149. */
150. abstract class ShapeMaker
151. {
152.     /**
153.      * Constructs a shape maker.
154.      * @param aPointCount the number of points needed to define this shape.
155.      */
156.     public ShapeMaker(int aPointCount)
157.     {
158.         pointCount = aPointCount;
159.     }
160.
161.     /**
162.      * Gets the number of points needed to define this shape.
163.      * @return the point count
164.      */
165.     public int getPointCount()
166.     {
167.         return pointCount;
168.     }
169.
170.     /**
171.      * Makes a shape out of the given point set.
172.      * @param p the points that define the shape
173.      * @return the shape defined by the points
174.      */
175.     public abstract Shape makeShape(Point2D[] p);
176.
177.     public String toString()
178.     {
179.         return getClass().getName();
180.     }
181.
182.     private int pointCount;
183. }
```

```
184.  
185  /**  
186. * Makes a line that joins two given points.  
187. */  
188 class LineMaker extends ShapeMaker  
189 {  
190     public LineMaker()  
191     {  
192         super(2);  
193     }  
194.  
195     public Shape makeShape(Point2D[] p)  
196     {  
197         return new Line2D.Double(p[0], p[1]);  
198     }  
199 }  
200.  
201 /**  
202. * Makes a rectangle that joins two given corner points.  
203. */  
204 class RectangleMaker extends ShapeMaker  
205 {  
206     public RectangleMaker()  
207     {  
208         super(2);  
209     }  
210.  
211     public Shape makeShape(Point2D[] p)  
212     {  
213         Rectangle2D s = new Rectangle2D.Double();  
214         s setFrameFromDiagonal(p[0], p[1]);  
215         return s;  
216     }  
217 }  
218.  
219 /**  
220. * Makes a round rectangle that joins two given corner points.  
221. */  
222 class RoundRectangleMaker extends ShapeMaker  
223 {  
224     public RoundRectangleMaker()  
225     {  
226         super(2);  
227     }  
228.  
229     public Shape makeShape(Point2D[] p)  
230     {  
231         RoundRectangle2D s = new RoundRectangle2D.Double(0, 0, 0, 0, 20, 20);  
232         s setFrameFromDiagonal(p[0], p[1]);  
233         return s;  
234     }  
235 }  
236.  
237 /**  
238. * Makes an ellipse contained in a bounding box with two given corner points.  
239. */  
240 class EllipseMaker extends ShapeMaker
```

```
241. {
242.     public EllipseMaker()
243.     {
244.         super(2);
245.     }
246.
247.     public Shape makeShape(Point2D[] p)
248.     {
249.         Ellipse2D s = new Ellipse2D.Double();
250.         s.setFrameFromDiagonal(p[0], p[1]);
251.         return s;
252.     }
253. }
254.
255. /**
256. * Makes an arc contained in a bounding box with two given corner points, and with starting
257. * and ending angles given by lines emanating from the center of the bounding box and ending
258. * in two given points. To show the correctness of the angle computation, the returned shape
259. * contains the arc, the bounding box, and the lines.
260. */
261. class ArcMaker extends ShapeMaker
262. {
263.     public ArcMaker()
264.     {
265.         super(4);
266.     }
267.
268.     public Shape makeShape(Point2D[] p)
269.     {
270.         double centerX = (p[0].getX() + p[1].getX()) / 2;
271.         double centerY = (p[0].getY() + p[1].getY()) / 2;
272.         double width = Math.abs(p[1].getX() - p[0].getX());
273.         double height = Math.abs(p[1].getY() - p[0].getY());
274.
275.         double skewedStartAngle = Math.toDegrees(Math.atan2(-(p[2].getY() - centerY)
276.             * width, (p[2].getX() - centerX)
277.             * height));
278.         double skewedEndAngle = Math.toDegrees(Math.atan2(-(p[3].getY() - centerY)
279.             * width, (p[3].getX() - centerX)
280.             * height));
281.         double skewedAngleDifference = skewedEndAngle - skewedStartAngle;
282.         if (skewedStartAngle < 0) skewedStartAngle += 360;
283.         if (skewedAngleDifference < 0) skewedAngleDifference += 360;
284.
285.         Arc2D s = new Arc2D.Double(0, 0, 0, 0, skewedStartAngle, skewedAngleDifference,
286.             Arc2D.OPEN);
287.         s.setFrameFromDiagonal(p[0], p[1]);
288.
289.         GeneralPath g = new GeneralPath();
290.         g.append(s, false);
291.         Rectangle2D r = new Rectangle2D.Double();
292.         r.setFrameFromDiagonal(p[0], p[1]);
293.         g.append(r, false);
294.         Point2D center = new Point2D.Double(centerX, centerY);
295.         g.append(new Line2D.Double(center, p[2]), false);
296.         g.append(new Line2D.Double(center, p[3]), false);
297.         return g;
```

```
298.     }
299. }
300.
301. /**
302. * Makes a polygon defined by six corner points.
303. */
304. class PolygonMaker extends ShapeMaker
305. {
306.     public PolygonMaker()
307.     {
308.         super(6);
309.     }
310.
311.     public Shape makeShape(Point2D[] p)
312.     {
313.         GeneralPath s = new GeneralPath();
314.         s.moveTo((float) p[0].getX(), (float) p[0].getY());
315.         for (int i = 1; i < p.length; i++)
316.             s.lineTo((float) p[i].getX(), (float) p[i].getY());
317.         s.closePath();
318.         return s;
319.     }
320. }
321.
322. /**
323. * Makes a quad curve defined by two end points and a control point.
324. */
325. class QuadCurveMaker extends ShapeMaker
326. {
327.     public QuadCurveMaker()
328.     {
329.         super(3);
330.     }
331.
332.     public Shape makeShape(Point2D[] p)
333.     {
334.         return new QuadCurve2D.Double(p[0].getX(), p[0].getY(), p[1].getX(), p[1].getY(), p[2]
335.                                         .getX(), p[2].getY());
336.     }
337. }
338.
339. /**
340. * Makes a cubic curve defined by two end points and two control points.
341. */
342. class CubicCurveMaker extends ShapeMaker
343. {
344.     public CubicCurveMaker()
345.     {
346.         super(4);
347.     }
348.
349.     public Shape makeShape(Point2D[] p)
350.     {
351.         return new CubicCurve2D.Double(p[0].getX(), p[0].getY(), p[1].getX(), p[1].getY(), p[2]
352.                                         .getX(), p[2].getY(), p[3].getX(), p[3].getY());
353.     }
354. }
```

**API** **java.awt.geom.RoundRectangle2D.Double 1.2**

- RoundRectangle2D.Double(double x, double y, double width, double height, double arcWidth, double arcHeight)

用给定的矩形边框和弧形尺寸构建一个圆角矩形。参见图7-4有关arcWidth和arcHeight参数的解释。

**API** **java.awt.geom.Arc2D.Double 1.2**

- Arc2D.Double(double x, double y, double w, double h, double startAngle, double arcAngle, int type)

用给定的矩形边框、起始角度、弧形角度和弧形类型构建一个弧形。startAngle和arcAngle在图7-5中已做介绍，type是Arc2D.OPEN、Arc2D.PIE和Arc2D.CHORD之一。

**API** **java.awt.geom.QuadCurve2D.Double 1.2**

- QuadCurve2D.Double(double x1, double y1, double ctrlx, double ctrlY, double x2, double y2)

用起始点、控制点和结束点构建一条二次曲线。

**API** **java.awt.geom.CubicCurve2D.Double 1.2**

- CubicCurve2D.Double(double x1, double y1, double ctrlx1, double ctrlY1, double ctrlx2, double ctrlY2, double x2, double y2)

用起始点、两个控制点和结束点构建一条三次曲线。

**API** **java.awt.geom.GeneralPath 1.2**

- GeneralPath()

构建一条空的普通路径。

**API** **java.awt.geom.Path2D.Float 6**

- void moveTo(float x, float y)

使(x, y)成为当前点，也就是下一个线段的起始点。

- void lineTo(float x, float y)

- void quadTo(float ctrlx, float ctrlY, float x, float y)

- void curveTo(float ctrlx1, float ctrlY1, float ctrlx2, float ctrlY2, float x, float y)

从当前点绘制一个线条、二次曲线或者三次曲线到达结束点(x, y)，并且使该结束点成为当前点。

**API** **java.awt.geom.Path2D 6**

- void append(Shape s, boolean connect)

将给定形状的边框添加到普通路径中去。如果布尔型变量connect的值是true的话，那

么该普通路径的当前点与添加进来的形状的起始点之间用一条直线连接起来。

- `void closePath()`

从当前点到路径的第一点之间绘制一条直线，从而使路径闭合。

### 7.3 区域

在上一节中，我们介绍了如何通过建立由线条和曲线构成的普通路径来绘制复杂的形状。通过使用足够数量的线条和曲线可以绘制出任何一种形状，例如，在屏幕上和打印文件上看到的字符的各种字体形状，都是由线条和三次曲线构成的。

有时候，使用各种不同形状的区域，比如矩形、多边形和椭圆形来建立形状，可能会更加容易描述。Java 2D API支持四种区域几何作图（constructive area geometry）操作，用于将两个区域组合成一个区域。

- `add`: 组合区域包含了所有位于第一个区域或第二个区域内的点。
- `subtract`: 组合区域包含了所有位于第一个区域内的点，但是不包括任何位于第二个区域内的点。
- `intersect`: 组合区域既包含了所有位于第一个区域内的点，又包含了所有位于第二个区域内的点。
- `exclusiveOr`: 组合区域包含了所有位于第一个区域内，或者是位于第二个区域内的所有点，但是这些点不能同时位于两个区域内。

图7-10显示了这些操作的结果。

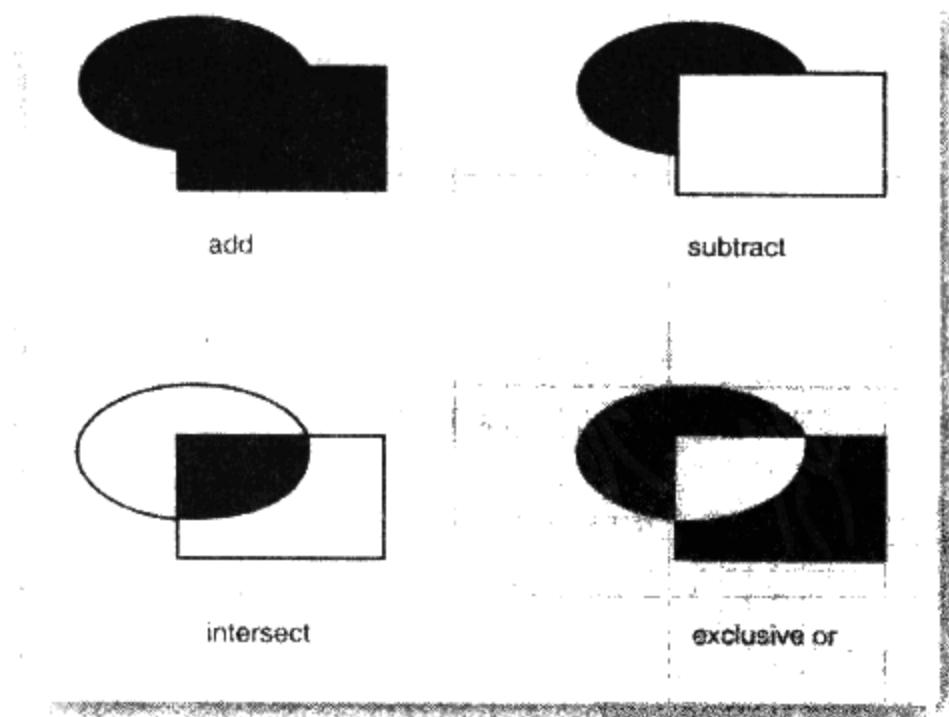


图7-10 区域几何作图操作

如果要构建一个复杂的区域，可以使用下面的方法先创建一个默认的区域对象。

```
Area a = new Area();
```

然后，将该区域和其他的形状组合起来：

```
a.add(new Rectangle2D.Double(...));
a.subtract(path);
...
```

**Area**类实现了**Shape**接口。可以用**draw**方法勾勒出该区域的边界，或者使用**Graphics2D**类的**fill**方法给区域的内部着色。

### API **java.awt.geom.Area**

- void add(Area other)
- void subtract(Area other)
- void intersect(Area other)
- void exclusiveOr(Area other)

对该区域和other所代表的另一个区域执行区域几何作图的操作，并且将该区域设置为执行的结果。

## 7.4 笔划

**Graphics2D**类的**draw**操作通过使用当前选定的笔划来绘制一个形状的边界。在默认的情况下，笔划是一条宽度为一个像素的实线。可以通过调用**setStroke**方法来选定不同的笔划，此时要提供一个实现了**Stroke**接口的类的对象。Java 2D API只定义了一个这样的类，即**BasicStroke**类。在本节中，我们将介绍**BasicStroke**类的功能。

你可以构建任意粗细的笔划。例如，下面的方法就绘制了一条粗细为10个像素的线条。

```
g2.setStroke(new BasicStroke(10.0F));
g2.draw(new Line2D.Double(...));
```

当一个笔划的粗细大于一个像素的宽度时，笔划的末端可采用不同的样式。图7-11显示了这些所谓的端头样式。端头样式有下面三种：

- 平头样式 (butt cap) 在笔划的末端处就结束了；
- 圆头样式 (round cap) 在笔划的末端处加了一个半圆；
- 方头样式 (square cap) 在笔划的末端处加了半个方块。

当两个较粗的笔划相遇时，有三种笔划的连接样式可供选择（参见图7-12）：

- 斜连接 (bevel join)，用一条直线将两个笔划连接起来，该直线与两个笔划之间的夹角的平分线相垂直。
- 圆连结 (round join)，延长了每个笔划，并使其带有一个圆头。
- 斜尖连接 (miter join)，通过增加一个尖峰，从而同时延长了两个笔划。

斜尖连接不适合小角度连接的线条。如果两条线连接后的角度小于斜尖连接的最小角度，那么应该使用斜连接。斜连接的使用，可以防止出现太长的尖峰。默认情况下，斜尖连接的最小角度是10度。

可以在**BasicStroke**构造器中设定这些选择，例如：

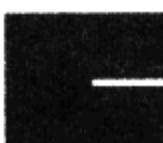
```
g2.setStroke(new BasicStroke(10.0F, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND));
g2.setStroke(new BasicStroke(10.0F, BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER,
    15.0F /* miter limit */));
```



平头样式



圆头样式



方头样式



斜连接



圆连接



斜尖连接

图7-11 笔划的端头样式

图7-12 笔划的连接样式

最后，通过设置一个虚线模式，可以指定需要使用的虚线。在程序清单7-2的程序中，可以选择一个虚线图案，构成摩斯电码中的SOS代码。虚线模式是一个float[]类型的数组，它包含了笔划中“连接 (on)”和“断开 (off)”的长度（见图7-13）。

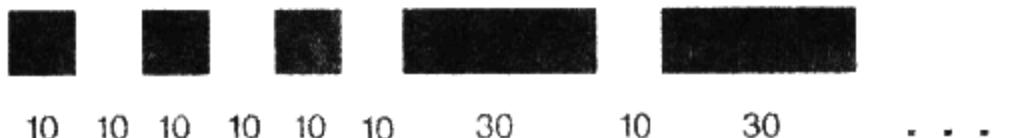


图7-13 一种虚线图案

当构建BasicStroke时，可以指定虚线模式和虚线相位（dash phase）。虚线相位用来表示每条线的虚线模式应该从哪里开始。通常情况下，应该把它的值设置为0。

```
float[] dashPattern = { 10, 10, 10, 10, 10, 10, 30, 10, 30, ... };
g2.setStroke(new BasicStroke(10.0F, BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER,
    10.0F /* miter limit */, dashPattern, 0 /* dash phase */));
```



**注意：**在虚线模式中，每一条虚线的末端都可以应用端头样式。

程序清单7-3中的程序可以设定端头样式、连接样式和虚线（见图7-14）。可以移动线段的端头，用以测试斜尖连接的最小角度：首先选定斜尖连接；然后，移动线段末端形成一个非常尖的锐角。可以看到斜尖连接变成了一个斜连接。

这个程序类似于程序清单7-1的程序。当点击一个线段的末端时，鼠标监听器就会记下操作，而鼠标动作监听器则控制对端点的拖曳操作。一组单选按钮用以表示用户选择的端头样式、连接样式以及实线或虚线。StrokePanel类的paintComponent方法构建了一个GeneralPath，它

由连接着用户可以用鼠标移动的三个点的两条线段构成。然后，它根据用户的选择构建一个 BasicStroke，最后绘制出这个路径。

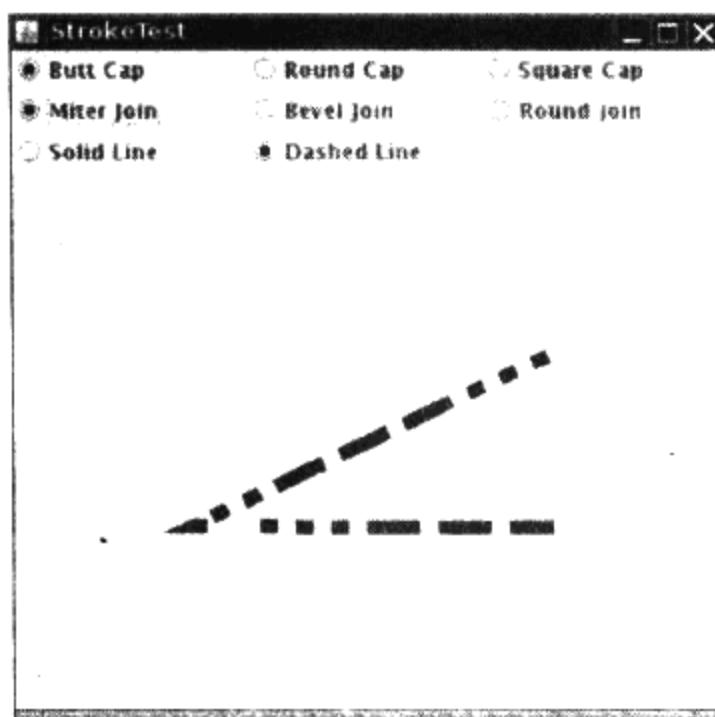


图7-14 StrokeTest程序

### 程序清单7-2 StrokeTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.awt.geom.*;
4. import javax.swing.*;
5.
6. /**
7. * This program demonstrates different stroke types.
8. * @version 1.03 2007-08-16
9. * @author Cay Horstmann
10.*/
11. public class StrokeTest
12. {
13.     public static void main(String[] args)
14.     {
15.         EventQueue.invokeLater(new Runnable()
16.         {
17.             public void run()
18.             {
19.                 JFrame frame = new StrokeTestFrame();
20.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21.                 frame.setVisible(true);
22.             }
23.         });
24.     }
25. }
26.
27. /**
28. * This frame lets the user choose the cap, join, and line style, and shows the resulting
29. * stroke.
30. */
```

```
31. class StrokeTestFrame extends JFrame
32. {
33.     public StrokeTestFrame()
34.     {
35.         setTitle("StrokeTest");
36.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
37.
38.         canvas = new StrokeComponent();
39.         add(canvas, BorderLayout.CENTER);
40.
41.         buttonPanel = new JPanel();
42.         buttonPanel.setLayout(new GridLayout(3, 3));
43.         add(buttonPanel, BorderLayout.NORTH);
44.
45.         ButtonGroup group1 = new ButtonGroup();
46.         makeCapButton("Butt Cap", BasicStroke.CAP_BUTT, group1);
47.         makeCapButton("Round Cap", BasicStroke.CAP_ROUND, group1);
48.         makeCapButton("Square Cap", BasicStroke.CAP_SQUARE, group1);
49.
50.         ButtonGroup group2 = new ButtonGroup();
51.         makeJoinButton("Miter Join", BasicStroke.JOIN_MITER, group2);
52.         makeJoinButton("Bevel Join", BasicStroke.JOIN_BEVEL, group2);
53.         makeJoinButton("Round Join", BasicStroke.JOIN_ROUND, group2);
54.
55.         ButtonGroup group3 = new ButtonGroup();
56.         makeDashButton("Solid Line", false, group3);
57.         makeDashButton("Dashed Line", true, group3);
58.     }
59.
60. /**
61. * Makes a radio button to change the cap style.
62. * @param label the button label
63. * @param style the cap style
64. * @param group the radio button group
65. */
66. private void makeCapButton(String label, final int style, ButtonGroup group)
67. {
68.     // select first button in group
69.     boolean selected = group.getButtonCount() == 0;
70.     JRadioButton button = new JRadioButton(label, selected);
71.     buttonPanel.add(button);
72.     group.add(button);
73.     button.addActionListener(new ActionListener()
74.     {
75.         public void actionPerformed(ActionEvent event)
76.         {
77.             canvas.setCap(style);
78.         }
79.     });
80. }
81.
82. /**
83. * Makes a radio button to change the join style.
84. * @param label the button label
85. * @param style the join style
86. * @param group the radio button group
87. */
```

```
88.     private void makeJoinButton(String label, final int style, ButtonGroup group)
89.     {
90.         // select first button in group
91.         boolean selected = group.getButtonCount() == 0;
92.         JRadioButton button = new JRadioButton(label, selected);
93.         buttonPanel.add(button);
94.         group.add(button);
95.         button.addActionListener(new ActionListener()
96.             {
97.                 public void actionPerformed(ActionEvent event)
98.                 {
99.                     canvas.setJoin(style);
100.                }
101.            });
102.        }
103.
104.    /**
105.     * Makes a radio button to set solid or dashed lines
106.     * @param label the button label
107.     * @param style false for solid, true for dashed lines
108.     * @param group the radio button group
109.     */
110.    private void makeDashButton(String label, final boolean style, ButtonGroup group)
111.    {
112.        // select first button in group
113.        boolean selected = group.getButtonCount() == 0;
114.        JRadioButton button = new JRadioButton(label, selected);
115.        buttonPanel.add(button);
116.        group.add(button);
117.        button.addActionListener(new ActionListener()
118.            {
119.                public void actionPerformed(ActionEvent event)
120.                {
121.                    canvas.setDash(style);
122.                }
123.            });
124.        }
125.
126.    private StrokeComponent canvas;
127.    private JPanel buttonPanel;
128.
129.    private static final int DEFAULT_WIDTH = 400;
130.    private static final int DEFAULT_HEIGHT = 400;
131. }
132.
133. /**
134.  * This component draws two joined lines, using different stroke objects, and allows the
135.  * user to drag the three points defining the lines.
136. */
137. class StrokeComponent extends JComponent
138. {
139.     public StrokeComponent()
140.     {
141.         addMouseListener(new MouseAdapter()
142.             {
143.                 public void mousePressed(MouseEvent event)
144.                 {
```

```
145.         Point p = event.getPoint();
146.         for (int i = 0; i < points.length; i++)
147.         {
148.             double x = points[i].getX() - SIZE / 2;
149.             double y = points[i].getY() - SIZE / 2;
150.             Rectangle2D r = new Rectangle2D.Double(x, y, SIZE, SIZE);
151.             if (r.contains(p))
152.             {
153.                 current = i;
154.                 return;
155.             }
156.         }
157.     }
158.
159.     public void mouseReleased(MouseEvent event)
160.     {
161.         current = -1;
162.     }
163. });
164.
165. addMouseMotionListener(new MouseMotionAdapter()
166. {
167.     public void mouseDragged(MouseEvent event)
168.     {
169.         if (current == -1) return;
170.         points[current] = event.getPoint();
171.         repaint();
172.     }
173. });
174.
175. points = new Point2D[3];
176. points[0] = new Point2D.Double(200, 100);
177. points[1] = new Point2D.Double(100, 200);
178. points[2] = new Point2D.Double(200, 200);
179. current = -1;
180. width = 8.0F;
181. }
182.
183. public void paintComponent(Graphics g)
184. {
185.     Graphics2D g2 = (Graphics2D) g;
186.     GeneralPath path = new GeneralPath();
187.     path.moveTo((float) points[0].getX(), (float) points[0].getY());
188.     for (int i = 1; i < points.length; i++)
189.         path.lineTo((float) points[i].getX(), (float) points[i].getY());
190.     BasicStroke stroke;
191.     if (dash)
192.     {
193.         float miterLimit = 10.0F;
194.         float[] dashPattern = { 10F, 10F, 10F, 10F, 10F, 10F, 30F, 10F, 30F, 10F, 30F, 10F,
195.             10F, 10F, 10F, 10F, 10F, 30F };
196.         float dashPhase = 0;
197.         stroke = new BasicStroke(width, cap, join, miterLimit, dashPattern, dashPhase);
198.     }
199.     else stroke = new BasicStroke(width, cap, join);
200.     g2.setStroke(stroke);
```

```
201     .g2.draw(path);
202 }
203
204 /**
205 * Sets the join style.
206 * @param j the join style
207 */
208 public void setJoin(int j)
209 {
210     join = j;
211     repaint();
212 }
213
214 /**
215 * Sets the cap style.
216 * @param c the cap style
217 */
218 public void setCap(int c)
219 {
220     cap = c;
221     repaint();
222 }
223
224 /**
225 * Sets solid or dashed lines
226 * @param d false for solid, true for dashed lines
227 */
228 public void setDash(boolean d)
229 {
230     dash = d;
231     repaint();
232 }
233
234 private Point2D[] points;
235 private static int SIZE = 10;
236 private int current;
237 private float width;
238 private int cap;
239 private int join;
240 private boolean dash;
241 }
```

### API **java.awt.Graphics2D 1.2**

- **void setStroke(Stroke s)**

将该图形上下文的笔划设置为实现了**Stroke**接口的给定对象。

### API **java.awt.BasicStroke 1.2**

- **BasicStroke(float width)**
- **BasicStroke(float width, int cap, int join)**
- **BasicStroke(float width, int cap, int join, float miterlimit)**
- **BasicStroke(float width, int cap, int join, float miterlimit, float[] dash)**

```
dash, float dashPhase)
```

用给定的属性构建一个笔划对象。

参数: width

画笔的宽度

cap

端头样式, 它是CAP\_BUTT、CAP\_ROUND和CAP\_SQUARE三种样式中的一个。

join

连接样式, 它是JOIN\_BEVEL、JOIN\_MITER和JOIN\_ROUND三种样式中的一个。

miterlimit

用度数表示的角度, 如果小于这个角度, 斜尖连接将呈现为斜连接。

dash

虚线笔划的填充部分和空白部分交替出现的一组长度。

dashPhase

虚线模式的“相位”; 位于笔划起始点前面的这段长度被假设为已经应用了该虚线模式。

## 7.5 着色

当填充一个形状的时候, 该形状的内部就上了颜色。使用setPaint方法, 把颜色的样式设定为一个实现了Paint接口的类的对象。Java 2D API提供了三个这样的类:

- Color类实现了Paint接口。如果要用单色填充形状, 只需要用Color对象调用setPaint方法即可, 例如:

```
g2.setPaint(Color.red);
```

- GradientPaint类通过在两个给定的颜色值之间进行渐变, 从而改变使用的颜色(参见图7-15)。

- TexturePaint类用一个图像重复地对一个区域进行着色(见图7-16)。

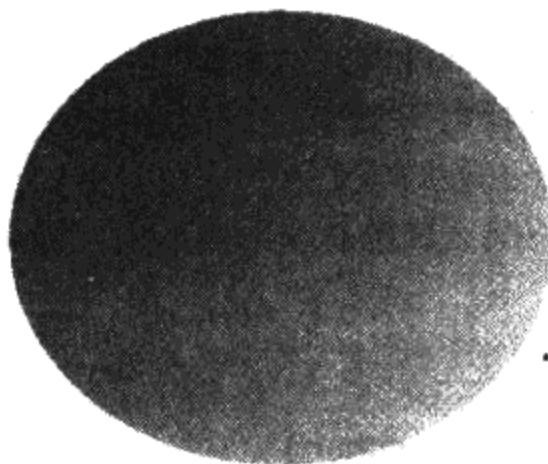


图7-15 渐变着色

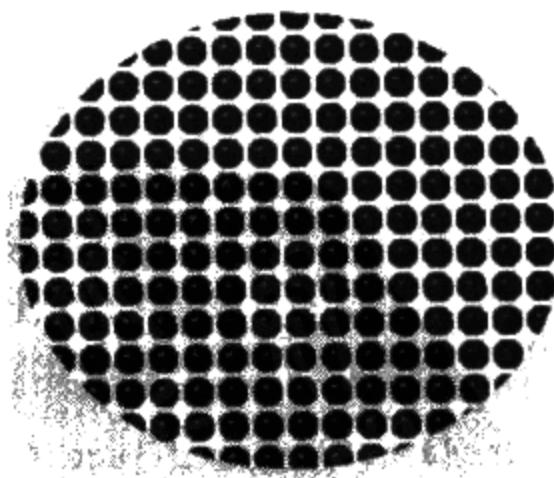


图7-16 纹理着色

可以通过指定两个点以及在这两个点上想使用的颜色来构建一个GradientPaint对象, 即:

```
g2.setPaint(new GradientPaint(p1, Color.RED, p2, Color.YELLOW));
```

上面语句将沿着连接两个点之间的直线对颜色进行渐变, 而沿着与该连接线垂直方向上的线条颜色则是不变的。超过线条端点的各个点被赋予端点上的颜色。

另外，如果调用的GradientPaint构造器的cyclic参数的值为true，即：

```
g2.setPaint(new GradientPaint(p1, Color.RED, p2, Color.YELLOW, true));
```

那么颜色将循环变换，并且在端点之外仍然保持这种变换。

如果要构建一个TexturePaint对象，需要制定一个BufferedImage和一个锚位矩形。

```
g2.setPaint(new TexturePaint(bufferedImage, anchorRectangle));
```

在本章后面部分详细讨论图像时，我们再介绍BufferedImage类。获取缓冲图像最简单的方式就是读入图像文件：

```
bufferedImage = ImageIO.read(new File("blue-ball.gif"));
```

锚位矩形在x和y方向上将不断地重复延伸，使之平铺到整个坐标平面。图像可以伸缩，以便纳入该锚位，然后复制到每一个平铺显示区中。

#### **API java.awt.Graphics2D 1.2**

- void setPaint(Paint s)

将图形上下文的着色设置为实现了Paint接口的给定对象。

#### **API java.awt.GradientPaint 1.2**

- GradientPaint(float x1, float y1, Color color1, float x2, float y2, Color color2)
- GradientPaint(float x1, float y1, Color color1, float x2, float y2, Color color2, boolean cyclic)
- GradientPaint(Point2D p1, Color color1, Point2D p2, Color color2)
- GradientPaint(Point2D p1, Color color1, Point2D p2, Color color2, boolean cyclic)

构建一个渐变着色的对象，以便用颜色来填充各个形状，其中，起始点的颜色为color1，结束点的颜色为color2，而两个点之间的颜色则是以线性的方式渐变。沿着连接起始点和结束点之间的线条相垂直的方向上的线条颜色是恒定不变的。在默认的情况下，渐变着色不是循环变换的。也就是说，起始点和结束点之外的各个点的颜色是分别与起始点和结束点的颜色相同的。如果渐变着色是循环的，那么颜色是连续变换的，首先返回到起始点的颜色，然后在两个方向上无限地重复。

#### **API java.awt.TexturePaint 1.2**

- TexturePaint(BufferedImage texture, Rectangle2D anchor)

建立纹理着色对象。用于定义着色的平铺空间的锚位矩形。该矩形在x和y方向上不断地重复延伸，纹理图像则被缩放，以便填充每个平铺空间。

## 7.6 坐标变换

假设要绘制一个对象，比如汽车。从制造商的规格说明书中可以了解到汽车的高度、轴距和整个车身的长度。如果设定了每米的像素个数，当然可以计算出所有像素的位置。但是，可

以使用更加容易的方法：让图形上下文来执行这种转换。

```
g2.scale(pixelsPerMeter, pixelsPerMeter);
g2.draw(new Line2D.Double(coordinates in meters)); // converts to pixels and draws scaled line
```

Graphics2D类的scale方法可以将图形上下文中的坐标变换设置为一个比例变换。这种变换能够将用户坐标（用户设定的单元）转换成设备坐标（pixel，即像素）。图7-17显示了如何进行这种变换的方法。

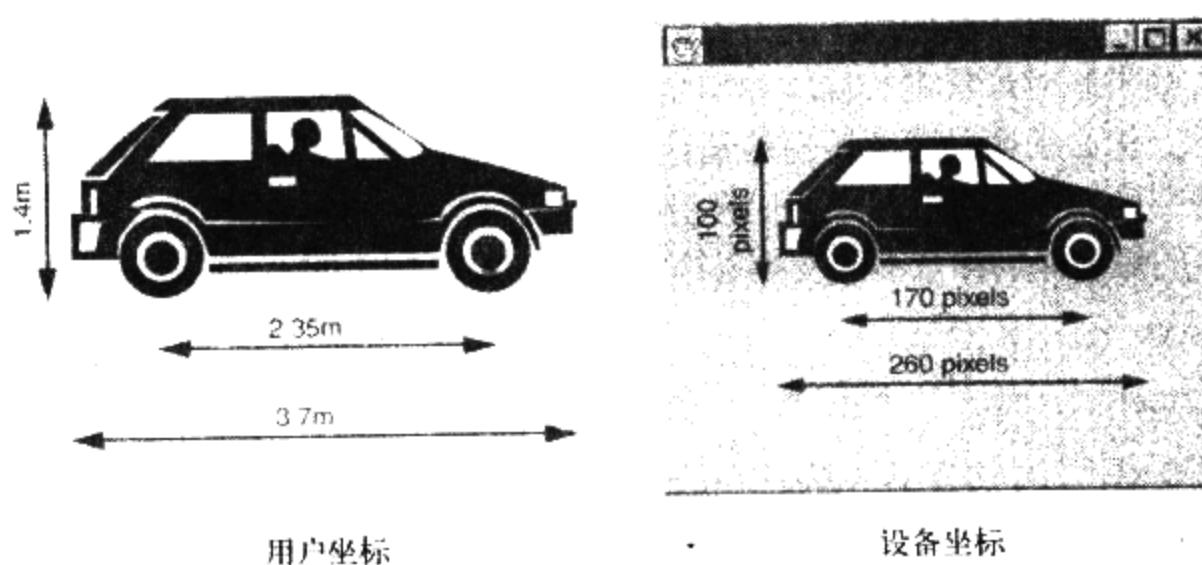


图7-17 用户坐标与设备坐标

坐标变换在实际应用中非常有用，可以使用方便的坐标值进行各种操作，图形上下文则负责执行将坐标值转换成像素的复杂工作。

这里有四种基本的变换：

- 比例缩放：放大和缩小从一个固定点出发的所有距离。
- 旋转：环绕着一个固定中心旋转所有点。
- 平移：将所有的点移动一个固定量。
- 切变：使一个线条固定不变，再按照与该固定线条之间的距离，成比例地将与该线条平行的各个线条“滑动”一个距离量。

图7-18显示了对一个单位的正方形进行这四种基本变换操作的效果。

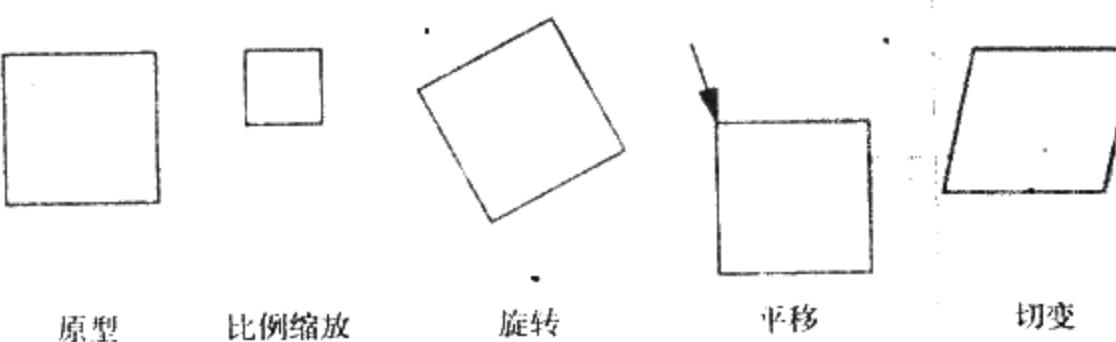


图7-18 基本的变换

`Graphics2D`类的`scale`、`rotate`、`translate`和`shear`等方法用以将图形上下文中的坐标变换设置成为以上这些基本变换中的一种。

可以组合不同的变换操作。例如，可能想对图形进行旋转和两倍尺寸放大的操作，这时，可以同时提供旋转和比例缩放的变换。

```
g2.rotate(angle);
g2.scale(2, 2);
g2.draw(...);
```

在这种情况下，变换方法的顺序是无关紧要的。然而，在大多数变换操作中，顺序却是很重要的。例如，如果想对形状进行旋转和切变操作，那么两种变换操作的不同执行序列，将会产生不同的图形。你必须明确想要得到的是什么样的图形，图形上下文将按照你所提供的相反顺序来应用这些变换操作。也就是说，你最后提供的方法会被最先使用。

可以根据你的需要提供任意多的变换操作。例如，假设你提供了下面这个变换操作序列：

```
g2.translate(x, y);
g2.rotate(a);
g2.translate(-x, -y);
```

最后一个变换操作（它是第一个被应用的）将把某个形状从点 $(x, y)$ 移动到原点，第二个卷积变换将使该形状围绕着原点旋转一个角度 $a$ ，最后一个变换方法又重新把该形状从原点移动到点 $(x, y)$ 处。总体效果就是该形状围绕着中心点 $(x, y)$ 进行了一次旋转（参见图7-19）。围绕着原点之外的任意点进行旋转是一个很常见的操作，所以我们采用下面的快捷方法：

```
g2.rotate(a, x, y);
```

如果对矩阵理论有所了解，那么就会知道所有操作（诸如旋转、平移、缩放、切变）和由这些操作组合起来的操作都能够以如下矩阵变换的形式表示出来：

$$\begin{bmatrix} x_{\text{new}} \\ y_{\text{new}} \\ 1 \end{bmatrix} = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

这种变换称为仿射变换（affine transformation）。Java 2D API中的`AffineTransform`类就是用于描述这种变换的。如果你知道某个特定变换矩阵的组成元素，就可以用下面的方法直接构造它：

```
AffineTransform t = new AffineTransform(a, b, c, d, e, f);
```

另外，工厂方法`getRotateInstance`、`getScaleInstance`、`getTranslateInstance`和`getShearInstance`能够构建出表示相应变换类型的数组。例如，调用下面的方法：

```
t = AffineTransform.getScaleInstance(2.0F, 0.5F);
```

将返回一个与下面这个数组相一致的变换。

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

最后，实例方法`setToRotation`、`setToScale`、`setToTranslation`和`setToShear`用于将

变换对象设置为一个新的类型。下面是一个例子：

```
t.setToRotation(angle); // sets t to a rotation
```

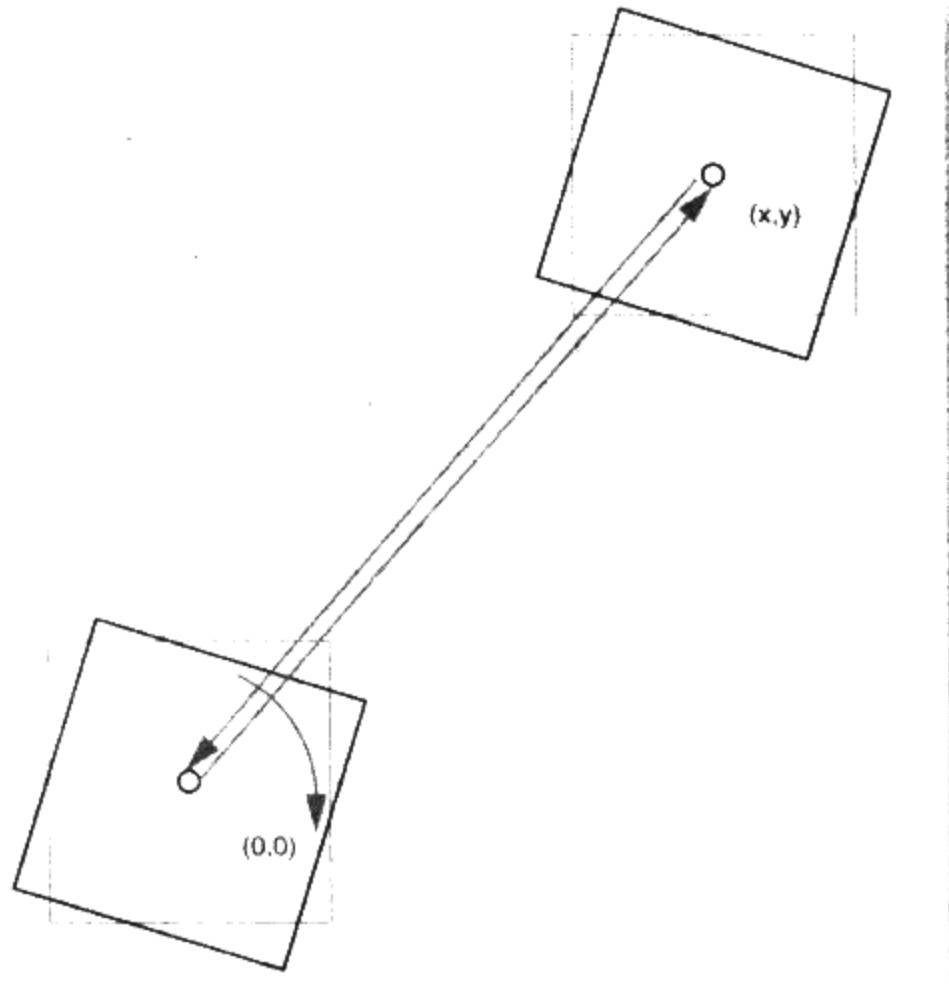


图7-19 组合变换操作的应用

可以把图形上下文的坐标变换设置为一个`AffineTransform`对象：

```
g2.setTransform(t); // replaces current transformation
```

不过，在实际运用中，不要调用`setTransform`操作，因为它会取代图形上下文中可能存在的任何现有的变换。例如，一个用以横向打印的图形上下文已经有了一个90°的卷积变换，如果调用方法`setTransfrom`，就会删除这样的旋转操作。可以调用`transform`方法作为替代方案。

```
g2.transform(t); // composes current transformation with t
```

它把现有的变换操作和新的`AffineTransform`对象组合起来。

如果只想临时应用某个变换操作，那么应该首先获得旧的变换操作，然后和新的变换操作组合起来，最后当你完成操作时，再还原旧的变换操作。

```
AffineTransform oldTransform = g2.getTransform(); // save old transform  
g2.transform(t); // apply temporary transform // now draw on g2  
g2.setTransform(oldTransform); // restore old transform
```



`java.awt.geom.AffineTransform 1.2`

- `AffineTransform(double a, double b, double c, double d, double e, double f)`

- `AffineTransform(float a, float b, float c, float d, float e, float f)`  
用下面的矩阵构建该仿射变换。

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

- `AffineTransform(double[] m)`

- `AffineTransform(float[] m)`

用下面的矩阵构建该仿射变换。

$$\begin{bmatrix} m[0] & m[2] & m[4] \\ m[1] & m[3] & m[5] \\ 0 & 0 & 1 \end{bmatrix}$$

- `static AffineTransform getRotateInstance(double a)`

创建一个围绕原点，旋转角度为a（弧度）的卷积变换。其变换矩阵是：

$$\begin{bmatrix} \cos(a) & -\sin(a) & 0 \\ \sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

如果a在0到 $\pi/2$ 之间，那么图形将沿着x轴正半轴向y轴正半轴的方向旋转。

- `static AffineTransform getRotateInstance(double a, double x, double y)`

创建一个围绕点(x,y)，旋转角度为a(弧度)的卷积变换。

- `static AffineTransform getScaleInstance(double sx, double sy)`

创建一个比例缩放变换。x轴缩放幅度为sx；y轴缩放幅度为sy。其变换矩阵是：

$$\begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- `static AffineTransform getShearInstance(double shx, double shy)`

创建一个切变变换。x轴切变shx；y轴切变shy。其变换矩阵是：

$$\begin{bmatrix} 1 & shx & 0 \\ shy & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- `static AffineTransform getTranslateInstance(double tx, double ty)`

创建一个平移变换。x轴平移tx；y轴平移ty。其变换矩阵是：

$$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$$

- `void setToRotation(double a)`

- void setToRotation(double a, double x, double y)
- void setToScale(double sx, double sy)
- void setToShear(double sx, double sy)
- void setToTranslation(double tx, double ty)

用给定的参数将该变换设置为一个的基本变换。如果要了解基本变换和它们的参数说明，请参见`getXxxInstance`方法。



### java.awt.Graphics2D 1.2

- void setTransform(AffineTransform t)  
以t来取代该图形上下文中现有的坐标变换。
- void transform(AffineTransform t)  
将该图形上下文的现有坐标变换和t组合起来。
- void rotate(double a)
- void rotate(double a, double x, double y)
- void scale(double sx, double sy)
- void shear(double sx, double sy)
- void translate(double tx, double ty)

将该图形上下文中现有的坐标变换和一个带有给定参数的基本变换组合起来。如果要了解基本变换和它们的参数说明，请参见`AffineTransform.getXxxInstance`方法。

## 7.7 剪切

通过在图形上下文中设置一个剪切形状，可以将所有的绘图操作限制在该剪切形状内部来进行。

```
g2.setClip(clipShape); // but see below  
g2.draw(shape); // draws only the part that falls inside the clipping shape
```

但是，在实际应用中，不应该调用这个`setClip`操作，因为它会取代图形上下文中可能存在的任何剪切形状。例如，正如在本章的后面部分所看到的那样，用于打印操作的图形上下文配有一个剪切矩形，以确保你不会在页边距上绘图。相反，你应该调用`clip`方法。

```
g2.clip(clipShape); // better
```

`clip`方法使得你所提供的新的剪切形状同现有的剪切形状相交。

如果只想临时地使用一个剪切区域的话，那么应该首先获得旧的剪切形状，然后添加新的剪切形状，最后，在完成操作时，再还原旧的剪切形状：

```
Shape oldClip = g2.getClip(); // save old clip  
g2.clip(clipShape); // apply temporary clip  
draw on g2  
g2.setClip(oldClip); // restore old clip
```

在图7-20的例子中，我们炫耀了一下剪切的功能，它绘制了一个按照复杂形状进行剪切的相当出色的线条图案，即一组字符的轮廓。



图7-20 按照字母形状剪切出的线条图案

如果要获得字符的外形，需要一个字体渲染上下文（font render context）。请使用Graphics2D类的getFontRenderContext方法。

```
FontRenderContext context = g2.getFontRenderContext();
```

接着，使用某个字符串、某种字体和字体渲染上下文来创建一个TextLayout对象。

```
TextLayout layout = new TextLayout("Hello", font, context);
```

这个文本布局对象用于描述由特定字体渲染上下文所渲染的一个字符序列的布局。这种布局依赖于字体渲染上下文，相同的字符在屏幕上或者打印机上看起来会有不同的显示。

对我们当前的应用来说，更重要的是，getOutline方法将会返回一个Shape对象，这个Shape对象用以描述在文本布局中的各个字符轮廓的形状。字符轮廓的形状从原点(0,0)开始，这并不适合大多数的绘图操作。因此，必须为getOutline操作提供一个仿射变换操作，以便设定希望的字体轮廓所显示的位置。

```
AffineTransform transform = AffineTransform.getTranslateInstance(0, 100);
Shape outline = layout.getOutline(transform);
```

接着，我们把字体的轮廓附加给剪切的形状。

```
GeneralPath clipShape = new GeneralPath();
clipShape.append(outline, false);
```

最后，我们设置剪切形状，并且绘制一组线条。线条仅仅在字符边界的内部显示。

```
g2.setClip(clipShape);
Point2D p = new Point2D.Double(0, 0);
for (int i = 0; i < NINES; i++)
{
    double x = . . .;
    double y = . . .;
    Point2D q = new Point2D.Double(x, y);
    g2.draw(new Line2D.Double(p, q)); // lines are clipped
}
```

在程序清单7-8中可以看到完整的代码。



### java.awt.Graphics 1.0

- void setClip(Shape s) 1.2

将当前的剪切形状设置为形状s。

- Shape getClip() 1.2

返回当前的剪切形状。

**API** **java.awt.Graphics2D 1.2**

- `void clip(Shape s)`

将当前的剪切形状和形状s相交。

- `FontRenderContext getFontRenderContext()`

返回一个构建TextLayout对象所必需的字体渲染上下文。

**API** **java.awt.font.TextLayout 1.2**

- `TextLayout(String s, Font f, FontRenderContext context)`

根据给定的字符串、字体和字体渲染上下文来构建文本布局对象。方法中使用字体渲染上下文来获取特定装置的字体属性。

- `float getAdvance()`

返回该文本布局的宽度。

- `float getAscent()`

- `float getDescent()`

返回基准线上方和下方该文本布局的高度。

- `float getLeading()`

返回该文本布局使用的字体中相邻两行之间的距离。

## 7.8 透明与组合

在标准的RGB颜色模型中，每种颜色都是由它的红、绿和蓝这三种成分来描述的。但是，用它来描述透明或者部分透明的图像区域也是非常方便的。当你将一个图像置于现有图像的上面时，透明的像素完全不会遮挡它们下面的像素，而部分透明的像素则与它们下面的像素相混合。图7-21显示了一个部分透明的矩形和一个图像相重叠所产生的效果。仍然可以透过矩形看到该图像的细节。

在Java 2D API中，透明是由一个透明度通道(alpha channel)来描述的。每个像素，除了它的红、绿和蓝色部分外，还有一个介于0(完全透明)和1(部分透明)之间的透明度(alpha)值。例如，图7-21中的矩形填充了一种淡黄色，透明度为50%：

```
new Color(0.7F, 0.7F, 0.0F, 0.5F);
```

现在让我们看一看如果将两个形状重叠在一起时将会出现什么情况。必须把源像素和目标像素的颜色和透明度值混和或者组合起来。从事计算机图形学研究的Porter和Duff已经阐明了在这个混和过程中的12种可能的组合原则，Java 2D API实现了所有的这些原则。在继续介绍这个问题之前，需要指出的是，这些原则中只有两个原则有实际的意义。如果你发现这些原则晦涩难懂或者难以搞清楚，那么只使用SRC\_OVER原则就可以了。它是Graphics2D对象的默认原则，并且它的直观效果最佳。

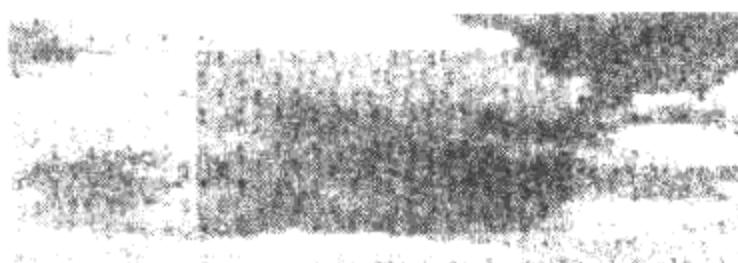


图7-21 一个部分透明的矩形和一个图像相重叠时所显示的效果

下面是这些规则的原理。假设你有了一个透明度值为 $a_s$ 的源像素，在该图像中，已经存在了一个透明度值为 $a_d$ 的目标像素，你想把两个像素组合起来。图7-22的示意图显示了如何设计一个像素的组合原则。

Porter和Duff将透明度值作为像素颜色将被使用的概率。从源像素的角度来看，存在一个概率 $a_s$ ，它是源像素颜色被使用的概率；还存在一个概率 $1 - a_s$ ，它是不使用该像素颜色的概率。同样的原则也适用于目标像素。当组合颜色时，我们假设源像素的概率和目标像素的概率是不相关的。那么正如图7-22所示，有四种组合情况。如果源像素想要使用它的颜色，而目标像素不使用，那么很自然的，我们就只使用源像素的颜色。这也是为什么右上角的矩形框用“S”来标志的原因了，这种情况的概率为 $a_s \cdot (1 - a_d)$ 。同理，左下角的矩形框用“D”来标志。如果源像素和目标像素都想选择自己的颜色，那该怎么办才好呢？这里就可以应用Porter-Duff原则了。如果我们认为源像素比较重要，那么我们也会在右下角的矩形框内标志上一个“S”。这个规则被称为SRC\_OVER。在这个规则中，我们赋予源像素颜色的权值 $a_s$ ，目标像素颜色的权值为 $(1 - a_s) \cdot a_d$ 。

这样产生的视觉效果是源像素与目标像素相混和的结果，并且优先选择给定的源像素的颜色。特别是，如果 $a_s$ 为1，那么根本就不用考虑目标像素的颜色。如果 $a_s$ 为0，那么源像素将是完全透明的，而目标像素颜色则是不变的。

还有其他的规则，可以根据置于概率示意图各个框中的字母来理解这些规则的概念。表7-1和图7-23显示了Java 2D API支持的所有这些规则。图7-23中的各个图像显示了当你使用透明度值为0.75的矩形源区域和透明度值为1.0的椭圆目标区域组合时，所显示的各种组合效果。

表7-1 Porter-Duff 组合规则

| 规 则      | 解 释                            |
|----------|--------------------------------|
| CLEAR    | 源像素清除目标像素                      |
| SRC      | 源像素覆盖目标像素和空像素                  |
| DST      | 源像素不影响目标像素                     |
| SRC_OVER | 源像素和目标像素混和，并且覆盖空像素             |
| DST_OVER | 源像素不影响目标像素，并且不覆盖空像素            |
| SRC_IN   | 源像素覆盖目标像素                      |
| SRC_OUT  | 源像素清除目标像素，并且覆盖空像素              |
| DST_IN   | 源像素的透明度值修改目标像素的透明度值            |
| DST_OUT  | 源像素的透明度值补充修改目标像素的透明度值          |
| SRC_ATOP | 源像素和目标像素相混和                    |
| DST_ATOP | 源像素的透明度值修改目标像素的透明度值。源像素覆盖空像素   |
| XOR      | 源像素的透明度值补充修改目标像素的透明度值。源像素覆盖空像素 |

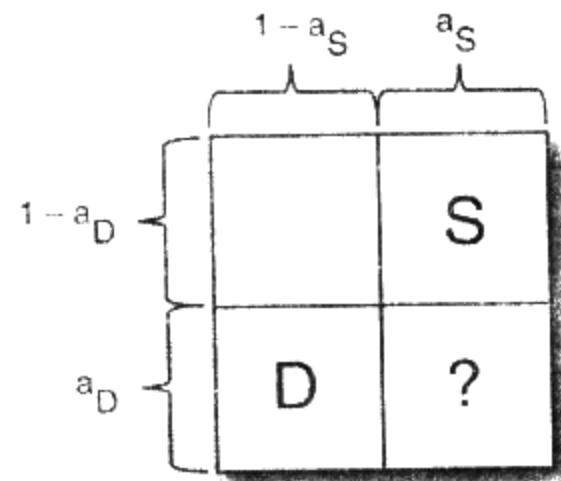


图7-22 设计一个像素组合的原则

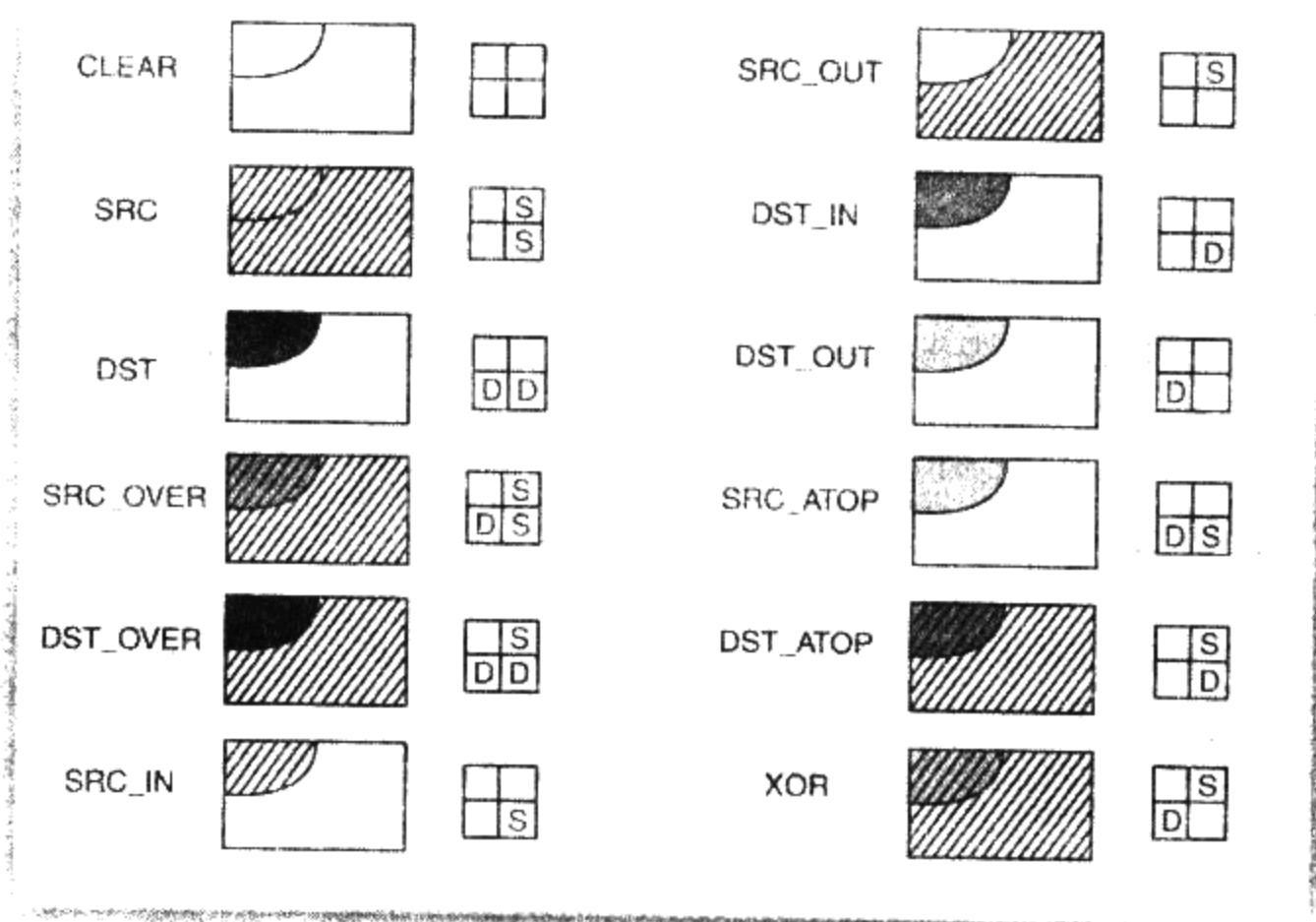


图7-23 Porter-Duff 组合规则

如你所见，大多数规则并不是非常有用。例如，**DST\_IN**规则就是一个极端的例子。它根本不考虑源像素颜色，但是却使用了源像素的透明度值来影响目标像素。**SRC**规则可能是有用的，它强制使用源像素颜色，而且去除了与目标像素相混和的特性。

如果要了解更多的关于Porter-Duff规则的信息，请参阅Foley、Dam和Feiner等撰写的《Computer Graphics: Principles and Practice, Second Edition》(计算机图形学：原理与应用，第2版)。

你可以使用**Graphics2D**类的**setComposite**方法传入一个实现了**Composite**接口的类的对象。Java 2D API提供了**AlphaComposite**这样的一个类，它实现了图7-23中的所有的Porter-Duff规则。

**AlphaComposite**类的工厂方法**getInstance**用来产生**AlphaComposite**对象，此时需要为将要使用的源像素提供规则和透明度值。例如，可以考虑使用下面的代码：

```
int rule = AlphaComposite.SRC_OVER;
float alpha = 0.5f;
g2.setComposite(AlphaComposite.getInstance(rule, alpha));
g2.setPaint(Color.blue);
g2.fill(rectangle);
```

这时，矩形将使用蓝色和值为0.5的透明度进行着色。因为该组合规则是**SRC\_OVER**，所以它透明地置于现有图像的上面。

程序清单7-3中的程序深入地研究了这些组合规则。可以从组合框中选择一个规则，调节滑动条来设置**AlphaComposite**对象的透明度值。

此外，对每一条规则该程序都显示了一条文字描述。请注意，描述是根据组合规则表计算得出来的。例如，第二行中的“DS”表示的就是“与目标像素相混和”。

该程序有一个重要的缺陷：它不能保证和屏幕相对应的图形上下文具有一个透明通道。（实际上，它通常没有这个透明通道）。当像素被放到没有透明通道的目标像素之上的时候，该像素的颜色与像素的透明度值相乘，然后透明度值被丢弃。因为许多Porter-Duff规则都使用目标像素的透明度值，因此目标像素的透明通道是很重要的。由于这个原因，我们使用了一个采用ARGB颜色模型的缓存图像来组合各种形状。在图像被组合后，我们就将产生的图像在屏幕上绘制出来。

```
BufferedImage image = new BufferedImage(getWidth(), getHeight(), BufferedImage.TYPE_INT_ARGB);
Graphics2D gImage = image.createGraphics();
// now draw to gImage
g2.drawImage(image, null, 0, 0);
```

程序清单7-3展示了整个程序的代码，图7-24所示是屏幕显示。当运行这个程序的时候，从左到右地移动alpha滑动条来观察所产生的组合形状的效果。特别是，请注意DST\_IN与DST\_OUT规则之间惟一的差别，那就是，当你改变源像素的透明度值时，目标（！）颜色将会发生什么样的变化。

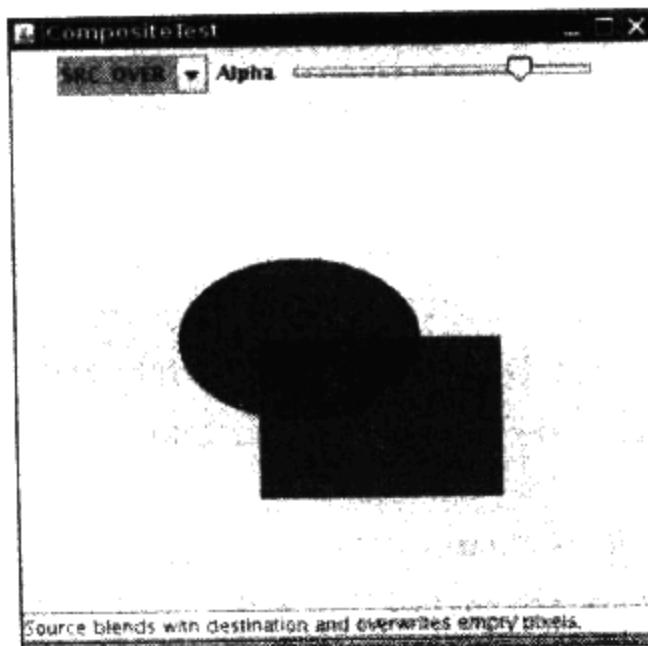


图7-24 CompositeTest 程序运行的结果

### 程序清单7-3 CompositeTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.awt.image.*;
4. import java.awt.geom.*;
5. import javax.swing.*;
6. import javax.swing.event.*;
7.
8. /**
9. * This program demonstrates the Porter-Duff composition rules.
10. * @version 1.03 2007-08-16
11. * @author Cay Horstmann
12. */
```

```
13. public class CompositeTest
14. {
15.     public static void main(String[] args)
16.     {
17.         EventQueue.invokeLater(new Runnable()
18.         {
19.             public void run()
20.             {
21.                 JFrame frame = new CompositeTestFrame();
22.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23.                 frame.setVisible(true);
24.             }
25.         });
26.     }
27. }
28.
29. /**
30. * This frame contains a combo box to choose a composition rule, a slider to change the
31. * source alpha channel, and a component that shows the composition.
32. */
33. class CompositeTestFrame extends JFrame
34. {
35.     public CompositeTestFrame()
36.     {
37.         setTitle("CompositeTest");
38.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
39.
40.         canvas = new CompositeComponent();
41.         add(canvas, BorderLayout.CENTER);
42.
43.         ruleCombo = new JComboBox(new Object[] { new Rule("CLEAR", " ", " "),
44.             new Rule("SRC", " S", " S"), new Rule("DST", " ", "DD"),
45.             new Rule("SRC_OVER", " S", "DS"), new Rule("DST_OVER", " S", "DD"),
46.             new Rule("SRC_IN", " ", " S"), new Rule("SRC_OUT", " S", " "),
47.             new Rule("DST_IN", " ", " D"), new Rule("DST_OUT", " ", "D "),
48.             new Rule("SRC_ATOP", " ", "DS"), new Rule("DST_ATOP", " S", " D"),
49.             new Rule("XOR", " S", "D ") });
50.         ruleCombo.addActionListener(new ActionListener()
51.         {
52.             public void actionPerformed(ActionEvent event)
53.             {
54.                 Rule r = (Rule) ruleCombo.getSelectedItem();
55.                 canvas.setRule(r.getValue());
56.                 explanation.setText(r.getExplanation());
57.             }
58.         });
59.
60.         alphaSlider = new JSlider(0, 100, 75);
61.         alphaSlider.addChangeListener(new ChangeListener()
62.         {
63.             public void stateChanged(ChangeEvent event)
64.             {
65.                 canvas.setAlpha(alphaSlider.getValue());
66.             }
67.         });
68.         JPanel panel = new JPanel();
```

```
69.     panel.add(ruleCombo);
70.     panel.add(new JLabel("Alpha"));
71.     panel.add(alphaSlider);
72.     add(panel, BorderLayout.NORTH);
73.
74.     explanation = new JTextField();
75.     add(explanation, BorderLayout.SOUTH);
76.
77.     canvas.setAlpha(alphaSlider.getValue());
78.     Rule r = (Rule) ruleCombo.getSelectedItem();
79.     canvas.setRule(r.getValue());
80.     explanation.setText(r.getExplanation());
81. }
82.
83. private CompositeComponent canvas;
84. private JComboBox ruleCombo;
85. private JSlider alphaSlider;
86. private JTextField explanation;
87. private static final int DEFAULT_WIDTH = 400;
88. private static final int DEFAULT_HEIGHT = 400;
89. }
90.
91. /**
92. * This class describes a Porter-Duff rule.
93. */
94. class Rule
95. {
96.     /**
97.      * Constructs a Porter-Duff rule
98.      * @param n the rule name
99.      * @param pd1 the first row of the Porter-Duff square
100.     * @param pd2 the second row of the Porter-Duff square
101.    */
102.    public Rule(String n, String pd1, String pd2)
103.    {
104.        name = n;
105.        porterDuff1 = pd1;
106.        porterDuff2 = pd2;
107.    }
108.
109.    /**
110.     * Gets an explanation of the behavior of this rule.
111.     * @return the explanation
112.     */
113.    public String getExplanation()
114.    {
115.        StringBuilder r = new StringBuilder("Source ");
116.        if (porterDuff2.equals(" ")) r.append("clears");
117.        if (porterDuff2.equals(" S")) r.append("overwrites");
118.        if (porterDuff2.equals("DS")) r.append("blends with");
119.        if (porterDuff2.equals(" D")) r.append("alpha modifies");
120.        if (porterDuff2.equals("D ")) r.append("alpha complement modifies");
121.        if (porterDuff2.equals("DD")) r.append("does not affect");
122.        r.append(" destination");
123.        if (porterDuff1.equals(" S")) r.append(" and overwrites empty pixels");
124.        r.append(".");
125.        return r.toString();
126.    }
127.
```

```
126.     }
127.
128.     public String toString()
129.     {
130.         return name;
131.     }
132.
133.     /**
134.      * Gets the value of this rule in the AlphaComposite class
135.      * @return the AlphaComposite constant value, or -1 if there is no matching constant.
136.      */
137.     public int getValue()
138.     {
139.         try
140.         {
141.             return (Integer) AlphaComposite.class.getField(name).get(null);
142.         }
143.         catch (Exception e)
144.         {
145.             return -1;
146.         }
147.     }
148.
149.     private String name;
150.     private String porterDuff1;
151.     private String porterDuff2;
152. }
153.
154. /**
155.  * This component draws two shapes, composed with a composition rule.
156. */
157. class CompositeComponent extends JComponent
158. {
159.     public CompositeComponent()
160.     {
161.         shape1 = new Ellipse2D.Double(100, 100, 150, 100);
162.         shape2 = new Rectangle2D.Double(150, 150, 150, 100);
163.     }
164.
165.     public void paintComponent(Graphics g)
166.     {
167.         Graphics2D g2 = (Graphics2D) g;
168.
169.         BufferedImage image = new BufferedImage(getWidth(), getHeight(),
170.                                         BufferedImage.TYPE_INT_ARGB);
171.         Graphics2D gImage = image.createGraphics();
172.         gImage.setPaint(Color.red);
173.         gImage.fill(shape1);
174.         AlphaComposite composite = AlphaComposite.getInstance(rule, alpha);
175.         gImage.setComposite(composite);
176.         gImage.setPaint(Color.blue);
177.         gImage.fill(shape2);
178.         g2.drawImage(image, null, 0, 0);
179.     }
180.
181. /**
```

```

182.     * Sets the composition rule.
183.     * @param r the rule (as an AlphaComposite constant)
184.     */
185.    public void setRule(int r)
186.    {
187.        rule = r;
188.        repaint();
189.    }
190.
191.    /**
192.     * Sets the alpha of the source
193.     * @param a the alpha value between 0 and 100
194.     */
195.    public void setAlpha(int a)
196.    {
197.        alpha = (float) a / 100.0F;
198.        repaint();
199.    }
200.
201.    private int rule;
202.    private Shape shape1;
203.    private Shape shape2;
204.    private float alpha;
205. }
```

**API** **java.awt.Graphics2D 1.2**

- **void setComposite(Composite s)**

把图形上下文的组合方式设置为实现了Composite接口的给定对象。

**API** **java.awt.AlphaComposite 1.2**

- **static AlphaComposite getInstance(int rule)**
- **static AlphaComposite getInstance(int rule, float Source Alpha)**

构建一个透明度 (alpha) 值的组合对象。规则是CLEAR, SRC, SRC\_OVER, DST\_OVER, SRC\_IN, SRC\_OUT, DST\_IN, DST\_OUT, DST\_ATOP, SRC\_ATOP, XOR等值之一。

## 7.9 绘图提示

在前面的部分，已经看到了绘图过程是非常复杂的。虽然在大多数情况下Java 2D API的运行速度奇快，但是在某些情况下，你可能希望控制绘图的速度和质量之间的平衡关系。可以通过设置绘图提示来达到此目的。使用Graphics2D类的setRenderingHint方法，可以设置一条单一的绘图提示，提示的键和值是在RenderingHints类中声明的。表7-2汇总了可以使用的选项。以\_DEFAULT结尾的值表示某种特定实现将其作为性能与质量之间的良好平衡而所选择的默认值。

这些设置中最有用的是消除图形锯齿现象的技术，这种技术消除了斜线和曲线中的“锯齿”(jaggies)。正如在图7-25所见的那样，斜线必须被绘制成为一个像素的“阶梯”。特别是在低分辨率的显示屏上，你所画的线条将非常难看。但是，如果不是完整地绘制或排除每一个像素，而是在线条所覆盖的像素中，用与被覆盖区域成比例的颜色，来着色被部分覆盖的元素，那么

所产生的线条看上去要平滑的多。这种技术被称为“消除图形锯齿状”技术。当然，使用这种技术所花费的时间要长一些，因为它需要花一定的时间去计算所有这些颜色的值。

表7-2 绘图提示

| 键                       | 值  | 解 释  |
|-------------------------|--|--|
| KEY_ANTIALIASING        | VALUE_ANTIALIAS_ON<br>VALUE_ANTIALIAS_OFF<br>VALUE_ANTIALIAS_DEFAULT   | 打开或者关闭形状的消除图形锯齿状的特征  |
| KEY_TEXT_ANTIALIASING   | VALUE_TEXT_ANTIALIAS_ON<br>VALUE_TEXT_ANTIALIAS_OFF<br>VALUE_TEXT_ANTIALIAS_DEFAULT<br>VALUE_TEXT_ANTIALIAS_GASP 6<br>VALUE_TEXT_ANTIALIAS_LCD_HRGB 6<br>VALUE_TEXT_ANTIALIAS_LCD_HBGR 6<br>VALUE_TEXT_ANTIALIAS_LCD_VRGB 6<br>VALUE_TEXT_ANTIALIAS_LCD_VBGR 6 | 打开或者关闭字体的消除图形锯齿状的特征。当使用VALUE_TEXT_ANTIALIAS_GASP这个值时，会查询字体“派生表”以确定字体的某种特定字号是否应该消除图形的锯齿状。LCD值强制对某种特定显示类型进行子像素绘制 |
| KEY_FRACTIONALMETRICS   | VALUE_FRACTIONALMETRICS_ON<br>VALUE_FRACTIONALMETRICS_OFF<br>VALUE_FRACTIONALMETRICS_DEFAULT   | 打开或者关闭部分字符尺寸计算的功能。使用部分字符尺寸的计算功能，将会更好的安排字符的位置   |
| KEY_RENDERING           | VALUE_RENDER_QUALITY<br>VALUE_RENDER_SPEED<br>VALUE_RENDER_DEFAULT   | 当其可用时，选定相应的绘图算法，以便获得更高的质量或速度   |
| KEY_STROKE_CONTROL 1.3  | VALUE_STROKE_NORMALIZE<br>VALUE_STROKE_PURE<br>VALUE_STROKE_DEFAULT  | 选定笔划的位置是由图形加速器（它也许会将笔划的位置向上调整半个像素）控制，还是由强制笔划穿越像素中心的“纯”规则计算出来   |
| KEY_DITHERING           | VALUE_DITHER_ENABLE<br>VALUE_DITHER_DISABLE<br>VALUE_DITHER_DEFAULT  | 打开或者关闭颜色的浓淡处理功能。通过绘制相似颜色的许多像素组，浓淡处理功能就可以确定颜色的近似值。（注意，消除锯齿功能可能会干涉浓淡处理功能）  |
| KEY_ALPHA_INTERPOLATION | VALUE_ALPHA_INTERPOLATION_QUALITY<br>VALUE_ALPHA_INTERPOLATION_SPEED<br>VALUE_ALPHA_INTERPOLATION_DEFAULT  | 打开或者关闭透明度（alpha）值组合的精确计算功能。  |
| KEY_COLOR_RENDERING     | VALUE_COLOR_RENDER_QUALITY<br>VALUE_COLOR_RENDER_SPEED<br>VALUE_COLOR_RENDER_DEFAULT   | 选定颜色渲染的质量或速度。只有在使用了不同的颜色空间时，才会涉及此问题  |
| KEY_INTERPOLATION       | VALUE_INTERPOLATION_NEAREST_NEIGHBOR<br>VALUE_INTERPOLATION_BILINEAR<br>VALUE_INTERPOLATION_BICUBIC  | 当对形状进行缩放或者旋转操作时，为像素的插换选择一个规则   |

例如，下面的代码说明了应该如何请求使用消除图形锯齿状技术。

```
g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
```

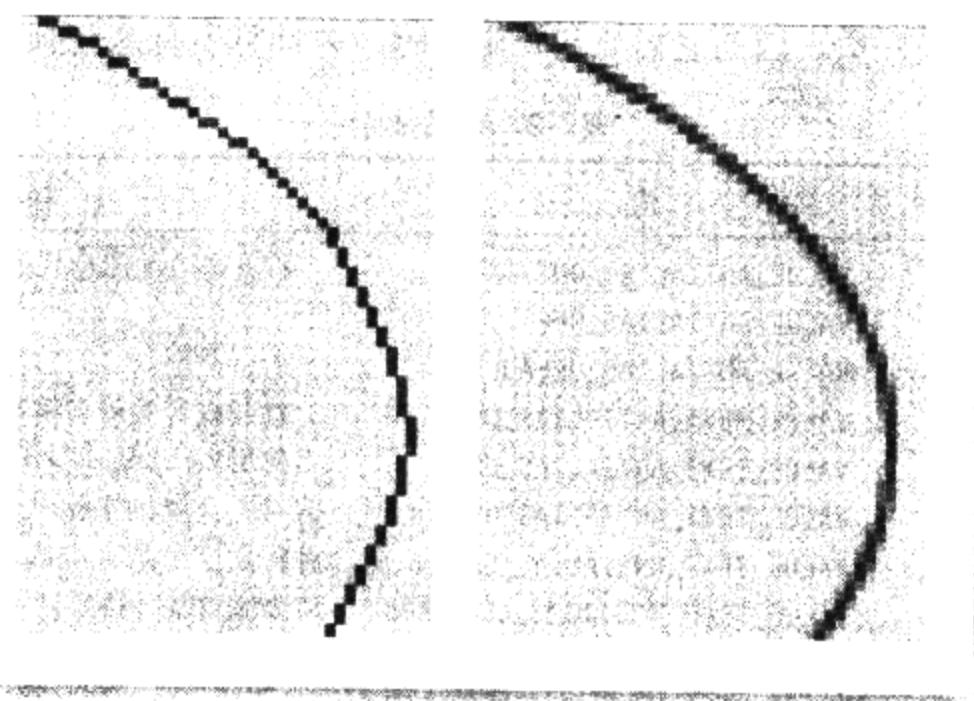


图7-25 消除图形锯齿现象的示意图

使用消除图形锯齿状技术对字体的绘制同样是有意义的。

```
g2.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING, RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
```

和上面的这些应用相比较，其他的绘图提示并不是很常用。

也可以把一组键/值提示信息对放入映射表中，并且通过调用`setRenderingHints`方法一次性的将它们全部设置好。任何实现了这个映射表接口的集合类都可以被使用，当然也可以使用`RenderingHints`类本身。它实现了`Map`接口，并且在用无参数的构造器来创建对象时，它会提供一个默认的映射表实现。例如，

```
RenderingHints hints = new RenderingHints(null);
hints.put(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
hints.put(RenderingHints.KEY_TEXT_ANTIALIASING, RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
g2.setRenderingHints(hints);
```

这就是我们在程序清单7-4中使用的技术。该程序绘制了一个图形，我们认为该绘图操作可能会从绘图提示中获得帮助。注意下面几点：

- 消除锯齿功能使椭圆变得平滑。
- 文本的消除锯齿功能使文本变得平滑。
- 在某些平台上，值为小数的文本距离会使字母之间彼此靠得更近一些。
- 选择`VALUE_RENDER_QUALITY`来平滑缩放的图像。（通过将`KEY_INTERPOLATION`设置为`VALUE_INTERPOLATION_BICUBIC`可以达到相同的效果）。
- 当消除锯齿功能关闭时，选择`VALUE_STROKE_NORMALIZE`会改变椭圆的外观和正方形对角线的位置。

图7-26显示了运行该程序时所截取的一个屏幕。

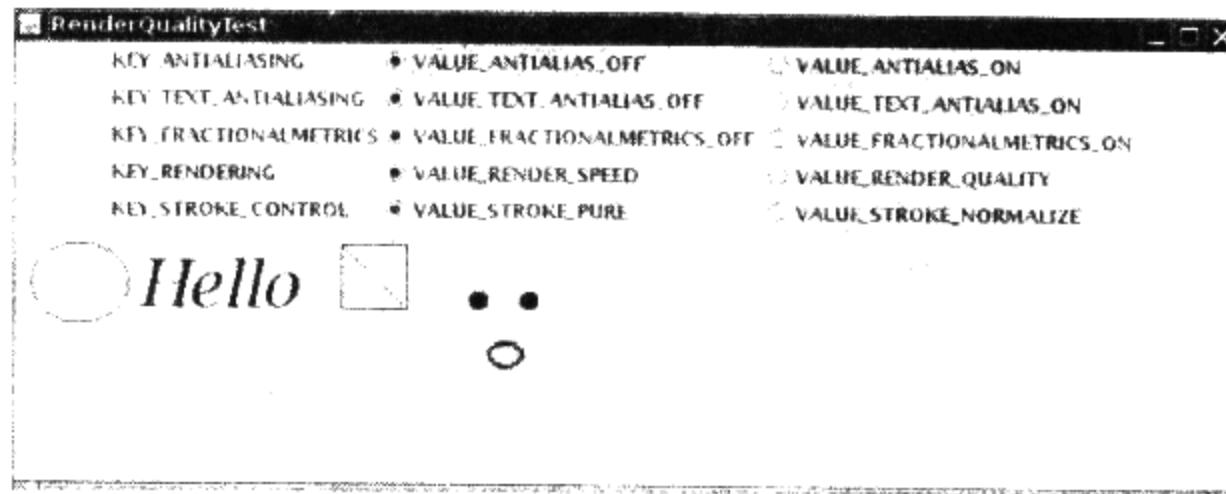


图7-26 绘图提示测试程序的效果

**程序清单7-4 RenderQualityTest.java**

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import java.awt.geom.*;
4 import java.io.*;
5 import javax.imageio.*;
6 import javax.swing.*;

7 /**
8 * This program demonstrates the effect of the various rendering hints.
9 * @version 1.10 2007-08-16
10 * @author Cay Horstmann
11 */
12
13 public class RenderQualityTest
14 {
15     public static void main(String[] args)
16     {
17         EventQueue.invokeLater(new Runnable()
18         {
19             public void run()
20             {
21                 JFrame frame = new RenderQualityTestFrame();
22                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23                 frame.setVisible(true);
24             }
25         });
26     }
27 }
28
29 /**
30 * This frame contains buttons to set rendering hints and an image that is drawn with
31 * the selected hints.
32 */
33 class RenderQualityTestFrame extends JFrame
34 {
35     public RenderQualityTestFrame()
36     {
37         setTitle("RenderQualityTest");
38         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
39     }
```

```
40.     buttonBox = new JPanel();
41.     buttonBox.setLayout(new GridBagLayout());
42.     hints = new RenderingHints(null);
43.
44.     makeButtons("KEY_ANTIALIASING", "VALUE_ANTIALIAS_OFF", "VALUE_ANTIALIAS_ON");
45.     makeButtons("KEY_TEXT_ANTIALIASING", "VALUE_TEXT_ANTIALIAS_OFF",
46.                 "VALUE_TEXT_ANTIALIAS_ON");
47.     makeButtons("KEY_FRACTIONALMETRICS", "VALUE_FRACTIONALMETRICS_OFF",
48.                 "VALUE_FRACTIONALMETRICS_ON");
49.     makeButtons("KEY_RENDERING", "VALUE_RENDER_SPEED", "VALUE_RENDER_QUALITY");
50.     makeButtons("KEY_STROKE_CONTROL", "VALUE_STROKE_PURE", "VALUE_STROKE_NORMALIZE");
51.     canvas = new RenderQualityComponent();
52.     canvas.setRenderingHints(hints);
53.
54.     add(canvas, BorderLayout.CENTER);
55.     add(buttonBox, BorderLayout.NORTH);
56. }
57.
58. /**
59. * Makes a set of buttons for a rendering hint key and values
60. * @param key the key name
61. * @param value1 the name of the first value for the key
62. * @param value2 the name of the second value for the key
63. */
64. void makeButtons(String key, String value1, String value2)
65. {
66.     try
67.     {
68.         final RenderingHints.Key k =
69.             (RenderingHints.Key) RenderingHints.class.getField(key).get(null);
70.         final Object v1 = RenderingHints.class.getField(value1).get(null);
71.         final Object v2 = RenderingHints.class.getField(value2).get(null);
72.         JLabel label = new JLabel(key);
73.
74.         buttonBox.add(label, new GBC(0, r).setAnchor(GBC.WEST));
75.         ButtonGroup group = new ButtonGroup();
76.         JRadioButton b1 = new JRadioButton(value1, true);
77.
78.         buttonBox.add(b1, new GBC(1, r).setAnchor(GBC.WEST));
79.         group.add(b1);
80.         b1.addActionListener(new ActionListener()
81.         {
82.             public void actionPerformed(ActionEvent event)
83.             {
84.                 hints.put(k, v1);
85.                 canvas.setRenderingHints(hints);
86.             }
87.         });
88.         JRadioButton b2 = new JRadioButton(value2, false);
89.
90.         buttonBox.add(b2, new GBC(2, r).setAnchor(GBC.WEST));
91.         group.add(b2);
92.         b2.addActionListener(new ActionListener()
93.         {
94.             public void actionPerformed(ActionEvent event)
95.             {
96.                 hints.put(k, v2);
```

```
97.         canvas.setRenderingHints(hints);
98.     }
99. }
100. hints.put(k, v1);
101. r++;
102. }
103. catch (Exception e)
104. {
105.     e.printStackTrace();
106. }
107. }

108. private RenderQualityComponent canvas;
109. private JPanel buttonBox;
110. private RenderingHints hints;
111. private int r;
112. private static final int DEFAULT_WIDTH = 750;
113. private static final int DEFAULT_HEIGHT = 300;
114. }
115. }

116. /**
117. * This component produces a drawing that shows the effect of rendering hints.
118. */
119. class RenderQualityComponent extends JComponent
120. {
121.     public RenderQualityComponent()
122.     {
123.         try
124.         {
125.             image = ImageIO.read(new File("face.gif"));
126.         }
127.         catch (IOException e)
128.         {
129.             e.printStackTrace();
130.         }
131.     }
132. }

133. public void paintComponent(Graphics g)
134. {
135.     Graphics2D g2 = (Graphics2D) g;
136.     g2.setRenderingHints(hints);

137.     g2.draw(new Ellipse2D.Double(10, 10, 60, 50));
138.     g2.setFont(new Font("Serif", Font.ITALIC, 40));
139.     g2.drawString("Hello", 75, 50);

140.     g2.draw(new Rectangle2D.Double(200, 10, 40, 40));
141.     g2.draw(new Line2D.Double(201, 11, 239, 49));

142.     g2.drawImage(image, 250, 10, 100, 100, null);
143. }

144. /**
145. * Sets the hints and repaints.
146. * @param h the rendering hints
147. */
148. public void setRenderingHints(RenderingHints h)
```

```

154  {
155.     hints = h;
156.     repaint();
157. }
158.
159. private RenderingHints hints = new RenderingHints(null);
160. private Image image;
161 }
```

### **API** `java.awt.Graphics2D 1.2`

- `void setRenderingHint(RenderingHints.Key key, Object value)`  
为该图形上下文设置绘图提示。
- `void setRenderingHints(Map m)`  
设置绘图提示，它们的键/值对存储在映射表中。

### **API** `java.awt.RenderingHints 1.2`

- `RenderingHints(Map<RenderingHints.Key, ?> m)`  
构建一个存放绘图提示的绘图提示映射表。如果m值为null，那么将会提供一个默认的绘图提示映射表。

## 7.10 图像的读取器和写入器

Java SE 在其1.4版本出现以前，只具备非常有限的对图像文件的读取和写入的功能。它可以读取GIF和JPEG格式的图像，但是对于写入图像的功能，它没有任何官方的支持。

现在这种情况已经得到了很大的改进。Java SE 1.4引入了`javax.imageio`包，它包含了对读取和写入数种常用文件格式进行支持的“附加”特性。同时还包含了一个框架，使得第三方能够为其他图像格式的文件添加读取器和写入器。从Java SE 6开始，GIF、JPEG、PNG、BMP（Windows位图）和WBMP（无线位图）等文件格式都得到了支持。（在较早的版本中，由于专利方面的原因，使得写入GIF格式文件不被支持）。

该类库的基本应用是极其直接的。要想装载一个图像，可以使用`ImageIO`类的静态`read`方法。

```
File f = ...;
BufferedImage image = ImageIO.read(f);
```

`ImageIO`类根据文件的类型，选择一个合适的读取器。它可以参考文件的扩展名和文件开头的“幻数”（magic number）来选择读取器。如果没有找到合适的读取器或者读取器不能解码文件的内容，那么`read`方法将返回null。

把图像写入到文件中也是一样简单的。

```
File f = ...;
String format = ...;
ImageIO.write(image, format, f);
```

这里，`format`字符串用来标识图像的格式，比如“JPEG”或者“PNG”。`ImageIO`类将选择一个合适的写入器以存储文件。

### 7.10.1 获得图像文件类型的读取器和写入器

对于那些超出ImageIO类的静态read和write方法能力范围的高级图像的读取和写入操作来说，首先需要获得合适的ImageReader和ImageWriter对象。ImageIO类列举了匹配下列条件之一的读取器和写入器。

- 图像格式（比如“JPEG”）
- 文件后缀（比如“jpg”）
- MIME类型（比如“image/jpeg”）

 注意：MIME（Multipurpose Internet Mail Extensions standard）是“多用途因特网邮件扩展标准”的英文缩写。MIME标准定义了常用的数据格式，比如“image/jpeg”和“application/pdf”等。如果要了解定义MIME格式的RFC（征求意见的文件）的HTML版本，请访问<http://www.oac.uci.edu/indiv/ehood/MIME>。

例如，可以用下面的代码来获取一个JPEG格式文件的读取器。

```
ImageReader reader = null;
Iterator<ImageReader> iter = ImageIO.getImageReadersByFormatName("JPEG");
if (iter.hasNext()) reader = iter.next();
```

getImageReadersBySuffix和getImageReadersByMIMEType这两个方法用于枚举与文件扩展名或MIME类型相匹配的读取器。

ImageIO类可能找到了多个读取器，而它们都能够读取某一特殊类型的图像文件。在这种情况下，必须从中选择一个，但是也许你不清楚怎么样才能选择一个最好的。如果要了解更多的关于读取器的信息，就要获取它的服务提供器接口：

```
ImageReaderSpi spi = reader.getOriginatingProvider();
```

然后，可以获得供应商的名字和版本号：

```
String vendor = spi.getVendor();
String version = spi.getVersion();
```

也许该信息能够帮助你决定选择哪一种读取器，或者你可以为你的程序用户提供一个读取器的列表，让他们做出选择。然而，目前来说，我们假定第一个列出来的读取器就能够满足用户的需求。

在程序清单7-5中，我们想查找所有可获得的读取器能够处理的所有文件的所有后缀，这样我们可以在文件过滤器中使用它们。从Java SE 6开始，我们可以使用静态的ImageIO.getReaderFileSuffixes方法来达到此目的：

```
String[] extensions = ImageIO.getReaderFileSuffixes();
chooser.setFileFilter(new FileNameExtensionFilter("Image files", extensions));
```

对于保存文件，我们遇到了一个类似的问题：我们希望为用户展示一个支持所有图像类型的菜单。可惜，IOImage类的getWriterFormateNames方法返回了一个相当奇怪的列表，里边包含了许多冗余的名字，比如：

jpg, BMP, bmp, JPG, jpeg, wbmp, png, JPEG, PNG, WBMP, GIF, gif

这些并不是人们想要在菜单中显示的东西。我们所需要的是“首选”格式名列表。我们提

供了一个用于此目的的助手方法`getWriterFormats`（参见程序清单7-5）。我们查找与其相关的第一个写入器。然后，询问写入器它的格式名是什么，从而希望它能够首先列出最流行的一个格式名。实际上，对JPEG写入器来说，这种方法确实很有效：它将“JPEG”列在其他选项的前面。（另一方面，PNG写入器把小写字母的“png”列在“PNG”的前面。我们希望这种行为能够在将来的某个时候得以解决。同时，我们强制将全小写名字转换为大写）。一旦挑选了首选名，我们就会将所有其他的候选名从最初的名字集中移除。之后，我们会继续执行直至所有的格式名都得到处理。

### 7.10.2 读取和写入带有多个图像的文件

有些文件，特别是GIF动画文件，都包含了多个图像。`ImageIO`类的`read`方法只能读取单一的图像文件。为了读取多个图像，应该将输入源（例如，输入流或者输入文件）转换成一个`ImageInputStream`。

```
InputStream in = . . .;
ImageInputStream imageIn = ImageIO.createImageInputStream(in);
```

接着把图像输入流作为参数传递给读取器的`setInput`方法：

```
reader.setInput(imageIn, true);
```

方法中的第二个参数值表示输入的方式是“只向前搜索”，否则，就采用随机访问的方式，要么是在读取时将缓冲输入流，要么是使用随机文件访问。对于某些操作来说，必须使用随机访问的方法。例如，为了在一个GIF文件中查找所有的图像个数，需要读入整个文件。这时，如果想获取某一图像的话，必须再次读入该输入文件。

只有当从一个流中读取图像，并且输入流中包含多个图像，而且在文件头中的图像格式部分没有所需要的信息（比如图像的个数）时，考虑使用上面的方法才是合适的。如果要从一个文件中读取图像信息的话，可直接使用下面的方法：

```
File f = . . .;
ImageInputStream imageIn = ImageIO.createImageInputStream(f);
reader.setInput(imageIn);
```

一旦拥有了一个读取器后，就可以通过调用下面的方法来读取输入流中的图像。

```
BufferedImage image = reader.read(index);
```

其中`index`是图像的索引，其值从0开始。

如果输入流采用“只向前搜索”的方式，那么应该持续不断地读取图像，直到`read`方法抛出一个`IndexOutOfBoundsException`为止。否则，可以调用`getNumImages`方法：

```
int n = reader.getNumImages(true);
```

在该方法中，它的参数表示允许搜索输入流来确定图像的数目。如果输入流采用“只向前搜索”的方式，那么方法将抛出一个`IllegalStateException`异常。要不然，可以把是否“允许搜索”参数设置为`false`。如果`getNumImages`方法在不搜索输入流的情况下无法确定图像的数目，那么它将返回-1。在这种情况下，必须转换到B方案，那就是持续不断地读取图像，直到获得一个`IndexOutOfBoundsException`异常为止。

有些文件包含一些缩略图，也就是图像用来预览的小版本。可以通过调用下面的方法来获

得某个图像的缩略图数量。

```
int count = reader.getNumThumbnails(index);
```

然后得到如下的一个特殊索引：

```
BufferedImage thumbnail = reader.getThumbnail(index, thumbnailIndex);
```

另一个问题是，有时你想在实际获得图像之前，了解该图像的大小。特别是，当图像是从一个较慢的网络连接中获取的时候，你更加希望能够事先了解到该图像的大小。那么请使用下面的方法：

```
int width = reader.getWidth(index);
int height = reader.getHeight(index);
```

通过上面两个方法可以获得具有给定索引的图像的大小。

如果要将多个图像写入到一个文件中，首先需要一个ImageWriter。ImageIO类能够枚举可以写入某种特定图像格式的所有写入器。

```
String format = . . .;
ImageWriter writer = null;
Iterator<ImageWriter> iter = ImageIO.getImageWritersByFormatName(format);
if (iter.hasNext()) writer = iter.next();
```

接着，将一个输出流或者输出文件转换成ImageOutputStream，并且将其作为参数传给写入器。例如，

```
File f = . . .;
ImageOutputStream imageOut = ImageIO.createImageOutputStream(f);
writer.setOutput(imageOut);
```

必须将每一个图像都包装到IIOImage对象中。可以根据情况提供一个缩略图和图像元数据（比如，图像的压缩算法和颜色信息）的列表。在本例中，我们把两者都设置为null；如果要了解详细信息，请参阅JDK文档。

```
IIOImage iioImage = new IIOImage(images[i], null, null);
```

使用write方法，写出第一个图像：

```
writer.write(new IIOImage(images[0], null, null));
```

对于后续的图像，使用下面的方法：

```
if (writer.canInsertImage(i))
    writer.writeInsert(i, iioImage, null);
```

上面方法中的第三个参数可以包含一个ImageWriteParam对象，用以设置图像写入的详细信息，比如是平铺还是压缩；用null作为其默认值。

并不是所有的图像格式都能够处理多个图像的。在这种情况下，如果i>0，canInsertImage方法将返回false值，而且只保存一个单一图像。

程序清单7-5中的程序使用Java类库所提供的读取器和写入器的格式来加载和保持文件。该程序显示了多个图像（见图7-27），但是没有缩略图。

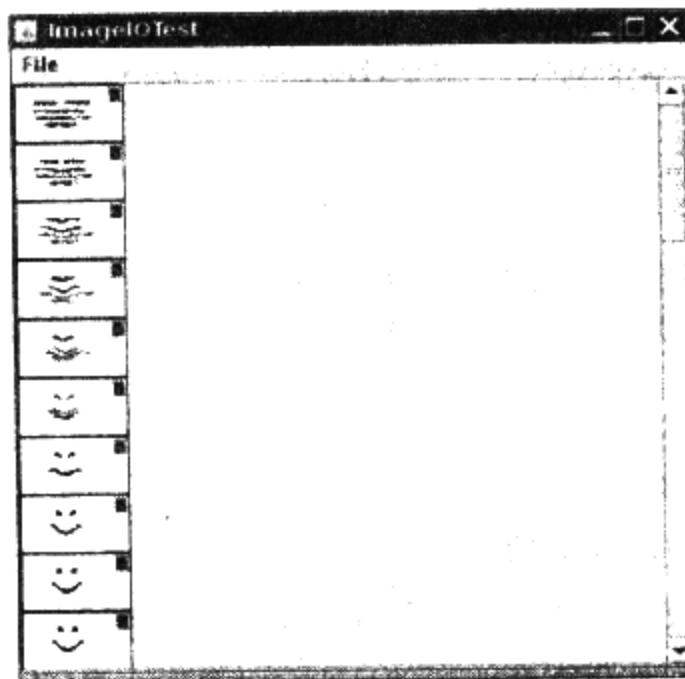


图7-27 一个GIF动画图像

**程序清单7-5 ImageIOTest.java**

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.awt.image.*;
4. import java.io.*;
5. import java.util.*;
6. import javax.imageio.*;
7. import javax.imageio.stream.*;
8. import javax.swing.*;
9. import javax.swing.filechooser.*;

10.
11 /**
12 * This program lets you read and write image files in the formats that the JDK supports.
13 * Multi-file images are supported.
14 * @version 1.02 2007-08-16
15 * @author Cay Horstmann
16 */
17 public class ImageIOTest
18 {
19     public static void main(String[] args)
20     {
21         EventQueue.invokeLater(new Runnable()
22         {
23             public void run()
24             {
25                 JFrame frame = new ImageIOFrame();
26                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27                 frame.setVisible(true);
28             }
29         });
30     }
31 }
32 /**
33 * This frame displays the loaded images. The menu has items for loading and saving files.
34 */
35
```

```
36. class ImageIOFrame extends JFrame
37. {
38.     public ImageIOFrame()
39.     {
40.         setTitle("ImageIOTest");
41.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
42.
43.         JMenu fileMenu = new JMenu("File");
44.         JMenuItem openItem = new JMenuItem("Open");
45.         openItem.addActionListener(new ActionListener()
46.         {
47.             public void actionPerformed(ActionEvent event)
48.             {
49.                 openFile();
50.             }
51.         });
52.         fileMenu.add(openItem);
53.
54.         JMenu saveMenu = new JMenu("Save");
55.         fileMenu.add(saveMenu);
56.         Iterator<String> iter = writerFormats.iterator();
57.         while (iter.hasNext())
58.         {
59.             final String formatName = iter.next();
60.             JMenuItem formatItem = new JMenuItem(formatName);
61.             saveMenu.add(formatItem);
62.             formatItem.addActionListener(new ActionListener()
63.             {
64.                 public void actionPerformed(ActionEvent event)
65.                 {
66.                     saveFile(formatName);
67.                 }
68.             });
69.         }
70.
71.         JMenuItem exitItem = new JMenuItem("Exit");
72.         exitItem.addActionListener(new ActionListener()
73.         {
74.             public void actionPerformed(ActionEvent event)
75.             {
76.                 System.exit(0);
77.             }
78.         });
79.         fileMenu.add(exitItem);
80.
81.         JMenuBar menuBar = new JMenuBar();
82.         menuBar.add(fileMenu);
83.         setJMenuBar(menuBar);
84.     }
85.
86. /**
87. * Open a file and load the images.
88. */
89. public void openFile()
90. {
91.     JFileChooser chooser = new JFileChooser();
92.     chooser.setCurrentDirectory(new File("."));
```

```
93.     String[] extensions = ImageIO.getReaderFileSuffixes();
94.     chooser.setFileFilter(new FileNameExtensionFilter("Image files", extensions));
95.     int r = chooser.showOpenDialog(this);
96.     if (r != JFileChooser.APPROVE_OPTION) return;
97.     File f = chooser.getSelectedFile();
98.     Box box = Box.createVerticalBox();
99.     try
100.    {
101.        String name = f.getName();
102.        String suffix = name.substring(name.lastIndexOf('.') + 1);
103.        Iterator<ImageReader> iter = ImageIO.getImageReadersBySuffix(suffix);
104.        ImageReader reader = iter.next();
105.        ImageInputStream imageIn = ImageIO.createImageInputStream(f);
106.        reader.setInput(imageIn);
107.        int count = reader.getNumImages(true);
108.        images = new BufferedImage[count];
109.        for (int i = 0; i < count; i++)
110.        {
111.            images[i] = reader.read(i);
112.            box.add(new JLabel(new ImageIcon(images[i])));
113.        }
114.    }
115.    catch (IOException e)
116.    {
117.        JOptionPane.showMessageDialog(this, e);
118.    }
119.    setContentPane(new JScrollPane(box));
120.    validate();
121. }
122.
123. /**
124. * Save the current image in a file
125. * @param formatName the file format
126. */
127. public void saveFile(final String formatName)
128. {
129.     if (images == null) return;
130.     Iterator<ImageWriter> iter = ImageIO.getImageWritersByFormatName(formatName);
131.     ImageWriter writer = iter.next();
132.     JFileChooser chooser = new JFileChooser();
133.     chooser.setCurrentDirectory(new File("."));
134.     String[] extensions = writer.getOriginatingProvider().getFileSuffixes();
135.     chooser.setFileFilter(new FileNameExtensionFilter("Image files", extensions));
136.
137.     int r = chooser.showSaveDialog(this);
138.     if (r != JFileChooser.APPROVE_OPTION) return;
139.     File f = chooser.getSelectedFile();
140.     try
141.     {
142.         ImageOutputStream imageOut = ImageIO.createImageOutputStream(f);
143.         writer.setOutput(imageOut);
144.
145.         writer.write(new IIOImage(images[0], null, null));
146.         for (int i = 1; i < images.length; i++)
147.         {
148.             IIOImage iioImage = new IIOImage(images[i], null, null);
149.             if (writer.canInsertImage(i)) writer.writeInsert(i, iioImage, null);

```

```
150.        }
151.    }
152.    catch (IOException e)
153.    {
154.        JOptionPane.showMessageDialog(this, e);
155.    }
156. }
157.
158. /**
159. * Gets a set of "preferred" format names of all image writers. The preferred format name
160. * is the first format name that a writer specifies.
161. * @return the format name set
162. */
163. public static Set<String> getWriterFormats()
164. {
165.     TreeSet<String> writerFormats = new TreeSet<String>();
166.     TreeSet<String> formatNames = new TreeSet<String>(Arrays.asList(ImageIO
167.         .getWriterFormatNames()));
168.     while (formatNames.size() > 0)
169.     {
170.         String name = formatNames.iterator().next();
171.         Iterator<ImageWriter> iter = ImageIO.getImageWritersByFormatName(name);
172.         ImageWriter writer = iter.next();
173.         String[] names = writer.getOriginatingProvider().getFormatNames();
174.         String format = names[0];
175.         if (format.equals(format.toLowerCase())) format = format.toUpperCase();
176.         writerFormats.add(format);
177.         formatNames.removeAll(Arrays.asList(names));
178.     }
179.     return writerFormats;
180. }
181.
182. private BufferedImage[] images;
183. private static Set<String> writerFormats = getWriterFormats();
184. private static final int DEFAULT_WIDTH = 400;
185. private static final int DEFAULT_HEIGHT = 400;
186. }
```

### API javax.imageio.ImageIO 1.4

- static BufferedImage read(File input)
- static BufferedImage read(InputStream input)
- static BufferedImage read(URL input)  
从input中读取一个图像。
- static boolean write(RenderedImage image, String formatName, File output)
- static boolean write(RenderedImage image, String formatName, OutputStream output)  
将给定格式的图像写入output中。如果没有找到合适的写入器，则返回false。
- static Iterator<ImageReader> getImageReadersByFormatName(String formatName)
- static Iterator<ImageReader> getImageReadersBySuffix(String fileSuffix)

- static Iterator<ImageReader> getImageReadersByMIMEType(String mimeType)
- static Iterator<ImageWriter> getImageWritersByFormatName(String formatName)
- static Iterator<ImageWriter> getImageWritersBySuffix(String fileSuffix)
- static Iterator<ImageWriter> getImageWritersByMIMEType(String mimeType)
 

获得能够处理给定格式（例如“JPEG”）、文件后缀（例如“jpg”）或者MIME类型（例如“image/jpeg”）的所有读取器和写入器。
- static String[] getReaderFormatNames()
- static String[] getReaderMIMETypes()
- static String[] getWriterFormatNames()
- static String[] getWriterMIMETypes()
- static String[] getReaderFileSuffixes() 6
- static String[] getWriterFileSuffixes() 6
 

获取读取器和写入器所支持的所有格式名、MIME类型名和文件后缀。
- ImageInputStream createImageInputStream(Object input)
- ImageOutputStream createImageOutputStream(Object output)
 

根据给定的对象来创建一个图像输入流或者图像输出流。该对象可能是一个文件、一个流、一个RandomAccessFile或者某个服务提供商能够处理的其他类型的对象。如果没有任何注册过的服务提供器能够处理这个对象，那么返回null值。

**javax.imageio.ImageReader 1.4**

- void setInput(Object input)

- void setInput(Object input, boolean seekForwardOnly)

设置读取器的输入源。

参数：input 一个ImageInputStream对象或者是这个读取器能够接受的其他对象

seekForwardOnly 如果读取器只应该向前阅读，则返回ture。默认的，读取器采用随机访问的方式，如果有必要，应该将图像数据放入缓存中

- BufferedImage read(int index)

读取给定图像索引的图像（索引从0开始）。如果没有这个图像，则抛出一个IndexOutOfBoundsException异常。

- int getNumImages(boolean allowSearch)

获取读取器中图像的数目。如果allowSearch值为false，并且不向前阅读就无法确定图像的数目，那么它将返回-1。如果allowSearch值是true，并且读取器采用了“只向前搜索”方式，那么就会抛出IllegalStateException异常。

- int getNumThumbnails(int index)

获取给定索引的图像的缩略图的数量。

- `BufferedImage readThumbnail(int index, int thumbnailIndex)`  
获取给定索引的图像的索引号为`thumbnailIndex`的缩略图。
- `int getWidth(int index)`
- `int getHeight(int index)`  
获取图像的宽度和高度。如果没有这样的图像，就抛出一个`IndexOutOfBoundsException`异常。
- `ImageReaderSpi getOriginatingProvider()`  
获取构建该读取器的服务提供器。

**API** `javax.imageio.spi.IIOServiceProvider` 1.4

- `String getVendorName()`
- `String getVersion()`  
获取该服务提供器提供者的名字和版本。

**API** `javax.imageio.spi.ImageReaderWriterSpi` 1.4

- `String[] getFormatNames()`
- `String[] getFileSuffixes()`
- `String[] getMIMETypes()`

获取由该服务提供器创建的读取器或者写入器所支持的图像格式名、文件的后缀和MIME类型。

**API** `javax.imageio.ImageWriter` 1.4

- `void setOutput(Object output)`  
设置该写入器的输出目标。  
参数：`output` 一个`ImageOutputStream`对象或者这个写入器能够接受的其他对象。
- `void write(IIOImage image)`
- `void write(RenderedImage image)`  
把单一的图像写入到输出流中。
- `void writeInsert(int index, IIOImage image, ImageWriteParam param)`  
把一个图像写入到一个包含多个图像的文件中。
- `boolean canInsertImage(int index)`  
如果在给定的索引处可以插入一个图像的话，则返回`true`值。
- `ImageWriterSpi getOriginatingProvider()`  
获取构建该写入器的服务提供器。

**API** `javax.imageio.IIOImage` 1.4

- `IIOImage(RenderedImage image, List thumbnails, IIOMetadata metadata)`  
根据一个图像、可选的缩略图和可选的元数据来构建一个`IIOImage`对象。

## 7.11 图像处理

假设你有一个图像，并且希望改善图像的外观。这时需要访问该图像的每一个像素，并用其他的像素来取代这些像素。或者，你也许想要从头计算某个图像的像素，例如，你想显示一下物理测量或者数学计算的结果。`BufferedImage`类提供了对图像中像素的控制能力，而实现了`BufferedImageOp`接口的类都可以对图像进行变换操作。

 **注意：**JDK1.0有一个完全不同且复杂得多的图像框架，它得到了优化，以支持对从Web下载的图像进行增量渲染（incremental rendering），即一次绘制一个扫描行。但是，操作这些图像很困难。我们在本书中不讨论这个框架。

### 7.11.1 构建光栅图像

你处理的大多数图像都是直接从图像文件中读入的。这些图像有的可能是数码相机产生的，有的是扫描仪扫描而产生的，还有的一些图像是绘图程序产生的。在本节中，我们将介绍一种构建图像的不同技术，也就是一次为图像增加一个像素。

为了构建一个图像，需要以通常的方法构建一个`BufferedImage`对象。

```
image = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
```

现在，调用`getRaster`方法来获得一个类型为`WritableRaster`的对象。使用这个对象来访问和修改该图像的各个像素。

```
WritableRaster raster = image.getRaster();
```

使用`setPixel`方法可以设置一个单独的像素。这项操作的复杂性在于不能只是为该像素设置一个`Color`值，还必须知道存放在缓冲中的图像是如何设定颜色的，这依赖于图像的类型。如果图像有一个`TYPE_INT_ARGB`的类型，那么每一个像素都用四个值来描述，即：红、绿、蓝和透明度（alpha）。每个值的取值范围都介于0和255之间。需要以包含四个整数值的一个数组的形式给出：

```
int[] black = { 0, 0, 0, 255 };
raster.setPixel(i, j, black);
```

用Java 2D API的行话来说，这些值被称为像素的样本值。

 **警告：**还有一些参数值是`float[]`和`double[]`类型的`setPixel`方法。然而，需要在这些数组中放置的值并不是介于0.0和1.0之间的规格化的颜色值。

```
float[] red = { 1.0F, 0.0F, 0.0F, 1.0F };
raster.setPixel(i, j, red); // ERROR
```

无论数组属于什么类型，都必须提供介于0和255之间的某个值。

可以使用`setPixels`方法提供批量的像素。需要设置矩形的起始像素的位置和矩形的宽度和高度。接着，提供一个包含所有像素的样本值的一个数组。例如，如果你缓冲的图像有一个`TYPE_INT_ARGB`类型，那么就应该提供第一个像素的红、绿、蓝和透明度的值（alpha），然后，提供第二个像素的红、绿、蓝和透明度的值，以此类推。

```
int[] pixels = new int[4 * width * height];
pixels[0] = . . . // red value for first pixel
```

```
pixels[1] = . . . // green value for first pixel  
pixels[2] = . . . // blue value for first pixel  
pixels[3] = . . . // alpha value for first pixel  
. . .  
raster.setPixels(x, y, width, height, pixels);
```

反过来，如果要读入一个像素，可以使用getPixel方法。这需要提供一个含有四个整数的数组，用以存放各个样本值。

```
int[] sample = new int[4];  
raster.getPixel(x, y, sample);  
Color c = new Color(sample[0], sample[1], sample[2], sample[3]);
```

可以使用getPixels方法来读取多个像素。

```
raster.getPixels(x, y, width, height, samples);
```

如果使用的图像类型不是TYPE\_INT\_ARGB，并且已知该类型是如何表示像素值的，那么仍旧可以使用getPixel/setPixel方法。不过，必须要知道该特定图像类型的样本值是如何进行编码的。

如果需要对任意未知类型的图像进行处理，那么你就要费神了。每一个图像类型都有一个颜色模型，它能够在样本值数组和标准的RGB颜色模型之间进行转换。



**注意：**RGB颜色模型并不像你想象中的那么标准。颜色值的确切样子依赖于成像设备的特性。数码相机、扫描仪、控制器和LCD显示器等都有它们独有的特性。结果是，同样的RGB值在不同的设备上看上去就存在很大的差别。国际配色联盟(<http://www.color.org>)推荐，所有的颜色数据都应该配有一个ICC配置特性，它用以设定各种颜色是如何映射到标准格式的，比如1931 CIE XYZ颜色技术规范。该规范是由国际照明委员会即CIE (Commission Internationale de l'Eclairage，其网址为：<http://www.cie.co.at/cie>)制定的。该委员会是负责提供涉及照明和颜色等相关领域事务的技术指导的国际性机构。该规范是显示肉眼能够察觉到的所有颜色的一个标准化方法。它采用称为X、Y、Z三元组坐标的方式来显示颜色。(关于1931 CIE XYZ 规范的详尽信息，可以参阅Foley、van Dam和Feiner等人所撰写的《Computer Graphics: Principles and Practice》一书的第13章。)

ICC配置特性非常复杂。然而，我们建议使用一个相对简单的标准，称为sRGB (请访问其网址<http://www.w3.org/Graphics/Color/sRGB.html>)。它设定了RGB值与1931 CIE XYZ值之间的具体转换方法，它可以非常出色地在通用的彩色监视器上运用。当需要在RGB与其他颜色模型之间进行转换的时候，Java 2D API就使用这种转换方式。

getColorModel方法返回一个颜色模型：

```
ColorModel model = image.getColorModel();
```

为了了解一个像素的颜色值，可以调用Raster类的getDataElements方法。这个方法返回了一个Object，它包含了有关该颜色值的与特定颜色模型相关的描述。

```
Object data = raster.getDataElements(x, y, null);
```

 注意：`getDataElements`方法返回的对象实际上是一个样本值的数组。在处理这个对象时，不必要了解到这些。但是，它却解释了为什么这个方法名叫做`getDataElements`的原因。

颜色模型能够将该对象转换成标准的ARGB的值。`getRGB`方法返回一个int类型的值，它把透明度（alpha）、红、绿和蓝的值打包成四个块，每块包含8位。也可以使用`Color(int argb, boolean hasAlpha)`构造器来构建一个颜色的值：

```
int argb = model.getRGB(data);
Color color = new Color(argb, true);
```

如果要把一个像素设置为某个特定的颜色值，需要与上述相反的步骤进行操作。`Color`类的`getRGB`方法产生一个包含透明度、红、绿和蓝值的int型值。把这个值提供给`ColorModel`类的`getDataElements`方法，其返回值是一个包含了该颜色值的特定颜色模型描述的Object。再将这个对象传递给`WritableRaster`类的`setDataElements`方法。

```
int argb = color.getRGB();
Object data = model.getDataElements(argb, null);
raster.setDataElements(x, y, data);
```

为了阐明如何使用这些方法来用各个像素构建图像，我们按照传统，绘制了一个Mandelbrot集，如图7-28所示。

Mandelbrot集的思想就是把平面上的每一点和一个数字序列关联在一起。如果数字序列是收敛的，该点就被着色。如果数字序列是发散的，该点就处于透明状态。

下面就是构建简单Manderbrot集的方法。对于每一个点(a, b)，你都能按照如下的公式得到一个点集序列，其开始于点(x, y)=(0, 0)，反复进行迭代：

$$\begin{aligned}x_{\text{new}} &= x^2 - y^2 + a \\y_{\text{new}} &= 2 \cdot x \cdot y + b\end{aligned}$$

结果证明，如果x或者y的值大于2，那么序列就是发散的。仅有那些与导致数字序列收敛的点(a, b)相对应的像素才会被着色。（该数字序列的计算公式基本上是从复杂的数学概念中推导出来的。我们只使用现成的公式。如果要了解更多的分形数学概念的详细说明，请查看<http://classes.yale.edu/fractals/>）

程序清单7-6显示了该代码。在此程序中，我们展示了如何使用`ColorModel`类将`Color`值转换成像素数据。这个过程和图像的类型是不相关的。为了增加些趣味，你可以把缓冲图像的颜色类型改变为`TYPE_BYTE_GRAY`。不必改变程序中的任何代码，该图像的颜色模型会自动地负责把颜色转换为样本值。

#### 程序清单7-6 RasterImageTest.java

```
1. import java.awt.*;
2. import java.awt.image.*;
```

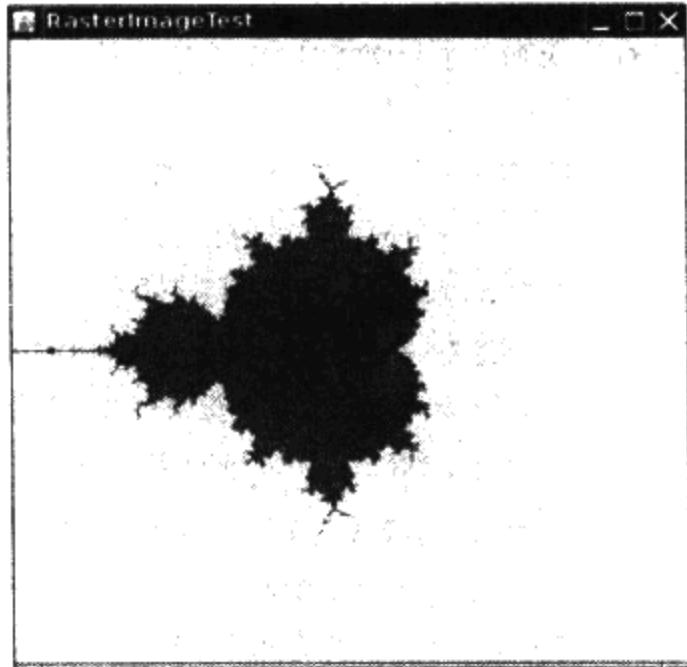


图7-28 Mandelbrot集

```
3. import javax.swing.*;
4.
5. /**
6. * This program demonstrates how to build up an image from individual pixels.
7. * @version 1.13 2007-08-16
8. * @author Cay Horstmann
9. */
10. public class RasterImageTest
11. {
12.     public static void main(String[] args)
13.     {
14.         EventQueue.invokeLater(new Runnable()
15.         {
16.             public void run()
17.             {
18.                 JFrame frame = new RasterImageFrame();
19.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20.                 frame.setVisible(true);
21.             }
22.         });
23.     }
24. }
25.
26. /**
27. * This frame shows an image with a Mandelbrot set.
28. */
29. class RasterImageFrame extends JFrame
30. {
31.     public RasterImageFrame()
32.     {
33.         setTitle("RasterImageTest");
34.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
35.         BufferedImage image = makeMandelbrot(DEFAULT_WIDTH, DEFAULT_HEIGHT);
36.         add(new JLabel(new ImageIcon(image)));
37.     }
38.
39. /**
40. * Makes the Mandelbrot image.
41. * @param width the width
42. * @param height the height
43. * @return the image
44. */
45. public BufferedImage makeMandelbrot(int width, int height)
46. {
47.     BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
48.     WritableRaster raster = image.getRaster();
49.     ColorModel model = image.getColorModel();
50.
51.     Color fractalColor = Color.red;
52.     int argb = fractalColor.getRGB();
53.     Object colorData = model.getDataElements(argb, null);
54.
55.     for (int i = 0; i < width; i++)
56.         for (int j = 0; j < height; j++)
57.         {
58.             double a = XMIN + i * (XMAX - XMIN) / width;
59.             double b = YMIN + j * (YMAX - YMIN) / height;
```

```

60.         if (!escapesToInfinity(a, b)) raster.setDataElements(i, j, colorData);
61.     }
62.     return image;
63. }
64.
65. private boolean escapesToInfinity(double a, double b)
66. {
67.     double x = 0.0;
68.     double y = 0.0;
69.     int iterations = 0;
70.     while (x <= 2 && y <= 2 && iterations < MAX_ITERATIONS)
71.     {
72.         double xnew = x * x - y * y + a;
73.         double ynew = 2 * x * y + b;
74.         x = xnew;
75.         y = ynew;
76.         iterations++;
77.     }
78.     return x > 2 || y > 2;
79. }
80.
81. private static final double XMIN = -2;
82. private static final double XMAX = 2;
83. private static final double YMIN = -2;
84. private static final double YMAX = 2;
85. private static final int MAX_ITERATIONS = 16;
86. private static final int DEFAULT_WIDTH = 400;
87. private static final int DEFAULT_HEIGHT = 400;
88 }

```

### **java.awt.image.BufferedImage 1.2**

- **BufferedImage(int width, int height, int imageType)**

构建一个被缓存的图像对象。

参数: **width, height** 图像的尺寸

**imageType** 图像的类型，最常用的类型是 **TYPE\_INT\_RGB**, **TYPE\_INT\_ARGB**, **TYPE\_BYTE\_GRAY** 和 **TYPE\_BYTE\_INDEXED**,

- **ColorModel getColorModel()**

返回被缓存图像的颜色模型。

- **WritableRaster getRaster()**

获得访问和修改该缓存图像像素的光栅。

### **java.awt.image.Raster 1.2**

- **Object getDataElements(int x, int y, Object data)**

返回某个光栅点的样本数据，该数据位于一个数组中，而该数组的长度和类型依赖于颜色模型。如果**data**不为**null**，那么它将被视为是适合于存放样本数据的数组，从而被填充。如果**data**为**null**，那么将分配一个新的数组，其元素的类型和长度依赖于颜色模型。

- **int[] getPixel(int x, int y, int[] sampleValues)**

- `float[] getPixel(int x, int y, float[] sampleValues)`
  - `double[] getPixel(int x, int y, double[] sampleValues)`
  - `int[] getPixels(int x, int y, int width, int height, int[] sampleValues)`
  - `float[] getPixels(int x, int y, int width, int height, float[] sampleValues)`
  - `double[] getPixels(int x, int y, int width, int height, double[] sampleValues)`
- 返回某个光栅点或者是由光栅点组成的某个矩形的样本值，该数据位于一个数组中，数组的长度依赖于颜色模型。如果`sampleValues`不为null，那么该数组被视为长度足够存放样本值，从而该数组被填充。如果`sampleValues`为null，就要分配一个新数组。仅当你知道某一颜色模型的样本值的具体含义的时候，这些方法才会有用。



#### `java.awt.image.WritableRaster 1.2`

- `void setDataElements(int x, int y, Object data)`

设置光栅点的样本数据。`data`是一个已经填入了某一像素样本值的数组。数组元素的类型和长度依赖于颜色模型。

- `void setPixel(int x, int y, int[] sampleValues)`
  - `void setPixel(int x, int y, float[] sampleValues)`
  - `void setPixel(int x, int y, double[] sampleValues)`
  - `void setPixels(int x, int y, int width, int height, int[] sampleValues)`
  - `void setPixels(int x, int y, int width, int height, float[] sampleValues)`
  - `void setPixels(int x, int y, int width, int height, double[] sampleValues)`
- 设置某个光栅点或由多个光栅点组成的矩形的样本值。只有当你知道颜色模型样本值的编码规则时，这些方法才会有用。



#### `java.awt.image.ColorModel 1.2`

- `int getRGB(Object data)`

返回对应于`data`数组中传递的样本数据的ARGB值。其元素的类型和长度依赖于颜色模型。

- `Object getDataElements(int argb, Object data);`

返回某个颜色值的样本数据。如果`data`不为null，那么该数组被视为非常适合于存放样本值，进而该数组被填充。如果`data`为null，那么将分配一个新的数组。`data`是一个填充了用于某个像素的样本数据的数组，其元素的类型和长度依赖于该颜色模型。



#### `java.awt.Color 1.0`

- `Color(int argb, boolean hasAlpha) 1.2`

如果`hasAlpha`的值是true，则用指定的ARGB组合值创建一种颜色。如果`hasAlpha`的值是false，则用指定的RGB值创建一种颜色。

- `int getRGB()`

返回和该颜色相对应的ARGB颜色值。

### 7.11.2 图像过滤

在前面的章节中，我们介绍了一个从头开始构建图像的方法。然而，你常常是因为另一个原因去访问图像数据的：你已经拥有了一个图像，并且想从某些方面对图像进行改进。

当然，可以使用前一节中的`getPixel/getDataElements`方法来读取和处理图像数据，然后把图像数据写回到文件中。不过，幸运的是，Java 2D API已经提供了许多过滤器，它们能够执行常用的图像处理操作。

图像处理都实现了`BufferedImageOp`接口。建立了图像处理的操作之后，只需调用`filter`方法，就可以把该图像转换成另一个图像。

```
BufferedImageOp op = . . .;
BufferedImage filteredImage
    = new BufferedImage(image.getWidth(), image.getHeight(), image.getType());
op.filter(image, filteredImage);
```

有些图像操作可以适当地（通过`op.filter(image, image)`方法）转换一个图像，但是大多数的图像操作都做不到这一点。

以下五个类实现了`BufferedImageOp`接口。

```
AffineTransformOp
RescaleOp
LookupOp
ColorConvertOp
ConvolveOp
```

`AffineTransformOp`类用于对各个像素执行仿射变换。例如，下面的代码就说明了如何使一个图像围绕着它的中心旋转。

```
AffineTransform transform = AffineTransform.getRotateInstance(Math.toRadians(angle),
    image.getWidth() / 2, image.getHeight() / 2);
AffineTransformOp op = new AffineTransformOp(transform, interpolation);
op.filter(image, filteredImage);
```

`AffineTransformOp`构造器需要一个仿射变换和一个渐变变换策略。如果源像素在目标像素之间的某处会发生变换的话，那么就必须使用渐变变换策略来确定目标图像的像素。例如，如果旋转源像素，那么通常它将无法精确地落在目标像素上。有两种渐变变换策略：`AffineTransformOp.TYPE_BILINEAR`和`AffineTransformOp.TYPE_NEAREST_NEIGHBOR`。双线性（Bilinear）渐变变换需要的时间较长，但是变换的效果却更好。

使用程序清单 7-7 的程序，可以把一个图像旋转 5 度（参见图 7-29）。

`RescaleOp`用于为图像中的所有的颜色构件执行一个调整其大小的变换操作（透明度构件不受影响）：

$$x_{\text{new}} = a \cdot x + b$$

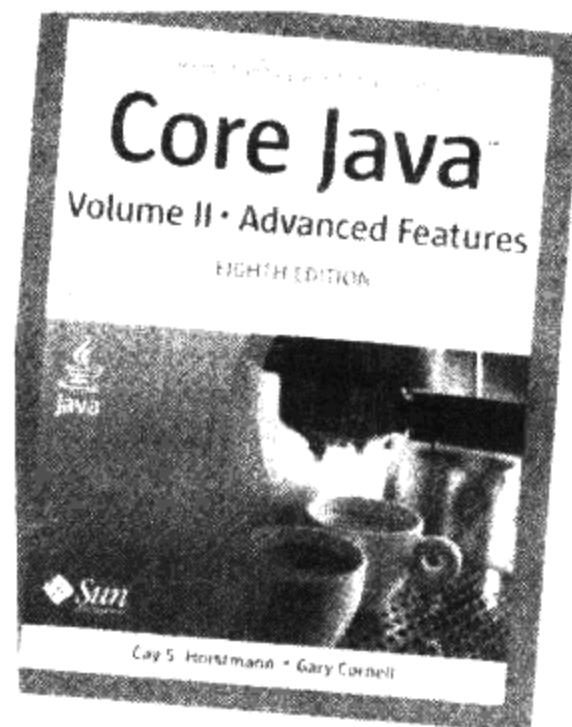


图 7-29 一个旋转 5° 的图像

用 $a > 1$ 进行调整，那么调整后的效果是使图像变亮。可以通过设定调整大小的参数和可选的绘图提示来构建RescaleOp。在程序清单7-7中，我们使用下面的设置：

```
float a = 1.1f;
float b = 0.0f;
RescaleOp op = new RescaleOp(a, b, null);
```

也可以为每个颜色构件提供单独的缩放值，参见API说明。

使用LookupOp操作，可以设定样本值的一个任意的映射操作。你提供一张表格，用于设定每一个样本值应该如何进行映射操作。在示例程序中，我们计算了所有颜色的反，即将颜色c变成 $255 - c$ 。

LookupOp构造器需要一个类型是LookupTable的对象和一个选项提示映射表。LookupTable是抽象类，其有两个实体子类：ByteLookupTable和ShortLookupTable。因为RGB颜色值是由字节组成的，所以ByteLookupTable类应该就够了。但是，考虑到在[http://bugs.sun.com/bugdatabase/view\\_bug.do?bug\\_id=6183251](http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6183251)中描述的缺陷，我们将使用ShortLookupTable。下面的代码说明了我们在程序清单中是如何构建一个LookupOp类的：

```
short negative[] = new short[256];
for (int i = 0; i < 256; i++) negative[i] = (short) (255 - i);
ShortLookupTable table = new ShortLookupTable(0, negative);
LookupOp op = new LookupOp(table, null);
```

此项操作可以分别应用于每个颜色值，但是不能应用于透明度值。也可以为每个颜色构件提供单独的查找表，参见API说明。



**注意：**不能将LookupOp用于带有索引颜色模型的图像。（在这些图像中，每个样本值都是调色板中的一个偏移值。）

ColorConvertOp对于颜色空间的转换非常有用。我们不准备在这里讨论这个问题了。

ConvolveOp是功能最强大的转换操作，它用于执行卷积变换。我们不想过分深入地介绍卷积变换的详尽细节。不过，其基本概念还是比较简单的。我们不妨看一下模糊过滤器的例子（见图7-30）。

这种模糊的效果是通过用像素和该像素临近的8个像素的平均值来取代每一个像素值而达到的。凭借直观感觉，就可以知道为什么这种变换操作能使得图像变模糊了。从数学理论上来说，这种平均法可以表示为一个以下面这个矩阵为内核的卷积变换操作：

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

卷积变换操作的内核是一个矩阵，用以说明在临近的

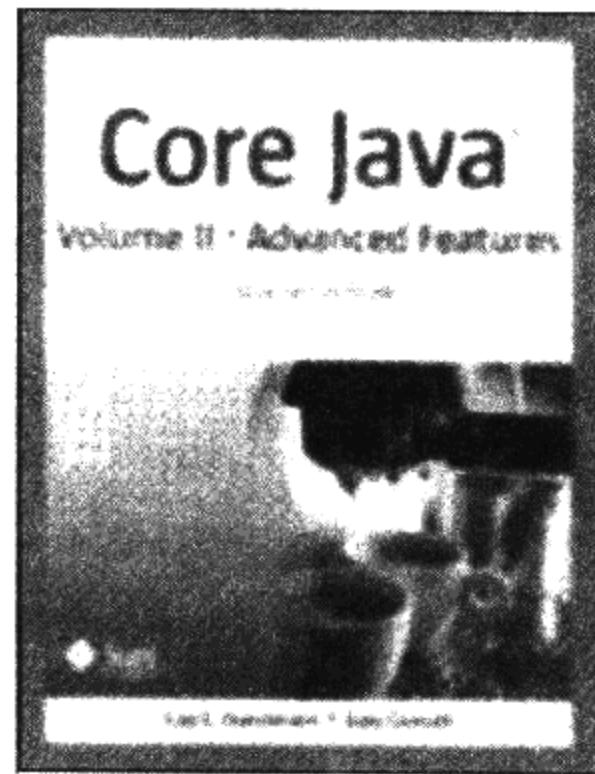


图7-30 对图像进行模糊处理

像素点上应用的加权值。应用上面的内核进行卷积变换，就会产生一个模糊图像。下面这个不同的内核用以进行图像的边缘检测，查找图像颜色变化的区域：

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

边缘检测是在分析摄影图片时使用的一项非常重要的技术（参见图7-31）。

如果要构建一个卷积变换操作，首先应为矩阵内核建立一个含有内核值的数组，并且构建一个Kernel对象。接着，根据内核对象建立一个ConvolveOp对象，进而执行过滤操作。

```
float[] elements =
{
    0.0f, -1.0f, 0.0f,
    -1.0f, 4.0f, -1.0f,
    0.0f, -1.0f, 0.0f
};
Kernel kernel = new Kernel(3, 3, elements);
ConvolveOp op = new ConvolveOp(kernel);
op.filter(image, filteredImage);
```

使用程序清单7-7的程序，用户可以装载一个GIF或者JPEG图像，并且执行我们已经介绍过的各种图像处理的操作。由于Java 2D API的图像处理的功能很强大，下面的程序非常简单。

### 程序清单7-7 ImageProcessingTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.awt.geom.*;
4. import java.awt.image.*;
5. import java.io.*;
6. import javax.imageio.*;
7. import javax.swing.*;
8. import javax.swing.filechooser.*;
9.
10. /**
11. * This program demonstrates various image processing operations.
12. * @version 1.03 2007-08-16
13. * @author Cay Horstmann
14. */
15. public class ImageProcessingTest
16. {
17.     public static void main(String[] args)
18.     {
19.         EventQueue.invokeLater(new Runnable()
20.         {
21.             public void run()
22.             {
```

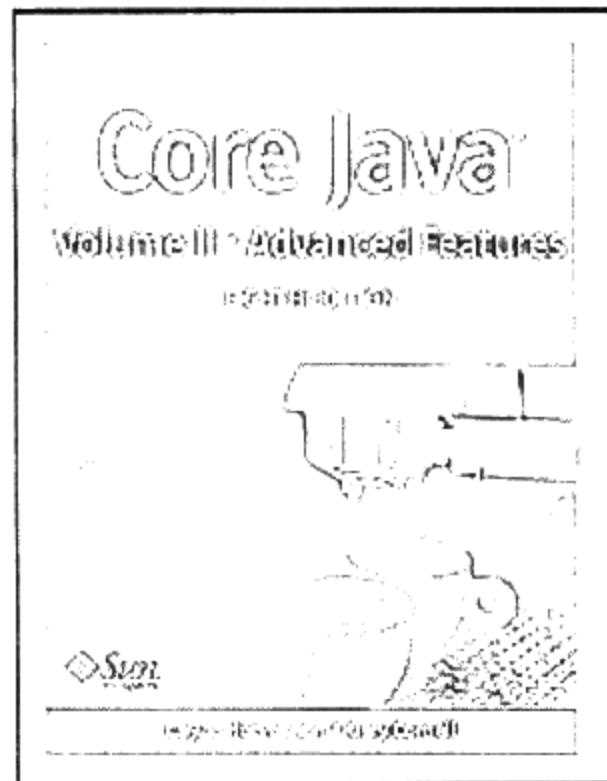


图7-31 边缘检测

```
23.         JFrame frame = new ImageProcessingFrame();
24.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
25.         frame.setVisible(true);
26.     }
27. });
28. }
29. */
30. /**
31. * This frame has a menu to load an image and to specify various transformations, and
32. * a component to show the resulting image.
33. */
34. */
35. class ImageProcessingFrame extends JFrame
36. {
37.     public ImageProcessingFrame()
38.     {
39.         setTitle("ImageProcessingTest");
40.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
41.
42.         add(new JComponent()
43.         {
44.             public void paintComponent(Graphics g)
45.             {
46.                 if (image != null) g.drawImage(image, 0, 0, null);
47.             }
48.         });
49.
50.         JMenu fileMenu = new JMenu("File");
51.         JMenuItem openItem = new JMenuItem("Open");
52.         openItem.addActionListener(new ActionListener()
53.         {
54.             public void actionPerformed(ActionEvent event)
55.             {
56.                 openFileDialog();
57.             }
58.         });
59.         fileMenu.add(openItem);
60.
61.         JMenuItem exitItem = new JMenuItem("Exit");
62.         exitItem.addActionListener(new ActionListener()
63.         {
64.             public void actionPerformed(ActionEvent event)
65.             {
66.                 System.exit(0);
67.             }
68.         });
69.         fileMenu.add(exitItem);
70.
71.         JMenu editMenu = new JMenu("Edit");
72.         JMenuItem blurItem = new JMenuItem("Blur");
73.         blurItem.addActionListener(new ActionListener()
74.         {
75.             public void actionPerformed(ActionEvent event)
76.             {
77.                 float weight = 1.0f / 9.0f;
78.                 float[] elements = new float[9];
79.                 for (int i = 0; i < 9; i++)
```

```
80.         elements[i] = weight;
81.         convolve(elements);
82.     }
83. });
84. editMenu.add(blurItem);
85.
86. JMenuItem sharpenItem = new JMenuItem("Sharpen");
87. sharpenItem.addActionListener(new ActionListener()
88. {
89.     public void actionPerformed(ActionEvent event)
90.     {
91.         float[] elements = { 0.0f, -1.0f, 0.0f, -1.0f, 5.f, -1.0f, 0.0f, -1.0f, 0.0f };
92.         convolve(elements);
93.     }
94. });
95. editMenu.add(sharpenItem);
96.
97. JMenuItem brightenItem = new JMenuItem("Brighten");
98. brightenItem.addActionListener(new ActionListener()
99. {
100.    public void actionPerformed(ActionEvent event)
101.    {
102.        float a = 1.1f;
103.        // float b = 20.0f;
104.        float b = 0;
105.        RescaleOp op = new RescaleOp(a, b, null);
106.        filter(op);
107.    }
108. });
109. editMenu.add(brightenItem);
110.
111. JMenuItem edgeDetectItem = new JMenuItem("Edge detect");
112. edgeDetectItem.addActionListener(new ActionListener()
113. {
114.     public void actionPerformed(ActionEvent event)
115.     {
116.         float[] elements = { 0.0f, -1.0f, 0.0f, -1.0f, 4.f, -1.0f, 0.0f, -1.0f, 0.0f };
117.         convolve(elements);
118.     }
119. });
120. editMenu.add(edgeDetectItem);
121.
122. JMenuItem negativeItem = new JMenuItem("Negative");
123. negativeItem.addActionListener(new ActionListener()
124. {
125.     public void actionPerformed(ActionEvent event)
126.     {
127.         short[] negative = new short[256 * 1];
128.         for (int i = 0; i < 256; i++)
129.             negative[i] = (short) (255 - i);
130.         ShortLookupTable table = new ShortLookupTable(0, negative);
131.         LookupOp op = new LookupOp(table, null);
132.         filter(op);
133.     }
134. });
135. editMenu.add(negativeItem);
136.
```

```
137.     JMenuItem rotateItem = new JMenuItem("Rotate");
138.     rotateItem.addActionListener(new ActionListener()
139.     {
140.         public void actionPerformed(ActionEvent event)
141.         {
142.             if (image == null) return;
143.             AffineTransform transform = AffineTransform.getRotateInstance(
144.                 Math.toRadians(5), image.getWidth() / 2, image.getHeight() / 2);
145.             AffineTransformOp op = new AffineTransformOp(transform,
146.                 AffineTransformOp.TYPE_BICUBIC);
147.             filter(op);
148.         }
149.     });
150.     editMenu.add(rotateItem);
151.
152.     JMenuBar menuBar = new JMenuBar();
153.     menuBar.add(fileMenu);
154.     menuBar.add(editMenu);
155.     setJMenuBar(menuBar);
156. }
157.
158. /**
159. * Open a file and load the image.
160. */
161. public void openFile()
162. {
163.     JFileChooser chooser = new JFileChooser();
164.     chooser.setCurrentDirectory(new File("."));
165.     String[] extensions = ImageIO.getReaderFileSuffixes();
166.     chooser.setFileFilter(new FileNameExtensionFilter("Image files", extensions));
167.     int r = chooser.showOpenDialog(this);
168.     if (r != JFileChooser.APPROVE_OPTION) return;
169.
170.     try
171.     {
172.         Image img = ImageIO.read(chooser.getSelectedFile());
173.         image = new BufferedImage(img.getWidth(null), img.getHeight(null),
174.             BufferedImage.TYPE_INT_RGB);
175.         image.getGraphics().drawImage(img, 0, 0, null);
176.     }
177.     catch (IOException e)
178.     {
179.         JOptionPane.showMessageDialog(this, e);
180.     }
181.     repaint();
182. }
183.
184. /**
185. * Apply a filter and repaint.
186. * @param op the image operation to apply
187. */
188. private void filter(BufferedImageOp op)
189. {
190.     if (image == null) return;
191.     image = op.filter(image, null);
192.     repaint();
193. }
```

```

194.
195. /**
196. * Apply a convolution and repaint.
197. * @param elements the convolution kernel (an array of 9 matrix elements)
198. */
199. private void convolve(float[] elements)
200. {
201.     Kernel kernel = new Kernel(3, 3, elements);
202.     ConvolveOp op = new ConvolveOp(kernel);
203.     filter(op);
204. }
205.
206. private BufferedImage image;
207. private static final int DEFAULT_WIDTH = 400;
208. private static final int DEFAULT_HEIGHT = 400;
209. }

```

### **API** `java.awt.image.BufferedImageOp 1.2`

- `BufferedImage filter(BufferedImage source, BufferedImage dest)`

将图像操作应用于源图像，并且将操作的结果存放在目标图像中。如果`dest`为null，一个新的目标图像将被创建。该目标图像将被返回。

### **API** `java.awt.image.AffineTransformOp 1.2`

- `AffineTransformOp(AffineTransform t, int interpolationType)`

构建一个仿射变换操作符。渐变变换的类型是`TYPE_BILINEAR`、`TYPE_BICUBIC`或者`TYPE_NEAREST_NEIGHBOR`中的一个。

### **API** `java.awt.image.RescaleOp 1.2`

- `RescaleOp(float a, float b, RenderingHints hints)`
- `RescaleOp(float[] as, float[] bs, RenderingHints hints)`

构建一个尺寸调整的操作符，它会执行缩放操作 $x_{\text{new}} = a \cdot x + b$ 。当使用第一个构造器时，所有的颜色构件（但不包括透明度构件）都将按照相同的系数进行缩放。当使用第二个构造器时，可以为每个颜色构件提供单独的值，在这种情况下，透明度构件不受影响，或者为每个颜色构件和透明度构件都提供单独的值。

### **API** `java.awt.image.LookupOp 1.2`

- `LookupOp(LookupTable table, RenderingHints hints)`

为给定的查找表构建一个查找操作符。

### **API** `java.awt.image.ByteLookupTable 1.2`

- `ByteLookupTable(int offset, byte[] data)`
- `ByteLookupTable(int offset, byte[][] data)`

为转化byte值构建一个字节查找表。在查找之前，从输入中减去偏移量。在第一个构造器中的值将提供给所有的颜色构件，但不包括透明度构件。当使用第二个构造器时，可

以为每个颜色构件提供单独的值，在这种情况下，透明度构件不受影响，或者为每个颜色构件和透明度构件都提供单独的值。

#### **java.awt.image.ShortLookupTable 1.2**

- `ShortLookupTable(int offset, short[] data)`
- `ShortLookupTable(int offset, short[][] data)`

为转化short值构建一个字节查找表。在查找之前，从输入中减去偏移量。在第一个构造器中的值将提供给所有的颜色构件，但不包括透明度构件。当使用第二个构造器时，可以为每个颜色构件提供单独的值，在这种情况下，透明度构件不受影响，或者为每个颜色构件和透明度构件都提供单独的值。

#### **java.awt.image.ConvolveOp 1.2**

- `ConvolveOp(Kernel kernel)`
- `ConvolveOp(Kernel kernel, int edgeCondition, RenderingHints hints)`

构建一个卷积变换操作符。边界条件是EDGE\_NO\_OP和EDGE\_ZERO\_FILL两种方式之一。由于边缘值没有足够的临近值来进行卷积变换的计算，所以边缘值必须被特殊处理。其默认值是EDGE\_ZERO\_FILL。

#### **java.awt.image.Kernel 1.2**

- `Kernel(int width, int height, float[] matrixElements)`

为指定的矩阵构建一个内核。

## 7.12 打印

最初的JDK根本不支持打印操作。它不可能从applets中进行打印操作。如果想在应用中使用打印操作，必须获得第三方的类库。JDK1.1提供了非常简单的打印支持，仅仅能够产生简单的打印输出，不过只要你对打印的质量没有太高的要求也就行了。JDK 1.1的打印模型被设计为允许浏览器供应商打印出现在一个网页中的applet外观（然而，浏览器供应商对此并不感兴趣）。

Java SE 1.2 开始推出了一种强大的打印模型，它和Java 2D图形实现了完全的集成。JDK1.4增加了许多重要的增强特性，比如，查找打印机的特性和用于服务器端打印管理的流式打印作业等。

在本节中，我们将介绍如何在单页纸上轻松地打印出一幅图画，如何来管理多页打印输出，还有如何利用Java 2D 图像模型的出色特性，以及如何方便地产生一个打印预览对话框。



**注意：**Java 平台还支持用户界面构件的打印功能。我们不会介绍这个问题，因为这个问题只有浏览器的实现者、屏幕图像抓取者等人最感兴趣。关于打印构件更详尽的信息，请访问网址：<http://java.sun.com/developer/onlineTraining/Programming/JDCBook/render.html>。

### 7.12.1 图形打印

在本节中，我们将处理最常用的打印情景，即打印一个2D图形，当然该图形可以含有不同字体组成的文本，甚至可能完全由文本构成。

如果要生成打印输出，必须完成下面这两个任务：

- 提供一个实现了Printable接口的对象。
- 启动一个打印作业。

Printable接口只有下面一个方法：

```
int print(Graphics g, PageFormat format, int page)
```

每当打印引擎需要对某一页面进行排版以便打印时，都要调用这个方法。你的代码绘制了准备在图形上下文上打印的文本和图像，页面排版显示了纸张的大小和页边距，页号显示了将要打印的页。

如果要启动一个打印作业，需要使用PrinterJob类。首先，应该调用静态方法getPrinterJob来获取一个打印作业对象。然后，设置要打印的Printable对象。

```
Printable canvas = ...;
PrinterJob job = PrinterJob.getPrinterJob();
job.setPrintable(canvas);
```

 **警告：**PrintJob这个类处理JDK1.1风格的打印操作，这个类已经被弃用了。请不要把PrinterJob类同其混淆在一起。

在开始打印作业之前，应该调用printDialog方法来显示一个打印对话框（见图7-32）。这个对话框为用户提供了机会去选择要使用的打印机（在多打印机可被利用的情况下），选择将要打印的页的范围，以及选择打印机的各种设置。

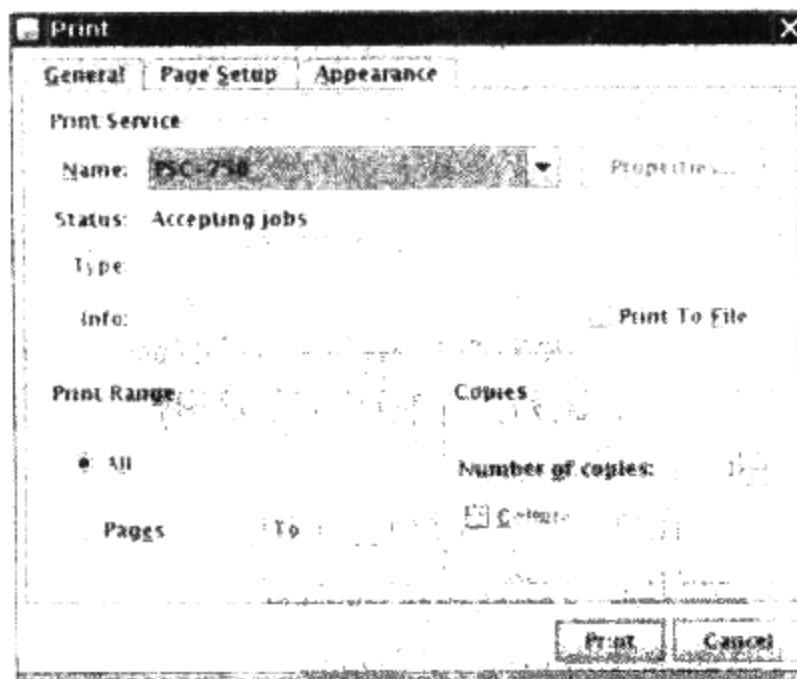


图7-32 一个跨平台的打印对话框

可以在一个实现了PrintRequestAttributeSet接口的类的对象中收集到各种打印机的设置，例如HashPrintRequestAttributeSet类：

```
HashPrintRequestAttributeSet attributes = new HashPrintRequestAttributeSet();
```

你可以添加属性设置，并且把attributes对象传递给printDialog方法。

如果用户点击OK，那么printDialog方法将返回true；如果用户关掉对话框，那么该方法将返回false。如果用户接受了设置，那么就可以调用PrinterJob类的print方法来启动打印进程。print方法可能会抛出一个PrinterException异常。下面是打印代码的基本框架：

```
if (job.printDialog(attributes))
{
    try
    {
        job.print(attributes);
    }
    catch (PrinterException exception)
    {
        ...
    }
}
```

 **注意：**在JDK1.4之前，打印系统使用的都是宿主平台本地的打印和页面设置对话框。要展示本地打印对话框，可以调用没有任何参数的printDialog方法。（不存在任何方式可以用来将用户设置收集到一个属性集中。）

在执行打印操作时，PrinterJob类的print方法不断地调用和此项打印作业相关的Printable对象的print方法。

由于打印作业不知道用户想要打印的页数，它只是不断地调用print方法。只要该print方法的返回值是Printable.PAGE\_EXISTS，打印作业就不断地产生输出页。当print方法返回Printable.NO\_SUCH\_PAGE时，打印作业就停止。

 **警告：**打印作业传递到print方法的打印页号是从0开始的。

因此，在打印操作完成之前，打印作业并不拥有一个准确的打印页数。为此，打印对话框无法显示正确的页码范围，而只能显示“Pages 1 to 1”（从第一页到第一页）。在下一节中，我们将介绍如何通过为打印作业提供一个Book对象来避免这个缺陷。

在打印的过程中，打印作业反复地调用Printable对象的print方法。打印作业被允许对同一页面多次调用print方法。因此不应该在print方法内对页进行计数，而是应始终依赖于页码参数来进行计数操作。打印作业之所以能够对某一页反复地调用print方法是有一定道理的。一些打印机尤其是在点阵式打印机和喷墨式打印机，都使用条带打印技术。它们在打印纸上一条接着一条的打印。即使是激光打印机，打印作业都有可能使用条带打印技术，其每次打印一整页。这为打印作业提供了一种对假脱机文件的大小进行管理的方法。

如果打印作业需要printable对象打印一个条带，那么它可以将图形上下文的剪切区域设置为所需要的条带，并且调用print方法。它的绘图操作将按照条带矩形区域进行剪切，同时，只有在条带中显示的那些图形元素才会被绘制出来。你的print方法不必晓得该过程，但是请注意：它不应该对剪切区域产生任何干扰。

**X 警告：**你的print方法获得的Graphics对象也是按照页边距进行剪切的。如果替换了剪切区域，那么就可以在边距外面进行绘图操作。尤其是在打印机的绘图上下文中，剪切区域是被严格遵守的。如果想进一步地限制剪切区域，可以调用clip方法，而不是setClip方法。如果必须要移除一个剪切区域，那么请务必在你的print方法开始处调用getClip方法，并还原该剪切区域。

print方法的PageFormat参数包含有关被打印页的信息。getWidth方法和getHeight方法返回该纸张的大小，它以磅为计量单位。1磅等于 $1/72$ 英寸<sup>⊖</sup>。例如，A4纸的大小大约是 $595 \times 842$ 磅，美国人使用的信纸大小为 $612 \times 792$ 磅。

磅是美国印刷业中通用的计量单位。让世界上其他地方的人感到苦恼的是，打印软件包使用磅这种计量单位仅仅为了两个目的，即纸张的大小和纸张的页边距都是用磅来计量的。对所有的图形上下文来说，默认的计量单位就是1磅。你可以在本节后面的示例程序中证明这一点。该程序打印了两行文本，这两行文本之间的距离为72磅。运行一下示例程序，并且测量一下基准线之间的距离。它们之间的距离恰好是1英寸或是25.4毫米。

PageFormat类的getWidth和getHeight方法给你的信息是完整的页面大小。并不是所有的纸张区域都会被用来打印。通常的情况是，用户会选择页边距，即使他们没有选择页边距，打印机也需要用某种方法来夹住纸张，因此在纸张的周围就出现了一个不能打印的区域。

getImageableWidth和getImageableHeight方法可以告诉你能够真正用来打印的区域的大小。然而，页边距没有必要是对称的，所以还必须知道可打印区域的左上角，见图7-33。可以通过调用getImageableX和getImageableY方法来获得。

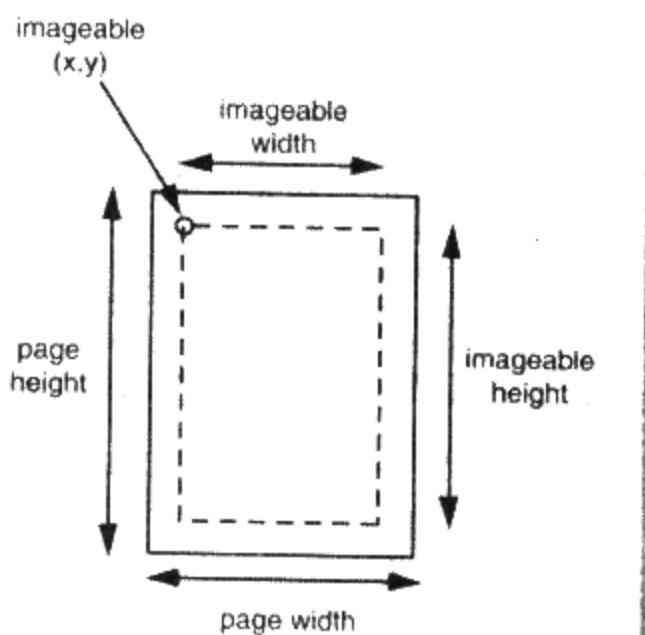


图7-33 页面格式计量

**!** 提示：在print方法中接收到的图形上下文是经过剪切后的图形上下文，它不包括页边距。但是，坐标系统的原点仍然是纸张的左上角。应该将该坐标系统转换成可打印区域的左上角，并以其为起点。只需让print方法以下面的代码开始即可：

```
g.translate(pageFormat.getImageableX(), pageFormat.getImageableY());
```

如果想让用户来选择设定页边距的值，或者让用户在不设置其他打印属性的情况下进行纵向和横向打印方式之间转换的话，应该调用PrinterJob类的pageDialog方法。

```
PageFormat format = job.pageDialog(attributes);
```

<sup>⊖</sup> 1英寸=0.0254米。

注意：打印对话框中有一个选项卡包含了页面设置对话框（参见图7-34）。在打印前，你仍然可以为用户提供一个选项来设置页面格式。特别是，如果你的程序给出了一个待打印页面的“所见即所得”的显示屏，那么就更应该提供这样的选项。`pageDialog`方法返回了一个含有用户设置的`PageFormat`对象。

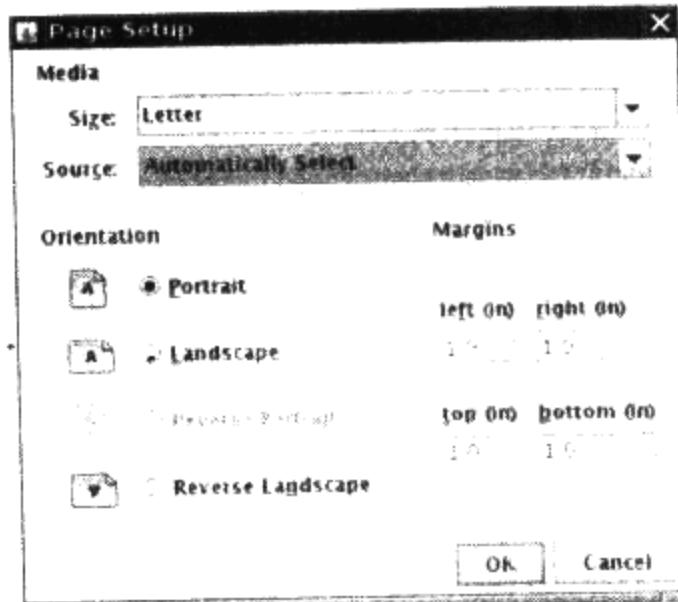


图7-34 一个跨平台的页面设置对话框

程序清单7-8显示了如何在屏幕和打印页面上绘制相同的一组形状的方法。`Jpanel`类的一个子类实现了`Printable`接口，该类中的`paintComponent`和`print`方法都调用了相同的方法来执行实际的绘图操作。

```
class PrintPanel extends JPanel implements Printable
{
    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        drawPage(g2);
    }

    public int print(Graphics g, PageFormat pf, int page)
        throws PrinterException
    {
        if (page >= 1) return Printable.NO_SUCH_PAGE;
        Graphics2D g2 = (Graphics2D) g;
        g2.translate(pf.getImageableX(), pf.getImageableY());
        drawPage(g2);
        return Printable.PAGE_EXISTS;
    }

    public void drawPage(Graphics2D g2)
    {
        // shared drawing code goes here
        ...
    }
}
```

该示例代码显示并且打印了图7-20。即消息“Hello, World”的框架，它被用作一个线条图

案的剪切区域（参见图7-22）。

可以点击Print按钮来开始打印，或者点击页面设置按钮来打开页面设置对话框。程序清单7-8显示了它的代码。

 注意：为了显示本地页面设置对话框，需要将默认的PageFormat对象传递给pageDialog方法。该方法会克隆这个对象，并根据用户在对话框中的选择来修改它，然后返回这个克隆的对象。

```
PageFormat defaultFormat = printJob.defaultPage();
PageFormat selectedFormat = printJob.pageDialog(defaultFormat);
```

### 程序清单7-8 PrintTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.awt.font.*;
4. import java.awt.geom.*;
5. import java.awt.print.*;
6. import javax.print.attribute.*;
7. import javax.swing.*;

8.
9. /**
10. * This program demonstrates how to print 2D graphics
11. * @version 1.12 2007-08-16
12. * @author Cay Horstmann
13. */
14. public class PrintTest
15. {
16.     public static void main(String[] args)
17.     {
18.         EventQueue.invokeLater(new Runnable()
19.         {
20.             public void run()
21.             {
22.                 JFrame frame = new PrintTestFrame();
23.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24.                 frame.setVisible(true);
25.             }
26.         });
27.     }
28. }
29.
30. /**
31. * This frame shows a panel with 2D graphics and buttons to print the graphics and to
32. * set up the page format.
33. */
34. class PrintTestFrame extends JFrame
35. {
36.     public PrintTestFrame()
37.     {
38.         setTitle("PrintTest");
39.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
40.
41.         canvas = new PrintComponent();
42.         add(canvas, BorderLayout.CENTER);
```

```
43.  
44.     attributes = new HashPrintRequestAttributeSet();  
45.  
46.     JPanel buttonPanel = new JPanel();  
47.     JButton printButton = new JButton("Print");  
48.     buttonPanel.add(printButton);  
49.     printButton.addActionListener(new ActionListener()  
50.     {  
51.         public void actionPerformed(ActionEvent event)  
52.         {  
53.             try  
54.             {  
55.                 PrinterJob job = PrinterJob.getPrinterJob();  
56.                 job.setPrintable(canvas);  
57.                 if (job.printDialog(attributes)) job.print(attributes);  
58.             }  
59.             catch (PrinterException e)  
60.             {  
61.                 JOptionPane.showMessageDialog(PrintTestFrame.this, e);  
62.             }  
63.         }  
64.     });  
65.  
66.     JButton pageSetupButton = new JButton("Page setup");  
67.     buttonPanel.add(pageSetupButton);  
68.     pageSetupButton.addActionListener(new ActionListener()  
69.     {  
70.         public void actionPerformed(ActionEvent event)  
71.         {  
72.             PrinterJob job = PrinterJob.getPrinterJob();  
73.             job.pageDialog(attributes);  
74.         }  
75.     });  
76.  
77.     add(buttonPanel, BorderLayout.NORTH);  
78. }  
79.  
80. private PrintComponent canvas;  
81. private PrintRequestAttributeSet attributes;  
82.  
83. private static final int DEFAULT_WIDTH = 300;  
84. private static final int DEFAULT_HEIGHT = 300;  
85. }  
86.  
87. /**  
88. * This component generates a 2D graphics image for screen display and printing.  
89. */  
90. class PrintComponent extends JComponent implements Printable  
91. {  
92.     public void paintComponent(Graphics g)  
93.     {  
94.         Graphics2D g2 = (Graphics2D) g;  
95.         drawPage(g2);  
96.     }  
97.  
98.     public int print(Graphics g, PageFormat pf, int page) throws PrinterException  
99.     {
```

```

100.     if (page >= 1) return Printable.NO_SUCH_PAGE;
101.     Graphics2D g2 = (Graphics2D) g;
102.     g2.translate(pf.getImageableX(), pf.getImageableY());
103.     g2.draw(new Rectangle2D.Double(0, 0, pf.getImageableWidth(), pf.getImageableHeight()));
104.
105.     drawPage(g2);
106.     return Printable.PAGE_EXISTS;
107. }
108.
109. /**
110. * This method draws the page both on the screen and the printer graphics context.
111. * @param g2 the graphics context
112. */
113. public void drawPage(Graphics2D g2)
114. {
115.     FontRenderContext context = g2.getFontRenderContext();
116.     Font f = new Font("Serif", Font.PLAIN, 72);
117.     GeneralPath clipShape = new GeneralPath();
118.
119.     TextLayout layout = new TextLayout("Hello", f, context);
120.     AffineTransform transform = AffineTransform.getTranslateInstance(0, 72);
121.     Shape outline = layout.getOutline(transform);
122.     clipShape.append(outline, false);
123.
124.     layout = new TextLayout("World", f, context);
125.     transform = AffineTransform.getTranslateInstance(0, 144);
126.     outline = layout.getOutline(transform);
127.     clipShape.append(outline, false);
128.
129.     g2.draw(clipShape);
130.     g2.clip(clipShape);
131.
132.     final int NLINES = 50;
133.     Point2D p = new Point2D.Double(0, 0);
134.     for (int i = 0; i < NLINES; i++)
135.     {
136.         double x = (2 * getWidth() * i) / NLINES;
137.         double y = (2 * getHeight() * (NLINES - 1 - i)) / NLINES;
138.         Point2D q = new Point2D.Double(x, y);
139.         g2.draw(new Line2D.Double(p, q));
140.     }
141. }
142. }

```

### java.awt.print.Printable 1.2

- int print(Graphics g, PageFormat format, int pageNumber)

绘制一个页面，并且返回PAGE\_EXISTS，或者返回NO\_SUCH\_PAGE。

参数：g 在上面绘制页面的图形上下文

format 要绘制的页面的格式

pageNumber 请求显示的页面的号码

**API** **java.awt.print.PrinterJob 1.2**

- static PrinterJob getPrinterJob()

返回一个打印机作业对象。

- PageFormat defaultPage()

为该打印机返回默认的页面格式。

- boolean printDialog(PrintRequestAttributeSet attributes)

- boolean printDialog()

打开打印对话框，允许用户选择将要打印的页面，并且改变打印设置。第一个方法将显示一个跨平台的打印对话框，第二个方法将显示一个本地的打印对话框。第一个方法修改了 attributes 对象来反映用户的设置。如果用户接受默认的设置，两种方法都返回 true。

- PageFormat pageDialog(PrintRequestAttributeSet attributes)

- PageFormat pageDialog(PageFormat defaults)

显示页面设置对话框。第一个方法将显示一个跨平台的对话框，第二个方法将显示一个本地的页面设置对话框。两种方法都返回了一个 PageFormat 对象，对象的格式是用户在对话框中请求得到的格式。第一个方法修改了 attributes 对象以反映用户的设置。第二个对象不修改 defaults 对象。

- void setPrintable(Printable p)

- void setPrintable(Printable p, PageFormat format)

设置该打印作业的 Printable 和一个可选的页面格式。

- void print()

- void print(PrintRequestAttributeSet attributes)

反复地调用 print 方法，以打印当前的 Printable，并将绘制的页面发送给打印机，直到没有更多的页面需要打印为止。

**API** **java.awt.print.PageFormat 1.2**

- double getWidth()

- double getHeight()

返回页面的宽度和高度。

- double getImageableWidth()

- double getImageableHeight()

返回可打印区域的页面宽度和高度。

- double getImageableX()

- double getImageableY()

返回可打印区域的左上角的位置。

- int getOrientation()

返回 PORTARIT、LANDSCAPE 和 REVERSE\_LANDSCAPE 三者之一。页面打印的方向对程序员来说是透明的，因为打印格式和图形上下文自动地反映了页面的打印方向。

### 7.12.2 打印多页文件

在实际的打印操作中，通常不应该将未处理过的Printable对象传递给打印作业。相反，应该获取一个实现了Pageable接口的类的对象。Java平台提供了这样的一个被称之为Book的类。一本书是由很多章节组成的，而每个章节都是一个Printable对象。可以通过添加Printable对象和相应的页数来构建一个Book对象。

```
Book book = new Book();
Printable coverPage = . . .;
Printable bodyPages = . . .;
book.append(coverPage, pageFormat); // append 1 page
book.append(bodyPages, pageFormat, pageCount);
```

然后，可以使用 setPageable方法把Book对象传递给打印作业。

```
printJob.setPageable(book);
```

现在，打印作业就知道将要打印的确切页数了。然后，打印对话框显示一个准确的页面范围，用户可以选择整个页面范围或可选择它的一个子范围。

**X** **警告：**当打印作业调用Printable章节的print方法时，它传递的是该书的当前页码，而不是每个章节的页码。这让人非常痛苦，因为每个章节必须知道它的上一个章节的页数，这样才能使得页码参数有意义。

从程序员的视角来看，使用Book类最大的挑战就是，当你打印它时，必须知道每一个章节究竟有多少页。你的Printable类需要一个布局算法，以便用来计算在打印页面上的素材布局。在打印开始前，要调用这个算法来计算出分页符的位置和页数。可以保留布局信息，从而可以在打印的过程中方便地使用它。

必须杜绝“用户已经修改过页面格式”这种情况的发生。如果用户修改了页面格式，即使是所打印的信息没有发生任何改变，也必须要重新计算布局。

程序清单7-9中显示了如何产生一个多页打印输出。该程序用很大的字符在多个页面上打印了一条消息（见图7-35）。然后，可以剪裁掉页边距，并将这些页面粘连起来，形成一个标语。

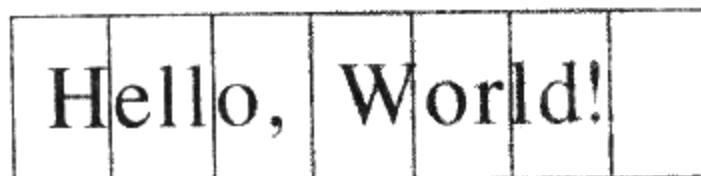


图7-35 一幅标语

Banner类的layoutPages方法用以计算页面的布局。我们首先展示了一个字体为72磅的消息字符串。然后，我们计算产生的字符串的高度，并且将其与该页面的可打印高度进行比较。我们根据这两个高度值得出一个比例因子。当打印该字符串时，我们按照比例因子来放大此字符串。

**X** 警告：如果要准确地布局打印信息，通常需要获得打印机的图形上下文。遗憾的是，只有当打印真正开始时，才能获得打印机的图形上下文。在我们的示例程序中使用的是屏幕的图形上下文，并且希望屏幕的字体度量单位与打印机的相匹配。

Banner类的getPageCount方法首先调用布局方法。然后，扩展字符串的宽度，并且将该宽度除以每一页的可打印宽度。得到的商值上取整，就是要打印的页数。

由于字符可能断开分布到多个页面上，所以上面打印标语的操作好像会有困难。然而，感谢Java 2D API提供的强大功能，这个问题现在不过是小菜一碟。当需要打印某一页时，我们只需要调用Graphics2D类的translate方法，将字符串的左上角向左平移。接着，设置一个大小是当前页面的剪切矩形（参见图7-36）。最后，我们用布局方法计算出的比例因子来扩展该图形上下文。

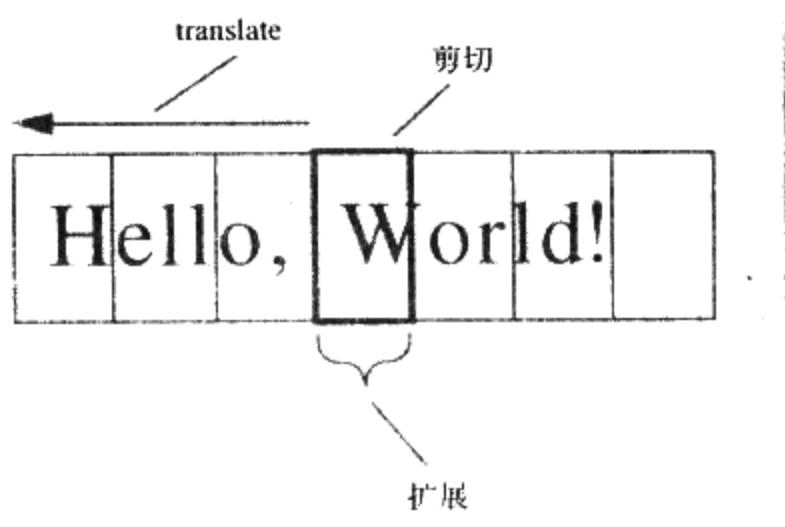


图7-36 打印一个标语页面

这个例子显示了图形变换操作的强大功能。绘图代码很简单，而图形变换操作负责执行将图形放到恰当位置上的所有操作。最后，剪切操作负责将落在页面外面的图像剪切掉。在下一节中，你将看到另一种必须使用变换操作的情况，即显示页面的打印预览。

### 7.12.3 打印预览

大多数专业的程序都有一个打印预览机制，使用户能够在显示屏幕上看到要打印的页面，这样就不必为不满意的打印输出而浪费纸张了。Java平台的打印类并没有提供一个标准的“打印预览”对话框，但是可以非常容易地设计出自己的打印预览对话框（参见图7-37）。在本节中，我们将要介绍如何来设计自己的打印预览对话框。程序清单7-9中的PrintPreviewDialog类是一个完全通用的类，你可以复用它来预览任何种类的打印输出。

如果要构建一个PrintPreviewDialog类，必须提供一个Printable或Book，并且还要提供一个PageFormat对象。对话框的表层包含一个PrintPreviewCanvas。当使用Next和Previous按钮来浏览页面时，paintComponent方法将为你所请求预览的页面调用Printable对象的print方法。

通常，print方法在一个打印机的图形上下文上绘制页面上下文。但是，我们提供了屏幕的图形上下文，并进行了合适的缩放，这样，打印的整个页面就可以被纳入到较小的屏幕矩形中。

```
float xoff = ...; // left of page
float yoff = ...; // top of page
float scale = ...; // to fit printed page onto screen
g2.translate(xoff, yoff);
g2.scale(scale, scale);
Printable printable = book.getPrintable(currentPage);
printable.print(g2, pageFormat, currentPage);
```

该print方法从来都不知道它实际上并不产生打印页面。它只是负责在图形上下文上进行绘制操作，从而在屏幕上产生一个微观的打印预览。这非常清楚地说明了Java 2D 图像模型的强大功能。

程序清单7-9中包括了打印标语的程序代码和打印预览对话框的程序代码。请将“Hello, World!”输入到文本框中，并且观察打印预览，然后把标语打印输出。

### 程序清单7-9 BookTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.awt.font.*;
4. import java.awt.geom.*;
5. import java.awt.print.*;
6. import javax.print.attribute.*;
7. import javax.swing.*;

8.
9. /**
10. * This program demonstrates the printing of a multipage book. It prints a "banner", by
11. * blowing up a text string to fill the entire page vertically. The program also contains a
12. * generic print preview dialog.
13. * @version 1.12 2007-08-16
14. * @author Cay Horstmann
15. */
16. public class BookTest
17. {
18.     public static void main(String[] args)
19.     {
20.         EventQueue.invokeLater(new Runnable()
21.         {
22.             public void run()
23.             {
24.                 JFrame frame = new BookTestFrame();
25.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
26.                 frame.setVisible(true);
27.             }
28.         });
29.     }
30. }
31.
32. /**
33. * This frame has a text field for the banner text and buttons for printing, page setup,
```

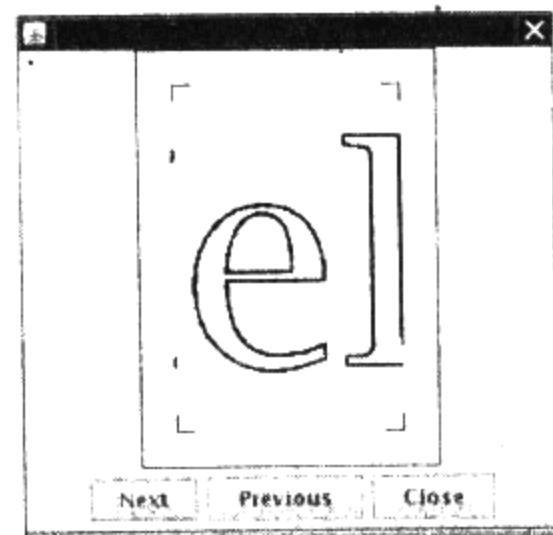


图7-37 打印预览对话框

```
34. * and print preview.
35. */
36. class BookTestFrame extends JFrame
37. {
38.     public BookTestFrame()
39.     {
40.         setTitle("BookTest");
41.
42.         text = new JTextField();
43.         add(text, BorderLayout.NORTH);
44.
45.         attributes = new HashPrintRequestAttributeSet();
46.
47.         JPanel buttonPanel = new JPanel();
48.
49.         JButton printButton = new JButton("Print");
50.         buttonPanel.add(printButton);
51.         printButton.addActionListener(new ActionListener()
52.         {
53.             public void actionPerformed(ActionEvent event)
54.             {
55.                 try
56.                 {
57.                     PrinterJob job = PrinterJob.getPrinterJob();
58.                     job.setPageable(makeBook());
59.                     if (job.printDialog(attributes))
60.                     {
61.                         job.print(attributes);
62.                     }
63.                 }
64.                 catch (PrinterException e)
65.                 {
66.                     JOptionPane.showMessageDialog(BookTestFrame.this, e);
67.                 }
68.             }
69.         });
70.
71.         JButton pageSetupButton = new JButton("Page setup");
72.         buttonPanel.add(pageSetupButton);
73.         pageSetupButton.addActionListener(new ActionListener()
74.         {
75.             public void actionPerformed(ActionEvent event)
76.             {
77.                 PrinterJob job = PrinterJob.getPrinterJob();
78.                 pageFormat = job.pageDialog(attributes);
79.             }
80.         });
81.
82.         JButton printPreviewButton = new JButton("Print preview");
83.         buttonPanel.add(printPreviewButton);
84.         printPreviewButton.addActionListener(new ActionListener()
85.         {
86.             public void actionPerformed(ActionEvent event)
87.             {
88.                 PrintPreviewDialog dialog = new PrintPreviewDialog(makeBook());
89.                 dialog.setVisible(true);
90.             }
91.         });
92.     }
93. }
```

```
91.         });
92.
93.         add(buttonPanel, BorderLayout.SOUTH);
94.         pack();
95.     }
96.
97.     /**
98.      * Makes a book that contains a cover page and the pages for the banner.
99.      */
100.    public Book makeBook()
101.    {
102.        if (pageFormat == null)
103.        {
104.            PrinterJob job = PrinterJob.getPrinterJob();
105.            pageFormat = job.defaultPage();
106.        }
107.        Book book = new Book();
108.        String message = text.getText();
109.        Banner banner = new Banner(message);
110.        int pageCount = banner.getPageCount((Graphics2D) getGraphics(), pageFormat);
111.        book.append(new CoverPage(message + " (" + pageCount + " pages)", pageFormat));
112.        book.append(banner, pageFormat, pageCount);
113.        return book;
114.    }
115.
116.    private JTextField text;
117.    private PageFormat pageFormat;
118.    private PrintRequestAttributeSet attributes;
119. }
120.
121. /**
122.  * A banner that prints a text string on multiple pages.
123. */
124. class Banner implements Printable
125. {
126.     /**
127.      * Constructs a banner
128.      * @param m the message string
129.      */
130.     public Banner(String m)
131.     {
132.         message = m;
133.     }
134.
135.     /**
136.      * Gets the page count of this section.
137.      * @param g2 the graphics context
138.      * @param pf the page format
139.      * @return the number of pages needed
140.      */
141.     public int getPageCount(Graphics2D g2, PageFormat pf)
142.     {
143.         if (message.equals("")) return 0;
144.         FontRenderContext context = g2.getFontRenderContext();
145.         Font f = new Font("Serif", Font.PLAIN, 72);
146.         Rectangle2D bounds = f.getStringBounds(message, context);
147.         scale = pf.getImageableHeight() / bounds.getHeight();
```

```
148.     double width = scale * bounds.getWidth();
149.     int pages = (int) Math.ceil(width / pf.getImageableWidth());
150.     return pages;
151. }
152.
153. public int print(Graphics g, PageFormat pf, int page) throws PrinterException
154. {
155.     Graphics2D g2 = (Graphics2D) g;
156.     if (page > getPageCount(g2, pf)) return Printable.NO_SUCH_PAGE;
157.     g2.translate(pf.getImageableX(), pf.getImageableY());
158.
159.     drawPage(g2, pf, page);
160.     return Printable.PAGE_EXISTS;
161. }
162.
163. public void drawPage(Graphics2D g2, PageFormat pf, int page)
164. {
165.     if (message.equals("")) return;
166.     page--; // account for cover page
167.
168.     drawCropMarks(g2, pf);
169.     g2.clip(new Rectangle2D.Double(0, 0, pf.getImageableWidth(), pf.getImageableHeight()));
170.     g2.translate(-page * pf.getImageableWidth(), 0);
171.     g2.scale(scale, scale);
172.     FontRenderContext context = g2.getFontRenderContext();
173.     Font f = new Font("Serif", Font.PLAIN, 72);
174.     TextLayout layout = new TextLayout(message, f, context);
175.     AffineTransform transform = AffineTransform.getTranslateInstance(0, layout.getAscent());
176.     Shape outline = layout.getOutline(transform);
177.     g2.draw(outline);
178. }
179.
180. /**
181. * Draws 1/2" crop marks in the corners of the page.
182. * @param g2 the graphics context
183. * @param pf the page format
184. */
185. public void drawCropMarks(Graphics2D g2, PageFormat pf)
186. {
187.     final double C = 36; // crop mark length = 1/2 inch
188.     double w = pf.getImageableWidth();
189.     double h = pf.getImageableHeight();
190.     g2.draw(new Line2D.Double(0, 0, 0, C));
191.     g2.draw(new Line2D.Double(0, 0, C, 0));
192.     g2.draw(new Line2D.Double(w, 0, w, C));
193.     g2.draw(new Line2D.Double(w, 0, w - C, 0));
194.     g2.draw(new Line2D.Double(0, h, 0, h - C));
195.     g2.draw(new Line2D.Double(0, h, C, h));
196.     g2.draw(new Line2D.Double(w, h, w, h - C));
197.     g2.draw(new Line2D.Double(w, h, w - C, h));
198. }
199.
200. private String message;
201. private double scale;
202. }
203.
204. /**
```

```
205. * This class prints a cover page with a title.  
206. */  
207. class CoverPage implements Printable  
208 {  
209.     /**  
210.      * Constructs a cover page.  
211.      * @param t the title  
212.     */  
213.     public CoverPage(String t)  
214.     {  
215.         title = t;  
216.     }  
217.  
218.     public int print(Graphics g, PageFormat pf, int page) throws PrinterException  
219.     {  
220.         if (page >= 1) return Printable.NO_SUCH_PAGE;  
221.         Graphics2D g2 = (Graphics2D) g;  
222.         g2.setPaint(Color.black);  
223.         g2.translate(pf.getImageableX(), pf.getImageableY());  
224.         FontRenderContext context = g2.getFontRenderContext();  
225.         Font f = g2.getFont();  
226.         TextLayout layout = new TextLayout(title, f, context);  
227.         float ascent = layout.getAscent();  
228.         g2.drawString(title, 0, ascent);  
229.         return Printable.PAGE_EXISTS;  
230.     }  
231.  
232.     private String title;  
233. }  
234.  
235. /**  
236.  * This class implements a generic print preview dialog.  
237. */  
238. class PrintPreviewDialog extends JDialog  
239 {  
240.     /**  
241.      * Constructs a print preview dialog.  
242.      * @param p a Printable  
243.      * @param pf the page format  
244.      * @param pages the number of pages in p  
245.     */  
246.     public PrintPreviewDialog(Printable p, PageFormat pf, int pages)  
247.     {  
248.         Book book = new Book();  
249.         book.append(p, pf, pages);  
250.         layoutUI(book);  
251.     }  
252.  
253.     /**  
254.      * Constructs a print preview dialog.  
255.      * @param b a Book  
256.     */  
257.     public PrintPreviewDialog(Book b)  
258.     {  
259.         layoutUI(b);  
260.     }  
261.
```

```
262. /**
263. * Lays out the UI of the dialog.
264. * @param book the book to be previewed
265. */
266. public void layoutUI(Book book)
267. {
268.     setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
269.
270.     canvas = new PrintPreviewCanvas(book);
271.     add(canvas, BorderLayout.CENTER);
272.
273.     JPanel buttonPanel = new JPanel();
274.
275.     JButton nextButton = new JButton("Next");
276.     buttonPanel.add(nextButton);
277.     nextButton.addActionListener(new ActionListener()
278.     {
279.         public void actionPerformed(ActionEvent event)
280.         {
281.             canvas.flipPage(1);
282.         }
283.     });
284.
285.     JButton previousButton = new JButton("Previous");
286.     buttonPanel.add(previousButton);
287.     previousButton.addActionListener(new ActionListener()
288.     {
289.         public void actionPerformed(ActionEvent event)
290.         {
291.             canvas.flipPage(-1);
292.         }
293.     });
294.
295.     JButton closeButton = new JButton("Close");
296.     buttonPanel.add(closeButton);
297.     closeButton.addActionListener(new ActionListener()
298.     {
299.         public void actionPerformed(ActionEvent event)
300.         {
301.             setVisible(false);
302.         }
303.     });
304.
305.     add(buttonPanel, BorderLayout.SOUTH);
306. }
307.
308. private PrintPreviewCanvas canvas;
309.
310. private static final int DEFAULT_WIDTH = 300;
311. private static final int DEFAULT_HEIGHT = 300;
312. }
313.
314. /**
315. * The canvas for displaying the print preview.
316. */
317. class PrintPreviewCanvas extends JComponent
```

```
318 {
319. /**
320. * Constructs a print preview canvas.
321. * @param b the book to be previewed
322. */
323. public PrintPreviewCanvas(Book b)
324 {
325     book = b;
326     currentPage = 0;
327 }
328
329. public void paintComponent(Graphics g)
330 {
331     Graphics2D g2 = (Graphics2D) g;
332     PageFormat pageFormat = book.getPageFormat(currentPage);
333
334     double xoff; // x offset of page start in window
335     double yoff; // y offset of page start in window
336     double scale; // scale factor to fit page in window
337     double px = pageFormat.getWidth();
338     double py = pageFormat.getHeight();
339     double sx = getWidth() - 1;
340     double sy = getHeight() - 1;
341     if (px / py < sx / sy) // center horizontally
342     {
343         scale = sy / py;
344         xoff = 0.5 * (sx - scale * px);
345         yoff = 0;
346     }
347     else
348     // center vertically
349     {
350         scale = sx / px;
351         xoff = 0;
352         yoff = 0.5 * (sy - scale * py);
353     }
354     g2.translate((float) xoff, (float) yoff);
355     g2.scale((float) scale, (float) scale);
356
357     // draw page outline (ignoring margins)
358     Rectangle2D page = new Rectangle2D.Double(0, 0, px, py);
359     g2.setPaint(Color.white);
360     g2.fill(page);
361     g2.setPaint(Color.black);
362     g2.draw(page);
363
364     Printable printable = book.getPrintable(currentPage);
365     try
366     {
367         printable.print(g2, pageFormat, currentPage);
368     }
369     catch (PrinterException e)
370     {
371         g2.draw(new Line2D.Double(0, 0, px, py));
372         g2.draw(new Line2D.Double(px, 0, 0, py));
373     }
374 }
```

```
375.  
376.     /**  
377.      * Flip the book by the given number of pages.  
378.      * @param by the number of pages to flip by. Negative values flip backwards.  
379.     */  
380.    public void flipPage(int by)  
381.    {  
382.        int nextPage = currentPage + by;  
383.        if (0 <= nextPage && nextPage < book.getNumberOfPages())  
384.        {  
385.            currentPage = nextPage;  
386.            repaint();  
387.        }  
388.    }  
389.  
390.    private Book book;  
391.    private int currentPage;  
392. }
```

#### API java.awt.print.PrinterJob 1.2

- void setPageable(Pageable p)

设置一个要打印的Pageable (比如, 一个Book)。

#### API java.awt.print.Book 1.2

- void append(Printable p, PageFormat format)
- void append(Printable p, PageFormat format, int pageCount)

为该书添加一个章节。如果页数没有被指定, 那么就添加第一页。

- Printable getPrintable(int page)

获取指定页面的可打印特性。

### 7.12.4 打印服务程序

到目前为止, 我们已经介绍了如何打印2D图形。然而, Java SE1.4中的打印API提供了更大的灵活性。该API定义了大量的数据类型, 使你可以找到能够打印这些数据类型的打印服务程序。这些类型有:

- GIF、JPEG或者PNG格式的图像。
- 纯文本、HTML、PostScript或者PDF格式的文档。
- 原始的打印代码数据。
- 实现了Printable、Pageable或RenderableImage的某个类的对象。

数据本身可以存放在一个字节源或字符源中, 比如一个输入流、一个URL或者一个数组中。文档风格 (document flavor) 描述了一个数据源和一个数据类型的组合。DocFlavor类为不同的数据源定义了许多内部类, 每一个内部类都定义了指定风格的常量。例如, 常量

DocFlavor.INPUT\_STREAM.GIF

描述了从输入流中读入一个GIF格式的图像。表7-3中列出了数据源和数据类型的各种组合。

表7-3 打印服务的文档风格

| 数据源               | 数据类型                | MIME类型                               |
|-------------------|---------------------|--------------------------------------|
| INPUT_STREAM      | GIF                 | image/gif                            |
| URL               | JPEG                | image/jpeg                           |
| BYTE_ARRAY        | PNG                 | image/png                            |
|                   | POSTSCRIPT          | application/postscript               |
|                   | PDF                 | application/pdf                      |
|                   | TEXT_HTML_HOST      | text/html (使用主机编码)                   |
|                   | TEXT_HTML_US_ASCII  | text/html; charset=us-ascii          |
|                   | TEXT_HTML_UTF_8     | text/html; charset=utf-8             |
|                   | TEXT_HTML_UTF_16    | text/html; charset=utf-16            |
|                   | TEXT_HTML_UTF_16LE  | text/html; charset=utf-16le (小尾数法)   |
|                   | TEXT_HTML_UTF_16BE  | text/html; charset=utf-16be (大尾数法)   |
|                   | TEXT_PLAIN_HOST     | text/plain (使用主机编码)                  |
|                   | TEXT_PLAIN_US_ASCII | text/plain; charset=us-ascii         |
|                   | TEXT_PLAIN_UTF_8    | text/plain; charset=utf-8            |
|                   | TEXT_PLAIN_UTF_16   | text/plain; charset=utf-16           |
|                   | TEXT_PLAIN_UTF_16LE | text/plain; charset=utf-16le (小尾数法)  |
|                   | TEXT_PLAIN_UTF_16BE | text/plain; charset=utf-16be (大尾数法)  |
|                   | PCL                 | application/vnd.hp-PCL (惠普公司打印机控制语言) |
|                   | AUTONSENSE          | application/octet-stream (原始打印数据)    |
| READER            | TEXT_HTML           | text/html; charset=utf-16            |
| STRING            | TEXT_PLAIN          | text/plain; charset=utf-16           |
| CHAR_ARRAY        |                     |                                      |
| SERVICE_FORMATTED | PRINTABLE           | 无                                    |
|                   | PAGEABLE            | 无                                    |
|                   | RENDERABLE_IMAGE    | 无                                    |

假设想打印一个位于文件中的GIF格式的图像。首先，确认是否有能够处理该打印任务的一个打印服务程序。PrintServiceLookup类的静态lookupPrintServices方法返回一个能够处理给定文档风格的PrintService对象的数组。

```
DocFlavor flavor = DocFlavor.INPUT_STREAM.GIF;
PrintService[] services
    = PrintServiceLookup.lookupPrintServices(flavor, null);
```

lookupPrintServices方法的第二个参数值为null，表示我们不想通过设定打印机属性来限制对文档的搜索。我们在下一节中介绍打印机的属性。

 **注意：**Java SE 6为图像和2D图形等基本文档风格提供了打印服务程序。但是，如果想要打印文本或者HTML文档，打印服务程序的查找将返回一个空数组。

如果对打印服务程序的查找返回的数组带有多个元素的话，要从打印服务程序列表中选择你需要的打印服务程序。通过调用PrintService类的getName方法，可以获得打印机的名称，然后让用户进行选择。

接着，获得该打印服务的文档打印作业：

```
DocPrintJob job = services[i].createPrintJob();
```

如果要执行打印操作，需要一个实现了Doc接口的类的对象。Java为此提供了一个SimpleDoc类。SimpleDoc类的构造器必须包含数据源对象、文档风格和一个可选的属性集。例如，

```
InputStream in = new FileInputStream(fileName);
Doc doc = new SimpleDoc(in, flavor, null);
```

最后，就可以执行打印输出了。

```
job.print(doc, null);
```

与前面一样，null参数可以被一个属性集取代。

请注意，这个打印进程和上一节的打印进程之间有很大的差异。这里不需要用户通过打印对话框来进行交互式的操作。例如，可以实现一个服务器端的打印机制，这样，用户就可以通过Web表单提交打印作业了。

程序清单7-10中的程序展示了如何使用打印服务程序来打印一个图像文件。

### 程序清单7-10 PrintServiceTest.java

```
1. import java.io.*;
2. import javax.print.*;
3.
4. /**
5.  * This program demonstrates the use of print services. The program lets you print a GIF
6.  * image to any of the print services that support the GIF document flavor.
7.  * @version 1.10 2007-08-16
8.  * @author Cay Horstmann
9. */
10. public class PrintServiceTest
11. {
12.     public static void main(String[] args)
13.     {
14.         DocFlavor flavor = DocFlavor.URL.GIF;
15.         PrintService[] services = PrintServiceLookup.lookupPrintServices(flavor, null);
16.         if (args.length == 0)
17.         {
18.             if (services.length == 0) System.out.println("No printer for flavor " + flavor);
19.             else
20.             {
21.                 System.out.println("Specify a file of flavor " + flavor
22.                     + "\nand optionally the number of the desired printer.");
23.                 for (int i = 0; i < services.length; i++)
24.                     System.out.println((i + 1) + ":" + services[i].getName());
25.             }
26.             System.exit(0);
27.         }
28.         String fileName = args[0];
29.         int p = 1;
30.         if (args.length > 1) p = Integer.parseInt(args[1]);
31.         try
32.         {
33.             if (fileName == null) return;
```

```

34.     FileInputStream in = new FileInputStream(fileName);
35.     Doc doc = new SimpleDoc(in, flavor, null);
36.     DocPrintJob job = services[p - 1].createPrintJob();
37.     job.print(doc, null);
38.   }
39.   catch (FileNotFoundException e)
40.   {
41.     e.printStackTrace();
42.   }
43.   catch (PrintException e)
44.   {
45.     e.printStackTrace();
46.   }
47. }
48. }
```

**API** **javax.print.PrintServiceLookup 1.4**

- **PrintService[] lookupPrintServices(DocFlavor flavor, AttributeSet attributes)**

查找能够处理给定文档风格和属性的打印服务程序。

参数: flavor 文档风格

attributes 需要的打印属性,如果不考虑打印属性的话,其值应该为null

**API** **javax.print.PrintService 1.4**

- **DocPrintJob createPrintJob()**

为实现了Doc接口(如SimpleDoc)的一个对象创建一个打印作业。

**API** **javax.print.DocPrintJob 1.4**

- **void print(Doc doc, PrintRequestAttributeSet attributes)**

打印带有给定属性的给定文档。

参数: doc 要打印的Doc

attributes 需要的打印属性,如果不需要任何打印属性的话,其值为null

**API** **javax.print.SimpleDoc 1.4**

- **SimpleDoc(Object data, DocFlavor flavor, DocAttributeSet attributes)**

构建一个能够用DocPrintJob打印的SimpleDoc对象。

参数: data 带有打印数据的对象,比如一个输入流或者一个Printable

flavor 打印数据的文档风格

attributes 文档属性,如果不需要文档属性,其值为null

### 7.12.5 流打印服务程序

打印服务程序将打印数据发送给打印机。流打印服务程序产生同样的打印数据,但是并不把数据发送给打印机,而是发给流。这么做的目的也许是为了延迟打印或者因为打印数据格式

可以由其他程序来进行解释。尤其是，如果打印数据格式是PostScript时，那么可将打印数据保存到一个文件中，因为有许多程序都能够处理PostScript文件。Java平台引入了一个流打印服务程序，它能够从图像和2D图形中产生PostScript输出。可以在任何系统中使用这种服务程序，即使这些系统中没有本地打印机，也可以使用该服务程序。

枚举流打印服务程序要比定位普通的打印服务程序复杂一些。既需要打印对象的DocFlavor又需要流输出的MIME类型。接着获得一个StreamPrintServiceFactory类型的数组，如下所示：

```
DocFlavor flavor = DocFlavor.SERVICE_FORMATTED.PRINTABLE;
String mimeType = "application/postscript";
StreamPrintServiceFactory[] factories
    = StreamPrintServiceFactory.lookupStreamPrintServiceFactories(flavor, mimeType);
```

StreamPrintServiceFactory类没有任何方法能够帮助我们区分不同的factory，所以我们只提取factories[0]。我们调用带有输出流参数的getPrintService方法来获得一个StreamPrintService对象。

```
OutputStream out = new FileOutputStream(fileName);
StreamPrintService service = factories[0].getPrintService(out);
```

StreamPrintService类是PrintService的子类。如果要产生一个打印输出，只要按照上一节介绍的步骤进行操作即可。

#### **javax.print.StreamPrintServiceFactory 1.4**

- StreamPrintServiceFactory[] lookupStreamPrintServiceFactories(DocFlavor flavor, String mimeType)

查找所需的流打印服务程序工厂，它能够打印给定文档风格，并且产生一个给定MIME类型的输出流。

- StreamPrintService getPrintService(OutputStream out)

获得一个打印服务程序，以便将打印输出发送给指定的输出流中。

#### 7.12.6 打印属性

打印服务程序API包含了一组复杂的接口和类，用以设定不同种类的属性。重要的属性共有四组，前两组属性用于设定对打印机的访问请求。

- 打印请求属性（Print request attributes）为一个打印作业中的所有doc对象请求特定的打印属性，例如，双面打印或者纸张的大小。

- Doc属性（Doc attributes）是仅作用在单个doc对象上的请求属性。

另外两组属性包含关于打印机和作业状态的信息。

- 打印服务属性（Print service attributes）提供了关于打印服务程序的信息，比如打印机的种类和型号，或者打印机当前是否接受打印作业。

- 打印作业属性（Print job attributes）提供了关于某个特定打印作业状态的信息，比如该打印作业是否已经完成。

如果要描述各种不同的打印属性，可以使用带有如下子接口的Attribute接口。

```

PrintRequestAttribute
DocAttribute
PrintServiceAttribute
PrintJobAttribute
SupportedValuesAttribute

```

各个属性类都实现了上面的一个或几个接口。例如，Copies类的对象描述了一个打印输出的拷贝数量。该类就实现了PrintRequestAttribute和PrintJobAttribute两个接口。显然，某个打印请求可能包含一个对多个拷贝的请求。相反，打印作业的某个属性可能表示的是实际上打印出来的拷贝数量。这个拷贝数量可能很小，也许是因为打印机的限制或者是因为打印机的纸张已经用完了。

SupportedValuesAttribute接口表示某个属性值反映的不是实际的打印请求或状态数据，而是某个服务程序的能力。例如，实现了SupportedValuesAttribute接口的CopiesSupported类，该类的对象可以用来描述某个打印机能够支持1~99份的打印输出。

图7-38显示了属性分层结构的类图。

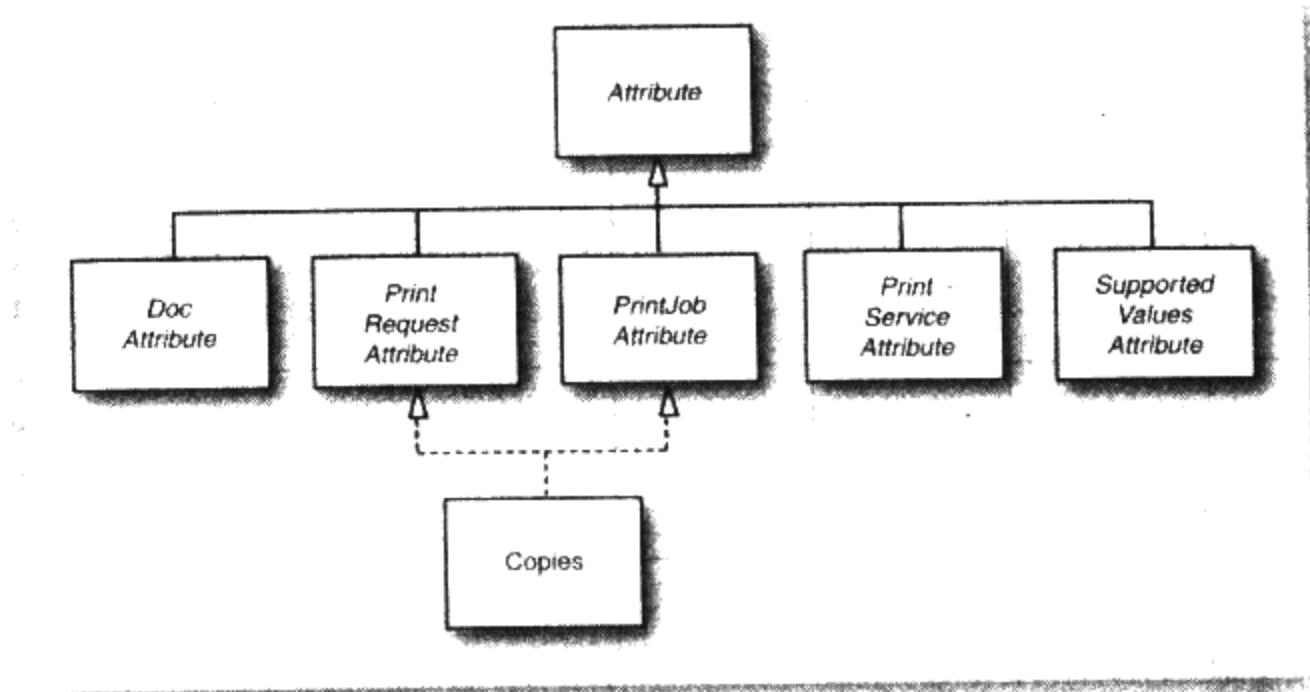


图7-38 显示了一个属性层次结构的类图

除了为各个属性定义的接口和类以外，打印服务程序API还为属性集定义了接口和类。父接口AttributeSet有四个子接口：

```

PrintRequestAttributeSet
DocAttributeSet
PrintServiceAttributeSet
PrintJobAttributeSet

```

对于每个这样的接口，都有一个实现类，用于产生下面5个类：

```

HashAttributeSet
HashPrintRequestAttributeSet
HashDocAttributeSet
HashPrintServiceAttributeSet
HashPrintJobAttributeSet

```

图7-39显示了属性集分层结构的类图。

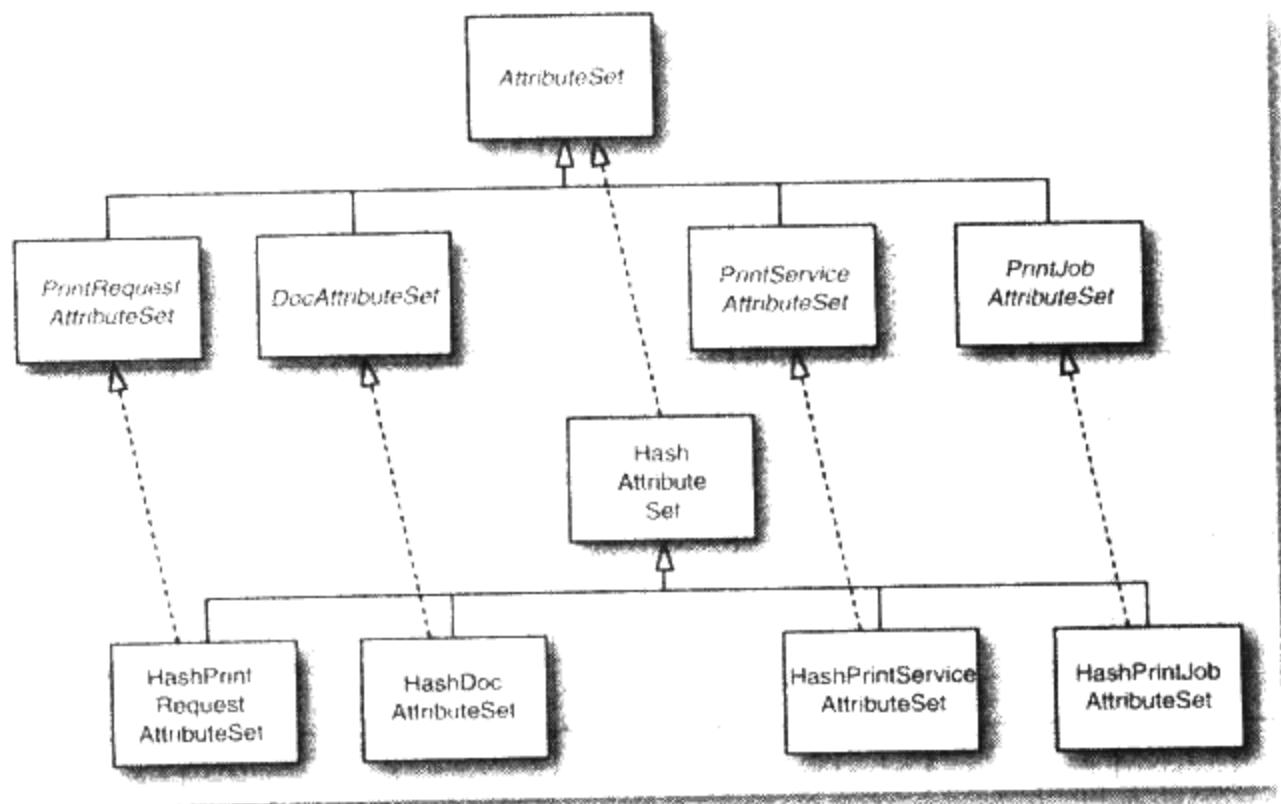


图7-39 属性集的分层结构

例如，可以用如下的方式构建一个打印请求属性集。

```
PrintRequestAttributeSet attributes = new HashPrintRequestAttributeSet();
```

当构建完属性集后，就不用担心使用Hash前缀的问题了。

为什么要配所有这些接口呢？因为，它们使得“检查属性是否被正确使用”成为了可能。例如，Doc.AttributeSet只接受实现了DocAttribute接口的对象。添加另一个属性的任何尝试都会导致运行期错误的产生。

属性集是一个特殊的映射表，其键是Class类型的，而值是一个实现了Attribute接口的类。例如，如果要插入一个对象

```
new Copies(10)
```

到属性集中，那么它的键就是Class对象Copies.class。该键被称为属性的类别。Attribute接口声明了下面这样一个方法：

```
Class getCategory()
```

该方法返回了一个属性的类别。Copies类定义了用以返回Copies.class对象的方法。但是，属性的类别和属性的类没有必要是相同的。

当将一个属性添加到属性集中时，属性的类别就会被自动地获取。你只需添加该属性的值：

```
attributes.add(new Copies(10));
```

如果后来添加了一个具有相同类别的另一个属性，那么新属性就会覆盖第一个属性。如果要检索一个属性，需要使用它的类别作为键，例如，

```
AttributeSet attributes = job.getAttributes();
Copies copies = (Copies) attribute.get(Copies.class);
```

最后，属性是按照它们拥有的值来进行组织的。`Copies`属性能够拥有任何整数值。`Copies`类继承了`IntegerSyntax`类，该类负责处理所有带有整数值的属性。`getValue`方法将返回属性的整数值，例如，

```
int n = copies.getValue();
```

下面这些类：

```
TextSyntax  
DateTimeSyntax  
URISyntax
```

用于封装一个字符串、日期与时间，或者URI（通用资源标识符）。

最后要说明的是，许多属性都能够接受数量有限的值。例如，`PrintQuality`属性有三个设置值：`draft`（草稿质量），`normal`（正常质量）和`high`（高质量），它们用三个常量来表示：

```
PrintQuality.DRAFT  
PrintQuality.NORMAL  
PrintQuality.HIGH
```

拥有有限数量值的属性类继承了`EnumSyntax`类，该类提供了许多便利的方法，用来以一种类型安全的方式设置这些枚举操作。当使用这样的属性时，不必担心该机制，只需要将带有名字的值添加给属性集即可：

```
attributes.add(PrintQuality.HIGH);
```

下面的代码说明了如何来检查一个属性的值：

```
if (attributes.get(PrintQuality.class) == PrintQuality.HIGH)
```

表7-4列出了各个打印属性。表中的第二列列出了属性类的超类（例如，`Copies`属性的`IntegerSyntax`类）或者是具有一组有限值属性的一组枚举值。最后四列表示该属性是否实现了`DocAttribute` (DA)、`PrintJobAttribute` (PJA)、`PrintRequestAttribute` (PRA)和`PrintServiceAttribute` (PSA)几个接口。

表7-4 打印属性一览表

| 属性                   | 超类或枚举常量  | DA | PJA | PRA | PSA |
|----------------------|--|----|-----|-----|-----|
| Chromaticity         | MONOCHROME, COLOR  | ✓  | ✓   | ✓   |     |
| ColorSupported       | SUPPORTED, NOT_SUPPORTED                                     |    |     |     | ✓   |
| Compression          | COMPRESS, DEFLATE, GZIP, NONE                                | ✓  |     |     |     |
| Copies               | IntegerSyntax  |    | ✓   | ✓   |     |
| DateTimeAtCompleted  | DateTimeSyntax   |    | ✓   |     |     |
| DateTimeAtCreation   | DateTimeSyntax   |    | ✓   |     |     |
| DateTimeAtProcessing | DateTimeSyntax   |    | ✓   |     |     |
| Destination          | URISyntax  |    | ✓   | ✓   |     |
| DocumentName         | TextSyntax   | ✓  |     |     |     |
| Fidelity             | FIDELITY_TRUE, FIDELITY_FALSE                                |    | ✓   | ✓   |     |
| Finishings           | NONE, STAPLE, EDGE_STITCH, BIND, SADDLE_STITCH, COVER, . . . | ✓  | ✓   | ✓   |     |
| JobHoldUntil         | DateTimeSyntax   |    | ✓   | ✓   |     |

(续)

| 属性                       | 超类或枚举常量   | DA | PJA | PRA | PSA |
|--------------------------|---|----|-----|-----|-----|
| JobImpressions           | IntegerSyntax   | ✓  | ✓   |     |     |
| JobImpressionsCompleted  | IntegerSyntax   | ✓  |     |     |     |
| JobKOctets               | IntegerSyntax   | ✓  | ✓   |     |     |
| JobKOctetsProcessed      | IntegerSyntax   | ✓  |     |     |     |
| JobMediaSheets           | IntegerSyntax   | ✓  | ✓   |     |     |
| JobMediaSheetsCompleted  | IntegerSyntax   | ✓  |     |     |     |
| JobMessageFromOperator   | TextSyntax  | ✓  |     |     |     |
| JobName                  | TextSyntax  | ✓  | ✓   |     |     |
| JobOriginatingUserName   | TextSyntax  | ✓  |     |     |     |
| JobPriority              | IntegerSyntax   | ✓  | ✓   |     |     |
| JobSheets                | STANDARD, NONE  | ✓  | ✓   |     |     |
| JobState                 | ABORTED, CANCELED, COMPLETED, PENDING,<br>PENDING_HELD, PROCESSING, PROCESSING_STOPPED  | ✓  |     |     |     |
| JobStateReason           | ABORTED_BY_SYSTEM, DOCUMENT_FORMAT_<br>ERROR, 其他  |    |     |     |     |
| JobStateReasons          | HashSet   | ✓  |     |     |     |
| MediaName                | ISO_A4_WHITE, ISO_A4_TRANSPARENT,<br>NA LETTER WHITE, NA LETTER TRANSPARENT   | ✓  | ✓   | ✓   |     |
| MediaSize                | ISO_A0 - ISO_A10, ISO_B0 - ISO_B10, ISO_C0 -<br>ISO_C10, NA LETTER, NA LEGAL, 各种其他纸张和信封<br>尺寸   |    |     |     |     |
| MediaSizeName            | ISO_A0 - ISO_A10, ISO_B0 - ISO_B10, ISO_C0 -<br>ISO_C10, NA LETTER, NA LEGAL, 各种其他纸张和信封<br>尺寸   | ✓  | ✓   | ✓   |     |
| MediaTray                | TOP, MIDDLE, BOTTOM, SIDE, ENVELOPE,<br>LARGE_CAPACITY, MAIN, MANUAL,   | ✓  | ✓   | ✓   |     |
| MultipleDocumentHandling | SINGLE_DOCUMENT, SINGLE_DOCUMENT_NEW_SHEET,<br>SEPARATE_DOCUMENTS_COLLATED_COPIES,<br>SEPARATE_DOCUMENTS_UNCOLLATED_COPIES            | ✓  | ✓   |     |     |
| NumberOfDocuments        | IntegerSyntax   | ✓  |     |     |     |
| NumberOfInterveningJobs  | IntegerSyntax   | ✓  |     |     |     |
| NumberUp                 | IntegerSyntax   | ✓  | ✓   | ✓   |     |
| OrientationRequested     | PORTRAIT, LANDSCAPE, REVERSE_PORTRAIT,<br>REVERSE_LANDSCAPE   | ✓  | ✓   | ✓   |     |
| OutputDeviceAssigned     | TextSyntax  | ✓  |     |     |     |
| PageRanges               | SetOfInteger  | ✓  | ✓   | ✓   |     |
| PagesPerMinute           | IntegerSyntax   |    |     |     | ✓   |
| PagesPerMinuteColor      | IntegerSyntax   |    |     |     | ✓   |
| PDLOverrideSupported     | ATTEMPTED, NOT_ATTEMPTED  |    |     |     | ✓   |
| PresentationDirection    | TORIGHT_TOBOTTOM, TORIGHT_TOTOP, TOBOTTOM_<br>TORIGHT, TOBOTTOM_TOLEFT, TOLEFT_TOBOTTOM,<br>TOLEFT_TOTOP, TOTOP_TORIGHT, TOTOP_TOLEFT | ✓  | ✓   |     |     |
| PrinterInfo              | TextSyntax  |    |     |     | ✓   |

(续)

| 属性                           | 超类或枚举常量   | DA | PJA | PRA | PSA |
|------------------------------|---|----|-----|-----|-----|
| PrinterIsAcceptingJobs       | ACCEPTING_JOBS, NOT_ACCEPTING_JOBS  |    |     |     | ✓   |
| PrinterLocation              | TextSyntax  |    |     |     | ✓   |
| PrinterMakeAndModel          | TextSyntax  |    |     |     | ✓   |
| PrinterMessageFromOperator   | TextSyntax  |    |     |     | ✓   |
| PrinterMoreInfo              | URISyntax   |    |     |     | ✓   |
| PrinterMoreInfoManufacturer  | URISyntax   |    |     |     | ✓   |
| PrinterName                  | TextSyntax  |    |     |     | ✓   |
| PrinterResolution            | ResolutionSyntax  | ✓  | ✓   | ✓   |     |
| PrinterState                 | PROCESSING, IDLE, STOPPED, UNKNOWN  |    |     |     | ✓   |
| PrinterStateReason           | COVER_OPEN, FUSER_OVER_TEMP, MEDIA_JAM,<br>其他                               |    |     |     |     |
| PrinterStateReasons          | HashMap   |    |     |     |     |
| PrinterURI                   | URISyntax   |    |     |     | ✓   |
| PrintQuality                 | DRAFT, NORMAL, HIGH   | ✓  | ✓   | ✓   |     |
| QueuedJobCount               | IntegerSyntax   |    |     |     | ✓   |
| ReferenceUriSchemesSupported | FILE, FTP, GOPHER, HTTP, HTTPS, NEWS, NNTP, WAIS                            |    |     |     |     |
| RequestingUserName           | TextSyntax  |    |     |     | ✓   |
| Severity                     | ERROR, REPORT, WARNING  |    |     |     |     |
| SheetCollate                 | COLLATED, UNCOLLATED  | ✓  | ✓   | ✓   |     |
| Sides                        | ONE_SIDED, DUPLEX (=TWO_SIDED_LONG_EDGE),<br>TUMBLE (=TWO_SIDED_SHORT_EDGE) | ✓  | ✓   | ✓   |     |

注意：可以看到，属性的数量很多，其中许多属性都是专用的。大多数属性都来源于因特网打印协议1.1版（RFC 2911）。

注意：打印API的早期版本引入了JobAttributes和PageAttributes类，其目的与本节所介绍的打印属性类似。这些类现在已经弃用了。

#### **javax.print.attribute.Attribute 1.4**

- `Class getCategory()`

获取该属性的类别。

- `String getName()`

获取该属性的名字。

#### **javax.print.attribute.AttributeSet 1.4**

- `boolean add(Attribute attr)`

向属性集中添加一个属性。如果有另一个属性和此属性有相同的类别，集中的属性被新添加的属性所取代。如果由于添加属性的操作改变了属性集，则返回true。

- `Attribute get(Class category)`

检索带有指定属性类别键的属性，如果属性不存在，则返回null。

- `boolean remove(Attribute attr)`
- `boolean remove(Class category)`

从属性集中删除给定属性，或者删除具有指定类别的属性。如果由于这个操作改变了属性集，则返回true。

- `Attribute[] toArray()`

返回一个带有该属性集中所有属性的数组。

#### **javax.print.PrintService 1.4**

- `PrintServiceAttributeSet getAttributes()`

获取打印服务程序的属性。

#### **javax.print.DocPrintJob 1.4**

- `PrintJobAttributeSet getAttributes()`

获取打印作业的属性。

我们总结了关于打印的讨论。现在，读者已经知道应该如何打印2D图形和其他文档类型，怎样枚举各种打印机和流打印服务程序，以及如何设置和获取打印属性。接下来，我们将介绍两个重要的用户接口：剪贴板和支持拖放的机制。

## 7.13 剪贴板

在图形用户界面环境（比如Windows 和 X Window系统）中，剪切和拷贝是最有用和最方便的用户接口机制之一。你可以在某个程序中选择一些数据，将它们剪切或者拷贝到剪贴板上。然后选择另一个程序，将剪切板中的内容粘贴到该应用中去。使用剪贴板，可以把文本、图像或者别的数据从一个文档移动到另一个文档中，当然也可以从文档的一个地方移动到该文档的另一个地方。剪切和粘贴操作是如此地普通，以至于大多数计算机的用户从来都没有考虑过它究竟是如何实现的。

尽管剪贴板从概念上来说是非常简单的，但是实现剪贴板服务却比想像中的要困难得多。假设你将文本从字处理程序拷贝到了剪贴板，如果你要将该文本粘贴到另一个字处理程序中，那么肯定希望该文本的字体和格式保持原样。也就是说，剪贴板中的文本必须保留原来的格式信息。但是如果要将该文本信息粘贴到一个纯文本域中，那么你希望只粘贴文本字符，而不包括附加的格式化的代码。为了支持这种灵活性，数据提供者可以提供多种格式的剪贴板数据，而数据使用者可以从多种格式中选择所需要的格式。

微软公司的Windows和苹果公司的Macintosh等操作系统的剪贴板实现方法是类似的。当然，它们之间会有略微的不同。然而，X Window系统的剪贴板机制的功能是非常有限的。它只支持纯文本的剪切和粘贴。当你试图运行本节中的程序时，应该考虑它的这些局限性。

 **注意：**请查看你的平台上的`jre/lib/flavormap.properties`文件，以便得知关于哪些类型的对象能够在Java程序和系统剪贴板之间进行传递。

通常，程序应该支持对系统剪贴板不能处理的那些数据类型的剪切和粘贴。数据传递API支持对同一个虚拟机中任何本地对象引用的传递。在不同的虚拟机之间，可以将序列化对象和对象的引用传递给远程对象。

表7-5汇总了剪贴板机制的数据传递的功能。

表7-5 Java数据传递机制的能力

| 传递方式                 | 传递的信息格式               |
|----------------------|-----------------------|
| 在一个Java程序和一个本地程序之间传递 | 文本、图像、文件列表……（依赖于本机平台） |
| 在两个协同操作的Java程序之间传递   | 序列化和远程对象              |
| 在一个Java程序的内部传递       | 任意对象                  |

### 7.13.1 数据传递的类和接口

在Java技术中，`java.awt.datatransfer`包实现了数据传递的功能。下面就是该包中最重要的类和接口的概述。

- 能够通过剪贴板来传递数据的对象必须实现`Transferable`接口。
- `Clipboard`类描述了一个剪贴板。可传递的对象是能够置于剪贴板之上或者从剪贴板上取走的惟一项。系统剪贴板是`Clipboard`类的一个具体实例。
- `DataFlavor`类用来描述存放到剪贴板中的数据风格。
- `StringSelection`类是一个实现了`Transferable`接口的实体类。它用于传递文本字符串。
- 当剪贴板的内容被别人改写时，如果一个类想得到这个情况的通知，那么就必须实现`ClipboardOwner`接口。剪贴板的所有权实现了复杂数据的“延迟格式化”。如果一个程序传递的是一个简单数据（比如一个字符串），那么它只需要设置剪贴板的内容，然后就可以继续进行接下来的操作了。但是，如果一个程序想将能够用多种风格来格式化的复杂数据放到剪贴板上，那么它实际上并不需要为此准备所有的风格，因为大多数的风格从来不会被用到的。不过，这时必须保存剪贴板中的数据，这样就能在以后被请求的时候，建立所需的风格。当剪贴板的内容被更改时，剪贴板的所有者必须得到通知（通过调用`lostOwnership`方法）。这样可以告诉它，这些信息已经不再需要了。在我们的示例程序中，并不用担心剪贴板的所有权问题。

### 7.13.2 传递文本

如果要了解数据传递类，最好的方法就是从最简单的情况开始：即在系统剪贴板上传递和获取文本信息。首先，获取一个系统剪贴板的引用：

```
Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
```

传递给剪贴板的字符串，必须被包装在`StringSelection`对象中。

```
String text = ...  
StringSelection selection = new StringSelection(text);
```

实际的传递操作是通过调用`setContents`方法来实现的，该方法将一个`StringSelection`对象和一个`ClipBoardOwner`作为参数。如果不指定一个剪贴板所有者话，可以把第二个参数

设置为null。

```
clipboard.setContents(selection, null);
```

下面是反过来的操作：从剪贴板中读取一个字符串：

```
DataFlavor flavor = DataFlavor.stringFlavor;  
if (clipboard.isDataFlavorAvailable(flavor))  
    String text = (String) clipboard.getData(flavor);
```

getContents方法的参数是一个请求访问对象的Object引用，但是因为Clipboard类的当前实现会忽略它，所以我们只是传入了null。

getContents方法的返回值可以是null。这表示剪贴板中可能是空的，或者它不存在任何Java平台知道应该如何检索的数据。

程序清单7-11介绍了如何在一个java应用和系统剪贴板之间进行剪切和粘贴操作。如果你选择文本区域中的一块文本区域，并且点击Copy，那么选中的文本就会被拷贝到系统剪贴板中，然后可以将其粘贴到任何文本编辑器中（参见图7-40）。反之，当从文本编辑器中拷贝文本时，也可以将其粘贴到我们的程序清单中。

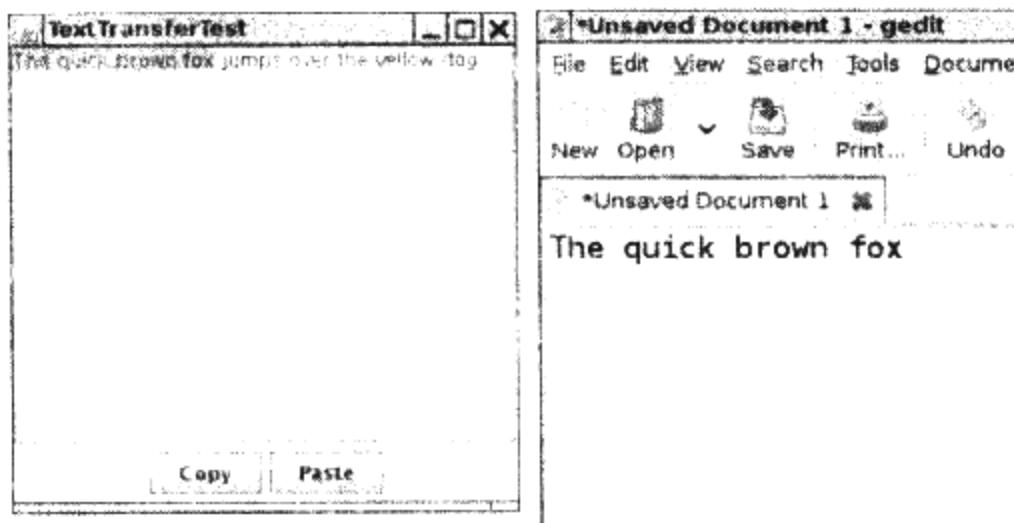


图7-40 TextTransferTest程序的运行情况

### 程序清单7-11 TextTransferTest.java

```
1. import java.awt.*;  
2. import java.awt.datatransfer.*;  
3. import java.awt.event.*;  
4. import java.io.*;  
5.  
6. import javax.swing.*;  
7.  
8. /**  
9. * This program demonstrates the transfer of text between a Java application and the system  
10 * clipboard.  
11 * @version 1.13 2007-08-16  
12 * @author Cay Horstmann  
13 */  
14. public class TextTransferTest  
15. {  
16.     public static void main(String[] args)  
17.     {
```

```
18.     EventQueue.invokeLater(new Runnable()
19.     {
20.         public void run()
21.         {
22.             JFrame frame = new TextTransferFrame();
23.             frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24.             frame.setVisible(true);
25.         }
26.     });
27. }
28. }
29.
30 /**
31 * This frame has a text area and buttons for copying and pasting text.
32 */
33 class TextTransferFrame extends JFrame
34 {
35     public TextTransferFrame()
36     {
37         setTitle("TextTransferTest");
38         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
39.
40         textArea = new JTextArea();
41         add(new JScrollPane(textArea), BorderLayout.CENTER);
42         JPanel panel = new JPanel();
43.
44         JButton copyButton = new JButton("Copy");
45         panel.add(copyButton);
46         copyButton.addActionListener(new ActionListener()
47         {
48             public void actionPerformed(ActionEvent event)
49             {
50                 copy();
51             }
52         });
53.
54         JButton pasteButton = new JButton("Paste");
55         panel.add(pasteButton);
56         pasteButton.addActionListener(new ActionListener()
57         {
58             public void actionPerformed(ActionEvent event)
59             {
60                 paste();
61             }
62         });
63.
64         add(panel, BorderLayout.SOUTH);
65     }
66.
67 /**
68 * Copies the selected text to the system clipboard.
69 */
70 private void copy()
71 {
72     Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
73     String text = textArea.getSelectedText();
```

```
74.     if (text == null) text = textArea.getText();
75.     StringSelection selection = new StringSelection(text);
76.     clipboard.setContents(selection, null);
77. }
78.
79. /**
80. * Pastes the text from the system clipboard into the text area.
81. */
82. private void paste()
83. {
84.     Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
85.     DataFlavor flavor = DataFlavor.stringFlavor;
86.     if (clipboard.isDataFlavorAvailable(flavor))
87.     {
88.         try
89.         {
90.             String text = (String) clipboard.getData(flavor);
91.             textArea.replaceSelection(text);
92.         }
93.         catch (UnsupportedFlavorException e)
94.         {
95.             JOptionPane.showMessageDialog(this, e);
96.         }
97.         catch (IOException e)
98.         {
99.             JOptionPane.showMessageDialog(this, e);
100.        }
101.    }
102. }
103.
104. private JTextArea textArea;
105.
106. private static final int DEFAULT_WIDTH = 300;
107. private static final int DEFAULT_HEIGHT = 300;
108. }
```



### java.awt.Toolkit 1.0

- Clipboard getSystemClipboard() 1.1

获取系统剪贴板。



### java.awt.datatransfer.Clipboard 1.1

- Transferable getContents(Object requester)

获取剪贴板中的内容。

参数: requester 请求剪贴板内容的对象; 该值实际上并不使用。

- void setContents(Transferable contents, ClipboardOwner owner)

将内容放入剪贴板中。

参数: contents 封装了内容的Transferable

owner 当新的信息被放入剪贴板上时, 要通知的对象 (通过调用lostOwnership方法)。如果不需要通知, 则值为null。

- `boolean isDataFlavorAvailable(DataFlavor flavor)` 5.0

如果该剪贴板中有给定风格的数据，那么返回true。

- `Object getData(DataFlavor flavor)` 5.0

获取给定风格的数据，如果给定风格的数据不存在，抛出`UnsupportedFlavorException`异常。

#### **API** `java.awt.datatransfer.ClipboardOwner` 1.1

- `void lostOwnership(Clipboard clipboard, Transferable contents)`

通知该对象，它已经不再是该剪贴板内容的所有者。

参数: `clipboard` 已放置内容的剪贴板

`contents` 该所有者放入剪贴板上的内容

#### **API** `java.awt.datatransfer.Transferable` 1.1

- `boolean isDataFlavorSupported(DataFlavor flavor)`

如果给定的风格是所支持的数据风格中的一种，则返回true，否则返回false。

- `Object getTransferData(DataFlavor flavor)`

返回用所请求风格格式化的数据。如果不支持所请求的风格，则抛出一个`UnsupportedFlavorException`异常。

### 7.13.3 可传递的接口和数据风格

`DataFlavor`是由下面两个特性来定义的：

- MIME类型的名字（比如“image/gif”）。
- 用于访问数据的表示类（比如`java.awt.Image`）。

此外，每一个数据风格都有一个适合人们阅读的名字（比如“GIF Image”）。

表示类可以用MIME类型的`class`参数设定，例如，

`image/gif;class=java.awt.Image`

 **注意：**这只是一个说明其语法的例子。对于传递GIF图像数据，并没有一个标准的数据风格。

如果没有给定任何`class`参数，那么表示类就是`InputStream`。

为了传递本地的、序列化的和远程的Java对象，Sun公司定义了如下三个MIME类型：

`application/x-java-jvm-local-objectref`  
`application/x-java-serialized-object`  
`application/x-java-remote-object`

 **注意：**x-前缀表示这是一个试用名，并不是IANA批准的名字，IANA是负责分配标准的MIME类型名的机构。

例如，标准的`stringFlavor`数据风格是由下面这个MIME类型描述的。

`application/x-java-serialized-object;class=java.lang.String`

可以让剪贴板列出所有可用的风格：

```
DataFlavor[] flavors = clipboard.getAvailableDataFlavors()
```

也可以在剪贴板上安装一个FlavorListener，当剪贴板上的数据风格集合产生变化时，通知风格监听器。细节请参阅API注释。

#### **API** `java.awt.datatransfer.DataFlavor 1.1`

- `DataFlavor(String mimeType, String humanPresentableName)`

创建一个数据风格，它描述了一个流数据，该流数据的格式是由一个MIME类型所描述的。

参数：mimeType 一个MIME类型字符串

humanPresentableName 一个更易于阅读的名字版本

- `DataFlavor(Class class, String humanPresentableName)`

创建一个用来描述Java平台类的数据风格。它的MIME类型是application/x-java-serialized-object;class=className。

参数：class 从Transferable检索到的类

humanPresentableName 一个可读的名字版本

- `String getMimeType()`

返回该数据风格的MIME类型字符串。

- `boolean isMimeTypeEqual(String mimeType)`

测试该数据风格是否有给定的MIME类型。

- `String getHumanPresentableName()`

为该数据风格的数据格式返回人们容易读取的名字。

- `Class getRepresentationClass()`

返回一个Class对象，它代表用该数据风格调用Transferable时返回的对象的类。它可以是MIME类型的class参数，也可以是InputStream。

#### **API** `java.awt.datatransfer.Clipboard 1.1`

- `DataFlavor[] getAvailableDataFlavors() 5.0`

返回一个可用风格的数组。

- `void addFlavorListener(FlavorListener listener) 5.0`

添加一个监听器，当可用的风格发生改变时，通知该监听器。

#### **API** `java.awt.datatransfer.Transferable 1.1`

- `DataFlavor[] getTransferDataFlavors()`

返回一个所支持风格的数组。

#### **API** `java.awt.datatransfer.FlavorListener 5.0`

- `void flavorsChanged(FlavorEvent event)`

当一个剪贴板中可用的风格发生变化时，就调用该方法。

#### 7.13.4 构建一个可传递的图像

想要通过剪贴板传递对象就必须实现Transferable接口。StringSelection类是目前Java标准库中惟一一个实现了Transferable接口的公有类。在这一节中，我们将介绍如何通过剪贴板来传递图像。因为Java并没有提供一个传递图像的类，所以读者必须自己去实现它。

这个类其实只是一个非常普通的类。它只需告知惟一可用的数据格式是DataFlavor.imageFlavor，并且它持有一个image对象即可。

```
class ImageTransferable implements Transferable
{
    public ImageTransferable(Image image)
    {
        theImage = image;
    }

    public DataFlavor[] getTransferDataFlavors()
    {
        return new DataFlavor[] { DataFlavor.imageFlavor };
    }

    public boolean isDataFlavorSupported(DataFlavor flavor)
    {
        return flavor.equals(DataFlavor.imageFlavor);
    }

    public Object getTransferData(DataFlavor flavor)
        throws UnsupportedFlavorException
    {
        if(flavor.equals(DataFlavor.imageFlavor))
        {
            return theImage;
        }
        else
        {
            throw new UnsupportedFlavorException(flavor);
        }
    }

    private Image theImage;
}
```

 注意：Java SE提供了DataFlavor.imageFlavor常量，并且负责执行所有复杂的操作，以便进行Java图像与本机剪贴板图像的转换。但是，奇怪的是，它并没有提供将图像放入剪贴板时所需要的包装类。

程序清单7-12中的程序展示了如何在Java应用程序和系统剪贴板之间进行图像传递。当程序开始运行时，它产生了一个包含红圈的图像。点击Copy按钮把图像拷贝到剪贴板上，然后将它粘贴到另一个应用中。（参见图7-41）之后再从另一个应用拷贝一个图像到系统剪贴板中，然后点击Paste按钮，就可以看到该图像被粘贴到程序清单中了（参见图7-42）。

该程序是直接在文本传递程序基础上进行修改而得到的。现在的数据风格是

DataFlavor.imageFlavor，并且我们使用的是ImageSelection类来将图像传递给系统剪贴板。

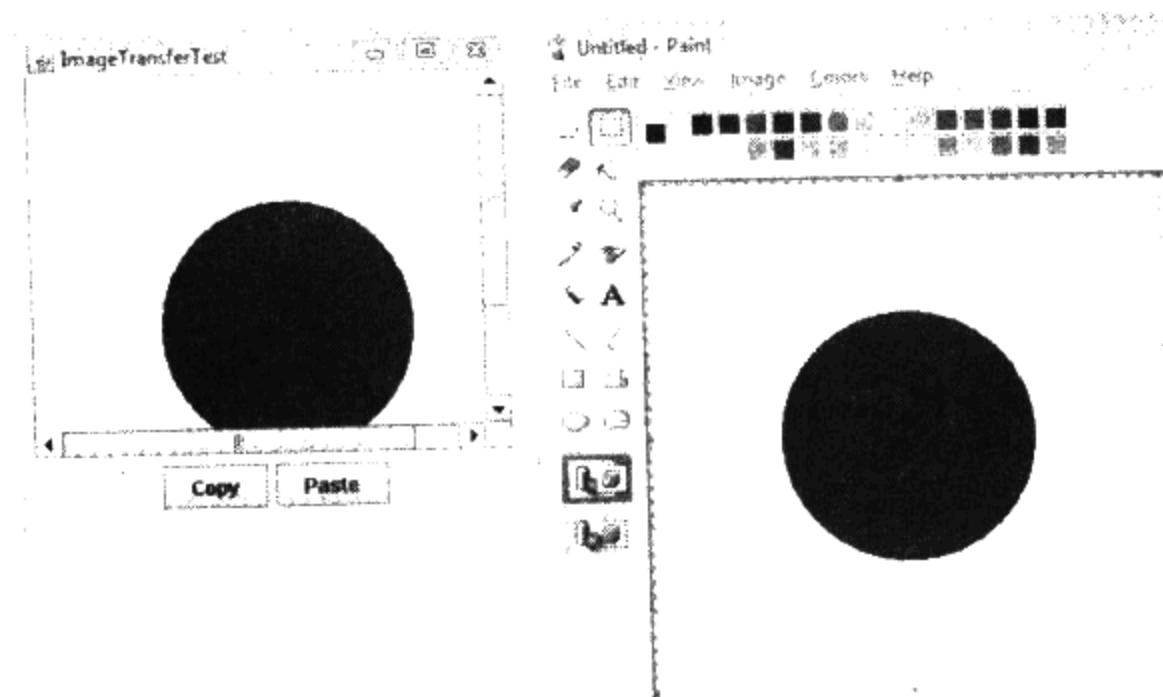


图7-41 将图像从一个Java程序拷贝到本机程序中

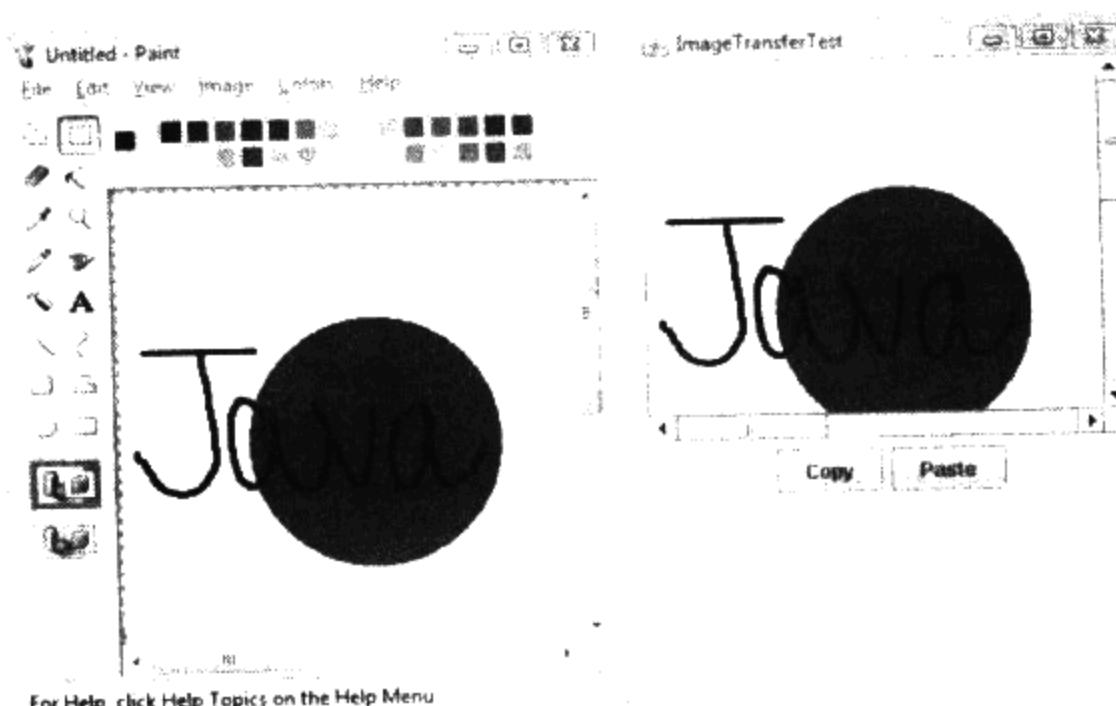


图7-42 将图像从本机程序拷贝到一个Java程序中

### 程序清单7-12 ImageTransferTest.java

```
1. import java.io.*;
2. import java.awt.*;
3. import java.awt.datatransfer.*;
4. import java.awt.event.*;
5. import java.awt.image.*;
6. import javax.swing.*;
7.
8. /**
9. * This program demonstrates the transfer of images between a Java application and the system
10. * clipboard.

```

```
11. * @version 1.22 2007-08-16
12. * @author Cay Horstmann
13. */
14. public class ImageTransferTest
15. {
16.     public static void main(String[] args)
17.     {
18.         EventQueue.invokeLater(new Runnable()
19.         {
20.             public void run()
21.             {
22.                 JFrame frame = new ImageTransferFrame();
23.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24.                 frame.setVisible(true);
25.             }
26.         });
27.     }
28. }
29.
30. /**
31. * This frame has an image label and buttons for copying and pasting an image.
32. */
33. class ImageTransferFrame extends JFrame
34. {
35.     public ImageTransferFrame()
36.     {
37.         setTitle("ImageTransferTest");
38.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
39.
40.         label = new JLabel();
41.         image = new BufferedImage(DEFAULT_WIDTH, DEFAULT_HEIGHT, BufferedImage.TYPE_INT_ARGB);
42.         Graphics g = image.getGraphics();
43.         g.setColor(Color.WHITE);
44.         g.fillRect(0, 0, DEFAULT_WIDTH, DEFAULT_HEIGHT);
45.         g.setColor(Color.RED);
46.         g.fillOval(DEFAULT_WIDTH / 4, DEFAULT_WIDTH / 4, DEFAULT_WIDTH / 2, DEFAULT_HEIGHT / 2);
47.
48.         label.setIcon(new ImageIcon(image));
49.         add(new JScrollPane(label), BorderLayout.CENTER);
50.         JPanel panel = new JPanel();
51.
52.         JButton copyButton = new JButton("Copy");
53.         panel.add(copyButton);
54.         copyButton.addActionListener(new ActionListener()
55.         {
56.             public void actionPerformed(ActionEvent event)
57.             {
58.                 copy();
59.             }
60.         });
61.
62.         JButton pasteButton = new JButton("Paste");
63.         panel.add(pasteButton);
64.         pasteButton.addActionListener(new ActionListener()
65.         {
66.             public void actionPerformed(ActionEvent event)
67.             {
```

```
68.         paste();
69.     }
70. };
71.
72.     add(panel, BorderLayout.SOUTH);
73. }
74.
75. /**
76. * Copies the current image to the system clipboard.
77. */
78. private void copy()
79. {
80.     Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
81.     ImageTransferable selection = new ImageTransferable(image);
82.     clipboard.setContents(selection, null);
83. }
84.
85. /**
86. * Pastes the image from the system clipboard into the image label.
87. */
88. private void paste()
89. {
90.     Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
91.     DataFlavor flavor = DataFlavor.imageFlavor;
92.     if (clipboard.isDataFlavorAvailable(flavor))
93.     {
94.         try
95.         {
96.             image = (Image) clipboard.getData(flavor);
97.             label.setIcon(new ImageIcon(image));
98.         }
99.         catch (UnsupportedFlavorException exception)
100.        {
101.            JOptionPane.showMessageDialog(this, exception);
102.        }
103.        catch (IOException exception)
104.        {
105.            JOptionPane.showMessageDialog(this, exception);
106.        }
107.    }
108. }
109.
110. private JLabel label;
111. private Image image;
112.
113. private static final int DEFAULT_WIDTH = 300;
114. private static final int DEFAULT_HEIGHT = 300;
115. }
116.
117. /**
118. * This class is a wrapper for the data transfer of image objects.
119. */
120. class ImageTransferable implements Transferable
121. {
122.     /**
123.      * Constructs the selection.
124.      * @param image an image
```

```
125.  */
126. public ImageTransferable(Image image)
127. {
128.     theImage = image;
129. }
130.
131. public DataFlavor[] getTransferDataFlavors()
132. {
133.     return new DataFlavor[] { DataFlavor.imageFlavor };
134. }
135.
136. public boolean isDataFlavorSupported(DataFlavor flavor)
137. {
138.     return flavor.equals(DataFlavor.imageFlavor);
139. }
140.
141. public Object getTransferData(DataFlavor flavor) throws UnsupportedFlavorException
142. {
143.     if (flavor.equals(DataFlavor.imageFlavor))
144.     {
145.         return theImage;
146.     }
147.     else
148.     {
149.         throw new UnsupportedFlavorException(flavor);
150.     }
151. }
152.
153. private Image theImage;
154. }
```

### 7.13.5 通过系统剪贴板传递Java对象

假设你想从一个Java应用拷贝和粘贴对象到另一个Java应用。在这种情况下，不能使用本地剪贴板，但是可以在系统剪贴板中放置序列化的Java对象。

程序清单7-13中的程序展示了这种能力。该程序显示了一个颜色选择器。Copy按钮将使当前的颜色以被序列化Color对象的方式拷贝到系统剪贴板上。Paste按钮可以用来检查系统剪贴板中是否包含了一个被序列化的Color对象。如果包含的话，它将提取该颜色，并且将它设置为颜色选择器的当前选择。

可以在两个Java应用程序之间传递被序列化的对象（参见图7-43）。运行两个SerialTransferTest程序。在第一个程序上点击Copy按钮，然后，在第二个程序上点击Paste按钮。这时，颜色对象便会从一个虚拟机传递到另一个虚拟机。

为了使能数据传递，Java平台将二进制数据放置到包含了被序列化对象的系统剪贴板上。另一个Java程序能够获取剪贴板中的数据，并且反序列化该对象，该Java程序没有必要与产生剪贴板数据的程序相同。

当然，一个非Java的应用将不知道如何处理剪贴板中的数据。出于这个原因，该示例程序提供了采用第二种风格的剪贴板数据，即文本数据。该文本是对被传递对象调用toString方法得到的结果。如果要查看第二种剪贴板的数据风格，则运行该程序，点击一种颜色，然后在你

的文本编辑器中选中Paste命令。类似于下面这样的一个字符串：

```
java.awt.Color[r=255,g=0,b=51]
```

就会被插入到你的文档中。

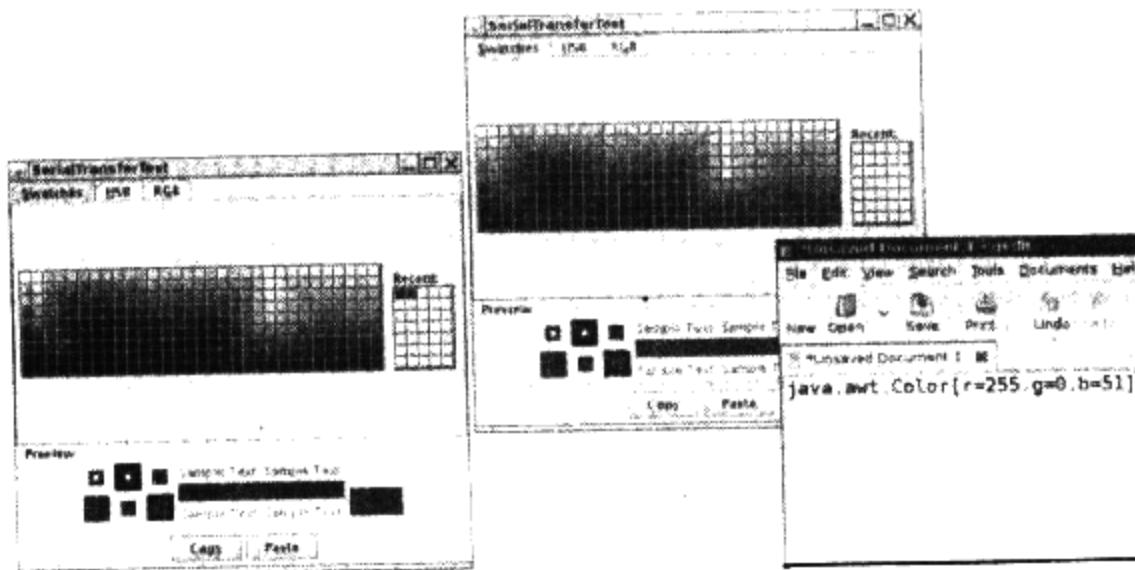


图7-43 数据在一个Java应用的两个实例之间进行拷贝

实际上并不需要进行额外的编程就可以传递一个被序列化的对象，我们可以使用MIME类型：

```
application/x-java-serialized-object;class=className
```

与前面一样，你必须构建自己的传递包装器，细节请参见示例代码。

### 程序清单7-13 SerialTransferTest.Java(continued)

```
1. import java.io.*;
2. import java.awt.*;
3. import java.awt.datatransfer.*;
4. import java.awt.event.*;
5. import javax.swing.*;
6.
7. /**
8. * This program demonstrates the transfer of serialized objects between virtual machines.
9. * @version 1.02 2007-08-16
10. * @author Cay Horstmann
11. */
12. public class SerialTransferTest
13. {
14.     public static void main(String[] args)
15.     {
16.         EventQueue.invokeLater(new Runnable()
17.         {
18.             public void run()
19.             {
20.                 JFrame frame = new SerialTransferFrame();
21.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22.                 frame.setVisible(true);
23.             }
24.         });
25.     }
26. }
```

```
27.  
28. /**  
29. * This frame contains a color chooser, and copy and paste buttons.  
30. */  
31. class SerialTransferFrame extends JFrame  
32. {  
33.     public SerialTransferFrame()  
34.     {  
35.         setTitle("SerialTransferTest");  
36.  
37.         chooser = new JColorChooser();  
38.         add(chooser, BorderLayout.CENTER);  
39.         JPanel panel = new JPanel();  
40.  
41.         JButton copyButton = new JButton("Copy");  
42.         panel.add(copyButton);  
43.         copyButton.addActionListener(new ActionListener()  
44.         {  
45.             public void actionPerformed(ActionEvent event)  
46.             {  
47.                 copy();  
48.             }  
49.         });  
50.  
51.         JButton pasteButton = new JButton("Paste");  
52.         panel.add(pasteButton);  
53.         pasteButton.addActionListener(new ActionListener()  
54.         {  
55.             public void actionPerformed(ActionEvent event)  
56.             {  
57.                 paste();  
58.             }  
59.         });  
60.  
61.         add(panel, BorderLayout.SOUTH);  
62.         pack();  
63.     }  
64.  
65. /**  
66. * Copies the chooser's color into the system clipboard.  
67. */  
68. private void copy()  
69. {  
70.     Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();  
71.     Color color = chooser.getColor();  
72.     Serializable selection = new Serializable(color);  
73.     clipboard.setContents(selection, null);  
74. }  
75.  
76. /**  
77. * Pastes the color from the system clipboard into the chooser.  
78. */  
79. private void paste()  
80. {  
81.     Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();  
82.     try  
83.     {
```

```
84.         DataFlavor flavor = new DataFlavor(
85.             "application/x-java-serialized-object;class=java.awt.Color");
86.         if (clipboard.isDataFlavorAvailable(flavor))
87.         {
88.             Color color = (Color) clipboard.getData(flavor);
89.             chooser.setColor(color);
90.         }
91.     }
92.     catch (ClassNotFoundException e)
93.     {
94.         JOptionPane.showMessageDialog(this, e);
95.     }
96.     catch (UnsupportedFlavorException e)
97.     {
98.         JOptionPane.showMessageDialog(this, e);
99.     }
100.    catch (IOException e)
101.    {
102.        JOptionPane.showMessageDialog(this, e);
103.    }
104. }
105.
106. private JColorChooser chooser;
107. }
108.
109 /**
110 * This class is a wrapper for the data transfer of serialized objects.
111 */
112 class SerialTransferable implements Transferable
113 {
114. /**
115. * Constructs the selection.
116. * @param o any serializable object
117. */
118. SerialTransferable(Serializable o)
119. {
120.     obj = o;
121. }
122.
123. public DataFlavor[] getTransferDataFlavors()
124. {
125.     DataFlavor[] flavors = new DataFlavor[2];
126.     Class<?> type = obj.getClass();
127.     String mimeType = "application/x-java-serialized-object;class=" + type.getName();
128.     try
129.     {
130.         flavors[0] = new DataFlavor(mimeType);
131.         flavors[1] = DataFlavor.stringFlavor;
132.         return flavors;
133.     }
134.     catch (ClassNotFoundException e)
135.     {
136.         return new DataFlavor[0];
137.     }
138. }
```

```

140.     public boolean isDataFlavorSupported(DataFlavor flavor)
141.     {
142.         return DataFlavor.stringFlavor.equals(flavor)
143.             || "application".equals(flavor.getPrimaryType())
144.             && "x-java-serialized-object".equals(flavor.getSubType())
145.             && flavor.getRepresentationClass().isAssignableFrom(obj.getClass());
146.     }
147.
148.     public Object getTransferData(DataFlavor flavor) throws UnsupportedFlavorException
149.     {
150.         if (!isDataFlavorSupported(flavor)) throw new UnsupportedFlavorException(flavor);
151.
152.         if (DataFlavor.stringFlavor.equals(flavor)) return obj.toString();
153.
154.         return obj;
155.     }
156.
157.     private Serializable obj;
158. }
```

### 7.13.6 使用本地剪贴板来传递对象引用

假设你需要拷贝和粘贴一个数据类型，但是该数据类型并不是系统剪贴板所支持的数据类型，且该数据类型是不可序列化的。如果要在同一个虚拟机内传递一个任意的Java对象引用，可以使用MIME类型。

`application/x-java-jvm-local-objectref;class=className`

这时需要为这种类型定义一个`Transferable`包装器，其过程与前面示例中介绍的`ImageSelection`包装器完全相似。

对象的引用只有在单个虚拟机中才有意义。出于这个原因，不能拷贝形状对象到系统剪贴板中。相反，要使用本地剪贴板：

`Clipboard clipboard = new Clipboard("local");`

构造器的参数是剪贴板的名字。

不过，使用本地剪贴板有一个重要的缺点。你必须使本地剪贴板和系统剪贴板同步，这样用户才不会将两者混淆。目前，Java平台并没有执行这个同步的功能。

#### API `java.awt.datatransfer.Clipboard` 1.1

- `Clipboard(String name)`

构建一个带有指定名字的本地剪贴板。

## 7.14 拖放操作

使用剪切和粘贴操作在两个程序之间传递信息时，剪贴板起到了一个中介的作用。拖放操作，打个比方来说就是去掉中间人，让两个程序之间直接通信。Java平台为拖放操作提供了基本的支持特性。可以在Java程序和本地程序之间进行拖放操作。本节将要介绍如何编写作为放置目标的Java应用，以及如何编写作为拖曳源的应用。

在深入介绍Java平台的拖放操作支持特性之前，让我们快速地浏览一些拖放操作的用户界面。我们使用Windows Explorer和WordPad程序作为示例。在其他平台上，读者可以使用本机可用的带有拖放操作的程序来做试验。

可以使用拖曳源中的一个示意动作来初始化一个拖曳操作，通常需要选定一个或者多个元素，然后将选定的目标拖离它的初始位置。当你在一个接收放置操作的放置目标上释放鼠标按键时，放置目标将查询拖曳源，进而了解关于放置元素的信息，并且启动某个恰当的操作。例如，如果将一个文件图标从从文件管理器中拖放到某个目录图标的上面，那么这个文件就会被移动到这个目录中。但是，如果将它拖放到一个文本编辑器中，那么文本编辑器就会打开这个文件。（当然，这要求我们所使用的文件管理器和文本编辑器支持拖放操作，例如Windows中的Explore与WordPad，以及Gnome中的Nautilus与gedit）。

如果在拖曳的时候按住CTRL键，那么放置操作的类型将从移动操作变为拷贝操作，该文件的一份拷贝被放入此目录中。如果同时按住了SHIFT和CTRL键，那么该文件的一个链接将被放入到此目录中。（其他平台可能使用别的按键组合来执行这些操作）

因此，有三种带有不同示意动作的放置操作：

- 移动；
- 拷贝；
- 链接。

链接操作的目的是建立一个对被放置元素的引用。这种链接通常需要得到本机操作系统的支持（比如用于文件的符号链接，或者用于文档构件的对象链接），并且它通常在跨平台的程序中没有太大的意思。在本节中，我们将着重介绍如何使用拖放操作来进行拷贝和移动。

拖曳操作通常能够产生某种直观的反馈信息，至少光标的形状会发生改变。当你把光标移动到可能的放置目标上时，光标的形状将会表示出放置操作是否可行。如果放置操作可行的话，光标的形状也会表示出放置动作的类型。表7-6显示了光标在放置目标上所显示的几种形状。

除了文件图标外，也可以拖曳别的元素。例如，可以在WordPad 中选择文本，然后拖曳之。请试着将文本字段放到你希望放置的对象中，并且观察它们做何反应。

表7-6 放置光标的形状

| 动 作  | Windows图标 | Gnome图标 |
|------|-----------|---------|
| 移动   |           |         |
| 拷贝   |           |         |
| 链接   |           |         |
| 不准放置 |           |         |

注意：这个试验显示了作为用户界面机制的拖放操作的一个缺点。用户很难预计究竟能拖曳什么，可以将它们放置到何处，以及当实施拖放操作时会出现什么情况。由于默认的移动操作能够删除原始的元素，因此用户在使用拖放操作时比较谨慎，这是可以理解的。

#### 7.14.1 Swing对数据传递的支持

从Java SE1.4开始，多种Swing构件就已经内置了对数据传递的支持（参加表7-7）。我们可以从大量的构件中拖曳选中的文本，也可以将文本放置到文本构件中。为了向后兼容性，我们

必须使用`setDragEnabled`方法来激活拖曳功能，而放置功能总是会得到支持的。

表7-7 Swing中支持数据传递的构件

| 构 件                             | 拖 曳 源        | 放 置 目 标   |
|---------------------------------|--------------|-----------|
| JFileChooser                    | 导出文件列表       | 无         |
| JColorChooser                   | 导出颜色对象       | 接收颜色对象    |
| JTextField JFormattedTextField  | 导出选定的文本      | 接收文本      |
| JPasswordField                  | 无（由于安全的原因）   | 接收文本      |
| JTextArea JTextPane JEditorPane | 导出选定的文本      | 接收文本和文件列表 |
| JList JTable JTree              | 导出所选择的HTML描述 | 无         |

 注意：`java.awt.dnd`包提供了一个低层的拖放API，它形成了Swing拖放的基础。我们在本书中不讨论这个API。

程序清单7-14中的程序演示了这种行为。在运行该程序时，应该注意下面几点：

- 你可以选择列表、表格或树中的多个项，并拖曳它们。
- 从表格中拖曳项有些尴尬，你需要先用鼠标选择，然后移走鼠标，之后再次点击它，这之后才能拖曳它。
- 当你在文本域中放置项时，可以看到被拖曳的信息是如何被格式化的：表格中的表元由制表符隔开，而每个选中的行都占据单独的一行（参见图7-44）。

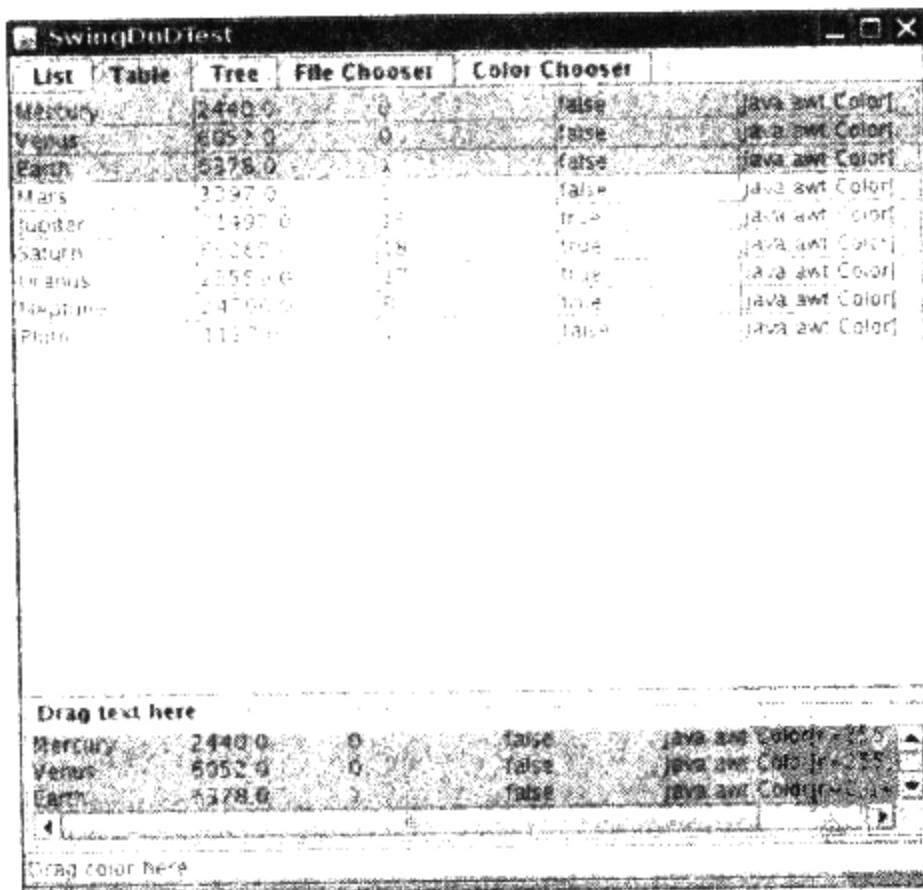


图7-44 Swing的拖放测试程序

- 你只能拷贝不能移动列表、表格、树、文件选择器或颜色选择器中的项。对于所有的数据模型来说，从列表、表格或树中移除项都是不可能的。在下一节中你将会看到当数据

模型可编辑时，可以如何实现这种移除能力。

- 你不能在列表、表格、树或文件选择器中拖曳。
- 如果你运行该程序的两个副本，就可以将颜色从一个颜色选择器中拖曳到另一个中。
- 你不能将文本从文本域中拖出，因为我们没有在文本域上调用setDragEnabled。

Swing包提供一个潜在的非常有用的机制，可以迅速地将一个构件转换成一个放置目标。我们可以为给定的属性安装一个传递处理器，例如，在示例程序中，我们调用了：

```
textField.setTransferHandler(new TransferHandler("background"));
```

现在，可以在文本框中拖曳颜色，其背景色会随之改变。

当发生了一个放置操作时，传递处理器将检查是否有一种数据风格的表示类为Color，如果确实如此，那么它便调用setBackground方法。

通过把传递处理器安装在文本区域中，就可以禁用标准的传递处理器。你再也不能在此文本域中进行剪切、拷贝、粘贴、拖曳或者放置操作了。一个Swing构件只能拥有一个传递处理器。

#### 程序清单7-14 SwingDnDTest.java

```
1. import java.awt.*;
2.
3. import javax.swing.*;
4. import javax.swing.border.*;
5. import javax.swing.event.*;
6.
7. /**
8. * This program demonstrates the basic Swing support for drag and drop.
9. * @version 1.10 2007-09-20
10. * @author Cay Horstmann
11. */
12. public class SwingDnDTest
13. {
14.     public static void main(String[] args)
15.     {
16.         EventQueue.invokeLater(new Runnable()
17.         {
18.             public void run()
19.             {
20.                 JFrame frame = new SwingDnDFrame();
21.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22.                 frame.setVisible(true);
23.             }
24.         });
25.     }
26. }
27.
28. class SwingDnDFrame extends JFrame
29. {
30.     public SwingDnDFrame()
31.     {
32.         setTitle("SwingDnDTest");
33.         JTabbedPane tabbedPane = new JTabbedPane();
34.
35.         JList list = SampleComponents.list();
36.         tabbedPane.addTab("List", list);
```

```

37.     JTable table = SampleComponents.table();
38.     tabbedPane.addTab("Table", table);
39.     JTree tree = SampleComponents.tree();
40.     tabbedPane.addTab("Tree", tree);
41.     JFileChooser fileChooser = new JFileChooser();
42.     tabbedPane.addTab("File Chooser", fileChooser);
43.     JColorChooser colorChooser = new JColorChooser();
44.     tabbedPane.addTab("Color Chooser", colorChooser);
45.
46.     final JTextArea textArea = new JTextArea(4, 40);
47.     JScrollPane scrollPane = new JScrollPane(textArea);
48.     scrollPane.setBorder(new TitledBorder(new EtchedBorder(), "Drag text here"));
49.
50.     JTextField textField = new JTextField("Drag color here");
51.     textField.setTransferHandler(new TransferHandler("background"));
52.
53.     tabbedPane.addChangeListener(new ChangeListener()
54.     {
55.         public void stateChanged(ChangeEvent e)
56.         {
57.             textArea.setText("");
58.         }
59.     });
60.
61.     tree.setDragEnabled(true);
62.     table.setDragEnabled(true);
63.     list.setDragEnabled(true);
64.     fileChooser.setDragEnabled(true);
65.     colorChooser.setDragEnabled(true);
66.     textField.setDragEnabled(true);
67.
68.     add(tabbedPane, BorderLayout.NORTH);
69.     add(scrollPane, BorderLayout.CENTER);
70.     add(textField, BorderLayout.SOUTH);
71.     pack();
72. }
73. }

```



### **javax.swing.JComponent 1.2**

- **void setTransferHandler(TransferHandler handler) 1.4**

设置一个用来处理数据传递操作（剪切、拷贝、粘贴、拖曳、放置）的传递处理器。



### **javax.swing.TransferHandler 1.4**

- **transferHandler(String propertyName)**

构建一个传递处理器，它可以在执行数据传递操作时读取或者写入带有给定名称的 JavaBeans 构件的属性。



### **javax.swing.JFileChooser 1.2**

### **javax.swing.JColorChooser 1.2**

### **javax.swing.JTextField 1.2**

### **javax.swing.JList 1.2**

**API** javax.swing.JTable 1.2**javax.swing.JTree 1.2**

- void setDragEnabled(boolean b) 1.4

使将数据从该构件中拖曳出去的操作可用或者禁用。

### 7.14.2 拖曳源

在前一节中，你已经看到了如何利用Swing中基本的拖放支持。在本节中，我们将向你展示如何将任意的构件配置成拖曳源。在下一节中，我们将讨论放置目标，并将一个示例构件同时设置为图像的拖曳源和放置目标。

为了定制Swing构件的拖放行为，必须子类化TransferHandler类。首先，覆写getSourceActions方法，以表示该构件支持什么样的行为（拷贝、移动、链接）。接下来，覆写getTransferable方法，以产生Transferable对象，其过程遵循向剪贴板拷贝对象的过程。

在示例程序中，我们从一个JList中拖曳图像，这个JList填充了图像的图标（参见图7-45）。下面是createTransferable方法的实现，被选中的图像直接放置到一个ImageTransferable包装器中。

```
protected Transferable createTransferable(Component source)
{
    JList list = (JList) source;
    int index = list.getSelectedIndex();
    if (index < 0) return null;
    ImageIcon icon = (ImageIcon) list.getModel().getElementAt(index);
    return new ImageTransferable(icon.getImage());
}
```

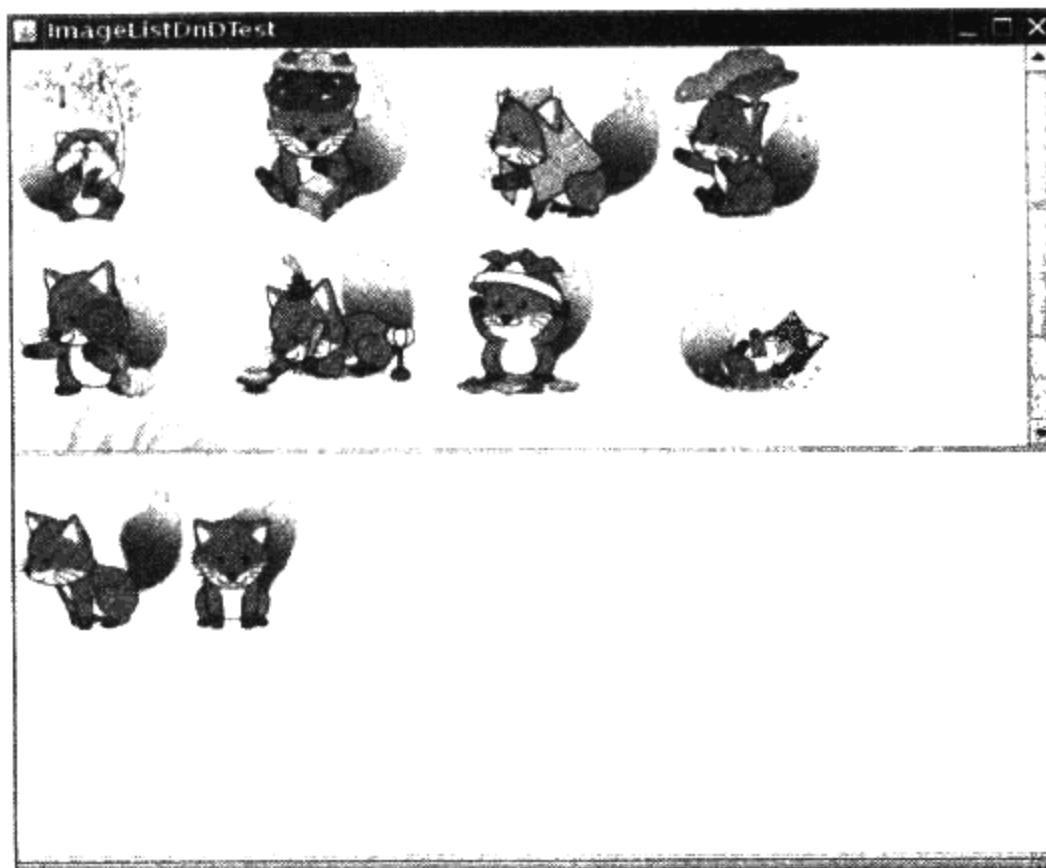


图7-45 ImageList的拖放应用

本例中，我们庆幸的是JList已经具备了启动拖曳姿态的机制，你只需通过调用setDragEnabled方法来激活这种机制。如果你希望向不识别拖曳姿态的构件中添加对拖曳操作的支持，则需要由你自己来启动这种传递。例如，下面的代码展示了如何在JLabel上启动拖曳：

```
label.addMouseListener(new MouseAdapter()
{
    public void mousePressed(MouseEvent evt)
    {
        int mode;
        if ((evt.getModifiers() & (InputEvent.CTRL_MASK | InputEvent.SHIFT_MASK)) != 0)
            mode = TransferHandler.COPY;
        else mode = TransferHandler.MOVE;
        JComponent comp = (JComponent) evt.getSource();
        TransferHandler th = comp.getTransferHandler();
        th.exportAsDrag(comp, evt, mode);
    }
});
```

这里，我们只是在用户在标签上点击时启动传递。更复杂的实现还可以观察引起鼠标微量拖曳的鼠标移动。

当用户完成放置行为后，拖曳源传递处理器的exportDone方法就会被调用，在这个方法中，如果用户执行了移动动作，则应该移除被传递的对象。下面是图像列表的相关实现：

```
protected void exportDone(JComponent source, Transferable data, int action)
{
    if (action == MOVE)
    {
        JList list = (JList) source;
        int index = list.getSelectedIndex();
        if (index < 0) return;
        DefaultListModel model = (DefaultListModel) list.getModel();
        model.remove(index);
    }
}
```

总结一下，为了使一个构件成为拖曳源，需要添加一个指定了下列内容的传递处理器：

- 可以支持哪些行为。
- 可以传输哪些数据。
- 在执行移动动作之后，如何移除原来的数据。

此外，如果拖曳源是表7-7中所列构件之外的构件，则还需要观察鼠标姿态和启动传递。



#### javax.swing.TransferHandler 1.4

- int getSourceActions(JComponent c)

覆盖该方法，让其返回在给定的构件上进行拖曳时，允许对拖曳源执行的动作（COPY、MOVE和LINK的位组合）。

- protected Transferable createTransferable(JComponent source)

覆盖该方法，让其为被拖曳的数据创建一个Transferable。

- void exportAsDrag(JComponent comp, InputEvent e, int action)

从给定的构件中启动一个拖曳姿态，action参数的值是COPY、MOVE或LINK。

- `protected void exportDone(JComponent source, Transferable data, int action)`  
覆写该方法，让其在传递成功之后调整拖曳源。

### 7.14.3 放置目标

在本节中，我们将展示如何实现放置目标。我们用到的示例还是填充了图像的图标的JList，在其中我们添加了对放置的支持，以便用户可以将图像放置到列表中。

要使一个构件成为放置目标，需要设置一个TransferHandler，并实现canImport和importData方法。

 **注意：**从Java SE6开始，我们可以向JFrame中添加传递处理器。其最常用到的地方就是在应用中放置文件，有效的放置位置包括窗体的装饰和菜单条，但是不包括窗体中包含的构件（它们有自己的传递处理器）。

当用户在放置的目标构件上移动鼠标时，canImport方法会被连续调用，如果放置是允许的，则返回true。这个信息对光标的图标产生影响，因为这个图标要对是否允许放置给出可视的反馈。

从Java SE6开始，canImport方法有了一个TransferHandler.TransferSupport类型的参数，通过这个参数，可以获取用户选择的放置动作、放置位置以及要传输的数据。（在Java SE6之前，调用的是与此不同的一个canImport方法，它只提供数据风格的列表。）

在canImport方法中，还可以覆写用户的放置动作。例如，如果用户选择了移动动作，但是移除原有项是不恰当的，那么就可以强制传递处理器使用拷贝动作取而代之。

下面是一个典型的示例：假设图像列表构件将接受放置文件列表和图像的请求，但是，如果一个文件列表被拖曳到了这个构件中，那么用户选择的MOVE动作就会改为COPY动作，这样图像文件就不会被删除。

```
public boolean canImport(TransferSupport support)
{
    if (support.isDataFlavorSupported(DataFlavor.javaFileListFlavor))
    {
        if (support.getUserDropAction() == MOVE) support.setDropAction(COPY);
        return true;
    }
    else return support.isDataFlavorSupported(DataFlavor.imageFlavor);
}
```

更复杂的实现可以检查拖曳的文件中是否确实包含图像。

当鼠标在放置目标上移动时，Swing构件JList、JTable、JTree和JTextComponent会给出有关插入位置的可视反馈。默认情况下，选中（对于JList、JTable和JTree而言）或脱字符（对于JTextComponent而言）被用来表示放置位置。这种方法对用户来说显得很不友好，而且也不灵活，它被设置成默认值只是为了向后兼容。应该调用setDropMode方法来选择更恰当的可视反馈。

可以控制被放置的数据是应该覆盖已有项，还是应该插入到已有项的中间。例如，在示例程序中，我们调用了`setDropMode(DropMode.ON_OR_INSERT)`；

以允许用户在某一项上放置（因此也就覆盖了这一项），或者在两项之间插入（参见图7-46）。表7-8给出了Swing构件支持的放置模式。



图7-46 在某一项上放置的可视指示器和在两项之间放置的可视指示器

表7-8 放置模式

| 构    件         | 支持的放置模式   |
|----------------|---|
| JList, JTree   | ON, INSERT, ON_OR_INSERT, USE_SELECTION   |
| JTable         | ON, INSERT, ON_OR_INSERT, INSERT_ROWS, INSERT_COLS, ON_OR_INSERT_ROWS, ON_OR_INSERT_COLS, USE_SELECTION |
| JTextComponent | INSERT, USE_SELECTION (实际上是移动到脱字符，而不是选中的字符)   |

一旦用户结束了放置姿态，`importData`方法就会被调用。此时需要从拖曳源获得数据，在`TransferSupport`参数上调用`getTransferable`方法就可以获得一个对`Transferable`对象的引用。这与拷贝和粘贴时使用的接口相同。

拖放最常用的一种数据类型是`DataFlavor.javaFileListFlavor`。文件列表描述了要放置到目标上的文件集合，而传递数据就是`List<File>`类型的一个对象。下面的代码可以获取这些文件：

```
 DataFlavor[] flavors = transferable.getTransferDataFlavors();
 if (Arrays.asList(flavors).contains(DataFlavor.javaFileListFlavor))
 {
     List<File> fileList = (List<File>) transferable.getTransferData(DataFlavor.javaFileListFlavor);
     for (File f : fileList)
     {
         do something with f;
     }
 }
```

在放置表7-8中列出的构件时，需要知道放置数据的精确位置。在`TransferSupport`参数上调用`getDropLocation`方法可以发现产生放置动作的位置，这个方法将返回一个`TransferHandler.DropLocation`的某个子类的对象。`JList`、`JTable`、`JTree`和`JTextComponent`类都定义了在特定的数据模型中指定位置的子类。例如，在列表中的位置可以是一个整数，但是树中的位置就必须是一个树的路径。下面的代码展示了如何在我们的图像列表中获取放置位置：

```
 int index;
 if (support.isDrop())
 {
```

```
JList.DropLocation location = (JList.DropLocation) support.getDropLocation();
index = location.getIndex();
}
else index = model.size();
```

JList.DropLocation子类有一个getIndex方法，该方法返回放置位置的索引。  
(JTree.DropLocation子类有一个类似的getPath方法。)

在数据通过CTRL+V组合键粘贴到构件中时，importData方法也会被调用。在这种情况下，getDropLocation方法将抛出IllegalStateException。因此，如果isDrop方法返回false，我们就只是将粘贴的数据追加到列表的尾部。

在向列表、表格或树中插入时，还需要检查数据是要插入到项之间，还是应该替换插入位置的项。对于列表，可以调用JList.DropLocation的isInsert方法，对于其他的构件，请查看本节末尾关于它们的放置位置类的API说明。

总结一下，为了使一个构件成为放置目标，需要添加一个指定了下列内容的传递处理器：

- 何时可以接受被拖曳的项。
- 如何导入被放置的数据。

此外，如果要向JList、JTable、JTree和JTextComponent添加对放置的支持，还应该设置放置模式。

程序清单7-15展示了完整的程序。注意，ImageList类既是拖曳源，又是放置目标。请尝试在两个列表之间拖曳图像，也可以从其他程序的文件选择器中拖曳图像文件到这些列表中。

### 程序清单7-15 ImageListDragDrop.java

```
1. import java.awt.*;
2. import java.awt.datatransfer.*;
3. import java.io.*;
4. import java.util.*;
5. import javax.imageio.*;
6. import javax.swing.*;
7. import java.util.List;
8.
9. /**
10. * This program demonstrates drag and drop in an image list.
11. * @version 1.00 2007-09-20
12. * @author Cay Horstmann
13. */
14. public class ImageListDnDTest
15. {
16.     public static void main(String[] args)
17.     {
18.         EventQueue.invokeLater(new Runnable()
19.         {
20.             public void run()
21.             {
22.                 JFrame frame = new ImageListDnDFrame();
23.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24.                 frame.setVisible(true);
25.             }
26.         });
27.     }
```

```
28. }
29.
30. class ImageListDnDFrame extends JFrame
31. {
32.     public ImageListDnDFrame()
33.     {
34.         setTitle("ImageListDnDTest");
35.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
36.
37.         list1 = new ImageList(new File("images1").listFiles());
38.         list2 = new ImageList(new File("images2").listFiles());
39.         setLayout(new GridLayout(2, 1));
40.         add(new JScrollPane(list1));
41.         add(new JScrollPane(list2));
42.     }
43.
44.     private ImageList list1;
45.     private ImageList list2;
46.     private static final int DEFAULT_WIDTH = 600;
47.     private static final int DEFAULT_HEIGHT = 500;
48. }
49.
50. class ImageList extends JList
51. {
52.     public ImageList(File[] imageFiles)
53.     {
54.         DefaultListModel model = new DefaultListModel();
55.         for (File f : imageFiles)
56.             model.addElement(new ImageIcon(f.getPath()));
57.
58.         setModel(model);
59.         setVisibleRowCount(0);
60.         setLayoutOrientation(JList.HORIZONTAL_WRAP);
61.         setDragEnabled(true);
62.         setDropMode(DropMode.ON_OR_INSERT);
63.         setTransferHandler(new ImageListTransferHandler());
64.     }
65. }
66.
67. class ImageListTransferHandler extends TransferHandler
68. {
69.     // Support for drag
70.
71.     public int getSourceActions(Component source)
72.     {
73.         return COPY_OR_MOVE;
74.     }
75.
76.     protected Transferable createTransferable(Component source)
77.     {
78.         JList list = (JList) source;
79.         int index = list.getSelectedIndex();
80.         if (index < 0) return null;
81.         ImageIcon icon = (ImageIcon) list.getModel().getElementAt(index);
82.         return new ImageTransferable(icon.getImage());
83.     }
84.
```

```
85.     protected void exportDone(Component source, Transferable data, int action)
86.     {
87.         if (action == MOVE)
88.         {
89.             JList list = (JList) source;
90.             int index = list.getSelectedIndex();
91.             if (index < 0) return;
92.             DefaultListModel model = (DefaultListModel) list.getModel();
93.             model.remove(index);
94.         }
95.     }
96.
97. // Support for drop
98.
99. public boolean canImport(TransferSupport support)
100. {
101.     if (support.isDataFlavorSupported(DataFlavor.javaFileListFlavor))
102.     {
103.         if (support.getUserDropAction() == MOVE) support.setDropAction(COPY);
104.         return true;
105.     }
106.     else return support.isDataFlavorSupported(DataFlavor.imageFlavor);
107. }
108.
109. public boolean importData(TransferSupport support)
110. {
111.     JList list = (JList) support.getComponent();
112.     DefaultListModel model = (DefaultListModel) list.getModel();
113.
114.     Transferable transferable = support.getTransferable();
115.     List<DataFlavor> flavors = Arrays.asList(transferable.getTransferDataFlavors());
116.
117.     List<Image> images = new ArrayList<Image>();
118.
119.     try
120.     {
121.         if (flavors.contains(DataFlavor.javaFileListFlavor))
122.         {
123.             List<File> fileList = (List<File>) transferable
124.                 .getTransferData(DataFlavor.javaFileListFlavor);
125.             for (File f : fileList)
126.             {
127.                 try
128.                 {
129.                     images.add(ImageIO.read(f));
130.                 }
131.                 catch (IOException ex)
132.                 {
133.                     // couldn't read image--skip
134.                 }
135.             }
136.         }
137.         else if (flavors.contains(DataFlavor.imageFlavor))
138.         {
139.             images.add((Image) transferable.getTransferData(DataFlavor.imageFlavor));
140.         }
141.     }
```

```

142.     int index;
143.     if (support.isDrop())
144.     {
145.         JList.DropLocation location = (JList.DropLocation) support.getDropLocation();
146.         index = location.getIndex();
147.         if (!location.isInsert()) model.remove(index); // replace location
148.     }
149.     else index = model.size();
150.     for (Image image : images)
151.     {
152.         model.add(index, new ImageIcon(image));
153.         index++;
154.     }
155.     return true;
156. }
157. catch (IOException ex)
158. {
159.     return false;
160. }
161. catch (UnsupportedFlavorException ex)
162. {
163.     return false;
164. }
165. }
166. }

```

### **javax.swing.TransferHandler 1.4**

- **boolean canImport(TransferSupport support) 6**

覆盖该方法，让其表示目标构件是否能够接受TransferSupport参数所描述的拖曳。

- **boolean importData(TransferSupport support) 6**

覆盖该方法，让其实现由TransferSupport参数描述的放置或粘贴姿态，并且在导入成功时返回true。

### **javax.swing.JFrame 1.2**

- **void setTransferHandler(TransferHandler handler) 6**

将传递处理器设置成为只处理放置和粘贴操作。

### **javax.swing.JList 1.2**

**javax.swing.JTable 1.2**

**javax.swing.JTree 1.2**

**javax.swing.JTextField 1.2**

- **void setDropMode(DropMode mode) 6**

将这个构件的放置模式设置成为表7-8中指定的值之一。

### **javax.swing.TransferHandler.TransferSupport 6**

- **Component getComponent()**

获取这个传递的目标构件。

- `DataFlavor[] getDataFlavors()`  
获取被传递数据的数据风格。
- `boolean isDrop()`  
如果这个传递是放置，则返回true，如果是粘贴则返回false。
- `int getUserDropAction()`  
获取由用户选择的放置动作(MOVE、COPY或LINK)。
- `getSourceDropActions()`  
获取拖曳源允许执行的放置动作。
- `getDropAction()`
- `setDropAction()`  
获取和设置这个传递的放置动作。最初，这是一个用户的放置动作，但是它可以被传递处理器所覆盖。
- `DropLocation getDropLocation()`  
获取放置的鼠标位置，如果该传递不是一个放置动作，则抛出IllegalStateException。



#### **javax.swing.TransferHandler.DropLocation 6**

- `Point getDropPoint()`  
获取在目标构件中放置的鼠标位置。



#### **javax.swing.JList.DropLocation 6**

- `boolean isInsert()`  
如果数据被插入到给定位置之前，则返回true，如果它们替换了已有数据，则返回false。
- `int getIndex()`  
获取模型中用于插入或替换的索引。



#### **javax.swing.JTable.DropLocation 6**

- `boolean isInsertRow()`
- `boolean isInsertColumn()`  
如果输入被插入到某行或某列之前，则返回true。
- `int getRow()`
- `int getColumn()`  
获取模型中用于插入或替换的行或列索引，如果放置发生在空区域，则返回-1。



#### **javax.swing.JTree.DropLocation 6**

- `TreePath getPath()`
- `int getChildIndex()`  
返回树的路径和孩子，以及目标构件的放置模式，该模式定义了拖放的位置，如表7-9所示。

表7-9 JTree中的放置位置的处理机制

| 放置模式                | 树编辑动作                                |
|---------------------|--------------------------------------|
| INSERT              | 作为该路径的一个孩子插入，插入到获得的孩子索引之前            |
| ON or USE_SELECTION | 替换该路径中的数据（没用到孩子索引）                   |
| INSERT_OR_ON        | 如果获得的孩子索引为-1，则以ON模式执行，否则，以INSERT模式执行 |

**API** javax.swing.JTextFieldComponent.DropLocation 6

- int getIndex()

插入数据处的索引。

## 7.15 平台集成

我们用新添加到Java SE6中的几个特性来结束本章，这些特性使得Java应用看起来更像是本地应用。闪屏特性使得应用在虚拟机启动时可以显示一个闪屏；java.awt.Desktop类使我们可以启动本地应用，例如默认的浏览器和e-mail程序；最后，可以像许多本地应用一样，对系统托盘进行访问，并可以用图标来塞满它。

### 7.15.1 闪屏

对Java应用最常见的抱怨就是启动时间太长。这是因为Java虚拟机花费了一段时间去加载所有必需的类，特别是对Swing应用，它们需要从Swing和AWT类库代码中抽取大量的内容。用户并不喜欢应用程序花费大量的时间去产生初始屏幕，他们甚至可能在不知道首次启动是否成功的情况下尝试着多次启动该应用程序。此问题的解决之道是采用闪屏，即迅速出现的小窗体，它可以告诉用户该应用程序已经成功启动了。

传统上，这对于Java应用来说很难实现。当然，我们可以在main方法开始之后立即呈现一个窗体，但是，main方法只有在类加载器加载了所有需要依赖的类之后才会被启动，而这一过程可能要等上一段时间。

Java SE 6通过支持虚拟机在启动时立即显示一幅图像而解决了这个问题。有两种机制可以指定这幅图像，一种是使用命令行参数-splash：

```
java -splash:myimage.png MyApp
```

另一种是在JAR文件的清单中指定：

```
Main-Class: MyApp
SplashScreen-Image: myimage.gif
```

这幅图像会在第一个AWT窗体可视时立即自动消失。我们可以使用任何GIF、JPEG或PNG图像，动画（GIF）和透明（GIF和PNG）都可以得到支持。

如果你的应用程序在达到main之后立即就可以执行，那么你就可以略过本节余下的内容。但是，许多应用使用了插件架构，其中有一个小内核，它将在启动时加载插件集。Eclipse和NetBeans就是典型的实例。在这种情况下，可以用闪屏来表示加载进度。

有两种方式来实现上述功能，即可以直接在闪屏上绘制，或者用含有相同内容的无边界窗

体来替换初始图像，然后在该窗体的内部绘制。我们的示例程序同时展示了这两种技术。

为了直接在闪屏上绘制，需要获取一个对闪屏的引用，以及它的图形上下文与尺寸：

```
SplashScreen splash = SplashScreen.getSplashScreen();
Graphics2D g2 = splash.createGraphics();
Rectangle bounds = splash.getBounds();
```

现在可以按照常规的方式来绘制了。当绘制完成后，调用`update`来确保绘制的图画被刷新。我们的示例程序绘制了一个简单的进度条，就像在图7-47中左边一幅图中看到的那样。

```
g.fillRect(x, y, width * percent / 100, height);
splash.update();
```



图7-47 初始的闪屏和无边界的后续视窗



**注意：**闪屏是单例对象，因此你不能创建自己的闪屏对象。如果在命令行或清单中没有设置任何闪屏，`getSplashScreen`方法将返回null。

直接在闪屏上绘制有一个缺陷，即计算所有的像素位置会显得很冗长，而且进度指示器不会去观察本地进度条。为了避免这些问题，可以在`main`方法启动后立即将初始闪屏用具有相同尺寸和内容的后续视窗替换。这个视窗可以包含任意的Swing构件。

程序清单7-16中的示例程序展示了这种技术。图7-47右边的那幅图展示了一个无边界的窗体，它有一个面板，绘制了闪屏并包含一个`JProgressBar`。现在我们对Swing API有了完整的访问能力，可以很轻松地添加消息字符串而不用受像素位置的困扰了。

请注意，我们不需要移除初始闪屏，它会在后续视窗可视之后被自动移除掉。



**警告：**遗憾的是，在闪屏被后续视窗替代时，会有明显的闪烁。

**程序清单7-16 SplashScreenTest.java**

```
1. import java.awt.*;
2. import java.util.List;
3. import javax.swing.*;
4.
5. /**
6. * This program demonstrates the splash screen API.
7. * @version 1.00 2007-09-21
8. * @author Cay Horstmann
9. */
10. public class SplashScreenTest
11 {
12     private static void drawOnSplash(int percent)
13     {
14         Rectangle bounds = splash.getBounds();
15         Graphics2D g = splash.createGraphics();
16         int height = 20;
17         int x = 2;
18         int y = bounds.height - height - 2;
19         int width = bounds.width - 4;
20         Color brightPurple = new Color(76, 36, 121);
21         g.setColor(brightPurple);
22         g.fillRect(x, y, width * percent / 100, height);
23         splash.update();
24     }
25.
26. /**
27. * This method draws on the splash screen.
28. */
29. private static void init1()
30 {
31     splash = SplashScreen.getSplashScreen();
32     if (splash == null)
33     {
34         System.err.println("Did you specify a splash image with -splash or in the manifest?");
35         System.exit(1);
36     }
37.
38     try
39     {
40         for (int i = 0; i <= 100; i++)
41         {
42             drawOnSplash(i);
43             Thread.sleep(100); // simulate startup work
44         }
45     }
46     catch (InterruptedException e)
47     {
48     }
49 }
50.
51. /**
52. * This method displays a frame with the same image as the splash screen.
53. */
54. private static void init2()
```

```
55.    {
56.        final Image img = Toolkit.getDefaultToolkit().getImage(splash.getImageURL());
57.
58.        final JFrame splashFrame = new JFrame();
59.        splashFrame.setUndecorated(true);
60.
61.        final JPanel splashPanel = new JPanel()
62.        {
63.            public void paintComponent(Graphics g)
64.            {
65.                g.drawImage(img, 0, 0, null);
66.            }
67.        };
68.
69.        final JProgressBar progressBar = new JProgressBar();
70.        progressBar.setStringPainted(true);
71.        splashPanel.setLayout(new BorderLayout());
72.        splashPanel.add(progressBar, BorderLayout.SOUTH);
73.
74.        splashFrame.add(splashPanel);
75.        splashFrame.setBounds(splash.getBounds());
76.        splashFrame.setVisible(true);
77.
78.        new SwingWorker<Void, Integer>()
79.        {
80.            protected Void doInBackground() throws Exception
81.            {
82.                try
83.                {
84.                    for (int i = 0; i <= 100; i++)
85.                    {
86.                        publish(i);
87.                        Thread.sleep(100);
88.                    }
89.                }
90.                catch (InterruptedException e)
91.                {
92.                }
93.                return null;
94.            }
95.
96.            protected void process(List<Integer> chunks)
97.            {
98.                for (Integer chunk : chunks)
99.                {
100.                    progressBar.setString("Loading module " + chunk);
101.                    progressBar.setValue(chunk);
102.                    splashPanel.repaint(); // because img is loaded asynchronously
103.                }
104.            }
105.
106.            protected void done()
107.            {
108.                splashFrame.setVisible(false);
109.
110.                JFrame frame = new JFrame();
111.                frame.setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
```

```

112         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
113         frame.setTitle("SplashScreenTest");
114         frame.setVisible(true);
115     }
116     }.execute();
117 }
118
119 public static void main(String args[])
120 {
121     init1();
122
123     EventQueue.invokeLater(new Runnable()
124     {
125         public void run()
126         {
127             init2();
128         }
129     });
130 }
131
132 private static SplashScreen splash;
133 private static final int DEFAULT_WIDTH = 300;
134 private static final int DEFAULT_HEIGHT = 300;
135 }

```

### **java.awt.SplashScreen** 6

- **static SplashScreen get SplashScreen()**

获取一个对闪屏的引用，如果目前没有任何闪屏，则返回null。

- **URL getImageURL()**

- **void setImageURL(URL imageURL)**

获取或设置闪屏图像的URL。设置该图像会更新闪屏。

- **Rectangle getBounds()**

获取闪屏的边界。

- **Graphics2D createGraphics()**

获取用于在闪屏上绘制的图形上下文。

- **void update()**

更新闪屏的显示。

- **void close()**

关闭闪屏。闪屏在第一个AWT视窗可视时会自动关闭。

## 7.15.2 启动桌面应用程序

**java.awt.Desktop**类使我们可以启动默认的浏览器和e-mail程序，我们还可以用注册为用于某类文件类型的应用程序来打开、编辑和打印这类文件。

其API是很直观的。首先，调用静态的`isDesktopSupported`方法，如果它返回true，则当前平台支持启动桌面应用程序。然后调用静态的`getDesktop`方法来获取一个`Desktop`实例。

并非所有桌面环境都支持所有的API操作。例如，在Linux上的Gnome桌面中，有可能可以打开文件，但是不能打印它们。（目前没有对文件关联中的“动词”进行支持。）要查明平台所支持的操作，可以调用`isSupported`方法，并将`Desktop.Action`枚举中的某个值传给它。我们的示例程序中包含了下面这样的测试：

```
if (desktop.isSupported(Desktop.Action.PRINT)) printButton.setEnabled(true);
```

为了打开、编辑和打印文件，首先要检查这个动作是否得到了支持，然后再调用`open`、`edit`和`print`方法。为了启动浏览器，需要传递一个URI。（有关URI的更多信息可参见第3章。）可以直接用包含一个http或https的URL的字符串来调用URI构造器。

为了启动默认的e-mail程序，需要构造一个具有特定格式的URI，即：

```
mailto:recipient?query
```

这里`recipient`是接收者的e-mail地址，例如`president@whitehouse.gov`，而`query`包含了用&分隔的`name=value`对，其中值是用百分号编码的。（百分号编码机制实质与第3章所描述的URL编码机制算法相同，但是空格被编码为%20，而不是+）。`subject=dinner%20RSVP&bcc=putin%40kremvax.ru`是一个实例。这种格式归档在RFC2368中（<http://www.ietf.org/rfc/rfc2368.txt>）。但是，URI类不了解有关mailto这类URI的任何信息，因此我们必须组装和编码自己的URI。更糟的是，在编写本书时，仍然没有任何用于处理非ASCII字符的标准。一种常用的方式（我们也这么用）是将每个字符都转换为UTF-8，并用百分号编码所产生的字节。

程序清单7-17的示例程序使你可以打开、编辑或打印你选择的文件，可以浏览一个URL，或者启动e-mail程序（参见图7-48）。

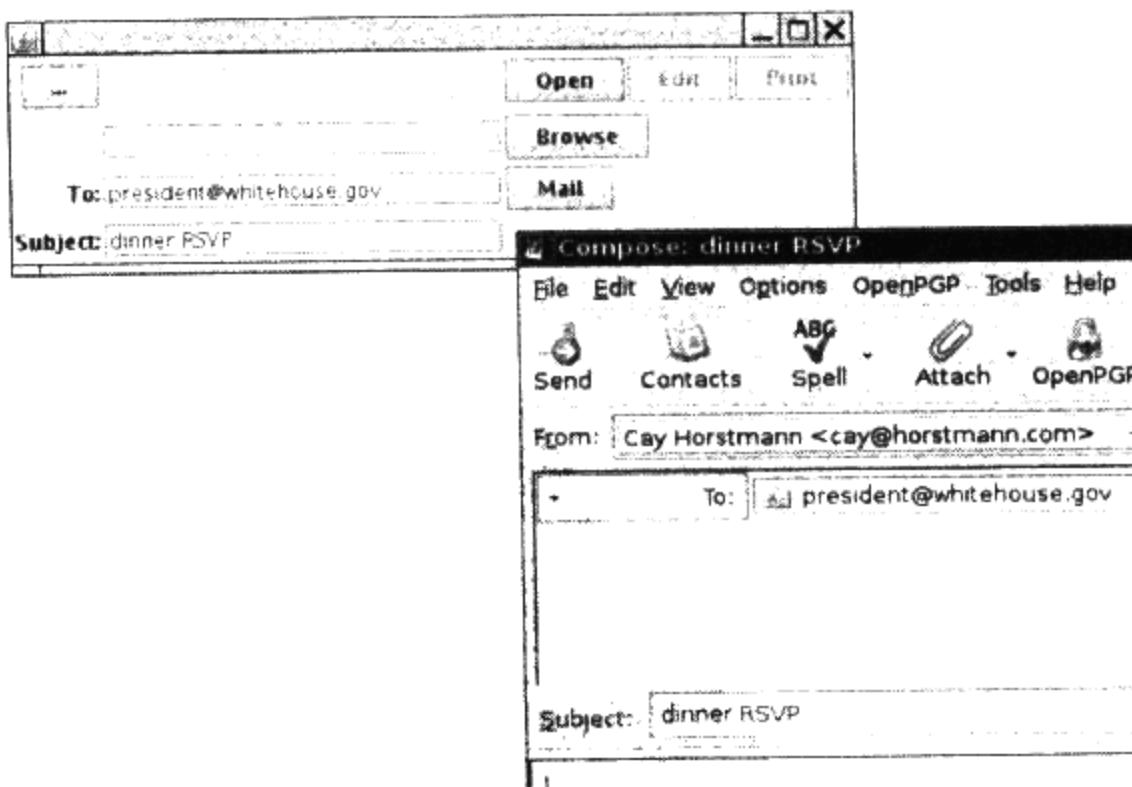


图7-48 启动一个桌面应用程序

### 程序清单7-17 DesktopAppTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
```

```
3. import java.io.*;
4. import java.net.*;
5. import javax.swing.*;
6.
7. /**
8. * This program demonstrates the desktop app API.
9. * @version 1.00 2007-09-22
10. * @author Cay Horstmann
11. */
12. public class DesktopAppTest
13. {
14.     public static void main(String[] args)
15.     {
16.         EventQueue.invokeLater(new Runnable()
17.         {
18.             public void run()
19.             {
20.                 JFrame frame = new DesktopAppFrame();
21.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22.                 frame.setVisible(true);
23.             }
24.         });
25.     }
26. }
27.
28. class DesktopAppFrame extends JFrame
29. {
30.     public DesktopAppFrame()
31.     {
32.         setLayout(new GridBagLayout());
33.         final JFileChooser chooser = new JFileChooser();
34.         JButton fileChooserButton = new JButton(...);
35.         final JTextField fileField = new JTextField(20);
36.         fileField.setEditable(false);
37.         JButton openButton = new JButton("Open");
38.         JButton editButton = new JButton("Edit");
39.         JButton printButton = new JButton("Print");
40.         final JTextField browseField = new JTextField();
41.         JButton browseButton = new JButton("Browse");
42.         final JTextField toField = new JTextField();
43.         final JTextField subjectField = new JTextField();
44.         JButton mailButton = new JButton("Mail");
45.
46.         openButton.setEnabled(false);
47.         editButton.setEnabled(false);
48.         printButton.setEnabled(false);
49.         browseButton.setEnabled(false);
50.         mailButton.setEnabled(false);
51.
52.         if (Desktop.isDesktopSupported())
53.         {
54.             Desktop desktop = Desktop.getDesktop();
55.             if (desktop.isSupported(Desktop.Action.OPEN)) openButton.setEnabled(true);
56.             if (desktop.isSupported(Desktop.Action.EDIT)) editButton.setEnabled(true);
57.             if (desktop.isSupported(Desktop.Action.PRINT)) printButton.setEnabled(true);
58.             if (desktop.isSupported(Desktop.Action.BROWSE)) browseButton.setEnabled(true);
59.             if (desktop.isSupported(Desktop.Action.MAIL)) mailButton.setEnabled(true);
```

```
60.    }
61.
62.    fileChooserButton.addActionListener(new ActionListener()
63.    {
64.        public void actionPerformed(ActionEvent e)
65.        {
66.            if (chooser.showOpenDialog(DesktopAppFrame.this) ==
67.                JFileChooser.APPROVE_OPTION)
68.                fileField.setText(chooser.getSelectedFile().getAbsolutePath());
69.        }
70.    });
71.
72.    openButton.addActionListener(new ActionListener()
73.    {
74.        public void actionPerformed(ActionEvent e)
75.        {
76.            try
77.            {
78.                Desktop.getDesktop().open(chooser.getSelectedFile());
79.            }
80.            catch (IOException ex)
81.            {
82.                ex.printStackTrace();
83.            }
84.        }
85.    });
86.
87.    editButton.addActionListener(new ActionListener()
88.    {
89.        public void actionPerformed(ActionEvent e)
90.        {
91.            try
92.            {
93.                Desktop.getDesktop().edit(chooser.getSelectedFile());
94.            }
95.            catch (IOException ex)
96.            {
97.                ex.printStackTrace();
98.            }
99.        }
100.    });
101.
102.    printButton.addActionListener(new ActionListener()
103.    {
104.        public void actionPerformed(ActionEvent e)
105.        {
106.            try
107.            {
108.                Desktop.getDesktop().print(chooser.getSelectedFile());
109.            }
110.            catch (IOException ex)
111.            {
112.                ex.printStackTrace();
113.            }
114.        }
115.    });
116.
```

```
116.  
117    browseButton.addActionListener(new ActionListener()  
118    {  
119        public void actionPerformed(ActionEvent e)  
120        {  
121            try  
122            {  
123                Desktop.getDesktop().browse(new URI(browseField.getText()));  
124            }  
125            catch (URISyntaxException ex)  
126            {  
127                ex.printStackTrace();  
128            }  
129            catch (IOException ex)  
130            {  
131                ex.printStackTrace();  
132            }  
133        }  
134    });  
135.  
136    mailButton.addActionListener(new ActionListener()  
137    {  
138        public void actionPerformed(ActionEvent e)  
139        {  
140            try  
141            {  
142                String subject = percentEncode(subjectField.getText());  
143                URI uri = new URI("mailto:" + toField.getText() + "?subject=" + subject);  
144.  
145                System.out.println(uri);  
146                Desktop.getDesktop().mail(uri);  
147            }  
148            catch (URISyntaxException ex)  
149            {  
150                ex.printStackTrace();  
151            }  
152            catch (IOException ex)  
153            {  
154                ex.printStackTrace();  
155            }  
156        }  
157    });  
158.  
159    JPanel buttonPanel = new JPanel();  
160    ((FlowLayout) buttonPanel.getLayout()).setHgap(2);  
161    buttonPanel.add(openButton);  
162    buttonPanel.add(editButton);  
163    buttonPanel.add(printButton);  
164.  
165    add(fileChooserButton, new GBC(0, 0).setAnchor(GBC.EAST).setInsets(2));  
166    add(fileField, new GBC(1, 0).setFill(GBC.HORIZONTAL));  
167    add(buttonPanel, new GBC(2, 0).setAnchor(GBC.WEST).setInsets(0));  
168    add(browseField, new GBC(1, 1).setFill(GBC.HORIZONTAL));  
169    add(browseButton, new GBC(2, 1).setAnchor(GBC.WEST).setInsets(2));  
170    add(new JLabel("To:"), new GBC(0, 2).setAnchor(GBC.EAST).setInsets(5, 2, 5, 2));  
171    add(toField, new GBC(1, 2).setFill(GBC.HORIZONTAL));  
172    add(mailButton, new GBC(2, 2).setAnchor(GBC.WEST).setInsets(2));
```

```
173     add(new JLabel("Subject:"), new GBC(0, 3).setAnchor(GBC.EAST).setInsets(5, 2, 5, 2));
174     add(subjectField, new GBC(1, 3).setFill(GBC.HORIZONTAL));
175
176     pack();
177 }
178
179 private static String percentEncode(String s)
180 {
181     try
182     {
183         return URLEncoder.encode(s, "UTF-8").replaceAll("[+]", "%20");
184     }
185     catch (UnsupportedEncodingException ex)
186     {
187         return null; // UTF-8 is always supported
188     }
189 }
190 }
```



### java.awt.Desktop 6

- **static boolean isDesktopSupported()**

如果该平台支持启动桌面应用程序，则返回true。

- **static Desktop getDesktop()**

返回用于启动桌面应用程序的Desktop对象。如果该平台不支持启动桌面操作，则抛出UnsupportedOperationException异常。

- **boolean isSupported(Desktop.Action action)**

如果支持给定的动作，则返回true。action是OPEN、EDIT、PRINT、BROWSE或MAIL之一。

- **void open(File file)**

启动注册为浏览给定文件的应用程序。

- **void edit(File file)**

启动注册为编辑给定文件的应用程序。

- **void print(File file)**

打印给定文件。

- **void browse(URI uri)**

用给定的URI启动默认浏览器。

- **void mail()**

- **void mail(URI uri)**

启动默认邮件程序。第二个版本可以用来填充e-mail消息的部分内容。

#### 7.15.3 系统托盘

许多桌面环境都有一个区域用于放置在后台运行的程序的图标，这些程序偶尔会将某些事件通知给用户。在Windows中，这个区域称为系统托盘，而这些图标称为托盘图标。Java API采纳了相同的术语命名规则。有一个这种程序的典型实例，那就是检查软件更新的监视器。如

如果有新的更新，监视器程序可以改变其图标的外观，并在图表附近显示一条消息。

坦白地讲，用户对系统托盘的应用过多。当他们发现又添加了新的托盘图标时，通常都会感到不痛快。我们的系统托盘应用程序示例也逃脱不了这条规则，这个程序可以分发虚拟的“幸福饼”。

`java.awt.SystemTray`类是跨平台的通向系统托盘的渠道，与前面讨论过的`Desktop`类相类似，首先要调用静态的`isSupported`方法来检查本地Java平台是否支持系统托盘。如果支持，则通过调用静态的`getSystemTray`方法来获取`SystemTray`的单例。

`SystemTray`类最重要的方法是`add`方法，它使得我们可以添加一个`TrayIcon`实例。托盘图标有三个主要的属性：

- 图标的图像。
- 当鼠标滑过图标时显示的工具提示。
- 当用户用鼠标右键点击图标时显示的弹出式菜单。

弹出式菜单是AWT类库中的`PopupMenu`类的一个实例，表示本地的弹出式菜单，而不是Swing菜单。可以在其中添加AWT的`MenuItem`实例，而这些实例每个都有一个动作监听器，就像Swing中的菜单项一样。

最后，托盘图标可以向用户显示通知信息（参见图7-49），这需要调用`TrayIcon`类的`displayMessage`方法，并指定标题、消息和消息类型。

```
trayIcon.displayMessage("Your Fortune", fortunes.get(index), TrayIcon.MessageType.INFO);
```



图7-49 从托盘图标中发出的通知

### 程序清单7-18 SystemTrayTest.java

```
1. import java.awt.*;
2. import java.util.*;
3. import java.util.List;
4. import java.awt.event.*;
5. import java.io.*;
6. import javax.swing.Timer;
```

```
7.  
8. /**  
9. * This program demonstrates the system tray API.  
10. * @version 1.00 2007-09-22  
11. * @author Cay Horstmann  
12. */  
13. public class SystemTrayTest  
14. {  
15.     public static void main(String[] args)  
16.     {  
17.         final TrayIcon trayIcon;  
18.  
19.         if (!SystemTray.isSupported())  
20.         {  
21.             System.err.println("System tray is not supported.");  
22.             return;  
23.         }  
24.  
25.         SystemTray tray = SystemTray.getSystemTray();  
26.         Image image = Toolkit.getDefaultToolkit().getImage("cookie.png");  
27.  
28.         PopupMenu popup = new PopupMenu();  
29.         MenuItem exitItem = new MenuItem("Exit");  
30.         exitItem.addActionListener(new ActionListener()  
31.         {  
32.             public void actionPerformed(ActionEvent e)  
33.             {  
34.                 System.exit(0);  
35.             }  
36.         });  
37.         popup.add(exitItem);  
38.  
39.         trayIcon = new TrayIcon(image, "Your Fortune", popup);  
40.  
41.         trayIcon.setImageAutoSize(true);  
42.         trayIcon.addActionListener(new ActionListener()  
43.         {  
44.             public void actionPerformed(ActionEvent e)  
45.             {  
46.                 trayIcon.displayMessage("How do I turn this off?",  
47.                     "Right-click on the fortune cookie and select Exit.",  
48.                     TrayIcon.MessageType.INFO);  
49.             }  
50.         });  
51.  
52.         try  
53.         {  
54.             tray.add(trayIcon);  
55.         }  
56.         catch (AWTException e)  
57.         {  
58.             System.err.println("TrayIcon could not be added.");  
59.             return;  
60.         }  
61.  
62.         final List<String> fortunes = readFortunes();  
63.         Timer timer = new Timer(10000, new ActionListener()
```

```
64.     {
65.         public void actionPerformed(ActionEvent e)
66.         {
67.             int index = (int) (fortunes.size() * Math.random());
68.             trayIcon.displayMessage("Your Fortune", fortunes.get(index),
69.                                     TrayIcon.MessageType.INFO);
70.         }
71.     });
72.     timer.start();
73. }
74.
75. private static List<String> readFortunes()
76. {
77.     List<String> fortunes = new ArrayList<String>();
78.     try
79.     {
80.         Scanner in = new Scanner(new File("fortunes"));
81.         StringBuilder fortune = new StringBuilder();
82.         while (in.hasNextLine())
83.         {
84.             String line = in.nextLine();
85.             if (line.equals("%"))
86.             {
87.                 fortunes.add(fortune.toString());
88.                 fortune = new StringBuilder();
89.             }
90.             else
91.             {
92.                 fortune.append(line);
93.                 fortune.append(' ');
94.             }
95.         }
96.     }
97.     catch (IOException ex)
98.     {
99.         ex.printStackTrace();
100.    }
101.    return fortunes;
102. }
103.
104. }
```



### java.awt.SystemTray 6

- **static boolean isSupported()**

如果这个平台支持对系统托盘的访问，则返回true。

- **static SystemTray getSystemTray()**

返回用于访问系统托盘的SystemTray对象。如果这个平台不支持对系统托盘的访问，则抛出UnsupportedOperationException异常。

- **Dimension getTrayIconSize()**

获取系统托盘中的图标的尺寸。

- **void add(TrayIcon trayIcon)**

- void remove(TrayIcon trayIcon)

添加或移除一个系统托盘图标。



### java.awt.TrayIcon 6

- TrayIcon(Image image)

- TrayIcon(Image image, String tooltip)

- TrayIcon(Image image, String tooltip, PopupMenu popupMenu)

用给定的图像、工具提示和弹出式菜单构建一个托盘图标。

- Image getImage()

- void setImage(Image image)

- String getTooltip()

- void setTooltip(String tooltip)

- PopupMenu getPopupMenu()

- void setPopupMenu(PopupMenu popupMenu)

获取或设置图像、工具提示，或该工具提示的弹出式菜单

- boolean isImageAutoSize()

- void setImageAutoSize(boolean autosize)

获取或设置imageAutoSize属性，如果设置了，那么图像就会缩放到适合工具提示图标区的大小。如果没有设置（默认值），那么图像就会被截除（如果图像太大），或者居中（如果图像太小）。

- void displayMessage(String caption, String text, TrayIcon.MessageType messageType)

在托盘图标附近显示消息。消息的类型为INFO、WARNING、ERROR或NONE

- public void addActionListener(ActionListener listener)

- public void removeActionListener(ActionListener listener)

如果被调用的监听器是平台依赖的，则添加和移除动作监听器。典型情况是在通知信息上点击或在托盘图标上双击。

现在，我们来到了本章的尾声，这长长的一章涵盖了高级AWT特性。在下一章，我们将讨论JavaBeans规范和它在GUI构建器中的用处。

# 第8章 JavaBean构件

- ▲ 为何使用Bean
- ▲ 编写Bean的过程
- ▲ 使用Bean来构造一个应用
- ▲ Bean属性与事件的命名模式
- ▲ Bean属性的类型
- ▲ Beaninfo类
- ▲ 属性编辑器
- ▲ 定制器
- ▲ Javabean持久化

在JavaBean规范中有bean的官方定义：“根据Sun的JavaBean规范，一个bean就是一个可重用的软件构件，并且能够在开发工具中可视化地操作。”

一旦实现了一个bean，其他人就能够在开发环境中使用它（例如NetBeans）。与必须编写冗长的代码不同，开发人员可以直接将bean拖曳到GUI表单中，并用对话框来定制它。

本章将解释如何实现bean，以便让其他的开发人员可以方便地使用它们。

 **注意：**在进入正题之前，我们先要指出一个常见的误解：本章讨论的JavaBean与Enterprise JavaBean (EJB) 没什么关系。企业JavaBeans是服务器端的构件，提供对事务、持久化、复制以及安全问题的支持。从最基本的角度来看，它们都是能够在开发环境中使用的构件。然而，企业JavaBeans技术比“标准版”JavaBean技术要复杂得多。

这并不意味着标准的JavaBean构件就仅限于客户端编程。Web技术就很依赖于JavaBean构件模型，例如JavaServer Faces (JSF) 和JavaServer Pages (JSP)。

## 8.1 为何使用Bean

有Visual Basic经验的程序员立刻就会明白，为什么bean如此重要。而那些来自“凡事都要亲自动手”的环境中的程序员通常很难相信Visual Basic是可重用对象技术最成功的例子之一。如果回想一下你是如何开发一个Visual Basic应用的，那么Visual Basic如此受欢迎的原因就会很清楚了。对于那些从未使用过Visual Basic的人，下面是一个简单的指导，教你如何构建Visual Basic应用：

- 1) 通过将构件（在Visual Basic中称为控件）拖拽到表单窗口上来开发界面。
- 2) 通过属性表单设置构件的属性，例如高度、颜色或其他行为。
- 3) 属性监视器还列出了构件能够应对的事件。某些事件可以通过对话框捕获处理，而对于其他事件，你需要编写简短的事件处理代码。

举例来说，在本书卷I第2章中，我们编写了一个程序，它在一个窗体中显示一幅图像。那需要超过一页的代码。而在Visual Basic中创建类似功能的程序，只需：

- 1) 添加两个控件到一个窗体中：一个Image控件用来显示图片，以及一个Common Dialog控件用来选择文件。
- 2) 设置CommonDialog控件的Filter属性，使它只显示能够处理的图像文件，如图8-1所示。

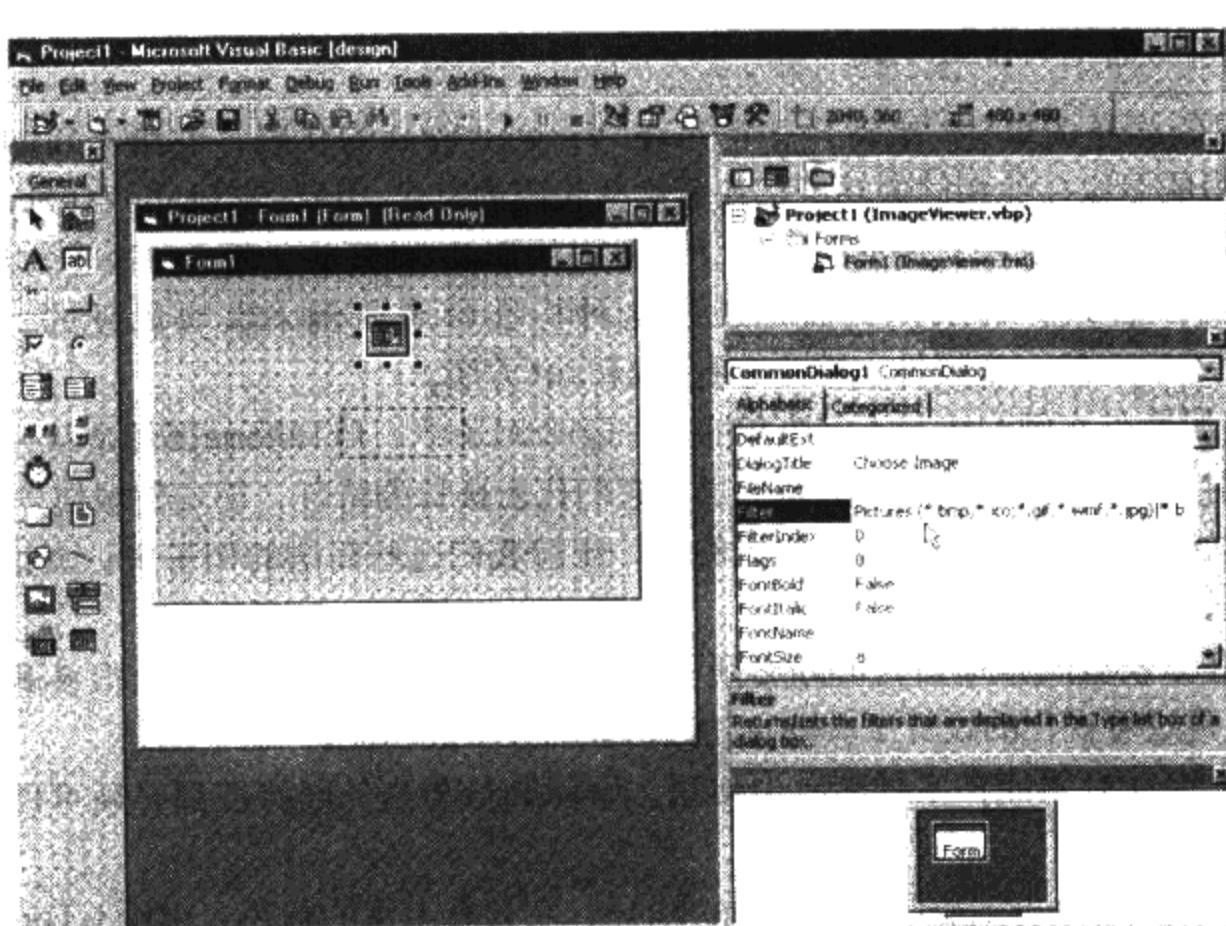


图8-1 Visual Basic中一个图像应用程序的属性窗口

- 3) 编写四行Visual Basic代码，它将在该工程首次运行的时候被激活。需要的全部代码如下所示：

```
Private Sub Form_Load()
    CommonDialog1.ShowOpen
    Image1.Picture = LoadPicture(CommonDialog1.FileName)
End Sub
```

上面的代码会弹出一个文件对话框，而且只有具备正确扩展名的文件会显示出来，因为我们设置了过滤器属性。在用户选择了一个图像文件之后，该代码会告诉图像控件将选择的图像显示出来。

仅此而已。进行一些布局操作，再添加这些语句，就完成了整整一页Java代码才能完成的功能。很明显，比起编写整整一页的代码，学习如何拖拽构件与设置属性可简单多了。

我们并不认为Visual Basic对于所有的问题都是最好的解决办法。但是对于UI驱动的那一类Windows问题，它确实是最佳选择。JavaBean技术就是为了使Java技术在该领域也具备竞争力。它促使供应商提供Visual Basic风格的开发环境。这些环境使得以最小的编程量来开发Java用户界面成为可能。

## 8.2 编写Bean的过程

编写bean并不是什么困难的技术，只不过要掌握一些新的类与接口。特别是，最简单的

bean与一个Java类没什么任何区别，只不过其方法采用了某种严格的命名惯例罢了。

 **注意：**有些作者说bean必须有一个默认的构造器。JavaBean规范并没有提到这一点。但是，大多数开发工具要求每一个bean都具备一个默认的构造器，这样他们才能在没有构造参数的情况下初始化bean。

本节末尾的程序清单8-1展示了一个ImageViewer bean的代码，它给予Java开发环境与Visual Basic中的图像控件相同的功能。在看这些代码的时候，请注意ImageViewerBean类，它看起来与别的类没什么区别。例如所有的访问器方法都以get开头，所有修改器方法都以set开头。很快你就会看到，开发工具就是使用这种标准命名惯例来发现属性的。举个例子，fileName是这个bean的一个属性，因为它有get和set方法。

请注意，属性与实例的域并不完全相同。在这个例子中，fileName属性是从file实例域计算得来的。属性在概念上比实例域更高一层，它们代表接口的特征，而实例域属于类的实现。

当你阅读这一章的示例时，需要牢记的是，比起我们这些简洁的例子来，真正的bean要精巧且冗长得多，这有两个原因。

- 1) 对于非专家级的程序员，bean必须便于使用。因此，你需要暴露出很多属性，以便用户无需编程，通过可视化的设计工具即可访问bean的大多数功能。

- 2) 同一个bean必须能在多种环境中使用。bean的行为与外部特征都必须可订制。这又要求暴露大量的属性。

有一个非常好的bean的例子：CalendarBean，它具有丰富的行为，由Kai Tödter开发（参见图8-2）。这个bean与其源代码都是可以从<http://www.toedter.com/en/jcalendar>免费得到的。这

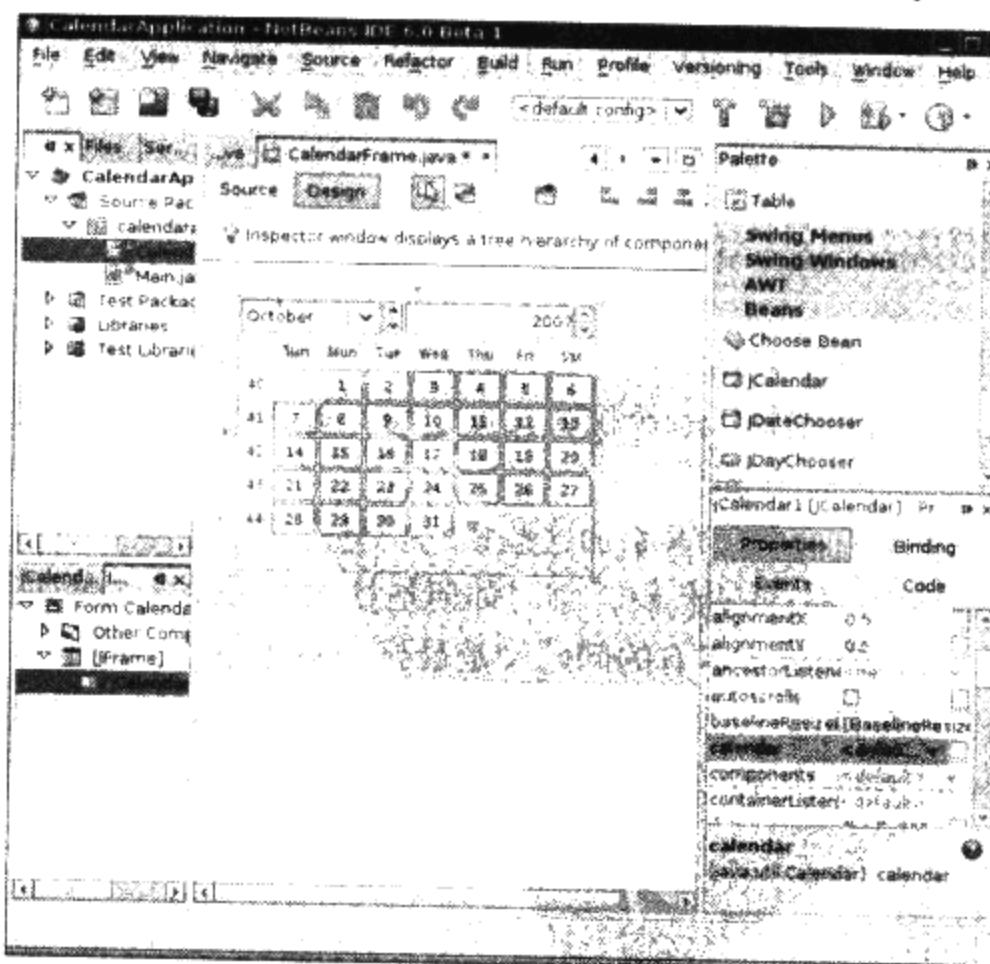


图8-2 一个日历bean

个bean使用户能够方便地输入日期，只需直接从显示的日历中选择日期即可。这项工作显然相当复杂，并不是人人都想从头开始编程实现这个东西。而使用这样的一个bean，直接拖拽该bean到开发工具中，就能够利用其他人的成果了。

幸运的是，要编写一个功能丰富的bean，你必须掌握的概念并不多。这一章的示例，虽然并不简单，但是我们已经尽量保持其简单，使其能够说明必要的概念即可。

### 程序清单8-1 ImageViewerBean.java

```
1. package com.horstmann.corejava;
2.
3. import java.awt.*;
4. import java.io.*;
5. import javax.imageio.*;
6. import javax.swing.*;
7.
8. /**
9. * A bean for viewing an image.
10. * @version 1.21 2001-08-15
11. * @author Cay Horstmann
12. */
13. public class ImageViewerBean extends JLabel
14. {
15.
16.     public ImageViewerBean()
17.     {
18.         setBorder(BorderFactory.createEtchedBorder());
19.     }
20.
21. /**
22. * Sets the fileName property.
23. * @param fileName the image file name
24. */
25. public void setFileName(String fileName)
26. {
27.     try
28.     {
29.         file = new File(fileName);
30.         setIcon(new ImageIcon(ImageIO.read(file)));
31.     }
32.     catch (IOException e)
33.     {
34.         file = null;
35.         setIcon(null);
36.     }
37. }
38.
39. /**
40. * Gets the fileName property.
41. * @return the image file name
42. */
43. public String getFileName()
44. {
45.     if (file == null) return "";
46.     else return file.getPath();
47. }
```

```

48.
49.     public Dimension getPreferredSize()
50.     {
51.         return new Dimension(XPREFSIZE, YPREFSIZE);
52.     }
53.
54.     private File file = null;
55.     private static final int XPREFSIZE = 200;
56.     private static final int YPREFSIZE = 200;
57. }
```

## 8.3 使用Bean构造应用程序

在我们深入到编写bean的机制之前，我希望读者先看一看如何使用或者测试bean。ImageViewerBean是一个非常实用的bean，但是在开发环境之外，它无法展示出自己独特的特点。

每个开发环境都有自己的一套策略，使程序员的生活更轻松。我们用到的环境是NetBeans，它集成了开发环境，可从<http://netbeans.org>获得。

在这个例子中，我们用到了两个bean，ImageViewerBean和FileNameBean。你已经看到了ImageViewerBean的代码。本章稍后我们将分析FileNameBean的代码。现在，你只需知道点击具有“...”标签的按钮就会出现打开一个文件选择器。

### 8.3.1 将Bean打包成JAR文件

为了使bean能够在开发工具中使用，需要将bean以及它使用到的所有类文件都打包进一个JAR文件。与JAR文件不同，bean的JAR文件需要一个manifest清单文件，用以详细说明档案中的哪些类文件是bean，需要被包括到工具箱中。例如，下面是一个针对ImageViewerBean的清单文件ImageViewerBean.mf。

Manifest-Version: 1.0

Name: com/horstmann/corejava/ImageViewerBean.class  
Java-Bean: True

注意，清单版本与bean名字之间有一个空行。

 **注意：**我们之所以将示例bean放入com.horstmann.corejava包中，是因为某些开发环境从默认包加载bean时会有问题。

如果你的bean包含多个类文件，只需在清单中指出哪些类文件是bean并且希望显示在工具箱中。例如，可以将ImageViewerBean和FileNameBean放入同一个JAR文件中，然后使用如下清单：

Manifest-Version: 1.0

Name: com/horstmann/corejava/ImageViewerBean.class  
Java-Bean: True

Name: com/horstmann/corejava/FileNameBean.class  
Java-Bean: True

 **警告：**某些开发工具对于清单很挑剔。必须确保每一行后面都没有空格，版本与bean条目之间要有空行，而且最后以一个新行结束。

制作JAR文件，可按以下步骤进行：

- 1) 编辑清单文件。
- 2) 在一个目录中聚齐所有需要的类文件。
- 3) 如下运行jar工具：

```
jar cvfm JarFile ManifestFile ClassFiles
```

例如

```
jar cvfm ImageViewerBean.jar ImageViewerBean.mf com/horstmann/corejava/*.class
```

还可以向JAR文件中添加其他东西，例如图标的图像。我们稍候会讨论bean的图标问题。

**X** **警告：**要确保你的bean所需的所有文件都在JAR文件中。特别要注意内部类，例如FileNameBean\$1.class。

开发环境具有添加新bean的机制，通常是通过加载JAR文件实现的。下面是向NetBeans(版本6)导入bean的过程。

编译ImageViewerBean和FileNameBean类，将它们打包成JAR文件。然后启动NetBeans并按以下步骤进行。

- 1) 从菜单中选择Tools -> Palette -> Swing/AWT Component。
- 2) 点击Add from JAR按钮。
- 3) 在文件对话框中，转到ImageViewerBean目录并选择ImageViewerBean.jar。
- 4) 这时会弹出一个对话框，列出了在JAR文件中发现的所有bean。选择ImageViewerBean。
- 5) 最后，它会问你想把这个bean放到哪一个选项板中。请选择Bean。(还有其他一些用于Swing构件、AWT构件以及其他构件的选项板。)
- 6) 检查一下Bean选项板，它现在包含了一个用图标表示的新bean。但是，该图标只是默认图标，不过稍后你会看到如何为bean添加图标。

对FileNameBean重复以上步骤。现在你就可以将这些bean组合到应用程序中了。

### 8.3.2 在开发环境中组合Bean

基于构件开发的方法允许我们用预制的构件经过最少量的编程来组成应用程序。在本节中，你将看见如何用ImageViewerBean和FileNameBean构件来组成应用程序。

在NetBeans中，请从菜单中选择File→New Project，这时会弹出一个对话框。然后选择Java，之后是Java Application(参见图8-3)。

点击Next按钮。在下面的屏幕中，为你的应用程序设置一个名字(例如ImageViewer)，然后点击Finish按钮。现在，可以看到左边是一个工程视图，中间是源代码编辑器。

鼠标右击工程视图中的工程名，从菜单中选择New→JFrame Form(参见图8-4)。

这时会弹出一个对话框。键入frame类的名字(例如ImageViewerFrame)，然后点击Finish按钮。然后你就得到了一个表单编辑器，它带有一个空白的窗体。向要想在该表单中添加一个bean，可以从位于表单编辑器右侧的选项板中选择bean，然后点击框架。

图8-5展示了将一个ImageViewerBean添加到窗体上的效果。

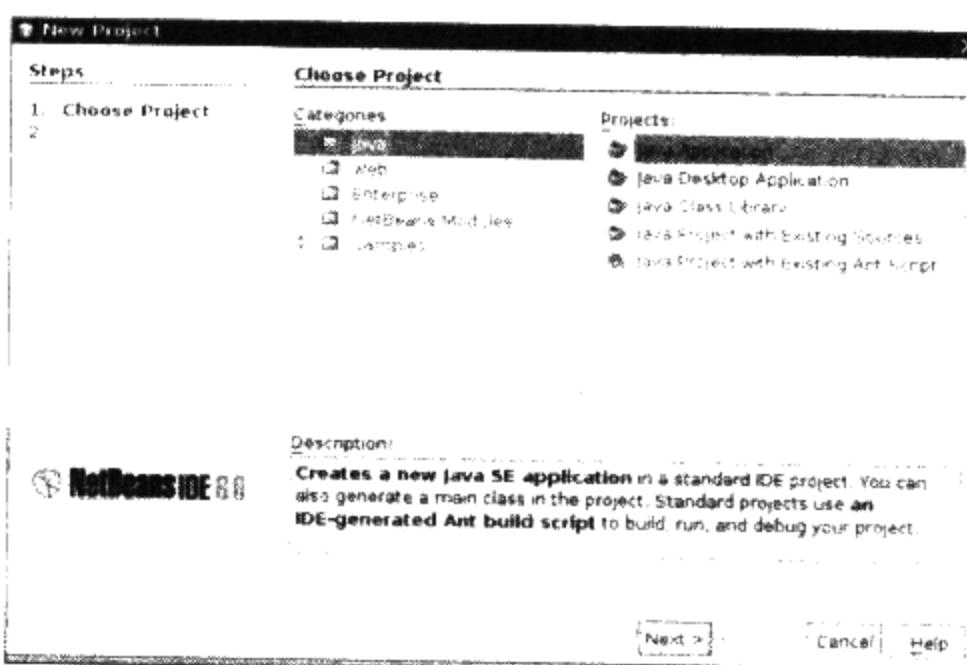


图8-3 创建一个新工程

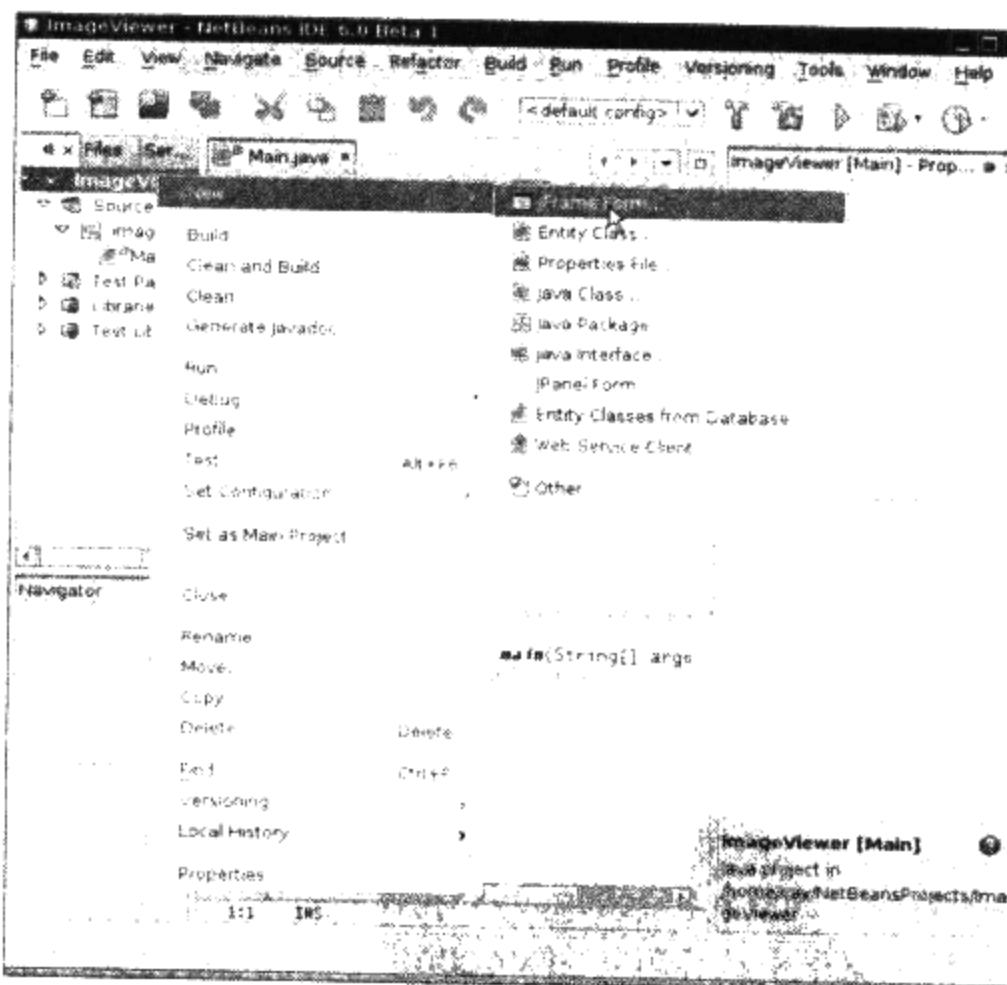


图8-4 创建一个表单视图

如果看一看代码编辑器窗口，就会发现源代码中包括了将bean对象添加到窗体中的Java指令（参见图8-6）。源代码被可怕的警告括了起来，说明你不应该编辑它。在你修改表单时，如果开发环境更新了代码，那么任何的编辑工作都可能会丢失。

 **注意：**在构建应用程序的时候，并不要求开发环境去更新源代码。开发环境可能在你进行编辑的时候生成源代码，将你定制的bean序列化，或者可能产生一个与你的构建活动完全不同的描述。

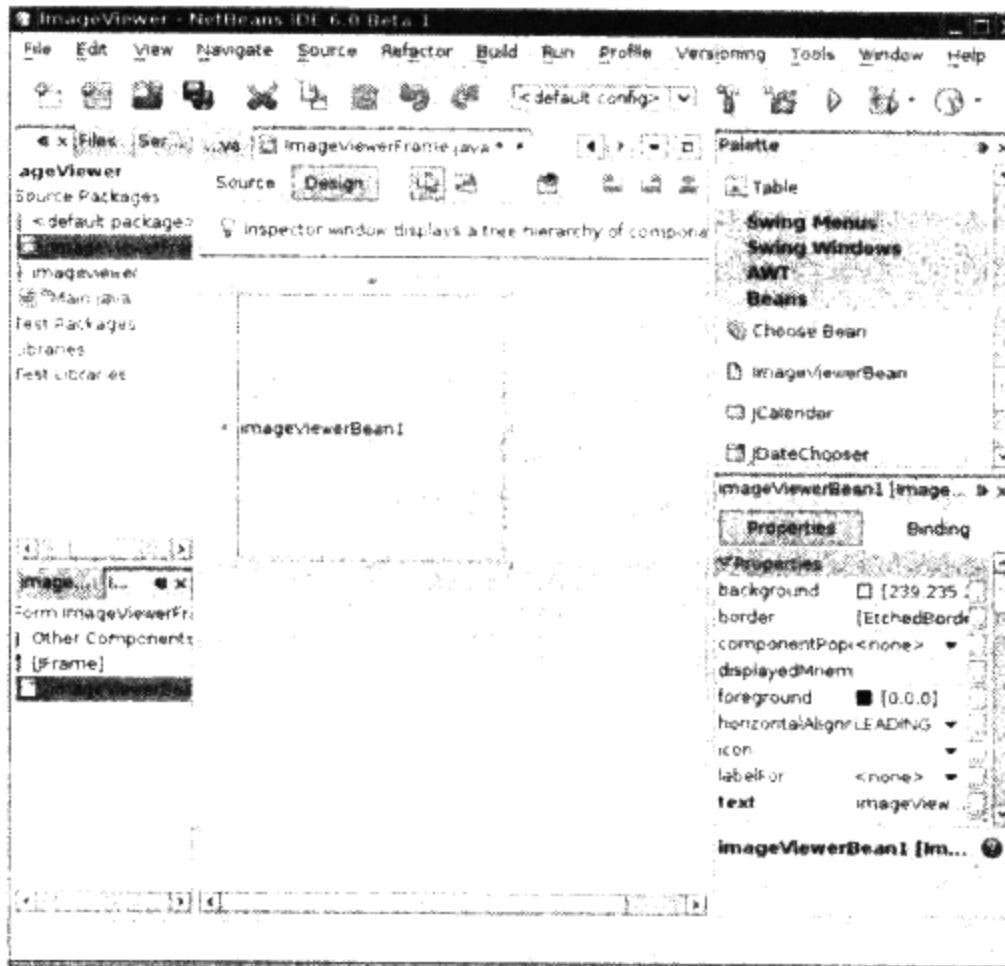


图8-5 添加一个bean

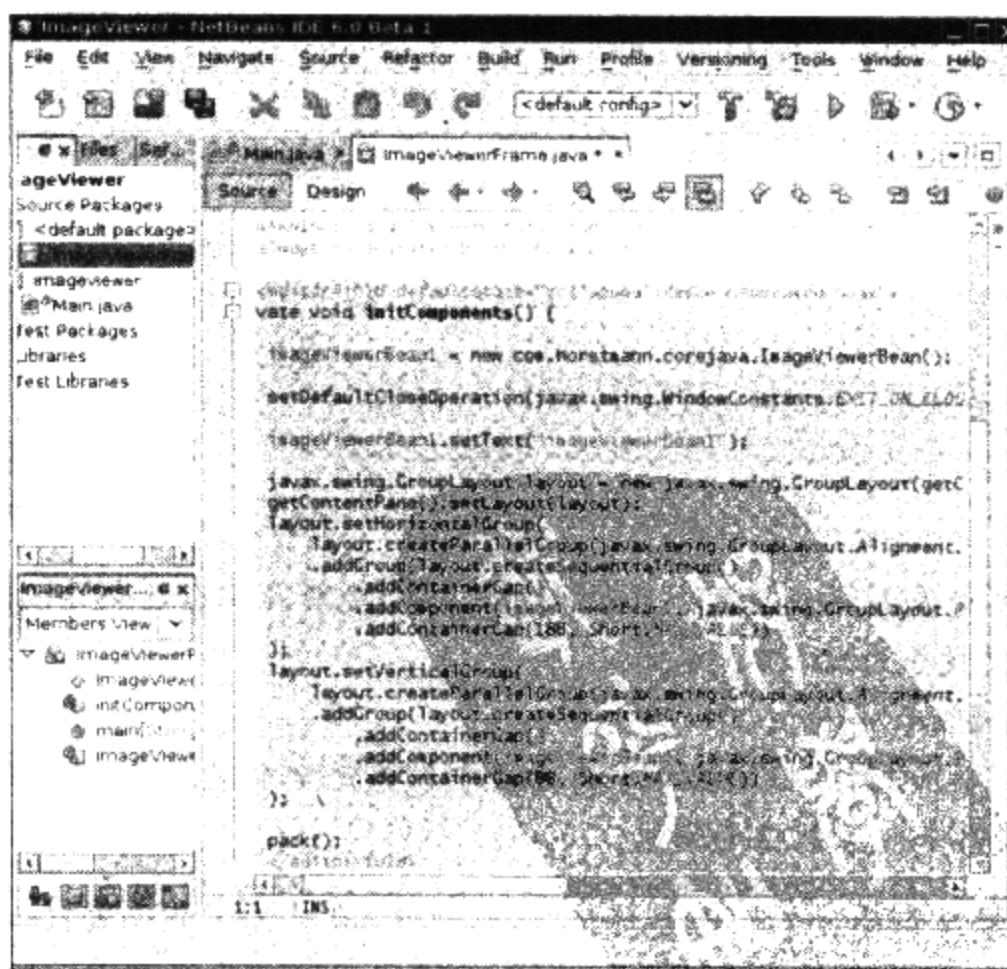


图8-6 为添加bean而生成的源代码

例如，<http://bean-builder.dev.java.net>上实验性的Bean Builder就允许你设计GUI应用程序，

而无需编写任何代码。

JavaBean机制并没有对开发工具强加什么实现策略。它旨在向开发工具提供bean的信息，开发工具可以选择按自己的方式使用这些信息。

现在让我们回到表单视图，点击表单中的ImageViewerBean。右手边是一个属性检查器，列出了bean各项属性的名字与当前值。这是基于构件的开发工具的核心部分，因为在设计阶段设置属性就是在设置一个构件的初始状态。

例如，可以修改图像bean所使用的标签的text属性，而这只需在属性检查器中键入一个新的名字即可。改变text属性很简单，只需在文本区中键入一个字符串。试着将标签文本改为“Hello”，以验证一下表单是否会立刻更新以反映你的修改（参见图8-7）。

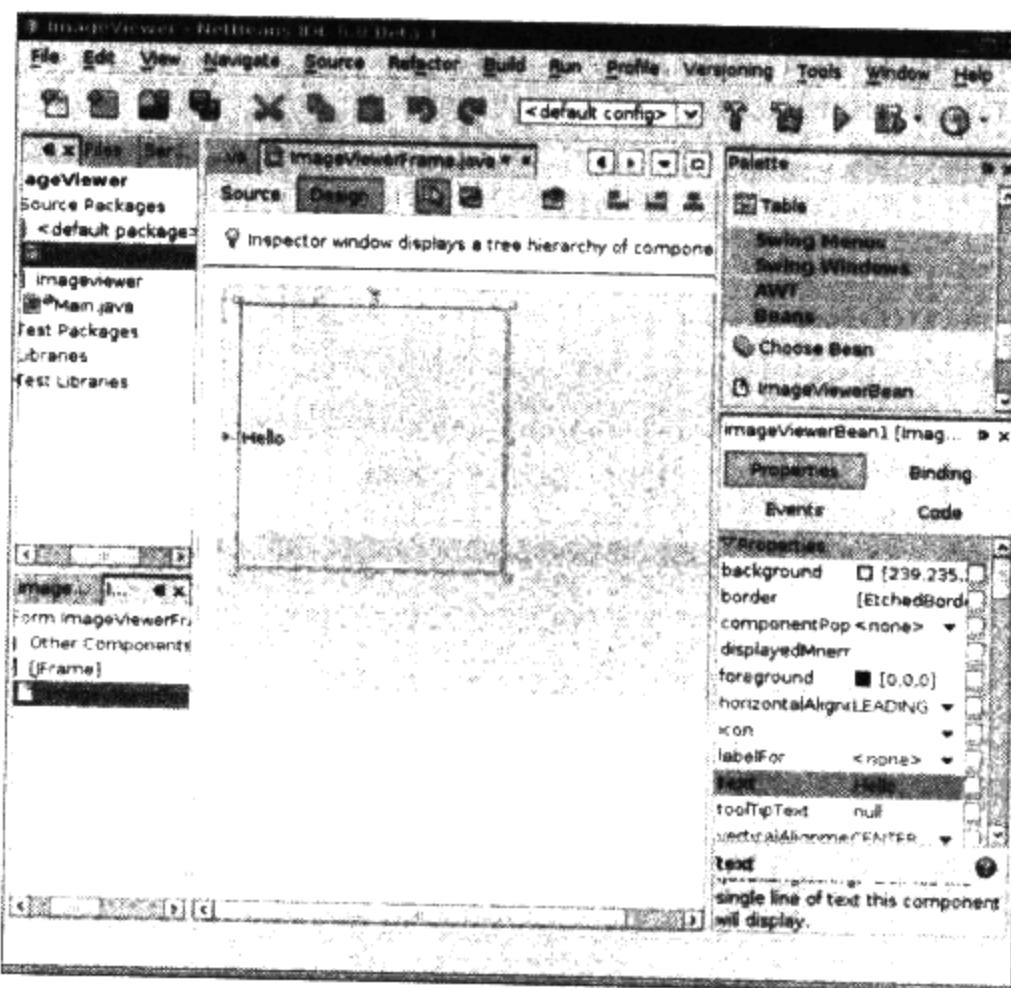


图8-7 在属性检查器中改变一个属性



**注意：**当改变某个属性的设置时，NetBeans环境会更新源码以反映你的动作。举例来说，如果将text字段设置为Hello，那么

```
imageViewBean.setText("Hello");
```

这条语句就会被添加到initComponents方法中。我们曾经提到过了，其他的开发工具可能采用不同的策略来记录属性的设置。

属性不一定非要是字符串，它可以是任何Java类型的值。为了使设置任何类型的属性成为可能，开发工具采用了专门的属性编辑器。（属性编辑器或者来自于开发工具，或者由bean的开发者提供。在本章中你会看到如何开发自己的属性编辑器。）

下面我们以foreground属性为例，来看看一个简单的属性编辑器是如何工作的。该属性的

类型为Color，你可以看到一个颜色编辑器，它带有一个包含[0,0,0]字符串的文本域，以及一个标记为“...”的按钮，它可以启动一个颜色选择器。接下来试一下改变前景颜色。注意，你立刻就能看到属性值的变化：标签的文本颜色改变了。

还有更有趣的，试一试在属性检查器中为图像文件这个属性选择一个文件名，一旦这么做了，ImageViewerBean就会自动显示该图像。

注意：如果仔细看一看NetBeans中的属性检查器，就会发现有些奇怪的属性具有一个很大的数字，例如focusCycleRoot和iconTextGap。它们继承自父类JLabel。后面将会看到如何在属性检查器中隐藏它们。

接下来继续开发我们的应用程序，将一个FileNameBean对象放在窗体的南端。现在，我们希望当FileNameBean的fileName属性发生变化时，图像即被加载。这可以通过一个PropertyChangeEvent事件来实现，我们稍候会讨论此类事件。

想要针对该事件做出反应，请选择FileNameBean，然后在属性检查器中选择Events标签。然后点击propertyChange旁边的“...”按钮，这时会出现一个对话框，说明该事件还没有相关的处理操作。点击该对话框的Add按钮，要求填入一个方法名（参见图8-8），输入loadImage。

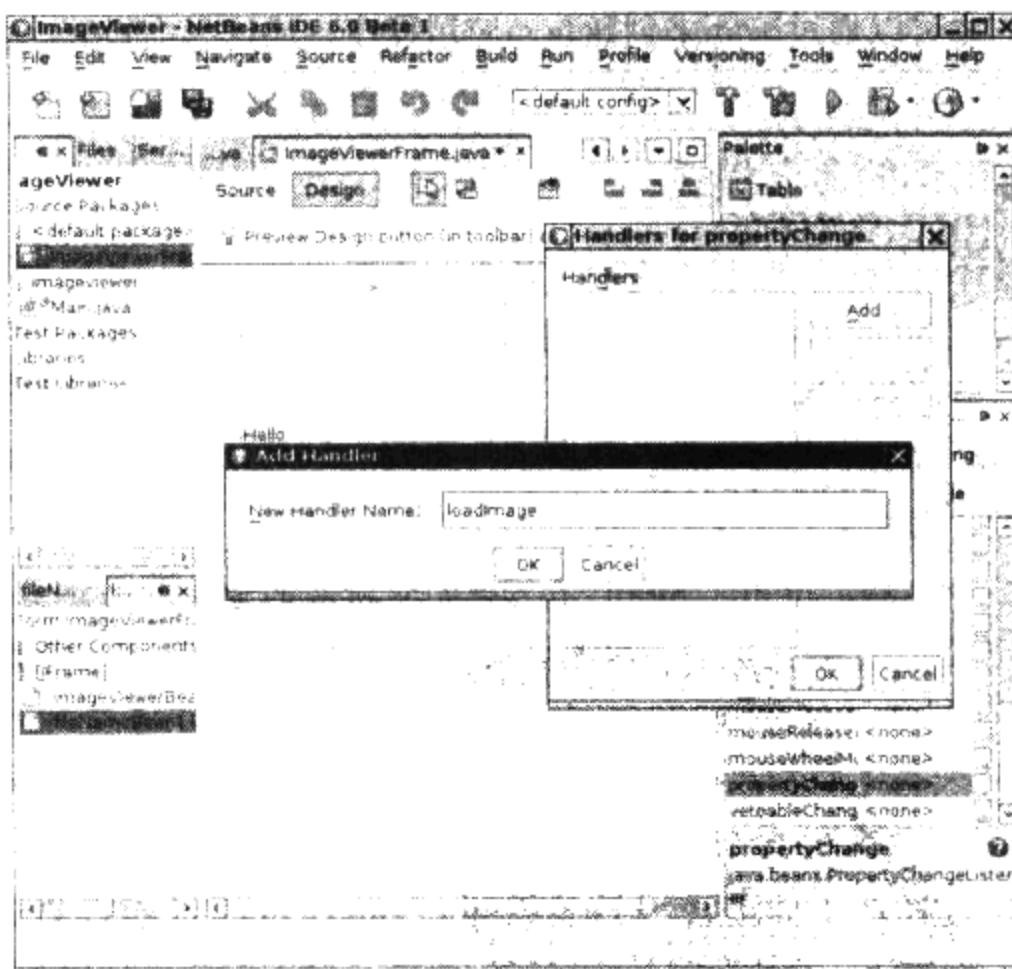


图8-8 向bean中添加一个事件

现在来看看代码编辑器，事件处理代码已经被添加了进来，而且生成了一个新方法：

```
private void loadImage(java.beans.PropertyChange evt)
{
    // TODO add your handling code here
}
```

将如下代码添加到该方法中：

```
imageViewerBean1.setFileName(fileNameBean1.getFileName());
```

然后编译并执行这个窗体类。现在，你就有了一个完整的图像浏览器应用程序。点击“...”按钮，选择一个图像文件，该图像就会显示在图像浏览器中。

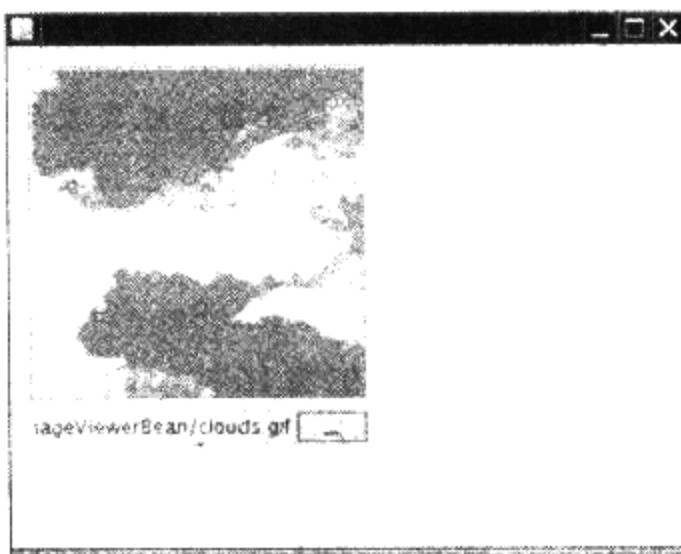


图8-9 图像浏览器应用

这个过程说明，你可以通过bean来创建Java应用程序，而这只需设置一些属性，并为事件处理提供很少量的代码。

## 8.4 Bean属性与事件的命名模式

这一节将介绍设计bean的一些基本规则。首先，需要强调一下，并没有什么通用的bean类能够供你继承。可视化的bean直接或间接地继承自Component类，但是非可视化的bean并不必继承任何特别的类。请记住，bean就是一个在开发工具中可操作的类。开发工具并不会根据某个类的父类来决定它是否是一个bean类，开发工具要做的只是分析bean中方法的名字。要想顺利通过此项分析，bean方法的命名就必须遵守某些模式。

 **注意：**确实有一个java.beans.Beans类，但是它所有的方法都是静态的。继承它没有什么意义，即便如此，你会发现它偶尔也会被用到，估计是为了“清晰度”更好。很明显，因为一个bean不能同时继承Beans和Component，因此这个方法对可视化bean不起作用。实际上，Beans类包含的方法被设计为由开发工具调用，举例来说，在设计期间或运行期间，用来检查工具是否正在运行中。

其他用于可视化设计环境的语言，例如Visual Basic和C#，都有诸如“Property”和“Event”之类的专有关键字，用以直接表达对应的概念。对于支持可视化编程，Java规范的设计者决定不增加关键字。因此，他们需要一点变通，以使开发工具能够分析bean，了解bean的属性或事件。事实上，我们有两种变通的机制。如果bean的开发者为属性与事件采用了标准的命名模式，那么开发工具就可以使用反射机制，从而了解bean打算暴露出来的属性和事件。还有一种方法，bean的开发者可以提供一个bean的信息类，由它告诉开发工具bean的属性和事件。我们决定采用命名模式，因为它们更易于上手。你会看到，本章稍后还介绍了如何提供bean的

信息类。

- 注意：**虽然相关文档称这些标准的命名模式为“设计模式”，但这些其实只是命名习惯罢了，与面向对象编程中用到的设计模式没有什么关系。

属性的命名模式非常简单：

```
public Type getPropertyName()  
public void setPropertyName(Type newValue)
```

像上面这样成对的方法就对应于一个读/写属性。

例如我们的ImageViewerBean，它只有一个读/写属性（表示要浏览的文件名），对应于以下这对方法：

```
public String getFileName()  
public void setFileName(String newValue)
```

如果有一个get方法，但是没有相应的set方法，说明你定义了一个只读的属性。与此对应，只有set方法而没有相应的get方法，即是定义了一个只写的属性。

- 注意：**你创建的get和set方法能够做的可不仅仅局限于获取或设置一个私有数据域。与其他Java方法一样，它们可以执行任意的动作。举例来说，ImageViewerBean类的setFileName方法不仅设置了fileName数据域的值，而且还打开了此文件并加载了图像。

- 注意：**在Visual Basic和C#中，属性也伴随着get和set方法。然而，在这两种语言中，你是显式地定义属性，而不是让开发工具通过分析方法的名字来猜测程序员的意图。在这些语言中，属性还有一个优点：在赋值操作的左边使用属性的名字，就会自动调用set方法。在表达式中使用属性的名字，会自动调用get方法。例如，在Visual Basic中你可以这样写

```
imageBean.fileName = "corejava.gif"
```

以代替

```
imageBean.setFileName("corejava.gif");
```

而这正是Java所采用的语法，不过Java语言设计者认为，用看起来是在访问域的语法来隐藏方法调的主意并不高明。

get/set命名模式有一个例外，就是布尔类型的属性应该使用is/set命名模式，如下所示：

```
public boolean isPropertyName()  
public void setPropertyName(boolean b)
```

例如，一个动画可能有一个running属性，它有两个方法：

```
public boolean isRunning()  
public void setRunning(boolean b)
```

setRunning方法可以启动和停止动画，isRunning方法用来报告它当前的状态。

- 注意：**布尔属性的访问器使用get前缀也是合法的（例如getRunning），不过最好使用is前缀。

请当心方法名中的大写字母，JavaBean规范的设计者决定，虽然我们例子中的get和set方法包含的是大写F (`getFileName`, `setFileName`)，但属性名应该是`fileName`，使用小写的f。bean分析器会执行一个称为字母小写化 (decapitalization) 的步骤，推导出属性的名字。(也就是get或set之后的第一个字符被转为小写字母。) 这个过程的合理之处在于，它使得方法和属性的名字对于程序员来说更为自然。

然而，如果开头两个字母都是大写字母 (例如`getURL`)，那么属性的第一个字母就不会被改成小写字母。毕竟，如果属性名为`uRL`会显得很可笑。

 **注意：**如果你的类有一对get和set方法，但是它们与你希望用户在属性检查器中操作的属性并不对应，那你该怎么办呢？如果是你自己的类，你当然可以重命名你的属性以避免这种情况。然而，如果你是继承自其他类，那么你就继承了父类中的方法名。举例来说，如果你的bean继承自`JPanel`或`JLabel`，这种情况就有可能会发生，因为属性检查器会列出大量你不感兴趣的属性。在本章稍后你会看到，如何通过提供bean信息来改写自动发现属性的过程。在bean的信息类中，可以为你的bean专门指定应该暴露的属性。

对于事件，命名模式也同样简单。当你提供用来添加或移除事件监听器的方法时，bean的开发环境能够推导出你的bean将要生成的事件。所有事件类的名字必须以Event结尾，且必须继承自`EventObject`类。

假设你的bean会生成`EventNameEvent`类型的事件，那么监听器接口必须命名为`EventNameListener`，同时，添加和删除监听器的方法必须命名为：

```
public void addEventNameListener(EventNameListener e)  
public void removeEventNameListener(EventNameListener e)  
public EventNameListener getEventNameListeners()
```

如果你浏览一下`ImageViewerBean`的代码，就会看到它没有要暴露的事件。然而，很多Swing构件都会产生事件，而它们都遵循此命名模式。例如，`AbstractButton`类会产生`ActionEvent`对象，而且它具有以下方法，用来管理`ActionListener`对象：

```
public void addActionListener(ActionListener e)  
public void removeActionListener(ActionListener e)  
ActionListener[] getActionListeners()
```

 **警告：**如果你的事件类不继承`EventObject`的话，你的代码也可能可以通过编译，因为你的代码可能没有用到`EventObject`类中的方法。然而，你的bean将会神秘地出现故障，因为自省机制 (introspection mechanism) 无法识别出你的事件。

## 8.5 Bean属性的类型

一个成熟的bean会具有多种不同种类的属性和事件。属性可以很简单，就像你在`ImageViewerBean`和`FileNameBean`中看到的`fileName`那样；也可以很精巧复杂，例如我们后面的例子中会有颜色值，甚至数据点的数组。JavaBean规范允许四种类型的属性，我们将用不同的示例来分别说明它们。

### 8.5.1 简单属性

简单属性只具有一个单独的值，例如一个字符串或者一个数字。ImageViewer的fileName属性就是一个简单属性的例子。编写简单属性非常容易：就使用我们前面讲过的set/get命名惯例即可。例如，可以参看程序清单8-1的代码，它实现了一个简单的字符串属性：

```
public void setFileName(String f)
{
    fileName = f;
    image = . . .
    repaint();
}

public String getFileName()
{
    if (file == null) return "";
    else return file.getPath();
}
```

### 8.5.2 索引属性

索引属性指定了一个数组。使用索引属性，需要提供两对get和set方法：一对给数组使用，一对给数组内的单个元素使用。它们的命名必须遵守这个模式：

```
Type[] getPropertyNames()
void setPropertyName(Type[] newValue)
Type getPropertyName(int i)
void setPropertyName(int i, Type newValue)
```

例如，FileNameBean对于文件扩展名使用了索引属性。它提供了四个方法：

```
public String[] getExtensions() { return extensions; }
public void setExtensions(String[] newValue) { extensions = newValue; }
public String getExtensions(int i)
{
    if (0 <= i && i < extensions.length) return extensions[i];
    else return "";
}
public void setExtensions(int i, String newValue)
{
    if (0 <= i && i < extensions.length) extensions[i] = newValue;
}
. . .
private String[] extensions;
```

setPropertyName(int i, Type x)方法不能用来扩充数组。要想扩充数组的容量，必须手动创建一个新的数组，然后将它传递给setPropertyName(Type[] x)方法。

### 8.5.3 绑定属性

绑定属性会在属性值发生变化时，通知所有相关的监听器。举例来说，FileNameBean中的fileName属性就是一个绑定属性。一旦文件名发生变化，就会自动通知ImageViewerBean，然后由它加载新的文件。

为了实现一个绑定属性，必须实现两个机制。

1) 无论何时,只要属性的值发生了变化,该bean必须发送一个PropertyChange事件给所有已注册的监听器。该变化可能发生在调用set方法时,或者某个其他方法(诸如“...”按钮的动作监听器)改变属性值时。

2) 为了使感兴趣的监听器能够进行注册,bean必须实现以下两个方法:

```
void addPropertyChangeListener(PropertyChangeListener listener)
void removePropertyChangeListener(PropertyChangeListener listener)
```

我们还推荐(但不是必需的)提供下面的方法。

```
PropertyChangeListener[] getPropertyChangeListeners()
```

java.beans包有一个很实用的类,叫做PropertyChangeSupport,它能够为你管理监听器。要使用这个类,需要添加一个类型为这个类的实例域:

```
private PropertyChangeSupport changeSupport = new PropertyChangeSupport(this);
```

这样就可以将添加和移除属性改变监听器的任务交给这个对象了。

```
public void addPropertyChangeListener(PropertyChangeListener listener)
{
    changeSupport.addPropertyChangeListener(listener);
}

public void removePropertyChangeListener(PropertyChangeListener listener)
{
    changeSupport.removePropertyChangeListener(listener);
}

public PropertyChangeListener[] getPropertyChangeListeners()
{
    return changeSupport.getPropertyChangeListeners();
}
```

无论何时,只要属性的值发生了变化,就应该使用PropertyChangeSupport对象的firePropertyChange方法,它会将一个事件发送给所有已注册的监听器。该方法有三个参数:属性的名字、旧的值以及新的值。下面是典型的用于绑定属性的设置器的样本代码:

```
public void setValue(Type newValue)
{
    Type oldValue = getValue();
    value = newValue;
    changeSupport.firePropertyChange("propertyName", oldValue, newValue);
}
```

要激发一个对索引属性的修改,可以调用:

```
changeSupport.fireIndexedPropertyChange("propertyName", index, oldValue, newValue);
```

**!** 提示:如果你的bean继承自某个类,而这个类最终继承自Component类,那么就不必实现addPropertyChangeListener、removePropertyChangeListener和getPropertyChangeListeners方法了。因为这些方法在超类Component中已经实现了。在属性变化时,要想通知监听器,只需调用超类Component中的firePropertyChange方法。但是,不支持对索引属性修改的激发。

当属性的值发生变化时，如果其他的bean也想收到通知，它们必须实现PropertyChangeListener接口。该接口只有一个方法：

```
void propertyChange(PropertyChangeEvent event)
```

PropertyChangeEvent对象持有属性的名字，以及旧值与新值，这些内容可以用getPropertyName、getOldValue和getNewValue方法获得。

如果属性的类型不是类，那么返回的对象是对应的包装类型。

#### 8.5.4 约束属性

一个约束属性是指它受到某种约束，即任何监听器都可以“否决”所提出的改变，强迫其还原旧的设置。Java库中只有很少的一些约束属性的例子。其中一个是JInternalFrame类的closed属性。如果某人在内部窗体上调用了setClosed(true)方法，那么它所有的VetoableChangeListeners都会收到通知。如果它们中的任何一个抛出PropertyVetoException异常，那么closed属性就不会改变，而且setClosed方法也会抛出同样的异常。特别是，如果某个VetoableChangeListener其中的内容还没有保存，它就可以否决改变该属性的提议。

要构建一个约束属性，你的bean必须包含下面两个方法，才能管理VetoableChangeListener对象：

```
public void addVetoableChangeListener(VetoableChangeListener listener);  
public void removeVetoableChangeListener(VetoableChangeListener listener);
```

它还应该有一个方法用来获取所有的监听器：

```
VetoableChangeListener[] getVetoableChangeListeners()
```

为了管理属性改变监听器，我们有一个很方便实用的类，同样，为了管理那些可否决修改的监听器，我们也有一个非常方便实用的类，叫做VetoableChangeSupport。你的bean应该包含该类的一个对象。

```
private VetoableChangeSupport vetoSupport = new VetoableChangeSupport(this);
```

于是，增加移除监听器的工作全都可以交给这个对象代理。例如：

```
public void addVetoableChangeListener(VetoableChangeListener listener)  
{  
    vetoSupport.addVetoableChangeListener(listener);  
}  
public void removeVetoableChangeListener(VetoableChangeListener listener)  
{  
    vetoSupport.removeVetoableChangeListener(listener);  
}
```

为了更新一个约束属性，bean要经历以下三个步骤：

1) 将修改属性值的意图通知所有可否决修改的监听器。（使用VetoableChangeSupport类的fireVetoableChange方法。）

2) 如果所有可否决修改的监听器都没有抛出PropertyVetoException，就更新属性的值。

3) 通知监听属性改变的所有监听器，确认有一个改变发生。

例如，

```
public void setValue(Type newValue) throws PropertyVetoException
```

```
{  
    Type oldValue = getValue();  
    vetoSupport.fireVetoableChange("value", oldValue, newValue);  
    // survived, therefore no veto  
    value = newValue;  
    changeSupport.firePropertyChange("value", oldValue, newValue);  
}
```

直到所有已注册的可否决修改的监听器都同意你提议的修改之后，属性的值才会真正改变，这一点很重要。相反地，可否决修改的监听器决不应该假设它同意的修改确实会发生。当某个修改确实发生的时候，惟一可靠的通知方式就是通过属性改变监听器。

 **注意：**如果你的bean扩展自JComponent类，就不需要一个单独的VetoableChangeSupport对象。直接调用JComponent超类的fireVetoableChange方法即可。注意，你不能对JComponent中的某个具体的属性安装可否决修改监听器，你需要监听所有的可否决修改。

我们就要结束关于JavaBean属性的讨论了，接下来是FileNameBean的完整代码（见程序清单8-2）。FileNameBean有一个索引属性extensions，和一个受约束的filename属性。因为FileNameBean继承自 JPanel类，我们就不必显式地使用PropertyChangeSupport对象了，可以直接依靠 JPanel类的能力来管理属性改变监听器。

### 程序清单8-2 FileNameBean.java

```
1. package com.horstmann.corejava;  
2.  
3. import java.awt.*;  
4. import java.awt.event.*;  
5. import java.io.*;  
6. import java.util.*;  
7. import javax.swing.*;  
8. import javax.swing.filechooser.*;  
9.  
10. /**  
11.  * A bean for specifying file names.  
12.  * @version 1.30 2007-10-03  
13.  * @author Cay Horstmann  
14. */  
15. public class FileNameBean extends JPanel  
16. {  
17.     public FileNameBean()  
18.     {  
19.         dialogButton = new JButton(...);  
20.         nameField = new JTextField(30);  
21.  
22.         chooser = new JFileChooser();  
23.         setPreferredSize(new Dimension(XPREFSIZE, YPREFSIZE));  
24.  
25.         setLayout(new GridBagLayout());  
26.         GridBagConstraints gbc = new GridBagConstraints();  
27.         gbc.weightx = 100;  
28.         gbc.weighty = 100;  
29.         gbc.anchor = GridBagConstraints.WEST;
```

```
30.     gbc.fill = GridBagConstraints.BOTH;
31.     gbc.gridwidth = 1;
32.     gbc.gridheight = 1;
33.     add(nameField, gbc);
34.
35.     dialogButton.addActionListener(new ActionListener()
36.     {
37.         public void actionPerformed(ActionEvent event)
38.         {
39.             chooser.setFileFilter(new FileNameExtensionFilter(Arrays.toString(extensions),
40.                 extensions));
41.             int r = chooser.showOpenDialog(null);
42.             if (r == JFileChooser.APPROVE_OPTION)
43.             {
44.                 File f = chooser.getSelectedFile();
45.                 String name = f.getAbsolutePath();
46.                 setFileName(name);
47.             }
48.         }
49.     });
50.     nameField.setEditable(false);
51.
52.     gbc.weightx = 0;
53.     gbc.anchor = GridBagConstraints.EAST;
54.     gbc.fill = GridBagConstraints.NONE;
55.     gbc.gridx = 1;
56.     add(dialogButton, gbc);
57. }
58.
59. /**
60. * Sets the fileName property.
61. * @param newValue the new file name
62. */
63. public void setFileName(String newValue)
64. {
65.     String oldValue = nameField.getText();
66.     nameField.setText(newValue);
67.     firePropertyChange("fileName", oldValue, newValue);
68. }
69.
70. /**
71. * Gets the fileName property.
72. * @return the name of the selected file
73. */
74. public String getFileName()
75. {
76.     return nameField.getText();
77. }
78.
79. /**
80. * Gets the extensions property.
81. * @return the default extensions in the file chooser
82. */
83. public String[] getExtensions()
84. {
85.     return extensions;
```

```

86.    }
87.
88.    /**
89.     * Sets the extensions property.
90.     * @param newValue the new default extensions
91.     */
92.    public void setExtensions(String[] newValue)
93.    {
94.        extensions = newValue;
95.    }
96.
97.    /**
98.     * Gets one of the extensions property values.
99.     * @param i the index of the property value
100.    * @return the value at the given index
101.    */
102.   public String getExtensions(int i)
103.   {
104.       if (0 <= i && i < extensions.length) return extensions[i];
105.       else return "";
106.   }
107.
108. /**
109.  * Sets one of the extensions property values.
110. * @param i the index of the property value
111. * @param newValue the new value at the given index
112. */
113. public void setExtensions(int i, String newValue)
114. {
115.     if (0 <= i && i < extensions.length) extensions[i] = newValue;
116. }
117.
118. private static final int XPREFSIZE = 200;
119. private static final int YPREFSIZE = 20;
120. private JButton dialogButton;
121. private JTextField nameField;
122. private JFileChooser chooser;
123. private String[] extensions = { "gif", "png" };
124. }

```

### **API** `java.beans.PropertyChangeListener 1.1`

- `void propertyChange(PropertyChangeEvent event)`

当发生了一个属性改变事件时该方法会被调用。

### **API** `java.beans.PropertyChangeSupport 1.1`

- `PropertyChangeSupport(Object sourceBean)`

构建一个PropertyChangeSupport对象，用来管理对给定bean的绑定属性的变化感兴趣的监听器。

- `void addPropertyChangeListener(PropertyChangeListener listener)`

- `void addPropertyChangeListener(String propertyName, PropertyChangeListener listener)` 1.2

注册一个监听器，它对所有的绑定属性发生的改变都感兴趣，或者只是为指定名字的绑定属性注册一个监听器。

- void removePropertyChangeListener(PropertyChangeListener listener)
- void removePropertyChangeListener(String propertyName, PropertyChangeListener listener) 1.2

移除一个之前注册过的属性变化监听器。

- void firePropertyChange(String propertyName, Object oldValue, Object newValue)
- void firePropertyChange(String propertyName, int oldValue, int newValue) 1.2
- void firePropertyChange(String propertyName, boolean oldValue, boolean newValue) 1.2

向已注册的监听器发送一个PropertyChangeEvent。

- void fireIndexedPropertyChange(String propertyName, int index, Object oldValue, Object newValue) 5.0
- void fireIndexedPropertyChange(String propertyName, int index, int oldValue, int newValue) 5.0
- void fireIndexedPropertyChange(String propertyName, int index, boolean oldValue, boolean newValue) 5.0

向已注册的监听器发送一个IndexedPropertyChangeEvent。

- PropertyChangeListener[] getPropertyChangeListeners() 1.4
- PropertyChangeListener[] getPropertyChangeListeners(String propertyName) 1.4

获取对所有绑定属性的改变都感兴趣的监听器，或者只是获取指定名字属性的监听器。

#### **API java.beans.PropertyChangeEvent 1.1**

- PropertyChangeEvent(Object sourceBean, String propertyName, Object oldValue, Object newValue)

构建一个新的PropertyChangeEvent对象，用来描述所给属性已经由oldValue变为newValue。

- Object getNewValue()

返回该属性的新值。

- Object getOldValue();

返回该属性的旧值。

- String getPropertyName()

返回属性的名字。

#### **API java.beans.IndexedPropertyChangeEvent 5.0**

- IndexedPropertyChangeEvent(Object sourceBean, String propertyName, int index, Object oldValue, Object newValue)

构建一个新的IndexedPropertyChangeEvent对象，用来描述所给属性在指定索引位置的

元素对象由oldValue变为newValue。

- int getIndex()

返回发生变化的元素的索引号。

#### **API** `java.beans.VetoableChangeListener 1.1`

- void vetoableChange(PropertyChangeEvent event)

当属性将发生变化时调用此方法。如果该变化不可接受，会抛出PropertyVetoException异常。

#### **API** `java.beans.VetoableChangeSupport 1.1`

- VetoableChangeSupport(Object sourceBean)

构建一个PropertyChangeSupport对象，用来管理参数bean上的约束属性的监听器。

- void addVetoableChangeListener(VetoableChangeListener listener)

- void addVetoableChangeListener(String propertyName, VetoableChangeListener listener) 1.2

注册一个监听器，它对所有约束属性的变化都感兴趣，或者只为给定名字的约束属性注册一个监听器。

- void removeVetoableChangeListener(VetoableChangeListener listener)

- void removeVetoableChangeListener(String propertyName, VetoableChangeListener listener) 1.2

移除之前注册的一个可否决修改的监听器。

- void fireVetoableChange(String propertyName, Object oldValue, Object newValue)

- void fireVetoableChange(String propertyName, int oldValue, int newValue) 1.2

- void fireVetoableChange(String propertyName, boolean oldValue, boolean newValue) 1.2

向已注册的监听器发送一个VetoableChangeEvent事件。

- VetoableChangeListener[] getVetoableChangeListeners() 1.4

- VetoableChangeListener[] getVetoableChangeListeners(String propertyName) 1.4

获取对所有约束属性的变化都感兴趣的监听器，或者返回指定名字的绑定属性的监听器。

#### **API** `java.awt.Component 1.0`

- void addPropertyChangeListener(PropertyChangeListener listener) 1.2

- void addPropertyChangeListener(String propertyName, PropertyChangeListener listener) 1.2

注册一个监听器，它对所有绑定属性的变化都感兴趣，或注册一个只对指定名字的绑定属性感兴趣的监听器。

- void removePropertyChangeListener(PropertyChangeListener listener) 1.2

- void removePropertyChangeListener(String propertyName, PropertyChangeListener

listener) 1.2

移除之前注册的一个属性改变监听器。

- void firePropertyChange(String propertyName, Object oldValue, Object newValue) 1.2  
向已注册监听器发送一个PropertyChangeEvent事件。

#### javax.swing.JComponent 1.2

- void addVetoableChangeListener(VetoableChangeListener listener)  
注册一个监听器，对所有约束属性的变化感兴趣，或者注册一个只对指定名字的约束属性感兴趣的监听器。
- void removeVetoableChangeListener(VetoableChangeListener listener)  
移除之前注册的一个可否决修改的监听器。
- void fireVetoableChange(String propertyName, Object oldValue, Object newValue)  
向已注册监听器发送一个VetoableChangeEvent事件。

#### java.beans.PropertyVetoException 1.1

- PropertyVetoException(String message, PropertyChangeEvent event)  
生成一个新的PropertyVetoException。
- PropertyChangeEvent getPropertyChangeEvent()  
返回PropertyChangeEvent，用来构建PropertyVetoException异常。

## 8.6 BeanInfo类

如果你的bean中的方法使用了标准的命名模式，那么开发工具就可以使用反射机制来确定bean的特征，例如属性以及事件。这个过程使得bean的编程非常容易，但是，命名模式最终也有其局限。当你的bean变得复杂的时候，bean就会有一些特征无法通过命名模式来揭示。还有我们已经提到过的，很多bean都具有不应该对应到bean属性的get/set方法对。

如果你需要更灵活的方式来描述有关bean的信息，可以定义一个实现了BeanInfo接口的对象。只要你提供了这样的对象，开发工具就会通过询问它来识别你的bean具有的特性。

bean信息类的名字必须是通过将BeanInfo添加到bean的名字后面构成。举例来说，与ImageViewerBean类关联的bean信息类必须命名为ImageViewerBeanBeanInfo。bean信息类还必须与bean在同一个包中。

一般情况下，你的类并不需要实现BeanInfo接口中的所有方法。但是，你应该继承SimpleBeanInfo类，它很实用，提供了BeanInfo接口中所有方法的默认实现。

使用BeanInfo类最常见的原因是获取对bean属性的控制权。这样的话，你只需提供属性名和所属的bean类，就可以为每个属性构建一个PropertyDescriptor。

```
PropertyDescriptor descriptor = new PropertyDescriptor("fileName", ImageViewerBean.class);
```

然后实现BeanInfo类中的getPropertyDescriptors方法，使它返回一个数组，其中包含了所有的属性描述符。

例如，假设 ImageViewerBean 想隐藏它由超类 JLabel 继承而来的所有属性，只暴露出 fileName 属性。那么下面的 BeanInfo 类可以做到这一点：

```
// bean info class for ImageViewerBean
class ImageViewerBeanBeanInfo extends SimpleBeanInfo
{
    public PropertyDescriptor[] getPropertyDescriptors()
    {
        return propertyDescriptors;
    }

    private PropertyDescriptor[] propertyDescriptors = new PropertyDescriptor[]
    {
        new PropertyDescriptor("fileName", ImageViewerBean.class);
    };
}
```

其他方法也可以返回 EventSetDescriptor 和 MethodDescriptor 数组，但是它们不常用。如果其中一个方法返回 null (SimpleBeanInfo 中的方法就是如此)，那么标准命名模式就派上用场了。无论如何，如果你覆盖了其中一个方法，使它返回一个非 null 数组，那么该数组必须包含所有的属性、事件、或者方法。

 **注意：**有的时候，读者可能希望编写具有通用性的代码，用来发现任意 bean 的属性或事件。这时可以调用 Introspector 类中的静态方法 getBeanInfo。Introspector 会构建出一个 BeanInfo 类，而它完整地描述了 bean，而你则可以参考这个 BeanInfo 类中携带的信息。

BeanInfo 接口中另一个有用的方法是 getIcon，可以利用它给你的 bean 自定义一个图标。开发工具将在选项板中显示该图标。实际上，你可以指定四个独立的图标位图。BeanInfo 接口有四个常量，涵盖了各种标准尺寸：

```
ICON_COLOR_16x16
ICON_COLOR_32x32
ICON_MONO_16x16
ICON_MONO_32x32
```

在下面的类中，我们使用了 SimpleBeanInfo 类中的便利方法 loadImage 来加载一个图标的图像：

```
public class ImageViewerBeanBeanInfo extends SimpleBeanInfo
{
    public ImageViewerBeanBeanInfo()
    {
        iconColor16 = loadImage("ImageViewerBean_COLOR_16x16.gif");
        iconColor32 = loadImage("ImageViewerBean_COLOR_32x32.gif");
        iconMono16 = loadImage("ImageViewerBean_MONO_16x16.gif");
        iconMono32 = loadImage("ImageViewerBean_MONO_32x32.gif");
    }

    public Image getIcon(int iconType)
    {
        if (iconType == BeanInfo.ICON_COLOR_16x16) return iconColor16;
        else if (iconType == BeanInfo.ICON_COLOR_32x32) return iconColor32;
        else if (iconType == BeanInfo.ICON_MONO_16x16) return iconMono16;
        else if (iconType == BeanInfo.ICON_MONO_32x32) return iconMono32;
    }
}
```

```
    else return null;
}

private Image iconColor16;
private Image iconColor32;
private Image iconMono16;
private Image iconMono32;
}
```

#### **java.beans.Introspector 1.1**

- static BeanInfo getBeanInfo(Class<?> beanClass)

获取指定类的bean信息。

#### **java.beans.BeanInfo 1.1**

- PropertyDescriptor[] getPropertyDescriptors()

返回bean属性的描述符。如果返回null，则表示应该用命名规则来查找属性。

- Image getIcon(int iconType)

返回一个图像对象，它在工具箱、工具条、或类似的地方代表该bean。如前所述，有四个常量代表四种标准类型的图标。

#### **java.beans.SimpleBeanInfo 1.1**

- Image loadImage(String resourceName)

返回与资源相关联的一个图像对象文件。资源名是一个路径名，该路径是相对于包含bean信息类的目录的路径。

#### **java.beans.FeatureDescriptor 1.1**

- String getName()

- void setName(String name)

获得或设置该特性在程序中的名字。

- String getDisplayName()

- void setDisplayName(String displayName)

获取或设置该特性用以显示的名字。默认值是getName的返回值。然而，当前还没有明确地支持以多种本地语言提供属性的名字。

- String getShortDescription()

- void setShortDescription(String text)

获取或设置一个字符串，开发工具使用它为该特性提供一个简短的描述。默认值是getDisplayName的返回值。

- boolean isExpert()

- void setExpert(boolean b)

获取或设置一个专家标记，开发工具可用它来决定是否对普通用户隐藏某个特性。

- boolean isHidden()

- void setHidden(boolean b)

获取或设置一个标记，开发工具根据它隐藏此特性。

#### **API** `java.beans.PropertyDescriptor 1.1`

- `PropertyDescriptor(String propertyName, Class<?> beanClass)`
- `PropertyDescriptor(String propertyName, Class<?> beanClass, String getMethod, String setMethod)`

构建一个 `PropertyDescriptor` 对象。在自省时如果发生错误，该方法会抛出 `IntrospectionException` 异常。第一个构造器假设你遵守了 `get` 和 `set` 方法的标准命名惯例。

- `Class<?> getPropertyType()`

针对属性的类型返回一个 `Class` 对象。

- `Method getReadMethod()`

- `Method getWriteMethod()`

返回获取和设置属性的方法。

#### **API** `java.beans.IndexedPropertyDescriptor 1.1`

- `IndexedPropertyDescriptor(String propertyName, Class<?> beanClass)`
- `IndexedPropertyDescriptor(String propertyName, Class<?> beanClass, String getMethod, String setMethod, String indexedGetMethod, String indexedSetMethod)`

为带索引的属性构造一个 `IndexedPropertyDescriptor`。第一个构造器假设你遵守了 `get` 和 `set` 方法标准的命名惯例。

- `Method getIndexedReadMethod()`

- `Method getIndexedWriteMethod()`

返回获取和设置属性中的索引值的方法。

## 8.7 属性编辑器

如果向 bean 添加一个整数或字符串的属性，它会自动在 bean 的属性检查器中显示出来。但是，如果你添加的属性的值不能在文本域中方便地编辑，例如日期或 `Color`，那会怎么样呢？这就需要你提供一个单独的构件，用户可以通过它来设定属性值。这样的构件就称为属性编辑器。例如，一个日期对象的属性编辑器，它可以是一个日历，用户可以滚动月份选择一个日期。而 `Color` 对象的属性编辑器则允许用户选择所需颜色的红、绿、蓝成分。

实际上，NetBeans 已经提供了颜色属性编辑器。当然，还有基本类型例如 `String`（一个文本域）和 `boolean`（一个检验栏）的属性编辑器。

我们简要介绍一下提供一个全新的属性编辑器的过程。首先，为你的 bean 创建一个 bean 信息类，并覆盖 `getPropertyDescriptors` 方法。这个方法返回一个 `PropertyDescriptor` 对象数组。为需要在属性编辑器中显示的每一个属性都创建一个对象，即使是只需要使用默认属性编辑器的那些属性也是如此。

用属性名和包含该属性的 bean 类，来构建一个 `PropertyDescriptor`。

```
PropertyDescriptor descriptor = new PropertyDescriptor("titlePosition", ChartBean.class);
```

然后调用PropertyDescriptor类上的setPropertyEditorClass方法。

```
descriptor.setPropertyEditorClass(TitlePositionEditor.class);
```

接下来，为你的bean的属性构建一个描述符数组。例如，我们讨论过的图表bean，它有五个属性：

- 一个Color属性，graphColor。
- 一个String属性，title。
- 一个int属性，titlePosition。
- 一个double[]属性，values。
- 一个boolean属性，inverse。

程序清单8-3就是ChartBeanBeanInfo类的代码，它为这些属性指定了属性编辑器，实现了以下两个目标：

- 1) getPropertyDescriptors方法为每个属性返回一个描述符。title和graphColor属性使用默认的编辑器，也就是开发工具自带的字符串编辑器和颜色编辑器。
- 2) titlePosition, values和inverse属性使用专门的编辑器，分别是TitlePositionEditor, DoubleArrayEditor和InverseEditor。

图8-10展示了图表bean。可以看到title在顶部。它的位置也可以设置为左、中、右。values属性指明了图的值。如果inverse属性为真，那么背景就会有颜色，而图表中的柱图就为白色。你可以在本书附带的代码中找到chart bean的代码，它修改自第1卷第10章的chart applet。

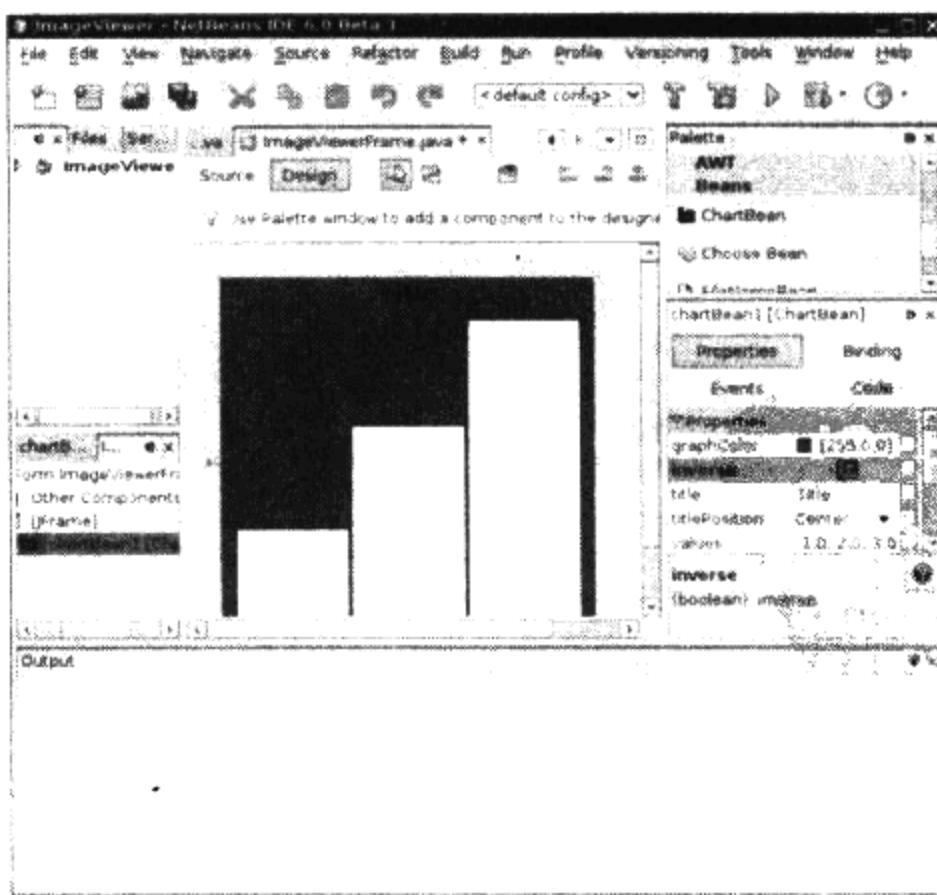


图8-10 chart bean

**程序清单8-3 ChartBeanBeanInfo.java**

```
1 package com.horstmann.corejava;
2
3 import java.awt.*;
4 import java.beans.*;
5
6 /**
7 * The bean info for the chart bean, specifying the property editors.
8 * @version 1.20 2007-10-05
9 * @author Cay Horstmann
10 */
11 public class ChartBeanBeanInfo extends SimpleBeanInfo
12 {
13     public ChartBeanBeanInfo()
14     {
15         iconColor16 = loadImage("ChartBean_COLOR_16x16.gif");
16         iconColor32 = loadImage("ChartBean_COLOR_32x32.gif");
17         iconMono16 = loadImage("ChartBean_MONO_16x16.gif");
18         iconMono32 = loadImage("ChartBean_MONO_32x32.gif");
19
20         try
21         {
22             PropertyDescriptor titlePositionDescriptor = new PropertyDescriptor("titlePosition",
23                 ChartBean.class);
24             titlePositionDescriptor.setPropertyEditorClass(TitlePositionEditor.class);
25             PropertyDescriptor inverseDescriptor = new PropertyDescriptor("inverse", ChartBean.class);
26             inverseDescriptor.setPropertyEditorClass(InverseEditor.class);
27             PropertyDescriptor valuesDescriptor = new PropertyDescriptor("values", ChartBean.class);
28             valuesDescriptor.setPropertyEditorClass(DoubleArrayEditor.class);
29             propertyDescriptors = new PropertyDescriptor[] {
30                 new PropertyDescriptor("title", ChartBean.class), titlePositionDescriptor,
31                 valuesDescriptor, new PropertyDescriptor("graphColor", ChartBean.class),
32                 inverseDescriptor };
33         }
34         catch (IntrospectionException e)
35         {
36             e.printStackTrace();
37         }
38     }
39
40     public PropertyDescriptor[] getPropertyDescriptors()
41     {
42         return propertyDescriptors;
43     }
44
45     public Image getIcon(int iconType)
46     {
47         if (iconType == BeanInfo.ICON_COLOR_16x16) return iconColor16;
48         else if (iconType == BeanInfo.ICON_COLOR_32x32) return iconColor32;
49         else if (iconType == BeanInfo.ICON_MONO_16x16) return iconMono16;
50         else if (iconType == BeanInfo.ICON_MONO_32x32) return iconMono32;
51         else return null;
52     }
53
54     private PropertyDescriptor[] propertyDescriptors;
55     private Image iconColor16;
56     private Image iconColor32;
```

```
57     private Image iconMono16;  
58     private Image iconMono32;  
59 }
```

### API java.beans.PropertyDescriptor 1.1

- `PropertyDescriptor(String name, Class <?> beanClass)`

构建一个PropertyDescriptor对象。

参数: name 属性的名字

beanClass 该属性所属的bean的类

- `void setPropertyEditorClass(Class <?> editorClass)`

设置该属性将使用的属性编辑器的类。

### API java.beans.BeanInfo 1.1

- `PropertyDescriptor[] getPropertyDescriptors()`

返回每一个属性所对应的描述符，这些属性都是会在bean的属性检查器中显示的属性。

## 编写一个属性编辑器

在深入编写属性编辑器的机制之前，我们还应该指出一点，编辑器受开发工具的控制，而不是受bean的控制。当开发工具显示属性检查器时，它会对每个bean属性执行下面的步骤：

- 1) 实例化一个属性编辑器。
- 2) 向bean询问该属性的当前值。
- 3) 然后要求属性编辑器显示该值。

属性编辑器必须提供默认的构造器，并实现PropertyEditor接口。通常我们都会去扩展便利的PropertyEditorSupport类，它提供了该接口中的方法的默认实现版本。

对于你所写的每个属性编辑器，都需要在三种显示和编辑属性值的方式中选择一种：

- 作为文本字符串处理（定义`getAsString`和`setAsString`）。
- 作为选择域处理（定义`getAsText`、`setAsText`和`getTags`）。
- 通过绘制按照图形化的方式处理（定义`isPaintable`、`paintValue`、`supportsCustomEditor`和`getCustomEditor`）。

下面我们将进一步讨论这些不同的选择。

### 1. 简单属性编辑器

简单属性编辑器使用的是文本字符串，需要覆盖`setAsString`和`getAsString`方法。例如对于我们自己的图表bean，可以设置它的一个属性来决定标题显示的位置：居左、居中或居右。这些选择其实是作为枚举来实现的。

```
public enum Position { LEFT, CENTER, RIGHT };
```

当然，我们不会让它们就以大写字符串LEFT、CENTER、RIGHT的形式显示在文本域中，除非我们想进入“用户接口恐怖名人堂”。因此，我们定义了一个属性编辑器，它的`getAsString`方

法会把让开发者喜欢的字符串取出来。

```
class TitlePositionEditor extends PropertyEditorSupport
{
    public String getAsText()
    {
        int index = ((ChartBean.Position) getValue()).ordinal();
        return tags[index];
    }
    ...
    private String[] tags = { "Left", "Center", "Right" };
}
```

理想情况下，这些字符串应该按照当前的Locale进行显示，而不一定非要是英语的，但是我们将这个任务留给读者作为一个练习去完成。

反过来，我们需要提供一个方法将文本字符串转换回属性值：

```
public void setAsText(String s)
{
    int index = Arrays.asList(tags).indexOf(s);
    if (index >= 0) setValue(ChartBean.Position.values()[index]);
}
```

如果我们只提供这两个方法，属性检查器就会提供一个文本框，它是通过对getAsText的调用进行初始化的，而在我们完成编辑时，setAsText会被调用。当然，在上述情况中，这个属性编辑器对于titlePosition属性来说并不是一个很好的选择，当然，除非我们正打算争夺进入“用户接口羞耻名人堂”的入场券。更好的办法是将所有可能的设置都显示出来（参见图8-11）。PropertyEditorSupport类给出了一个简单的机制，用于表示恰当的方式是使用组合框。我们只需编写一个getTags方法，它返回一个字符串数组。

```
public String[] getTags() { return tags; }
```

getTags方法默认返回null，文本框适合用来编辑该属性值。

即便提供了getTags方式，我们仍需提供getAsText和setAsText方法。getTags方法直接指出哪些字符串应该呈现给用户。而getAsText和setAsText方法负责在字符串与该属性的数据类型（可以是字符串、整数、或某种完全不同的类型）之间的转换工作。

最后，属性编辑器应该实现getJavaInitializationString方法。通过使用这个方法，可以向开发工具提供将一个属性设置为其当前值的Java代码，开发工具将用这些代码字符串进行自动代码生成。下面是用于TitlePositionEditor的该方法：

```
public String getJavaInitializationString()
{
    return ChartBean.Position.class.getName().replace('$', '.') + "." + getValue();
}
```

这个方法将返回诸如"com.horstmann.corejava.ChartBean.Position.LEFT"这样的字符串。试着在NetBeans中使用它：如果你编辑titlePosition属性，那么NetBeans就会插入像下面这样的代码：

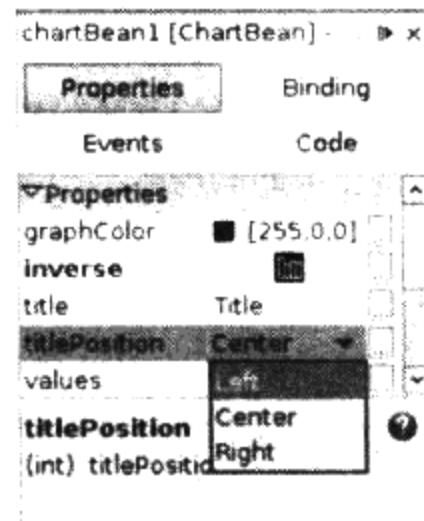


图8-11 自定义的属性编辑器

```
chartBean1.setTitlePosition(com.horstmann.corejava.ChartBean.Position.LEFT);
```

本例中，这个代码有些麻烦，因为ChartBean.Position.class.getName()的返回值是字符串"com.horstmann.corejava.ChartBean\$Position"，我们需要用句点来替换\$，并追加在枚举值上调用toString方法得到的结果。

 **注意：**如果一个属性有自定义的编辑器，而这个编辑器没有实现getJavaInitializationString方法，那么NetBeans就不了解应该如何生成代码，于是就产生了具有???参数的设置方法。

程序清单8-4包含了此属性编辑器的完整代码。

#### 程序清单8-4 TitlePositionEditor.java

```
1. package com.horstmann.corejava;
2.
3. import java.beans.*;
4. import java.util.*;
5.
6. /**
7. * A custom editor for the titlePosition property of the ChartBean. The editor lets the user
8. * choose between Left, Center, and Right
9. * @version 1.20 2007-12-14
10. * @author Cay Horstmann
11. */
12. public class TitlePositionEditor extends PropertyEditorSupport
13. {
14.     public String[] getTags()
15.     {
16.         return tags;
17.     }
18.
19.     public String getJavaInitializationString()
20.     {
21.         return ChartBean.Position.class.getName().replace('$', '.') + "." + getValue();
22.     }
23.
24.     public String getAsText()
25.     {
26.         int index = ((ChartBean.Position) getValue()).ordinal();
27.         return tags[index];
28.     }
29.
30.     public void setAsText(String s)
31.     {
32.         int index = Arrays.asList(tags).indexOf(s);
33.         if (index >= 0) setValue(ChartBean.Position.values()[index]);
34.     }
35.
36.     private String[] tags = { "Left", "Center", "Right" };
37. }
```

#### 2. 基于GUI的属性编辑器

更复杂的属性类型则不应该作为文本来编辑，而是应该在属性检查器中显示图形化的表示，

这个小区域在其他情况下持有的一个文本框或者组合框。当用户点击该区域时，会弹出一个自定义的编辑器对话框（参见图8-12）。这个对话框包含一个由属性编辑器提供的用来编辑属性值的构件，还有由开发工具提供的各种按钮。在本例中，定制的编辑器非常简约，只包含一个按钮。本书附带的代码包含一个用于编辑图表值的更精致的编辑器。

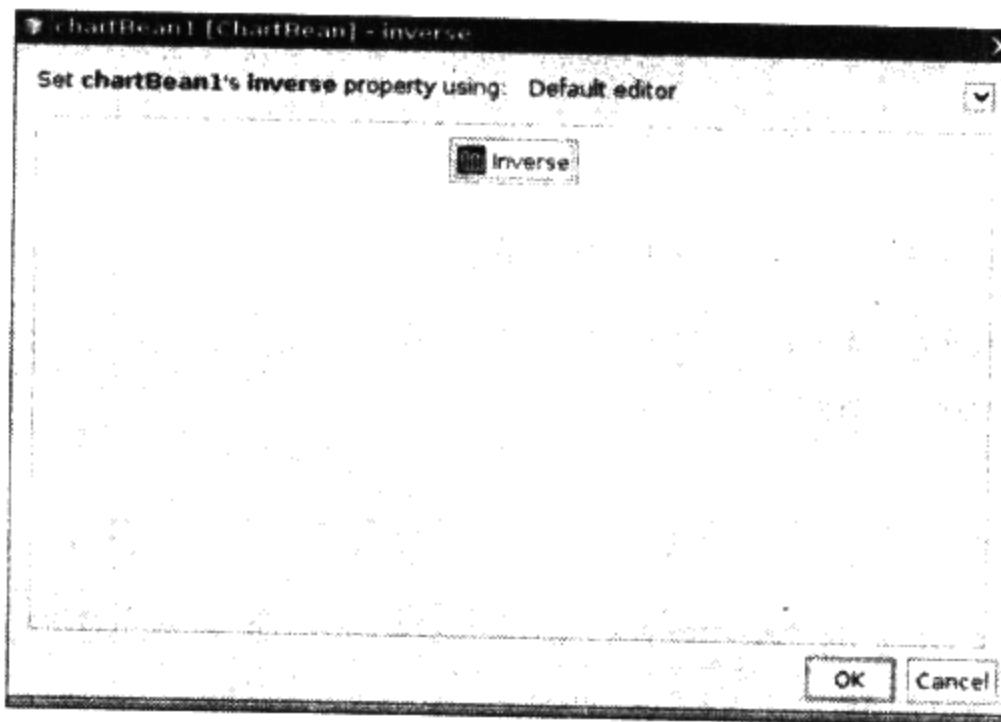


图8-12 一个自定义的编辑器对话框

要开发一个基于GUI的属性编辑器，首先要告诉属性检查器你将画出属性的值，而不是使用字符串。

然后覆盖PropertyEditor接口中的getAsText方法，使之返回null，并且令isPaintable方法返回true。

然后，要实现paintValue方法。它接收一个Graphics上下文和相应的矩形区域作为参数，你将在该矩形区域内画图。请注意，该矩形通常比较小，因此你不能使用很复杂的表达方式。我们只是画了两个图标中的一个。（参见图8-11）

```
public void paintValue(Graphics g, Rectangle box)
{
    ImageIcon icon = (Boolean) getValue() ? inverseIcon : normalIcon;
    int x = bounds.x + (bounds.width - icon.getIconWidth()) / 2;
    int y = bounds.y + (bounds.height - icon.getIconHeight()) / 2;
    g.drawImage(icon.getImage(), x, y, null);
}
```

当然，这个图形化的表示是不可编辑的。用户必须点击它，才能弹出一个自定义的编辑器。

通过覆写PropertyEditor接口的supportsCustomEditor方法，使之返回true，就可以表示你将使用一个自定义的编辑器。

接下来，由PropertyEditor接口的getCustomEditor方法构造并返回一个自定义编辑器类的对象。

程序清单8-5展示了在属性检查器中显示当前属性值的InverseEditor的代码。

程序清单8-6展示了用于修改这个值的自定义的编辑器面板的代码。

**程序清单8-5 InverseEditor.java**

```
1. package com.horstmann.corejava;
2.
3. import java.awt.*;
4. import java.beans.*;
5. import javax.swing.*;
6.
7. /**
8. * The property editor for the inverse property of the ChartBean. The inverse property toggles
9. * between colored graph bars and colored background.
10. * @version 1.30 2007-10-03
11. * @author Cay Horstmann
12. */
13. public class InverseEditor extends PropertyEditorSupport
14. {
15.     public Component getCustomEditor()
16.     {
17.         return new InverseEditorPanel(this);
18.     }
19.
20.     public boolean supportsCustomEditor()
21.     {
22.         return true;
23.     }
24.
25.     public boolean isPaintable()
26.     {
27.         return true;
28.     }
29.
30.     public String getAsText()
31.     {
32.         return null;
33.     }
34.
35.     public String getJavaInitializationString()
36.     {
37.         return "" + getValue();
38.     }
39.
40.     public void paintValue(Graphics g, Rectangle bounds)
41.     {
42.         ImageIcon icon = (Boolean) getValue() ? inverseIcon : normalIcon;
43.         int x = bounds.x + (bounds.width - icon.getIconWidth()) / 2;
44.         int y = bounds.y + (bounds.height - icon.getIconHeight()) / 2;
45.         g.drawImage(icon.getImage(), x, y, null);
46.     }
47.
48.     private ImageIcon inverseIcon = new ImageIcon(getClass().getResource(
49.             "ChartBean_INVERSE_16x16.gif"));
50.     private ImageIcon normalIcon =
51.         new ImageIcon(getClass().getResource("ChartBean_MONO_16x16.gif"));
52. }
```

## 程序清单8-6 InverseEditorPanel.java

```
1. package com.horstmann.corejava;
2.
3. import java.awt.event.*;
4. import java.beans.*;
5. import javax.swing.*;
6.
7. /**
8. * The panel for setting the inverse property. It contains a button to toggle between normal
9. * and inverse coloring.
10.* @version 1.30 2007-10-03
11.* @author Cay Horstmann
12.*/
13. public class InverseEditorPanel extends JPanel
14. {
15.     public InverseEditorPanel(PropertyEditorSupport ed)
16.     {
17.         editor = ed;
18.         button = new JButton();
19.         updateButton();
20.         button.addActionListener(new ActionListener()
21.         {
22.             public void actionPerformed(ActionEvent event)
23.             {
24.                 editor.setValue(!(Boolean) editor.getValue());
25.                 updateButton();
26.             }
27.         });
28.         add(button);
29.     }
30.
31.     private void updateButton()
32.     {
33.         if ((Boolean) editor.getValue())
34.         {
35.             button.setIcon(inverseIcon);
36.             button.setText("Inverse");
37.         }
38.         else
39.         {
40.             button.setIcon(normalIcon);
41.             button.setText("Normal");
42.         }
43.     }
44.
45.     private JButton button;
46.     private PropertyEditorSupport editor;
47.     private ImageIcon inverseIcon = new ImageIcon(getClass().getResource(
48.         "ChartBean_INVERSE_16x16.gif"));
49.     private ImageIcon normalIcon =
50.         new ImageIcon(getClass().getResource("ChartBean_MONO_16x16.gif"));
51. }
```



### java.beans.PropertyEditor 1.1

- **Object getValue()**

返回属性的当前值，基本类型会被封装到对象包装器中。

- **void setValue(Object newValue)**

设置属性返回新值，基本类型必须封装到对象包装器中。

参数：newValue 对象的新值，应该是一个该属性可以拥有的新创建的对象

- **String getAsText()**

覆盖该方法，使其返回属性当前值的一个字符串表示。默认情况下返回null，表示该属性不能表示成字符串。

- **void setAsText(String text)**

覆盖该方法，让其用通过解析文本获得的新值来设置属性。如果这个文本表示的不是合法值，或者这个属性不能用字符串表示，则有可能会抛出IllegalArgumentException。

- **String[] getTags()**

覆盖该方法，让其返回属性值所有可能的字符串表示，以使得它们可以显示在选择框中。默认情况下返回null，表示不存在字符串值的有限集。

- **boolean isPaintable()**

如果该类使用paintValue方法来显示属性，就应该覆写此方法，使之返回true。

- **void paintValue(Graphics g, Rectangle bands)**

应该覆写该方法，使其通过将属性值画入一个图形上下文环境中来表示该属性的值，该图形上下文环境位于属性检查器所用构件上指定的位置。

- **boolean supportsCustomEditor()**

如果该属性编辑器有一个自定义编辑器，那么就应该覆写该方法，使其返回true。

- **Component getCustomEditor()**

应该覆写该方法，使其返回一个构件，该构件包含了用来编辑属性值的自定义的GUI。

- **String getJavaInitializationString()**

应该覆写该方法，返回一个Java代码字符串，它将用来生成初始化属性值的代码。例如 "0", "new Color(64, 64, 64)"。

## 8.8 定制器

属性编辑器是为了让用户在某个时刻设置一个属性。但是，如果某个bean的特定属性与其他属性有关联时，它应该向用户提供更友好的方法，以便用户在同一时刻能够编辑多个属性。要使用这种特性，你需要提供一个定制器来取代多属性编辑器，或者在多属性编辑器的基础上增加一个定制器。

此外，某些bean可能会有一些特性并不希望作为属性暴露出来，因此也就不能通过属性检查器来编辑。对于这样的bean，定制器就很关键了。

在这一节的程序清单中，我们将为图表bean开发一个定制器。这个定制器允许你在一个对话框中设置多个属性，如图8-13所示。

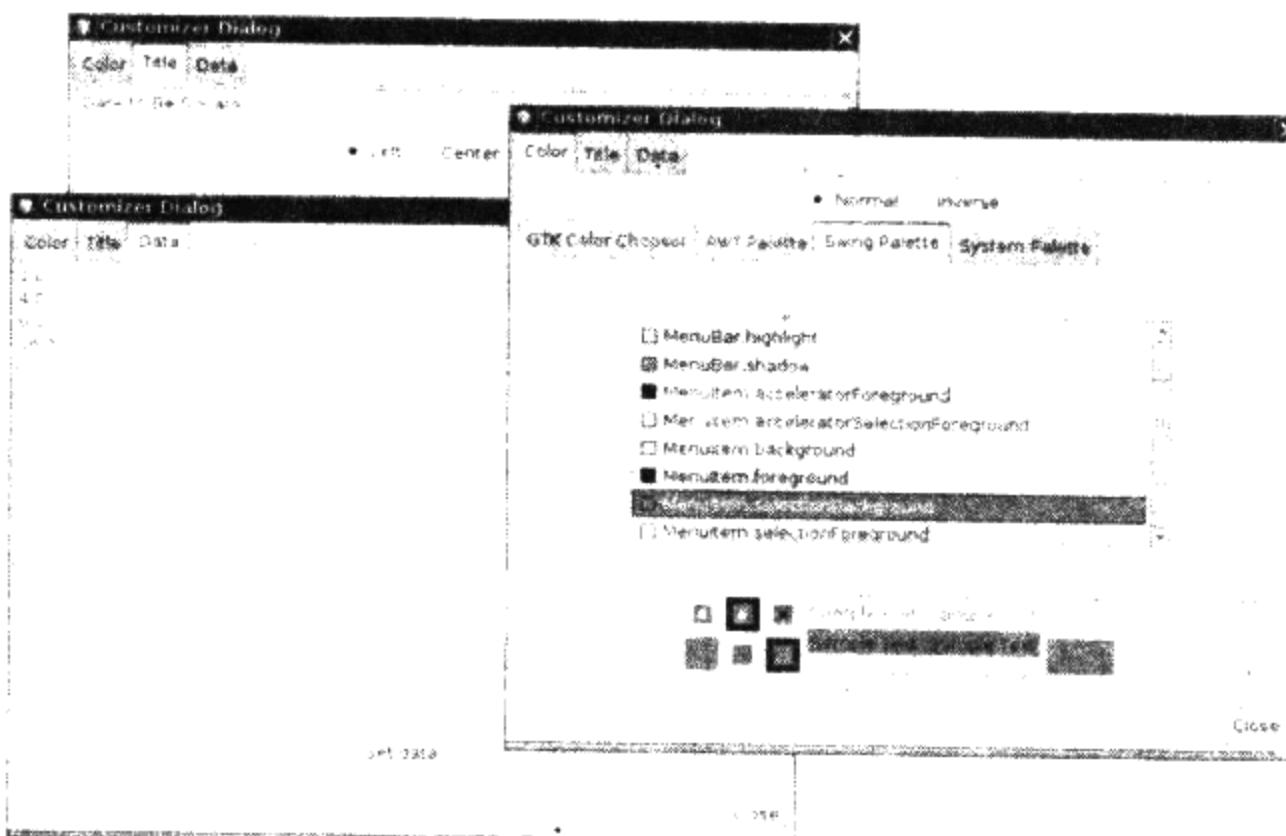


图8-13 ChartBean的定制器

要想为你的bean添加一个定制器，必须提供一个BeanInfo类，并且覆盖其getBeanDescriptor方法，例如：

```
public ChartBean2BeanInfo extends SimpleBeanInfo
{
    public BeanDescriptor getBeanDescriptor()
    {
        return beanDescriptor;
    }

    private BeanDescriptor beanDescriptor
        = new BeanDescriptor(ChartBean2.class, ChartBean2Customizer.class);
}
```

注意，定制器不需要任何命名模式。（不过，通常习惯于将定制器命名为BeanNameCustomizer。）

在下一节，你会看到如何实现一个定制器。

#### **API** `java.beans.BeanInfo 1.1`

- `BeanDescriptor getBeanDescriptor()`

返回一个BeanDescriptor对象，它描述了bean的特性。

#### **API** `java.beans.BeanDescriptor 1.1`

- `BeanDescriptor(Class <?> beanClass, Class <?> customizerClass)`

为具有定制器的bean构建一个BeanDescriptor对象。

参数：beanClass

bean对应的Class对象

customizerClass

bean的定制器对应的Class对象

## 编写一个定制器类

任何定制器类都必须有一个默认的构造器、扩展Component类，并实现Customizer接口。该接口只有三个方法：

- setObject方法，包含一个参数用于指定要被自定义的bean。
- addPropertyChangeListener和removePropertyChangeListener方法，用来管理监听器集合，当定制器中的属性发生变化时就会通知这些监听器。

只要是用户改变了某个属性，就应该通过广播一个PropertyChangeEvent事件来更新目标bean的可视化外观，而不要仅仅只在用户结束定制过程的时候才更新bean。这是个不错的主意。

与属性编辑器不同，定制器不会自动显示出来。在NetBeans中，必须右键点击bean，并且选择Customize菜单选项才会弹出定制器。在这一时刻，开发工具会调用定制器上的setObject方法。请注意，你的定制器就是这样被创建出来的，先创建定制器，然后再与你的bean建立实际的关联。因此，在定制器中，你不能假设拥有与bean状态有关的任何信息。而且，你必须提供一个没有参数的默认构造器。

因为定制器通常用来向用户表现多种选项，所以使用标签面板的用户界面一般比较便于使用。我们就是采用这样方法，使定制器继承自JTabbedPane类。

我们的定制器在三个面板中收集了以下信息：

- 图形颜色和反转模式；
- 标题与标题的位置；
- 数据点。

当然，开发这一类的用户界面可能很乏味，我们的例子用了100多行的代码，只是为了在构造器中装配界面。然而，这项工作只需要普通的Swing编程技术就够了，而我们也不想在这里多谈其中的细节。

有一个技巧很值得学习：你经常需要在一个定制器中编辑属性的值，与其为了设置特定类的某个属性而实现一个新的接口，不如直接找到一个已经存在的属性编辑器，然后把它添加到你的用户界面中。以我们的ChartBean2定制器为例，我们需要设置图形颜色。由于我们知道BeanBox已经有一个很好的颜色属性编辑器，于是我们可以找到它：

```
PropertyEditor colorEditor = PropertyEditorManager.findEditor(Color.class);
Component colorEditorComponent = colorEditor.getCustomEditor();
```

一旦我们排列好所有的构件，就可以在setObject方法中初始化它们的值。当定制器显示出来的时候，会调用setObject方法。它的参数是将要被定制的bean。接下来，我们保存了该bean的引用，在稍后通知bean属性有所改变时需要用到它。然后，我们初始化每一个用户界面构件。下面是图表bean的定制器的setObject方法的片断，由它完成初始化的工作。

```
public void setObject(Object obj)
{
    bean = (ChartBean2) obj;
    titleField.setText(bean.getTitle());
    colorEditor.setValue(bean.getGraphColor());
    ...
}
```

最后，我们要挂钩事件处理器，以跟踪用户的行动。无论何时，只要用户改变了某个构件的值，该构件就发出一个必须由我们的定制器来处理的事件。事件处理器必须更新属性的值，而且必须也发出一个PropertyChangeEvent事件，这样其他监听器（例如属性检查器）才能收到更新。接下来我们就按照这个过程，来完成图表bean定制器中的两个用户界面元素。

当用户输入一个新的标题时，我们希望更新title属性。于是我们给用户输入标题的文本域添加一个DocumentListener。

```
titleField.getDocument().addDocumentListener(new
    DocumentListener()
{
    public void changedUpdate(DocumentEvent event)
    {
        setTitle(titleField.getText());
    }
    public void insertUpdate(DocumentEvent event)
    {
        setTitle(titleField.getText());
    }
    public void removeUpdate(DocumentEvent event)
    {
        setTitle(titleField.getText());
    }
});
```

这个监听器的三个方法都调用定制器的setTitle方法。而setTitle方法要求bean更新属性的值，然后发出一个属性改变事件。（该更新仅对那些没有绑定的属性是必需的。）下面就是setTitle方法的代码。

```
public void setTitle(String newValue)
{
    if (bean == null) return;
    String oldValue = bean.getTitle();
    bean.setTitle(newValue);
    firePropertyChange("title", oldValue, newValue);
}
```

当颜色属性编辑器中的值发生变化时，我们希望更新bean的图形颜色。通过将一个监听器附着到属性编辑器上，我们就可以跟踪颜色的变化。容易令人混淆的是，编辑器本身也会发出属性改变的事件。

```
colorEditor.addPropertyChangeListener(new
    PropertyChangeListener()
{
    public void propertyChange(PropertyChangeEvent event)
    {
        setGraphColor((Color) colorEditor.getValue());
    }
});
```

程序清单8-7提供了图表bean定制器完整的代码。

### 程序清单8-7 ChartBean2Customizer.java

```
1. package com.horstmann.corejava;
```

```
2.
3. import java.awt.*;
4. import java.awt.event.*;
5. import java.beans.*;
6. import java.util.*;
7. import javax.swing.*;
8. import javax.swing.event.*;
9.
10. /**
11. * A customizer for the chart bean that allows the user to edit all chart properties in a
12. * single tabbed dialog.
13. * @version 1.12 2007-10-03
14. * @author Cay Horstmann
15. */
16. public class ChartBean2Customizer extends JTabbedPane implements Customizer
17. {
18.     public ChartBean2Customizer()
19.     {
20.         data = new JTextArea();
21.         JPanel dataPane = new JPanel();
22.         dataPane.setLayout(new BorderLayout());
23.         dataPane.add(new JScrollPane(data), BorderLayout.CENTER);
24.         JButton dataButton = new JButton("Set data");
25.         dataButton.addActionListener(new ActionListener()
26.             {
27.                 public void actionPerformed(ActionEvent event)
28.                 {
29.                     setData(data.getText());
30.                 }
31.             });
32.         JPanel panel = new JPanel();
33.         panel.add(dataButton);
34.         dataPane.add(panel, BorderLayout.SOUTH);
35.
36.         JPanel colorPane = new JPanel();
37.         colorPane.setLayout(new BorderLayout());
38.
39.         normal = new JRadioButton("Normal", true);
40.         inverse = new JRadioButton("Inverse", false);
41.         panel = new JPanel();
42.         panel.add(normal);
43.         panel.add(inverse);
44.         ButtonGroup group = new ButtonGroup();
45.         group.add(normal);
46.         group.add(inverse);
47.         normal.addActionListener(new ActionListener()
48.             {
49.                 public void actionPerformed(ActionEvent event)
50.                 {
51.                     setInverse(false);
52.                 }
53.             });
54.
55.         inverse.addActionListener(new ActionListener()
56.             {
57.                 public void actionPerformed(ActionEvent event)
58.                 {
59.                     setInverse(true);
```

```
60.        }
61.    });
62.
63.    colorEditor = PropertyEditorManager.findEditor(Color.class);
64.    colorEditor.addPropertyChangeListener(new PropertyChangeListener()
65.    {
66.        public void propertyChange(PropertyChangeEvent event)
67.        {
68.            setGraphColor((Color) colorEditor.getValue());
69.        }
70.    });
71.
72.    colorPane.add(panel, BorderLayout.NORTH);
73.    colorPane.add(colorEditor.getCustomEditor(), BorderLayout.CENTER);
74.
75.    JPanel titlePane = new JPanel();
76.    titlePane.setLayout(new BorderLayout());
77.
78.    group = new ButtonGroup();
79.    position = new JRadioButton[3];
80.    position[0] = new JRadioButton("Left");
81.    position[1] = new JRadioButton("Center");
82.    position[2] = new JRadioButton("Right");
83.
84.    panel = new JPanel();
85.    for (int i = 0; i < position.length; i++)
86.    {
87.        final ChartBean2.Position pos = ChartBean2.Position.values()[i];
88.        panel.add(position[i]);
89.        group.add(position[i]);
90.        position[i].addActionListener(new ActionListener()
91.        {
92.            public void actionPerformed(ActionEvent event)
93.            {
94.                setTitlePosition(pos);
95.            }
96.        });
97.    }
98.
99.    titleField = new JTextField();
100.   titleField.getDocument().addDocumentListener(new DocumentListener()
101.   {
102.       public void changedUpdate(DocumentEvent evt)
103.       {
104.           setTitle(titleField.getText());
105.       }
106.
107.       public void insertUpdate(DocumentEvent evt)
108.       {
109.           setTitle(titleField.getText());
110.       }
111.
112.       public void removeUpdate(DocumentEvent evt)
113.       {
114.           setTitle(titleField.getText());
115.       }
116.   });
117.
```

```
118.     titlePane.add(titleField, BorderLayout.NORTH);
119.     JPanel panel2 = new JPanel();
120.     panel2.add(panel);
121.     titlePane.add(panel2, BorderLayout.CENTER);
122.     addTab("Color", colorPane);
123.     addTab("Title", titlePane);
124.     addTab("Data", dataPane);
125.
126. }
127.
128. /**
129. * Sets the data to be shown in the chart.
130. * @param s a string containing the numbers to be displayed, separated by white space
131. */
132. public void setData(String s)
133. {
134.     StringTokenizer tokenizer = new StringTokenizer(s);
135.
136.     int i = 0;
137.     double[] values = new double[tokenizer.countTokens()];
138.     while (tokenizer.hasMoreTokens())
139.     {
140.         String token = tokenizer.nextToken();
141.         try
142.         {
143.             values[i] = Double.parseDouble(token);
144.             i++;
145.         }
146.         catch (NumberFormatException e)
147.         {
148.         }
149.     }
150.     setValues(values);
151. }
152.
153. /**
154. * Sets the title of the chart.
155. * @param newValue the new title
156. */
157. public void setTitle(String newValue)
158. {
159.     if (bean == null) return;
160.     String oldValue = bean.getTitle();
161.     bean.setTitle(newValue);
162.     firePropertyChange("title", oldValue, newValue);
163. }
164.
165. /**
166. * Sets the title position of the chart.
167. * @param i the new title position (ChartBean2.LEFT, ChartBean2.CENTER, or ChartBean2.RIGHT)
168. */
169. public void setTitlePosition(ChartBean2.Position pos)
170. {
171.     if (bean == null) return;
172.     ChartBean2.Position oldValue = bean.getTitlePosition();
173.     bean.setTitlePosition(pos);
174.     firePropertyChange("titlePosition", oldValue, pos);
175. }
```

```
176.  
177.    /**  
178.     * Sets the inverse setting of the chart.  
179.     * @param b true if graph and background color are inverted  
180.     */  
181.    public void setInverse(boolean b)  
182.    {  
183.        if (bean == null) return;  
184.        boolean oldValue = bean.isInverse();  
185.        bean.setInverse(b);  
186.        firePropertyChange("inverse", oldValue, b);  
187.    }  
188.  
189.    /**  
190.     * Sets the values to be shown in the chart.  
191.     * @param newValue the new value array  
192.     */  
193.    public void setValues(double[] newValue)  
194.    {  
195.        if (bean == null) return;  
196.        double[] oldValue = bean.getValues();  
197.        bean.setValues(newValue);  
198.        firePropertyChange("values", oldValue, newValue);  
199.    }  
200.  
201.    /**  
202.     * Sets the color of the chart  
203.     * @param newValue the new color  
204.     */  
205.    public void setGraphColor(Color newValue)  
206.    {  
207.        if (bean == null) return;  
208.        Color oldValue = bean.getGraphColor();  
209.        bean.setGraphColor(newValue);  
210.        firePropertyChange("graphColor", oldValue, newValue);  
211.    }  
212.  
213.    public void setObject(Object obj)  
214.    {  
215.        bean = (ChartBean2) obj;  
216.  
217.        data.setText("");  
218.        for (double value : bean.getValues())  
219.            data.append(value + "\n");  
220.  
221.        normal.setSelected(!bean.isInverse());  
222.        inverse.setSelected(bean.isInverse());  
223.  
224.        titleField.setText(bean.getTitle());  
225.  
226.        for (int i = 0; i < position.length; i++)  
227.            position[i].setSelected(i == bean.getTitlePosition().ordinal());  
228.  
229.        colorEditor.setValue(bean.getGraphColor());  
230.    }  
231.  
232.    private ChartBean2 bean;  
233.    private PropertyEditor colorEditor;
```

```
234  
235     private JTextArea data;  
236     private JRadioButton normal;  
237     private JRadioButton inverse;  
238     private JRadioButton[] position;  
239     private JTextField titleField;  
240 }
```

### API java.beans.Customizer 1.1

- void setObject(Object bean)

指定要定制的bean。

## 8.9 JavaBean持久化

JavaBean持久化是指以流的形式，用JavaBean的属性来保存bean，并在之后的某个时刻，或者在另一个虚拟机中再将它们读回来。在这一点上，JavaBean的持久化与对象序列化相似。（有关序列化的更多内容参见本书第1章。）然而，它们有一个非常重要的区别：JavaBean持久化适合于长期存储。

当一个对象被序列化后，它的实例域都被写到了一个流中。如果类的实现发生了变化，它的实例域也可能会产生变化，这样就无法直接读取包含了旧版中被序列化的对象的文件。探测到版本的不同，然后在新旧数据表现之间进行转换是可以实现的。然而，这个过程非常令人厌烦，而且应该只有在无路可走的情况下才使用这种手段。简单地说，序列化不适合长期存储。基于此原因，所有的Swing构件在其文档中都有如下的信息：“警告：该类的序列化对象与未来版本的Swing不兼容。当前对序列化的支持仅适合于短期存储或者应用程序之间的RMI。”

为了解决此问题，人们发明了长期存储机制。不过最初是为了拖拽GUI设计工具。设计工具为了保存鼠标点击窗体、面板、按钮以及其他Swing构件的结果，于是将此结果以长期持久化的形式保存在一个文件中。运行中的程序会直接打开该文件。这个方法减少了令人厌烦的布局、装备Swing构件的代码。可惜的是，它并没有广泛地实现。

 注意：<http://bean-builder.dev.java.net> 的Bean Builder是一个实验性的GUI开发工具，它支持长期持久化。

JavaBean持久化背后的想法非常简单。假设你想将一个JFrame对象保存到一个文件中，以便以后可以将它取回。如果看一看JFrame类及其父类的源代码，你会看到几十个实例域。如果窗体被序列化，那么所有这些域的值都需要写下来。但是想一想窗体是如何构造的吧：

```
JFrame frame = new JFrame();  
frame.setTitle("My Application");  
frame.setVisible(true);
```

默认的构造器初始化所有的实例域，然后由你设置一两个属性。如果你保存这个窗体对象，JavaBean持久化机制会精确地将这些语句以XML的形式保存下来：

```
<object class="javax.swing.JFrame">  
  <void property="title">  
    <string>My Application</string>
```

```

</void>
<void property="visible">
  <boolean>true</boolean>
</void>
</object>

```

当该对象被读回的时候，下面这些语句会被执行：构造一个JFrame对象，将它的title和visible设置为指定值。它完全不管JFrame的内部表示是否发生了变化。它只关心你可以通过设置属性来存储该对象。

请注意，只有那些与默认值不同的属性会保存下来。XMLEncoder制作了一个默认的JFrame，并将它的属性与被处理的窗体做比较。只有那些与默认值不同的属性，才会生成属性设置语句。这个过程称为消除冗余（redundancy elimination）。结果是，存储下来的东西通常比序列化的结果要小。（尤其是序列化Swing构件时，区别更是非常明显，因为Swing对象有很多状态，大多数从生成默认值的一刻起就从未改变过。）

当然，此方法还有一些小的技术障碍。例如，

```
frame.setSize(600, 400);
```

这并不是属性设置方法，然而XMLEncoder能够处理它，它会编写这样的语句。

```

<void property="bounds">
  <object class="java.awt.Rectangle">
    <int>0</int>
    <int>0</int>
    <int>600</int>
    <int>400</int>
  </object>
</void>

```

要将一个对象保存为流，需要使用XMLEncoder：

```

XMLEncoder out = new XMLEncoder(new FileOutputStream(. . .));
out.writeObject(frame);
out.close();

```

要将它读回来，需要使用XMLDecoder：

```

XMLDecoder in = new XMLDecoder(new FileInputStream(. . .));
JFrame newFrame = (JFrame) in.readObject();
in.close();

```

程序清单8-8中的程序演示了一个窗体是如何读取以及保存它自己的（参见图8-14）。当你运行该程序时，首先点击Save按钮，将此窗体保存为一个文件。然后移动原始的窗体到另一个位置，再点击Load按钮，你会看到在原来的位置弹出了另一个窗体。我们再来看一看该程序生成的XML文件。

如果仔细观察输出的XML文件，你会发现保存窗体时，XMLEncoder完成了大量的工作。XMLEncoder生成的语句完成了以下操作：

- 设置各种不同的窗体属性：size、layout、defaultCloseOperation、title等等。



图8-14 PersistentFrameTest程序

- 向窗体添加按钮。
- 为按钮添加动作监听器。

在这里，我们必须使用**EventHandler**类构建动作监听器。因为**XMLEncoder**不能保存任何内部类，但是它知道如何处理**EventHandler**对象。

### 程序清单8-8 PersistentFrameTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.beans.*;
4. import java.io.*;
5. import javax.swing.*;
6.
7. /**
8. * This program demonstrates the use of an XML encoder and decoder to save and restore a frame.
9. * @version 1.01 2007-10-03
10. * @author Cay Horstmann
11. */
12. public class PersistentFrameTest
13. {
14.     public static void main(String[] args)
15.     {
16.         chooser = new JFileChooser();
17.         chooser.setCurrentDirectory(new File("."));
18.         PersistentFrameTest test = new PersistentFrameTest();
19.         test.init();
20.     }
21.
22.     public void init()
23.     {
24.         frame = new JFrame();
25.         frame.setLayout(new FlowLayout());
26.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27.         frame.setTitle("PersistentFrameTest");
28.         frame.setSize(400, 200);
29.
30.         JButton loadButton = new JButton("Load");
31.         frame.add(loadButton);
32.         loadButton.addActionListener(EventHandler.create(ActionListener.class, this, "load"));
33.
34.         JButton saveButton = new JButton("Save");
35.         frame.add(saveButton);
36.         saveButton.addActionListener(EventHandler.create(ActionListener.class, this, "save"));
37.
38.         frame.setVisible(true);
39.     }
40.
41.     public void load()
42.     {
43.         // show file chooser dialog
44.         int r = chooser.showOpenDialog(null);
45.
46.         // if file selected, open
47.         if(r == JFileChooser.APPROVE_OPTION)
48.     }
```

```
49.     try
50.     {
51.         File file = chooser.getSelectedFile();
52.         XMLDecoder decoder = new XMLDecoder(new FileInputStream(file));
53.         decoder.readObject();
54.         decoder.close();
55.     }
56.     catch (IOException e)
57.     {
58.         JOptionPane.showMessageDialog(null, e);
59.     }
60. }
61. }
62.
63. public void save()
64. {
65.     if (chooser.showSaveDialog(null) == JFileChooser.APPROVE_OPTION)
66.     {
67.         try
68.         {
69.             File file = chooser.getSelectedFile();
70.             XMLEncoder encoder = new XMLEncoder(new FileOutputStream(file));
71.             encoder.writeObject(frame);
72.             encoder.close();
73.         }
74.         catch (IOException e)
75.         {
76.             JOptionPane.showMessageDialog(null, e);
77.         }
78.     }
79. }
80.
81. private static JFileChooser chooser;
82. private JFrame frame;
83. }
```

### 8.9.1 JavaBean持久化可用于任何数据

JavaBean持久化并不局限于Swing构件的存储。可以使用该机制存储任意对象的集合，只要遵守一些简单的规则即可。在下一节中，将学习如何将JavaBean持久化作为一种长期的存储形式来保存你自己的数据。

XMLEncoder内置了对下列类型的支持：

- null
- 所有基本类型及其包装器类型
- 枚举（从Java SE 6开始）
- String
- 数组
- 集合与映射表
- 反射类型Class、Field、Method和Proxy
- AWT类型Color、Cursor、Dimension、Font、Insets、Point、Rectangle和

### ImageIcon

- AWT和Swing构件、边界、布局管理器和模型
- 事件处理器

#### 1. 编写一个持久化代理 (Persistence Delegate) 来构造对象

如果你原本就可以通过设置属性得到每个对象的状态，那么使用JavaBean持久化就没什么意义了。但是，在真实的程序中，总有些类无法这样工作。例如，思考一下卷I第4章中的Employee类。Employee并不是一个循规蹈矩的bean。它没有默认的构造器，而且它也没有setName、setSalary、setHireDay方法。要克服这个问题，可以安装一个持久化代理(persistence delegate)，该代理负责生成一种对象的XML编码机制。

Employee类的持久化代理覆写了instantiate方法，生成一个构造对象的表达式。

```
PersistenceDelegate delegate = new
    DefaultPersistenceDelegate()
{
    protected Expression instantiate(Object oldInstance, Encoder out)
    {
        Employee e = (Employee) oldInstance;
        GregorianCalendar c = new GregorianCalendar();
        c.setTime(e.getHireDay());
        return new Expression(oldInstance, Employee.class, "new",
            new Object[]
            {
                e.getName(),
                e.getSalary(),
                c.get(Calendar.YEAR),
                c.get(Calendar.MONTH),
                c.get(Calendar.DATE)
            });
    }
};
```

这意味着：“要重新生成oldInstance，可以调用Employee.class对象上的new方法（例如构造器），并提供指定的参数。”名为oldInstance的参数容易令人误解——它就是被保存的实例。

安装持久性代理有两种选择：一种是将其与某个具体的XMLWriter相关联：

```
out.setPersistenceDelegate(Employee.class, delegate);
```

或者，可以设置Beaninfo的bean描述符的persistenceDelegate属性：

```
BeanInfo info = Introspector.getBeanInfo(GregorianCalendar.class);
info.getBeanDescriptor().setValue("persistenceDelegate", delegate);
```

一旦安装了代理，就可以保存Employee对象了。例如：

```
Object myData = new Employee("Harry Hacker", 50000, 1989, 10, 1);
out.writeObject(myData);
```

该语句会生成如下的输出：

```
<object class="Employee">
    <string>Harry Hacker</string>
    <double>50000.0</double>
    <int>1989</int>
    <int>10</int>
```

```
<int>1</int>
</object>
```

 注意：你只需要控制编码的过程，并没有任何专门的解码方法。解码器将直接执行在输入的XML中所找到的语句和表达式。

## 2. 由属性构造对象

如果构造器所需的参数都可以通过访问oldInstance的属性获得，那你就不必自己编写instantiate方法了。你可以直接构造一个DefaultPersistenceDelegate，然后提供属性的名字即可。

例如，下面的语句为Rectangle2D.Double类设置了持久化代理：

```
out.setPersistenceDelegate(Rectangle2D.Double.class,
    new DefaultPersistenceDelegate(new String[] { "x", "y", "width", "height" }));
```

它告诉编码器：“要编码一个Rectangle2D.Double对象，就要取得它的x、y、width和height属性，并且用它们来调用构造器。”包含了一个元素的输出如下所示：

```
<object class="java.awt.geom.Rectangle2D$Double">
    <double>5.0</double>
    <double>10.0</double>
    <double>20.0</double>
    <double>30.0</double>
</object>
```

如果你是类的作者，那么你甚至可以做得更好，即用@ConstructorProperties注解构造器。例如，假设Employee类有一个接收三个参数的构造器（名字、薪水和受雇日），然后我们对其进行注解：

```
@ConstructorProperties({"name", "salary", "hireDay"})
public Employee(String n, double s, Date d)
```

这可以告诉编码器去调用getName、getSalary和getHireDay这三个属性获取方法，并将得到的值写入object表达式。

@ConstructorProperties注解是在Java SE 6中引入的，因此只在Java管理扩展（JMX）API中得到了应用。

## 3. 使用工厂方法构造对象

有时，你需要保存那些通过工厂方法获得的对象，而不是通过构造器产生的对象。举例来说，思考一下你是如何得到一个InetAddress对象的：

```
byte[] bytes = new byte[] { 127, 0, 0, 1 };
InetAddress address = InetAddress.getByAddress(bytes);
```

PersistenceDelegate的instantiate方法产生对工厂方法的一次调用。

```
protected Expression instantiate(Object oldInstance, Encoder out)
{
    return new Expression(oldInstance, InetAddress.class, "getByAddress",
        new Object[] { ((InetAddress) oldInstance).getAddress() });
}
```

下面是一个输出样例：

```

<object class="java.net.Inet4Address" method="getByAddress">
    <array class="byte" length="4">
        <void index="0">
            <byte>127</byte>
        </void>
        <void index="3">
            <byte>1</byte>
        </void>
    </array>
</object>

```

 **警告：**安装这样的代理必须是针对实体子类，例如Inet4Address，而不能是抽象的InetAddress类！

#### 4. 构造后的工作

某些类的状态是通过调用非属性设置方法<sup>⊖</sup>来建立的。你可以通过覆写DefaultPersistenceDelegate的initialize方法来处理这种状况。initialize方法会在instantiate方法之后调用。你可以编写一系列的语句，它们会被记录在保存下来的bean中。

例如，思考一下BitSet类。为了重新生成一个BitSet对象，你要设置原对象中存在的所有位。下面的initialize方法生成了必要的语句：

```

protected void initialize(Class<?> type, Object oldInstance, Object newInstance, Encoder out)
{
    super.initialize(type, oldInstance, newInstance, out);
    BitSet bs = (BitSet) oldInstance;
    for (int i = bs.nextSetBit(0); i >= 0; i = bs.nextSetBit(i + 1))
        out.writeStatement(new Statement(bs, "set", new Object[] { i, i + 1, true } ));
}

```

输出样例如下所示：

```

<object class="java.util.BitSet">
    <void method="set">
        <int>1</int>
        <int>2</int>
        <boolean>true</boolean>
    </void>
    <void method="set">
        <int>4</int>
        <int>5</int>
        <boolean>true</boolean>
    </void>
</object>

```

 **注意：**编写new Statement(bs, "set", new Object[] { i })会更有效，不过，如果那样的话，XMLWriter会使用一个空的名字来设置属性，而这样的语句很不好。

#### 5. 瞬态属性

偶尔的，类中会包含能够被XMLDecoder发现的带有getXxx和setXxx，但是你不希望在存档文件中包含该属性的值。为了禁止某个属性被存档，可以在属性描述符中将它标记为

<sup>⊖</sup> 即非Set Xxx方法。——编辑注

*transient*。例如，下面的语句把DamageReporter类（在下一节你将会看到这个类的细节）的removeMode属性标记成瞬时的。

```
BeanInfo info = Introspector.getBeanInfo(DamageReport.class);
for (PropertyDescriptor desc : info.getPropertyDescriptors())
    if (desc.getName().equals("removeMode"))
        desc.setValue("transient", Boolean.TRUE);
```

程序清单8-9中的程序演示了工作中的各种持久化代理。请记住，这个程序实际上是一个应用程序的反面教材，其中的很多类其实无需使用代理就可以存档保存。

### 程序清单8-9 PersistenceDelegateTest.java

```
1. import java.awt.geom.*;
2. import java.beans.*;
3. import java.net.*;
4. import java.util.*;
5.
6. /**
7. * This program demonstrates various persistence delegates.
8. * @version 1.01 2007-10-03
9. * @author Cay Horstmann
10. */
11. public class PersistenceDelegateTest
12. {
13.     public static class Point
14.     {
15.         @ConstructorProperties( { "x", "y" })
16.         public Point(int x, int y)
17.         {
18.             this.x = x;
19.             this.y = y;
20.         }
21.
22.         public int getX()
23.         {
24.             return x;
25.         }
26.
27.         public int getY()
28.         {
29.             return y;
30.         }
31.
32.         private final int x, y;
33.     }
34.
35.     public static void main(String[] args) throws Exception
36.     {
37.         PersistenceDelegate delegate = new PersistenceDelegate()
38.         {
39.             protected Expression instantiate(Object oldInstance, Encoder out)
40.             {
41.                 Employee e = (Employee) oldInstance;
42.                 GregorianCalendar c = new GregorianCalendar();
43.                 c.setTime(e.getHireDay());
```

```

44.         return new Expression(oldInstance, Employee.class, "new", new Object[] {
45.             e.getName(), e.getSalary(), c.get(Calendar.YEAR), c.get(Calendar.MONTH),
46.             c.get(Calendar.DATE) });
47.     }
48. };
49. BeanInfo info = Introspector.getBeanInfo(Employee.class);
50. info.getBeanDescriptor().setValue("persistenceDelegate", delegate);
51.
52. XMLEncoder out = new XMLEncoder(System.out);
53. out.setExceptionListener(new ExceptionListener()
54. {
55.     public void exceptionThrown(Exception e)
56.     {
57.         e.printStackTrace();
58.     }
59. });
60.
61. out.setPersistenceDelegate(Rectangle2D.Double.class, new DefaultPersistenceDelegate(
62.     new String[] { "x", "y", "width", "height" }));
63.
64. out.setPersistenceDelegate(InetAddress.class, new DefaultPersistenceDelegate()
65. {
66.     protected Expression instantiate(Object oldInstance, Encoder out)
67.     {
68.         return new Expression(oldInstance, InetAddress.class, "getByAddress",
69.             new Object[] { ((InetAddress) oldInstance).getAddress() });
70.     }
71. });
72.
73. out.setPersistenceDelegate(BitSet.class, new DefaultPersistenceDelegate()
74. {
75.     protected void initialize(Class<?> type, Object oldInstance, Object newInstance,
76.         Encoder out)
77.     {
78.         super.initialize(type, oldInstance, newInstance, out);
79.         BitSet bs = (BitSet) oldInstance;
80.         for (int i = bs.nextSetBit(0); i >= 0; i = bs.nextSetBit(i + 1))
81.             out.writeStatement(new Statement(bs, "set",
82.                 new Object[] { i, i + 1, true }));
83.     }
84. });
85.
86. out.writeObject(new Employee("Harry Hacker", 50000, 1989, 10, 1));
87. out.writeObject(new Point(17, 29));
88. out.writeObject(new java.awt.geom.Rectangle2D.Double(5, 10, 20, 30));
89. out.writeObject(InetAddress.getLocalHost());
90. BitSet bs = new BitSet();
91. bs.set(1, 4);
92. bs.clear(2, 3);
93. out.writeObject(bs);
94. out.close();
95. }
96. }

```

### 8.9.2 一个JavaBean持久化的完整示例

我们将以一个完整的示例来结束有关JavaBean持久化的内容（参见图8-15）。这个应用程序为出租车编写受损报告。出租车的代理商负责填写出租记录，选择车的类型，用鼠标来点击轿车的受损区域，然后保存该报告。这个应用程序还可以读取已经保存的受损报告。程序清单8-10是该程序的代码。

这个应用程序使用了JavaBean持久化机制来保存和读取DamageReport对象（见程序清单8-11）。它演示了持久化机制的几个方面：

- 属性是自动保存并存储下来的。对于rentalRecord和carType属性而言，不需要再做任何事情。
- 构造后的工作是存储受损的位置。此持久化代理生成了调用click方法的语句。
- Point2D.Double类需要一个DefaultPersistenceDelegate，它负责通过x和y属性来构造一个点。
- removeMode属性（它指明了鼠标点击是在增加还是在移除受损标志）是瞬态的，因此它不应该被保存在受损报告中。

这里有一份受损报告的例子：

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.5.0" class="java.beans.XMLDecoder">
    <object class="DamageReport">
        <object class="java.lang.Enum" method="valueOf">
            <class>DamageReport$CarType</class>
            <string>SEDAN</string>
        </object>
        <void property="rentalRecord">
            <string>12443-19</string>
        </void>
        <void method="click">
            <object class="java.awt.geom.Point2D$Double">
                <double>181.0</double>
                <double>84.0</double>
            </object>
        </void>
        <void method="click">
            <object class="java.awt.geom.Point2D$Double">
                <double>162.0</double>
                <double>66.0</double>
            </object>
        </void>
    </object>
</java>
```

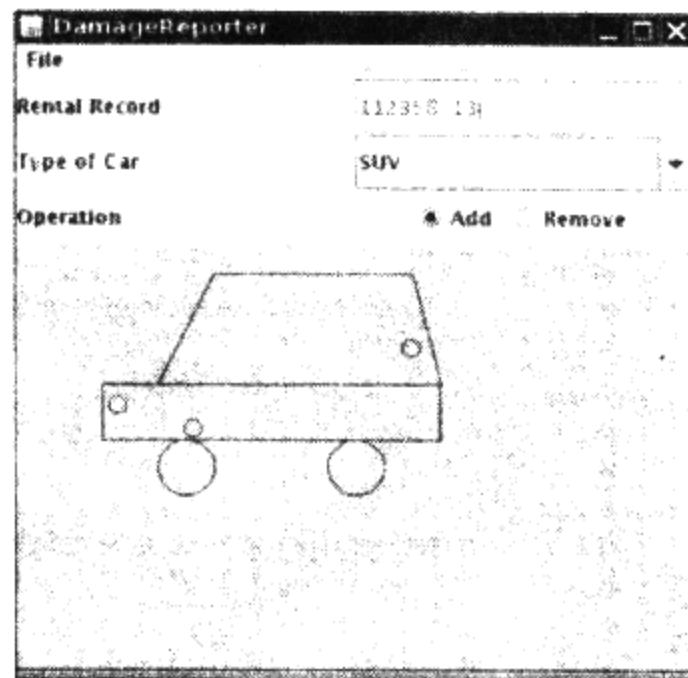


图8-15 受损报告应用程序



**注意：**这个示例程序并没有使用JavaBean持久化来保存应用程序的GUI。开发工具的创造者也许会对它更感兴趣，不过我们在这里讨论的是如何使用持久化机制来存储应用程序的数据。

我们以这个例子来结束JavaBean持久化机制的讨论。下面总结一下，JavaBean持久化的存档是：

- 适合长期存储的。
- 小而且快的。
- 易于生成的。
- 易于人们进行编辑的。
- 是标准Java的一部分。

#### 程序清单8-10 DamageReporter.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.awt.geom.*;
4. import java.beans.*;
5. import java.io.*;
6. import java.util.*;
7. import javax.swing.*;

8.
9. /**
10. * This program demonstrates the use of an XML encoder and decoder. All GUI and drawing
11. * code is collected in this class. The only interesting pieces are the action listeners for
12. * openItem and saveItem. Look inside the DamageReport class for encoder customizations.
13. * @version 1.01 2004-10-03
14. * @author Cay Horstmann
15. */
16. public class DamageReporter extends JFrame
17. {
18.     public static void main(String[] args)
19.     {
20.         JFrame frame = new DamageReporter();
21.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22.         frame.setVisible(true);
23.     }
24.
25.     public DamageReporter()
26.     {
27.         setTitle("DamageReporter");
28.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
29.
30.         chooser = new JFileChooser();
31.         chooser.setCurrentDirectory(new File("."));
32.
33.         report = new DamageReport();
34.         report.setCarType(DamageReport.CarType.SEDAN);
35.
36.         // set up the menu bar
37.         JMenuBar menuBar = new JMenuBar();
38.         setJMenuBar(menuBar);
```

```
39.     JMenu menu = new JMenu("File");
40.     menuBar.add(menu);
41.
42.
43.     JMenuItem openItem = new JMenuItem("Open");
44.     menu.add(openItem);
45.     openItem.addActionListener(new ActionListener()
46.     {
47.         public void actionPerformed(ActionEvent evt)
48.         {
49.             // show file chooser dialog
50.             int r = chooser.showOpenDialog(null);
51.
52.             // if file selected, open
53.             if (r == JFileChooser.APPROVE_OPTION)
54.             {
55.                 try
56.                 {
57.                     File file = chooser.getSelectedFile();
58.                     XMLDecoder decoder = new XMLDecoder(new FileInputStream(file));
59.                     report = (DamageReport) decoder.readObject();
60.                     decoder.close();
61.                     rentalRecord.setText(report.getRentalRecord());
62.                     carType.setSelectedItem(report.getCarType());
63.                     repaint();
64.                 }
65.                 catch (IOException e)
66.                 {
67.                     JOptionPane.showMessageDialog(null, e);
68.                 }
69.             }
70.         }
71.     });
72.
73.     JMenuItem saveItem = new JMenuItem("Save");
74.     menu.add(saveItem);
75.     saveItem.addActionListener(new ActionListener()
76.     {
77.         public void actionPerformed(ActionEvent evt)
78.         {
79.             report.setRentalRecord(rentalRecord.getText());
80.             chooser.setSelectedFile(new File(rentalRecord.getText() + ".xml"));
81.
82.             // show file chooser dialog
83.             int r = chooser.showSaveDialog(null);
84.
85.             // if file selected, save
86.             if (r == JFileChooser.APPROVE_OPTION)
87.             {
88.                 try
89.                 {
90.                     File file = chooser.getSelectedFile();
91.                     XMLEncoder encoder = new XMLEncoder(new FileOutputStream(file));
92.                     report.configureEncoder(encoder);
93.                     encoder.writeObject(report);
94.                     encoder.close();
95.                 }
96.             }
97.         }
98.     });
99.
```

```
96.             catch (IOException e)
97.             {
98.                 JOptionPane.showMessageDialog(null, e);
99.             }
100.         }
101.     });
102. );
103.
104. JMenuItem exitItem = new JMenuItem("Exit");
105. menu.add(exitItem);
106. exitItem.addActionListener(new ActionListener()
107. {
108.     public void actionPerformed(ActionEvent event)
109.     {
110.         System.exit(0);
111.     }
112. });
113.
114. // combo box for car type
115. rentalRecord = new JTextField();
116. carType = new JComboBox();
117. carType.addItem(DamageReport.CarType.SEDAN);
118. carType.addItem(DamageReport.CarType.WAGON);
119. carType.addItem(DamageReport.CarType.SUV);
120.
121. carType.addActionListener(new ActionListener()
122. {
123.     public void actionPerformed(ActionEvent event)
124.     {
125.         DamageReport.CarType item = (DamageReport.CarType) carType.getSelectedItem();
126.         report.setCarType(item);
127.         repaint();
128.     }
129. });
130.
131. // component for showing car shape and damage locations
132. carComponent = new JComponent()
133. {
134.     public void paintComponent(Graphics g)
135.     {
136.         Graphics2D g2 = (Graphics2D) g;
137.         g2.setColor(new Color(0.9f, 0.9f, 0.45f));
138.         g2.fillRect(0, 0, getWidth(), getHeight());
139.         g2.setColor(Color.BLACK);
140.         g2.draw(shapes.get(report.getCarType()));
141.         report.drawDamage(g2);
142.     }
143. };
144. carComponent.addMouseListener(new MouseAdapter()
145. {
146.     public void mousePressed(MouseEvent event)
147.     {
148.         report.click(new Point2D.Double(event.getX(), event.getY()));
149.         repaint();
150.     }
151. });
152.
```

```
153.     // radio buttons for click action
154.     addButton = new JRadioButton("Add");
155.     removeButton = new JRadioButton("Remove");
156.     ButtonGroup group = new ButtonGroup();
157.     JPanel buttonPanel = new JPanel();
158.     group.add(addButton);
159.     buttonPanel.add(addButton);
160.     group.add(removeButton);
161.     buttonPanel.add(removeButton);
162.     addButton.setSelected(!report.getRemoveMode());
163.     removeButton.setSelected(report.getRemoveMode());
164.     addButton.addActionListener(new ActionListener()
165.     {
166.         public void actionPerformed(ActionEvent event)
167.         {
168.             report.setRemoveMode(false);
169.         }
170.     });
171.     removeButton.addActionListener(new ActionListener()
172.     {
173.         public void actionPerformed(ActionEvent event)
174.         {
175.             report.setRemoveMode(true);
176.         }
177.     });
178.
179.     // layout components
180.     JPanel gridPanel = new JPanel();
181.     gridPanel.setLayout(new GridLayout(0, 2));
182.     gridPanel.add(new JLabel("Rental Record"));
183.     gridPanel.add(rentalRecord);
184.     gridPanel.add(new JLabel("Type of Car"));
185.     gridPanel.add(carType);
186.     gridPanel.add(new JLabel("Operation"));
187.     gridPanel.add(buttonPanel);
188.
189.     add(gridPanel, BorderLayout.NORTH);
190.     add(carComponent, BorderLayout.CENTER);
191. }
192.
193. private JTextField rentalRecord;
194. private JComboBox carType;
195. private JComponent carComponent;
196. private JRadioButton addButton;
197. private JRadioButton removeButton;
198. private DamageReport report;
199. private JFileChooser chooser;
200.
201. private static final int DEFAULT_WIDTH = 400;
202. private static final int DEFAULT_HEIGHT = 400;
203.
204. private static Map<DamageReport.CarType, Shape> shapes =
205.     new EnumMap<DamageReport.CarType, Shape>(DamageReport.CarType.class);
206.
207. static
208. {
```

```
209.     int width = 200;
210.     int x = 50;
211.     int y = 50;
212.     Rectangle2D.Double body = new Rectangle2D.Double(x, y + width / 6, width - 1, width / 6);
213.     Ellipse2D.Double frontTire = new Ellipse2D.Double(x + width / 6, y + width / 3,
214.             width / 6, width / 6);
215.     Ellipse2D.Double rearTire = new Ellipse2D.Double(x + width * 2 / 3, y + width / 3,
216.             width / 6, width / 6);
217.
218.     Point2D.Double p1 = new Point2D.Double(x + width / 6, y + width / 6);
219.     Point2D.Double p2 = new Point2D.Double(x + width / 3, y);
220.     Point2D.Double p3 = new Point2D.Double(x + width * 2 / 3, y);
221.     Point2D.Double p4 = new Point2D.Double(x + width * 5 / 6, y + width / 6);
222.
223.     Line2D.Double frontWindshield = new Line2D.Double(p1, p2);
224.     Line2D.Double roofTop = new Line2D.Double(p2, p3);
225.     Line2D.Double rearWindshield = new Line2D.Double(p3, p4);
226.
227.     GeneralPath sedanPath = new GeneralPath();
228.     sedanPath.append(frontTire, false);
229.     sedanPath.append(rearTire, false);
230.     sedanPath.append(body, false);
231.     sedanPath.append(frontWindshield, false);
232.     sedanPath.append(roofTop, false);
233.     sedanPath.append(rearWindshield, false);
234.     shapes.put(DamageReport.CarType.SEDAN, sedanPath);
235.
236.     Point2D.Double p5 = new Point2D.Double(x + width * 11 / 12, y);
237.     Point2D.Double p6 = new Point2D.Double(x + width, y + width / 6);
238.     roofTop = new Line2D.Double(p2, p5);
239.     rearWindshield = new Line2D.Double(p5, p6);
240.
241.     GeneralPath wagonPath = new GeneralPath();
242.     wagonPath.append(frontTire, false);
243.     wagonPath.append(rearTire, false);
244.     wagonPath.append(body, false);
245.     wagonPath.append(frontWindshield, false);
246.     wagonPath.append(roofTop, false);
247.     wagonPath.append(rearWindshield, false);
248.     shapes.put(DamageReport.CarType.WAGON, wagonPath);
249.
250.     Point2D.Double p7 = new Point2D.Double(x + width / 3, y - width / 6);
251.     Point2D.Double p8 = new Point2D.Double(x + width * 11 / 12, y - width / 6);
252.     frontWindshield = new Line2D.Double(p1, p7);
253.     roofTop = new Line2D.Double(p7, p8);
254.     rearWindshield = new Line2D.Double(p8, p6);
255.
256.     GeneralPath suvPath = new GeneralPath();
257.     suvPath.append(frontTire, false);
258.     suvPath.append(rearTire, false);
259.     suvPath.append(body, false);
260.     suvPath.append(frontWindshield, false);
261.     suvPath.append(roofTop, false);
262.     suvPath.append(rearWindshield, false);
263.     shapes.put(DamageReport.CarType.SUV, suvPath);
264. }
```

265. }

### 程序清单8-11 DamageReport.java

```
1. import java.awt.*;
2. import java.awt.geom.*;
3. import java.beans.*;
4. import java.util.*;
5.
6. /**
7. * This class describes a vehicle damage report that will be saved and loaded with the
8. * long-term persistence mechanism.
9. * @version 1.21 2004-08-30
10. * @author Cay Horstmann
11. */
12. public class DamageReport
13. {
14.     public enum CarType
15.     {
16.         SEDAN, WAGON, SUV
17.     }
18.
19.     // this property is saved automatically
20.     public void setRentalRecord(String newValue)
21.     {
22.         rentalRecord = newValue;
23.     }
24.
25.     public String getRentalRecord()
26.     {
27.         return rentalRecord;
28.     }
29.
30.     // this property is saved automatically
31.     public void setCarType(CarType newValue)
32.     {
33.         carType = newValue;
34.     }
35.
36.     public CarType getCarType()
37.     {
38.         return carType;
39.     }
40.
41.     // this property is set to be transient
42.     public void setRemoveMode(boolean newValue)
43.     {
44.         removeMode = newValue;
45.     }
46.
47.     public boolean getRemoveMode()
48.     {
49.         return removeMode;
50.     }
51.
52.     public void click(Point2D p)
```

```
53. {
54.     if (removeMode)
55.     {
56.         for (Point2D center : points)
57.         {
58.             Ellipse2D circle = new Ellipse2D.Double(center.getX() - MARK_SIZE, center.getY()
59.                 - MARK_SIZE, 2 * MARK_SIZE, 2 * MARK_SIZE);
60.             if (circle.contains(p))
61.             {
62.                 points.remove(center);
63.                 return;
64.             }
65.         }
66.     }
67.     else points.add(p);
68. }
69.
70. public void drawDamage(Graphics2D g2)
71. {
72.     g2.setPaint(Color.RED);
73.     for (Point2D center : points)
74.     {
75.         Ellipse2D circle = new Ellipse2D.Double(center.getX() - MARK_SIZE, center.getY()
76.             - MARK_SIZE, 2 * MARK_SIZE, 2 * MARK_SIZE);
77.         g2.draw(circle);
78.     }
79. }
80.
81. public void configureEncoder(XMLEncoder encoder)
82. {
83.     // this step is necessary to save Point2D.Double objects
84.     encoder.setPersistenceDelegate(Point2D.Double.class, new DefaultPersistenceDelegate(
85.         new String[] { "x", "y" }));
86.
87.     // this step is necessary because the array list of points is not
88.     // (and should not be) exposed as a property
89.     encoder.setPersistenceDelegate(DamageReport.class, new DefaultPersistenceDelegate()
90.     {
91.         protected void initialize(Class<?> type, Object oldInstance, Object newInstance,
92.             Encoder out)
93.         {
94.             super.initialize(type, oldInstance, newInstance, out);
95.             DamageReport r = (DamageReport) oldInstance;
96.
97.             for (Point2D p : r.points)
98.                 out.writeStatement(new Statement(oldInstance, "click", new Object[] { p }));
99.         }
100.    });
101. }
102.
103. // this step is necessary to make the removeMode property transient
104. static
105. {
106.     try
107.     {
108.         BeanInfo info = Introspector.getBeanInfo(DamageReport.class);
```

```

110.     for (PropertyDescriptor desc : info.getPropertyDescriptors())
111.         if (desc.getName().equals("removeMode")) desc.setValue("transient", Boolean.TRUE);
112.     }
113.     catch (IntrospectionException e)
114.     {
115.         e.printStackTrace();
116.     }
117. }
118.
119. private String rentalRecord;
120. private CarType carType;
121. private boolean removeMode;
122. private ArrayList<Point2D> points = new ArrayList<Point2D>();
123.
124. private static final int MARK_SIZE = 5;
125. }

```

**API** **java.beans.XMLEncoder 1.4**

- **XMLEncoder(OutputStream out)**

构造一个XMLEncoder，它将其输出发送到指定的流。

- **void writeObject(Object obj)**

存档给定的对象。

- **void writeStatement(Statement stat)**

编写给定的语句来存档。该方法应该只在持久化代理中调用。

**API** **java.beans.Encoder 1.4**

- **void setPersistenceDelegate(Class<?> type, PersistenceDelegate delegate)**

- **PersistenceDelegate getPersistenceDelegate(Class<?> type)**

设置或获取指定类型存档对象的代理。

- **void setExceptionListener(ExceptionListener listener)**

- **ExceptionListener getExceptionListener()**

设置或获取异常监听器，该监听器在编码过程中发生异常时会收到通知。

**API** **java.beans.ExceptionListener 1.4**

- **void exceptionThrown(Exception e)**

在编码或解码期间，当异常被抛出时调用该方法。

**API** **java.beans.XMLDecoder 1.4**

- **XMLDecoder(InputStream in)**

构造一个XMLDecoder，它从制定的输入流中读取一个存档。

- **Object readObject()**

从存档中读取下一个对象。

- **void setExceptionListener(ExceptionListener listener)**

- **ExceptionListener getExceptionListener()**

设置或获取异常监听器，如果在编码过程中发生异常，该监听器会收到通知。

#### **java.beans.PersistenceDelegate 1.4**

- **protected abstract Expression instantiate(Object oldInstance, Encoder out)**  
为初始化对象返回一个表达式，该对象等价于oldInstance。
- **protected void initialize(Class<?> type, Object oldInstance, Object newInstance, Encoder out)**  
为out编写语句，将newInstance转换成一个等价于oldInstance的对象。

#### **java.beans.DefaultPersistenceDelegate 1.4**

- **DefaultPersistenceDelegate()**  
为带有零参数构造器的类构造一个持久化代理。
- **DefaultPersistenceDelegate(String[] propertyNames)**  
为一个类构造一个持久化代理，这个类的构造器参数正是给定的属性的值。
- **protected Expression instantiate(Object oldInstance, Encoder out)**  
返回一个用来调用构造器的表达式，该构造器要么不含参数，要么使用构造器中指定的属性值作为参数。
- **protected void initialize(Class<?> type, Object oldInstance, Object newInstance, Encoder out)**  
为out编写语句，它提供了newInstance的属性设置器，并试图将其转换为等价于oldInstance的对象。

#### **java.beans.Expression 1.4**

- **Expression(Object value, Object target, String methodName, Object[] parameters)**  
构造一个表达式，它使用给定的参数来调用target上的指定方法。表达式的结果将赋给value。如果想要调用一个构造器，那么target应该是一个Class对象，而且methodName应该是“new”。

#### **java.beans.Statement 1.4**

- **Statement(Object target, String methodName, Object[] parameters)**  
构造一个语句，使用给定的参数调用target上指定的方法。

你现在已经通读了篇幅很长的有关运用Swing、AWT和JavaBeans进行GUI编程的三章。在下一章，我们将转向完全不同的主题：安全。安全已经成为Java平台的核心特征之一。由于我们生存和计算的世界变得越来越危险，因此对Java安全地透彻理解对于许多开发者来说就显得尤为重要。

# 第9章 安全

- ▲ 类加载器
- ▲ 字节码校验
- ▲ 安全管理器与访问权限
- ▲ 用户认证

- ▲ 数字签名
- ▲ 代码签名
- ▲ 加密

当Java技术刚刚问世时，令人激动的并不是因为它是一个设计完美的编程语言，而是因为它能够安全地运行通过因特网传播的各种applet（有关applet的更多信息，参见卷I第10章）。很显然，只有当用户确信applet的代码不会破坏他的计算机时，用户才会接受在网上传播的可执行的applet。正因为如此，无论过去还是现在，安全都是设计人员和Java技术使用者所关心的一个重大问题。这就意味着，Java技术与其他的语言和系统有所不同，在那些语言和系统中安全是在事后才想到要去实现的，或者仅仅是对破坏的一种应对措施，而对Java技术来说，安全机制是一个不可分割的组成部分。

Java技术提供了以下三种确保安全的机制：

- 语言设计特性（对数组的边界进行检查，无不受检查类型的转换，无指针算法等）。
- 访问控制机制，用于控制代码能够执行的功能（比如文件访问，网络访问等）。
- 代码签名，利用该特性，代码的作者就能够用标准的加密算法来标明Java代码的身份。这样，该代码的使用者就能够准确地知道谁创建了该代码，以及代码被标识后是否被修改过。

首先，我们来讨论类加载器，它可以在将类加载到虚拟机中的时候检查类的完整性。我们将展示这种机制是如何探测类文件中的损坏的。

为了获得最大的安全性，无论是加载类的默认机制，还是自定义的类加载器，都需要与负责控制代码运行的安全管理器类协同工作。后面我们还要详细介绍如何配置Java平台的安全性。

最后，我们要介绍java.security包提供的加密算法，来进行代码的标识和用户身份认证。与我们的一贯宗旨一样，我们将重点介绍应用程序编程人员最感兴趣的话题。如果要深入研究，推荐阅读Li Gong、Gary Ellison和Mary Dageforde撰写的《*Inside Java 2 Platform Security: Architecture, API Design, and Implementation*》一书，该书由Prentice Hall出版社于2003年出版。

## 9.1 类加载器

Java编译器会为虚拟机转换源指令。虚拟机代码存储在以.class为扩展名的类文件中。每个类文件都包含某个类或者接口的定义和代码实现。这些类文件必须由一个解释器进行解释，该

解释器能够将虚拟机的指令集翻译成目标机器的机器语言。

请注意，虚拟机只加载程序执行时所需要的类文件。例如，假设程序从MyProgram.class开始运行，下面是虚拟机执行的步骤：

- 1) 虚拟机有一个用于加载类文件的机制，例如，从磁盘上读取文件或者请求Web上的文件；它使用该机制来加载MyProgram类文件中的内容。
- 2) 如果MyProgram类拥有类型为另一个类的实例变量，或者是拥有超类，那么这些类文件也会被加载。（加载某个类所依赖的所有类的过程称为类的解析。）
- 3) 接着，虚拟机执行MyProgram中的main方法（它是静态的，无需创建类的实例）。
- 4) 如果main方法或者main调用的方法要用到更多的类，那么接下来就会加载这些类。

然而，类加载机制并非只使用单个的类加载器。每个Java程序至少拥有三个类加载器：

- 引导类加载器
- 扩展类加载器
- 系统类加载器（有时也称为应用类加载器）

引导类加载器负责加载系统类（通常从JAR文件rt.jar中进行加载）。它是虚拟机整体中的一部分，而且通常是用C语言来实现的。引导类加载器没有对应的ClassLoader对象，例如，该方法：

```
String.class.getClassLoader()
```

将返回null。

扩展类加载器用于从jre/lib/ext目录加载“标准的扩展”。可以将JAR文件放入该目录，这样即使没有任何类路径，扩展类加载器也可以找到其中的各个类。（有些人推荐使用该机制来避免“可恶的类路径”，不过请看看下面提到的警告事项。）

系统类加载器用于加载应用类。它在由CLASSPATH环境变量或者-classpath命令行选项设置的类路径中的目录里或者是JAR/ZIP文件里查找这些类。

在Sun公司的Java语言实现中，扩展类加载器和系统类加载器都是用Java来实现的。它们都是URLClassLoader类的实例。

 **警告：**如果将JAR文件放入jre/lib/ext目录中，并且在它的类中有一个类需要调用系统类或者扩展类，那么就会遇到麻烦。扩展类加载器并不使用类路径。在使用扩展目录来解决类文件的冲突之前，要牢记这种情况。

 **注意：**除了所有已经提到的位置，还可以从jre/lib/endorsed目录中加载类。这种机制只能用于将某个标准的Java类库替换为更新的版本（例如那些支持XML和CORBA的类）。更多细节请查看<http://java.sun.com/javase/6/docs/technotes/guides/standards/index.html>。

### 9.1.1 类加载器的层次结构

类加载器有一种父/子关系。除了引导类加载器外，每个类加载器都有一个父类加载器。根据规定，类加载器会为它的父类加载器提供一个机会，以便加载任何给定的类，并且只有在其父类加载器加载失败时，它才会加载该给定类。例如，当要求系统类加载器加载一个系统类（比如，java.util.ArrayList）时，它首先要求扩展类加载器进行加载，该扩展类加载器则

首先要求引导类加载器进行加载。引导类加载器查找并加载rt.jar中的这个类，而无须其他类加载器做更多的搜索。

某些程序具有插件架构，其中代码的某些部分是作为可选的插件打包的。如果插件被打包为JAR文件，那就可以直接用URLClassLoader类的实例去加载这些类。

```
URL url = new URL("file:///path/to/plugin.jar");
URLClassLoader pluginLoader = new URLClassLoader(new URL[] { url });
Class<?> cl = pluginLoader.loadClass("mypackage.MyClass");
```

因为在URLClassLoader构造器中指定父类加载器，因此pluginLoader的父亲就是系统类加载器。图9-1展示了这种层次结构。

大多数时候，你不必操心类加载的层次结构。通常，类是由于其他的类需要它而被加载的，而这个过程对你是透明的。

偶尔，你也会需要干涉和指定类加载器。考虑下面的例子：

- 你的应用的代码包含一个助手方法，它要调用Class.forName(classNameString)。
- 这个方法是从一个插件类中被调用的。
- 而classNameString指定的正是一个包含在这个插件的JAR中的类。

插件的作者很合理地期望这个类应该被加载。但是，助手方法的类是由系统类加载器加载的，这正是Class.forName所使用的类加载器。而对于它来说，在插件JAR中的类是不可见的，这种现象称为类加载器倒置。

要解决这个问题，助手方法需要使用恰当的类加载器，它可以要求类加载器作为其一个参数。或者，它可以要求将恰当的类加载器设置成为当前线程的上下文类加载器，这种策略在许多框架中都得到了应用（例如我们在第2章和第4章讨论过的JAXP和JNDI框架）。

每个线程都有一个对类加载器的引用，称为上下文类加载器。主线程的上下文类加载器是系统类加载器。当新线程创建时，它的上下文类加载器会被设置成为创建线程的上下文类加载器。因此，如果你不做任何特殊的操作，那么所有线程就都将它们的上下文类加载器设置为系统类加载器。

但是，我们也可以通过下面的调用将其设置成为任何类加载器。

```
Thread t = Thread.currentThread();
t.setContextClassLoader(loader);
```

然后助手方法可以获取这个上下文类加载器：

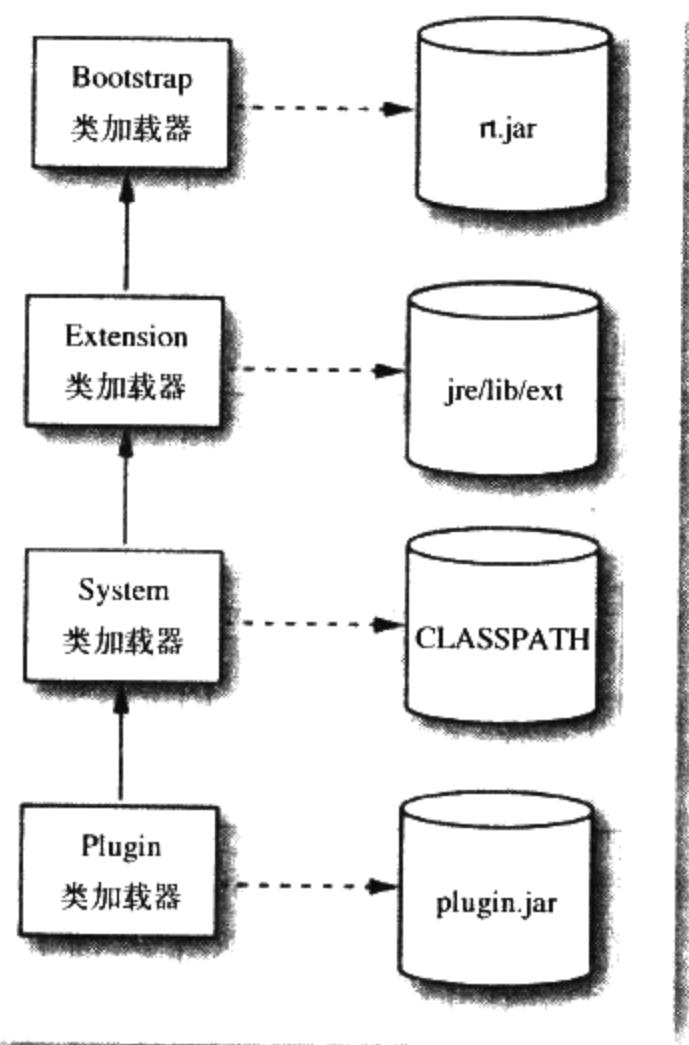


图9-1 类加载器的层次结构

```
Thread t = Thread.currentThread();
ClassLoader loader = t.getContextClassLoader();
Class c1 = loader.loadClass(className);
```

当上下文类加载器设置为插件类加载器时，问题依旧存在。应用设计者必须作出决策。通常，当调用由不同的类加载器加载的插件类的方法时，进行上下文类加载器的设置是一种好的思路。或者，助手方法的调用者也可以设置上下文类加载器。

**!** 提示：如果你编写了一个按名字来加载类的方法，那么让调用者在传递显式的类加载器和使用上下文类加载器之间进行选择就是一种好的做法。不要直接使用该方法所属的类的类加载器。

### 9.1.2 将类加载器作为命名空间

每个Java程序员都知道，包的命名是为了消除名字冲突。在标准类库中，有两个名为Date的类，它们的实际名字分别为java.util.Date和java.sql.Date。使用简单的名字只是为了方便程序员，它们要求程序包含恰当的import语句。在一个正在执行的程序中，所有的类名都包含它们的包名。

然而，令人惊奇的是，在同一个虚拟机中，可以有两个类，它们的类名和包名都是相同的。类是由它的全名和类加载器来确定的。这项技术在加载来自多处的代码时很有用。例如，浏览器为每一个web页都使用了一个独立的applet类加载器类的实例。这样，虚拟机就能区分来自不同web页的各个类，而不用管它们的名字是什么。图9-2展示了一个实例。假设有一个Web页面包含两个由不同的广告上提供的applet，其中每个applet都有一个称为Banner的类。因为每个applet都是由单独的类加载器加载的，因此这些类可以彻底地区分开而没有任何冲突。

**✓ 注意：**这种技术还有其他用处，例如servlets和EJB的“热部署”。详细信息请访问<http://java.sun.com/developer/TechTips/2000/tt1027.html>。

### 9.1.3 编写你自己的类加载器

我们可以编写自己的用于特殊目的的类加载器，这使得我们可以在向虚拟机传递字节码之前执行定制的检查。例如，我们可以编写一个类加载器，它可以拒绝加载没有标记为“paid for”的类。

如果要编写自己的类加载器，只需要继承ClassLoader类，然后覆盖下面这个方法

```
findClass(String className)
```

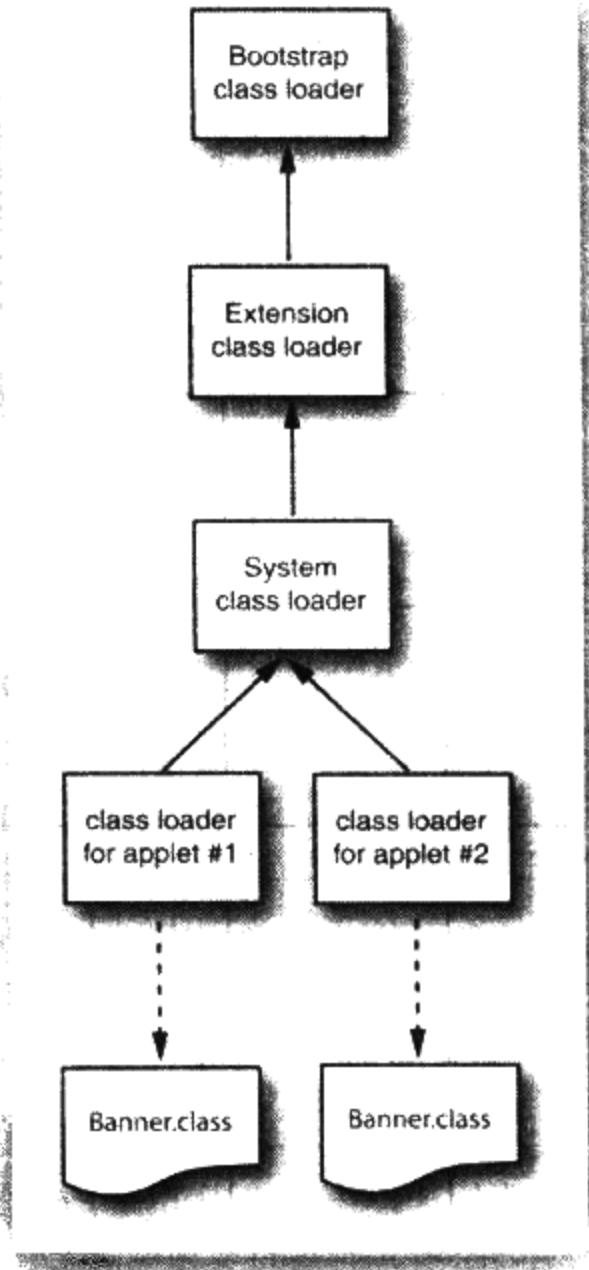


图9-2 两个类加载器分别加载具有相同名字的两个类

ClassLoader超类的loadClass方法用于将类的加载操作委托给其父类加载器去进行，只有当该类尚未加载并且父类加载器也无法加载该类时，才调用findClass方法。

如果要实现该方法，必须做到以下几点：

- 1) 为来自本地文件系统或者其他来源的类加载其字节码。
- 2) 调用ClassLoader超类的defineClass方法，向虚拟机提供字节码。

在程序清单9-3中，我们建立了一个类加载器，用于加载加密过的类文件。该程序要求用户输入第一个要加载的类的名字（即包含main方法的类）和密钥。然后，使用一个专门的类加载器来加载指定的类并调用main方法。该类加载器对指定的类和所有被其引用的非系统类进行解密。最后，该程序调用已加载类的main方法（参见图9-3）。

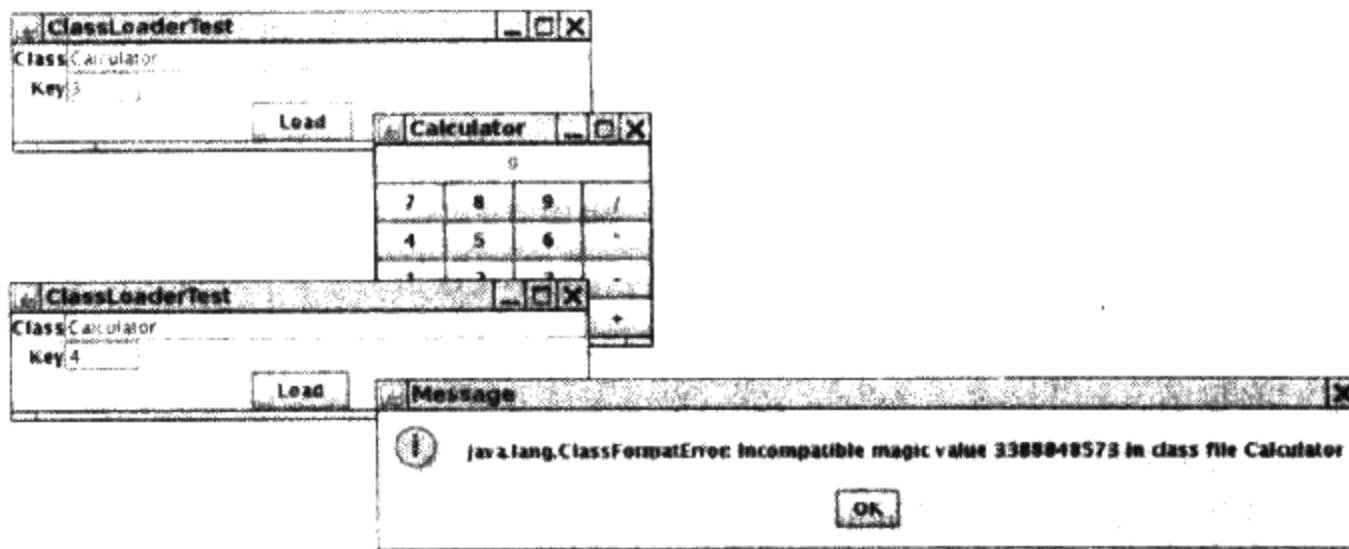


图9-3 ClassLoaderTest程序

为了简单起见，我们忽略了密码学领域2000年来所取得的技术进展，而是采用了传统的Caesar密码对类文件进行加密。

 **注意：**David Kahn的佳作《The Codebreakers》，纽约Macmillan出版社1967年出版，原书第84页，一书中称Suetonius是Caesar密码的发明人。Caesar将罗马字母表的24个字母移动了3个字母的位置，在那个时代这可以迷惑对手。

在撰写本书时，美国政府限制高强度加密方法的出口。因此，我们在实例中使用的是Caesar的加密方法，因为该方法的出口显然是合法的。

我们的Caesar密码版本使用的密钥是从1~255之间的一个数字，解密时，只需将密钥与每个字节相加，然后对256取余。程序清单9-2的Caesar.java程序就实现了这种加密行为。

为了不与通常的类加载器相混淆，我们对加密的类文件使用了不同的扩展名.caesar。

解密时，类加载器只需要将每个字节减去该密钥即可。在本书的程序代码中，可以找到4个类文件，它们都是用“3”这个传统的密钥值进行加密的。为了运行加密程序，需要使用在我们的ClassLoaderTest程序中定义的定制类加载器。

对类文件进行加密有很大的用途（当然，使用的密码的强度应该高于Caesar密码），如果没有加密密钥，类文件就毫无用处。它们既不能由标准虚拟机来执行，也不能轻易地被反汇编。

这就是说，可以使用定制的类加载器来认证类用户的身份，或者确保程序在运行之前已经

支付了软件费用。当然，加密只是定制类加载器的应用之一。可以使用其他类型的加载器来解决别的问题，例如，将类文件存储到数据库中。

### 程序清单9-1 ClassLoaderTest.java

```
1. import java.io.*;
2. import java.lang.reflect.*;
3. import java.awt.*;
4. import java.awt.event.*;
5. import javax.swing.*;
6.
7. /**
8. * This program demonstrates a custom class loader that decrypts class files.
9. * @version 1.22 2007-10-05
10. * @author Cay Horstmann
11. */
12. public class ClassLoaderTest
13. {
14.     public static void main(String[] args)
15.     {
16.         EventQueue.invokeLater(new Runnable()
17.         {
18.             public void run()
19.             {
20.
21.                 JFrame frame = new ClassLoaderFrame();
22.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23.                 frame.setVisible(true);
24.             }
25.         });
26.     }
27. }
28.
29. /**
30. * This frame contains two text fields for the name of the class to load and the decryption key.
31. */
32. class ClassLoaderFrame extends JFrame
33. {
34.     public ClassLoaderFrame()
35.     {
36.         setTitle("ClassLoaderTest");
37.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
38.         setLayout(new GridBagLayout());
39.         add(new JLabel("Class"), new GBC(0, 0).setAnchor(GBC.EAST));
40.         add(nameField, new GBC(1, 0).setWeight(100, 0).setAnchor(GBC.WEST));
41.         add(new JLabel("Key"), new GBC(0, 1).setAnchor(GBC.EAST));
42.         add(keyField, new GBC(1, 1).setWeight(100, 0).setAnchor(GBC.WEST));
43.         JButton loadButton = new JButton("Load");
44.         add(loadButton, new GBC(0, 2, 2, 1));
45.         loadButton.addActionListener(new ActionListener()
46.         {
47.             public void actionPerformed(ActionEvent event)
48.             {
49.                 runClass(nameField.getText(), keyField.getText());
50.             }
51.         });
52.     }
53. }
```

```
52.     pack();
53. }
54.
55. /**
56. * Runs the main method of a given class.
57. * @param name the class name
58. * @param key the decryption key for the class files
59. */
60. public void runClass(String name, String key)
61. {
62.     try
63.     {
64.         ClassLoader loader = new CryptoClassLoader(Integer.parseInt(key));
65.         Class<?> c = loader.loadClass(name);
66.         Method m = c.getMethod("main", String[].class);
67.         m.invoke(null, (Object) new String[] {});
68.     }
69.     catch (Throwable e)
70.     {
71.         JOptionPane.showMessageDialog(this, e);
72.     }
73. }
74.
75. private JTextField keyField = new JTextField("3", 4);
76. private JTextField nameField = new JTextField("Calculator", 30);
77. private static final int DEFAULT_WIDTH = 300;
78. private static final int DEFAULT_HEIGHT = 200;
79. }
80.
81. /**
82. * This class loader loads encrypted class files.
83. */
84. class CryptoClassLoader extends ClassLoader
85. {
86.     /**
87.      * Constructs a crypto class loader.
88.      * @param k the decryption key
89.      */
90.     public CryptoClassLoader(int k)
91.     {
92.         key = k;
93.     }
94.
95.     protected Class<?> findClass(String name) throws ClassNotFoundException
96.     {
97.         byte[] classBytes = null;
98.         try
99.         {
100.             classBytes = loadClassBytes(name);
101.         }
102.         catch (IOException e)
103.         {
104.             throw new ClassNotFoundException(name);
105.         }
106.
107.         Class<?> c1 = defineClass(name, classBytes, 0, classBytes.length);
108.         if (c1 == null) throw new ClassNotFoundException(name);
```

```
109     return c1;
110 }
111
112 /**
113 * Loads and decrypt the class file bytes.
114 * @param name the class name
115 * @return an array with the class file bytes
116 */
117 private byte[] loadClassBytes(String name) throws IOException
118 {
119     String cname = name.replace('.', '/') + ".caesar";
120     FileInputStream in = null;
121     in = new FileInputStream(cname);
122     try
123     {
124         ByteArrayOutputStream buffer = new ByteArrayOutputStream();
125         int ch;
126         while ((ch = in.read()) != -1)
127         {
128             byte b = (byte) (ch - key);
129             buffer.write(b);
130         }
131         in.close();
132         return buffer.toByteArray();
133     }
134     finally
135     {
136         in.close();
137     }
138 }
139
140 private int key;
141 }
```

### 程序清单9-2 Caesar.java

```
1 import java.io.*;
2
3 /**
4 * Encrypts a file using the Caesar cipher.
5 * @version 1.00 1997-09-10
6 * @author Cay Horstmann
7 */
8 public class Caesar
9 {
10    public static void main(String[] args)
11    {
12        if (args.length != 3)
13        {
14            System.out.println("USAGE: java Caesar in out key");
15            return;
16        }
17
18        try
19        {
20            FileInputStream in = new FileInputStream(args[0]);
```

```

21.     FileOutputStream out = new FileOutputStream(args[1]);
22.     int key = Integer.parseInt(args[2]);
23.     int ch;
24.     while ((ch = in.read()) != -1)
25.     {
26.         byte c = (byte) (ch + key);
27.         out.write(c);
28.     }
29.     in.close();
30.     out.close();
31. }
32. catch (IOException exception)
33. {
34.     exception.printStackTrace();
35. }
36. }
37. }
```

**API `java.lang.Class` 1.0**

- `ClassLoader getClassLoader()`

获取加载该类的类加载器。

**API `java.lang.ClassLoader` 1.0**

- `ClassLoader getParent() 1.2`

返回父类加载器，如果父类加载器是引导类加载器，则返回`null`。

- `static ClassLoader getSystemClassLoader() 1.2`

获取系统类加载器，即用于加载第一个应用类的类加载器。

- `protected Class findClass(String name) 1.2`

类加载器应该覆盖该方法，以查找类的字节码，并通过调用`defineClass`方法将字节码传给虚拟机。在类的名字中，使用`.`作为包名分隔符，并且不使用`.class`后缀。

- `Class defineClass(String name, byte[] byteCodeData, int offset, int length)`

将一个新的类添加到虚拟机中，其字节码在给定的数据范围内。

**API `java.net.URLClassLoader` 1.2**

- `URLClassLoader(URL[] urls)`

- `URLClassLoader(URL[] urls, ClassLoader parent)`

构建一个类加载器，它可以从给定的URL处加载类。如果URL以`/`结尾，那么它表示的一个目录，否则，它表示的是一个JAR文件。

**API `java.lang.Thread` 1.0**

- `ClassLoader getContextClassLoader() 1.2`

获取类加载器，该线程的创建者将其指定为执行该线程时最适合使用的类加载器。

- `void setContextClassLoader(ClassLoader loader) 1.2`

为该线程中的代码设置一个类加载器，以获取要加载的类。如果在启动一个线程时没有

显式地设置上下文类加载器，则使用父线程的上下文类加载器。

## 9.2 字节码校验

当类加载器将新加载的Java平台类的字节码传递给虚拟机时，这些字节码首先要接受校验器（verifier）的校验。校验器负责检查那些指令无法执行的明显有破坏性的操作。除了系统类外，所有的类都要被校验，不过，可以使用非正式的`-noverify`选项来钝化校验。

例如，可以使用下面的命令行：

```
java -noverify Hello
```

下面是校验器执行的一些检查：

- 变量要在使用之前进行初始化。
- 方法调用与对象引用类型之间要匹配。
- 访问私有数据和方法的规则没有被违反。
- 对本地变量的访问都在运行在堆栈内。
- 运行时堆栈没有溢出。

如果以上这些检查中任何一条没有通过，那么该类就被认为遭到了破坏，并且不予加载。



**注意：**如果熟悉Gödel's定理，那么你可能想知道，校验器究竟是如何证明某个类文件不存在类型不匹配、变量没有初始化和堆栈溢出等问题的。根据Gödel's定理，你无法设计相应的算法，使其能够处理程序文件，并决定输入程序是否有特定的属性（比如不出现堆栈溢出问题）。这是否属于Sun公司的公共关系部门和逻辑法则之间的矛盾呢？不——事实上，按照Gödel's的观点，校验器并非是一个决策算法。如果校验器接受了一个程序，那么该程序就确实是安全的。然而，也有许多程序尽管是安全的，但却被校验器拒绝了。（在强制用哑元值来初始化一个变量时，你就会碰到这个问题，因为编译器无法了解这个变量应该如何正确地初始化。）

这种严格的校验是出于安全上的考虑，有一些偶然性的错误，比如变量没有初始化，如果这类错误没有被捕获，就很容易地对系统造成严重的破坏。更为重要的是，在因特网这样开放的环境中，你必须保护自己以防恶意的程序员对你实施攻击，因为他们的目的就是要造成恶劣的影响。例如，通过修改运行时堆栈中的值，或者向系统对象的私有数据字段写入数据，某个程序就会突破浏览器的安全防线。

当然你可能想知道，为什么要有一个专门的校验器来检查这些特性呢。毕竟，编译器绝不会允许你生成一个这样的类文件：该类文件中有未初始化的变量或者可以通过另一个类来访问该类的某个私有数据字段。实际上，用Java语言编译器生成的类文件总是可以通过校验的。然而，类文件中使用的字节码格式是有很好的文档记录的，对于具有汇编程序设计经验并且拥有十六进制编辑器的人来说，要手工地创建一个对Java虚拟机来说，由合法的但是不安全的指令构成的类文件，是一件非常容易的事情。再次提醒你，要记住，校验器总是在防范被故意篡改的类文件，而不仅仅只是检查编译器产生的类文件。

下面的例子将展示如何创建一个变动过的类文件。我们从程序清单9-3的程序Verifier

Test.java开始。这是一个简单的程序，它调用一个方法，并且显示方法的运行结果。该程序既可以在控制台运行，也可以作为一个Applet程序来运行。其中的fun方法本身只是负责计算1+2。

```
static int fun()
{
    int m;
    int n;
    m = 1;
    n = 2;
    int r = m + n;
    return r;
}
```

作为一次实验，请尝试编译下面这个对该程序进行修改后的文件。

```
static int fun()
{
    int m = 1;
    int n;
    m = 1;
    m = 2;
    int r = m + n;
    return r;
}
```

在这种情况下，n没有被初始化，它可以是任何随机值。当然，编译器能够检测到这个问题并拒绝编译该程序。如果要建立一个不良的类文件，我们必须得多花点功夫。首先，运行javap程序，以便知晓编译器是如何翻译fun方法的。下面这个命令：

```
javap -c VerifierTest
```

用助记（mnemonic）格式显示了类文件中的字节码。

```
Method int fun()
  0  iconst_1
  1  istore_0
  2  iconst_2
  3  istore_1
  4  iload_0
  5  iload_1
  6  iadd
  7  istore_2
  8  iload_2
  9  ireturn
```

我们使用一个十六进制编辑器将指令3从istore\_1改为istore\_0，也就是说，局部变量0（即m）被初始化了两次，而局部变量1（即n）则根本没有初始化。我们必须知道这些指令的十六进制值，这些值在Tim Lindholm和Frank Yellin撰写的《The Java Virtual Machine Specification》一书中很容易就可以找到，该书由Addison-Wesley出版社于1999年出版。

```
0  iconst_1 04
1  istore_0 3B
2  iconst_2 05
3  istore_1 3C
4  iload_0 1A
5  iload_1 1B
6  iadd   60
```

```
7 istore_2 3D
8 iload_2 1C
9 ireturn AC
```

可以使用十六进制编辑器（例如DataWorkshop，可以从<http://www.dataworkshop.de>处下载）来进行修改操作。当然，你也可以使用hex1-mode的emacs进行修改。在图9-4中，你可以看到类文件VerifierTest.class被加载到了DataWorkshop编辑器中，fun方法的字节码已经被选定。

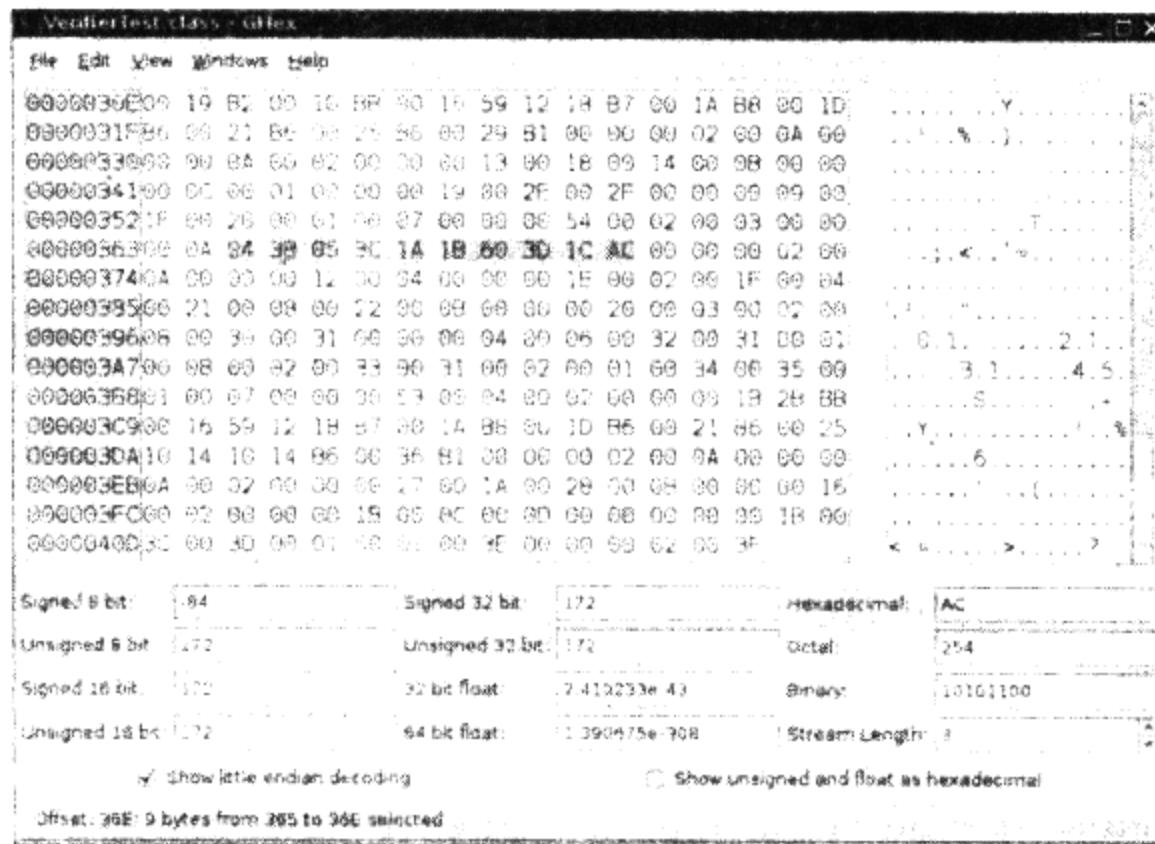


图9-4 使用十六进制编辑器修改字节码

将3C改为3B并保存类文件。然后设法运VerifierTest程序。将会看到下面的出错信息：

```
Exception in thread "main" java.lang.VerifyError: (class: VerifierTest, method:fun signature: ()I) Accessing value from uninitialized register 1
```

这很好——虚拟机发现了我们所做的修改。

现在用-noverify选项（或者-Xverify:none）来运行程序：

```
java -noverify VerifierTest
```

从表面上看，fun方法似乎返回了一个随机值。但实际上，该值是2与存储在尚未初始化的变量n中的值相加得到的结果。下面是典型的输出结果：

```
1 + 2 == 15102330
```

为了观察浏览器是如何处理校验的，我们编写的这段程序，既可以作为一个应用程序来运行，也可以作为一个applet来运行。把applet加载到浏览器中，并使用一个文件URL来访问，例如：

```
file:///C:/CoreJavaBook/v2ch9/VerifierTest/VerifierTest.html
```

这时，就会出现一个出错消息，这表明校验失败了（参见图9-5）。

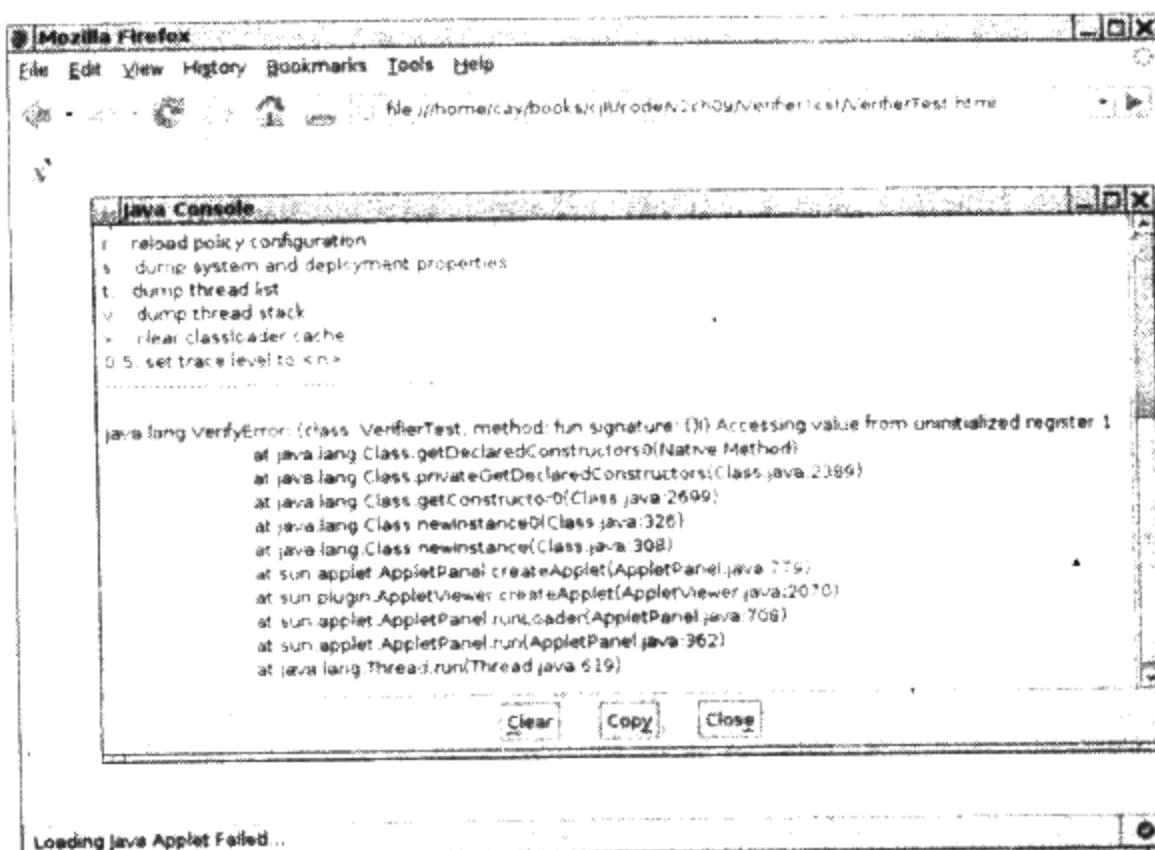


图9-5 加载受损类文件将会产生一个方法校验错误

## 程序清单9-3 VerifierTest.java

```

1. import java.applet.*;
2. import java.awt.*;
3.
4. /**
5. * This application demonstrates the bytecode verifier of the virtual machine. If you use a
6. * hex editor to modify the class file, then the virtual machine should detect the tampering.
7. * @version 1.00 1997-09-10
8. * @author Cay Horstmann
9. */
10. public class VerifierTest extends Applet
11. {
12.     public static void main(String[] args)
13.     {
14.         System.out.println("1 + 2 == " + fun());
15.     }
16.
17. /**
18. * A function that computes 1 + 2
19. * @return 3, if the code has not been corrupted
20. */
21. public static int fun()
22. {
23.     int m;
24.     int n;
25.     m = 1;
26.     n = 2;
27.     // use hex editor to change to "m = 2" in class file
28.     int r = m + n;
29.     return r;
30. }
```

```
31.  
32.     public void paint(Graphics g)  
33.     {  
34.         g.drawString("1 + 2 == " + fun(), 20, 20);  
35.     }  
36. }
```

### 9.3 安全管理器与访问权限

一旦某个类被加载到虚拟机中，并由检验器检查过之后，Java平台的第二种安全机制就会启动，这个机制就是安全管理器。安全管理器是一个负责控制某个操作是否允许执行的类。安全管理器负责检查的操作包括以下几个：

- 创建一个新的类加载器；
- 退出虚拟机；
- 使用反射访问另一个类的成员；
- 访问本地文件；
- 打开socket连接；
- 启动打印作业；
- 访问系统剪贴板；
- 访问AWT事件队列；
- 打开一个顶层窗口。

整个Java类库中还有许多其他类似的检查。

在运行Java应用程序时，默认的设置是不安装安全管理器的，这样所有的操作都是允许的。另一方面，applet浏览器会执行一个功能受限的安全策略。

例如，applet不允许退出虚拟机。如果它们试图调用exit方法，就会抛出一个安全异常。下面将详细说明这种情况。Runtime类的exit方法调用安全管理器的checkExit方法，下面是exit方法的全部代码：

```
public void exit(int status)  
{  
    SecurityManager security = System.getSecurityManager();  
    if (security != null)  
        security.checkExit(status);  
    exitInternal(status);  
}
```

这时安全管理器要检查退出请求是来自浏览器还是单个的applet程序。如果安全管理器同意了退出请求，那么checkExit便直接返回并继续处理下面正常的操作。但是，如果安全管理器不同意退出请求，那么checkExit方法就会抛出一个SecurityException异常。

只有当没有任何异常发生时，exit方法才能继续执行。然后它调用本地私有的exitInternal方法，实际终止虚拟机的运行。没有其他的方法可以终止虚拟机的运行，因为exitInternal方法是私有的，任何其他类都不能调用它。因此，任何试图退出虚拟机的代码都必须通过exit方法，从而在不触发安全异常的情况下，通过checkExit安全检查。

显然，安全策略的完整性依赖于谨慎的编码。标准类库中系统服务的提供者，在试图继续

任何敏感的操作之前，都必须与安全管理器进行协商。

Java平台的安全管理器，不仅允许系统管理员，而且允许程序员对各个安全访问权限实施细致的控制。我们将在下一节介绍这些特性。首先，我们将介绍Java 2 平台的安全模型的概况，然后介绍如何使用策略文件对各个权限实施控制。最后，我们要介绍如何来定义你自己的权限类型。

 **注意：**实现并安装自己的安全管理器是可行的，但是你不应该进行这种尝试，除非你是计算机安全方面的专家。配置标准的安全管理器更加安全。

### 9.3.1 Java平台安全性

JDK 1.0具有一个非常简单的安全模型，即本地类拥有所有的权限，而远程类只能在沙盒里运行。就像儿童只能在沙盒里玩沙子一样，远程代码只被允许打印屏幕和与用户进行交互。applet的安全管理器拒绝了远程代码对本地资源的所有访问。JDK 1.1 对此进行了微小的修改，如果远程代码带有可信赖的实体的签名，将被赋予和本地类相同的访问权限。不过，JDK 1.0 和1.1这两个版本提供的都是一种“要么都有，要么都没有”的权限赋予方法。程序要么拥有所有的访问权限，要么必须在沙盒里运行。

从Java SE 1.2 开始，Java平台拥有了更灵活的安全机制，它的安全策略建立了代码来源和访问权限集之间的映射关系（参见图9-6）。

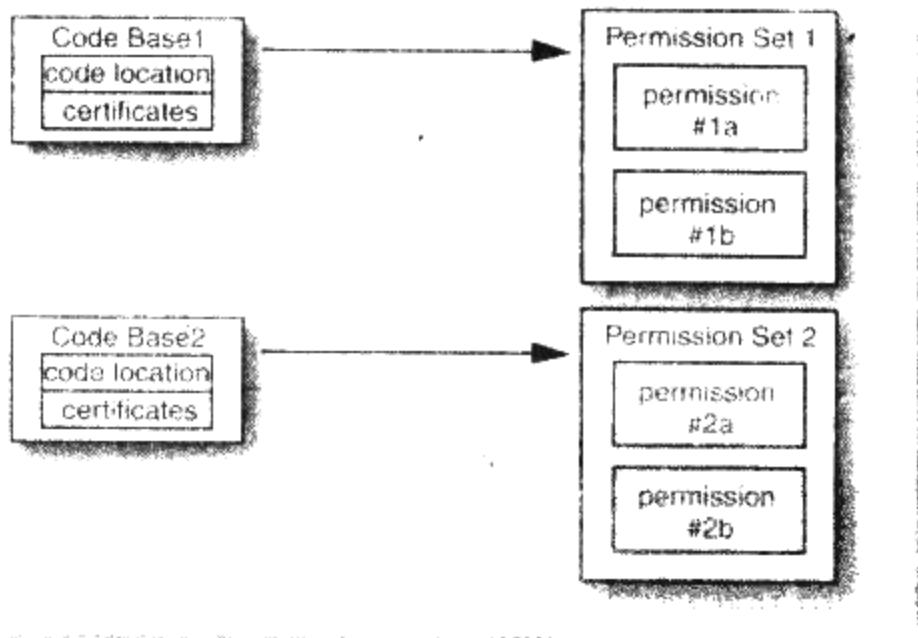


图9-6 一个安全策略

代码来源（code source）是由一个代码位置和一个证书指定的。代码位置指定了代码的来源。例如，远程applet代码的代码位置是下载applet的HTTP URL，JAR文件中代码位置是一个文件URL。证书的目的是要由某一方来保障代码没有被篡改过。我们将在本章的后面部分讨论证书。

权限（permission）是指由安全管理器负责检查的任何属性。Java平台支持许多访问权限类，每个类都封装了特定权限的详细信息。例如，下面这个FilePermission类的实例规定，它能够在/tmp目录下读取和写入任何文件。

```
FilePermission p = new FilePermission("/tmp/*", "read,write");
```

更为重要的是，Policy类的默认实现可从访问权限文件中读取权限。在权限文件中，同样的读权限表示为：

```
permission java.io.FilePermission "/tmp/*", "read,write";
```

我们将在下一节介绍权限文件。

图9-7显示了Java SE 1.2中提供的权限类的层次结构。JDK的后续版本添加了更多的权限类。

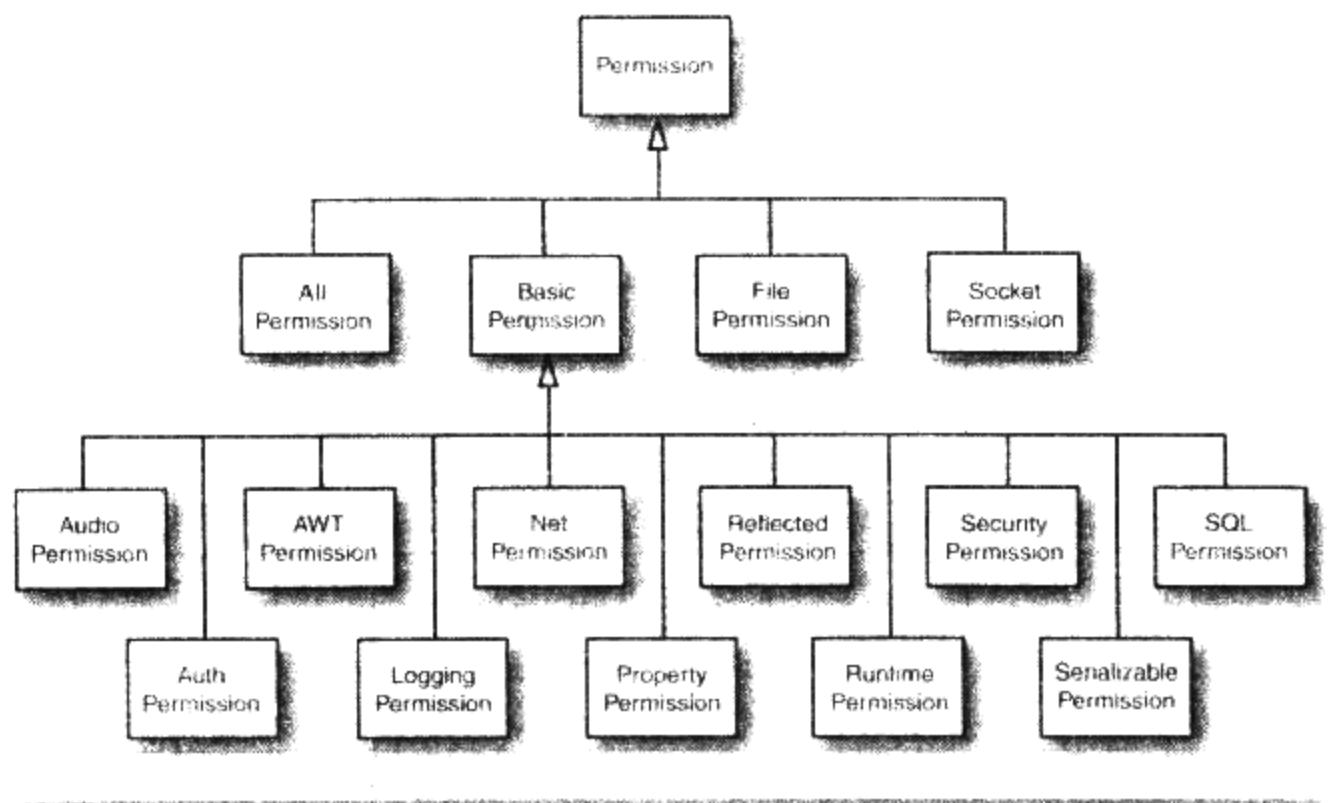


图9-7 权限类的层次结构

在上一节中，看到了SecurityManager类有许多诸如checkExit的安全检查方法，这些方法的存在，只是为了程序员的方便和向后的兼容性，它们都已被映射为标准的权限检查，例如，下面是checkExit方法的源代码：

```
public void checkExit()
{
    checkPermission(new RuntimePermission("exitVM"));
}
```

每个类都有一个保护域，它是一个用于封装类的代码来源和权限集合的对象。当SecurityManager类需要检查某个权限时，它要查看当前位于调用堆栈上的所有方法的类，然后它要获得所有类的保护域，并且询问每个保护域，该域的权限集合是否允许执行目前正在被检查的操作。如果所有的域都同意，那么检查得以通过。否则，就会抛出一个SecurityException异常。

为什么在调用堆栈上的所有方法都必须允许某个特定的操作呢？让我们通过一个实例来说明这个问题。假设一个applet的init方法想要打开一个文件，它可能会调用下面的语句：

```
Reader in = new FileReader(name);
```

`FileReader`构造器调用`InputStream`构造器，而`InputStream`构造器调用安全管理器的`checkRead`方法，安全管理器最后调用一个带有`FilePermission(name, "read")`对象的`checkPermission`。表9-1显示了该调用堆栈。

表9-1 权限检查期间的调用堆栈

| 类               | 方 法             | 代码来源               | 权 限                |
|-----------------|-----------------|--------------------|--------------------|
| SecurityManager | checkPermission | null               | AllPermission      |
| SecurityManager | checkRead       | null               | AllPermission      |
| InputStream     | constructor     | null               | AllPermission      |
| FileReader      | constructor     | null               | AllPermission      |
| applet          | init            | applet code source | applet permissions |

`InputStream`和`SecurityManager`类都属于系统类，它们的`CodeSource`为`null`，它们的权限都是由`AllPermission`类的一个实例组成的，`AllPermission`类允许执行所有的操作。显然地，仅仅根据它们的权限是无法确定检查结果的。正如我们所看到的那样，`checkPermission`方法必须考虑`applet`类的受限制的权限问题。通过检查整个调用堆栈，安全机制就能够确保一个类决不会要求另一个类代表自己去执行某个敏感的操作。

 **注意：**上面关于如何进行权限检查的简要介绍，向你展示了这方面的基本概念。不过我们在这里省略了对许多技术细节的说明。对于安全性的细节问题，我们建议你阅读Li Gong撰写的著作，以便了解更多的内容。有关Java平台安全模型的更多重要信息，请查阅Gary McGraw和Ed Felten撰写的《Securing Java: Getting Down to Business with Mobile Code, 第二版》一书，该书由Wiley出版社于1999年出版。你可以在下面的网站上找到该书的在线版本：<http://www.securingjava.com>。



### **java.lang.SecurityManager 1.0**

- `void checkPermission(Permission p)` 1.2

检查当前安全管理器是否授予给定的权限。如果没有授予该权限，本方法抛出一个`SecurityException`异常。



### **java.lang.Class 1.0**

- `ProtectionDomain getProtectionDomain()` 1.2

获取该类的保护域，如果该类被加载时没有保护域，则返回`null`。



### **java.security.ProtectionDomain 1.2**

- `ProtectionDomain(CodeSource source, PermissionCollection permissions)`

用给定的代码来源和权限构建一个保护域。

- `CodeSource getCodeSource()`

获取该保护域的代码来源。

- boolean implies(Permission p)  
如果该保护域允许给定的权限，则返回true。

### **java.security.CodeSource 1.2**

- Certificate[] getCertificates()  
获取与该代码来源相关的类文件签名的证书。
- URL getLocation()  
获取与该代码来源相关的类文件地址。

#### 9.3.2 安全策略文件

策略类要读取相应的策略文件，这些文件包含了将代码来源映射为权限的指令。下面是一个典型的策略文件：

```
grant codeBase "http://www.horstmann.com/classes"
{
    permission java.io.FilePermission "/tmp/*", "read,write";
}
```

该文件给所有下载自http://www.horstmann.com/classes的代码授予在/tmp目录下读取和写入文件的权限。

可以将策略文件安装在标准位置上。默认情况下，有两个位置可以安装策略文件：

- Java平台主目录的java.policy文件。
- 用户主目录的.java.policy文件（注意文件名前面的圆点）。

 注意：可以在java.security配置文件中修改这些文件的位置，默认位置设定为：

```
policy.url.1=file:${java.home}/lib/security/java.policy
policy.url.2=file:${user.home}/.java.policy
```

系统管理员可以修改java.security文件，并可以指定驻留在另外一台服务器上并且用户无法修改的策略URL。策略文件中允许存放任何数量的策略URL（这些URL带有连续的号码）。所有文件的权限都被组合了在一起。

如果你想将策略文件存储到文件系统之外，那么去可以实现Policy的一个收集权限的子类。

然后在java.security配置文件中更改下面这行：

```
policy.provider=sun.security.provider.PolicyFile
```

在测试期间，我们不喜欢经常地修改这些标准文件。因此，我们更愿意为每一个应用程序配置显式的策略文件。这样将权限写入一个独立的文件（比如MyApp.policy）中即可。要应用这个策略文件，可以有两个选择。一种是在应用程序的main方法内部设置系统属性：

```
System.setProperty("java.security.policy", "MyApp.policy");
```

或者，可以像下面这样启动虚拟机：

```
java -Djava.security.policy=MyApp.policy MyApp
```

对于applet，则可以用如下的启动命令。

```
appletviewer -J-Djava.security.policy=MyApplet.policy MyApplet.html
```

(可以用appletviewer的-J选项将任何命令行参数传给虚拟机)。

在这些例子中，MyApp.policy文件被添加到了其他有效的策略中。如果在命令行中添加了第二个等号，比如：

```
java -Djava.security.policy==MyApp.policy MyApp
```

那么应用程序就只使用指定的策略文件，而标准策略文件将被忽略。

 **警告：**在测试期间，一个容易犯的错误是在当前目录中留下了一个.java.policy文件，该文件授予了许许多多的权限，甚至可能授予了AllPermission。如果发现你的应用程序似乎没有应用策略文件中的规定，就应该检查当前目录下的.java.policy文件。如果使用的是UNIX系统，就更容易犯这样的错误，因为在UNIX中，文件名以圆点开头的文件默认是不显示的。

正如前面所说，在默认情况下，Java应用程序是不安装安全管理器的。因此，在安装安全管理器之前，看不到策略文件的作用。当然，可以将这行代码：

```
System.setSecurityManager(new SecurityManager());
```

添加到main方法中，或者在启动虚拟机的时候添加命令行选项-Djava.security.manager。

```
java -Djava.security.manager -Djava.security.policy=MyApp.policy MyApp
```

在本节的剩余部分，我们将要详细介绍如何描述策略文件的权限。我们将介绍整个策略文件的格式，不过不包括代码证书部分，代码证书将在本章的后面部分介绍。

一个策略文件包含一系列grant项。每一项都具有以下的形式：

```
grant codesource
{
    permission1;
    permission2;
    ...
};
```

代码来源包含一个代码基（如果某一项适用于所有来源的代码，则代码基可以省略）和值得信赖的用户特征（principal）与证书签名者的名字（如果不要求对该项签名，则可以省略）。

代码基可以设定为：

```
codeBase "url"
```

如果URL以“/”结束，那么它是一个目录。否则，它将被视为一个JAR文件的名字。例如：

```
grant codeBase "www.horstmann.com/classes/" { . . . };
grant codeBase "www.horstmann.com/classes/MyApp.jar" { . . . };
```

代码基是一个URL并且总是以斜杠作为文件分隔符，即使是Windows中的文件URL，也是如此。例如：

```
grant codeBase "file:C:/myapps/classes/" { . . . };
```

 **注意：**大家都知道http格式的URL都以双斜杠（http://）开头的，但是它很容易与file格式的URL搞混淆，策略文件阅读器接受两种格式的file URL，即file://localFile和file:localFile。此外，Windows驱动器名前面的斜杠是可有可无的。也就是说，下面的各种表示都是可以接受的：

```
file:C:/dir/filename.ext  
file:/C:/dir/filename.ext  
file:///C:/dir/filename.ext  
file:///C:/dir/filename.ext
```

实际上，我们的测试结果是file:///C:/dir/filename.ext也是允许的，对此我们不做解释。

权限采用下面的结构：

```
permission className targetName, actionList;
```

类名是权限类的全称类名（比如java.io.FilePermission）。目标名是个特定的权限值，例如，用于文件权限的目录名或者文件名，或者是用于socket权限的主机和端口。操作列表同样是与特定权限相关的，它是一个操作方式的列表，比如read或者connect等操作，用逗号分隔。有些权限类不需要目标名和操作列表。表9-2列出了标准的权限和它们执行的操作。

表9-2 权限及其相关的目标和操作

| 权 限                          | 目 标   | 操 作                              |
|------------------------------|---|----------------------------------|
| java.io.FilePermission       | 文件目标（见正文）   | read, write, execute, delete     |
| java.net.SocketPermission    | Socket 目标（见正文）  | accept, connect, listen, resolve |
| java.util.PropertyPermission | 属性目标（见正文）   | read, write                      |
| java.lang.RuntimePermission  | createClassLoader<br>getClassLoader<br>setContextClassLoader<br>enableContextClassLoaderOverride<br>createSecurityManager<br>setSecurityManager<br>exitVM<br>getenv.variableName<br>shutdownHooks<br>setFactory<br>setIO<br>modifyThread<br>stopThread<br>modifyThreadGroup<br>getProtectionDomain<br>readFileDescriptor<br>writeFileDescriptor<br>loadLibrary.libraryName<br>accessClassInPackage.packageName<br>defineClassInPackage.packageName<br>accessDeclaredMembers.className<br>queuePrintJob<br>getStackTrace | (无)                              |

(续)

| 权 限                                 | 目 标  | 操 作 |
|-------------------------------------|--|-----|
| java.lang.RuntimePermission         | setDefaultUncaughtExceptionHandler<br>preferences<br>usePolicy   | (无) |
| java.awt.AWTPermission              | showWindowWithoutWarningBanner<br>accessClipboard<br>accessEventQueue<br>createRobot<br>fullScreenExclusive<br>listenToAllAWTEvents<br>readDisplayPixels<br>replaceKeyboardFocusManager<br>watchMousePointer<br>setWindowAlwaysOnTop<br>setAppletStub  | (无) |
| java.net.NetPermission              | setDefaultAuthenticator<br>specifyStreamHandler<br>requestPasswordAuthentication<br>setProxySelector<br>getProxySelector<br>setCookieHandler<br>getCookieHandler<br>setResponseCache<br>getResponseCache   | (无) |
| java.lang.reflect.ReflectPermission | suppressAccessChecks   | (无) |
| java.io.SerializablePermission      | enableSubclassImplementation<br>enableSubstitution   | (无) |
| java.security.SecurityPermission    | createAccessControlContext<br>getDomainCombiner<br>getPolicy<br>setPolicy<br>getProperty.keyName<br>setProperty.keyName<br>insertProvider.providerName<br>removeProvider.providerName<br>setSystemScope<br>setIdentityPublicKey<br>setIdentityInfo<br>addIdentityCertificate<br>removeIdentityCertificate<br>printIdentity<br>clearProviderProperties.providerName | (无) |

(续)

| 权 限                                 | 目 标   | 操 作 |
|-------------------------------------|---|-----|
| java.security.SecurityPermission    | putProviderProperty.providerName<br>removeProviderProperty.providerName<br>getSignerPrivateKey<br>setSignerKeyPair  | (无) |
| java.security.AllPermission         |   | (无) |
| javax.audio.AudioPermission         | play<br>record  | (无) |
| javax.security.auth.AuthPermission  | doAs<br>doAsPrivileged<br>getSubject<br>getSubjectFromDomainCombiner<br>setReadOnly<br>modifyPrincipals<br>modifyPublicCredentials<br>modifyPrivateCredentials<br>refreshCredential<br>destroyCredential<br>createLoginContext.contextName<br>getLoginConfiguration<br>setLoginConfiguration<br>refreshLoginConfiguration | (无) |
| java.util.logging.LoggingPermission | control   | (无) |
| java.sql.SQLPermission              | setLog  | (无) |

正如表9-2中所示，大部分权限只允许执行某个特定的操作。可以将操作视为带有一个隐含操作“permit”的目标。这些权限类都继承自BasicPermission类（参见本章图9-7）。然而，文件、socket和属性权限的目标都比较复杂，我们必须对它们进行详细介绍。

文件权限的目标可以有下面几种形式：

|               |                  |
|---------------|------------------|
| file          | 文件               |
| directory/    | 目录               |
| directory/*   | 目录中的所有文件         |
| *             | 当前目录中的所有文件       |
| directory/-   | 目录或者它的子目录中的所有文件  |
| -             | 当然目录或者它的子目录中所有文件 |
| <<ALL FILES>> | 文件系统中的所有文件       |

例如，下面的权限项赋予对/myapp目录和它的子目录中的所有文件的访问权限。

```
permission java.io.FilePermission "/myapp/-", "read,write,delete";
```

必须使用\\转义字符序列来表示Window文件名中的反斜杠。

```
permission java.io.FilePermission "c:\\myapp\\-", "read,write,delete";
```

Socket权限的目标由主机和端口范围组成。对主机的描述具有下面几种形式：

|                    |                 |
|--------------------|-----------------|
| hostname或IPAddress | 单个主机            |
| localhost 或空字符串    | 本地主机            |
| *.domainSuffix     | 以给定后缀结尾的域中所有的主机 |
| *                  | 所有主机            |

端口范围是可选的，具有下面几种形式：

|        |              |
|--------|--------------|
| :n     | 单个端口         |
| :n-    | 编号大于等于n的所有端口 |
| :-n    | 编号小于等于n的所有端口 |
| :n1-n2 | 位于给定范围内的所有端口 |

下面是一个权限的实例：

```
permission java.net.SocketPermission "*.horstmann.com:8000-8999", "connect";
```

最后，属性权限的目标可以采用下面两种形式之一：

|                  |             |
|------------------|-------------|
| property         | 一个特定的属性     |
| propertyPrefix.* | 带有给定前缀的所有属性 |

“java.home” 和 “java.vm.\*” 就是这样的例子。

例如，下面的权限项允许程序读取以java.vm开头的所有属性。

```
permission java.util.PropertyPermission "java.vm.*", "read";
```

可以在策略文件中使用系统属性。\${property} 标记表示一个属性值。例如，\${user.home} 表示用户的主目录。下面是在访问权限项中使用系统属性的典型应用。

```
permission java.io.FilePermission "${user.home}", "read,write";
```

为了创建平台无关的策略文件，使用file.separator属性而不是使用显式的/或者\\分隔符绝对是个好主意。如果要使它更加简单，可以使用符号\${/}作为\${file.separator}的缩写。例如，

```
permission java.io.FilePermission "${user.home}${/}-", "read,write";
```

是一个可移植的项，用于授予对在用户的主目录及其子目录中的文件进行读写的权限。



**注意:** JDK提供了一个名为policytool的基础工具，可以用它编辑策略文件（参见图9-8）。

当然，该工具对完全不清楚其大部分设置的用户来说是不适用的。这正好证明了这样一种观念，管理工具可能只能供那些关心“定位-点击操作”，而不关心具体语法的系统管理员使用。尽管如此，它所欠缺的是对于非专家的用户来说非常具有实际意义的级别设置（例如低，中或者高安全性设置）。一般来说，我们相信Java 2平台肯定包含了所有级别的细粒度安全模型，但是将这些模型提供给最终用户和系统管理员会获得更大的收益。

### 9.3.3 定制权限

在本节中，我们将要介绍如何把自己的权限类提供给用户，以使得他们可以在策略文件中引用这些权限类。

如果要实现自己的权限类，可以继承Permission类，并提供以下方法：

- 带有两个String参数的构造器，这两个参数分别是目标和操作列表

- String getActions()
- boolean equals()
- int hashCode()
- boolean implies(Permission other)

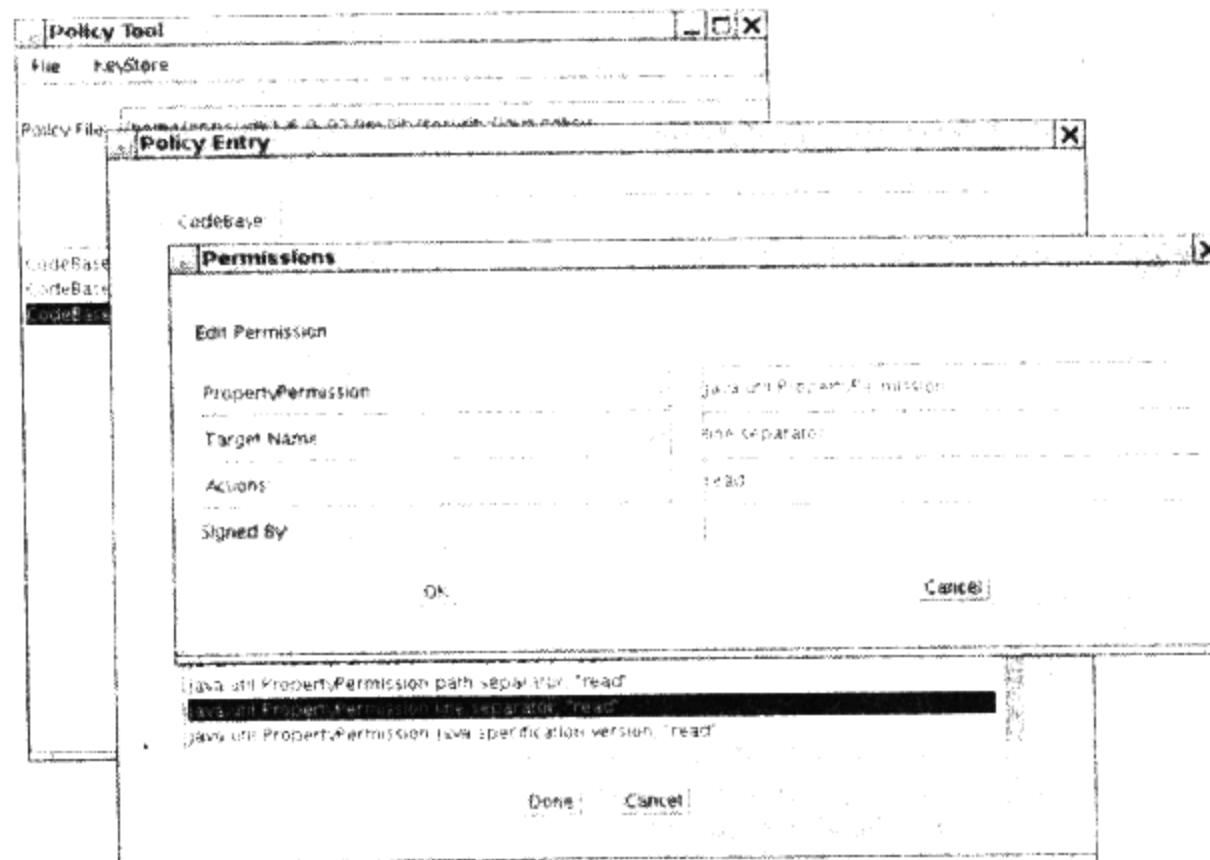


图9-8 策略工具

最后一个方法是最重要的。权限有一个次序，其中更加通用的权限隐含了更加具体的权限。请考虑下面的文件权限：

```
p1 = new FilePermission("/tmp/-", "read, write");
```

该权限允许读写/tmp目录以及子目录中的任何文件。

该权限隐含了其他更加具体的权限：

```
p2 = new FilePermission("/tmp/-", "read");
p3 = new FilePermission("/tmp/aFile", "read, write");
p4 = new FilePermission("/tmp/aDirectory/-", "write");
```

换句话说，如果

- p1的目标文件集包含p2的目标文件集。

- p1的操作集包含p2的操作集。

那么，文件访问权限p1就隐含了另一个文件访问权限p2。

请考虑下面关于implies方法的用法举例。当`FileInputStream`构造器想要打开一个文件，以读取该文件时，要检查它是否拥有操作权限。如果要执行这种检查，就应将一个特定的文件权限对象传递给`checkPermission`方法：

```
checkPermission(new FilePermission(fileName, "read"));
```

现在安全管理器询问所有适用的权限是否隐含了该权限。如果其中某个隐含了该权限，就

通过了检查。

特别地，`AllPermission`隐含了其他所有的权限。

如果你定义了自己的权限类，那么必须对权限对象定义一个合适的隐含法则。例如，假设你为采用Java技术的机顶盒定义一个`TVPermission`，那么下面这个访问权限

```
new TVPermission("Tommy:2-12:1900-2200", "watch,record")
```

将允许Tommy在19点到22点之间对2至12频道的电视节目进行观看和录像。必须实现`implies`方法，以隐含像下面这样的特定权限。

```
new TVPermission("Tommy:4:2000-2100", "watch")
```

#### 9.3.4 实现权限类

在下面这个示例程序中，我们实现了一个新的权限，用于监视将文本插入到文本域的操作。该程序会确保你不能输入“不良单词”，例如`sex`、`drugs`以及`C++`等。我们使用一个定制的权限类，以便在策略文件中提供这些不良单词。

下面这个`JTextArea`的子类询问安全管理器是否准备好了去添加新文本。

```
class WordCheckTextArea extends JTextArea
{
    public void append(String text)
    {
        WordCheckPermission p = new WordCheckPermission(text, "insert");
        SecurityManager manager = System.getSecurityManager();
        if (manager != null) manager.checkPermission(p);
        super.append(text);
    }
}
```

如果安全管理器赋予了`WordCheckPermission`权限，那么该文本就可以追加。否则，`checkPermission`方法就会抛出一个异常。

单词检查权限有两个可能的操作，一个是`insert`（用于插入特定文本的权限），另一个是`avoid`（添加不包含某些不良单词的任何文本的权限）。应该用下面的策略文件运行这个程序：

```
grant
{
    permission WordCheckPermission "sex,drugs,C++", "avoid";
};
```

这个策略文件赋予的权限，可以插入除不良单词`sex`、`drugs`和`C++`之外的任何文本。

当设计`WordCheckPermission`类时，我们必须特别注意`implies`方法，下面是控制权限`p1`是否隐含`p2`的规则：

- 如果`p1`有`avoid`操作，`p2`有`insert`操作，那么`p2`的目标必须避开`p1`中的所有单词。例如，下面这个权限：

```
WordCheckPermission "sex,drugs,C++", "avoid"
```

隐含了下面这个权限：

```
WordCheckPermission "Mary had a little lamb", "insert"
```

- 如果`p1`和`p2`都有`avoid`操作，那么`p2`的单词集合必须包含`p1`单词集合中的所有单词。例如，

下面这个权限：

```
WordCheckPermission "sex,drugs", "avoid"
```

隐含了下面这个权限：

```
WordCheckPermission "sex,drugs,C++", "avoid"
```

- 如果p1和p2都有insert操作，那么p1的文本必须包含p2的文本。例如，下面这个权限：

```
WordCheckPermission "Mary had a little lamb", "insert"
```

包含了下面这个权限：

```
WordCheckPermission "a little lamb", "insert"
```

可以在程序清单9-4中看到该类的具体实现。

请注意，可以用Permission类中名字容易混淆的getName方法来获取权限的目标。

由于在策略文件中权限是由一对字符串来表示的，因此，权限类需要准备好解析这些字符串。特别地，我们应该使用下面的方法，将用逗号分隔的avoid权限的不良单词表转换为一个真正的Set。

```
public Set<String> badWordSet()
{
    Set<String> set = new HashSet<String>();
    set.addAll(Arrays.asList(getName().split(",")));
    return set;
}
```

该代码允许我们用equals和containsAll方法来比较这些集。正如我们在第2章中所介绍的那样，如果两个集包含任意次序的相同元素，那么集类的equals方法可以判定它们相等。例如，由“sex,drugs,C++”和“C++,drugs,sex”产生的两个集是相等的集。

**X 警告：**务必把你的权限类设为public。策略文件加载器不能加载位于引导类路径之外的可视包的类，并且它会悄悄忽略它无法找到的所有类。

程序清单9-5中的程序展示了WordCheckPermission类是如何工作的。请在文本框内输入任意文本，然后按下Insert按钮。如果文本通过了安全检查，该文本就会被添加到文本区域中。如果没有通过检查，就会弹出一个消息（参见图9-9）。

**X 警告：**如果仔细看看图9-9，就会看到窗体中有一个警告框，该框体带有误导性的标题“Java Applet Window”。该窗口的标题是由java.awt.AWTPermission的showWindowWithoutWarningBanner目标决定的。如果你愿意，可以编辑策略文件来赋予该权限。

#### 程序清单9-4 WordCheckPermission.java

```
1 import java.security.*;
```

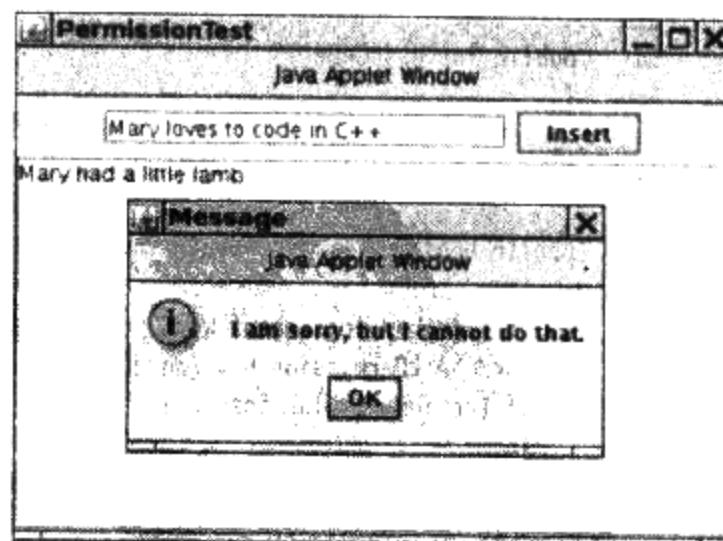


图9-9 PermissionTest程序

```
2. import java.util.*;  
3.  
4. /**  
5. * A permission that checks for bad words.  
6. * @version 1.00 1999-10-23  
7. * @author Cay Horstmann  
8. */  
9. public class WordCheckPermission extends Permission  
10.{  
11.    /**  
12.     * Constructs a word check permission  
13.     * @param target a comma separated word list  
14.     * @param anAction "insert" or "avoid"  
15.    */  
16.    public WordCheckPermission(String target, String anAction)  
17.    {  
18.        super(target);  
19.        action = anAction;  
20.    }  
21.  
22.    public String getActions()  
23.    {  
24.        return action;  
25.    }  
26.  
27.    public boolean equals(Object other)  
28.    {  
29.        if (other == null) return false;  
30.        if (!getClass().equals(other.getClass())) return false;  
31.        WordCheckPermission b = (WordCheckPermission) other;  
32.        if (!action.equals(b.action)) return false;  
33.        if (action.equals("insert")) return getName().equals(b.getName());  
34.        else if (action.equals("avoid")) return badWordSet().equals(b.badWordSet());  
35.        else return false;  
36.    }  
37.  
38.    public int hashCode()  
39.    {  
40.        return getName().hashCode() + action.hashCode();  
41.    }  
42.  
43.    public boolean implies(Permission other)  
44.    {  
45.        if (!(other instanceof WordCheckPermission)) return false;  
46.        WordCheckPermission b = (WordCheckPermission) other;  
47.        if (action.equals("insert"))  
48.        {  
49.            return b.action.equals("insert") && getName().indexOf(b.getName()) >= 0;  
50.        }  
51.        else if (action.equals("avoid"))  
52.        {  
53.            if (b.action.equals("avoid")) return b.badWordSet().containsAll(badWordSet());  
54.            else if (b.action.equals("insert"))  
55.            {  
56.                for (String badWord : badWordSet())  
57.                    if (b.getName().indexOf(badWord) >= 0) return false;  
58.            }  
59.        }  
60.    }
```

```

59.     }
60.     else return false;
61.   }
62.   else return false;
63. }
64.
65. /**
66. * Gets the bad words that this permission rule describes.
67. * @return a set of the bad words
68. */
69. public Set<String> badWordSet()
70. {
71.   Set<String> set = new HashSet<String>();
72.   set.addAll(Arrays.asList(getName().split(",")));
73.   return set;
74. }
75.
76. private String action;
77 }

```

### 程序清单9-5 PermissionTest.java

```

1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.swing.*;
4.
5. /**
6. * This class demonstrates the custom WordCheckPermission.
7. * @version 1.03 2007-10-06
8. * @author Cay Horstmann
9. */
10. public class PermissionTest
11 {
12.   public static void main(String[] args)
13.   {
14.     System.setProperty("java.security.policy", "PermissionTest.policy");
15.     System.setSecurityManager(new SecurityManager());
16.     EventQueue.invokeLater(new Runnable()
17.     {
18.       public void run()
19.       {
20.         JFrame frame = new PermissionTestFrame();
21.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22.         frame.setVisible(true);
23.       }
24.     });
25.   }
26. }
27.
28. /**
29. * This frame contains a text field for inserting words into a text area that is protected
30. * from "bad words".
31. */
32. class PermissionTestFrame extends JFrame
33. {
34.   public PermissionTestFrame()

```

```
35. {
36.     setTitle("PermissionTest");
37.     setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
38.
39.     textField = new JTextField(20);
40.     JPanel panel = new JPanel();
41.     panel.add(textField);
42.     JButton openButton = new JButton("Insert");
43.     panel.add(openButton);
44.     openButton.addActionListener(new ActionListener()
45.     {
46.         public void actionPerformed(ActionEvent event)
47.         {
48.             insertWords(textField.getText());
49.         }
50.     });
51.
52.     add(panel, BorderLayout.NORTH);
53.
54.     textArea = new WordCheckTextArea();
55.     add(new JScrollPane(textArea), BorderLayout.CENTER);
56. }
57.
58. /**
59. * Tries to insert words into the text area. Displays a dialog if the attempt fails.
60. * @param words the words to insert
61. */
62. public void insertWords(String words)
63. {
64.     try
65.     {
66.         textArea.append(words + "\n");
67.     }
68.     catch (SecurityException e)
69.     {
70.         JOptionPane.showMessageDialog(this, "I am sorry, but I cannot do that.");
71.     }
72. }
73.
74. private JTextField textField;
75. private WordCheckTextArea textArea;
76. private static final int DEFAULT_WIDTH = 400;
77. private static final int DEFAULT_HEIGHT = 300;
78. }
79.
80. /**
81. * A text area whose append method makes a security check to see that no bad words are added.
82. */
83. class WordCheckTextArea extends JTextArea
84. {
85.     public void append(String text)
86.     {
87.         WordCheckPermission p = new WordCheckPermission(text, "insert");
88.         SecurityManager manager = System.getSecurityManager();
89.         if (manager != null) manager.checkPermission(p);
90.         super.append(text);
```

```
91. }
92. }
```

### API `java.security.Permission` 1.2

- `Permission(String name)`  
用指定的目标名构建一个权限。
- `String getName()`  
返回该权限的对象名称。
- `boolean implies(Permission other)`

检查该权限是否隐含了其他权限。如果其他权限描述了一个更加具体的条件，而这个具体条件是由该权限所描述的条件所产生的结果，就需要进行这样的检查。

## 9.4 用户认证

Java认证和授权服务（JAAS，Java Authentication and Authorization Service）是Java SE 1.4及以上版本的一部分。“认证”部分主要负责确定程序使用者的身份，而“授权”将各个用户映射到相应的权限。

JAAS是一个可插拔的API，可以将Java应用程序与实现认证的特定技术分离开来。除此之外，JAAS还支持UNIX登录、NT登录、Kerberos认证和基于证书的认证。

一旦用户通过认证，就可以为其附加一组权限。例如，这里我们赋予Harry一个特定的权限，而其他用户则没有。它的语法规则如下：

```
grant principal com.sun.security.auth.UnixPrincipal "harry"
{
    permission java.util.PropertyPermission "user.*", "read";
    ...
};
```

在该语法中，`com.sun.security.auth.UnixPrincipal`类检查运行该程序的UNIX用户名，它的`getName`方法将返回UNIX登录名，然后我们就可以检查该名称是否等于“harry”。

可以使用一个`LoginContext`以使得安全管理器能够检查这样的授权语句。下面是登录代码的基本轮廓：

```
try
{
    System.setSecurityManager(new SecurityManager());
    LoginContext context = new LoginContext("Login1"); // defined in JAAS configuration file
    context.login();
    // get the authenticated Subject
    Subject subject = context.getSubject();

    ...
    context.logout();
}
catch (LoginException exception) // thrown if login was not successful
{
    exception.printStackTrace();
}
```

这里，`subject`是指已经被认证的个体。

LoginContext构造器中的字符串参数“Login1”是指JAAS配置文件中具有相同名字的项。下面是一个简单的配置文件：

```

Login1
{
    com.sun.security.auth.module.UnixLoginModule required;
    com.whizzbang.auth.module.RetinaScanModule sufficient;
};

Login2
{
    ...
};

```

当然，JDK中没有包含任何使用biometric的登录模块。JDK在com.sun.security.auth.module包中包含以下模块：

```

UnixLoginModule
NTLoginModule
Krb5LoginModule
JndiLoginModule
KeyStoreLoginModule

```

一个登录策略由一个登录模块序列组成，每个模块被标记为required、sufficient、requisite或optional。这些关键字的含义在下面的算法中进行了描述：

- 1) 各个模块依次执行，直到有一个sufficient的模块认证成功，或者有一个requisite的模块认证失败，或者已经执行到最后一个模块时才停止。
- 2) 当标记为required和requisite的所有模块都认证成功，或者它们都没有被执行，但至少有一个sufficient或optional的模块认证成功时，这次认证就成功了。

登录时要对登录的主体(subject)进行认证，该主体可以拥有多个特征(principal)。特征描述了主体的某些属性，比如用户名、组ID或角色等。我们在grant语句中可以看到，特征控制着各个权限。com.sun.security.auth.UnixPrincipal类描述了UNIX登录名，UnixNumericGroupPrincipal类可以用来检测UNIX用户组中的成员。

使用下面的语法，grant语句可以对一个特征进行测试：

```
grant principalClass "principalName"
```

例如：

```
grant com.sun.security.auth.UnixPrincipal "harry"
```

当用户登录后，然后就在独立的访问控制上下文中，运行要求检查用户特征的代码。使用静态的doAs或doAsPrivileged方法，启动一个新的PrivilegedAction，其run方法将执行这个代码。

这两个方法都可以通过使用主体特征的权限来调用某个对象的run方法去执行某个操作，该对象是实现了PrivilegedAction接口的对象。

```

PrivilegedAction<T> action = new
    PrivilegedAction<T>()
{
    public T run()
    {
        // run with permissions of subject principals
    }
}

```

```
    }
};

T result = Subject.doAs(subject, action); // or Subject.doAsPrivileged(subject, action, null)
```

如果该操作会抛出检查异常，那么必须改为实现`PrivilegedExceptionAction`接口。

`doAs`和`doAsPrivileged`方法之间的区别是微小的。`doAs`方法开始于当前的访问控制上下文，而`doAsPrivileged`方法则开始于一个新的上下文。后者允许将登录代码和“业务逻辑”的权限相分离。在我们的示例应用程序中，登录代码有如下权限：

```
permission javax.security.auth.AuthPermission "createLoginContext.Login1";
permission javax.security.auth.AuthPermission "doAsPrivileged";
```

通过认证的用户有一个权限：

```
permission java.util.PropertyPermission "user.*", "read";
```

如果我们用`doAs`代替了`doAsPrivileged`，那么登录代码也需要这个权限！

程序清单9-6和程序清单9-7的程序展示了如何限制某些用户的权限。`AuthTest`程序对用户的身份进行认证，然后运行一个简单的操作，获得一个系统属性。

要使该例子能够运行，必须将登录类和操作类的代码封装到两个独立的JAR文件中：

```
javac *.java
jar cvf login.jar AuthTest.class
jar cvf action.jar SysPropAction.class
```

如果查看程序清单9-8中的策略文件，将会看到名为harry的UNIX用户拥有读取所有文件的权限。将harry改为你自己的登录名，然后运行下面的命令

```
java -classpath login.jar:action.jar
-Djava.security.policy=AuthTest.policy
-Djava.security.auth.login.config=jaas.config
AuthTest
```

程序清单9-12展示了的登录配置。

在Windows下运行时，请在`AuthTest.policy`和`jaas.config`两个文件中都将Unix改成NT，并且用分号来分隔各个JAR文件：

```
java -classpath login.jar;action.jar ...
```

`AuthTest`程序现在将显示`user.home`属性的值。但是，如果更改了`AuthTest.policy`文件中的登录名，那么就应该抛出一个安全异常，因为你不再拥有必需的权限了。



**警告：**必须严格按照这些指令来运行。如果对程序进行了一些看上去无关紧要的更改，那就很容易使你的设置出错。

### 程序清单9-6 AuthTest.java

```
1. import java.security.*;
2. import javax.security.auth.*;
3. import javax.security.auth.login.*;
4.
5. /**
6. * This program authenticates a user via a custom login and then executes the SysPropAction
7. * with the user's privileges.
```

```

8. * @version 1.01 2007-10-06
9. * @author Cay Horstmann
10. */
11. public class AuthTest
12. {
13.     public static void main(final String[] args)
14.     {
15.         System.setSecurityManager(new SecurityManager());
16.         try
17.         {
18.             LoginContext context = new LoginContext("Login1");
19.             context.login();
20.             System.out.println("Authentication successful.");
21.             Subject subject = context.getSubject();
22.             System.out.println("subject=" + subject);
23.             PrivilegedAction<String> action = new SysPropAction("user.home");
24.             String result = Subject.doAsPrivileged(subject, action, null);
25.             System.out.println(result);
26.             context.logout();
27.         }
28.         catch (LoginException e)
29.         {
30.             e.printStackTrace();
31.         }
32.     }
33. }

```

### 程序清单9-7 SysPropAction.java

```

1. import java.security.*;
2.
3. /**
4.  * This action looks up a system property.
5. * @version 1.01 2007-10-06
6. * @author Cay Horstmann
7. */
8. public class SysPropAction implements PrivilegedAction<String>
9. {
10.    /**
11.     Constructs an action for looking up a given property.
12.     @param propertyName the property name (such as "user.home")
13.    */
14.    public SysPropAction(String propertyName) { this.propertyName = propertyName; }
15.
16.    public String run()
17.    {
18.        return System.getProperty(propertyName);
19.    }
20.
21.    private String propertyName;
22. }

```

### 程序清单9-8 AuthTest.policy

```

1. grant codebase "file:login.jar"
2.

```

```
3 permission javax.security.auth.AuthPermission "createLoginContext.Login1";
4 permission javax.security.auth.AuthPermission "doAsPrivileged";
5 };
6
7 grant principal com.sun.security.auth.UnixPrincipal "harry"
8 {
9     permission java.util.PropertyPermission "user.*", "read";
10 };
```

### API **javax.security.auth.login.LoginContext 1.4**

- **LoginContext(String name)**

创建一个登录上下文。name对应于JAAS配置文件中的登录描述符。

- **void login()**

建立一个登录操作，如果登录失败，则抛出一个LoginException异常。请调用JAAS配置文件中的管理器上的login方法。

- **void logout()**

Subject退出登录。请调用JAAS配置文件中的管理器上的logout方法。

- **Subject getSubject()**

返回认证过的Subject。

### API **javax.security.auth.Subject 1.4**

- **Set<Principal> getPrincipals()**

获取该Subject的各个Principal。

- **static Object doAs(Subject subject, PrivilegedAction action)**

- **static Object doAs(Subject subject, PrivilegedExceptionAction action)**

- **static Object doAsPrivileged(Subject subject, PrivilegedAction action, AccessControlContext context)**

- **static Object doAsPrivileged(Subject subject, PrivilegedExceptionAction action, AccessControlContext context)**

以subject的身份执行的特许操作。它将返回run方法的返回值。doAsPrivileged方法在给定的访问控制上下文中执行该操作。你可以提供一个在前面调用静态方法AccessController.getContext()时所获得的“上下文快照”，或者指定为null，以便使其在一个新的上下文中执行该代码。

### API **java.security.PrivilegedAction 1.4**

- **Object run()**

必须定义该方法，以执行你想要代表某个主体去执行的代码。

### API **java.security.PrivilegedExceptionAction 1.4**

- **Object run()**

必须定义该方法，以执行你想要代表某个主体去执行的代码。本方法可以抛出任何受检

查的异常。

### API `java.security.Principal` 1.1

- `String getName()`

返回该特征的身份标识。

## JAAS登录模块

在本节中，我们将要用一个JAAS例子向读者介绍：

- 如何实现你自己的登录模块；
- 如何实现基于角色的认证。

如果登录信息存储在数据库中，那么使用自己的登录模块就非常有用。尽管你可能很喜欢默认的登录模块，但是学习如何定制自己的模块将有助于你理解JAAS配置文件的各个选项。

基于角色的认证对于大量用户的管理来说是十分必要的。将所有合法用户的名字都写入策略文件是不切实际的。而登录模块应该将用户映射到诸如“admin”或“HR”等角色，并且权限的赋予也要基于这些角色。

登录模块的工作之一是组装被认证的主体的特征集。如果一个登录模块支持某些角色，该模块就会增加`Principal`对象来描述这些角色。JDK并没有提供相应的类，所以我们写了自己的类（见程序清单9-9）。该类直接存储了一个描述/值对，例如`role=admin`。该类的`getName`方法用于返回该描述/值对，因此我们就可以添加基于角色的权限到策略文件中：

```
grant principal SimplePrincipal "role=admin" { . . . }
```

我们的登录模块会在包含如下行的文本文件中查找用户、密码和角色：

```
harry|secret|admin  
carl|guessme|HR
```

当然，在实际的登录模块中，你可能会将这些信息存储在数据库或者目录中。

在程序清单9-10中可以找到`SimpleLoginModule`的代码。其`checkLogin`方法用于检查输入的用户名和密码是否与密码文件中的用户记录相匹配。如果匹配成功，则会添加两个`SimplePrincipal`对象到主体的特征集中。

```
Set<Principal> principals = subject.getPrincipals();  
principals.add(new SimplePrincipal("username", username));  
principals.add(new SimplePrincipal("role", role));
```

`SimpleLoginModule`剩余的部分就非常直截了当了。`initialize`方法接收下面几个参数：

- 用于认证的`Subject`；
- 一个获取登录信息的`handler`；
- 一个`sharedState`映射表，它可以用于登录模块之间的通信；
- 一个`options`映射表，它包含了登录配置文件中设置的名/值对。

例如，我们将模块做如下配置：

```
SimpleLoginModule required pwfile="password.txt";
```

则登录模块从`options`映射表中获取`pwfile`设置。

该登录模块并没有收集用户名和密码，这是单独的`handler`需要做的工作。这种功能上的分

离有助于你在各种情况下使用相同的登录模块，而不用关心登录信息是来自GUI对话框、控制台提示符还是配置文件。

当创建LoginContext时，handler已经被指定。例如，

```
LoginContext context = new LoginContext("Login1",
    new com.sun.security.auth.callback.DialogCallbackHandler());
```

DialogCallbackHandler弹出一个简单的GUI对话框，以获取用户名和密码。而com.sun.security.auth.callback.TextCallbackHandler则从控制台获取这些信息。

但是，在我们的应用程序中，是通过自己编写的GUI来获得用户名和密码的（参见图9-10）。我们创建了一个简单的handler，仅仅用于存储和返回那些信息（见程序清单9-11）。

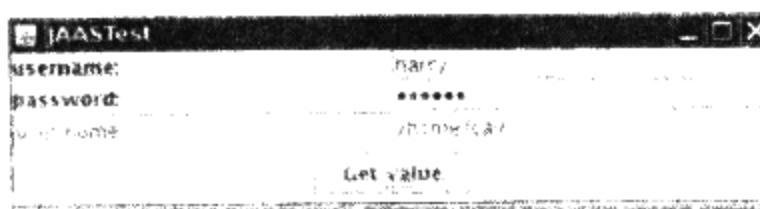


图9-10 一个定制的登录模块

该handler有一个简单的方法handle，用于处理Callback对象数组，有很多预定义类，比如NameCallback和PasswordCallback等，都实现了Callback接口。也可以添加自己的类，比如RetinaScanCallback等。下面这段handler代码可能有些不雅致，因为它要分析callback对象的类型：

```
public void handle(Callback[] callbacks)
{
    for (Callback callback : callbacks)
    {
        if (callback instanceof NameCallback) . .
        else if (callback instanceof PasswordCallback) . .
        else . .
    }
}
```

登录模块提供callback数组以满足认证的需要。

```
NameCallback nameCall = new NameCallback("username: ");
PasswordCallback passCall = new PasswordCallback("password: ", false);
callbackHandler.handle(new Callback[] { nameCall, passCall });
```

然后它从callback中获取所要的信息。

程序清单9-12中的程序将显示一个表格，用于输入登录信息和系统属性名。如果用户通过了认证，属性值可以从PrivilegedAction中取出。从程序清单9-13的策略文件中可以看到，具有admin角色的用户具有对属性的读取权限。

正如前一节中所讲到的，必须将登录和操作代码分开。首先创建两个JAR文件：

```
javac *.java
jar cvf login.jar JAAS*.class Simple*.class
jar cvf action.jar SysPropAction.class
```

然后以下方式运行程序：

```
java -classpath login.jar:action.jar
```

```
-Djava.security.policy=JAASTest.policy  
-Djava.security.auth.login.config=jaas.config  
JAASTest
```

程序清单9-14说明了登录的配置。

 **注意：**如果登录配置文件中所有的模块都认证成功，该登录才会被提交。通过这种方式，就可以支持一个更加复杂的两阶段协议。更多详细信息，请参阅下面地址的登录模块开发指南：<http://java.sun.com/j2se/5.0/docs/guide/security/jaas/JAASLMDevGuide.html>。

### 程序清单9-9 SimplePrincipal.java

```
1. import java.security.*;  
2.  
3. /**  
4. * A principal with a named value (such as "role=HR" or "username=harry").  
5. * @version 1.0 2004-09-14  
6. * @author Cay Horstmann  
7. */  
8. public class SimplePrincipal implements Principal  
9. {  
10.    /**  
11.     * Constructs a SimplePrincipal to hold a description and a value.  
12.     * @param roleName the role name  
13.    */  
14.    public SimplePrincipal(String descr, String value)  
15.    {  
16.        this.descr = descr;  
17.        this.value = value;  
18.    }  
19.  
20.    /**  
21.     * Returns the role name of this principal  
22.     * @return the role name  
23.    */  
24.    public String getName()  
25.    {  
26.        return descr + "=" + value;  
27.    }  
28.  
29.    public boolean equals(Object otherObject)  
30.    {  
31.        if (this == otherObject) return true;  
32.        if (otherObject == null) return false;  
33.        if (getClass() != otherObject.getClass()) return false;  
34.        SimplePrincipal other = (SimplePrincipal) otherObject;  
35.        return getName().equals(other.getName());  
36.    }  
37.  
38.    public int hashCode()  
39.    {  
40.        return getName().hashCode();  
41.    }  
42.  
43.    private String descr;  
44.    private String value;
```

---

45. }**程序清单9-10 SimpleLoginModule.java**

```
1. import java.io.*;
2. import java.security.*;
3. import java.util.*;
4. import javax.security.auth.*;
5. import javax.security.auth.callback.*;
6. import javax.security.auth.login.*;
7. import javax.security.auth.spi.*;
8.
9. /**
10. * This login module authenticates users by reading usernames, passwords, and roles from a
11. * text file.
12. * @version 1.0 2004-09-14
13. * @author Cay Horstmann
14. */
15. public class SimpleLoginModule implements LoginModule
16. {
17.     public void initialize(Subject subject, CallbackHandler callbackHandler,
18.                           Map<String, ?> sharedState, Map<String, ?> options)
19.     {
20.         this.subject = subject;
21.         this.callbackHandler = callbackHandler;
22.         this.options = options;
23.     }
24.
25.     public boolean login() throws LoginException
26.     {
27.         if (callbackHandler == null) throw new LoginException("no handler");
28.
29.         NameCallback nameCall = new NameCallback("username: ");
30.         PasswordCallback passCall = new PasswordCallback("password: ", false);
31.         try
32.         {
33.             callbackHandler.handle(new Callback[] { nameCall, passCall });
34.         }
35.         catch (UnsupportedCallbackException e)
36.         {
37.             LoginException e2 = new LoginException("Unsupported callback");
38.             e2.initCause(e);
39.             throw e2;
40.         }
41.         catch (IOException e)
42.         {
43.             LoginException e2 = new LoginException("I/O exception in callback");
44.             e2.initCause(e);
45.             throw e2;
46.         }
47.
48.         return checkLogin(nameCall.getName(), passCall.getPassword());
49.     }
50.
51. /**
52. * Checks whether the authentication information is valid. If it is, the subject acquires
```

```

53.     * principals for the user name and role.
54.     * @param username the user name
55.     * @param password a character array containing the password
56.     * @return true if the authentication information is valid
57.     */
58.    private boolean checkLogin(String username, char[] password) throws LoginException
59.    {
60.        try
61.        {
62.            Scanner in = new Scanner(new FileReader("") + options.get("pwfile")));
63.            while (in.hasNextLine())
64.            {
65.                String[] inputs = in.nextLine().split("\\|");
66.                if (inputs[0].equals(username) && Arrays.equals(inputs[1].toCharArray(), password))
67.                {
68.                    String role = inputs[2];
69.                    Set<Principal> principals = subject.getPrincipals();
70.                    principals.add(new SimplePrincipal("username", username));
71.                    principals.add(new SimplePrincipal("role", role));
72.                    return true;
73.                }
74.            }
75.            in.close();
76.            return false;
77.        }
78.        catch (IOException e)
79.        {
80.            LoginException e2 = new LoginException("Can't open password file");
81.            e2.initCause(e);
82.            throw e2;
83.        }
84.    }
85.
86.    public boolean logout()
87.    {
88.        return true;
89.    }
90.
91.    public boolean abort()
92.    {
93.        return true;
94.    }
95.
96.    public boolean commit()
97.    {
98.        return true;
99.    }
100.
101.    private Subject subject;
102.    private CallbackHandler callbackHandler;
103.    private Map<String, ?> options;
104. }
```

### 程序清单9-11 SimpleCallbackHandler.java

```
1. import javax.security.auth.callback.*;
```

```
2.
3. /**
4. * This simple callback handler presents the given user name and password.
5. * @version 1.0 2004-09-14
6. * @author Cay Horstmann
7. */
8. public class SimpleCallbackHandler implements CallbackHandler
9. {
10.    /**
11.     * Constructs the callback handler.
12.     * @param username the user name
13.     * @param password a character array containing the password
14.     */
15.    public SimpleCallbackHandler(String username, char[] password)
16.    {
17.        this.username = username;
18.        this.password = password;
19.    }
20.
21.    public void handle(Callback[] callbacks)
22.    {
23.        for (Callback callback : callbacks)
24.        {
25.            if (callback instanceof NameCallback)
26.            {
27.                ((NameCallback) callback).setName(username);
28.            }
29.            else if (callback instanceof PasswordCallback)
30.            {
31.                ((PasswordCallback) callback).setPassword(password);
32.            }
33.        }
34.    }
35.
36.    private String username;
37.    private char[] password;
38.}
```

### 程序清单9-12 JAATest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.security.auth.*;
4. import javax.security.auth.login.*;
5. import javax.swing.*;
6.
7. /**
8. * This program authenticates a user via a custom login and then executes the SysPropAction
9. * with the user's privileges.
10. * @version 1.0 2004-09-14
11. * @author Cay Horstmann
12. */
13. public class JAATest
14. {
15.    public static void main(final String[] args)
16.    {
17.        System.setSecurityManager(new SecurityManager());
```

```
18.     EventQueue.invokeLater(new Runnable()
19.     {
20.         public void run()
21.         {
22.             JFrame frame = new JAASFrame();
23.             frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24.             frame.setVisible(true);
25.         }
26.     });
27. }
28. */
29.
30. /**
31. * This frame has text fields for user name and password, a field for the name of the requested
32. * system property, and a field to show the property value.
33. */
34. class JAASFrame extends JFrame
35. {
36.     public JAASFrame()
37.     {
38.         setTitle("JAASTest");
39.
40.         username = new JTextField(20);
41.         password = new JPasswordField(20);
42.         propertyName = new JTextField(20);
43.         PropertyValue = new JTextField(20);
44.         PropertyValue.setEditable(false);
45.
46.         JPanel panel = new JPanel();
47.         panel.setLayout(new GridLayout(0, 2));
48.         panel.add(new JLabel("username:"));
49.         panel.add(username);
50.         panel.add(new JLabel("password:"));
51.         panel.add(password);
52.         panel.add(propertyName);
53.         panel.add(PropertyValue);
54.         add(panel, BorderLayout.CENTER);
55.
56.         JButton getValueButton = new JButton("Get Value");
57.         getValueButton.addActionListener(new ActionListener()
58.         {
59.             public void actionPerformed(ActionEvent event)
60.             {
61.                 getValue();
62.             }
63.         });
64.         JPanel buttonPanel = new JPanel();
65.         buttonPanel.add(getValueButton);
66.         add(buttonPanel, BorderLayout.SOUTH);
67.         pack();
68.     }
69.
70.     public void getValue()
71.     {
72.         try
73.         {
```

```
74.     LoginContext context = new LoginContext("Login1", new SimpleCallbackHandler(username
75.         .getText(), password.getPassword()));
76.     context.login();
77.     Subject subject = context.getSubject();
78.     propertyName.setText("")
79.         + Subject.doAsPrivileged(subject, new SysPropAction(propertyName.getText(), null));
80.     context.logout();
81. }
82. catch (LoginException e)
83. {
84.     JOptionPane.showMessageDialog(this, e);
85. }
86. }
87.
88. private JTextField username;
89. private JPasswordField password;
90. private JTextField propertyName;
91. private JTextField PropertyValue;
92. }
```

### 程序清单9-13 JAATest.policy

```
1. grant codebase "file:login.jar"
2. {
3.     permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
4.     permission javax.security.auth.AuthPermission "createLoginContext.Login1";
5.     permission javax.security.auth.AuthPermission "doAsPrivileged";
6.     permission javax.security.auth.AuthPermission "modifyPrincipals";
7.     permission java.io.FilePermission "password.txt", "read";
8. };
9.
10. grant principal SimplePrincipal "role=admin"
11. {
12.     permission java.util.PropertyPermission "*", "read";
13. };
```

### 程序清单9-14 jaas.config

```
1. Login1
2. {
3.     SimpleLoginModule required pwfile="password.txt";
4. };
```

## API javax.security.auth.callback.CallbackHandler 1.4

- void handle(Callback[] callbacks)

处理给定的callback，与期望的用户进行交互，并且将安全信息存储到callback对象中。

## API javax.security.auth.callback.NameCallback 1.4

- NameCallback(String prompt)
- NameCallback(String prompt, String defaultName)

用给定的提示符和默认的名字构建一个NameCallback。

- `void setName(String name)`
- `String getName()`  
设置或者获取该callback所收集到的名字。
- `String getPrompt()`  
获取查询该名字时所使用的提示符。
- `String getDefaultName()`  
获取查询该名字时所使用的默认名字。

**API** `javax.security.auth.callback.PasswordCallback 1.4`

- `PasswordCallback(String prompt, boolean echoOn)`  
用给定提示符和回显标记构建一个PasswordCallback。
- `void setPassword(char[] password)`
- `char[] getPassword()`  
设置或者获取该callback所收集到的密码。
- `String getPrompt()`  
获取查询该密码时所使用的提示符。
- `boolean isEchoOn()`  
获取查询该密码时所使用的回显标记。

**API** `javax.security.auth.spi.LoginModule 1.4`

- `void initialize(Subject subject, CallbackHandler handler, Map<String,?> sharedState, Map<String,?> options)`  
为了认证给定的subject，初始化该LoginModule。在登录处理期间，用给定的handler来收集登录信息。使用sharedState映射表与其他登录模块进行通信。options映射表包含该模块实例的登录配置中指定的名/值对。
- `boolean login()`  
执行认证过程，并组装主体的特征集。如果登录成功，则返回true。
- `boolean commit()`  
对于需要两阶段提交的登录场景，当所有的登录模块都成功后，调用该方法。如果操作成功，则返回true。
- `boolean abort()`  
如果某一登录模块失败导致登录过程中断，就调用该方法。如果操作成功，则返回true。
- `boolean logout()`  
注销当前的主体。如果操作成功，则返回true。

## 9.5 数字签名

正如我们前面所说，applet是在Java平台上开始流行起来的。实际上，人们发现尽管他们可以编写出像著名的“nervous text”那样栩栩如生的applet，但是在JDK 1.0 安全模式下无法发挥

其一整套非常有用的作用。例如，由于JDK 1.0下的applet要受到严密的监督，因此，即使applet在公司安全内部网上运行时风险相对较小，applet也无法在企业内部网上发挥很大的作用。Sun公司很快就认识到，要使applet真正变得非常有用，用户必须可以根据applet的来源为其分配不同的安全级别。如果applet来自值得信赖的提供商，并且没有被篡改过，那么applet的用户就可以决定是否给applet授予更多的运行特权。

如果要给予applet更多的信赖，你必须知道下面两件事：

- 1) applet来自哪里？
- 2) 在传输过程中代码是否被破坏？

在过去的50年里，数学家和计算机科学家已经开发出各种各样成熟的算法，用于确保数据和电子签名的完整性，在java.security包中包含了许多这些算法的实现，而且幸运的是，你无需掌握相应的数学基础知识，就可以使用java.security包中的算法。在下面几节中，我们将要介绍消息摘要是如何检测数据文件中的变化的，以及数字签名是如何证明签名者的身份的。

### 9.5.1 消息摘要

消息摘要（message digest）是数据块的数字指纹。例如，所谓的SHA1(安全散列算法#1)可将任何数据块，无论其数据有多长，都压缩为160位（20字节）的序列。与真实的指纹一样，人们希望任何两条消息都不会有相同的SHA1指纹。当然这是不可能的——因为只存在 $2^{160}$ 个SHA1指纹，所以肯定会有某些消息具有相同的指纹。因为 $2^{160}$ 是一个很大的数字，所以存在重复指纹的可能性微乎其微，那么这种重复的可能性到底小到什么程度呢？根据James Walsh在他的《True Odds: How Risks Affect Your Everyday Life》，Merritt Publishing出版社1996年出版，一书中所叙述的，人死于雷击的概率为三万分之一。现在，假设有9个人，比如你不喜欢的9个经理或者教授，你和他们所有的人都死于雷击的概率，比伪造的消息与原有消息具有相同的SHA1指纹的概率还要高。（当然，可能有你不认识的其他10个以上的人会死于雷击，但这里我们讨论的是你选择的特定的人的死亡概率。）

消息摘要具有两个基本属性：

- 1) 如果数据的1位或者几位改变了，那么消息摘要也将改变。
- 2) 拥有给定消息的伪造者不能创建与原消息具有相同摘要的假消息。

当然，第二个属性又是一个概率问题。让我们来看看下面这位亿万富翁留下的遗嘱：

“我死了之后，我的财产将由我的孩子平分，但是，我的儿子George应该拿不到一个子。”

这份遗嘱的SHA1指纹为：

2D 8B 35 F3 BF 49 CD B1 94 04 E0 66 21 2B 5E 57 70 49 E1 7E

这位有疑心病的父亲将这份遗嘱交给一位律师保存，而将指纹交给另一位律师保存。现在，假设George能够贿赂那位保存遗嘱的律师，他想修改这份遗嘱，使得Bill一无所得。当然，这需要将原指纹改为下面这样完全不同的位模式：

2A 33 0B 4B B3 FE CC 1C 9D 5C 01 A7 09 51 0B 49 AC 8F 98 92

那么George能够找到与该指纹相匹配的其他文字吗？如果从地球形成之时，他就很自豪地拥有10亿台计算机，每台计算机每秒钟能处理一百万条信息，他依然无法找到一个能够替

换的遗嘱。

人们已经设计出大量的算法，用于计算这些消息摘要，其中最著名的两种算法是SHA1和MD5。SHA1是由美国国家标准和技术学会开发的加密散列算法，MD5是由麻省理工学院的Ronald Rivest发明的算法。这两种算法都使用了独特巧妙的方法对消息中的各个位进行扰乱。如果要了解这些方法的详细信息，请参阅William Stallings撰写的《Cryptography and Network Security》一书，该书由Prentice Hall出版社于2005年出版。值得注意的是，最近人们在这两种算法中发现了某些微妙的规律性，因此许多密码人员建议最好避免使用MD5，而应该使用SHA1算法，直到有更强的加密算法出现。（查看<http://www.rsa.com/rsalabs/node.asp?id=2834>以了解更多的信息）。

Java编程语言已经实现了SHA1和MD5。MessageDigest类是用于创建封装了指纹算法的对象的“工厂”，它的静态方法getInstance返回继承了MessageDigest类的某个类的对象。这意味着MessageDigest类能够承担下面的双重职责：

- 作为一个工厂类。
- 作为所有消息摘要算法的超类。

例如，下面是如何获取一个能够计算SHA指纹的对象的方法：

```
MessageDigest alg = MessageDigest.getInstance("SHA-1");
```

（如果要获取计算MD5的对象，请使用字符串“MD5”作为getInstance的参数。）

当你已经获取MessageDigest对象之后，通过反复调用update方法，将信息中的所有字节提供给该对象。例如，下面的代码将文件中的所有字节传给上面建立的alg对象，以执行指纹算法：

```
InputStream in = . . .
int ch;
while ((ch = in.read()) != -1)
    alg.update((byte) ch);
```

另外，如果这些字节存放在一个数组中，那就  
可以一次完成整个数组的更新：

```
byte[] bytes = . . .;
alg.update(bytes);
```

当完成上述操作后，调用digest方法。该方法填充输入信息——指纹算法需要的——并且进行相应的计算，然后以字节数组的形式返回消息摘要。

```
byte[] hash = alg.digest();
```

程序清单9-15中的程序计算了一个消息摘要，既可以用SHA，也可以使用MD5来计算。可以从文件加载需要计算摘要的数据，也可以直接将信息输入文本区域。图9-11显示了该应用程序的画面。

### 程序清单9-15 MessageDigestTest.java

```
1 import java.io.*;
2 import java.security.*;
```



图9-11 计算一个消息摘要

```
3. import java.awt.*;
4. import java.awt.event.*;
5. import javax.swing.*;
6.
7. /**
8. * This program computes the message digest of a file or the contents of a text area.
9. * @version 1.13 2007-10-06
10. * @author Cay Horstmann
11. */
12. public class MessageDigestTest
13. {
14.     public static void main(String[] args)
15.     {
16.         EventQueue.invokeLater(new Runnable()
17.         {
18.             public void run()
19.             {
20.                 JFrame frame = new MessageDigestFrame();
21.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22.                 frame.setVisible(true);
23.             }
24.         });
25.     }
26. }
27.
28. /**
29. * This frame contains a menu for computing the message digest of a file or text area, radio
30. * buttons to toggle between SHA-1 and MD5, a text area, and a text field to show the
31. * message digest.
32. */
33. class MessageDigestFrame extends JFrame
34. {
35.     public MessageDigestFrame()
36.     {
37.         setTitle("MessageDigestTest");
38.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
39.
40.         JPanel panel = new JPanel();
41.         ButtonGroup group = new ButtonGroup();
42.         addRadioButton(panel, "SHA-1", group);
43.         addRadioButton(panel, "MD5", group);
44.
45.         add(panel, BorderLayout.NORTH);
46.         add(new JScrollPane(message), BorderLayout.CENTER);
47.         add(digest, BorderLayout.SOUTH);
48.         digest.setFont(new Font("Monospaced", Font.PLAIN, 12));
49.
50.         setAlgorithm("SHA-1");
51.
52.         JMenuBar menuBar = new JMenuBar();
53.         JMenu menu = new JMenu("File");
54.         JMenuItem fileDigestItem = new JMenuItem("File digest");
55.         fileDigestItem.addActionListener(new ActionListener()
56.         {
57.             public void actionPerformed(ActionEvent event)
58.             {
59.                 loadFile();
```

```
60.        }
61.    });
62.    menu.add(fileDigestItem);
63.    JMenuItem textDigestItem = new JMenuItem("Text area digest");
64.    textDigestItem.addActionListener(new ActionListener()
65.    {
66.        public void actionPerformed(ActionEvent event)
67.        {
68.            String m = message.getText();
69.            computeDigest(m.getBytes());
70.        }
71.    });
72.    menu.add(textDigestItem);
73.    menuBar.add(menu);
74.    setJMenuBar(menuBar);
75. }
76.
77. /**
78. * Adds a radio button to select an algorithm.
79. * @param c the container into which to place the button
80. * @param name the algorithm name
81. * @param g the button group
82. */
83. public void addRadioButton(Container c, final String name, ButtonGroup g)
84. {
85.     ActionListener listener = new ActionListener()
86.     {
87.         public void actionPerformed(ActionEvent event)
88.         {
89.             setAlgorithm(name);
90.         }
91.     };
92.     JRadioButton b = new JRadioButton(name, g.getButtonCount() == 0);
93.     c.add(b);
94.     g.add(b);
95.     b.addActionListener(listener);
96. }
97.
98. /**
99. * Sets the algorithm used for computing the digest.
100. * @param alg the algorithm name
101. */
102. public void setAlgorithm(String alg)
103. {
104.     try
105.     {
106.         currentAlgorithm = MessageDigest.getInstance(alg);
107.         digest.setText("");
108.     }
109.     catch (NoSuchAlgorithmException e)
110.     {
111.         digest.setText("") + e);
112.     }
113. }
114.
115. /**
```

```
116.     * Loads a file and computes its message digest.  
117.     */  
118.    public void loadFile()  
119.    {  
120.        JFileChooser chooser = new JFileChooser();  
121.        chooser.setCurrentDirectory(new File("."));  
122.  
123.        int r = chooser.showOpenDialog(this);  
124.        if (r == JFileChooser.APPROVE_OPTION)  
125.        {  
126.            try  
127.            {  
128.                String name = chooser.getSelectedFile().getAbsolutePath();  
129.                computeDigest(loadBytes(name));  
130.            }  
131.            catch (IOException e)  
132.            {  
133.                JOptionPane.showMessageDialog(null, e);  
134.            }  
135.        }  
136.    }  
137.  
138.    /**  
139.     * Loads the bytes in a file.  
140.     * @param name the file name  
141.     * @return an array with the bytes in the file  
142.     */  
143.    public byte[] loadBytes(String name) throws IOException  
144.    {  
145.        FileInputStream in = null;  
146.  
147.        in = new FileInputStream(name);  
148.        try  
149.        {  
150.            ByteArrayOutputStream buffer = new ByteArrayOutputStream();  
151.            int ch;  
152.            while ((ch = in.read()) != -1)  
153.                buffer.write(ch);  
154.            return buffer.toByteArray();  
155.        }  
156.        finally  
157.        {  
158.            in.close();  
159.        }  
160.    }  
161.  
162.    /**  
163.     * Computes the message digest of an array of bytes and displays it in the text field.  
164.     * @param b the bytes for which the message digest should be computed.  
165.     */  
166.    public void computeDigest(byte[] b)  
167.    {  
168.        currentAlgorithm.reset();  
169.        currentAlgorithm.update(b);  
170.        byte[] hash = currentAlgorithm.digest();  
171.        String d = "";
```

```

172.     for (int i = 0; i < hash.length; i++)
173.     {
174.         int v = hash[i] & 0xFF;
175.         if (v < 16) d += "0";
176.         d += Integer.toString(v, 16).toUpperCase() + " ";
177.     }
178.     digest.setText(d);
179. }
180.
181. private JTextArea message = new JTextArea();
182. private JTextField digest = new JTextField();
183. private MessageDigest currentAlgorithm;
184. private static final int DEFAULT_WIDTH = 400;
185. private static final int DEFAULT_HEIGHT = 300;
186. }
```

### **API** java.security.MessageDigest 1.1

- static MessageDigest getInstance(String algorithm Name)

返回实现特定算法的MessageDigest对象。如果没有提供该算法，则抛出一个NoSuchAlgorithmException异常。

- void update(byte input)
- void update(byte[] input)
- void update(byte[] input, int offset, int len)  
使用指定的字节来更新摘要。
- byte[] digest()  
完成散列计算，返回计算所得的摘要，并复位算法对象。
- void reset()  
复位摘要。

## 9.5.2 消息签名

在上一节中，我们介绍了如何计算原始消息的消息摘要和指纹的方法。如果消息改变了，那么改变后的消息的指纹与原消息的指纹将不匹配。如果消息和它的指纹是分开传送的，那么接收者就可以检查消息是否被篡改过。但是，如果消息和指纹同时被截获了，对消息进行修改，再重新计算指纹，这是一件很容易的事情。毕竟，消息摘要算法是公开的，不需要使用任何密钥。在这种情况下，假消息和新指纹的接收者永远不会知道消息已经被篡改。数字签名解决了这个问题。

为了了解数字签名的工作原理，我们需要解释关于公共密钥加密技术领域中的几个概念。公共密钥加密技术是基于公共密钥和私有密钥这个两个基本概念的。它的设计思想是你可以将公共密钥告诉世界上的任何人，但是，只有自己才拥有私有密钥，重要的是你要保护你的私有密钥，不将它泄漏给其他任何人。这些密钥之间存在一定的数学关系，但是这种关系的具体性质对于实际的编程来说并不重要。（如果你有兴趣，可以参阅<http://www.cacr.math.uwaterloo.ca/hac/>站点上的《The Handbook of Applied Cryptography》一书。）

密钥非常长，而且很复杂。例如，下面是一对匹配的数字签名算法（DSA）公共密钥和私有密钥。

公共密钥：

p:  
fca682ce8e12caba26efccf7110e526db078b05edecbcd1eb4a208f3ae1617ae01f35b91a47e6df63413c5e12ed0899  
bcd132acd50d99151bdc43ee737592e17  
  
q: 962eddcc369cba8ebb260ee6b6a126d9346e38c5  
  
g:678471b27a9cf44ee91a49c5147db1a9aaf244f05a434d6486931d2d14271b9e35030b71fd73da179069b32e29356  
30e  
1c2062354d0da20a6c416e50be794ca4  
  
y:  
c0b6e67b4ac098eb1a32c5f8c4c1f0e7e6fb9d832532e27d0bdab9ca2d2a8123ce5a8018b8161a760480fadd040b927  
281ddb22cb9bc4df596d7de4d1b977d50

私有密钥：

p:  
fca682ce8e12caba26efccf7110e526db078b05edecbcd1eb4a208f3ae1617ae01f35b91a47e6df63413c5e12ed0899  
bcd132acd50d99151bdc43ee737592e17  
  
q: 962eddcc369cba8ebb260ee6b6a126d9346e38c5  
  
g:  
678471b27a9cf44ee91a49c5147db1a9aaf244f05a434d6486931d2d14271b9e35030b71fd73da179069b32e2935630  
e1c2062354d0da20a6c416e50be794ca4  
  
x: 146c09f881656cc6c51f27ea6c3a91b85ed1d70a

在现实中，几乎不可能用一个密钥去推算出另一个密钥。也就是说，即使每个人都知道你的公共密钥，不管他们拥有多少计算资源，他们一辈子也无法计算出你的私有密钥。

任何人都无法根据公共密钥来推算私有密钥，这似乎让人难以置信。但是时至今日，还没有人能够找到一种算法，来为现在常用的加密算法进行这种推算。如果密钥足够长，那么要是使用穷举法——也就是直接试验所有可能的密钥——所需要的计算机将比用太阳系中的所有原子来制造的计算机还要多，而且还得花费数千年的时间。当然，可能会有人提出比穷举更灵活的计算密钥的算法。例如，RSA算法（该加密算法由Rivest, Shamir和Adleman发明）就利用了对数值巨大的数字进行因子分解的困难性。在最近20年里，许多优秀的数学家都在尝试提出好的因子分解算法，但是迄今为止都没有成功。据此，大多数密码学者认为，拥有2000位或者更多位“模数”的密钥目前是完全安全的，可以抵御任何攻击。DSA被认为具有类似的安全性。

图9-12展示了这项工作的处理过程。

假设Alice想要给Bob发送一个消息，Bob想知道该消息是否来自Alice，而不是冒名顶替者。Alice写好了消息，并且用她的私有密钥对该消息摘要签名。Bob得到了她的公共密钥的拷贝，然后Bob用公共密钥对该签名进行校验。如果通过了校验，则Bob可以确认以下两个事实：

- 1) 原始消息没有被篡改过。
- 2) 该消息是由Alice签名的，她是私有密钥的持有者，该私有密钥就是与Bob用于校验的公共密钥相匹配的密钥。

你可以看到私有密钥的安全性为什么是最重要的。如果某个人偷了Alice的私有密钥，或者政府要求她交出私有密钥，那么她就麻烦了。小偷或者政府代表就可以假扮她的身份来发送消息和资金转账指令等等，而其他人则会相信这些消息确实来自于Alice。

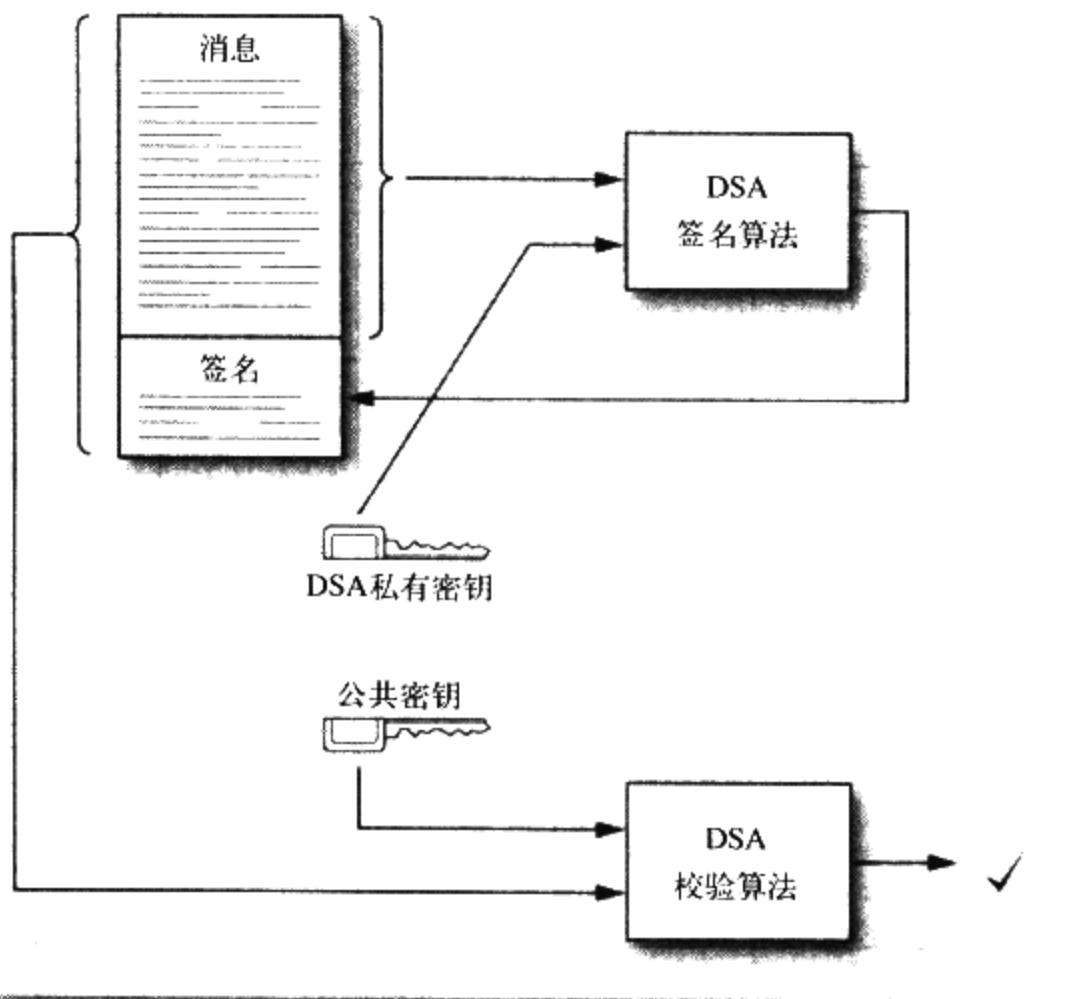


图9-12 使用DSA进行公共密钥签名的交换

### 9.5.3 X.509证书格式

为了利用公共密钥这种密码系统，必须将公共密钥分发出去。最通用的一种签名证书格式称为X.509格式。X.509格式的证书被VeriSign、微软、网景和其他许多公司广泛应用于对电子邮件消息进行签名，对程序代码进行认证，以及对许多其他类型的数据进行认证等等。X.509标准是由国际电话标准机构，即国际电报电话咨询委员会（CCITT）提出的用于目录服务的X.500系列建议的组成部分。

X.509证书的具体结构是用一种形式化表示来描述的，称为“抽象语法表示法#1”（abstract syntax notation）即ASN.1。图9-13显示了第3版X.509格式的ASN.1定义。虽然具体的语法对我们并不重要，但是你可以看到，ASN.1为证书文件的结构给出了精确的定义。“基本编码规则”（basic encoding rules），即BER，精确地描述了如何将该结构保存为二进制文件。也就是说，BER描述了如何对整数、字符串、位串以及诸如SEQUENCE、CHOICE和OPTIONAL的结构进行编码的方法。

**注意：**如果要了解关于ASN.1的更多信息——请参阅Burton S. Kaliski, Jr.撰写的《A Layman's Guide to a Subset of ASN.1, BER, and DER》，该书的网址为`ftp://ftp.rsa.com/pub/pkcs/ps/layman.ps`。Olivier Dubuisson撰写的《ASN.1 Communication Between Heterogeneous Systems》（该书由Academic Press出版社于2000年出版，该书的网址为`http://www.oss.com/asn1/dubuisson.html`）以及John Larmouth撰写的《ASN.1 Complete》（该书

由Morgan Kaufmann Publishers出版社于1999年出版，该书的网址为http://www.nokalva.com/asn1/larmouth.html)。

```
[Certificate ::= SEQUENCE {
    tbsCertificate     TBSCertificate,
    signatureAlgorithm AlgorithmIdentifier,
    signature          BIT STRING }

TBSCertificate ::= SEQUENCE {
    version           [0]  EXPLICIT Version DEFAULT v1,
    serialNumber      CertificateSerialNumber,
    signature         AlgorithmIdentifier,
    issuer            Name,
    validity          Validity,
    subject           Name,
    subjectPublicKeyInfo SubjectPublicKeyInfo,
    issuerUniqueID   [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                        -- If present, version must be v2 or v3
    subjectUniqueID  [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                        -- If present, version must be v2 or v3
    extensions        [3]  EXPLICIT Extensions OPTIONAL
                        -- If present, version must be v3
}

Version ::= INTEGER { v1(0), v2(1), v3(2) }

CertificateSerialNumber ::= INTEGER

Validity ::= SEQUENCE {
    notBefore       CertificateValidityDate,
    notAfter        CertificateValidityDate }

CertificateValidityDate ::= CHOICE {
    utcTime         UTCTime,
    generalTime     GeneralizedTime }

UniqueIdentifier ::= BIT STRING

SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm       AlgorithmIdentifier,
    subjectPublicKey BIT STRING }

Extensions ::= SEQUENCE OF Extension

Extension ::= SEQUENCE {
    extnID          OBJECT IDENTIFIER,
    critical        BOOLEAN DEFAULT FALSE,
    extnValue       OCTET STRING }
```

图9-13 X.509v3的ASN.1定义

#### 9.5.4 校验签名

JDK配有一个keytool程序，该程序是一个命令行工具，用于生成和管理一组证书。我们猜想，该工具的功能最终将会被嵌入到其他的、使用更加方便的程序中去。但我们现在要做的是，使用keytool工具来展示Alice是如何对一个文档进行签名并且将它发送给Bob的，而Bob又是如何校验该文档确实是由Alice签名，而不是冒名顶替的。

keytool程序负责管理密钥库、证书数据库和私有密钥。密钥库中的每一项都有一个“别

名”。下面展示的是Alice如何创建一个密钥库alice.certs并且用别名生成一个密钥对的。

```
keytool -genkeypair -keystore alice.certs -alias alice
```

当新建或者打开一个密钥库时，系统将提示你输入密钥库口令，在下面的这个例子中，口令就使用Secret，如果你要将keytool生成的密钥库用于重要的应用，那么你需要选择一个好的口令来保护这个文件。

当生成一个密钥时，系统提示你输入下面这些信息：

```
Enter keystore password: secret
Reenter new password: secret
What is your first and last name?
[Unknown]: Alice Lee
What is the name of your organizational unit?
[Unknown]: Engineering Department
What is the name of your organization?
[Unknown]: ACME Software
What is the name of your City or Locality?
[Unknown]: San Francisco
What is the name of your State or Province?
[Unknown]: CA
What is the two-letter country code for this unit?
[Unknown]: US
Is <CN=Alice Lee, OU=Engineering Department, O=ACME Software, L=San Francisco, ST=CA, C=US> correct?
[no]: yes
```

keytool工具使用X.500的区分式名字，它包含常用名(CN)、机构单位(OU)、机构(O)、地点(L)、州(ST)和国别(C)等成分，以确定密钥持有者和证书发行者的身份。

最后，必须设定一个密钥口令，或者按回车键，将密钥库口令作为密钥口令来使用。

假设Alice想把她的公共密钥提供给Bob，她必须导出一个证书文件：

```
keytool -exportcert -keystore alice.certs -alias alice -file alice.cer
```

这时，Alice就可以把证书发送给Bob。当Bob收到该证书时，他就可以将证书打印出来：

```
keytool -printcert -file alice.cer
```

打印的结果如下：

```
Owner: CN=Alice Lee, OU=Engineering Department, O=ACME Software, L=San Francisco, ST=CA, C=US
Issuer: CN=Alice Lee, OU=Engineering Department, O=ACME Software, L=San Francisco, ST=CA, C=US
Serial number: 470835ce
Valid from: Sat Oct 06 18:26:38 PDT 2007 until: Fri Jan 04 17:26:38 PST 2008
Certificate fingerprints:
    MD5: BC:18:15:27:85:69:48:B1:5A:C3:0B:1C:C6:11:B7:81
    SHA1: 31:0A:A0:B8:C2:88:3B:B6:85:7C:EF:C0:57:E5:94:95:61:47:6D:34
    Signature algorithm name: SHA1withDSA
    Version: 3
```

如果Bob想检查他是否得到了正确的证书，可以给Alice打电话，让她在电话里读出证书的指纹。

 注意：有些证书发放者将证书指纹公布在他们的网站上。例如，要检查jre/lib/security目录中的密钥库里的VeriSign公司的证书，可以使用-list选项：

```
keytool -list -v -keystore jre/lib/security/cacerts
```

该密钥库的口令是changeit。在该密钥库中有一个证书是：

```
Owner: OU=VeriSign Trust Network, OU="(c) 1998 VeriSign, Inc. - For authorized use only",
OU=Class 1 Public Primary Certification Authority - G2, O="VeriSign, Inc.", C=US
Issuer: OU=VeriSign Trust Network, OU="(c) 1998 VeriSign, Inc. - For authorized
use only", OU=Class 1 Public Primary Certification Authority - G2, O="VeriSign, Inc.",
C=US
Serial number: 4cc7eaaa983e71d39310f83d3a899192
Valid from: Sun May 17 17:00:00 PDT 1998 until: Tue Aug 01 16:59:59 PDT 2028
Certificate fingerprints:
MD5: DB:23:3D:F9:69:FA:4B:B9:95:80:44:73:5E:7D:41:83
SHA1: 27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47
```

通过访问网址<http://www.verisign.com/repository/root.html>，就可以核实该证书的有效性。  
一旦Bob信任该证书，他就可以将它导入密钥库中。

```
keytool -importcert -keystore bob.certs -alias alice -file alice.cer
```

 **警告：**绝对不要将你并不完全信任的证书导入到密钥库中。一旦证书添加到密钥库中，使用密钥库的任何程序都会认为这些证书可以用来对签名进行校验。

现在Alice就可以给Bob发送签过名的文档了。jarsigner工具负责对JAR文件进行签名和校验，Alice只需要将文档添加到要签名的JAR文件中。

```
jar cvf document.jar document.txt
```

然后她使用jarsigner工具将签名添加到文件中，她必须指定要使用的密钥库、JAR文件和密钥的别名。

```
jarsigner -keystore alice.certs document.jar alice
```

当Bob收到JAR文件时，他可以使用jarsigner程序的-verify选项，对文件进行校验。

```
jarsigner -verify -keystore bob.certs document.jar
```

Bob不需要设定密钥别名。该jarsigner程序会在数字签名中找到密钥所有者的X.500名字，并在密钥库中搜寻匹配的证书。

如果JAR文件没有受到破坏而且签名匹配，那么jarsigner程序将打印：

```
jar verified.
```

否则，程序将显示一个出错消息。

### 9.5.5 认证问题

假设你从朋友那接收到一个消息，该消息是你朋友用他的私有密钥签名的，使用的签名方法就是我们刚刚介绍的方法。你可能已经有了他的公共密钥，或者你能够容易地获得他的公共密钥，比如向他要一个密钥拷贝，或者从他的web页中获得密钥。这时，你就可以校验该消息是否是你朋友签过名的，并且有没有被破坏过。现在，假设你从一个声称代表某著名软件公司的陌生人那里获得了一个消息，他要求你运行消息附带的程序。这个陌生人甚至将他的公共密钥的拷贝发送给你，以便让你校验他是否是该消息的作者。你检查后会发现该签名是有效的，这就证明该消息是用匹配的私有密钥签名的，并且没有遭到破坏。

此时你要小心：你仍然不清楚谁写的这条消息。任何人都可以生成一对公共密钥和私有密

钥，再用私有密钥对消息进行签名，然后把签名好的消息和公共密钥发送给你。这种确定发送者身份的问题称为“认证问题”。

解决这个认证问题的通常做法是比较简单的。假设陌生人和你有一个你们俩都值得信赖的共同熟人。假设陌生人亲自约见了该熟人，将包含公共密钥的磁盘交给了他。后来，你的熟人与你见面，向你担保他与该陌生人见了面，并且该陌生人确实在那家著名的软件公司工作，然后将磁盘交给你（参见图9-14）。这样一来，你的熟人就证明了陌生人身份的真实性。

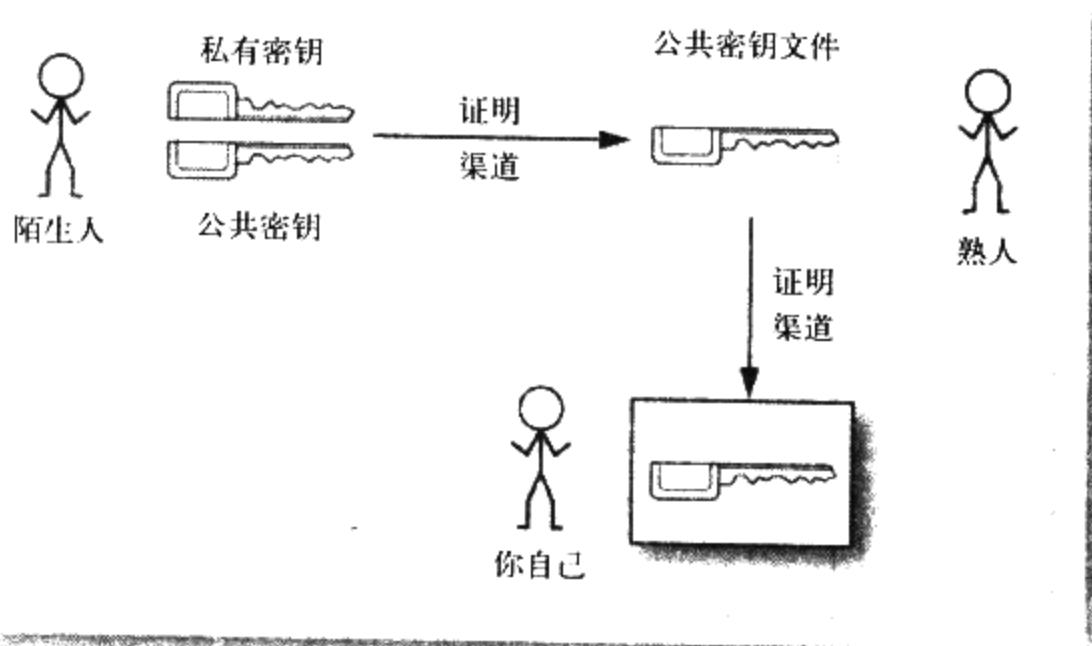


图9-14 通过一个值得信赖的中间人进行认证

事实上，你的熟人并不需要与你见面。取而代之的是，他可以将他的私有签名应用于陌生人的公共密钥文件之上即可（参见图9-15）。

当你拿到公共密钥文件之后，就可以检验你的熟人的签名是否真实，由于你信任他，因此你确信他在添加他的签名之前，确实核实了陌生人的身份。

然而，你们之间可能没有共同的熟人。有些信任模型假设你们之间总是存在一个“信任链”——即一个共同熟人的链路——这样你就可以信任该链中的每个成员。当然，实际情况并不总是这样。你可能信任你的熟人Alice，而且你知道Alice信任Bob，但是你不了解Bob，因此你没有把握究竟是不是该信任他。其他的信任模型则假设有一个我们大家都信任的慈善大佬，在扮演这个角色的公司中，最有名的是VeriSign公司 (<http://www.verisign.com>)。

你常常会遇到由负责担保他人身份的一个或多个实体签署的数字签名。你必须评估一下究竟能够在多大程度上信任这些身份认证人。你可能非常信赖VeriSign公司，因为也许你在许多网页中都看到过他们公司的标志，或者你曾经听说过，每当有新的万能密钥产生时，他们就会要求在一个非常保密的会议室中聚集众多揣着黑色公文包的人进行磋商。

然而，对于实际被认证的对象，你应该抱有一个符合实际的期望。在认证公共密钥时，VeriSign公司的CEO也不会亲自去会见每个人或者公司代表。直接在Web页面上填一份表格，并支付少量的费用，就可以获得一个“第一类（class 1）”ID。包含在证书中的密钥将被发送到指定的邮件地址。因此，你有理由相信该电子邮件是真实的，但是密钥申请人也可能填入任

意名字和机构。还有其他对身份信息的检验更加严格的ID类别。例如，如果是“第三类(class 3)”ID，VeriSign将要求密钥申请人必须进行身份公证，公证机构将要核实企业申请者的财务信用资质。其他认证机构将采用不同的认证程序。因此，当你收到一条经过认证的消息时，重要的是你应该明白它实际上认证了什么。

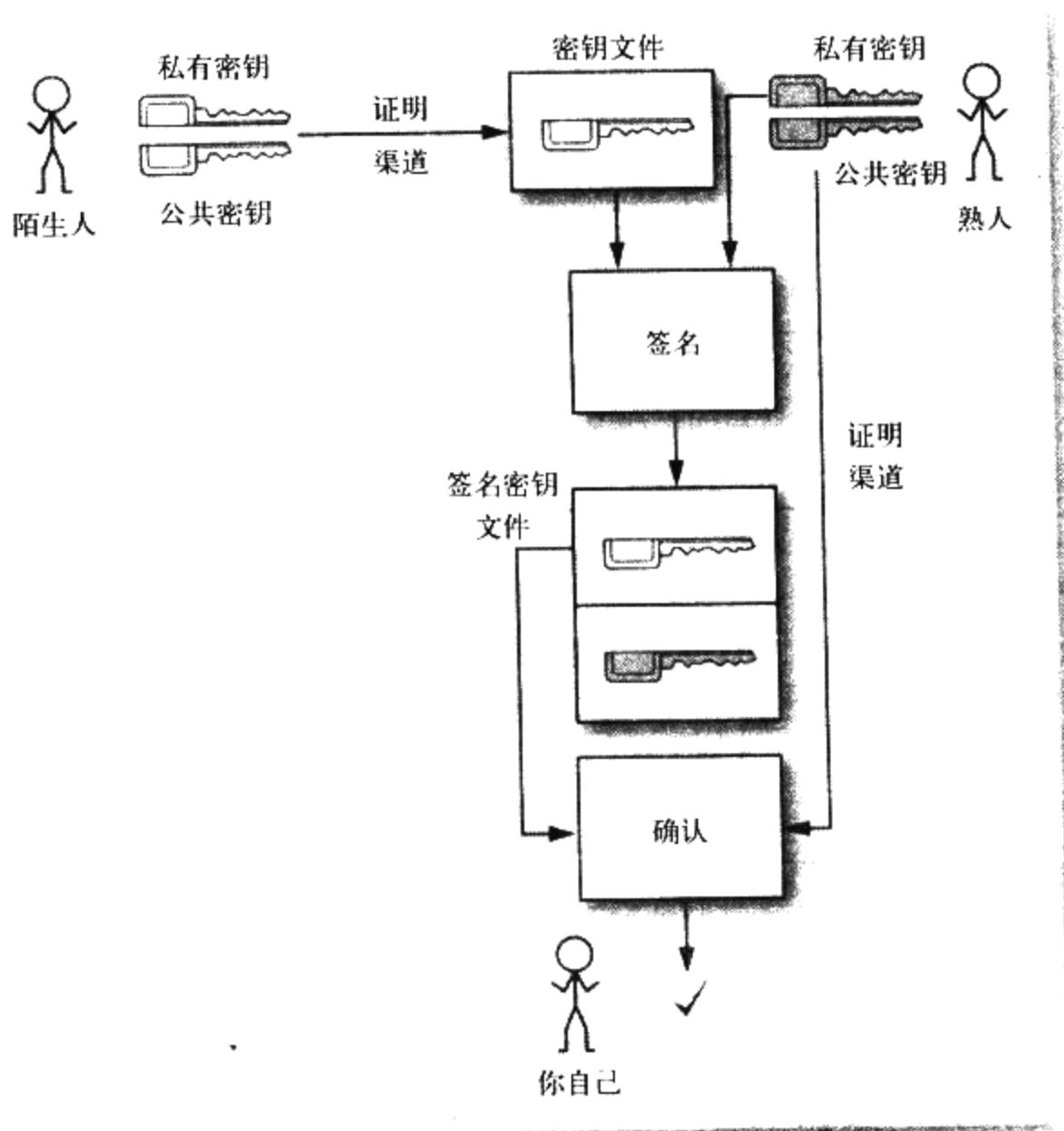


图9-15 通过受信赖的中间人的签名进行认证

### 9.5.6 证书签名

在“校验签名”一节中，你已经看到了Alice如何使用自签名的证书向Bob分发公共密钥。但是，Bob需要通过校验Alice的指纹以确保这个证书是有效的。

假设Alice想要给同事Cindy发送一条经过签名的消息，但是Cindy并不希望因为要校验许多签名指纹而受到困扰。因此，假设有一个Cindy信任的实体来校验这些签名。在这个例子中，Cindy信任ACME软件公司的信息资源部。

这个部门负责证书授权（CA）的运作。ACME的每个人在其密钥库中都有CA的公共密钥，这是由一个专门负责详细核查密钥指纹的系统管理员安装的。CA对ACME雇员的密钥进行签名，当他们在安装彼此的密钥时，密钥库将隐含地信任这些密钥，因为它们是由一个可信任的

密钥签名的。

下面显示了可以如何模仿这个过程，需要创建一个密钥库acmesoft.certs，生成一个密钥对并导出公共密钥。

```
keytool -genkeypair -keystore acmesoft.certs -alias acmeroot  
keytool -exportcert -keystore acmesoft.certs -alias acmeroot -file acmeroot.cer
```

其中的公共密钥被导入到了一个自签名的证书中，然后将其添加到每个雇员的密钥库中：

```
keytool -importcert -keystore cindy.certs -alias acmeroot -file acmeroot.cer
```

如果Alice要发送消息给Cindy以及ACME软件公司的其他任何人，她需要将她自己的证书签名一并提交给信息资源部。但是，这个功能在keytool程序中是缺失的。在本书附带的代码中，我们提供了一个CertificateSigner类来弥补这个问题。ACME软件公司的授权机构成员将负责核实Alice的身份，并且生成如下的签名证书：

```
java CertificateSigner -keystore acmesoft.certs -alias acmeroot  
-infile alice.cer -outfile alice_signedby_acmeroot.cer
```

证书签名者程序必须拥有对ACME软件公司密钥库的访问权限，并且该公司成员必须知道密钥库的口令，显然这是一项敏感的操作。

现在Alice将文件alice\_signedby\_acmeroot.cert交给Cindy和ACME软件公司的其他任何人。或者，ACME软件公司直接将该文件存储在公司的目录中。请记住，该文件包含了Alice的公共密钥和ACME软件公司的声明，证明该密钥确实属于Alice。

现在，Cindy将签名的证书导入到她的密钥库中：

```
keytool -importcert -keystore cindy.certs -alias alice -file alice_signedby_acmeroot.cer
```

密钥库要进行校验，以确定该密钥是由密钥库中已有的受信赖的根密钥签过名的，而且Cindy不必对证书的指纹进行校验。

一旦Cindy添加了根证书和经常给她发送文档的人的证书后，她就再也不用担心密钥库了。

### 9.5.7 证书请求

在前一节中，我们用密钥库和CertificateSigner工具模拟了一个CA。但是，大多数CA都运行着更加复杂的软件来管理证书，并且使用的证书格式也略有不同。本节将展示与这些软件包进行交互时需要增加的处理步骤。

我们将用OpenSSL软件包作为实例。许多Linux系统和Mac OS X都预装了这个软件，并且Cygwin端口也可用这个软件。你也可以到<http://www.openssl.org>网站下载。

为了创建一个CA，需要运行CA脚本，其确切位置依赖于你的操作系统。在Ubuntu上，运行

```
/usr/lib/ssl/misc/CA.pl -newca
```

这个脚本会当前目录中创建一个demoCA子目录，这个目录包含了一个根密钥对和有关证书与证书撤销列表的存储。

你希望将这个公共密钥导入到所有雇员的Java密钥库中，但是它的格式是隐私增强型邮件(PEM)格式，而不是密钥库更容易接受的DER格式。将文件demoCA/cacert.pem复制成文件acmeroot.pem，然后在文本编辑器中打开这个文件。移除下面这行之前的所有内容：

-----BEGIN CERTIFICATE-----

以及下面这行之后的所有内容：

-----END CERTIFICATE-----

现在可以按照通常的方式将acmeroot.pem导入到每个密钥库中了：

```
keytool -importcert -keystore cindy.certs -alias alice -file acmeroot.pem
```

这看起来有点不可思议，keytool竟然不能自己去执行这种编辑操作。

要对Alice的公共密钥签名，需要生成一个证书请求，它包含这个PEM格式的证书：

```
keytool -certreq -keystore alice.store -alias alice -file alice.pem
```

要签名这个证书，需要运行：

```
openssl ca -in alice.pem -out alice_signedby_acmeroot.pem
```

与前面一样，在alice\_signedby\_acmeroot.pem中切除BEGIN CERTIFICATE/END CERTIFICATE标记之外的所有内容。然后，将其导入到密钥库中：

```
keytool -importcert -keystore cindy.certs -alias alice -file alice_signedby_acmeroot.pem
```

你可以使用相同的步骤，使一个证书得到诸如VeriSign这样的公共证书权威机构的签名。

## 9.6 代码签名

认证技术最重要的一个应用是对可执行程序进行签名。如果从网上下载一个程序，自然会关心该程序可能带来的危害，例如，该程序可能已经感染了病毒。如果知道代码从何而来，并且它从离开源头后就没有被篡改过，那么放心程度会比不清楚这些信息时要高得多。事实上，如果该程序是用Java语言编写的，那么就可以利用这些信息来理性地决定应该让该程序拥有什么样的优先权。或许只想让它像普通applet一样在沙盒里运行，或者也可能想为它授予一组不同的权限和限制条件。例如，你下载了一个文字处理器，你可能想授予它访问打印机和某个子目录中的文件的权限，但是不想授予它建立网络连接的权限，这样，该程序就无法在你不知道的情况下将你的文件发送给第三方。

现在你已经知道了应该如何实现这个复杂的方案：

- 1) 使用认证来校验代码的来源。
- 2) 使用安全策略来运行代码，根据程序来源，实施你想赋予该程序的权限。

### 9.6.1 JAR文件签名

在本节中，我们将要介绍如何给applet和用Java插件从网络启动的应用程序进行签名。下面是两种情况：

- 1) 在企业内联网上传递。
- 2) 在公众因特网上传递。

在第一种情况下，系统管理员在本地机器上安装证书和策略文件。每当Java插件工具加载经过签名的代码时，它就会查询密钥库的签名和策略文件中的权限。安装证书和策略非常简单，每个桌面只需要安装一次。最终用户可以在沙盒外面运行经过签名的企业代码。每当创建新程序或者更新已有代码时，该程序必须进行签名，然后部署到Web服务器上。但是，程序运行时，

无需对桌面进行任何操作。我们认为这是一个合理的情况，是在桌面上部署企业应用的富有吸引力的可选方案。

在第二种情况下，软件供应商获取由证书发放权威（如VeriSign公司）签名的证书。当最终用户访问一个包含已签名applet的Web站点时，就会弹出一个对话框，显示软件供应商的相关信息，并给最终用户两个选择，一个是给予applet全部权限，另一个是继续在沙盒中运行。在9.6.2“软件开发者证书”一节中我们将要详细介绍这种不太令人引人注意的情况。

在本节的剩余部分，我们将要介绍如何建立策略文件，来为已知来源的代码赋予特定的权限。创建和部署这些策略文件不是普通最终用户要做的，然而，系统管理员在准备部署企业内联网程序时需要做这些工作。

假设ACME软件公司想让它的用户运行某些需要具备本地文件访问权限的程序，并且想要通过浏览器来部署这些程序，就像applet或者Web启动应用。由于这些程序不能在沙盒里运行，因此ACME软件公司需要在员工机器上安装策略文件。

正如在本章前面部分看到的那样，ACME可以根据applet的代码基来确定它们的身份，但是那将意味着每当applet代码移动到不同的Web服务器时，ACME都需要更新策略文件。为此，ACME决定对含有程序代码的JAR文件进行签名。

首先，ACME生成根证书：

```
keytool -genkeypair -keystore acmesoft.certs -alias acmeroot
```

当然，包含根密钥的密钥库必须存放在一个安全的地方。因此，我们为公共证书建立第二个密钥库Client.certs，并将公共的acmeroot证书添加进去。

```
keytool -exportcert -keystore acmesoft.certs -alias acmeroot -file acmeroot.cer  
keytool -importcert -keystore client.certs -alias acmeroot -file acmeroot.cer
```

为了创建一个经过签名的JAR文件，首先将各个类文件添加到JAR文件中，例如：

```
javac FileReadApplet.java  
jar cvf FileReadApplet.jar *.class
```

然后ACME中某个信任的人运行jarsigner工具，指定JAR文件和私有密钥的别名：

```
jarsigner -keystore acmesoft.certs FileReadApplet.jar acmeroot
```

被签名的applet现在就已经准备好在Web服务器中部署了。

接着，让我们转而配置客户机。必须将一个策略文件发布到每一台客户机上。

为了引用密钥库，策略文件将以下面这行开头：

```
keystore "keystoreURL", "keystoreType";
```

其中，URL可以是绝对的或相对的。如果密钥库是由keytool工具生成的，则它的类型是JKS。例如：

```
keystore "client.certs", "JKS";
```

grant子句可以有signedBy “alias” 后缀，例如：

```
grant signedBy "acmeroot"  
{  
    ...  
};
```

所有可以用与别名相关的公共密钥进行校验的签名代码现在都已经在grant语句中被授予了权限。

可以使用程序清单9-16中的applet来进行上面的代码签名过程。该applet试图读取一个本地文件，默认的安全策略只允许applet读取它的代码基及其子目录中的文件。用appletviewer来运行该applet，然后检验是否只能读取代码基目录中的文件，而不能读取其他目录下的文件。

现在，创建一个包含如下内容的策略文件applet.policy：

```
keystore "client.certs", "JKS";
grant signedBy "acmeroot"
{
    permission java.lang.RuntimePermission "usePolicy";
    permission java.io.FilePermission "/etc/*", "read";
};
```

usePolicy权限覆盖了用于被签名的applet默认的“全部拥有或全部没有”权限。这里，我们声明任何由acmeroot签名的applet都被允许读取/etc目录中的文件。（Windows用户：可以替换为其他诸如C:\Windows这样的目录。）

最后，告诉applet浏览器使用该策略文件：

```
appletviewer -J-Djava.security.policy=applet.policy FileReadApplet.html
```

现在该applet可以读取/etc目录中的所有文件了，这表示签名机制发挥了作用。

作为最后一项测试，你可以在浏览器中运行applet（参见图9-16）。这需要将权限文件和密钥库复制到Java部署目录中。如果你运行的是UNIX或Linux，这个目录就是你的主目录下的.java/deployment子目录下。在Windows Vista中，这个目录是C:\Users\yourLoginName\AppData\Sun\Java\Deployment目录。在下面的内容中，我们将用这个目录deploydir来引用这个目录。

将applet.policy和client.certs复制到deploydir/security目录中。在这个目录中，将applet.policy重命名为java.policy。（仔细检查你是否覆盖了已有的java.policy文件。如果已有该文件，则将applet.policy的内容添加到其中。）

**!** 提示：更多有关配置客户端Java安全的细节，可以参阅<http://java.sun.com/javase/6/docs/technotes/guides/deployment/deployment-guide/overview.html>部署指南中之“部署配置文件和属性”以及“Java控制面板”部分。

重新启动浏览器，并加载FileReadApplet.html，你应该不会看到提示你接受某类证书的信息。检查你是否能够加载/etc目录以及applet被加载的目录中的所有文件，而其他目录中的文件都不能加载。

测试完毕后，记着清理deploydir/security目录，将java.policy和client.certs文件移除。重启浏览器，清除之后，如果再次加载该applet，你将无法再从本地文件系统中读取文件，而且，你会看到关于证书的提示。我们将在下一节讨论安全证书。

### 9.6.2 软件开发者证书

到现在为止，我们所讨论的情况下，程序都是在企业内联网上传输的，而且是由系统管理

员来配置一个安全策略，以控制程序的权限。但是，这种策略只适合于已知来源的程序。

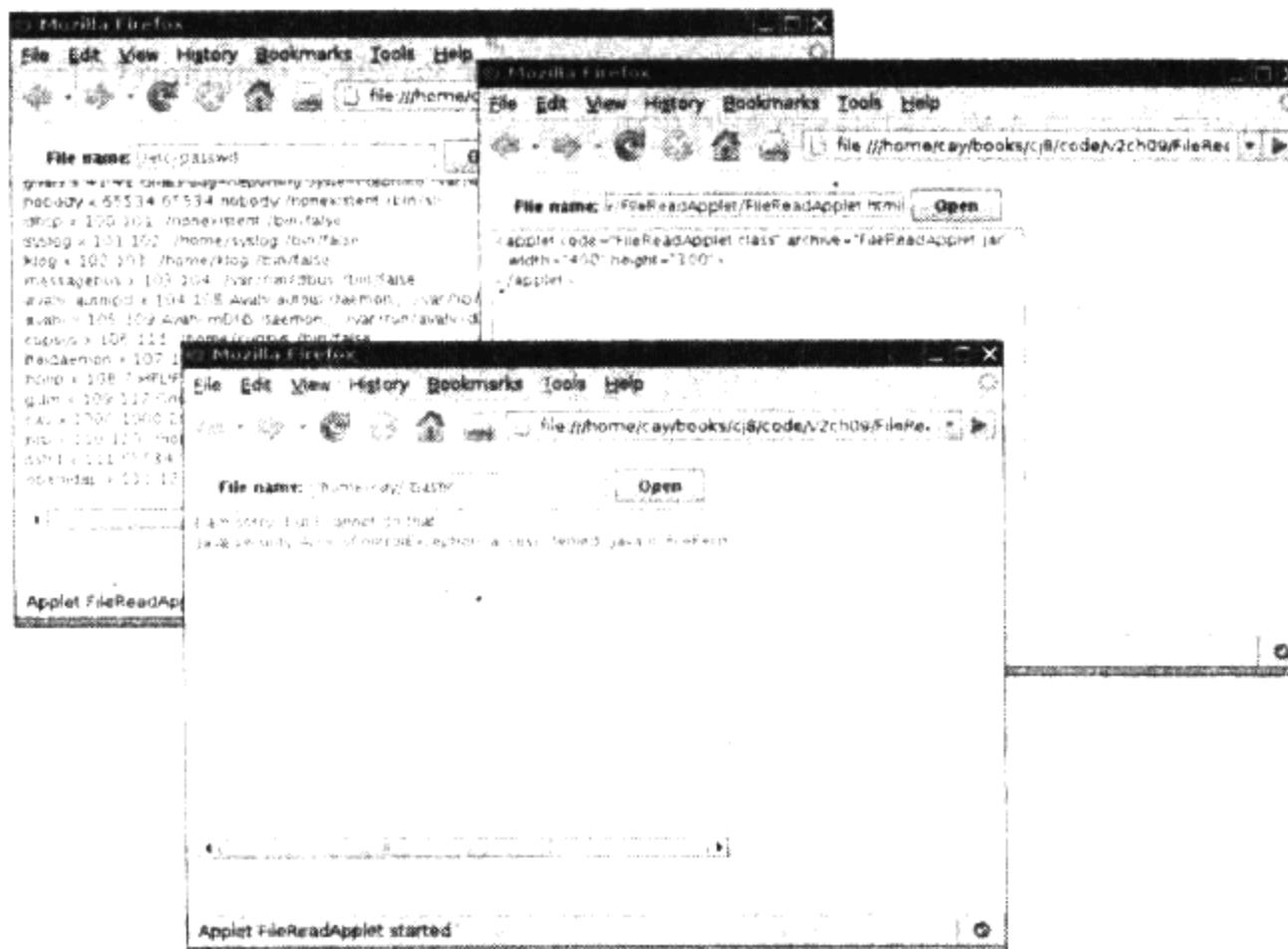


图9-16 经过签名的applet可以读取本地文件

### 程序清单9-16 FileReadApplet.java

```

1. import java.awt.*;
2. import java.awt.event.*;
3. import java.io.*;
4. import java.util.*;
5. import javax.swing.*;

6.
7. /**
8. * This applet can run "outside the sandbox" and read local files when it is given the right
9. * permissions.
10. * @version 1.11 2007-10-06
11. * @author Cay Horstmann
12. */
13. public class FileReadApplet extends JApplet
14. {
15.     public void init()
16.     {
17.         EventQueue.invokeLater(new Runnable()
18.         {
19.             public void run()
20.             {
21.                 fileNameField = new JTextField(20);
22.                 JPanel panel = new JPanel();
23.                 panel.add(new JLabel("File name:"));
24.                 panel.add(fileNameField);
25.                 JButton openButton = new JButton("Open");
26.                 panel.add(openButton);

```

```

27.     ActionListener listener = new ActionListener()
28.     {
29.         public void actionPerformed(ActionEvent event)
30.         {
31.             loadFile(fileNameField.getText());
32.         }
33.     };
34.     fileNameField.addActionListener(listener);
35.     openButton.addActionListener(listener);
36.
37.     add(panel, "North");
38.
39.     fileText = new JTextArea();
40.     add(new JScrollPane(fileText), "Center");
41. }
42. });
43. }
44.
45. /**
46. * Loads the contents of a file into the text area.
47. * @param filename the file name
48. */
49. public void loadFile(String filename)
50. {
51.     try
52.     {
53.         fileText.setText("");
54.         Scanner in = new Scanner(new FileReader(filename));
55.         while (in.hasNextLine())
56.             fileText.append(in.nextLine() + "\n");
57.         in.close();
58.     }
59.     catch (IOException e)
60.     {
61.         fileText.append(e + "\n");
62.     }
63.     catch (SecurityException e)
64.     {
65.         fileText.append("I am sorry, but I cannot do that.\n");
66.         fileText.append(e + "\n");
67.     }
68. }
69. private JTextField fileNameField;
70. private JTextArea fileText;
71. }

```

假设当你在因特网上冲浪时，遇到了一个Web站点，倘若你通过弹出的对话框为它授予了需要的权限，它就会运行一个来自不明提供商的applet或者Web启动应用（参见图9-17）。这样的程序是用由证书权威机构发放的“软件开发者”证书进行签名的。弹出的对话框用于确定软件开发者和证书发放者的身份。现在你有两个选择：

- 用全部权限运行程序；
- 将程序限制在沙盒中运行。（对话框中的Cancel按钮是一种误导。如果点击这个按钮，applet不会被取消，而是运行在沙盒中。）

那么什么样的因素可能会影响你的决定呢？下面是已经了解的情况：

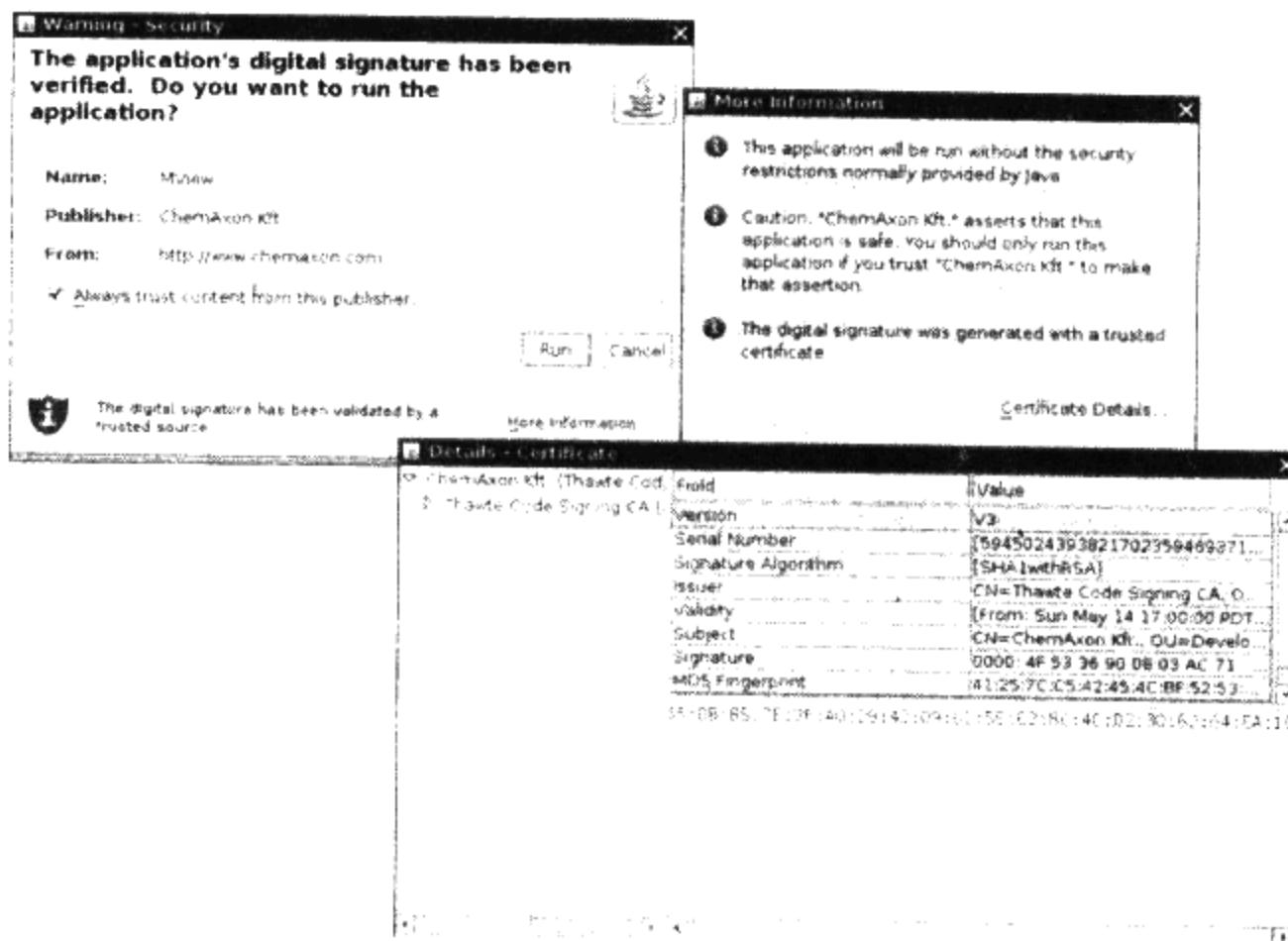


图9-17 启动一个签过名的applet

- 1) Thawte公司将一个证书卖给了软件开发人员。
- 2) 程序确实是用该证书签名的，并且在传输过程中没有被篡改过。
- 3) 该证书确实是由Thawte签名的，它是用本地cacerts文件中的公共密钥校验的。

这是否就意味着该代码可以安全运行了？如果你只知道供应商的名字，以及Thawte公司卖给他们一个软件开发者证书这个事实，那么你会信赖该供应商吗？如果想要担保ChemAxon Kft.不是个彻头彻尾的破解者，恐怕连Thawte公司自己也会陷入麻烦之中。然而，没有一个证书发放者会对软件供应商的诚信度和资格能力进行广泛的审查。

在不了解供应商的情况下，最终用户根本没能力做出明智的决定，以判断究竟是否应该让这个程序拥有本地应用程序的全部权限，以运行在沙盒之外。如果供应商是个众所周知的公司，那么用户至少可以考虑一下该公司过去的绩效记录。

**注意：**某些用户有可能会用非常弱的证书来对代码进行签名，这个网址：<http://www.dallaway.com/acad/webstart>就是一个值得冷静思考的例子。有些开发人员甚至会指示用户将不信任的证书添加到他们的证书库中——例如这个网址：[http://www.agsrhichome.bnl.gov/Controls/doc/javaws/javaws\\_howto.html](http://www.agsrhichome.bnl.gov/Controls/doc/javaws/javaws_howto.html)。从安全角度来讲，这非常的糟糕。

我们不愿意出现这样的情况：程序要求“给我全部权限，否则我决不运行”。缺乏经验的用户对于那些可能会使他们陷入危险的访问权限，在进行授权时常常会被吓住。

如果每个程序都对它所需要的权力进行解释，然后寻求获得这些权力的特定权限，这是否对我们有帮助呢？遗憾的是，如你所见，这样需要更高的技术要求。对于最终用户来说，要求

他去思考某个applet究竟是否真的应该拥有观察AWT事件队列的权力，这似乎不太合理。

对于用软件开发者证书，我们没有太大的热情。如果在公共互联网上的applet和网络启动的应用程序更努力地保持在它们各自的沙盒中运行，而这些沙盒又能够得到改进，那情况就会好得多。在第1卷第10章中讨论过的Web Start API就是向正确的方向迈出的一步。

## 9.7 加密

到现在为止，我们已经介绍了一种在Java安全API中实现的重要密码技术，即通过数字签名的认证。安全性的第二个重要方面是加密。当信息通过认证之后，该信息本身是直白可见的。数字签名只不过负责检验信息有没有被篡改过。相比之下，信息被加密后，是不可见的，只能用匹配的密钥进行解密。

认证对于代码签名已足够了——没必要将代码隐藏起来。但是，当applet或者应用程序传输机密信息时，比如信用卡号码和其他个人数据等，就有必要进行加密了。

由于专利和出口控制的原因，许多公司直到最近（包括Sun公司在内）都不提供高强度的加密技术。幸运的是，现在对加密技术的出口控制已经不是那么严格了，某些重要算法的专利也将到期。从Java SE1.4开始，出色的加密支持已经成为标准类库的一部分。

### 9.7.1 对称密码

Java密码扩展包含了一个Cipher类，该类是所有加密算法的超类。通过调用下面的getInstance方法可以获得一个密码对象：

```
Cipher cipher = Cipher.getInstance(algorithmName);
```

或者调用下面这个方法：

```
Cipher cipher = Cipher.getInstance(algorithmName, providerName);
```

JDK中是由名为“SunJCE”的提供商提供密码，如果没有指定其他提供商，则会默认该提供商。如果要使用特定的算法，而对该算法Sun公司没有提供支持，那么也可以指定其他的提供商。

算法名称是一个字符串，比如“AES”或者“DES/CBC/PKCS5Padding”。

DES，即数据加密标准，是一个密钥长度为56位的古老的分组密码。DES加密算法在现在看来已经是过时了，因为可以用穷举法将它破译（参见该网页中的例子[http://www.eff.org/Privacy/Crypto/Crypto\\_mis/DESCracker/](http://www.eff.org/Privacy/Crypto/Crypto_mis/DESCracker/)）。更好的选择是采用它的后续版本，即高级加密标准(AES)，更多详细信息，请访问网址<http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf>。下面我们以AES为例。

一旦获得了一个密码对象，就可以通过设置模式和密钥来对它初始化。

```
int mode = ...;
Key key = ...;
cipher.init(mode, key);
```

模式有以下几种：

```
Cipher.ENCRYPT_MODE
Cipher.DECRYPT_MODE
Cipher.WRAP_MODE
```

Cipher.UNWRAP\_MODE

wrap和unwrap模式用一个密钥对另一个密钥进行加密，具体例子请参见下一节。现在可以反复调用update方法来对数据块进行加密。

```
int blockSize = cipher.getBlockSize();
byte[] inBytes = new byte[blockSize];
... // read inBytes
int outputSize = cipher.getOutputSize(blockSize);
byte[] outBytes = new byte[outputSize];
int outLength = cipher.update(inBytes, 0, outputSize, outBytes);
... // write outBytes
```

完成上述操作后，还必须调用一次doFinal方法。如果有最后一个输入数据块（其字节数小于blockSize），那么就要调用：

```
outBytes = cipher.doFinal(inBytes, 0, inLength);
```

如果所有的输入数据都已经加密，则用下面的方法调用来代替：

```
outBytes = cipher.doFinal();
```

对doFinal的调用是必要的，以便对最后的块进行“填充”。就拿DES密码来说，它的数据块的大小是8个字节。假设输入数据的最后一个数据块少于8个字节，当然我们可以将其余的字节全部用0填充，从而得到一个8字节的最终数据块，然后对它进行加密。但是，当对数据块进行解密时，数据块的结尾会附加若干个0字节，因此它与原始输入文件之间会略有不同。这肯定是个问题，我们需要一个填充方案来避免这个问题。常用的填充方案是RSA Security公司在公共密钥密码标准#5中（Public Key Cryptography Standard，简称PKCS）描述的方案（该方案的网址为`ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs5v2-0.pdf`）。在该方案中，最后一个数据块不是全部用填充值0进行填充，而是用等于填充字节数量的值作为填充值进行填充。换句话说，如果L是最后一个（不完整的）数据块，那么它将按如下方式进行填充：

|                     |                  |
|---------------------|------------------|
| L 01                | if length(L) = 7 |
| L 02 02             | if length(L) = 6 |
| L 03 03 03          | if length(L) = 5 |
| ...                 |                  |
| L 07 07 07 07 07 07 | if length(L) = 1 |

最后，如果输入的数据长度确实能被8整除，那么就会将下面这个数据块：

```
08 08 08 08 08 08 08 08
```

附加到数据块后，并进行加密。在解密时，明文的最后一个字节就是要丢弃的填充字符数。

### 9.7.2 密钥生成

为了加密，我们需要生成密钥。每个密码都有不同的用于密钥的格式，我们需要确保密钥的生成是随机的。这需要遵循下面的步骤：

- 1) 为加密算法获取KeyGenerator。
- 2) 用随机源来初始化密钥发生器。如果密码块的长度是可变的，还需要指定期望的密码块长度。
- 3) 调用generateKey方法。

例如，下面是如何生成AES密钥的方法：

```
KeyGenerator keygen = KeyGenerator.getInstance("AES");
SecureRandom random = new SecureRandom(); // see below
keygen.init(random);
Key key = keygen.generateKey();
```

或者，可以从一组固定的原始数据（也许是由口令或者随机击键产生的）来生成一个密钥，这时可以使用如下的SecretKeyFactory：

```
SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("AES");
byte[] keyData = . . .; // 16 bytes for AES
SecretKeySpec keySpec = new SecretKeySpec(keyData, "AES");
Key key = keyFactory.generateSecret(keySpec);
```

如果要生成密钥，必须使用“真正的随机”数。例如，在Random类中的常规的随机数发生器，是根据当前的日期和时间来产生随机数的，因此它不够随机。例如，假设计算机时钟可以精确到1/10秒，那么，每天最多存在864 000个种子。如果攻击者知道发布密钥的日期（通常可以由截止日期推算出来），那么就可以很容易地生成那一天所有可能的种子。

SecureRandom类产生的随机数，远比由Random类产生的那些数字安全得多。你仍然需要提供一个种子，以便在一个随机点上开始生成数字序列。要这样做，最好的方法是从一个诸如白噪声发生器之类的硬件设备那里获取输入。另一个合理的随机输入源是请用户在键盘上进行随心所欲的盲打，但是每次敲击键盘只为随机种子提供1位或者2位。一旦你在字节数组中收集到这种随机位后，就可以将它传递给setSeed方法。

```
SecureRandom secrand = new SecureRandom();
byte[] b = new byte[20];
// fill with truly random bits
secrand.setSeed(b);
```

如果没有为随机数发生器提供种子，那么它将通过启动线程，使它们睡眠，然后测量它们被唤醒的准确时间，来计算自己的20个字节的种子。

 注意：这个算法仍然未被认为是安全的。而且，在过去，依靠对诸如硬盘访问时间之类的其他的计算机组件进行计时的算法，后来也被证明并不是完全随机的。

本节结尾处的示例程序将应用AES密码（参见程序清单9-17）。如果要使用该程序，首先生成一个密钥，运行如下命令行：

```
java AESTest -genkey secret.key
```

密钥就被保存在secret.key文件中了。

现在可以用如下命令进行加密：

```
java AESTest -encrypt plaintextFile encryptedFile secret.key
```

用如下命令进行解密：

```
java AESTest -decrypt encryptedFile decryptedFile secret.key
```

该程序非常直观。使用-genkey选项将产生一个新的密钥，并且将其序列化到给定的文件中。该操作需要花费较长的时间，因为密钥随机生成器的初始化非常耗费时间。-encrypt和-decrypt选项都调用相同的crypt方法，而crypt方法则调用密码的update和doFinal方法。请

注意update方法和doFinal方法是怎样被调用的，只要输入数据块具有全长度（长度能够被8整除），就要调用update方法，而如果输入数据块不具有全长度（长度不能被8整除，此时需要填充），或者没有更多额外的数据（以便生成一个填充字节），那么就要调用doFinal方法。

### 程序清单9-17 AESTest.java

```
1. import java.io.*;
2. import java.security.*;
3. import javax.crypto.*;
4.
5. /**
6. * This program tests the AES cipher. Usage:<br>
7. * java AESTest -genkey keyfile<br>
8. * java AESTest -encrypt plaintext encrypted keyfile<br>
9. * java AESTest -decrypt encrypted decrypted keyfile<br>
10. * @author Cay Horstmann
11. * @version 1.0 2004-09-14
12. */
13. public class AESTest
14. {
15.     public static void main(String[] args)
16.     {
17.         try
18.         {
19.             if (args[0].equals("-genkey"))
20.             {
21.                 KeyGenerator keygen = KeyGenerator.getInstance("AES");
22.                 SecureRandom random = new SecureRandom();
23.                 keygen.init(random);
24.                 SecretKey key = keygen.generateKey();
25.                 ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(args[1]));
26.                 out.writeObject(key);
27.                 out.close();
28.             }
29.             else
30.             {
31.                 int mode;
32.                 if (args[0].equals("-encrypt")) mode = Cipher.ENCRYPT_MODE;
33.                 else mode = Cipher.DECRYPT_MODE;
34.
35.                 ObjectInputStream keyIn = new ObjectInputStream(new FileInputStream(args[3]));
36.                 Key key = (Key) keyIn.readObject();
37.                 keyIn.close();
38.
39.                 InputStream in = new FileInputStream(args[1]);
40.                 OutputStream out = new FileOutputStream(args[2]);
41.                 Cipher cipher = Cipher.getInstance("AES");
42.                 cipher.init(mode, key);
43.
44.                 crypt(in, out, cipher);
45.                 in.close();
46.                 out.close();
47.             }
48.         }
49.         catch (IOException e)
```

```
50     {
51         e.printStackTrace();
52     }
53     catch (GeneralSecurityException e)
54     {
55         e.printStackTrace();
56     }
57     catch (ClassNotFoundException e)
58     {
59         e.printStackTrace();
60     }
61 }
62
63 /**
64 * Uses a cipher to transform the bytes in an input stream and sends the transformed bytes
65 * to an output stream.
66 * @param in the input stream
67 * @param out the output stream
68 * @param cipher the cipher that transforms the bytes
69 */
70 public static void crypt(InputStream in, OutputStream out, Cipher cipher)
71     throws IOException, GeneralSecurityException
72 {
73     int blockSize = cipher.getBlockSize();
74     int outputSize = cipher.getOutputSize(blockSize);
75     byte[] inBytes = new byte[blockSize];
76     byte[] outBytes = new byte[outputSize];
77
78     int inLength = 0;
79     boolean more = true;
80     while (more)
81     {
82         inLength = in.read(inBytes);
83         if (inLength == blockSize)
84         {
85             int outLength = cipher.update(inBytes, 0, blockSize, outBytes);
86             out.write(outBytes, 0, outLength);
87         }
88         else more = false;
89     }
90     if (inLength > 0) outBytes = cipher.doFinal(inBytes, 0, inLength);
91     else outBytes = cipher.doFinal();
92     out.write(outBytes);
93 }
94 }
```



#### javax.crypto.Cipher 1.4

- static Cipher getInstance(String algorithmName)
- static Cipher getInstance(String algorithmName, String providerName)

返回实现了指定加密算法的Cipher对象。如果未提供该算法，则抛出一个NoSuchAlgorithmException异常。

- int getBlockSize()

返回密码块的大小，如果该密码不是一个分组密码，则返回0。

- `int getOutputSize(int inputLength)`

如果下一个输入数据块拥有给定的字节数，则返回所需的输出缓冲的大小。本方法的运行要考虑到密码对象中的所有已缓冲的字节数量。

- `void init(int mode, Key key)`

对加密算法对象进行初始化。Mode是`ENCRYPT_MODE`, `DECRYPT_MODE`, `WRAP_MODE`, 或者`UNWRAP_MODE`之一。

- `byte[] update(byte[] in)`

- `byte[] update(byte[] in, int offset, int length)`

- `int update(byte[] in, int offset, int length, byte[] out)`

对输入数据块进行转换。前两个方法返回该输出，第三个方法返回放入out的字节数。

- `byte[] doFinal()`

- `byte[] doFinal(byte[] in)`

- `byte[] doFinal(byte[] in, int offset, int length)`

- `int doFinal(byte[] in, int offset, int length, byte[] out)`

转换输入的最后一个数据块，并刷新该加密算法对象的缓冲。前三个方法返回输出，第四个方法返回放入out的字节数。



### `javax.crypto.KeyGenerator 1.4`

- `static KeyGenerator getInstance(String algorithmName)`

返回实现指定加密算法的`KeyGenerator`对象。如果未提供该加密算法，则抛出一个`NoSuchAlgorithmException`异常。

- `void init(SecureRandom random)`

- `void init(int keySize, SecureRandom random)`

对密钥生成器进行初始化。

- `SecretKey generateKey()`

生成一个新的密钥。



### `javax.crypto.SecretKeyFactory 1.4`

- `static SecretKeyFactory getInstance(String algorithmName)`

- `static SecretKeyFactory getInstance(String algorithmName, String providerName)`

返回指定加密算法的`SecretKeyFactory`对象。

- `SecretKey generateSecret(KeySpec spec)`

根据给定描述规范生成一个新的密钥。



### `javax.crypto.spec.SecretKeySpec 1.4`

- `SecretKeySpec(byte[] key, String algorithmName)`

创建一个密钥描述规范。

### 9.7.3 密码流

JCE库提供了一组使用非常方便的数据类，用于对流进行自动加密或解密。例如，下面是对文件数据进行加密的方法：

```
Cipher cipher = . . .;
cipher.init(Cipher.ENCRYPT_MODE, key);
CipherOutputStream out = new CipherOutputStream(new FileOutputStream(outputFileName), cipher);
byte[] bytes = new byte[BLOCKSIZE];
int inLength = getData(bytes); // get data from data source
while (inLength != -1)
{
    out.write(bytes, 0, inLength);
    inLength = getData(bytes); // get more data from data source
}
out.flush();
```

同样地，可以使用CipherInputStream，对文件的数据进行读取和解密：

```
Cipher cipher = . . .;
cipher.init(Cipher.DECRYPT_MODE, key);
CipherInputStream in = new CipherInputStream(new FileInputStream(inputFileName), cipher);
byte[] bytes = new byte[BLOCKSIZE];
int inLength = in.read(bytes);
while (inLength != -1)
{
    putData(bytes, inLength); // put data to destination
    inLength = in.read(bytes);
}
```

密码流类能够透明地调用update和doFinal方法，所以非常方便。

#### API javax.crypto.CipherInputStream 1.4

- `CipherInputStream(InputStream in, Cipher cipher)`

构建一个输入流，以读取in中的数据，并且使用指定的密码对数据进行解密和加密。

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

读取输入流中的数据，该数据是自动进行解密和加密的。

#### API javax.crypto.CipherOutputStream 1.4

- `CipherOutputStream(OutputStream out, Cipher cipher)`

构建一个输出流，以便将数据写入out，并且使用指定的密码对数据进行加密和解密。

- `void write(int ch)`
- `void write(byte[] b, int off, int len)`

将数据写入输出流，该数据是自动进行加密和解密的。

- `void flush()`

刷新密码缓冲，如果需要的话，执行填充操作。

#### 9.7.4 公共密钥密码

在前面小节中看到的AES密码是一种对称密码，加密和解密都使用相同的密钥。对称密码的致命缺点在于密钥的分发。如果Alice给Bob发送一个加密方法，那么Bob需要使用与Alice相同的密钥。如果Alice修改了密钥，那么她必须在给Bob发送信息的同时，还要通过安全信道发送新的密钥，但是也许她并不拥有到达Bob的安全信道，这就是为什么她必须首先对她发送给Bob的信息进行加密的原因。

公共密钥密码技术解决了这个问题。在公共密钥密码中，Bob拥有一个密钥对，包括一个公共密钥和一个相匹配的私有密钥。Bob可以在任何地方公布公共密钥，但是他必须严格保守他的私有密钥。Alice只需要使用公共密钥对她发送给Bob的信息进行加密即可。

实际上，加密过程并没有那么简单。所有已知的公共密钥算法的操作速度都比对称密钥算法（比如DES或AES等）慢得多，使用公共密钥算法对大量的信息进行加密是不切实际的。但是，如果像下面这样，将公共密钥密码与快速的对称密码结合起来，这个问题就可以得到解决：

- 1) Alice生成一个随机对称加密密钥，她用该密钥对明文进行加密。
- 2) Alice用Bob的公共密钥对称密钥进行加密。
- 3) Alice将加密后的对称密钥和加密后的明文同时发送给Bob。
- 4) Bob用他的私有密钥对称密钥解密。
- 5) Bob用解密后的对称密钥对信息解密。

除了Bob之外，其他人无法给对称密钥进行解密，因为只有Bob拥有解密的私有密钥。因此，昂贵的公共密钥加密技术只适用于给少量的关键数据加密。

最普通的公共密钥算法是Rivest, Shamir和Adleman发明的RSA算法。直到2000年10月，该算法一直受RSA Security公司授予的专利保护。该专利的转让许可证价格昂贵，通常要支付3%的专利权使用费，每年至少付款50 000美元。现在该加密算法已经公开，Java SE 5.0及以后的版本都支持RSA算法。

 **注意：**如果你使用的是旧版本的JDK，可以使用Legion of Bouncy Castle（网址 <http://www.bouncycastle.org>），它提供了一个支持RSA的密码提供商和其他许多SunJCE提供商没有提供的特性。该提供商已经由Sun公司签名，因此可以将它与JDK结合起来使用。

如果要使用RSA算法，需要一对公共/私有密钥。你可以按如下方法使用KeyPairGenerator来获得：

```
KeyPairGenerator pairgen = KeyPairGenerator.getInstance("RSA");
SecureRandom random = new SecureRandom();
pairgen.initialize(KEYSIZE, random);
KeyPair keyPair = pairgen.generateKeyPair();
Key publicKey = keyPair.getPublic();
Key privateKey = keyPair.getPrivate();
```

程序清单9-18中的程序有三个选项。**-genkey**选项用于产生一个密钥对，**-encrypt**选项用于生成AES密钥，并且用公共密钥对其进行包装。

```
Key key = . . .; // an AES key
Key publicKey = . . .; // a public RSA key
Cipher cipher = Cipher.getInstance("RSA");
```

```
cipher.init(Cipher.WRAP_MODE, publicKey);
byte[] wrappedKey = cipher.wrap(key);
```

然后它便生成一个包含下列内容的文件：

- 包装过的密钥的长度。
- 包装过的密钥字节。
- 用AES密钥加密的明文。

-decrypt选项用于对这样的文件进行解密。请试运行该程序，首先生成RSA密钥：

```
java RSATest -genkey public.key private.key
```

然后对一个文件进行加密：

```
java RSATest -encrypt plaintextFile encryptedFile public.key
```

最后，对文件进行解密，并且检验解密后的文件是否与明文相匹配：

```
java RSATest -decrypt encryptedFile decryptedFile private.key
```

### 程序清单9-18 RSATest.java

```
1 import java.io.*;
2 import java.security.*;
3 import javax.crypto.*;
4
5 /**
6 * This program tests the RSA cipher. Usage:<br>
7 * java RSATest -genkey public private<br>
8 * java RSATest -encrypt plaintext encrypted public<br>
9 * java RSATest -decrypt encrypted decrypted private<br>
10 * @author Cay Horstmann
11 * @version 1.0 2004-09-14
12 */
13 public class RSATest
14 {
15     public static void main(String[] args)
16     {
17         try
18         {
19             if (args[0].equals("-genkey"))
20             {
21                 KeyPairGenerator pairgen = KeyPairGenerator.getInstance("RSA");
22                 SecureRandom random = new SecureRandom();
23                 pairgen.initialize(KEYSIZE, random);
24                 KeyPair keyPair = pairgen.generateKeyPair();
25                 ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(args[1]));
26                 out.writeObject(keyPair.getPublic());
27                 out.close();
28                 out = new ObjectOutputStream(new FileOutputStream(args[2]));
29                 out.writeObject(keyPair.getPrivate());
30                 out.close();
31             }
32             else if (args[0].equals("-encrypt"))
33             {
34                 KeyGenerator keygen = KeyGenerator.getInstance("AES");
35                 SecureRandom random = new SecureRandom();
36                 keygen.init(random);
37                 SecretKey key = keygen.generateKey();
```

```
38.
39.        // wrap with RSA public key
40.        ObjectInputStream keyIn = new ObjectInputStream(new FileInputStream(args[3]));
41.        Key publicKey = (Key) keyIn.readObject();
42.        keyIn.close();
43.
44.        Cipher cipher = Cipher.getInstance("RSA");
45.        cipher.init(Cipher.WRAP_MODE, publicKey);
46.        byte[] wrappedKey = cipher.wrap(key);
47.        DataOutputStream out = new DataOutputStream(new FileOutputStream(args[2]));
48.        out.writeInt(wrappedKey.length);
49.        out.write(wrappedKey);
50.
51.        InputStream in = new FileInputStream(args[1]);
52.        cipher = Cipher.getInstance("AES");
53.        cipher.init(Cipher.ENCRYPT_MODE, key);
54.        crypt(in, out, cipher);
55.        in.close();
56.        out.close();
57.    }
58.    else
59.    {
60.        DataInputStream in = new DataInputStream(new FileInputStream(args[1]));
61.        int length = in.readInt();
62.        byte[] wrappedKey = new byte[length];
63.        in.read(wrappedKey, 0, length);
64.
65.        // unwrap with RSA private key
66.        ObjectInputStream keyIn = new ObjectInputStream(new FileInputStream(args[3]));
67.        Key privateKey = (Key) keyIn.readObject();
68.        keyIn.close();
69.
70.        Cipher cipher = Cipher.getInstance("RSA");
71.        cipher.init(Cipher.UNWRAP_MODE, privateKey);
72.        Key key = cipher.unwrap(wrappedKey, "AES", Cipher.SECRET_KEY);
73.
74.        OutputStream out = new FileOutputStream(args[2]);
75.        cipher = Cipher.getInstance("AES");
76.        cipher.init(Cipher.DECRYPT_MODE, key);
77.
78.        crypt(in, out, cipher);
79.        in.close();
80.        out.close();
81.    }
82.}
83. catch (IOException e)
84. {
85.     e.printStackTrace();
86. }
87. catch (GeneralSecurityException e)
88. {
89.     e.printStackTrace();
90. }
91. catch (ClassNotFoundException e)
92. {
93.     e.printStackTrace();
94. }
95. }
```

```
96
97.     /**
98.      * Uses a cipher to transform the bytes in an input stream and sends the transformed bytes
99.      * to an output stream.
100.     * @param in the input stream
101.     * @param out the output stream
102.     * @param cipher the cipher that transforms the bytes
103.     */
104.    public static void crypt(InputStream in, OutputStream out, Cipher cipher)
105.        throws IOException, GeneralSecurityException
106.    {
107.        int blockSize = cipher.getBlockSize();
108.        int outputSize = cipher.getOutputSize(blockSize);
109.        byte[] inBytes = new byte[blockSize];
110.        byte[] outBytes = new byte[outputSize];
111.
112.        int inLength = 0;
113.        ;
114.        boolean more = true;
115.        while (more)
116.        {
117.            inLength = in.read(inBytes);
118.            if (inLength == blockSize)
119.            {
120.                int outLength = cipher.update(inBytes, 0, blockSize, outBytes);
121.                out.write(outBytes, 0, outLength);
122.            }
123.            else more = false;
124.        }
125.        if (inLength > 0) outBytes = cipher.doFinal(inBytes, 0, inLength);
126.        else outBytes = cipher.doFinal();
127.        out.write(outBytes);
128.    }
129.
130.    private static final int KEYSIZE = 512;
131. }
```

你现在已经看到了Java安全模型是如何允许我们去控制代码的执行的，这是Java平台的一个独一无二且越来越重要的方面。你已经看到了Java类库提供的认证和加密服务。但是我们没有涉及许多高级和专有的话题，比如：

- 提供了对Kerberos协议进行支持的“通用安全服务”的GSS-API（原则上同样支持其他安全信息交换协议）。下面这个网址上有一份JDK的指南<http://java.sun.com/j2se/5.0/docs/guide/security/jgss/tutorials/index.html>。
- 对SASL的支持，SASL即简单认证和安全层，可以为LDAP和IMAP协议所使用。如果想在自己的应用程序中实现SASL，请浏览下面这个网址：<http://java.sun.com/j2se/5.0/docs/guide/security/sasl/sasl-refguide.html>。
- 对SSL的支持，SSL即安全套接层。在HTTP上使用SSL对应用程序的编程人员是透明的，只需要直接使用以https开头的URL即可。如果想要给你的应用程序添加SSL支持，请参阅下面网址中的JSSE（Java安全套接扩展）参考指南<http://java.sun.com/j2se/5.0/docs/guide/security/jsse/JSSERefGuide.html>。

现在我们已经结束了对Java安全的学习，接下来转向第10章分布式计算。

# 第10章 分布式对象

- ▲ 客户与服务器的角色
- ▲ 远程方法调用
- ▲ RMI编程模型

- ▲ 远程方法中的参数传递
- ▲ 远程对象激活
- ▲ Web Services与SOAP

每过一段时间，程序员社区就会开始考虑将“无所不在的对象”作为所有问题的解决之道。其思想是所有相互协作的对象就像一个和睦的家庭，彼此都知道对方在哪里。当在一台计算机上的某个对象需要调用在另一台计算机上的某个对象时，它就会发送一个包含这个请求的详细信息的网络消息。这个远程对象可能通过访问数据库也可能通过与其他对象通信来计算出响应。一旦该远程对象得到了客户端请求的东西，就将它送回客户端。在概念上讲，这个过程显得十分简单，但是你不能停留在表面，应该深入理解如何有效地利用分布式对象。

在本章，我们将聚焦Java分布式编程技术，特别是用于两个Java虚拟机（可以运行在不同的计算机上）之间通信的远程方法调用（RMI）协议。然后我们将简略地浏览一下用于产生对Web服务的远程调用的JAX-WS技术。

## 10.1 客户与服务器的角色

所有分布式编程技术的基本思想都很简单：客户计算机产生一个请求，然后将这个请求通过网络发送到服务器。服务器处理这个请求，并发送回一个针对该客户端的响应，供客户端进行分析。图10-1展示了这个过程。

一开始我们就必须要声明：这些请求和响应并不是在Web应用程序中看到的请求和响应。这里的客户端并非是Web浏览器，它可以是执行具有任意复杂度的业务规则的任何应用程序。客户端应用程序可以与人类用户之间进行交互，但也可以没有这种交互。如果有这种交互，它可以拥有一个命令行或Swing用户界面。用于请求和响应数据的协议允许传递任意对象，而传统的Web应用程序只限于对请求使用HTTP协议，对响应使用HTML。

我们真正想要的是这样一种机制，客户端程序员以常规的方式进行方法调用，而无需操

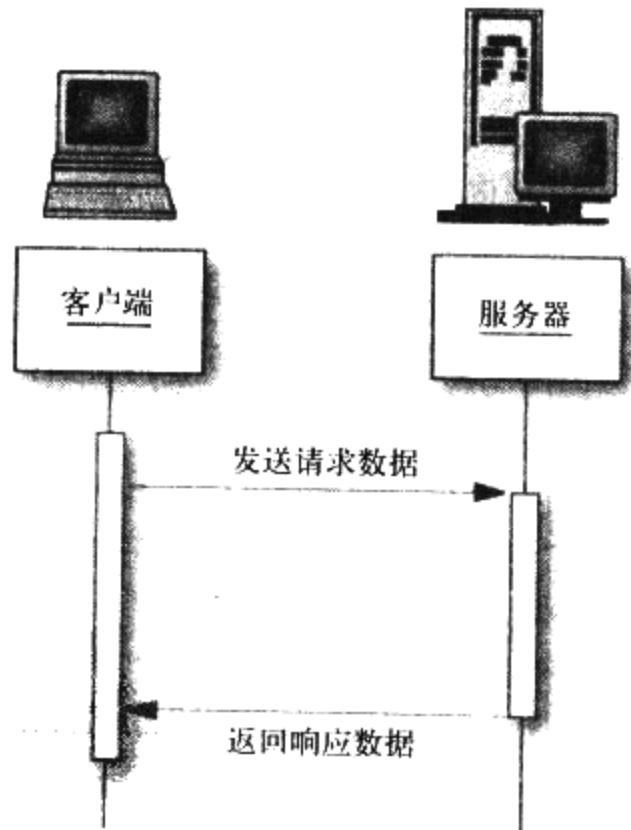


图10-1 在客户端与服务器之间传递对象

心数据在网络上传输或者解析响应之类的问题。解决的办法是，在客户端为远程对象安装一个代理（proxy）。代理是位于客户端虚拟机中的一个对象，它对于客户端程序来说，看起来就像是要访问的远程对象一样。客户调用此代理，进行常规的方法调用。而客户端代理负责与服务器进行联系。

同样的，实现服务的程序员也不希望因与客户端之间的通信被绊住。解决方法是在服务器端安装第二个代理对象。该服务器代理与客户端代理进行通信，并且它将以常规方式调用服务器对象上的方法（参见图10-2）。

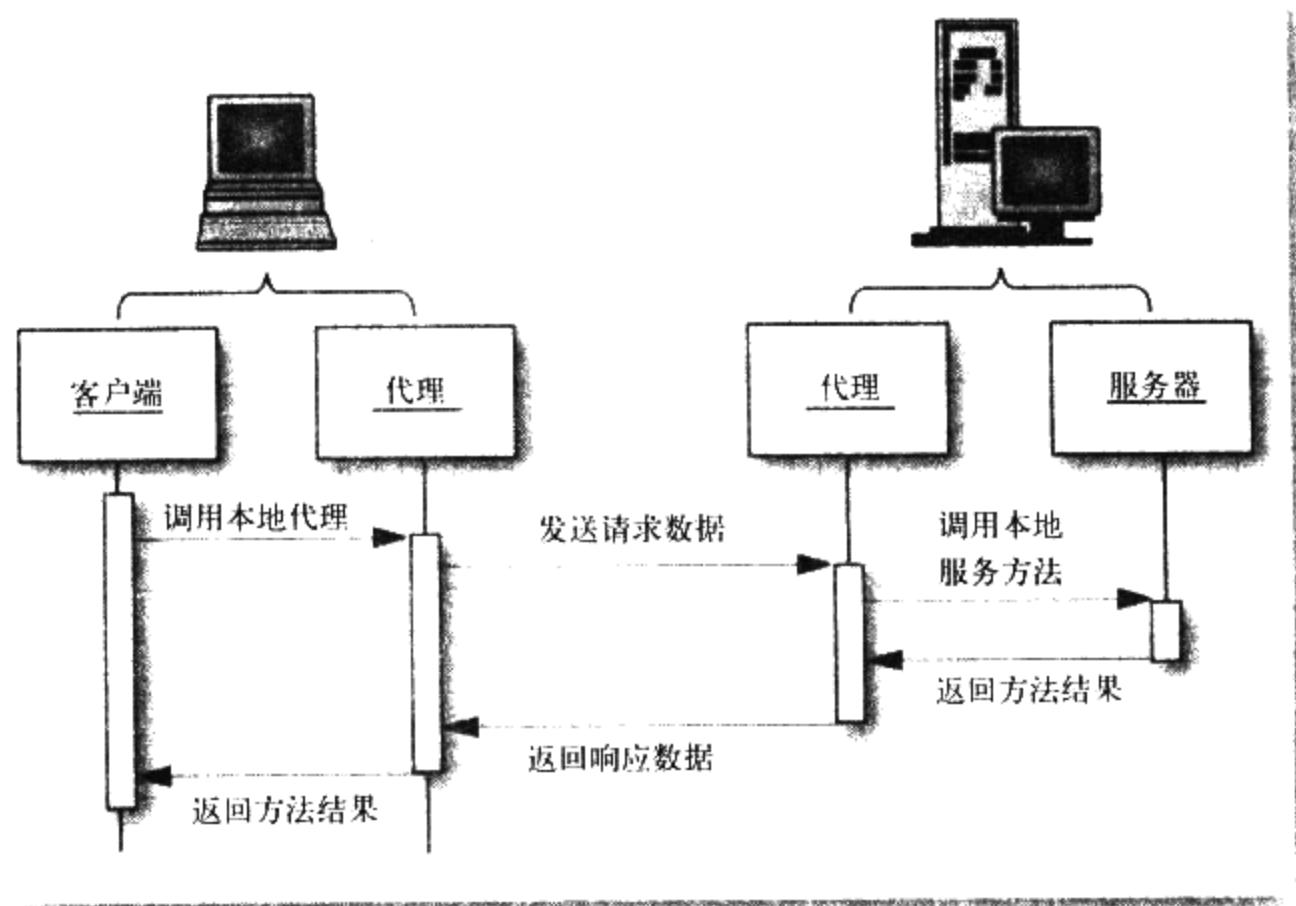


图10-2 使用代理的远程方法调用

代理之间是如何通信的呢？这要看以什么技术来实现它们。通常有三种选择：

- RMI，Java的远程方法调用技术，支持Java的分布式对象之间的方法调用。
- CORBA，通用对象请求代理架构，支持任何编程语言编写的对象之间的方法调用。CORBA使用Internet Inter-ORB协议（IIOP）支持对象间的通信。
- Web服务架构是一个协议集，有时统一描述为WS-\*。它也独立于编程语言的，不过它使用基于XML的通信格式。用于传输对象的格式则是简单对象访问协议（SOAP）。

如果互相通信的程序都是由Java实现的，那么，CORBA与WS-\*的通用性与复杂性统统都是不需要的。Sun开发了一个更简单的机制，称为远程方法调用（RMI），专门针对Java应用之间的通信。

即使你在自己的应用程序中不是RMI，学习它也是很值得的。通过一个直观的架构，你将会学习到用于分布式应用程序编程的精髓。而且，如果你使用了企业Java技术，那么对RMI有个基本了解就显得很有用了，因为它是企业Java bean（EJB）之间通信时所使用的协议。EJB

是服务器端构件，它可以用来构建运行在多个服务器上的复杂应用。要想有效地使用EJB，你就需要对远程调用相关的开销有清晰的理解。

与RMI不同，CORBA与SOAP都是完全独立于语言的。客户端与服务器程序可以由C、C++、Java或者其他语言编写。你只需要提供一个接口描述，以说明你的对象能够处理的方法的签名与数据类型。该接口描述是用一种特殊的语言编写的，对于CORBA来说是接口定义语言（IDL），对于Web服务而言则是Web服务描述语言（WSDL）。

多年来，很少有人相信CORBA就是未来的对象模型。坦白地说，CORBA的声誉是，对于复杂实现与互操作性问题，有时值得用，并且也只取得了一定程度的成功。我们在本书的第5版中讨论了Java与CORBA之间的互操作性，但是由于读者缺乏兴趣，我们之后删掉了这部分内容。在我们深入研究了一些CORBA的问题之后，我们对于CORBA的态度变得与法国总统Charles De Gaulle对于巴西的态度一样：它会有很好的未来，总会有的。

Web服务在首次出现时喧噪一时，因为它承诺会更加简单，而且建立在WWW和XML的精华之上。但是，随着时间的推移和许多委员会工作的推进，其协议栈已经变得不简单了，因为它被要求具备越来越多CORBA一直具有的特性。XML协议具有可阅读性的优点（其实也只是一点点），这有助于查错。另一方面，处理XML是很大的性能瓶颈。最近，WS-\*栈的光环已经褪去了许多，并且它也获得了相同的声誉，即对于复杂实现和互操作问题，有时只得用。

我们将用一个消费Web服务的实例结束本章，在此之前我们会浏览其底层协议，使你可以了解不同的编程语言之间的通信是如何实现的。

## 10.2 远程方法调用

分布式计算的关键是远程方法调用。在一台机器（称为客户端）上的某些代码希望调用在另一台机器（远程对象）上的某个对象的一个方法。要实现这一点，方法的参数必须以某种方式传递到另一台机器上，而服务器必须得到通知，去定位远程对象并执行要调用的方法，并且必须将返回值传递回去。

在详细浏览整个过程之前，我们需要指出，客户端/服务器这一术语用法只适用于单个的方法调用。调用远程方法的计算机对于这个调用来说是客户端，而处理这个调用的对象的宿主计算机对于这个调用来说是服务器。在某些情况下这两个角色完全有可能会颠倒过来，前一个调用的服务器在调用驻留在另一台计算机上的对象的某个远程方法时，它自己就变成了客户端。

### 存根与参数编组

当客户代码要在远程对象上调用一个远程方法时，实际上调用的是代理对象上的一个普通的方法，我们称此代理对象为存根（stub）。例如，

```
Warehouse centralWarehouse = get stub object;
double price = centralWarehouse.getPrice("Blackwell Toaster");
```

存根位于客户端机器上，而非服务器上。它知道如何通过网络与服务器联系。存根将远程方法所需的参数打包成一组字节。对参数编码的过程称作参数编组（parameter marshalling），参数编组的目的是将参数转换成适合在虚拟机之间进行传递的格式。在RMI协议中，对象是使

用序列化机制进行编码的，第1章描述了这种机制。在SOAP协议中，对象被编码为XML。

总的来说，客户端的存根方法构造了一个信息块，它由以下几部分组成：

- 被使用的远程对象的标识符；
- 被调用的方法的描述；
- 编组后的参数。

然后，存根将此信息发送给服务器。在服务器一端，接收对象执行以下动作：

- 定位要调用的远程对象；
- 调用所需的方法，并传递客户端提供的参数；
- 捕获返回值或该调用产生的异常；
- 将返回值编组，打包送回给客户端存根。

客户端存根对来自服务器端的返回值或异常进行反编组，就成为了调用存根的返回值。如果远程方法抛出了一个异常，那么存根就在客户端发起调用的处理空间中重新抛出该异常。图10-3展示了一次远程方法调用的信息流。

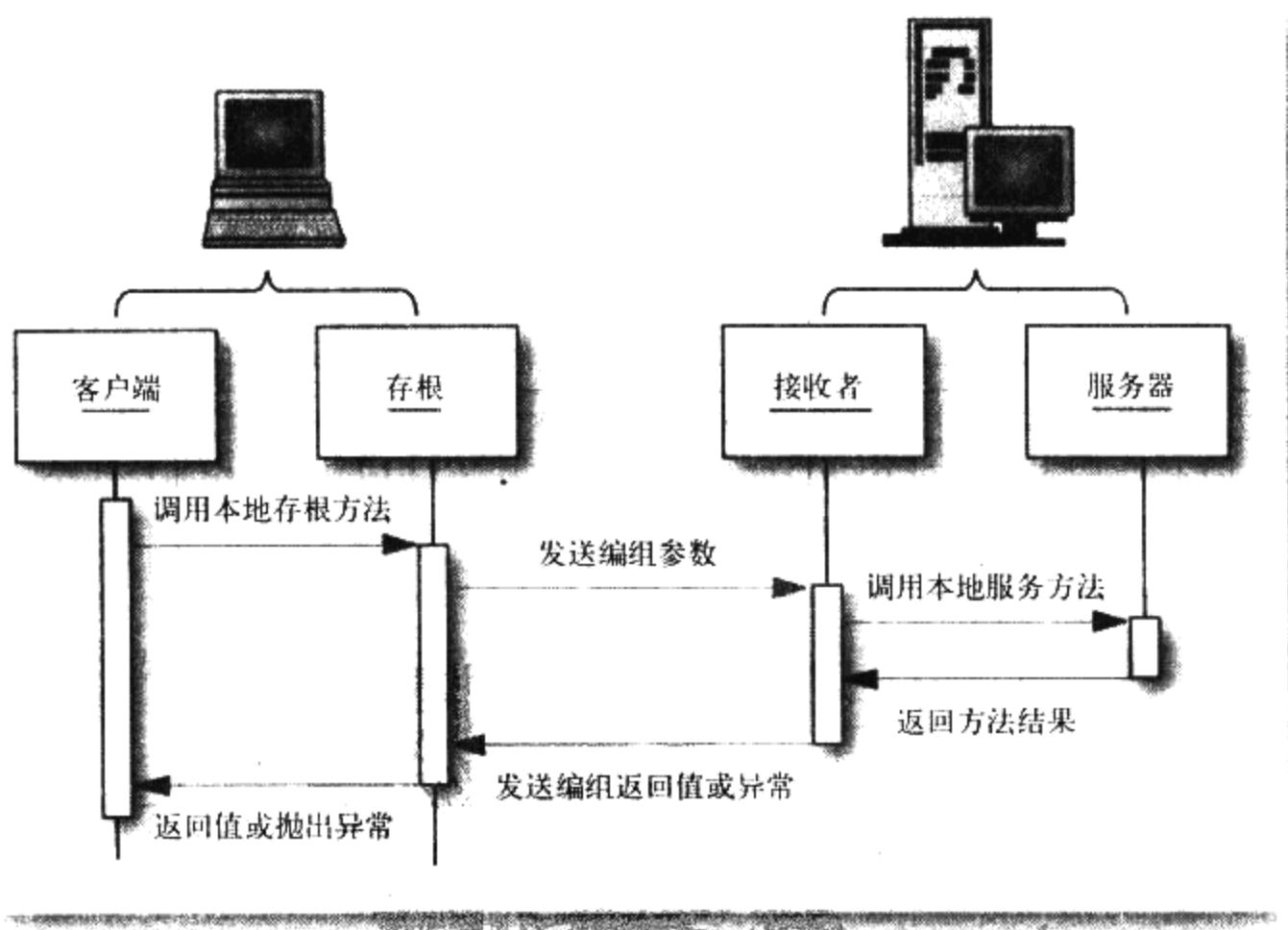


图10-3 参数编组

这个过程显然很复杂，不过好消息是，这一切都是完全自动的，而且在很大程度上，它对程序员是透明性。

实现远程对象和获取客户端存根的细节有赖于分布式对象采用的技术。在下面的小节中，我们将深入学习RMI。

## 10.3 配置远程方法调用

为了介绍RMI编程模型，我们从一个简单的实例入手。远程对象表示的是一个仓库，而客户端程序向仓库询问某个产品的价格。在下面的小节中，你将看到如何实现和启动服务器与客户端程序。

### 10.3.1 接口与实现

远程对象的能力是由在客户端和服务器之间共享的接口所表示的。例如，程序清单10-1中的接口描述了远程仓库对象所提供的服务。

程序清单10-1 Warehouse.java

```
1. import java.rmi.*;  
2.  
3. /**  
4.  * The remote interface for a simple warehouse.  
5.  * @version 1.0 2007-10-09  
6.  * @author Cay Horstmann  
7. */  
8. public interface Warehouse extends Remote  
9. {  
10.    double getPrice(String description) throws RemoteException;  
11.}
```

远程对象的接口必须扩展Remote接口，它位于java.rmi包中。接口中的所有方法还必须声明抛出RemoteException异常，这是因为远程方法调用与生俱来就缺乏本地调用的可靠性，远程调用总是存在失败的可能。例如，服务器或者网络连接可能暂时不可用，或者可能出现网络故障。客户端代码必须时刻准备好处理这些问题。基于这些原因，Java编程语言要求每一次远程方法调用都必须捕获RemoteException，并且说明当调用不成功时相应的处理操作。

接下来，在服务器端，必须提供这样的类，它真正实现了在远程接口中声明的工作。参见程序清单10-2。

程序清单10-2 WarehouseImpl.java

```
1. import java.rmi.*;  
2. import java.rmi.server.*;  
3. import java.util.*;  
4.  
5. /**  
6.  * This class is the implementation for the remote Warehouse interface.  
7.  * @version 1.0 2007-10-09  
8.  * @author Cay Horstmann  
9. */  
10. public class WarehouseImpl extends UnicastRemoteObject implements Warehouse  
11. {  
12.    public WarehouseImpl() throws RemoteException  
13.    {  
14.      prices = new HashMap<String, Double>();  
15.      prices.put("Blackwell Toaster", 24.95);  
16.      prices.put("ZapXpress Microwave Oven", 49.95);  
17.    }
```

```
17.     }
18.
19.    public double getPrice(String description) throws RemoteException
20.    {
21.        Double price = prices.get(description);
22.        return price == null ? 0 : price;
23.    }
24.
25.    private Map<String, Double> prices;
26. }
```

 注意：WarehouseImpl的构造器声明了会抛出RemoteException异常，因为其超类的构造器会抛出这个异常，这发生在无法连接到跟踪远程对象的网络服务时。

你可以看出这个类是远程方法调用的目标，因为它继承自UnicastRemoteObject，这个类的构造器使得它的对象可供远程访问。“最小阻抗路径”是从UnicastRemoteObject中导出的，本章中的所有服务实现类也都是如此。

有时候可能不希望服务器类继承UnicastRemoteObject，也许是因为实现类已经继承了其他的类。在这种情况下，读者需要亲自初始化远程对象，并将它们传给静态的exportObject方法。如果不继承UnicastRemoteObject，可以在远程对象的构造器中像下面这样调用exportObject方法。

```
UnicastRemoteObject.exportObject(this, 0);
```

第二个参数是0，表明任意合适的端口都可用来监听客户连接。

 注意：Unicast这个术语是指我们是通过产生对单一的IP地址和端口的调用来定位远程对象的这一事实。这是Java SE中惟一支持的机制。更复杂的分布式对象系统（诸如JINI）会考虑到对在多个不同的服务器上的远程对象的“Multicast”查找。

### 10.3.2 RMI注册表

要访问服务器上的一个远程对象时，客户端首先需要一个本地的存根对象。可是客户端如何对该存根发出请求呢？最普通的方法是调用另一个服务对象上的一个远程方法，以返回值的方式取得存根对象。然而，这就成了先有鸡还是先有蛋的问题。第一个远程对象总要通过某种方式进行定位。为此，JDK提供了自举注册服务（bootstrap registry service）。

服务器程序使用自举注册服务来注册至少一个远程对象。要注册一个远程对象，需要一个RMI URL和一个对实现对象的引用。

RMI的URL以rmi:开头，后接服务器以及一个可选的端口号，接着是远程对象的名字。例如：  
rmi://regserver.mycompany.com:99/central\_warehouse

默认情况下，主机名是localhost，端口为1099。服务器告诉注册表在给定位置将该对象关联或“绑定”到这个名字。

下面的代码将一个WarehouseImpl对象注册到了同一个服务器上的RMI注册表中：

```
WarehouseImpl centralWarehouse = new WarehouseImpl();
Context namingContext = new InitialContext();
namingContext.bind("rmi:central_warehouse", centralWarehouse);
```

程序清单10-3中的程序只是构造并注册了一个WarehouseImpl对象。

### 程序清单10-3 WarehouseServer.java

```

1 import java.rmi.*;
2 import javax.naming.*;
3
4 /**
5  * This server program instantiates a remote warehouse object, registers it with the naming
6  * service, and waits for clients to invoke methods.
7  * @version 1.12 2007-10-09
8  * @author Cay Horstmann
9  */
10
11 public class WarehouseServer
12 {
13     public static void main(String[] args) throws RemoteException, NamingException
14     {
15         System.out.println("Constructing server implementation...");
16         WarehouseImpl centralWarehouse = new WarehouseImpl();
17
18         System.out.println("Binding server implementation to registry...");
19         Context namingContext = new InitialContext();
20         namingContext.bind("rmi://regserver.mycompany.com/central_warehouse", centralWarehouse);
21
22         System.out.println("Waiting for invocations from clients...");
23     }
24 }
```

 **注意：**基于安全原因，一个应用只有当它与注册表运行在同一个服务器时，该应用才可以绑定、取消绑定，或重绑定注册对象的引用。

客户端可以通过下面的调用枚举所有注册过的RMI对象：

```
Enumeration<NameClassPair> e = namingContext.list("rmi://regserver.mycompany.com");
```

NameClassPair是一个助手类，它包含绑定对象的名字和该对象所属类的名字。例如，下面的代码可以显示所有注册对象的名字：

```
while (e.hasMoreElements())
    System.out.println(e.nextElement().getName());
```

客户端可以通过下面的方式，来指定服务器和远程对象的名字，以此获得访问远程对象所需的存根。

```
String url = "rmi://regserver.mycompany.com/central_warehouse";
Warehouse centralWarehouse = (Warehouse) namingContext.lookup(url);
```

 **注意：**在一个全局注册表中，想保持一个名字的唯一性非常困难，因此不应该将此技术作为定位服务器端对象的一般方法。相反，自举服务只应该用来注册非常少的远程对象。然后使用这些对象来定位其他的对象。

程序清单10-4展示了客户端如何获得远程仓库对象的存根，并调用远程的getPrice方法。图10-4展示了其控制流。客户端获得了一个Warehouse存根，并在其上调用了getPrice方法。在后台，存根与服务器联系，并在WarehouseImpl对象上调用getPrice方法。

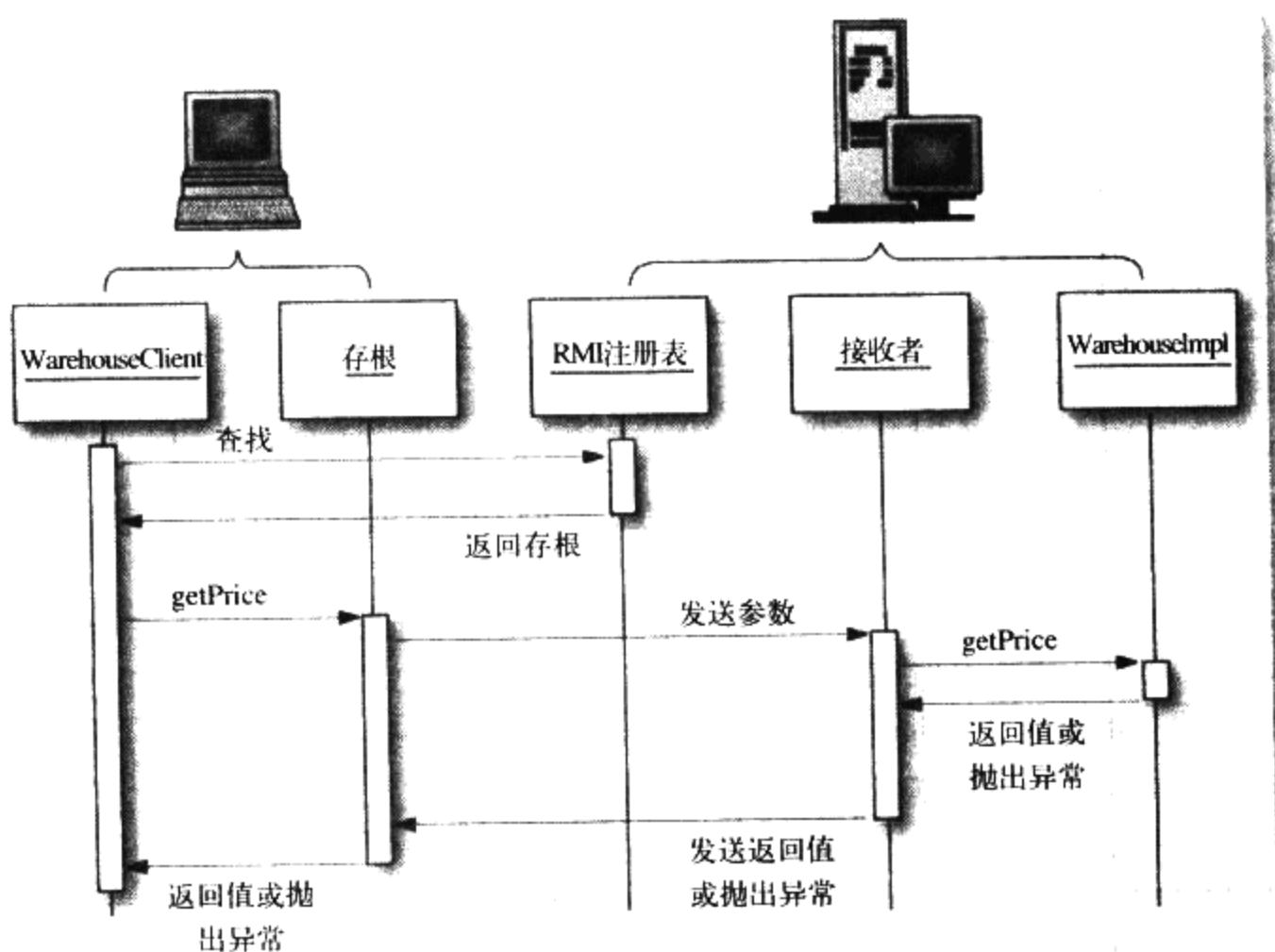


图10-4 调用远程的getDescription方法

## 程序清单10-4 WarehouseClient.java

```

1. import java.rmi.*;
2. import java.util.*;
3. import javax.naming.*;
4.
5. /**
6. * A client that invokes a remote method.
7. * @version 1.0 2007-10-09
8. * @author Cay Horstmann
9. */
10 public class WarehouseClient
11 {
12     public static void main(String[] args) throws NamingException, RemoteException
13     {
14         Context namingContext = new InitialContext();
15.
16         System.out.print("RMI registry bindings: ");
17         Enumeration<NameClassPair> e = namingContext.list("rmi://localhost/");
18         while (e.hasMoreElements())
19             System.out.println(e.nextElement().getName());
20.
21         String url = "rmi://localhost/central_warehouse";
22         Warehouse centralWarehouse = (Warehouse) namingContext.lookup(url);
23.
24         String descr = "Blackwell Toaster";
  
```

```

25     double price = centralWarehouse.getPrice(descr);
26     System.out.println(descr + ":" + price);
27   }
28 }
```

**API** **javax.naming.InitialContext 1.3**

- **InitialContext()**

构建一个命名上下文，用来访问RMI注册表。

**API** **javax.naming.Context 1.3**

- **static Object lookup(String name)**

返回给定名字的对象。如果该名字尚未绑定则抛出**NamingException**异常。

- **static void bind(String name, Object obj)**

将name与obj对象绑定。如果该对象已经绑定则抛出**NameAlreadyBoundException**异常。

- **static void unbind(String name)**

解除该名字的绑定。解除一个不存在的绑定是合法的。

- **static void rebind(String name, Object obj)**

将name与obj对象绑定。替换掉以前的绑定。

- **NamingEnumeration<NameClassPair> list(String name)**

返回一个枚举列表，其中列出了所有匹配的绑定对象。使用“rmi:”调用该方法可以列出所有RMI对象。

**API** **javax.naming.NameClassPair 1.3**

- **String getName()**

获取已命名对象的名字。

- **String getClassName()**

获取已命名对象所属的类名。

**API** **java.rmi.Naming 1.1**

- **static Remote lookup(String url)**

返回URL对应的远程对象。如果该名字尚未绑定，则抛出**NotBoundException**异常。

- **static void bind(String name, Remote obj)**

将name与远程对象obj绑定。如果该对象已经绑定则抛出**AlreadyBoundException**异常。

- **static void unbind(String name)**

解除对该名字的绑定。如果该名字没有绑定则抛出**NotBound**异常。

- **static void rebind(String name, Remote obj)**

将name与远程对象obj绑定。替换掉以前的绑定。

- **static String[] list(String url)**

参数url指定了某个注册表，返回注册表中的所有URL组成得字符串数组。此数组代表注册表中当前所有名字的一个快照。

### 10.3.3 部署程序

部署一个使用RMI的应用是比较棘手的。因为很多事情都有可能出错，而一旦出错，可以得到的错误信息又相当匮乏。我们发现，按照客户端和服务器将类进行分隔，从而在真实条件下对部署进行测试是很有价值的。

创建两个目录，分别存放用于启动服务器和客户端的类。

```
server/
  WarehouseServer.class
  Warehouse.class
  WarehouseImpl.class

client/
  WarehouseClient.class
  Warehouse.class
```

当部署RMI应用时，通常需要动态地将类交付为运行程序，其中一个例子就是RMI注册表。请记住注册表的一个实例要服务许多不同的RMI应用。RMI注册表需要访问注册的服务接口的类文件，但是，当注册表启动时，无法预测将来会产生所有注册请求。因此，RMI注册表会动态地加载之前从未遇到过的所有远程接口的类文件。

动态交付的类文件是通过标准的Web服务器发布的。在我们所举的情况下，服务器程序需要使Warehouse.class文件对于RMI注册表来说是可获得的，因此我们将这个文件放到了第三个称为download的目录中：

```
download/
  Warehouse.class
```

我们使用Web服务器来服务这个目录中的内容。

当应用被部署时，服务器、RMI注册表、Web服务器和客户端可以定位到四台不同的计算机上，参见图10-5。但是，出于测试的目的，我们将只使用一台计算机。

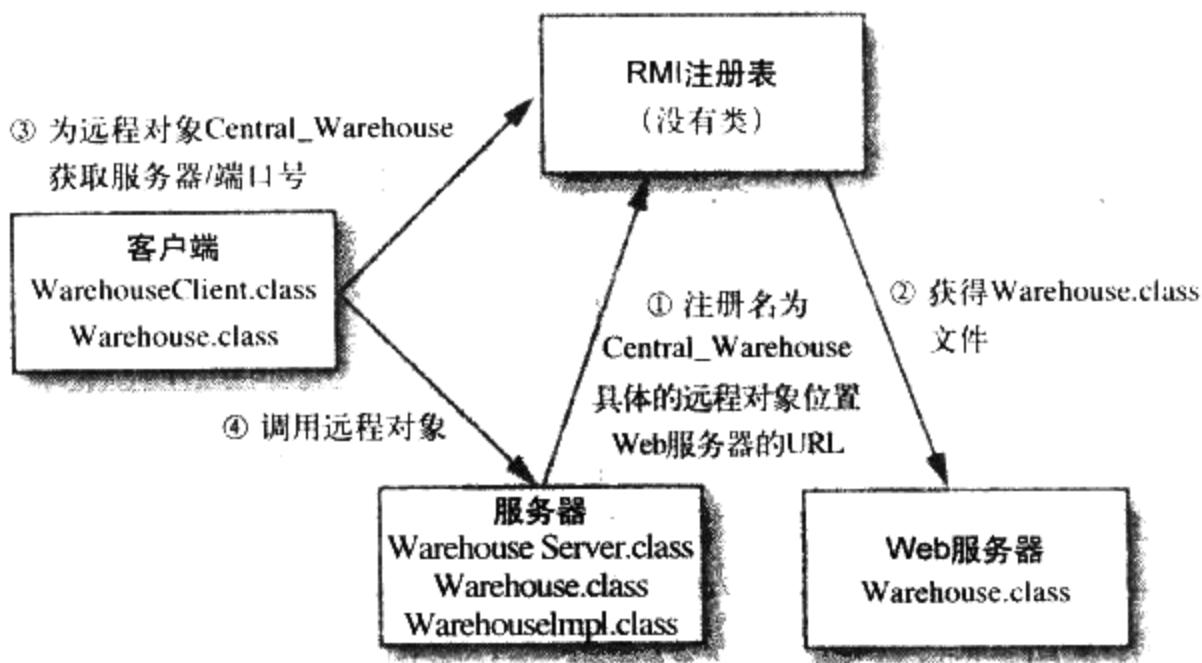


图10-5 在仓库应用中的服务器调用

 注意：由于安全的原因，作为JDK一部分的rmiregistry服务只允许来自同一台主机的绑定调用。也就是说，服务器和rmiregistry进程需要定位在同一台计算机上。但是，RMI架构允许更通用的支持多台服务器的RMI注册表实现。

为了测试这个示范应用，需要使用NanoHTTPD Web服务器，可以从<http://elonen.iki.fi/code/nanohttpd>处得到这个服务器。这个小型的Web服务器是在一个Java源文件中实现的。打开一个新的控制台窗口，转到download目录，然后将NanoHTTPD.java复制到这个目录中。使用下面的命令来编译该源文件并启动这个Web服务器。

```
java NanoHTTPD 8080
```

其中，命令行参数是端口号。如果你的机器上的8080端口已被占用，只需使用任何可用的端口。

接下来，打开另一个控制台窗口，转到不包含任何类文件的某个目录，并启动RMI注册表：  
rmiregistry

 警告：在启动RMI注册表之前，请确保CLASSPATH环境变量没有进行任何设置，并仔细检查当前目录是否确实不包含任何类文件。否则，RMI注册表可能会找到一些假冒的类文件，这使得注册表需要从另一个不同的来源下载额外的类文件时，产生混淆。这种行为的原因，可以查看<http://java.sun.com/javase/6/docs/technotes/guides/rmi/codebase.html>。简单地讲，每个存根对象都有一个代码基项，指出了它是从何处加载的。这个代码基被用来加载它所依赖的类。如果RMI注册表在本地找到了这样的类，那么它的代码基就会被设置为错误的值。

现在已经准备好启动服务器了。打开第三个控制台窗口，转到server目录，并执行下面的命令：

```
java -Djava.rmi.server.codebase=http://localhost:8080/ WarehouseServer
```

java.rmi.server.codebase属性指出了服务类文件的URL。服务器程序将这个URL传递给RMI注册表。

看一眼运行NanoHTTPD的控制台窗口，你就会发现一条消息，它在说明Warehouse.class文件已经注册到了RMI注册表中。

 警告：确保代码基URL以斜杠“/”结尾非常重要。

注意，服务器程序并没有退出。这看起来有些奇怪，毕竟这个程序只是创建并注册了一个WarehouseImpl对象。实际上，main方法在注册完之后并没有像你预料的那样立即退出。当你创建了一个扩展自UnicastRemoteObject的类的对象时，将会启动一个单独的线程，它将保持该程序无限地存活下去。因此，该程序仍旧在那里允许客户端连接它。

最后，打开第四个控制台窗口，转到client目录，运行：

```
java WarehouseClient
```

你将会看到一条短消息，表示运行方法被成功调用（参见图10-6）。

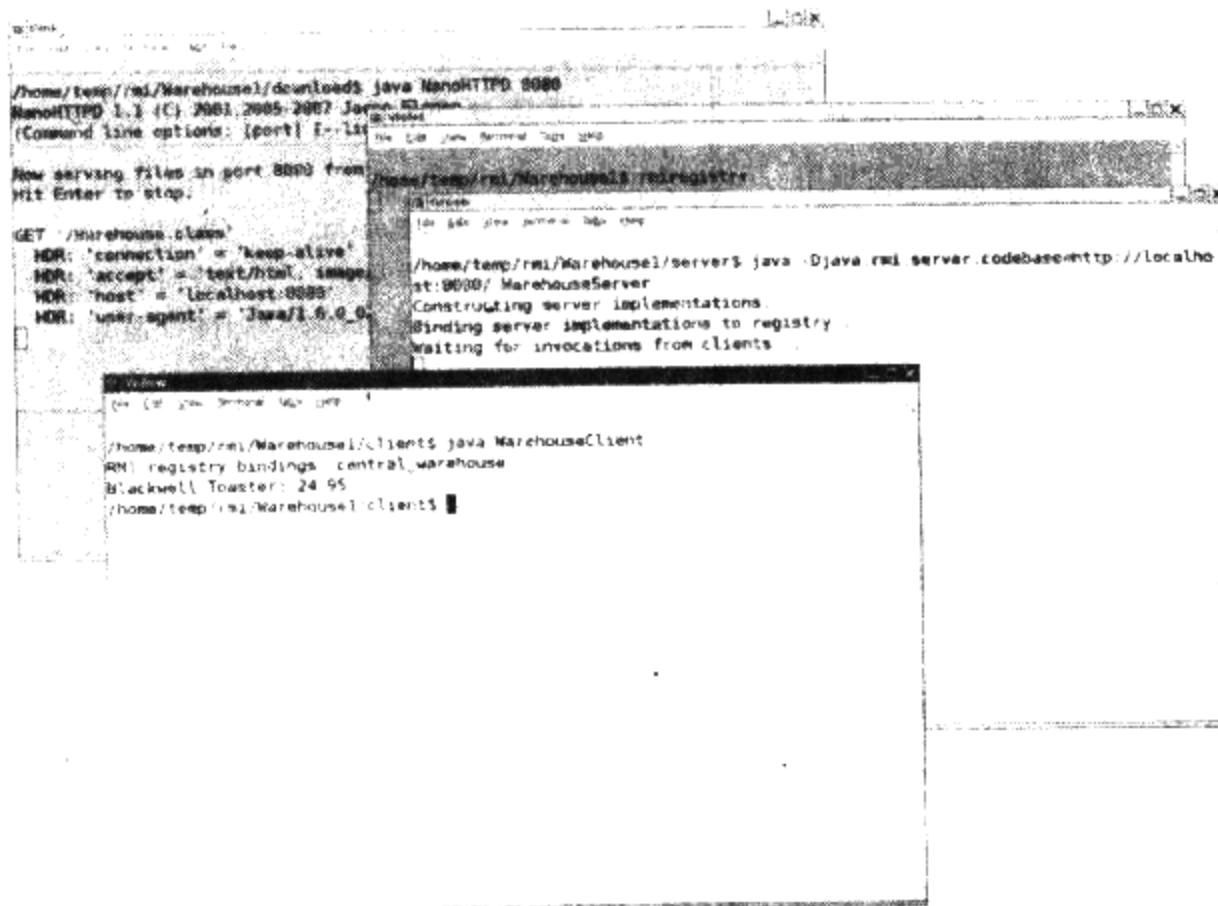


图10-6 测试一个RMI应用

 注意：如果你只想测试基本的程序逻辑，那么可以将客户端和服务器的类文件放在同一个目录下。然后在这个目录中启动RMI注册表、服务器和客户端。但是，因为RMI类的加载很容易造成失败和混淆，所以我们觉得最好还是向你展示正确的用于动态类加载的设置。

#### 10.3.4 记录RMI活动

如果用下面的选项启动服务器：

```
-Djava.rmi.server.logCalls=true WarehouseServer &
```

那么服务器会在其控制台上记录所有的远程方法调用。试着运行一下，你会对RMI的流量有深刻的印象。

如果想查看额外的日志信息，就必须用标准的Java日志API配置日志记录器。（参见第1卷第11章以了解有关日志记录的更多信息。）

可以用下面的内容创建一个logging.properties文件。

```
handlers=java.util.logging.ConsoleHandler
.level=FINE
java.util.logging.ConsoleHandler.level=FINE
java.util.logging.ConsoleHandler.formatter=java.util.logging.SimpleFormatter
```

我们可以对设置进行调整，方法是为每一个记录器设置单独的等级而不是设置全局等级。例如，要跟踪类的加载行为，可以设置为：

```
sun.rmi.loader.level=FINE
```

用以下选项启动RMI注册表

```
-Djava.util.logging.config.file=directory/logging.properties
```

表10-1 RMI记录器

| 记录器名                | 记录的活动    | 记录器名                    | 记录的活动          |
|---------------------|----------|-------------------------|----------------|
| sun.rmi.server.call | 服务器端远程调用 | sun.rmi.loader          | RMIClassLoader |
| sun.rmi.server.ref  | 服务器端远程引用 | sun.rmi.transport.misc  | 传输层            |
| sun.rmi.client.call | 客户端远程调用  | sun.rmi.transport.tcp   | TCP绑定与连接       |
| sun.rmi.client.ref  | 客户端远程引用  | sun.rmi.transport.proxy | HTTP通道         |
| sun.rmi.dgc         | 分布式垃圾收集  |                         |                |

用以下选项启动客户端与服务器

-Djava.util.logging.config.file=directory/logging.properties

下面有一个消息记录的例子，展示了类加载时的一个问题：RMI注册表找不到Warehouse类，因为Web服务器已经关闭了。

```
FINE: RMI TCP Connection(1)-127.0.1.1: (port 1099) op = 80
Oct 13, 2007 4:43:30 PM sun.rmi.server.LoaderHandler loadProxyClass
FINE: RMI TCP Connection(1)-127.0.1.1: interfaces = [java.rmi.Remote, Warehouse], codebase =
"http://localhost:8080/"
Oct 13, 2007 4:43:30 PM sun.rmi.server.LoaderHandler loadProxyClass
FINE: RMI TCP Connection(1)-127.0.1.1: proxy class resolution failed
java.lang.ClassNotFoundException: Warehouse
```

## 10.4 远程方法中的参数和返回值

在开始进行远程方法调用时，调用参数需要从客户端的虚拟机中移动到服务器的虚拟机中。在调用完成之后，返回值需要进行反方向传递。对于从一个虚拟机向另一个虚拟机传值，我们将其区分为两种情况：传递远程对象和传递非远程对象。例如，假设WarehouseServer的客户端将一个对Warehouse的引用（即，通过它可以调用远程的仓库对象的一个存根）传递给了另一个远程方法，这就是传递远程对象的实例。但是，大多数的方法参数都是普通的Java对象，而不是远程对象的存根。在我们的第一个示范应用中的getPrice方法的String参数就是这样的一个实例。

### 10.4.1 传递远程对象

当一个对远程对象的引用从一个虚拟机传递到另一个虚拟机时，该远程对象的发送者和接收者都将持有一个对同一个实体的引用。这个引用并非是一个内存位置（内存位置在单个虚拟机内才有意义），而是由网络地址和该远程对象的唯一标识符构成的。这个信息给封装在存根对象中。

从概念上讲，传递远程引用与在虚拟机内部传递本地对象引用很相似。但是，需要始终牢记的是在远程引用上的方法调用明显比在本地引用上的方法调用执行得慢，并且潜在地也更不可靠。

### 10.4.2 传递非远程对象

考虑一下getPrice方法的String参数。这个字符串值需要从客户端复制到服务器上，我们不难想象字符串的副本是如何经由网络传输的。RMI机制也可以生成更复杂的对象的副本，只

要这些对象是可序列化的。RMI使用第1章中描述的序列化机制来经由网络连接发送对象。这意味着任何实现了Serializable接口的类都可以用作参数和返回类型。

通过序列化来传递参数对远程方法的语义会产生很微妙的影响。当我们将对象传递给一个本地方法时，传递的是对象引用。如果该方法用某个修改方法对这个参数对象进行了操作，那么调用者会观察到这个变化。但是如果远程方法修改了某个序列化的参数，它修改的只是传递给它的副本，而调用者不会看到这种修改。

总结一下，在虚拟机之间传递值有两种机制：

- 实现了Remote接口的类的对象将作为远程引用传递。
- 实现了Serializable接口，但是没有实现Remote接口的类的对象将使用序列化进行复制。

这两种机制都是自动化的，而且不需要任何程序员的干预。请记住，序列化对于大型对象来说会比较慢，而且远程方法不能改变被序列化的参数。当然，你可以通过传递远程引用来避免这些问题。但是这么做代价太大：在远程引用上调用方法与调用本地方法相比，代价高得多。意识到这些开销有助于在设计远程服务时进行选择。



**注意：**远程对象是被自动垃圾回收的，就像本地对象一样。但是，分布式的垃圾收集器明显要复杂得多。当本地垃圾收集器发现对某个远程引用只有更多的本地使用时，它会通知分布式收集器这个服务器不再被该客户端所引用。当一个服务器不再被任何客户端所使用时，它就会被标记成垃圾。

我们的下一个示例程序将展示远程和可序列化对象的传递。我们将Warehouse接口修改为程序清单10-5的样子，如果给定关键词列表，这个仓库将返回最佳匹配的Product。

#### 程序清单10-5 Warehouse.java

```
1 import java.rmi.*;
2 import java.util.*;
3
4 /**
5  * The remote interface for a simple warehouse.
6  * @version 1.0 2007-10-09
7  * @author Cay Horstmann
8 */
9 public interface Warehouse extends Remote
10 {
11     double getPrice(String description) throws RemoteException;
12     Product getProduct(List<String> keywords) throws RemoteException;
13 }
```

getProduct方法的参数具有List<String>类型。因此，参数值必须属于实现了List<String>接口的可序列化的类，例如ArrayList<String>。（我们的示范客户端传递了一个通过调用Arrays.asList方法而获得的值。幸运的是，这个方法也可以保证返回一个可序列化的列表。）

返回类型Product封装了产品的描述、价格和位置，参见程序清单10-6。

注意，Product类是可序列化的，服务器构建了一个Product对象，而客户端获取了它的一

个副本（参见图10-7）。

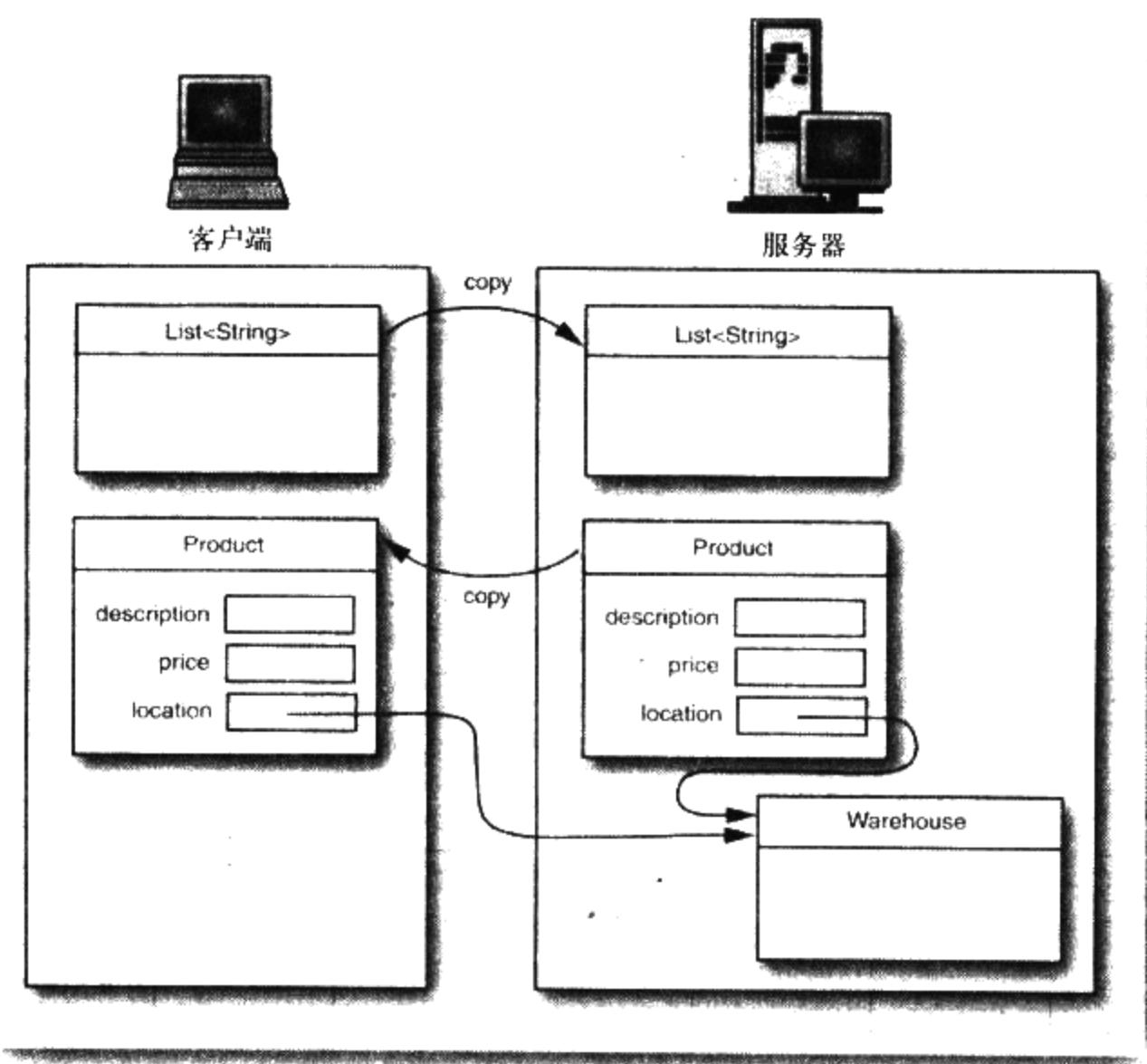


图10-7 复制本地参数和结果对象

### 程序清单10-6 Product.java

```

1. import java.io.*;
2.
3. public class Product implements Serializable
4. {
5.     public Product(String description, double price)
6.     {
7.         this.description = description;
8.         this.price = price;
9.     }
10.
11.    public String getDescription()
12.    {
13.        return description;
14.    }
15.
16.    public double getPrice()
17.    {
18.        return price;

```

```
19. }
20.
21. public Warehouse getLocation()
22. {
23.     return location;
24. }
25.
26. public void setLocation(Warehouse location)
27. {
28.     this.location = location;
29. }
30.
31. private String description;
32. private double price;
33. private Warehouse location;
34. }
```

有些细节值得注意。Product类有一个实例域，类型为远程接口Warehouse。仓库对象不是可序列化的，而它又包含大量的状态。因此，客户端接收了一个远程Warehouse对象的存根。这个存根可能与getProduct方法被调用的那个centralWarehouse存根会有所不同。在我们的实现中，有两种产品，烤箱和书，它们位于不同的仓库中。

#### 10.4.3 动态类加载

在我们下一个示范程序中，还有另一个微妙之处。关键词列表发送到了服务器，而且仓库也返回了Product类的一个实例。当然，客户端程序在编译时需要Product.class类文件。但是，只要我们的服务器程序无法找到关键词的匹配，它就会返回一件肯定让每个人都很高兴的产品：《Java核心技术》这本书。这个对象是Book类的实例，而Book类是Product的子类。

当客户端编译时，它也许从未看到过Book类。但是在运行时，它需要能够执行覆盖了Product方法的Book方法，这说明客户端需要拥有在运行时加载额外类的能力。客户端使用了与RMI注册表相同的机制，即类由Web服务器提供服务，RMI服务器类将URL传递给客户端，客户端创建要求下载这些类的HTTP请求。

只要一个程序从网络位置加载新的代码，就会涉及安全问题。正是由于这个原因，我们需要在动态加载类的RMI应用中使用安全管理器。（参见第9章有关类加载器和安全管理器的更多信息）。

使用RMI的程序应该安装一个安全管理器，去控制动态加载类的行为。可以用下面的指令安装：

```
System.setSecurityManager(new SecurityManager());
```



**注意：**如果所有的类在本地都可以获得，那么实际上就不需要安全管理器。如果你在部署时刻就了解程序中所有的类文件，那么你就可以全部在本地部署它们。但是，经常发生的情况是，客户端或服务器程序会随时间的推移而不断地演化，并添加进新的类。这时你就能从动态类加载中受益。只要你从其他来源加载代码，就需要安全管理器。

默认情况下，SecurityManager会限制程序中所有的代码去建立网络连接，但是，我们的

程序需要建立到三个远程位置的网络连接：

- 加载远程类的Web服务器
- RMI注册表
- 远程对象

为了允许这些操作，需要提供一个策略文件。（我们在第9章中十分详细地介绍过策略文件。）下面的策略文件允许一个应用建立任何到端口号至少为1024的某个端口的网络连接。（RMI端口默认为1099，远程对象使用的端口也大于等于1024。我们使用8080端口来下载类。）

```
grant
{
    permission java.net.SocketPermission
        "*:1024-65535", "connect";
};
```

需要通过将`java.security.policy`属性设置为这个策略文件名，来指示安全管理器去读取该策略文件。可以使用下面这个调用。

```
System.setProperty("java.security.policy", "rmi.policy");
```

或者，可以在命令行指定系统属性设置。

```
-Djava.security.policy=rmi.policy
```

为了运行示范程序，请确保已经关闭了在前面的示范程序中启动的RMI注册表、Web服务器和服务器程序。打开四个控制台窗口，然后执行下面的步骤：

- 1) 编译接口、实现、客户端和服务器类的源文件。

```
javac *.java
```

- 2) 创建三个目录：`client`、`server`和`download`，然后按下面这样组装它们：

```
client/
    WarehouseClient.class
    Warehouse.class
    Product.class
    client.policy
server/
    Warehouse.class
    Product.class
    Book.class
    WarehouseImpl.class
    WarehouseServer.class
    server.policy
download
    Warehouse.class
    Product.class
    Book.class
```

- 3) 在第一个控制台窗口中，转到不包含任何类文件的某个目录，启动RMI注册表。
- 4) 在第二个控制台窗口中，转到`download`目录下，启动NanoHTTP
- 5) 在第三个控制台窗口中，转到`server`目录下，启动服务器

```
java -Djava.rmi.server.codebase=http://localhost:8080/ WarehouseServer
```

- 6) 在第四个控制台窗口中，转到`client`目录下，运行客户端

```
java WarehouseClient
```

程序清单10-7展示了Book类的代码。注意，getDescription方法被覆盖为显示ISBN。当客户端程序运行时，它显示了《Java核心技术》这本书的ISBN，这证明Book类被动态加载了。程序清单10-8展示了仓库的实现。仓库拥有对一个备份仓库的引用。如果在这个仓库中找不到某个物品，就会搜索备份仓库。程序清单10-9展示了服务器程序。只有中心仓库进入了RMI注册表。注意，对备份仓库的远程引用可以传递给客户端，尽管它没有包括在RMI注册表中。当没有关键词的任何匹配，并且《Java核心技术》这本书（其location域引用的是备份仓库）被发送给客户端时，就属于这种情况。

### 程序清单10-7 Book.java

```
1. /**
2. * A book is a product with an ISBN number.
3. * @version 1.0 2007-10-09
4. * @author Cay Horstmann
5. */
6. public class Book extends Product
7 {
8     public Book(String title, String isbn, double price)
9     {
10         super(title, price);
11         this.isbn = isbn;
12     }
13
14     public String getDescription()
15     {
16         return super.getDescription() + " " + isbn;
17     }
18
19     private String isbn;
20 }
```

### 程序清单10-8 WarehouseImpl.java

```
1. import java.rmi.*;
2. import java.rmi.server.*;
3. import java.util.*;
4.
5. /**
6. * This class is the implementation for the remote Warehouse interface.
7. * @version 1.0 2007-10-09
8. * @author Cay Horstmann
9. */
10 public class WarehouseImpl extends UnicastRemoteObject implements Warehouse
11 {
12     /**
13      * Constructs a warehouse implementation.
14. */
15     public WarehouseImpl(Warehouse backup) throws RemoteException
16     {
17         products = new HashMap<String, Product>();
18         this.backup = backup;
19     }
20 }
```

```
21.     public void add(String keyword, Product product)
22.     {
23.         product.setLocation(this);
24.         products.put(keyword, product);
25.     }
26.
27.     public double getPrice(String description) throws RemoteException
28.     {
29.         for (Product p : products.values())
30.             if (p.getDescription().equals(description)) return p.getPrice();
31.         if (backup == null) return 0;
32.         else return backup.getPrice(description);
33.     }
34.
35.     public Product getProduct(List<String> keywords) throws RemoteException
36.     {
37.         for (String keyword : keywords)
38.         {
39.             Product p = products.get(keyword);
40.             if (p != null) return p;
41.         }
42.         if (backup != null)
43.             return backup.getProduct(keywords);
44.         else if (products.values().size() > 0)
45.             return products.values().iterator().next();
46.         else
47.             return null;
48.     }
49.
50.     private Map<String, Product> products;
51.     private Warehouse backup;
52. }
```

### 程序清单10-9 WarehouseServer.java

```
1. import java.rmi.*;
2. import javax.naming.*;
3.
4. /**
5. * This server program instantiates a remote warehouse objects, registers it with the naming
6. * service, and waits for clients to invoke methods.
7. * @version 1.12 2007-10-09
8. * @author Cay Horstmann
9. */
10.
11. public class WarehouseServer
12. {
13.     public static void main(String[] args) throws RemoteException, NamingException
14.     {
15.         System.setProperty("java.security.policy", "server.policy");
16.         System.setSecurityManager(new SecurityManager());
17.
18.         System.out.println("Constructing server implementation...");
19.         WarehouseImpl backupWarehouse = new WarehouseImpl(null);
20.         WarehouseImpl centralWarehouse = new WarehouseImpl(backupWarehouse);
21.     }
}
```

```
22.     centralWarehouse.add("toaster", new Product("Blackwell Toaster", 23.95));
23.     backupWarehouse.add("java", new Book("Core Java vol. 2", "0132354799", 44.95));
24.
25.     System.out.println("Binding server implementation to registry...");
26.     Context namingContext = new InitialContext();
27.     namingContext.bind("rmi:central_warehouse", centralWarehouse);
28.
29.     System.out.println("Waiting for invocations from clients...");
30. }
31. }
```

#### 10.4.4 具有多重接口的远程引用

一个远程类可以实现多个接口。考虑远程接口ServiceCenter。

```
public interface ServiceCenter extends Remote
{
    int getReturnAuthorization(Product prod) throws RemoteException;
}
```

现在假设WarehouseImp1类实现了这个接口与Warehouse接口。当一个对这种服务中心的引用被传递到另一个虚拟机时，接收者会获得一个可以访问在ServiceCenter与Warehouse接口中的所有远程方法的存根。可以使用instanceof操作符来查看一个特定的远程对象是否实现了某个接口。假设我们通过一个类型为Warehouse的变量得到了一个远程对象：

```
Warehouse location = product.getLocation();
```

这个远程对象可能是一个服务中心，但也可能不是。要确定到底是不是，可以使用下面的测试：

```
if (location instanceof ServiceCenter)
```

如果测试通过，就可以将location转型为ServiceCenter，并调用getReturnAuthorization方法。

#### 10.4.5 远程对象与equals、hashCode和clone方法

存入集中的对象必须覆写equals方法。如果是散列集或散列映射表，则需定义hashCode方法。然而，比较远程对象时有一个问题。如果要比较两个远程对象是否具有相同的内容，调用equals则必须联系包含这些对象的服务器，然后比较它们的内容。而该调用可能会失败。但是，Object类中的equals方法并未声明会抛出RemoteException异常，而远程接口中的所有方法都必须抛出该异常。因为子类的方法不能比它覆写的父类的方法抛出更多的异常，所以不能在远程接口中定义equals方法。hashCode也是这样。

相反，存根对象的equals和hashCode方法只是查看远程对象的位置。只要它们指向相同的远程对象，equals方法就认为两个存根相同。指向不同远程对象的存根绝不可能相同，即使那两个对象有一样的内容。相似的，散列码只能通过对对象的标识符来计算。

由于相同的技术上的原因，远程引用也没有clone方法。如果真的可以调用clone方法，让服务器克隆出一个远程对象，那么clone方法就有可能抛出RemoteException异常。然而，在根类Object中定义clone方法时，就已经保证绝不会抛出CloneNotSupportedException之外的

任何异常。

总之，可以使用集与散列表中的远程引用，但是必须记住，对它们进行等价测试以及散列计算并不会考虑远程对象的内容。你不能直接克隆远程引用。

## 10.5 远程对象激活

在前面的例子中，我们使用了一个服务器程序来初始化和注册对象，以便客户端能够对它们进行远程调用。然而，在某些情况下，初始化大量的远程对象，是一种浪费，因为无论客户端是否使用它们，它们都在一直等待连接。激活机制（activation）允许延迟构造远程对象，仅当至少有一个客户端调用远程对象上的远程方法时，才真正去构造该远程对象。

要享用激活机制的好处，客户端代码完全无需改动。客户端只是简单的请求一个远程引用，然后通过它进行调用而已。

然而，服务器程序则需由一个激活程序来代替。该程序构造了对象的激活描述符（activation descriptors），这样的对象可以延迟构造，并且该程序通过命名服务为远程方法调用绑定了接收者。第一次对这样的对象进行方法调用时，激活描述符中的信息将会用来构造该对象。

这样的远程对象必须继承Activatable类而不是UnicastRemoteObject类，当然，还需实现一个或多个远程接口。例如，

```
class WarehouseImpl
    extends Activatable
    implements Warehouse
{
    ...
}
```

因为对象的构造是延迟进行的，所以它必须以标准方式实现。因此，构造器必须包含以下两个参数：

- 一个激活ID（只需传递给父类的构造器）。
- 一个包含所有构造信息的对象，包装为MarshalledObject。

如果需要多个构造参数，必须将它们打包为一个对象。通常可以使用一个Object[]数组或一个ArrayList来达到此目的。

在构建激活描述符时，需要像下面这样从构造信息中构造一个MarshalledObject：

```
MarshalledObject<T> param = new MarshalledObject<T>(constructionInfo);
```

在实现对象的构造器中，使用MarshalledObject类的get方法来获得反序列化之后的构造信息：

```
T constructionInfo = param.get();
```

为了演示激活，我们修改了WarehouseImpl类，使得构造信息是一个由描述和价格构成的映射表。这个信息被封装到MarshalledObject中，并且在构造器中拆包出来：

```
public WarehouseImpl(ActivationID id, MarshalledObject<Map<String, Double>> param)
    throws RemoteException, ClassNotFoundException, IOException
{
    super(id, 0);
    prices = param.get();
```

```
    System.out.println("Warehouse implementation constructed.");
}
```

将0作为父类构造器的第二个参数，代表RMI类库应该分配一个合适的端口号作为监听端口。构造器会打印一个信息，这样就可以看到所需的产品对象已经被激活了。

 **注意：**其实远程对象不是一定要继承Activatable类，例如，可以将下面的静态方法调用放在服务器类的构造器中。

```
Activatable.exportObject(this, id, 0)
```

现在，我们来看看激活程序。首先，需要定义一个激活组。一个激活组描述了启动远程对象所在的虚拟机所需的公共参数，其中最重要的参数是安全策略。

然后如下构造一个激活组描述符：

```
Properties props = new Properties();
props.put("java.security.policy", "/path/to/server.policy");
ActivationGroupDesc group = new ActivationGroupDesc(props, null);
```

第二个参数描述了一个特殊的命令选项，在这个例子中不需要任何选项，所以我们传递了一个null引用。

接下来，创建一个组ID

```
ActivationGroupID id = ActivationGroup.getSystem().registerGroup(group);
```

现在就可以构造一个激活描述符了。对于需要构造的每一个对象，都应该包括：

- 激活组ID，对象将在与其对应的虚拟机上被构造；
- 类的名字（例如“ProductImpl”或“com.mycompany.MyClassImpl”）；
- URL字符串，由该URL加载类文件。这应该是基本URL，不含包的路径；
- 编组后的构造信息。

例如，

```
MarshalledObject param = new MarshalledObject(constructionInfo);
ActivationDesc desc = new ActivationDesc(id, "WarehouseImpl",
                                         "http://myserver.com/download/", param);
```

将此描述符传递给静态的Activatable.register方法。它返回一个对象，该对象实现了实现类的远程接口。可以使用命名服务绑定该对象：

```
Warehouse centralWarehouse = (Warehouse) Activatable.register(desc);
namingContext.bind("rmi:central_warehouse", centralWarehouse);
```

与前例的服务器程序不同，激活程序在注册与绑定激活接收者之后就会退出。仅当远程方法调用第一次发生时，才会构造远程对象。

程序清单10-10与10-11展示了激活程序和可激活的仓库类。仓库接口与客户端程序不用改变。要启动该程序，可按以下步骤进行：

- 1) 编译所有源文件。
- 2) 像下面这样发布类文件：

```
client/
  WarehouseClient.class
  Warehouse.class
```

```

server/
  WarehouseActivator.class
  Warehouse.class
  WarehouseImpl.class
  server.policy
download/
  Warehouse.class
  WarehouseImpl.class
rmi/
  rmid.policy

```

3) 在rmi目录(它不包含任何类文件)中启动RMI注册表。

4) 在rmi目录中启动RMI激活守护程序。

```
rmid -J-Djava.security.policy=rmid.policy
```

rmid程序监听激活请求，并且激活另一个虚拟机中的对象。要启动一个虚拟机，rmid程序需要一定的权限。这些权限都在一个策略文件中说明了(参见程序清单10-12)。使用-J，可以传递一个选项给运行激活守护程序的虚拟机。

5) 在download目录中启动NanoHTTP Web服务器

6) 从server目录运行激活程序。

```
java -Djava.rmi.server.codebase=http://localhost:8080/ WarehouseActivator
```

在使用命名服务对激活接收者进行注册之后，该程序就会退出。(你可能想知道为什么需要指定代码基，因为在激活描述符的构造器中已经提供了代码基。这是由于在激活描述符中的这个信息只是由RMI激活守护程序处理的。RMI注册表仍旧需要代码基去加载远程接口类。)

7) 从client目录运行客户端程序。

```
java WarehouseClient
```

该客户端程序会打印熟悉的产品描述。第一次运行客户端程序时，在激活守护程序所在的shell窗口中还可以看到构造器的消息。

#### **程序清单10-10 WarehouseActivator.java**

```

1. import java.io.*;
2. import java.rmi.*;
3. import java.rmi.activation.*;
4. import java.util.*;
5. import javax.naming.*;
6.
7. /**
8.  * This server program instantiates a remote warehouse object, registers it with the naming
9.  * service, and waits for clients to invoke methods.
10. * @version 1.12 2007-10-09
11. * @author Cay Horstmann
12. */
13.
14. public class WarehouseActivator
15. {
16.     public static void main(String[] args) throws RemoteException, NamingException,
17.         ActivationException, IOException
18.     {
19.         System.out.println("Constructing activation descriptors...");

```

```
20.  
21.     Properties props = new Properties();  
22.     // use the server.policy file in the current directory  
23.     props.put("java.security.policy", new File("server.policy").getCanonicalPath());  
24.     ActivationGroupDesc group = new ActivationGroupDesc(props, null);  
25.     ActivationGroupID id = ActivationGroup.getSystem().registerGroup(group);  
26.  
27.     Map<String, Double> prices = new HashMap<String, Double>();  
28.     prices.put("Blackwell Toaster", 24.95);  
29.     prices.put("ZapXpress Microwave Oven", 49.95);  
30.  
31.     MarshalledObject<Map<String, Double>> param = new MarshalledObject<Map<String, Double>>(  
32.         prices);  
33.  
34.     String codebase = "http://localhost:8080/";  
35.  
36.     ActivationDesc desc = new ActivationDesc(id, "WarehouseImpl", codebase, param);  
37.  
38.     Warehouse centralWarehouse = (Warehouse) Activatable.register(desc);  
39.  
40.     System.out.println("Binding activable implementation to registry...");  
41.     Context namingContext = new InitialContext();  
42.     namingContext.bind("rmi:central_warehouse", centralWarehouse);  
43.     System.out.println("Exiting...");  
44. }  
45. }
```

### 程序清单 10-11 WarehouseImpl.java

```
1. import java.io.*;  
2. import java.rmi.*;  
3. import java.rmi.activation.*;  
4. import java.util.*;  
5.  
6. /**  
7.  * This class is the implementation for the remote Warehouse interface.  
8.  * @version 1.0 2007-10-20  
9.  * @author Cay Horstmann  
10. */  
11. public class WarehouseImpl extends Activatable implements Warehouse  
12 {  
13.     public WarehouseImpl(ActivationID id, MarshalledObject<Map<String, Double>> param)  
14.         throws RemoteException, ClassNotFoundException, IOException  
15.     {  
16.         super(id, 0);  
17.         prices = param.get();  
18.         System.out.println("Warehouse implementation constructed.");  
19.     }  
20.  
21.     public double getPrice(String description) throws RemoteException  
22.     {  
23.         Double price = prices.get(description);  
24.         return price == null ? 0 : price;  
25.     }  
26.  
27.     private Map<String, Double> prices;
```

28 }

**程序清单10-12 rmid.policy**

```

1 grant
2 {
3     permission com.sun.rmi.rmid.ExecPermission
4         "${java.home}${/}bin${/}java";
5     permission com.sun.rmi.rmid.ExecOptionPermission
6         "-Djava.security.policy=";
7 }

```

**API java.rmi.activation.Activatable 1.2**

- **protected Activatable(ActivationID id, int port)**  
构造一个可激活对象，并设立一个监听端口。端口为0表示自动分配一个端口。
- **static Remote exportObject(Remote obj, ActivationID id, int port)**  
使一个远程对象成为可激活的对象。该方法返回一个可供远程调用的发起者使用的激活接收者。端口为0代表一个自动分配的端口。
- **static Remote register(ActivationDesc desc)**  
为一个可激活对象注册其描述符，做好接收远程调用的准备。该方法返回一个可供远程调用的发起者使用的激活接收者。

**API java.rmi.MarshalledObject 1.2**

- **MarshalledObject(Object obj)**  
构造一个对象，它包含给定对象序列化后的数据。
- **Object get()**  
将存储的对象数据反序列化，然后返回该对象。

**API java.rmi.activation.ActivationGroupDesc 1.2**

- **ActivationGroupDesc(Properties props, ActivationGroupDesc.CommandEnvironment env)**  
构造一个激活组的描述符，设置被激活对象所在虚拟机的一些属性。参数env包含启动虚拟机的路径和命令行选项，如果不需要特别的设置，可以使用null。

**API java.rmi.activation.ActivationGroup 1.2**

- **static ActivationSystem getSystem()**  
返回一个对激活系统的引用。

**API java.rmi.activation.ActivationSystem 1.2**

- **ActivationGroupID registerGroup(ActivationGroupDesc group)**  
注册一个激活组，并返回该组的ID。

**java.rmi.activation.ActivationDesc 1.2**

- ActivationDesc(ActivationGroupID id, String className, String classFileURL, MarshalledObject data)

构造一个激活描述符。

## 10.6 Web Services与JAX-WS

最近几年，Web服务已经呈现为远程方法调用的一种流行技术。从技术上讲，Web服务有两个组成部分：

- 一个通过将简单对象访问协议（SOAP）作为传输层协议就可以被访问到的服务器。
- 一个采用Web服务描述语言（WSDL）格式的服务描述文件。

SOAP是一个用于调用远程方法的XML协议，它与RMI用于客户端和服务器通信的协议类似。就像不用了解RMI协议的任何细节，就可以编写RMI应用程序一样，要调用一个Web服务，也不必了解SOAP的任何细节。

WSDL是一种接口描述语言，它也是基于XML的。WSDL文件描述了Web服务的接口：可以被调用的方法，以及它们的参数和返回值类型。在这一节中，我们从以Java实现的服务中生成了一个WSDL文件，这个文件包含了客户端程序调用该服务所需的所有信息，无论该客户端程序是用Java还是其他编程语言编写的。在下一节，我们将使用Amazon提供的WSDL编写一个调用Amazon的e-commerce服务的Java程序。我们并不知道该服务是用何种语言编写的。

### 10.6.1 使用JAX-WS

有若干种工具都可以支持用Java实现Web服务。在本节，我们讨论包含在Java SE6及以上版本中的JAX-WS。

如果使用JAX-WS，我们就不用为Web服务提供接口，而是可以用@WebService注解一个类，就像程序清单10-13所展示的那样。还要注意description参数的@WebParam注解，它赋予该参数一个人类可阅读的名字。（这个注解是可选的。默认情况下，该参数称为arg0。）

程序清单10-13 Warehouse.java

```
1 package com.horstmann.corejava;
2 import java.util.*;
3 import javax.jws.*;
4
5 /**
6 * This class is the implementation for a Warehouse web service
7 * @version 1.0 2007-10-09
8 * @author Cay Horstmann
9 */
10
11 @WebService
12 public class Warehouse
13 {
14     public Warehouse()
15     {
```

```

16.     prices = new HashMap<String, Double>();
17.     prices.put("Blackwell Toaster", 24.95);
18.     prices.put("ZapXpress Microwave Oven", 49.95);
19. }
20.
21. public double getPrice(@WebParam(name="description") String description)
22. {
23.     Double price = prices.get(description);
24.     return price == null ? 0 : price;
25. }
26.
27. private Map<String, Double> prices;
28. }

```

在RMI中，存根类是动态生成的，但是在使用JAX-WS时，需要运行工具来生成它们。转到Webservices1源文件的基目录，然后像下面这样运行wsgen类：

```
wsgen -classpath . com.horstmann.corejava.Warehouse
```

 注意：wsgen工具要求提供Web服务的类包含在默认包以外的包中。

这个工具会在com.horstmann.corejava.jaxws包中生成两个非常平凡的类。第一个类封装了Web服务中的getPrice调用的所有参数：

```

public class GetPrice
{
    private String description;
    public String getDescription() { return this.description; }
    public void setDescription(String description) { this.description = description; }
}

```

第二类封装了返回值：

```

public class GetPriceResponse
{
    private double _return;
    public double get_return() { return this._return; }
    public void set_return(double _return) { this._return = _return; }
}

```

典型情况下，人们会用一个复杂的服务器基础设施来部署Web服务，我们在这里就不讨论了。JDK包含了一个用于测试服务的非常简单的机制，我们可以直接调用Endpoint.publish方法。服务器将在给定的URL上启动，参见程序清单10-14。

#### 程序清单10-14 WarehouseServer.java

```

1. package com.horstmann.corejava;
2.
3. import javax.xml.ws.*;
4.
5. public class WarehouseServer
6. {
7.     public static void main(String[] args)
8.     {
9.         Endpoint.publish("http://localhost:8080/WebServices/warehouse", new Warehouse());
}

```

```
10  }
11 }
```

至此，应该编译服务器类，运行wsgen，然后启动服务器：

```
java com.horstmann.corejava.WarehouseServer
```

现在用Web浏览器浏览http://localhost:8080/WebServices/warehouse?wsdl，就会得到下面的WSDL文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://corejava.horstmann.com/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  targetNamespace="http://corejava.horstmann.com/" name="WarehouseService">
  <types>
    <xsd:schema>
      <xsd:import schemaLocation="http://localhost:8080/WebServices/warehouse?xsd=1"
        namespace="http://corejava.horstmann.com/"></xsd:import>
    </xsd:schema>
  </types>
  <message name="getPrice">
    <part element="tns:getPrice" name="parameters"></part>
  </message>
  <message name="getPriceResponse">
    <part element="tns:getPriceResponse" name="parameters"></part>
  </message>
  <portType name="Warehouse">
    <operation name="getPrice">
      <input message="tns:getPrice"></input>
      <output message="tns:getPriceResponse"></output>
    </operation>
  </portType>
  <binding name="WarehousePortBinding" type="tns:Warehouse">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"></soap:binding>
    <operation name="getPrice">
      <soap:operation soapAction=""></soap:operation>
      <input><soap:body use="literal"></soap:body></input>
      <output><soap:body use="literal"></soap:body></output>
    </operation>
  </binding>
  <service name="WarehouseService">
    <port name="WarehousePort" binding="tns:WarehousePortBinding">
      <soap:address location="http://localhost:8080/WebServices/warehouse"></soap:address>
    </port>
  </service>
</definitions>
```

这个描述告诉我们它提供了一个getPrice操作，其输入是tns:getPrice，而输出是tns:getPriceResponse。（这里，tns是目标名字空间http://corejava.horstmann.com的名字空间别名。）

要理解这些类型，可以让浏览器浏览http://localhost:8080/WebServices/warehouse?xsd=1。这样可以获得下面的XSL文件：

```
<xss:schema targetNamespace="http://corejava.horstmann.com/" version="1.0">
  <xss:element name="getPrice" type="tns:getPrice"/>
  <xss:element name="getPriceResponse" type="tns:getPriceResponse"/>
```

```

<xs:complexType name="getPrice">
    <xs:sequence><xs:element name="description" type="xs:string" minOccurs="0"/></xs:sequence>
</xs:complexType>
<xs:complexType name="getPriceResponse">
    <xs:sequence><xs:element name="return" type="xs:double"/></xs:sequence>
</xs:complexType>
</xs:schema>

```

现在可以看到getPrice有一个类型为String的description元素，而getPriceResponse有一个类型为double的return元素。

**注意：**WSDL文件没有指出服务要干什么，它只指出了参数和返回类型。

### 10.6.2 Web服务的客户端

让我们转而实现客户端。请记住，客户端对服务器，除了从WSDL中获得信息外，一无所知。为了生成可以和服务器通信的Java类，需要使用wsimport工具生成客户端类的一个集合：

```
wsimport -keep -p com.horstmann.corejava.server http://localhost:8080/WebServices/warehouse?wsdl
```

-keep选项将保持源文件，以防万一你还想浏览它们。下面的类和接口都是生成的：

```

GetPrice
GetPriceResponse
Warehouse
WarehouseService
ObjectFactory

```

你已经看到过GetPrice和GetPriceResponse类；

Warehouse接口定义了远程方法getPrice：

```

public interface Warehouse
{
    @WebMethod public double getPrice(@WebParam(name = "description") String description);
}

```

关于WarehouseService类，你只需要知道一件事：它的getPort方法生成了一个类型为Warehouse的存根，通过这个存根可以调用服务，参见程序清单10-15。

你可以忽略ObjectFactory类以及定义包级别注解的package-info.java文件。（我们将在第11章详细讨论注解。）

**注意：**对于所生成的类，可以用任何便利的形式打包。如果仔细观察，就会注意到GetPrice和GetPriceResponse类位于在服务器和客户端上不同的包中。这不会有问题是，毕竟，无论是服务器还是客户端都不了解彼此的Java实现。它们甚至不知道对方是否是用Java实现的。

#### 程序清单10-15 WarehouseClient.java

```

1 import java.rmi.*;
2 import javax.naming.*;
3 import com.horstmann.corejava.server.*;
4
5 /**

```

```
6 * The client for the warehouse program.  
7 * @version 1.0 2007-10-09  
8 * @author Cay Horstmann  
9 */  
10 public class WarehouseClient  
11 {  
12     public static void main(String[] args) throws NamingException, RemoteException  
13     {  
14         WarehouseService service = new WarehouseService();  
15         Warehouse port = service.getPort(Warehouse.class);  
16  
17         String descr = "Blackwell Toaster";  
18         double price = port.getPrice(descr);  
19         System.out.println(descr + ": " + price);  
20     }  
21 }
```

现在已经准备好运行客户端程序了。仔细检查服务器是否仍在运行，打开另一个shell窗口，并执行：

```
java WarehouseClient
```

你将会得到有关烤箱价格的非常熟悉的消息。

 **注意：**你可能想知道，这里为什么没有与RMI注册表等价的东西。这是因为在定位RMI的远程对象时，客户端不需要知道该对象被定位到了那个服务器上。它只需要知道如何定位注册表。但是，为了产生一个Web服务调用，客户端需要服务器的URL，它被硬编码在WarehouseService类中。

我们使用了一个网络探测器来查看客户端和服务器实际上是如何通信的（参见图10-8）。客户端将下面的请求发送给服务器：

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"  
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ns1="http://corejava.horstmann.com/">  
    <soapenv:Body>  
        <ns1:getPrice><description>Blackwell Toaster</description></ns1:getPrice>  
    </soapenv:Body>  
</soapenv:Envelope>
```

服务器将响应：

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"  
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ns1="http://corejava.horstmann.com/">  
    <soapenv:Body>  
        <ns1:getPriceResponse><return>24.95</return></ns1:getPriceResponse>  
    </soapenv:Body>  
</soapenv:Envelope>
```

在本节，你已经看到了有关Web服务的本质：

- 服务器是由WSDL文件定义的，该文件是XML格式的。
- 实际的请求和响应方法使用了SOAP，这是另一种XML格式。
- 客户端和服务器可以用任何语言编写。

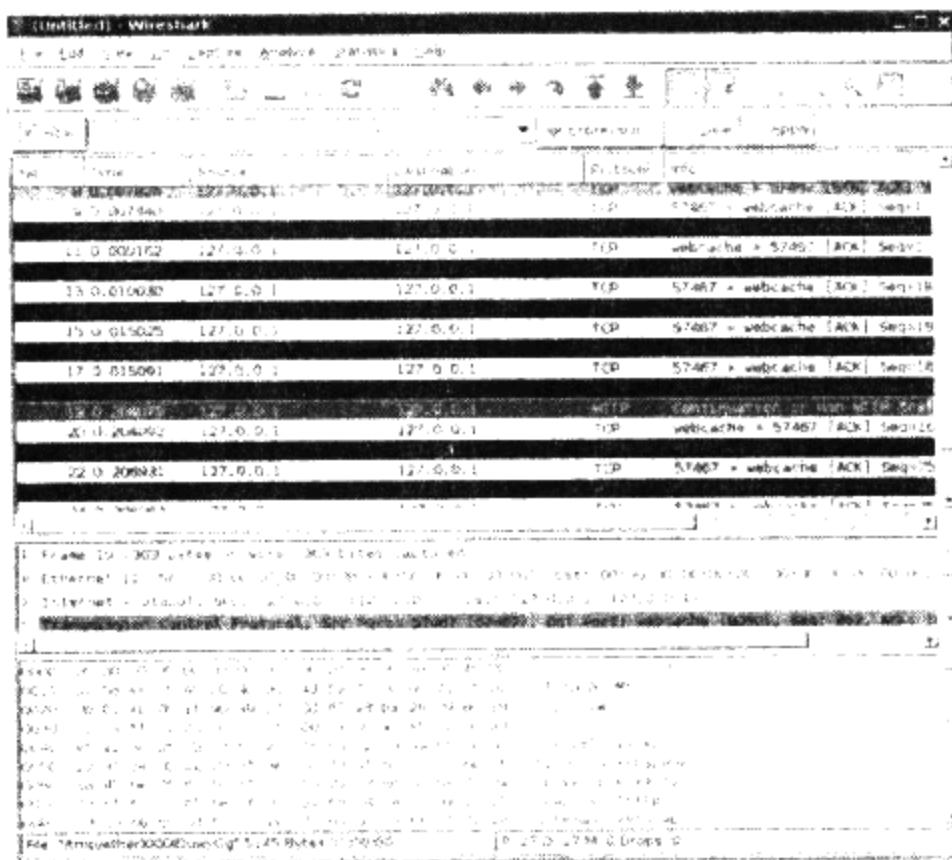


图10-8 分析SOAP的流量

### 10.6.3 Amazon的E-Commerce服务

为了使对Web服务的讨论更加有趣，我们先看一个实际的例子：Amazon的Web服务，在`http://www.amazon.com/gp/aws/landing.html`有其描述。这个电子商务web服务使程序员可以按自己所需与Amazon系统进行交互。举例来说，你可以指定作者或书名，从而列出所有相关书籍，也可以填满购物车然后下订单。Amazon使这样的服务成为可能，于是那些想把东西卖给顾客的公司都可以使用它，将Amazon的系统作为实现的后台。要运行我们的例子，必须到Amazon注册，以取得一个免费的开发者令牌，有了它才可以连接该服务。

或者，你可以采用本节描述的技巧，应用于其他的Web服务。`http://www.xmethods.com`网站列出了许多免费可用的Web服务。

让我们更仔细地看看Amazon的E-Commerce服务的WSDL（位于`http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl`），它描述了一个ItemSearch操作：

```

<operation name="ItemSearch">
  <input message="tns:ItemSearchRequestMsg"/>
  <output message="tns:ItemSearchResponseMsg"/>
</operation>

...
<message name="ItemSearchRequestMsg">
  <part name="body" element="tns:ItemSearch"/>
</message>
<message name="ItemSearchResponseMsg">
  <part name="body" element="tns:ItemSearchResponse"/>
</message>
```

下面是ItemSearch和ItemSearchResponse类型的定义：

```
<xss:element name="ItemSearch">
```

```

<xs:complexType>
  <xs:sequence>
    <xs:element name="MarketplaceDomain" type="xs:string" minOccurs="0"/>
    <xs:element name="AWSAccessKeyId" type="xs:string" minOccurs="0"/>
    <xs:element name="SubscriptionId" type="xs:string" minOccurs="0"/>
    <xs:element name="AssociateTag" type="xs:string" minOccurs="0"/>
    <xs:element name="XMLEscaping" type="xs:string" minOccurs="0"/>
    <xs:element name="Validate" type="xs:string" minOccurs="0"/>
    <xs:element name="Shared" type="tns:ItemSearchRequest" minOccurs="0"/>
    <xs:element name="Request" type="tns:ItemSearchRequest" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="ItemSearchResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:OperationRequest" minOccurs="0"/>
      <xs:element ref="tns:Items" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

通过使用JAX-WS技术， ItemSearch操作就变成了一个方法调用：

```

void itemSearch(String marketPlaceDomain, String awsAccessKeyId,
  String subscriptionId, String associateTag, String xmlEscaping, String validate,
  ItemSearchRequest shared, List<ItemSearchRequest> request,
  Holder<OperationRequest> opHolder, Holder<List<Items>> responseHolder)

```

ItemSearchRequest参数类型定义如下：

```

<xs:complexType name="ItemSearchRequest">
  <xs:sequence>
    <xs:element name="Actor" type="xs:string" minOccurs="0"/>
    <xs:element name="Artist" type="xs:string" minOccurs="0"/>
    ...
    <xs:element name="Author" type="xs:string" minOccurs="0"/>
    ...
    <xs:element name="ResponseGroup" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    ...
    <xs:element name="SearchIndex" type="xs:string" minOccurs="0"/>
    ...
  </xs:sequence>
</xs:complexType>

```

这个描述被转译成了一个类：

```

public class ItemSearchRequest
{
  public ItemSearchRequest() { ... }
  public String getActor() { ... }
  public void setActor(String newValue) { ... }
  public String getArtist() { ... }
  public void setArtist(String newValue) { ... }
  ...
  public String getAuthor() { ... }
  public void setAuthor(String newValue) { ... }
  ...
  public List<String> getResponseGroup() { ... }
}

```

```

...
public void setSearchIndex(String newValue) { ... }
...
}

```

要调用该查询服务，就要构造一个ItemSearchRequest对象，然后调用一个“port”对象的itemSearch方法。

```

ItemSearchRequest request = new ItemSearchRequest();
request.getResponseGroup().add("ItemAttributes");
request.setSearchIndex("Books");
Holder<List<Items>> responseHolder = new Holder<List<Items>>();
request.setAuthor(name);
port.itemSearch("", accessKey, "", "", "", "", request, null, null, responseHolder);

```

port类是自动生成的。port对象将Java对象翻译成SOAP消息，并将此消息传递给Amazon服务器，然后将返回的消息翻译为一个ItemSearchResponse对象，将响应置于“持有器”对象中。

 **注意：** Amazon关于参数和返回值的文档写得异常简要。但是，你可以填充<http://awszone.com/scratchpads/index.aws>处的表格，以查看SOAP请求和响应。这有助于你去猜测需要提供什么样的参数值以及可以得到什么样的返回值。

我们的示范应用（见程序清单10-16）很简单。首先，用户指定一个作者的名字，然后点击查询按钮。我们只是显示响应的第一页（见图10-9），这说明Web服务成功了。我们将扩充该应用的功能作为练习留给读者。

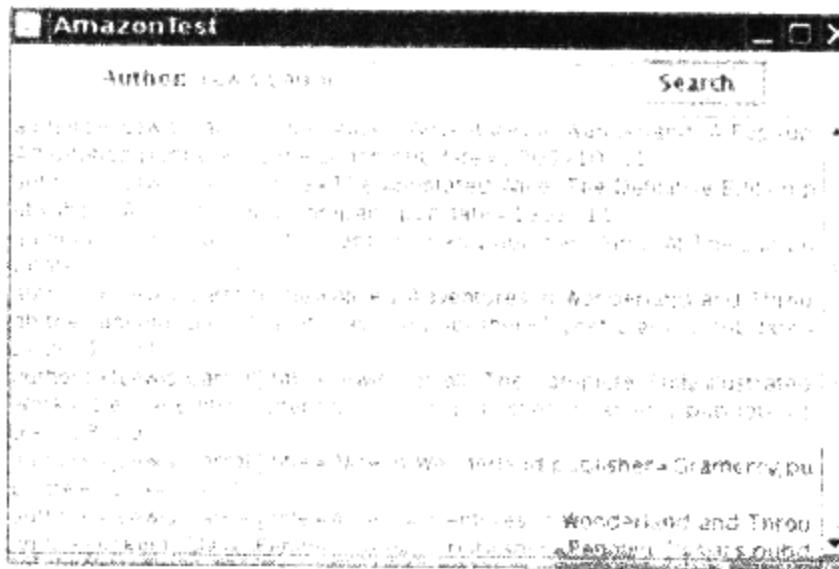


图10-9 连接到一个Web服务

要运行这个应用，首先要生成客户端需要的类：

```

wsimport -p com.horstmann.amazon
http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl

```

然后编辑AmazonTest.java文件，使其中包含你的Amazon密钥，然后编译运行：

```

javac AmazonTest.java
java AmazonTest

```

这个程序示例展示了调用Web服务从根本上与创建其他所有的远程方法调用一样，即程序员在一个代理对象上调用本地方法，而这个代理将连接到服务器上。由于Web服务在蓬勃发展，

因此它已经成为了应用程序员非常感兴趣的一种技术。

### 程序清单10-16 AmazonTest.java

```
1 import com.horstmann.amazon.*;
2 import java.awt.*;
3 import java.awt.event.*;
4 import java.util.List;
5 import javax.swing.*;
6 import javax.xml.ws.*;
7
8 /**
9  * The client for the Amazon e-commerce test program.
10 * @version 1.10 2007-10-20
11 * @author Cay Horstmann
12 */
13
14 public class AmazonTest
15 {
16     public static void main(String[] args)
17     {
18         JFrame frame = new AmazonTestFrame();
19         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20         frame.setVisible(true);
21     }
22 }
23
24 /**
25 * A frame to select the book author and to display the server response.
26 */
27 class AmazonTestFrame extends JFrame
28 {
29     public AmazonTestFrame()
30     {
31         setTitle("AmazonTest");
32         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
33
34         JPanel panel = new JPanel();
35
36         panel.add(new JLabel("Author:"));
37         author = new JTextField(20);
38         panel.add(author);
39
40         JButton searchButton = new JButton("Search");
41         panel.add(searchButton);
42         searchButton.addActionListener(new ActionListener()
43         {
44             public void actionPerformed(ActionEvent event)
45             {
46                 result.setText("Please wait...");
47                 new SwingWorker<Void, Void>()
48                 {
49                     @Override
50                     protected Void doInBackground() throws Exception
51                     {
52                         String name = author.getText();
53                         String books = searchByAuthor(name);
54                     }
55                 }.execute();
56             }
57         });
58     }
59 }
```

```
54.             result.setText(books);
55.             return null;
56.         }
57.     }.execute();
58. }
59. });
60.
61. result = new JTextArea();
62. result.setLineWrap(true);
63. result.setEditable(false);
64.
65. if (accessKey.equals("your key here"))
66. {
67.     result.setText("You need to edit the Amazon access key.");
68.     searchButton.setEnabled(false);
69. }
70.
71. add(panel, BorderLayout.NORTH);
72. add(new JScrollPane(result), BorderLayout.CENTER);
73. }
74.
75. /**
76. * Calls the Amazon web service to find titles that match the author.
77. * @param name the author name
78. * @return a description of the matching titles
79. */
80. private String searchByAuthor(String name)
81. {
82.     AWSECommerceService service = new AWSECommerceService();
83.     AWSECommerceServicePortType port = service.getPort(AWSECommerceServicePortType.class);
84.     ItemSearchRequest request = new ItemSearchRequest();
85.     request.getResponseGroup().add("ItemAttributes");
86.     request.setSearchIndex("Books");
87.
88.     Holder<List<Items>> responseHolder = new Holder<List<Items>>();
89.     request.setAuthor(name);
90.     port.itemSearch("", accessKey, "", "", "", "", request, null, null, responseHolder);
91.
92.     List<Item> response = responseHolder.value.get(0).getItem();
93.
94.     StringBuilder r = new StringBuilder();
95.     for (Item item : response)
96.     {
97.         r.append("authors=");
98.         List<String> authors = item.getItemAttributes().getAuthor();
99.         r.append(authors);
100.        r.append(",title=");
101.        r.append(item.getItemAttributes().getTitle());
102.        r.append(",publisher=");
103.        r.append(item.getItemAttributes().getPublisher());
104.        r.append(",pubdate=");
105.        r.append(item.getItemAttributes().getPublicationDate());
106.        r.append("\n");
107.    }
108.    return r.toString();
109. }
110.
```

```
111.     private static final int DEFAULT_WIDTH = 450;
112.     private static final int DEFAULT_HEIGHT = 300;
113.
114.     private static final String accessKey = "12Y1EEATQ8DDYJCVQYR2";
115.
116.     private JTextField author;
117.     private JTextArea result;
118. }
```

你已经看到了RMI机制，这是一种用于Java程序的非常复杂的分布式编程模型，它在Java EE架构中得到了广泛的运用。你还对Web服务有了初步的了解，它允许你以独立于编程语言的方式连接到客户端和服务器。在下一章，我们将转向Java编程的另一个完全不同的方面：与同一台机器上用不同的编程语言编写的“本地”代码进行交互。

# 第11章 脚本、编译与注解处理

▲ Java平台的脚本

▲ 编译器API

▲ 使用注解

▲ 注解语法

▲ 标准注解

▲ 源码级注解处理

▲ 字节码工程

本章介绍了三种用于处理代码的技术：脚本API使你可以调用诸如JavaScript和Groovy这样的脚本语言代码；当你希望在应用程序内部编译Java代码时，可以使用编译器API；注解处理器可以在包含注解的Java源代码和类文件上进行操作。如你所见，有许多应用程序都可以用来处理注解，从简单的诊断到“字节码工程”，后者可以将字节码插入到类文件中，甚至可以插入到运行着的程序中。

## 11.1 Java平台的脚本

脚本语言是一种通过在运行时解释程序文本，从而避免使用通常的编辑/编译/链接/运行循环的语言。脚本语言有许多优势：

- 便于快速变更，鼓励不断试验。
- 可以修改运行着的程序的行为。
- 支持程序用户的定制化。

另一方面，大多数脚本语言都缺乏可以使编写复杂应用受益的特性，例如强类型、封装和模块化。

因此人们在尝试将脚本语言和传统语言的优势相结合。脚本API使你可以在Java平台上实现这个目的，它支持在Java程序中对用JavaScript、Groovy、Ruby，甚至是更奇异的诸如Scheme和Haskell等语言编写的脚本进行调用。（至于另一个方向，即从脚本语言中访问Java则属于脚本语言提供商的职责。大多数运行在Java虚拟机上的脚本语言都具有这种能力。）

在下面的小节中，我们将向你展示如何为某种特定的语言选择一个引擎，如何执行脚本，以及如何利用某些脚本引擎提供的先进特性。

### 11.1.1 获取脚本引擎

脚本引擎是一个可以执行用某种特定语言编写的脚本的类库。当虚拟机启动时，它会发现可用的脚本引擎。为了枚举这些引擎，需要构造一个`ScriptEngineManager`，并调用`getEngineFactories`方法。可以向每个引擎工厂去询问它们所支持的引擎名、MIME类型和文件扩展名。表11-1显示了这些内容的典型值。

表11-1 脚本引擎工厂的属性

| 引 擎                    | 名 字   | MIME类型s   | 文 件 扩 展            |
|------------------------|---|---|--------------------|
| Rhino (包含在 Java SE 6中) | js, rhino, JavaScript, javascript, ECMAScript, ecmascript | application/javascript, application/ecmascript, text/javascript, text/ecmascript] | js                 |
| Groovy                 | groovy  | 无   | groovy             |
| SISC Scheme            | scheme, sisc  | 无   | scc, sce, scm, shp |

通常，你知道所需要的引擎，因此可以直接通过名字、MIME类型或文件扩展来请求它，例如：

```
ScriptEngine engine = manager.getEngineByName("JavaScript");
```

Java SE 6 包含Rhino的一个版本，这是由Mozilla基金开发的一个JavaScript解释器。可以通过在类路径中提供必要的JAR文件来添加对额外的语言支持。这时一般需要两个JAR文件集合，脚本语言自身是由单个的JAR文件或一个JAR集合实现的，而将这种语言适配到脚本API的引擎通常还需要一个额外的JAR。<http://scripting.dev.java.net>站点提供的引擎可以适用于大量的脚本语言，例如，为了添加对Groovy的支持，类路径必须包含groovy/lib/\*（来自<http://groovy.codehaus.org>）和groovy-engine.jar（来自<http://scripting.dev.java.net>）。

#### API javax.script.ScriptEngineManager 6

- `List<ScriptEngineFactory> getEngineFactories()`  
获取所有发现的引擎工厂的列表。
- `ScriptEngine getEngineByName(String name)`
- `ScriptEngine getEngineByExtension(String extension)`
- `ScriptEngine getEngineByMimeType(String mimeType)`  
获取给定名字、脚本文件扩展名或MIME类型的脚本引擎。

#### API javax.script.ScriptEngineFactory 6

- `List<String> getNames()`
  - `List<String> getExtensions()`
  - `List<String> getMimeTypes()`
- 获取该工厂所了解的名字、脚本文件扩展名和MIME类型。

### 11.1.2 脚本赋值与绑定

一旦拥有了引擎，就可以通过下面的调用来直接调用脚本：

```
Object result = engine.eval(scriptString);
```

如果脚本存储在文件中，那么需要先打开一个Reader，然后调用：

```
Object result = engine.eval(reader);
```

可以在同一个引擎上调用多个脚本。如果一个脚本定义了变量、函数或类，那么大多数引擎都会保留这些定义，以供将来使用。例如：

```
engine.eval("n = 1728");
Object result = engine.eval("n + 1");
```

将返回1729。

 **注意：**要想知道在多个线程中并发执行脚本是否安全，可以调用

```
Object param = factory.getParameter("THREADING");
```

其返回的是下列值之一：

- `null`: 并发执行不安全。
- "`MULTITHREADED`": 并发执行安全。一个线程的执行效果对另外的线程有可能是可视的。
- "`THREAD-ISOLATED`": 除了"`MULTITHREADED`"之外，会为每个线程维护不同的变量绑定。
- "`STATELESS`": 除了"`THREAD-ISOLATED`"之外，脚本不会改变变量绑定。

我们经常希望能够向引擎中添加新的变量绑定。绑定由名字及其关联的Java对象构成。例如，考虑下面的语句：

```
engine.put(k, 1728);
Object result = engine.eval("k + 1");
```

脚本代码从“引擎作用域”中的绑定里读取`k`的定义。这一点非常重要，因为大多数脚本语言都可以访问Java对象，通常使用的是比Java语法更简单的语法。例如，

```
engine.put(b, new JButton());
engine.eval("f.text = 'Ok'");
```

反过来，也可以获取由脚本语句绑定的变量：

```
engine.eval("n = 1728");
Object result = engine.get("n");
```

除了引擎作用域之外，还有全局作用域。任何添加到`ScriptEngineManager`中的绑定对所有引擎都是可视的。

除了向引擎或全局作用域添加绑定之外，还可以将绑定收集到一个类型为`Bindings`的对象中，然后将其传递给`eval`方法：

```
Bindings scope = engine.createBindings();
scope.put(b, new JButton());
engine.eval(scriptString, scope);
```

如果绑定集不应该为了将来对`eval`方法的调用而进行持久化，那么这么做就很有用。

 **注意：**你可能希望除了引擎作用域和全局作用域之外还有其他的作用域。例如，web容器可能需要请求作用域或会话作用域。但是，这需要你自己去解决。你需要实现一个类，它实现了`ScriptContext`接口，并管理着一个作用域集合。每个作用域都是由一个整数标识的，而且越小的数字应该越先被搜索。（标准类库提供了`SimpleScriptContext`类，但是它只能持有全局作用域和引擎作用域。）

**API** **javax.script.ScriptEngine 6**

- Object eval(String script)
- Object eval(Reader reader)
- Object eval(String script, Bindings bindings)
- Object eval(Reader reader, Bindings bindings)

对由字符串或读取器给定的脚本赋值，并服从给定的绑定。

- Object get(String key)
- void put(String key, Object value)

在引擎作用域内获取或放置一个绑定。

- Bindings createBindings()

创建一个适合该引擎的空Bindings对象。

**API** **javax.script.ScriptEngineManager 6**

- Object get(String key)
- void put(String key, Object value)

在全局作用域内获取或放置一个绑定。

**API** **javax.script.Bindings 6**

- Object get(String key)
- void put(String key, Object value)

在由该Bindings对象表示的作用域内获取或放置一个绑定。

### 11.1.3 重定向输入和输出

可以通过调用脚本上下文的setReader和setWriter方法来重定向脚本的标准输入和输出。例如，

```
StringWriter writer = new StringWriter();
engine.getContext().setWriter(new PrintWriter(writer, true));
```

在上例中，任何用JavaScript的print和println函数产生的输出都会被发送到writer。

**X** **警告：**可以向setWriter方法传递任何Writer，但是如果传递的不是PrintWriter，Rhino引擎会抛出异常。

setReader和setWriter方法只会影响脚本引擎的标准输入和输出源。例如，如果执行下面的JavaScript代码：

```
println("Hello");
java.lang.System.out.println("World");
```

则只有第一个输出会被重定向。

Rhino引擎没有标准输入源的概念，因此调用setReader没有任何效果。

**API** javax.script.ScriptEngine 6

- ScriptContext getContext()

获得该引擎的默认的脚本上下文。

**API** javax.script.ScriptContext 6

- Reader getReader()
- void setReader(Reader reader)
- Writer getWriter()
- void setWriter(Writer writer)
- Writer getErrorWriter()
- void setErrorWriter(Writer writer)

获取或设置用于输入的读入器或用于正常与错误输出的写出器。

#### 11.1.4 调用脚本的函数和方法

在使用许多脚本引擎时，都可以调用脚本语言的函数，而不必对实际的脚本代码赋值。如果允许用户用他们所选择的脚本语言来实现服务，那么这种机制就很有用了。

提供这种功能的脚本引擎实现了`Invocable`接口。特别是，Rhino引擎就是实现了`Invocable`接口。

要调用一个函数，需要用函数名来调用`invokeFunction`方法，函数名后面是函数的参数：

```
if (engine implements Invocable)
    ((Invocable) engine).invokeFunction("aFunction", param1, param2);
```

如果脚本语言是面向对象的，那就可以像下面这样调用方法：

```
((Invocable) engine).invokeMethod(implicitParam, "aMethod", explicitParam1, explicitParam2);
```

这里，`implicitParam`对象是脚本语言编写的对象的一个代理，它必须是前一个对脚本引擎调用的结果。

 **注意：**即使脚本引擎没有实现`Invocable`接口，你也可能仍旧可以以一种独立于语言的方式来调用某个方法。`ScriptEngineFactory`类的`getMethodCallSyntax`方法可以产生一个字符串，你可以将其传递给`eval`方法。但是，所有的方法参数必须都与名字绑定，尽管可以用任意值调用`invokeMethod`。

我们可以更进一步，让脚本引擎去实现一个Java接口，然后就可以用Java方法调用的语法来调用脚本函数。

其细节依赖于脚本引擎，但是典型情况是我们需要为该接口中的每个方法都提供一个函数。例如，考虑下面的Java接口：

```
public interface Greeter
{
    String greet(String whom);
}
```

在Rhino中，可以提供下面的函数：

```
function greet(x) { return "Hello, " + x + "!"; }
```

这段代码必须先赋值。然后可以调用：

```
Greeter g = ((Invocable) engine).getInterface(Greeter.class);
```

现在可以产生一个普通的Java方法调用：

```
String result = g.greet("World");
```

在幕后，JavaScript的greet方法将被调用。这种方式与第10章讨论的产生远程方法调用类似。

在面向对象的脚本语言中，可以通过相匹配的Java接口来访问一个脚本类。例如，考虑下面的JavaScript代码，它定义了一个SimpleGreeter类。

```
function SimpleGreeter(salutation) { this.salutation = salutation; }
SimpleGreeter.prototype.greet = function(whom) { return this.salutation + ", " + whom + "!"; }
```

你可以使用这个类来构造使用不同的问候语（例如Hello、Goodbye等）打招呼的致敬者。



**注意：**有关如何用JavaScript定义类的更多信息，可以参阅David Flanagan撰写的《*JavaScript-The Definitive Guide*, 5th ed.》<sup>Θ</sup>一书。（O'Reilly出版社2006年出版）

在对JavaScript的类定义赋值之后，可以调用：

```
Object goodbyeGreeter = engine.eval("new SimpleGreeter('Goodbye')");
Greeter g = ((Invocable) engine).getInterface(goodbyeGreeter, Greeter.class);
```

当调用g.greet("World")时，greet方法会在JavaScript对象goodbyeGreeter上被调用。其结果是字符串"Goodbye, World!"。

总之，如果你希望从Java中调用脚本代码，同时又不想因这种脚本语言的语法而受到困扰，那么Invocable接口就很有用。



#### javax.script.Invocable 6

- Object invokeFunction(String name, Object... parameters)
- Object invokeMethod(Object implicitParameter, String name, Object... explicitParameters)

用给定的名字调用函数或方法，并传递给定的参数。

- <T> T getInterface(Class<T> iface)

返回给定接口的实现，该实现用脚本引擎中的函数实现了接口中的方法。

- <T> T getInterface(Object implicitParameter, Class<T> iface)

返回给定接口的实现，该实现用给定对象的方法实现了接口中的方法。

### 11.1.5 编译脚本

某些脚本引擎出于对执行效率的考虑，可以将脚本代码编译为某种中间格式。这些引擎实现了Compilable接口。下面的示例展示了如何编译和赋值包含在脚本文件中的代码：

```
Reader reader = new FileReader("myscript.js");
CompiledScript script = null;
if (engine implements Compilable)
```

<sup>Θ</sup> 《JavaScript权威指南》中文版由机械工业出版社出版，书号为：7-111-11091-9——编辑注。

```
CompiledScript script = ((Compilable) engine).compile(reader);
```

一旦该脚本被编译，就可以执行它。下面的代码将会在编译成功的情况下执行编译后的脚本，如果引擎不支持编译，则执行原始的脚本。

```
if (script != null)
    script.eval();
else
    engine.eval(reader);
```

当然，只有需要重复执行时，我们才希望编译脚本。

#### **API** javax.script.Compilable 6

- CompiledScript compile(String script)
- CompiledScript compile(Reader reader)

编译由字符串或读入器给定的脚本。

#### **API** javax.script.CompiledScript 6

- Object eval()
- Object eval(Bindings bindings)

对该脚本赋值。

### 11.1.6 一个示例：用脚本处理GUI事件

为了演示脚本API，我们将开发一个示范程序，它允许用户指定使用他们所选择的脚本语言编写事件处理器。

让我们看看程序清单11-1中的程序，ButtonFrame类与第1卷中介绍的事件处理演示程序类似，但是有两个差异：

- 每个构件都有其自己的name属性集。
- 没有任何事件处理器。

事件处理器是在属性文件中定义的。每个属性定义都具有下面的形式：

*componentName.eventName* = *scriptCode*

例如，如果选择使用JavaScript，那就要在js.properties文件中提供事件处理器：

```
yellowButton.action=panel.background = java.awt.Color.YELLOW
blueButton.action=panel.background = java.awt.Color.BLUE
redButton.action=panel.background = java.awt.Color.RED
```

本书附带的代码还包括用于Groovy和SISC Scheme的文件。

该程序以加载在命令行中指定的语言所需的引擎开始，这看起来大体上是个好主意。而且，Scheme解释器需要某些麻烦的初始化工作，我们不希望在每个事件处理器的脚本中都包括这部分工作。

接下来，我们递归地遍历所有的子构件，并将绑定（名字，对象）添加到引擎作用域中。

然后，我们读入language.properties文件。对于每一个属性，都合成其事件处理器代理，使得脚本代码可执行。其细节有些技术性，如果你希望了解实现的细节，请参阅第1卷第6章有关代理的小节，和本卷第8章有关JavaBean事件的小节。但是，其精髓部分还是每个事件处理

器都调用。

```
engine.eval(scriptCode);
```

让我们详细看看yellowButton。当下面一行被处理时，

```
yellowButton.action=panel.background = java.awt.Color.YELLOW
```

我们找到了具有“yellowButton”名字的JButton构件，然后附着一个ActionListener，它拥有actionPerformed方法，该方法将执行下面的脚本。

```
panel.setBackground( java.awt.Color.YELLOW );
```

引擎包含一个将名字“panel”与这个JPanel对象绑定在一起的绑定。当事件发生时，该面板的setBackground方法就会执行，并且其颜色也会改变。

只需要执行下面的调用，就可以用JavaScript事件处理器运行这个程序：

## java ScriptTest

对于Groovy处理器，需要使用

```
java -classpath .:groovy/lib/*:jsr223-engines/groovy/build/groovy-engine.jar ScriptTest groovy
```

这里，`groovy`是Groovy的安装目录，`jsr223-engines`是包含从<http://scripting.dev.java.net>获得的引擎适配器的目录。

要试验Scheme，则需要从<http://sisc-scheme.org/>下载SISC Scheme，并运行：

```
java -classpath .:sisc/*:jsr223-engines/scheme/build/scheme-engine.jar ScriptTest scheme
```

这个应用演示了如何在Java GUI编程中使用脚本。大家可以更进一步，用XML文件来描述GUI，就像在第2章中看到的那样。然后我们的程序就会变成解释器，去解释那些由XML文件定义可视化表示以及用脚本语言定义行为的GUI。请注意这与动态HTML页面或动态服务器端脚本环境之间的相似性。

### 程序清单11-1 ScriptTest.java

```
23.         else language = args[0];
24.
25.         ScriptEngineManager manager = new ScriptEngineManager();
26.         System.out.println("Available factories: ");
27.         for (ScriptEngineFactory factory : manager.getEngineFactories())
28.             System.out.println(factory.getEngineName());
29.         final ScriptEngine engine = manager.getEngineByName(language);
30.
31.         if (engine == null)
32.         {
33.             System.err.println("No engine for " + language);
34.             System.exit(1);
35.         }
36.
37.         ButtonFrame frame = new ButtonFrame();
38.
39.         try
40.         {
41.             File initFile = new File("init." + language);
42.             if (initFile.exists())
43.             {
44.                 engine.eval(new FileReader(initFile));
45.             }
46.
47.             getComponentBindings(frame, engine);
48.
49.             final Properties events = new Properties();
50.             events.load(new FileReader(language + ".properties"));
51.             for (final Object e : events.keySet())
52.             {
53.                 String[] s = ((String) e).split("\\.");
54.                 addListener(s[0], s[1], (String) events.get(e), engine);
55.             }
56.         }
57.         catch (Exception e)
58.         {
59.             e.printStackTrace();
60.         }
61.
62.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
63.         frame.setTitle("ScriptTest");
64.         frame.setVisible(true);
65.     }
66. );
67. }
68.
69. /**
70. * Gathers all named components in a container.
71. * @param c the component
72. * @param namedComponents
73. */
74. private static void getComponentBindings(Component c, ScriptEngine engine)
75. {
76.     String name = c.getName();
77.     if (name != null) engine.put(name, c);
78.     if (c instanceof Container)
79.     {
```

```
80.         for (Component child : ((Container) c).getComponents())
81.             getComponentBindings(child, engine);
82.     }
83. }
84.
85. /**
86. * Adds a listener to an object whose listener method executes a script.
87. * @param beanName the name of the bean to which the listener should be added
88. * @param eventName the name of the listener type, such as "action" or "change"
89. * @param scriptCode the script code to be executed
90. * @param engine the engine that executes the code
91. * @param bindings the bindings for the execution
92. */
93. private static void addListener(String beanName, String eventName, final String scriptCode,
94.                                 final ScriptEngine engine) throws IllegalArgumentException, IntrospectionException,
95.                                 IllegalAccessException, InvocationTargetException
96. {
97.     Object bean = engine.get(beanName);
98.     EventSetDescriptor descriptor = getEventSetDescriptor(bean, eventName);
99.     if (descriptor == null) return;
100.    descriptor.getAddListenerMethod().invoke(
101.        bean,
102.        Proxy.newProxyInstance(null, new Class[] { descriptor.getListenerType() },
103.                               new InvocationHandler()
104.                               {
105.                                   public Object invoke(Object proxy, Method method, Object[] args)
106.                                       throws Throwable
107.                                   {
108.                                       engine.eval(scriptCode);
109.                                       return null;
110.                                   }
111.                               }));
112.
113. }
114.
115. private static EventSetDescriptor getEventSetDescriptor(Object bean, String eventName)
116.     throws IntrospectionException
117. {
118.     for (EventSetDescriptor descriptor : Introspector.getBeanInfo(bean.getClass())
119.          .getEventSetDescriptors())
120.         if (descriptor.getName().equals(eventName)) return descriptor;
121.     return null;
122. }
123. }
124.
125. class ButtonFrame extends JFrame
126. {
127.     public ButtonFrame()
128.     {
129.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
130.
131.         panel = new JPanel();
132.         panel.setName("panel");
133.         add(panel);
134.
135.         yellowButton = new JButton("Yellow");
136.         yellowButton.setName("yellowButton");
```

```
137.     blueButton = new JButton("Blue");
138.     blueButton.setName("blueButton");
139.     redButton = new JButton("Red");
140.     redButton.setName("redButton");
141.
142.     panel.add(yellowButton);
143.     panel.add(blueButton);
144.     panel.add(redButton);
145. }
146.
147. public static final int DEFAULT_WIDTH = 300;
148. public static final int DEFAULT_HEIGHT = 200;
149.
150. private JPanel panel;
151. private JButton yellowButton;
152. private JButton blueButton;
153. private JButton redButton;
154. }
```

## 11.2 编译器API

在前面的小节中，你看到了如何与用脚本语言编写的代码进行交互。现在我们转向不同的场景：编译Java代码的Java程序。有许多工具都需要调用Java编译器，例如：

- 开发环境。
- Java教学和辅导程序。
- 自动化的构建和测试工具。
- 处理Java代码段的模板工具，例如JavaServer Pages (JSP)。

在过去，应用程序是通过在`jdk/lib/tools.jar`类库中未归档的类调用Java编译器的。从Java SE 6开始，用于编译的一个公共API成为了Java平台的一部分，并且它再也不需要使用`tools.jar`了。本节将解释这个编译器API。

### 11.2.1 编译便捷之法

调用编译器非常简单，下面是一个示范调用：

```
JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();
OutputStream outStream = ..., errStream = ...;
int result = compiler.run(null, outStream, errStream, "-sourcepath", "src", "Test.java");
```

返回值为0表示编译成功。

编译器会向提供给它的流发送输出和错误消息。如果将这些参数设置为`null`，就会使用`System.out`和`System.err`。`run`方法的第一个参数是输入流，由于编译器不会接受任何控制台输入，因此总是应该让其保持为`null`。（`run`方法是从泛化的`Tool`接口继承而来的，它考虑到某些工具需要读取输入。）

如果在命令行调用javac，那么`run`方法其余的参数就会传递给javac变量。这些变量是一些选项或文件名。

### 11.2.2 使用编译工具

可以通过使用`CompilationTask`对象来对编译过程进行更多的控制。特别是，你可以：

- 控制程序代码的来源，例如，在字符串构建器而不是文件中提供代码。
- 控制类文件的位置，例如，存储在数据库中。
- 监听在编译过程中产生的错误和警告信息。
- 在后台运行编译器。

源代码和类文件的位置是由`JavaFileManager`控制的，它负责确定源代码和类文件的`JavaFileObject`实例。`JavaFileObject`可以对应于磁盘文件，或者可以提供读写其内容的其他机制。

为了监听错误消息，需要安装一个`DiagnosticListener`。这个监听器在编译器报告警告或错误消息时会收到一个`Diagnostic`对象。`DiagnosticCollector`类实现了这个接口，它将收集所有的诊断信息，使得你可以在编译完成之后遍历这些信息。

`Diagnostic`对象包含有关问题位置的信息（包括文件名、行号和列号）以及人类可阅读的描述。

可以通过调用`JavaCompiler`类的`getTask`方法来获得`CompilationTask`对象。这时需要指定：

- 一个用于所有编译器输出的`Writer`，它不会将输出作为`Diagnostic`报告。如果是`null`，则使用`System.err`。
- 一个`JavaFileManager`，如果为`null`，则使用编译器的标准文件管理器。
- 一个`DiagnosticListener`。
- 选项字符串，如果没有选项，则为`null`。
- 用于注解处理的类文件，如果没有指定文件，则为`null`。（我们将在本章后面的内容中讨论注解处理。）
- 用于源文件的`JavaFileObject`实例。

需要为最后三个参数提供`Iterable`对象。例如，选项序列可以指定为：

```
Iterable<String> options = Arrays.asList("-g", "-d", "classes");
```

或者，可以使用任何集合类。

如果希望编译器从磁盘读去源文件，那么可以让`StandardJavaFileManager`将文件名字符串或`File`对象转译成`JavaObject`实例。例如，

```
StandardJavaFileManager fileManager = compiler.getStandardFileManager(null, null, null);
Iterable<JavaFileObject> fileObjects = fileManager.getJavaFileObjectsFromStrings(fileName);
```

但是，如果希望编译器从磁盘文件之外的其他地方读取源代码，那么可以提供自己的`JavaFileObject`子类。程序清单11-2展示了一种源文件对象的代码，这种对象的数据包含在一个`StringBuilder`中。这个类扩展自`SimpleJavaFileObject`便利类，并覆盖了`getCharContent`方法，让其返回字符串构建器中的内容。我们在示例程序中使用这个类来动态产生一个Java类的代码，然后编译了这些代码。

`CompilationTask`类实现了`Callable<Boolean>`接口，可以将其传递给一个`Executor`，使其可以在另一个线程中执行，或者可以直接调用`call`方法。返回值如果是`Boolean.FALSE`，则

表示调用失败。

```
Callable<Boolean> task = new JavaCompiler.CompilationTask(null, fileManager, diagnostics,
    options, null, fileObjects);
if (!task.call())
    System.out.println("Compilation failed");
```

如果只是想让编译器在磁盘上生成类文件，则不需要定制JavaFileManager。但是，我们的示范应用是将类文件生成在字节数组中，稍后会使用特殊的类加载器将其从内存中读出。程序清单11-3定义了一个实现了JavaFileObject接口的类，其openOutputStream方法将返回编译器在其中放置字节码的ByteArrayOutputStream。

事实证明，要告知编译器的文件管理器去使用这些文件对象还是比较棘手的，因为类库没有提供实现了StandardJavaFileManager接口。因此，我们需要子类化ForwardingJavaFileManager类，该类将所有的调用都代理给了给定的文件管理器。在我们所处的情况下，我们只想修改getJavaFileForOutput方法，我们通过下面的代码框架达到了这个目的：

```
JavaFileManager fileManager = compiler.getStandardFileManager(diagnostics, null, null);
fileManager = new ForwardingJavaFileManager<JavaFileManager>(fileManager)
{
    public JavaFileObject getJavaFileForOutput(Location location, final String className,
        Kind kind, FileObject sibling) throws IOException
    {
        return custom file object
    }
};
```

总之，如果只是想以常规的方式调用编译器，那就只需要调用JavaCompiler任务的run方法，去读写磁盘文件。可以捕获输出和错误消息，但是需要你自己去解析它们。

如果想对文件处理和错误报告进行更多的控制，可以使用CompilationTask类。它的API非常复杂，但是可以控制编译过程的各个方面。

### 程序清单11-2 StringBuilderJavaSource.java

```
1 import java.net.*;
2 import javax.tools.*;
3
4 /**
5 * A Java source that holds the code in a string builder.
6 * @version 1.00 2007-11-02
7 * @author Cay Horstmann
8 */
9 public class StringBuilderJavaSource extends SimpleJavaFileObject
10 {
11     /**
12      * Constructs a new StringBuilderJavaSource
13      * @param name the name of the source file represented by this file object
14      */
15     public StringBuilderJavaSource(String name)
16     {
17         super(URI.create("string://" + name.replace('.', '/') + Kind.SOURCE.extension),
18             Kind.SOURCE);
```

```
19.     code = new StringBuilder();
20. }
21.
22. public CharSequence getCharContent(boolean ignoreEncodingErrors)
23. {
24.     return code;
25. }
26.
27. public void append(String str)
28. {
29.     code.append(str);
30.     code.append('\n');
31. }
32.
33. private StringBuilder code;
34. }
```

### 程序清单11-3 ByteArrayJavaClass.java

```
1. import java.io.*;
2. import java.net.*;
3. import javax.tools.*;
4.
5. /**
6. * A Java class that holds the bytecodes in a byte array.
7. * @version 1.00 2007-11-02
8. * @author Cay Horstmann
9. */
10. public class ByteArrayJavaClass extends SimpleJavaFileObject
11. {
12.     /**
13.      * Constructs a new ByteArrayJavaClass
14.      * @param name the name of the class file represented by this file object
15.      */
16.     public ByteArrayJavaClass(String name)
17.     {
18.         super(URI.create("bytes://" + name), Kind.CLASS);
19.         stream = new ByteArrayOutputStream();
20.     }
21.
22.     public OutputStream openOutputStream() throws IOException
23.     {
24.         return stream;
25.     }
26.
27.     public byte[] getBytes()
28.     {
29.         return stream.toByteArray();
30.     }
31.
32.     private ByteArrayOutputStream stream;
33. }
```



### javax.tools.Tool 6

- int run(InputStream in, OutputStream out, OutputStream err, String...)

arguments)

用给定的输入、输出、错误流，以及给定的参数来运行工具。返回值为0表示成功，非0值表示失败。

#### API **javax.tools.JavaCompiler** 6

- `StandardJavaFileManager getStandardFileManager(DiagnosticListener<? super JavaFileObject> diagnosticListener, Locale locale, Charset charset)`

获取该编译器的标准文件管理器。对于默认的错误报告机制、locale和字符集等参数，可以提供null。

- `JavaCompiler.CompilationTask getTask(Writer out, JavaFileManager fileManager, DiagnosticListener<? super JavaFileObject> diagnosticListener, Iterable<String> options, Iterable<String> classesForAnnotationProcessing, Iterable<? extends JavaFileObject> sourceFiles)`

获取编译器任务，在被调用时，该任务将编译给定的源文件。参见前一节中有关这部分内容的详细讨论。

#### API **javax.tools.StandardJavaFileManager** 6

- `Iterable<? extends JavaFileObject> getJavaFileObjectsFromStrings(Iterable<String> fileNames)`
- `Iterable<? extends JavaFileObject> getJavaFileObjectsFromFiles(Iterable<? extends File> files)`

将文件名或文件序列转译成一个JavaFileObject实例序列。

#### API **javax.tools.JavaCompiler.CompilationTask** 6

- `Boolean call()`

执行编译任务。

#### API **javax.tools.DiagnosticCollector<S>** 6

- `DiagnosticCollector<S> DiagnosticCollector()`

构造一个空收集器。

- `List<Diagnostic<? extends S>> getDiagnostics()`

获取收集到的诊断信息。

#### API **javax.tools.Diagnostic<S>** 6

- `S getSource()`

获取与该诊断信息相关联的源对象。

- `Diagnostic.Kind getKind()`

获取该诊断信息的类型，返回值为ERROR, WARNING, MANDATORY\_WARNING, NOTE或OTHER之一。

- `String getMessage(Locale locale)`

获取一条消息，这条消息描述了由该诊断信息所揭示的问题。

- `long getLineNumber()`

- `long getColumnNumber()`

获取由该诊断信息所揭示的问题的位置。

#### **API** `javax.tools.SimpleJavaFileObject` 6

- `CharSequence getCharContent(boolean ignoreEncodingErrors)`

对于表示源文件的文件对象，需要覆盖该方法，并产生源代码。

- `OutputStream openOutputStream()`

对于表示类文件的文件对象，需要覆盖该方法，并产生一个字节码可以向其中写入的流。

#### **API** `javax.tools.ForwardingJavaFileManager<M extends JavaFileManager>` 6

- `protected ForwardingJavaFileManager(M fileManager)`

构造一个JavaFileManager，它将所有的调用都代理给给定的文件管理器。

- `FileObject getFileForOutput(JavaFileManager.Location location, String className, JavaFileObject.Kind kind, FileObject sibling)`

如果希望替换文件对象，从而可以写出类文件，则需要拦截该调用。`kind`的值是 `SOURCE`, `CLASS`, `HTML`或`OTHER`之一

### 11.2.3 一个示例：动态Java代码生成

在用于动态Web页面的JSP技术中，可以在HTML中混杂Java代码，例如：

```
<p>The current date and time is <b><%= new java.util.Date() %></b>.</p>
```

JSP引擎动态地将Java代码编译到Servlet中。在示范应用中，我们使用了一个更简单的示例，它可以动态生成Swing代码。其基本思想是使用GUI构建器在窗体中放置构件，并在一个外部文件中指定构件的行为。程序清单11-4展示了一个非常简单的窗体类实例，而程序清单11-5展示了按钮动作的代码。请注意，窗体类的构造器调用了抽象方法`addEventHandlers`。我们的代码生成器将产生一个实现了`addEventHandlers`方法的子类，并且对`action.properties`类的每一行都添加了动作监听器。（我们给读者留下了一个典型的练习，即扩展代码的生成功能，使其支持其他的事件类型。）

我们将这个子类置于名字为x的包中，因为我们不希望在程序的其他地方用到它。所生成的代码有如下形式：

```
package x;
public class Frame extends SuperclassName {
    protected void addEventHandlers() {
        componentName1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent) {
                code for event handler1
            }
        });
        // repeat for the other event handlers ...
    }
}
```

程序清单11-6的程序中的buildSource方法构建了这些代码，并将它们放到了StringBuilderJavaSource对象中。该对象会传递给Java编译器。

我们使用了一个ForwardingJavaFileManager对象，它具有getJavaFileForOutput方法，该方法将为x包中的每个类构造一个ByteArrayJavaClass对象，而这些对象会捕获x.Frame类被编译时所生成的类文件。该方法将每个文件对象都添加到了一个列表中，然后将其返回，以使得我们稍后可以定位这些字节码。请注意，编译x.Frame类会为主类生成一个类文件，并为每个监听器类生成一个类文件。

在编译之后，我们构建了一个映射表，它将类名与字节码数组关联在一起。（程序清单11-7所示的）一个简单的类加载器可以用来加载在这个映射表中存储的类。

我们让类加载器去加载刚刚编译过的类，然后构建并显示该应用的窗体类。

```
ClassLoader loader = new MapClassLoader(byteCodeMap);
Class<?> c1 = loader.loadClass("x.Frame");
Frame frame = (JFrame) c1.newInstance();
frame.setVisible(true);
```

当点击按钮时，背景色会按照常规方式进行修改。为了查看这些动作是动态编译的，需要更改action.properties文件中一行，例如，修改成下面这样：

```
yellowButton=panel.setBackground(java.awt.Color.YELLOW); yellowButton.setEnabled(false);
```

再次运行这个程序，现在，黄色按钮在点击之后就变得禁用了。再看看代码目录，你不会发现x包中的类的任何源文件和类文件。这个示例向你演示了如何通过内存中的源文件和类文件使用动态编译。

#### 程序清单11-4 ButtonFrame.java

```
1. package com.horstmann.corejava;
2. import javax.swing.*;
3.
4. /**
5. * @version 1.00 2007-11-02
6. * @author Cay Horstmann
7. */
8. public abstract class ButtonFrame extends JFrame
9. {
10.     public ButtonFrame()
11.     {
12.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
13.
14.         panel = new JPanel();
15.         add(panel);
16.
17.         yellowButton = new JButton("Yellow");
18.         blueButton = new JButton("Blue");
19.         redButton = new JButton("Red");
20.
21.         panel.add(yellowButton);
22.         panel.add(blueButton);
23.         panel.add(redButton);
24.
25.         addEventHandlers();
```

```
26     }
27
28     protected abstract void addEventHandlers();
29
30     public static final int DEFAULT_WIDTH = 300;
31     public static final int DEFAULT_HEIGHT = 200;
32
33     protected JPanel panel;
34     protected JButton yellowButton;
35     protected JButton blueButton;
36     protected JButton redButton;
37 }
```

**程序清单11-5 action.properties**

```
1. yellowButton=panel.setBackground(java.awt.Color.YELLOW);
2. blueButton=panel.setBackground(java.awt.Color.BLUE);
3. redButton=panel.setBackground(java.awt.Color.RED);
```

**程序清单11-6 CompilerTest.java**

```
1. import java.awt.*;
2. import java.io.*;
3. import java.util.*;
4. import java.util.List;
5. import javax.swing.*;
6. import javax.tools.*;
7. import javax.tools.JavaFileObject.*;
8
9. /**
10. * @version 1.00 2007-11-02
11. * @author Cay Horstmann
12. */
13 public class CompilerTest
14 {
15     public static void main(final String[] args) throws IOException
16     {
17         JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();
18
19         final List<ByteArrayJavaClass> classFileObjects = new ArrayList<ByteArrayJavaClass>();
20
21         DiagnosticCollector<JavaFileObject> diagnostics = new DiagnosticCollector<JavaFileObject>()
22
23         JavaFileManager fileManager = compiler.getStandardFileManager(diagnostics, null, null);
24         fileManager = new ForwardingJavaFileManager<JavaFileManager>(fileManager)
25         {
26             public JavaFileObject getJavaFileForOutput(Location location,
27                 final String className, Kind kind, FileObject sibling) throws IOException
28             {
29                 if (className.startsWith("x."))
30                 {
31                     ByteArrayJavaClass fileObject = new ByteArrayJavaClass(className);
32                     classFileObjects.add(fileObject);
33                     return fileObject;
34                 }
35                 else return super.getJavaFileForOutput(location, className, kind, sibling);
36             }
37         };
38     }
39 }
```

```
36.         }
37.     );
38.
39.     JavaFileObject source = buildSource("com.horstmann.corejava.ButtonFrame");
40.     JavaCompiler.CompilationTask task = compiler.getTask(null, fileManager, diagnostics,
41.             null, null, Arrays.asList(source));
42.     Boolean result = task.call();
43.
44.     for (Diagnostic<? extends JavaFileObject> d : diagnostics.getDiagnostics())
45.         System.out.println(d.getKind() + ": " + d.getMessage(null));
46.     fileManager.close();
47.     if (!result)
48.     {
49.         System.out.println("Compilation failed.");
50.         System.exit(1);
51.     }
52.
53.     EventQueue.invokeLater(new Runnable()
54.     {
55.         public void run()
56.         {
57.             try
58.             {
59.                 Map<String, byte[]> byteCodeMap = new HashMap<String, byte[]>();
60.                 for (ByteArrayJavaClass cl : classFileObjects)
61.                     byteCodeMap.put(cl.getName().substring(1), cl.getBytes());
62.                 ClassLoader loader = new MapClassLoader(byteCodeMap);
63.                 Class<?> cl = loader.loadClass("x.Frame");
64.                 JFrame frame = (JFrame) cl.newInstance();
65.                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
66.                 frame.setTitle("CompilerTest");
67.                 frame.setVisible(true);
68.             }
69.             catch (Exception ex)
70.             {
71.                 ex.printStackTrace();
72.             }
73.         }
74.     });
75. }
76.
77. /**
78. * Builds the source for the subclass that implements the addEventHandlers method.
79. * @return a file object containing the source in a string builder
80. */
81. static JavaFileObject buildSource(String superclassName) throws IOException
82. {
83.     StringBuilderJavaSource source = new StringBuilderJavaSource("x.Frame");
84.     source.append("package x;\n");
85.     source.append("public class Frame extends " + superclassName + " {\n");
86.     source.append("protected void addEventHandlers() {\n");
87.     Properties props = new Properties();
88.     props.load(new FileReader("action.properties"));
89.     for (Map.Entry<Object, Object> e : props.entrySet())
90.     {
91.         String beanName = (String) e.getKey();
92.         String eventCode = (String) e.getValue();
```

```
93.         source.append(beanName + ".addActionListener(new java.awt.event.ActionListener() {");
94.         source.append("public void actionPerformed(java.awt.event.ActionEvent event) {" );
95.         source.append(eventCode);
96.         source.append("} } );");
97.     }
98.     source.append("} }");
99.     return source;
100. }
101. }
```

### 程序清单11-7 MapClassLoader.java

```
1. import java.util.*;
2.
3. /**
4. * A class loader that loads classes from a map whose keys are class names and whose
5. * values are byte code arrays.
6. * @version 1.00 2007-11-02
7. * @author Cay Horstmann
8. */
9. public class MapClassLoader extends ClassLoader
10. {
11.     public MapClassLoader(Map<String, byte[]> classes)
12.     {
13.         this.classes = classes;
14.     }
15.
16.     protected Class<?> findClass(String name) throws ClassNotFoundException
17.     {
18.         byte[] classBytes = classes.get(name);
19.         if (classBytes == null) throw new ClassNotFoundException(name);
20.         Class<?> c1 = defineClass(name, classBytes, 0, classBytes.length);
21.         if (c1 == null) throw new ClassNotFoundException(name);
22.         return c1;
23.     }
24.
25.     private Map<String, byte[]> classes;
26. }
```

## 11.3 使用注解

注解是那些插入到源代码中用于某种工具处理的标签。这些标签可以在源码层次上进行操作，或者可以处理编译器将它们纳入到注解类文件中。

注解不会改变对编写的程序的编译方式。Java编译器对于包含注解和不包含注解的代码会生成相同的虚拟机指令。

为了能够受益于注解，你需要选择一个处理工具，然后向你的处理工具可以理解的代码中插入注解，之后运用该处理工具。

注解的使用范围还是很广泛的，并且这种广泛性最初显得杂乱无章。下面是关于注解的一些可能的用法：

- 附属文件的自动生成，例如部署描述符或者bean信息类。
- 测试、日志、事务语义等代码的自动生成。

我们首先介绍基本概念，然后将这些概念运用到一个具体示例中：我们将某些方法标注为AWT构件的事件监听器，然后向你显示一个能够分析注解和连接监听器的注解处理器。然后，我们对其语法规则进行详细讨论。最后我们以两个注解处理的高级示例结束本章。其中一个可以处理源代码级别的注解。另外一个使用了Apache的字节码工程类库，可以向注解过的方法中添加额外的字节码。

下面是一个简单注解的示例：

```
public class MyClass
{
    ...
    @Test public void checkRandomInsertions()
}
```

注解@**Test**用于注解**checkRandomInsertions**方法。

在Java中，注解是当作一个修饰符来使用的，它被置于被注解项之前，中间没有分号。（修饰符就是诸如**public**和**static**之类的关键词。）每一个注解的名称前面都加上了@符号，这有点类似于Javadoc的注解。然而，Javadoc注解出现在/\*\*...\*/定界符的内部，而注解是代码的一部分。

@**Test**注解自身并不会作任何事情，它需要工具支持才会有用。例如，当测试一个类的时候，JUnit4测试工具（可以从<http://junit.org>处获得）可能会调用所有标识为@**Test**的方法。另一个工具可能会删除一个类文件中的所有测试方法，以便在对这个类测试完毕后，不会将这些测试方法与程序装载在一起。

注解可以定义成包含元素的形式，例如：

```
@Test(timeout="10000")
```

这些元素可以被阅读这些注解的工具去处理。其他形式的元素也是有可能的；我们将会在本章的随后部分进行讨论。

除了方法外，还可以注解类、成员以及本地变量，这些注解可以存在于任何可以放置一个像**public**或者**static**这样的修饰符的地方。

每个注解都必须通过一个注解接口进行定义。这些接口中的方法与注解中的元素相对应。例如，JUnit的注解**TestCase**可以用下面这个接口进行定义：

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Test
{
    long timeout() default 0L;
    ...
}
```

@**interface**声明创建了一个真正的Java接口。处理注解的工具将接收那些实现了这个注解接口的对象。一个工具可以调用**timeout**方法来检索某个特定**Test**注解的**timeout**元素。

注解**Target**和**Retention**是元注解。它们注解了**Test**注解，即将**Test**注解标识成一个只能运用到方法上的注解，并且当类文件载入到虚拟机的时候，仍可以保留下。我们将会在11.5.3详细讨论这些元注解。

你现在已经清楚了程序的元数据和注解这两个概念。在接下来的小节中，我们将深入到一

个注解处理的具体示例中继续探讨。

### 一个示例：注解事件处理器

在用户界面编程中，一件更令人讨厌的事情就是包装事件源上的监听器。很多监听器是下面这种形式的：

```
myButton.addActionListener(new
    ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        doSomething();
    }
});
```

在本节，我们设计了一个注解来免除这种苦差事。该注解的形式如下：

```
@ActionListenerFor(source="myButton") void doSomething() { . . . }
```

程序员不再被迫去调用`addActionListener`。相反地，每个方法直接用一个注解标记起来。程序清单11-8显示了第1卷第8章的`ButtonTest`程序，只不过这里使用上述这类注解重新实现了一遍。

我们还需要定义一个注解接口，代码在程序清单11-9中。

程序清单11-8 `ButtonFrame.java`

```
1. import java.awt.*;
2. import javax.swing.*;
3.
4. /**
5. * A frame with a button panel
6. * @version 1.00 2004-08-17
7. * @author Cay Horstmann
8. */
9. public class ButtonFrame extends JFrame
10. {
11.     public ButtonFrame()
12.     {
13.         setTitle("ButtonTest");
14.         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
15.
16.         panel = new JPanel();
17.         add(panel);
18.
19.         yellowButton = new JButton("Yellow");
20.         blueButton = new JButton("Blue");
21.         redButton = new JButton("Red");
22.
23.         panel.add(yellowButton);
24.         panel.add(blueButton);
25.         panel.add(redButton);
26.
27.         ActionListenerInstaller.processAnnotations(this);
28.     }
29.
```

```

30.     @ActionListenerFor(source = "yellowButton")
31.     public void yellowBackground()
32.     {
33.         panel.setBackground(Color.YELLOW);
34.     }
35.
36.     @ActionListenerFor(source = "blueButton")
37.     public void blueBackground()
38.     {
39.         panel.setBackground(Color.BLUE);
40.     }
41.
42.     @ActionListenerFor(source = "redButton")
43.     public void redBackground()
44.     {
45.         panel.setBackground(Color.RED);
46.     }
47.
48.     public static final int DEFAULT_WIDTH = 300;
49.     public static final int DEFAULT_HEIGHT = 200;
50.
51.     private JPanel panel;
52.     private JButton yellowButton;
53.     private JButton blueButton;
54.     private JButton redButton;
55. }
```

### 程序清单11-9 ActionListenerFor.java

```

1. import java.lang.annotation.*;
2.
3. /**
4. * @version 1.00 2004-08-17
5. * @author Cay Horstmann
6. */
7.
8. @Target(ElementType.METHOD)
9. @Retention(RetentionPolicy.RUNTIME)
10. public @interface ActionListenerFor
11. {
12.     String source();
13. }
```

当然，这些注解本身不会做任何事情。它们只是存在于源文件中。编译器将它们置于类文件中，并且虚拟机会将它们载入。我们现在需要的是一个分析注解以及安装行为监听器的机制。这也是类ActionListenerInstaller的职责所在。ButtonFrame构造器将调用下面的方法：

```
ActionListenerInstaller.processAnnotations(this);
```

静态的processAnnotations方法可以枚举出某个对象接收到的所有方法。对于每一个方法，它先获取ActionListenerFor注解对象，然后再对它进行处理。

```

Class<?> c1 = obj.getClass();
for (Method m : c1.getDeclaredMethods())
{
    ActionListenerFor a = m.getAnnotation(ActionListenerFor.class);
```

```

if (a != null) . . .
}

```

这里，我们使用了定义在AnnotatedElement接口中的getAnnotation方法。Method、Constructor、Field、Class和Package这些类都实现了这个接口。

源文件名是存储在注解对象中的。我们可以通过调用source方法对它进行检索，然后查找匹配的成员域。

```

String fieldName = a.source();
Field f = cl.getDeclaredField(fieldName);

```

这表明我们的注解有点局限。源元素必须是一个成员域的名字，而不能是本地变量。

代码的剩余部分相当具有技术性。对于每一个被注解的方法，我们构造了一个实现了ActionListener接口的代理对象，其actionPerformed方法将调用这个被注解过的方法。（关于代理的更多信息见第1卷第6章。）细节并不重要。关键要知道注解的功能是通过processAnnotations方法建立起来的。

图11-1展示了在本例中注解是如何被处理的。

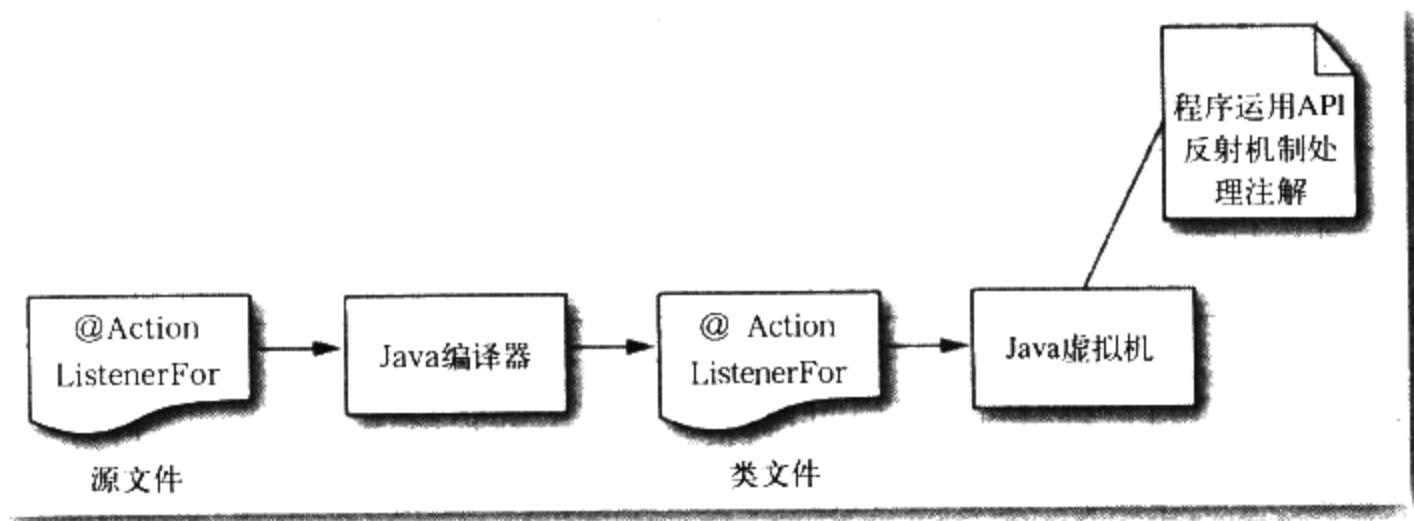


图11-1 在运行时处理注解

在这个示例中，注解是在运行时进行处理的。另外也有可能在源码级别上对它们进行处理。源代码生成器可能已经产生了用于添加监听器的代码。另外，注解可能已经在字节码级别上进行过处理。字节码编辑器可能已经将addActionListener调用置入框体构造器中了。听起来似乎很复杂，不过可以利用一些类库相对直截了当地实现这项任务。

对于用户界面程序员来说，我们这个示例并不能看作是一个很严格的工具。因为，用于添加监听器的实用方法就像添加一条注解那样方便。（实际上，java.beans.EventHandler类试图实现的就是这些。通过在这个类中提供一个可以添加事件处理器的方法，而不仅仅只是构建它，就可以很容易地对它进行改进。）

不过，这个示例展示了对一个程序进行注解以及对这些注解进行分析的机制。既然你已经领会了这个具体示例，那么，现在可能已经为遵循完备详述的注解语法条款做好了更充分的准备（这也是我们所希望的）。

**程序清单11-10 ActionListenerInstaller.java**

```
1. import java.awt.event.*;
2. import java.lang.reflect.*;
3.
4. /**
5. * @version 1.00 2004-08-17
6. * @author Cay Horstmann
7. */
8. public class ActionListenerInstaller
9. {
10.    /**
11.     * Processes all ActionListenerFor annotations in the given object.
12.     * @param obj an object whose methods may have ActionListenerFor annotations
13.     */
14.    public static void processAnnotations(Object obj)
15.    {
16.        try
17.        {
18.            Class<?> c1 = obj.getClass();
19.            for (Method m : c1.getDeclaredMethods())
20.            {
21.                ActionListenerFor a = m.getAnnotation(ActionListenerFor.class);
22.                if (a != null)
23.                {
24.                    Field f = c1.getDeclaredField(a.source());
25.                    f.setAccessible(true);
26.                    addListener(f.get(obj), obj, m);
27.                }
28.            }
29.        }
30.        catch (Exception e)
31.        {
32.            e.printStackTrace();
33.        }
34.    }
35.
36.    /**
37.     * Adds an action listener that calls a given method.
38.     * @param source the event source to which an action listener is added
39.     * @param param the implicit parameter of the method that the listener calls
40.     * @param m the method that the listener calls
41.     */
42.    public static void addListener(Object source, final Object param, final Method m)
43.        throws NoSuchMethodException, IllegalAccessException, InvocationTargetException
44.    {
45.        InvocationHandler handler = new InvocationHandler()
46.        {
47.            public Object invoke(Object proxy, Method mm, Object[] args) throws Throwable
48.            {
49.                return m.invoke(param);
50.            }
51.        };
52.
53.        Object listener = Proxy.newProxyInstance(null,
54.            new Class[] { java.awt.event.ActionListener.class }, handler);
55.        Method adder = source.getClass().getMethod("addActionListener", ActionListener.class);
```

```
56     adder.invoke(source, listener);
57 }
58 }
```

### API `java.lang.AnnotatedElement 5.0`

- `boolean isAnnotationPresent(Class<? extends Annotation> annotationType)`  
如果该项具有给定类型的注解，则返回true。
- `<T extends Annotation> T getAnnotation(Class<T> annotationType)`  
获得给定类型的注解，如果该项不具有这样的注解，则返回null。
- `Annotation[] getAnnotations()`  
获得用于表示该项的所有注解，包括继承而来的注解。如果没有出现任何注解，那么将返回一个长度为0的数组。
- `Annotation[] getDeclaredAnnotations()`  
获得声明该项的所有注解，不包含继承而来的注解。如果没有出现任何注解，那么将返回一个长度为0的数组。

## 11.4 注解语法

在本小节，我们将介绍你必须了解的每一个注解语法。

一个注解是由一个注解接口来定义的：

```
modifiers @interface AnnotationName
{
    element declaration1
    element declaration2
    ...
}
```

每个元素声明具有下面这种形式：

```
type elementName();
```

或者

```
type elementName() default value;
```

举例来说，下面这个注解具有两个元素：`assignedTo`和`severity`。

```
public @interface BugReport
{
    String assignedTo() default "[none]";
    int severity() = 0;
}
```

每个注解都具有下面这种格式：

```
@AnnotationName(elementName1=value1, elementName2=value2, ...)
```

举例来说，

```
@BugReport(assignedTo="Harry", severity=10)
```

元素的顺序无关紧要。下面这个注解和前面那个一样。

```
@BugReport(severity=10, assignedTo="Harry")
```

如果某个元素的值并未指定，那么就使用声明的默认值。例如，考虑一下下面这个注解：

```
@BugReport(severity=10)
```

元素`assignedTo`的值是字符串"`[none]`"。

 **警告：**默认值并不是和注解存储在一起的；相反地，它们是动态计算而来的。例如，如果你将元素`assignedTo`的默认值更改为"`[]`"，然后重新编译`BugReport`接口，那么注解`@BugReport(severity=10)`将使用这个新的默认值，甚至在那些在默认值修改之前就已经编译过的类文件中也是如此。

有两个特殊的快捷方式可以用来简化注解。

如果没有指定元素，要么是因为注解中没有任何元素，要么是因为所有元素都使用默认值，那么你就不需要使用圆括号了。例如，

```
@BugReport
```

和下面这个注解是一样的

```
@BugReport(assignedTo="[none]", severity=0)
```

这样的注解又称为标记注解。

另外一种快捷方式是单值注解。如果一个元素具有特殊的名字`value`，并且没有指定其他元素，那么你就可以忽略掉这个元素名以及等号。例如，既然我们已经在前面将`ActionListenerFor`注解接口定义为如下形式：

```
public @interface ActionListenerFor
{
    String value();
}
```

那么，我们可以将这个注解书写成如下形式：

```
@ActionListenerFor("yellowButton")
```

而不是

```
@ActionListenerFor(value="yellowButton")
```

所有的注解接口隐式地扩展自`java.lang.annotation.Annotation`接口。这个接口是一个正规接口，不是一个注解接口。请查看本章最后为该接口提供的一些方法所做的API注解。

你无法扩展注解接口。换句话说，所有的注解接口都直接扩展自`java.lang.annotation.Annotation`。

你从来不用提供那些实现了注解接口的类。相反地，虚拟机会在需要的时候产生一些代理类及对象。例如，当请求一个`ActionListenerFor`注解的时候，虚拟机实现了一个和下面相似的操作：

```
return Proxy.newProxyInstance(classLoader, ActionListenerFor.class,
    new
        InvocationHandler()
    {
        public Object invoke(Object proxy, Method m, Object[] args) throws Throwable
        {
```

```
    if (m.getName().equals("source")) return value of source annotation;  
    . . .  
});
```

注解接口中的元素声明实际上是方法声明。一个注解接口的方法可以没有任何参数，没有任何throws语句，并且它们也不能是泛型的。

注解元素的类型为下列之一：

- 一个基本类型 (int、short、long、byte、char、double、float或者boolean)。
- 一个String。
- 一个Class (具有一个可供选择的类型参数，例如Class<? extends MyClass>)。
- 一个enum类型。
- 一个注解类型。
- 一个由前面所述类型组成的数组 (由数组组成的数组不是合法的元素类型)。

下面是一些合法的元素声明的例子：

```
public @interface BugReport  
{  
    enum Status { UNCONFIRMED, CONFIRMED, FIXED, NOTABUG };  
    boolean showStopper() default false;  
    String assignedTo() default "[none]";  
    Class<?> testCase() default Void.class;  
    Status status() default Status.UNCONFIRMED;  
    Reference ref() default @Reference(); // an annotation type  
    String[] reportedBy();  
}
```

因为注解是由编译器计算而来的，因此，所有元素值必须是编译期常量。例如，

```
@BugReport(showStopper=true, assignedTo="Harry", testCase=MyTestCase.class,  
           status=BugReport.Status.CONFIRMED, . . .)
```

 **警告：**一个注解元素从来不能设置为null，甚至不允许其默认值为null。这样在实际应用中会相当不方便。你必须使用其他的默认值，例如""或者Void.class。

如果元素值是一个数组，那么要将它的值用括号括起来，像下面这样：

```
@BugReport(. . ., reportedBy={"Harry", "Carl"})
```

如果该元素具有单值，那么可以忽略这些括号：

```
@BugReport(. . ., reportedBy="Joe") // OK, same as {"Joe"}
```

既然一个注解元素可以是另一个注解，那么就可以创建出任意复杂的注解。例如，

```
@BugReport(ref=@Reference(id="3352627"), . . .)
```

 **注意：**在注解中引入循环依赖是一种错误。例如，因为BugReport具有一个注解类型为TestCase的元素，所以TestCase就不能再拥有一个类型为BugReport的元素。

可以向注解中添加如下一些项：

- 包
- 类 (包括enum) . . .

- 接口（包括注解接口）
- 方法
- 构造器
- 实例成员（包含enum常量）
- 本地变量
- 参数变量

不过，对本地变量的注解只能在源码级别上进行处理。类文件无法描述本地变量。因此，所有的本地变量注解在编译完一个类的时候会被遗弃掉。同样地，对包的注解不能在源码级别之外存在。

 **注意：**注解在文件package-info.java中，只包含包的声明，在这个文件中可以对包进行注解，这需要以注解为文件的开始。

一个项可以具有多个注解，只要它们属于不同的类型即可。当注解一个特定项的时候，不能多次使用同一个注解类型。例如，

```
@BugReport(showStopper=true, reportedBy="Joe")
@BugReport(reportedBy={"Harry", "Carl"})
void myMethod()
```

就是一种编译期错误。如果你认为这是一个问题，那么可以设计一个注解，它的值是一个由更简单的注解组成的数组：

```
@BugReports([
    @BugReport(showStopper=true, reportedBy="Joe"),
    @BugReport(reportedBy={"Harry", "Carl"}))
void myMethod()
```



### java.lang.annotation.Annotation 5.0

- Class<? extends Annotation> annotationType()

返回Class对象，它用于描述该注解对象的注解接口。注意：调用注解对象上的getClass方法可以返回真正的类，而不是接口。

- boolean equals(Object other)

如果other是一个作为该注解对象来实现同一注解接口的对象，并且如果该对象和other的所有元素彼此相等。那么返回True。

- int hashCode()

返回一个与equals方法兼容、由注解接口名以及元素值衍生而来的散列码。

- String toString()

返回一个包含注解接口名以及元素值的字符串表示，例如，@BugReport(assignedTo=[none], severity=0)。

## 11.5 标准注解

Java SE在java.lang、java.lang.annotation和javax.annotation包中定义了大量的注

解接口。其中四个是元注解，用于描述注解接口的行为属性，其他的三个是规则接口，可以用它们来注解你的源代码中的项。表11-2列出了这些注解。我们将会在随后的两个小节中给予详细介绍。

表11-2 标准注解

| 注解接口             | 应用场合          | 目的  |
|------------------|---------------|---|
| Deprecated       | 全部            | 将项标记为过时的                                  |
| SuppressWarnings | 除了包和注解之外的所有情况 | 阻止某个给定类型的警告信息                             |
| Override         | 方法            | 检查该方法是否覆盖了某一个超类方法                         |
| PostConstruct    | 方法            | 被标记的方法应该在构造之后或移除之前立即被调用                   |
| PreDestroy       |               |   |
| Resource         | 类、接口、方法、域     | 在类或接口上：标记为在其他地方要用到的资源。<br>在方法或域上：为“注入”而标记 |
| Resources        | 类、接口          | 一个资源数组                                    |
| Generated        | 全部            |   |
| Target           | 注解            | 指定可以应用这个注解的那些项                            |
| Retention        | 注解            | 指定这个注解可以保留多久                              |
| Documented       | 注解            | 指定这个注解应该包含在注解项的文档中                        |
| Inherited        | 注解            | 指定一个注解，当将它应用于一个类的时候，能够自动被它的子类继承           |

### 11.5.1 用于编译的注解

@Deprecated注解可以被添加到任何不再鼓励使用的项上。所以，当你使用一个已过时的项时，编译器将会发出警告。这个注解与Java文档标签@deprecated具有同等功效。

@SuppressWarnings注解会告知编译器阻止特殊类型的警告信息，例如，

```
@SuppressWarnings("unchecked")
```

@Override这种注解只能应用到方法上。编译器会阻止具有这种注解的方法去覆盖一个来自于超类的方法。例如，如果你声明：

```
public MyClass
{
    @Override public boolean equals(MyClass other);
    ...
}
```

那么编译器会报告一个错误。毕竟，equals方法不能覆盖Object类的equals方法。因为那个方法有一个类型为Object而不是MyClass的参数。

@Generated注解的目的是供代码生成工具来使用。任何生成的源代码都可以被注解，从而与程序员提供的代码区分开。例如，代码编辑器可以隐藏生成的代码，或者代码生成器可以移除生成代码的旧版本。每个注解都必须包含一个表示代码生成器的唯一标识符。日期字符串（ISO8601格式）和注释字符串是可选的。例如，

```
@Generated("com.horstmann.beanproperty", "2008-01-04T12:08:56.235-0700");
```

### 11.5.2 用于管理资源的注解

`@PostConstruct`和`@PreDestroy`注解用于控制对象生命周期的环境中，例如Web容器和应用服务器。标记了这些注解的方法应该在对象被构建之后，或者在对象被移除之前，紧接着调用。

`@Resource`注解用于资源注入。例如，考虑一下访问数据库的Web应用。当然，数据库访问信息不应该被硬编码到Web应用中。而是应该让Web容器提供某种用户接口，以便设置连接参数和数据库资源的JNDI名字。在这个Web应用中，可以向下面这样引用数据源：

```
@Resource(name="jdbc/mydb")
private DataSource source;
```

当包含这个域的对象被构造时，容器会“注入”一个对该数据源的引用。

### 11.5.3 元注解

`@Target`元注解可以应用于一个注解，以限制该注解可以应用到哪些项上。例如，

```
@Target({ElementType.TYPE, ElementType.METHOD})
public @interface BugReport
```

表11-3显示了所有可能的取值情况。它们属于枚举类型`ElementType`。可以指定任意数量的元素类型，用括号括起来。

表11-3 `@Target`注解的元素类型

| 元素类型            | 注解适用场合                   | 元素类型           | 注解适用场合        |
|-----------------|--------------------------|----------------|---------------|
| ANNOTATION_TYPE | 注解类型声明                   | CONSTRUCTOR    | 构造器           |
| PACKAGE         | 包                        | FIELD          | 成员域（包括enum常量） |
| TYPE            | 类（包括enum）及接口<br>（包括注解类型） | PARAMETER      | 方法或构造器参数      |
| METHOD          | 方法                       | LOCAL_VARIABLE | 本地变量          |

一条没有`@Target`限制的注解可以应用于任何项上。编译器只检查你是否将一条注解应用到了某个允许的项上。例如，如果将`@BugReport`应用于一个成员域上，则会导致一个编译器错误。

`@Retention`元注解用于指定一条注解应该保留多长时间。只能将其指定为表11-4中的任意值，其默认值是`RetentionPolicy.CLASS`。

表11-4 用于`@Retention`注解的保留策略

| 保留规则    | 描述                           |
|---------|------------------------------|
| SOURCE  | 不包括在类文件中的注解                  |
| CLASS   | 类文件中的注解，但是虚拟机不需要将它们载入        |
| RUNTIME | 类文件中的注解，并由虚拟机载入。通过反射API可获得它们 |

在程序清单11-9中，`@ActionListenerFor`注解声明为具有`RetentionPolicy.RUNTIME`，因为我们是使用反射机制进行注解处理的。在随后的两个小节里，你将会看到一些在源码级别和类文件级别上怎样对注解进行处理的示例。

`@Documented`元注解为像Javadoc这样的文档工具提供了一些提示。文档化的注解应该按照其他一些像`protected`或`static`这样用于文档目的修饰符来处理。其他注解的使用不属于文档范畴。例如，假定我们将`@ActionListenerFor`作为一个文档化注解来声明：

```
@Documented
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface ActionListenerFor
```

现在每一个已注解方法的文档都含有这条注解，如图11-2所示。

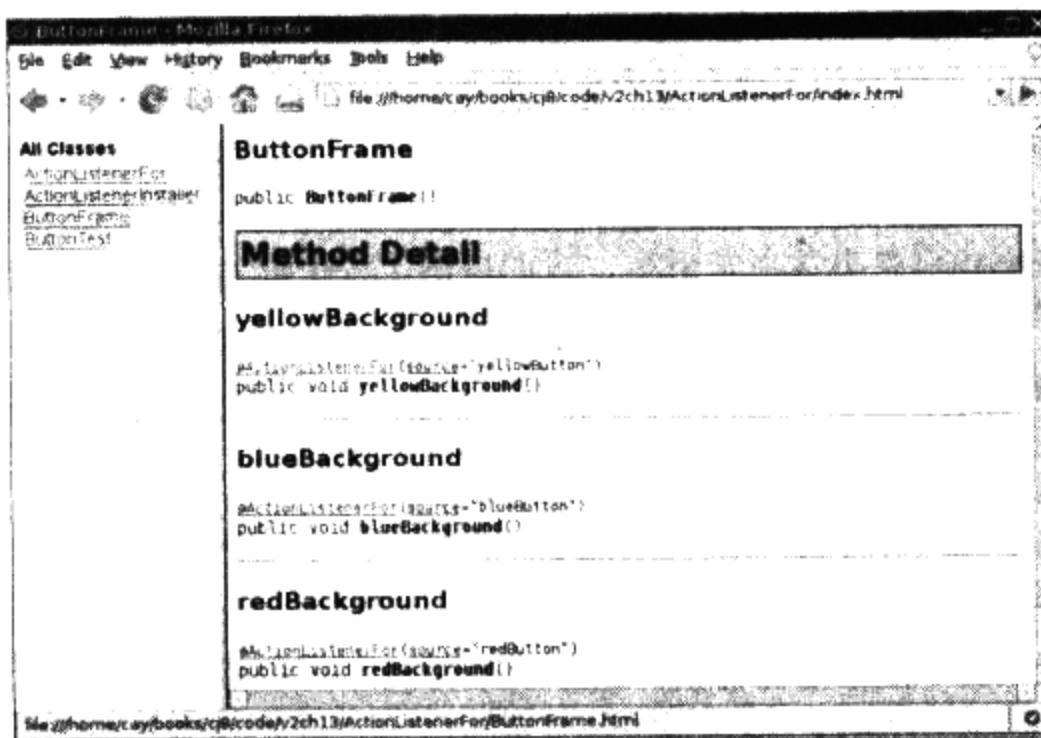


图11-2 文档化注解

如果某个注解是暂时的（例如`@BugReport`），那么就不应该对它们的用法进行归档。

**注意：**将一个注解应用到它自身上是合法的。例如，`@Documented`注解被它自身视为`@Documented`。因此，用于注解的Javadoc文档表明了它们是否可以文档化。

`@Inherited`元注解只能应用于对类的注解。如果一个类具有继承注解，那么它的所有子类都自动具有同样的注解。这使得创建一个与`Serializable`这样的接口修饰符具有同样的运行方式的注解变得很容易。

实际上，`@Serializable`注解应该比没有任何方法的`Serializable`修饰符接口更恰当。一个类之所以可以被序列化，是因为存在着对它的成员域进行读写的运行期支持，而不是因为任何面向对象的设计原理。注解比接口继承更擅长描述这一事实。当然了，可序列化接口是在JDK1.1中产生的，远比注解出现得早。

假设定义了一个继承注解`@Persistent`来指明一个类的对象可以存储到数据库中。那么该持久类的子类就会自动被注解为是持久性的。

```
@Inherited @Persistent { }
@Persistent class Employee { . . . }
class Manager extends Employee { . . . } // also @Persistent
```

在持久性机制去查找存储在数据库中的对象的时候，它就会探测到`Employee`对象以及

Manager对象的存在。

## 11.6 源码级注解处理

注解的用处之一就是自动生成包含程序额外信息的“附文件”。在过去，Java的“企业版本”由于让程序员过分讲究样板代码而变得臭名昭著。Java EE 5使用注解极大地简化了编程模型。

在本节，我们用一个比较简单的示例来演示这项技术。我们编写了一个程序，可以自动产生bean信息类。可以使用一个注解来标记bean的属性，然后运行某个工具对这个源文件进行解析，分析其注解，最后输出bean信息类的源文件。

回忆一下第8章，我们介绍了比自动内省处理更能精确描述bean的bean信息类。bean信息类列举了bean的所有属性。这些属性可以具有可选属性编辑器，第8章的ChartBeanBeanInfo类就是一个典型示例。

为了避免编写bean信息类这项苦差事，我们提供了一个@Property注解。可以标记属性的获取器或设置器，像下面这样：

```
@Property String getTitle() { return title; }
```

或者

```
@Property(editor="TitlePositionEditor")
public void setTitlePosition(int p) { titlePosition = p; }
```

程序清单11-11包含了对@Property注解的定义。注意这个注解具有SOURCE保留级别。我们只能在源码级别上对这个注解进行分析。由于它未被包含在类文件中，因此在反射期间无法访问到它。

### 程序清单11-11 Property.java

```
1. package com.horstmann.annotations;
2. import java.lang.annotation.*;
3.
4. @Documented
5. @Target(ElementType.METHOD)
6. @Retention(RetentionPolicy.SOURCE)
7. public @interface Property
8. {
9.     String editor() default "";
10. }
```



**注意：**将editor元素声明为Class类型是具有实际意义的。不过，注解处理器不能获取Class类型的注解，因为一个类的含义要依赖于外部因素（例如类路径或者类加载器）。因此，我们使用一个字符串来指定这个编辑器的类名。

为了自动产生一个名字为*BeanClass*的bean信息类。我们要实现下面这些任务：

- 1) 编写一个源文件*BeanClassBeanInfo.java*。将*BeanClassBeanInfo.java*声明为继承自*SimpleBeanInfo*，并覆盖*getPropertyDescriptors*方法。
- 2) 对于每个已注解的方法，通过去除掉get或set前缀，然后“小写化”剩余部分，就可以恢复属性名。

- 3) 对于每个属性，编写一条用于构建PropertyDescriptor的语句。
- 4) 如果该属性具有一个编辑器，那么编写一个方法去调用setPropertyEditorClass。
- 5) 编写代码返回一个包含所有属性描述符的数组。

举例来说，对于下面这个在ChartBean类中的注解

```
@Property(editor="TitlePositionEditor")
public void setTitlePosition(int p) { titlePosition = p; }
```

将被转换成

```
public class ChartBeanBeanInfo extends java.beans.SimpleBeanInfo
{
    public java.beans.PropertyDescriptor[] getProperties()
    {
        java.beans.PropertyDescriptor titlePositionDescriptor
            = new java.beans.PropertyDescriptor("titlePosition", ChartBean.class);
        titlePositionDescriptor.setPropertyEditorClass(TitlePositionEditor.class)

        return new java.beans.PropertyDescriptor[]
        {
            titlePositionDescriptor,
            ...
        }
    }
}
```

(样板代码显示为更亮的灰色。)

如果我们可以定位所有已经用@Property属性标记过的方法，那么所有这些都很容易实现。

从Java SE6开始，我们可以将注解处理器添加到Java编译器中。(在Java SE 5中，有一个称为apt的单独的工具被用于相同的目的。)为了调用注解处理机制，需要运行

```
javac -processor ProcessorClassName1,ProcessorClassName2,... sourceFiles
```

编译器会定位源代码中的注解，然后选择恰当的注解处理器。每个注解处理器会依次执行。如果某个注解处理器创建了一个新的源文件，那么将重复执行这个处理过程。如果某次处理循环没有再产生任何新的源文件，那么就编译所有的源文件。图11-3展示了@Property注解是如何处理的。

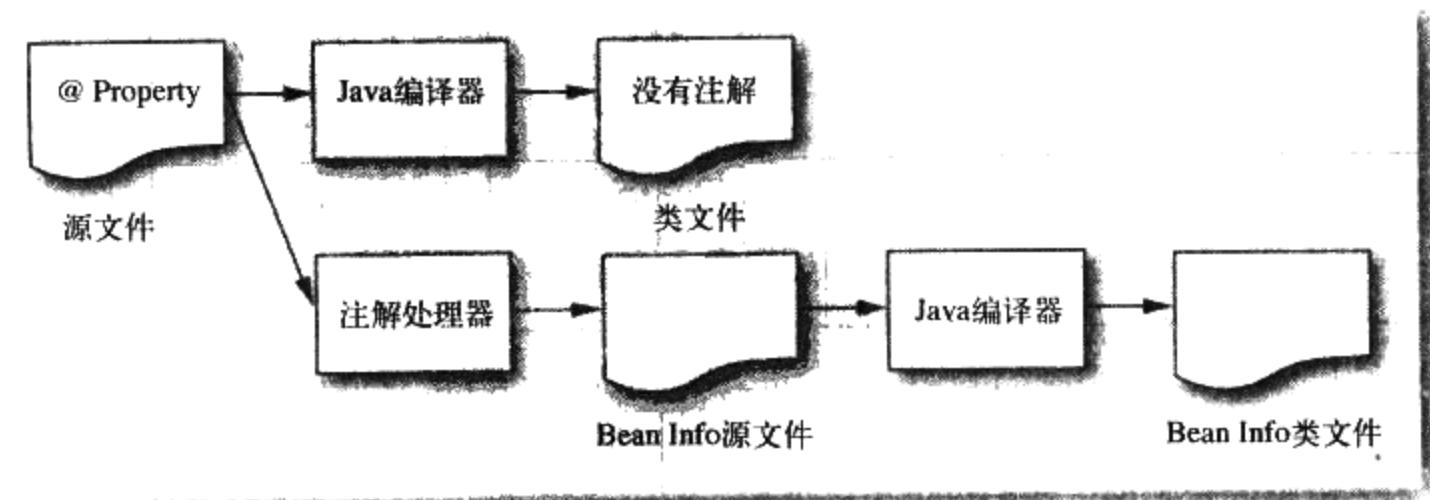


图11-3 处理源代码级的注解

我们不想在这里详细讨论注解处理的API，不过，我们将在程序清单11-12中进行了演示，让你对它的功能能够有所了解。

注解处理器通常通过扩展AbstractProcessor类实现了Processor接口。你需要指定你的处理器支持哪些注解，因为注解处理API的设计者们喜欢注解，所以他们使用注解来实现此目的：

```
@SupportedAnnotationTypes("com.horstmann.annotations.Property")
public class BeanInfoAnnotationProcessor extends AbstractProcessor
```

处理器可以声明具体的注解类型或诸如“com.horstmann.\*”（所有在com.horstmann包及其所有子包中的注解）这样的通配符，甚至可以是“\*”（所有注解）。

BeanInfoAnnotationProcessor具有惟一的公共方法——process，它可以供每个文件调用。process方法有两个参数：一个是在本次循环中要进行处理的注解集，另一个是包含有关当前处理循环信息的RoundEnv引用。

在process方法中，我们迭代所有注解过的方法。对于每个方法，我们剥离其名字中的get、set和is前缀，并将随后紧挨着的字母改为小写，从而得到属性名。下面是代码的大概轮廓：

```
public boolean process(Set<? extends TypeElement> annotations, RoundEnvironment roundEnv)
{
    for (TypeElement t : annotations)
    {
        Map<String, Property> props = new LinkedHashMap<String, Property>();
        for (Element e : roundEnv.getElementsAnnotatedWith(t))
        {
            props.put(property name, e.getAnnotation(Property.class));
        }
    }
    write bean info source file
}
return true;
}
```

如果这个处理器对所有传递给它的注解都进行了声明，那么process方法就应该返回true。也就是说，这些注解不应该传递给其他的处理器。

用于编写源文件的代码简单明了，它仅仅是一系列out.print语句。注意我们是用下面的调用来创建输出文件的：

```
JavaFileObject sourceFile = processingEnv.getFiler().createSourceFile(beanClassName + "BeanInfo");
PrintWriter out = new PrintWriter(sourceFile.openWriter());
```

AbstractProcessor类有一个受保护的域processingEnv，用于访问各种处理服务。Filer接口负责创建新的文件，并追踪它们，以便在随后的处理循环中仍能够对它们进行处理。

当一个注解处理器检测到一条错误的时候，会使用一个Messager与用户交互。举例来说，如果一个方法已经用@Property进行注解过，但是它的名字不是以get、set或者is开头，那么我们就会发布一条错误消息：

```
if (!found) processingEnv.getMessager().printMessage(Kind.ERROR,
    "@Property must be applied to getXxx, setXxx, or isXxx method", e);
```

在本书附带的代码中，我们提供了一个被注解的文件ChartBean.java。编译注解处理器：

```
javac BeanInfoAnnotationProcessor.java
```

然后运行：

```
javac -processor BeanInfoAnnotationProcessor com/horstmann/corejava/ChartBean.java
```

之后可以查看自动生成的ChartBeanBeanInfo.java文件。

要查看注解处理的行为，可以在javac命令中添加命令行选项XprintRounds。这样就可以得到下面的输出：

```
Round 1:  
    input files: {com.horstmann.corejava.ChartBean}  
    annotations: [com.horstmann.annotations.Property]  
    last round: false  
  
Round 2:  
    input files: {com.horstmann.corejava.ChartBeanBeanInfo}  
    annotations: []  
    last round: false  
  
Round 3:  
    input files: {}  
    annotations: []  
    last round: true
```

这个示例展示了工具是怎样获取源文件注解以产生其他文件的。生成的文件不必是源文件。注解处理器可以选择生成XML描述符、属性文件、shell脚本以及HTML文档等等。

 注意：一些人已经建议使用注解去免除更大的琐碎工作。如果琐碎的获取器和设置器可以自动生成，会更好吗？举例来说，下面这个注解

```
@Property private String title;
```

可以产生下面这些方法：

```
public String getTitle() { return title; }  
public void setTitle(String title) { this.title = title; }
```

不过，需要将这些方法添加到同一个类中。这样就要编辑源文件，而不仅仅是生成另一个文件，并且这也超出了注解处理器的能力。可以建立另外一个工具实现此目的，但是这样的一个工具可能会超越注解的职责所在。注解之所以存在，是因为它被当作是代码项的一种描述，而并不是被当作一种添加或修改代码的指示。

### 程序清单11-12 BeanInfoAnnotationFactory.java

```
1. import java.beans.*;  
2. import java.io.*;  
3. import java.util.*;  
4. import javax.annotation.processing.*;  
5. import javax.lang.model.*;  
6. import javax.lang.model.element.*;  
7. import javax.tools.*;  
8. import javax.tools.Diagnostic.*;  
9. import com.horstmann.annotations.*;  
10.  
11. /**  
12. * This class is the processor that analyzes Property annotations.  
13. * @version 1.10 2007-10-27  
14. * @author Cay Horstmann  
15. */
```

```
16.
17. @SupportedAnnotationTypes("com.horstmann.annotations.Property")
18. @SupportedSourceVersion(SourceVersion.RELEASE_6)
19. public class BeanInfoAnnotationProcessor extends AbstractProcessor
20. {
21.     @Override
22.     public boolean process(Set<? extends TypeElement> annotations, RoundEnvironment roundEnv)
23.     {
24.         for (TypeElement t : annotations)
25.         {
26.             Map<String, Property> props = new LinkedHashMap<String, Property>();
27.             String beanClassName = null;
28.             for (Element e : roundEnv.getElementsAnnotatedWith(t))
29.             {
30.                 String mname = e.getSimpleName().toString();
31.                 String[] prefixes = { "get", "set", "is" };
32.                 boolean found = false;
33.                 for (int i = 0; !found && i < prefixes.length; i++)
34.                     if (mname.startsWith(prefixes[i]))
35.                     {
36.                         found = true;
37.                         int start = prefixes[i].length();
38.                         String name = Introspector.decapitalize(mname.substring(start));
39.                         props.put(name, e.getAnnotation(Property.class));
40.                     }
41.
42.                 if (!found) processingEnv.getMessager().printMessage(Kind.ERROR,
43.                     "@Property must be applied to getXxx, setXxx, or isXxx method", e);
44.                 else if (beanClassName == null)
45.                     beanClassName = ((TypeElement) e.getEnclosingElement()).getQualifiedName()
46.                         .toString();
47.             }
48.             try
49.             {
50.                 if (beanClassName != null) writeBeanInfoFile(beanClassName, props);
51.             }
52.             catch (IOException e)
53.             {
54.                 e.printStackTrace();
55.             }
56.         }
57.         return true;
58.     }
59.
60. /**
61. * Writes the source file for the BeanInfo class.
62. * @param beanClassName the name of the bean class
63. * @param props a map of property names and their annotations
64. */
65. private void writeBeanInfoFile(String beanClassName, Map<String, Property> props)
66.     throws IOException
67. {
68.     JavaFileObject sourceFile = processingEnv.getFiler().createSourceFile(
69.         beanClassName + "BeanInfo");
70.     PrintWriter out = new PrintWriter(sourceFile.openWriter());
71.     int i = beanClassName.lastIndexOf(".");
72.     if (i > 0)
```

```
73.    {
74.        out.print("package ");
75.        out.print(beanClassName.substring(0, i));
76.        out.println(";");
77.    }
78.    out.print("public class ");
79.    out.print(beanClassName.substring(i + 1));
80.    out.println("BeanInfo extends java.beans.SimpleBeanInfo");
81.    out.println("{");
82.    out.println("    public java.beans.PropertyDescriptor[] getPropertyDescriptors()");
83.    out.println("    {");
84.    out.println("        try");
85.    out.println("        {");
86.    for (Map.Entry<String, Property> e : props.entrySet())
87.    {
88.        out.print("            java.beans.PropertyDescriptor ");
89.        out.print(e.getKey());
90.        out.println("Descriptor");
91.        out.print("            = new java.beans.PropertyDescriptor(\"");
92.        out.print(e.getKey());
93.        out.print("\", ");
94.        out.print(beanClassName);
95.        out.println(".class');");
96.        String ed = e.getValue().editor().toString();
97.        if (!ed.equals(""))
98.        {
99.            out.print("            ");
100.           out.print(e.getKey());
101.           out.print("Descriptor.setPropertyEditorClass(");
102.           out.print(ed);
103.           out.println(".class);");
104.        }
105.    }
106.    out.println("        return new java.beans.PropertyDescriptor[]");
107.    out.print("    }");
108.    boolean first = true;
109.    for (String p : props.keySet())
110.    {
111.        if (first) first = false;
112.        else out.print(",");
113.        out.println();
114.        out.print("            ");
115.        out.print(p);
116.        out.print("Descriptor");
117.    }
118.    out.println();
119.    out.println("        }');");
120.    out.println("    }");
121.    out.println("    catch (java.beans.IntrospectionException e)");
122.    out.println("    {");
123.    out.println("        e.printStackTrace();");
124.    out.println("        return null;");
125.    out.println("    }");
126.    out.println("}");
127.    out.println("}");
128.    out.println("}");
out.close();
```

---

```
129. }
130. }
```

## 11.7 字节码工程

你已经看到了我们是怎样在运行期或者在源码级别上对注解进行处理的。还有第三种可能：在字节码级别上进行处理。除非将注解在源码级别上删除，否则它们会一直存在于类文件中。类文件格式是文档化的（参阅<http://java.sun.com/docs/books/vmspec>）。这种格式相当复杂，并且在没有特殊类库的情况下，处理类文件具有很大的挑战性。BCEL，即字节码工程类库，就是这样的特殊类库之一，可以从网站<http://jakarta.apache.org/bcel>上获得。

在本小节，我们使用BCEL向已注解方法中添加日志信息。如果一个方法这样注解过：

```
@LogEntry(logger=loggerName)
```

那么，在方法的开始部分，我们将添加下面这条语句的字节码：

```
Logger.getLogger(loggerName).entering(className, methodName);
```

举例来说，如果对Item类的hashCode方法做了如下注解：

```
@LogEntry(logger="global") public int hashCode()
```

那么，在任何时候调用该方法，都会报告一条与下面打印出来的消息相似的消息：

```
Aug 17, 2004 9:32:59 PM Item hashCode
FINER: ENTRY
```

为了实现这项任务，我们需要遵循下面几点：

- 加载类文件中的字节码。
- 定位所有的方法。
- 对于每个方法，检查它是不是有一个LogEntry注解。
- 如果有，在方法开始部分添加下面所列指令的字节码：

```
ldc loggerName
invokestatic java/util/logging/Logger.getLogger:(Ljava/lang/String;)Ljava/util/logging/Logger;
ldc className
ldc methodName
invokevirtual java/util/logging/Logger.entering:(Ljava/lang/String;Ljava/lang/String;)V
```

插入这些字节码看起来相当棘手，不过BCEL却使它变得相当简单。我们不会详细描述和分析插入字节码的过程。关键之处是程序清单11-13中的程序编辑了一个类文件，并且在已经用LogEntry注解标注过的方法的开头部分插入了日志调用。

 **注意：**如果你对字节码工程的具体细节感兴趣，我们建议你通篇阅读一下网站<http://jakarta.apache.org/bcel/manual.html>上提供的BCEL手册。

你需要用5.3或以上版本的BCEL类库来编译和运行EntryLogger程序。（在编写本章时，5.3版仍在改进中。如果你阅读本章时它还没有完成改进，那么就请从Subversion库中签出其主干代码。）

举例来说，下面展示了应该怎样向程序清单11-14中的Item.java文件添加记录日志指令。

```
javac Item.java
javac -classpath .:bcel-version.jar EntryLogger.java
```

```
java -classpath .:bcel-version.jar EntryLogger Item
```

在对Item类文件被修改之前和之后分别试运行一下：

```
javap -c Item
```

可以看到在hashCode、equals以及compareTo方法开始部分插入的那些指令。

```
public int hashCode();
Code:
 0: ldc   #85; //String global
 2: invokestatic #80; //Method java/util/logging/Logger.getLogger:(Ljava/lang/String;)Ljava/util/logging/Logger;
 5: ldc   #86; //String Item
 7: ldc   #88; //String hashCode
 9: invokevirtual #84; //Method java/util/logging/Logger.entering:(Ljava/lang/String;Ljava/lang/String;)V
12: bipush 13
14: aload_0
15: getfield    #2; //Field description:Ljava/lang/String;
18: invokevirtual #15; //Method java/lang/String.hashCode:()I
21: imul
22: bipush 17
24: aload_0
25: getfield    #3; //Field partNumber:I
28: imul
29: iadd
30: ireturn
```

程序清单11-15中的SetTest程序将Item对象插入到一个散列集中。你运行修改过类文件的该程序的时候，会看到下面的记录日志信息：

```
Aug 18, 2004 10:57:59 AM Item hashCode
FINER: ENTRY
Aug 18, 2004 10:57:59 AM Item hashCode
FINER: ENTRY
Aug 18, 2004 10:57:59 AM Item hashCode
FINER: ENTRY
Aug 18, 2004 10:57:59 AM Item equals
FINER: ENTRY
[[description=Toaster, partNumber=1729], [description=Microwave, partNumber=4104]]
```

当将同一项插入两次的时候，请注意一下equals调用。

这个示例显示了字节码工程的强大。注解可以用来向程序中添加一些指示，而字节码编辑工具则可以提取这些指示，然后修改虚拟机指令。

### 程序清单11-13 EntryLogger.java

```
1. import java.io.*;
2. import org.apache.bcel.*;
3. import org.apache.bcel.classfile.*;
4. import org.apache.bcel.generic.*;
5.
6. /**
7.  * Adds "entering" logs to all methods of a class that have the LogEntry annotation.
8.  * @version 1.10 2007-10-27
9.  * @author Cay Horstmann
10. */
11. public class EntryLogger
12. {
13.     /**
14.      * Adds entry logging code to the given class
15.      * @param args the name of the class file to patch
16.     */
17. }
```

```
17. public static void main(String[] args)
18. {
19.     try
20.     {
21.         if (args.length == 0) System.out.println("USAGE: java EntryLogger classname");
22.         else
23.         {
24.             JavaClass jc = Repository.lookupClass(args[0]);
25.             ClassGen cg = new ClassGen(jc);
26.             EntryLogger el = new EntryLogger(cg);
27.             el.convert();
28.             File f = new File(Repository.lookupClassFile(cg.getClassName()).getPath());
29.             cg.getJavaClass().dump(f.getPath());
30.         }
31.     }
32.     catch (Exception e)
33.     {
34.         e.printStackTrace();
35.     }
36. }
37.
38. /**
39. * Constructs an EntryLogger that inserts logging into annotated methods of a given class
40. * @param cg the class
41. */
42. public EntryLogger(ClassGen cg)
43. {
44.     this.cg = cg;
45.     cpg = cg.getConstantPool();
46. }
47.
48. /**
49. * converts the class by inserting the logging calls.
50. */
51. public void convert() throws IOException
52. {
53.     for (Method m : cg.getMethods())
54.     {
55.         AnnotationEntry[] annotations = m.getAnnotationEntries();
56.         for (AnnotationEntry a : annotations)
57.         {
58.             if (a.getAnnotationType().equals("LLogEntry;"))
59.             {
60.                 for (ElementValuePair p : a.getElementValuePairs())
61.                 {
62.                     if (p.getNameString().equals("logger"))
63.                     {
64.                         String loggerName = p.getValue().stringifyValue();
65.                         cg.replaceMethod(m, insertLogEntry(m, loggerName));
66.                     }
67.                 }
68.             }
69.         }
70.     }
71. }
72.
73. /**
```

```

74.     * Adds an "entering" call to the beginning of a method.
75.     * @param m the method
76.     * @param loggerName the name of the logger to call
77.     */
78.    private Method insertLogEntry(Method m, String loggerName)
79.    {
80.        MethodGen mg = new MethodGen(m, cg.getClassName(), cpg);
81.        String className = cg.getClassName();
82.        String methodName = mg.getMethod().getName();
83.        System.out.printf("Adding logging instructions to %s.%s%n", className, methodName);
84.
85.        int getLoggerIndex = cpg.addMethodref("java.util.logging.Logger", "getLogger",
86.            "(Ljava/lang/String;)Ljava/util/logging/Logger;");
87.        int enteringIndex = cpg.addMethodref("java.util.logging.Logger", "entering",
88.            "(Ljava/lang/String;Ljava/lang/String;)V");
89.
90.        InstructionList il = mg.getInstructionList();
91.        InstructionList patch = new InstructionList();
92.        patch.append(new PUSH(cpg, loggerName));
93.        patch.append(new INVOKESTATIC(getLoggerIndex));
94.        patch.append(new PUSH(cpg, className));
95.        patch.append(new PUSH(cpg, methodName));
96.        patch.append(new INVOKEVIRTUAL(enteringIndex));
97.        InstructionHandle[] ihs = il.getInstructionHandles();
98.        il.insert(ihs[0], patch);
99.
100.       mg.setMaxStack();
101.       return mg.getMethod();
102.    }
103.
104.    private ClassGen cg;
105.    private ConstantPoolGen cpg;
106. }
```

#### 程序清单11-14 Item.java

```

1. /**
2.  * An item with a description and a part number.
3.  * @version 1.00 2004-08-17
4.  * @author Cay Horstmann
5.  */
6 public class Item
7 {
8. /**
9.  * Constructs an item.
10. * @param aDescription the item's description
11. * @param aPartNumber the item's part number
12. */
13. public Item(String aDescription, int aPartNumber)
14. {
15.     description = aDescription;
16.     partNumber = aPartNumber;
17. }
18.
19. /**
20.  * Gets the description of this item.
```

```

21.     * @return the description
22.     */
23.    public String getDescription()
24.    {
25.        return description;
26.    }
27.
28.    public String toString()
29.    {
30.        return "[description=" + description + ", partNumber=" + partNumber + "]";
31.    }
32.
33.    @LogEntry(logger = "global")
34.    public boolean equals(Object otherObject)
35.    {
36.        if (this == otherObject) return true;
37.        if (otherObject == null) return false;
38.        if (getClass() != otherObject.getClass()) return false;
39.        Item other = (Item) otherObject;
40.        return description.equals(other.description) && partNumber == other.partNumber;
41.    }
42.
43.    @LogEntry(logger = "global")
44.    public int hashCode()
45.    {
46.        return 13 * description.hashCode() + 17 * partNumber;
47.    }
48.
49.    private String description;
50.    private int partNumber;
51. }
```

### 程序清单11-15 SetTest.java

```

1. import java.util.*;
2. import java.util.logging.*;
3.
4. /**
5.  * @version 1.01 2007-10-27
6.  * @author Cay Horstmann
7. */
8. public class SetTest
9. {
10.    public static void main(String[] args)
11.    {
12.        Logger.getLogger(Logger.GLOBAL_LOGGER_NAME).setLevel(Level.FINEST);
13.        Handler handler = new ConsoleHandler();
14.        handler.setLevel(Level.FINEST);
15.        Logger.getLogger(Logger.GLOBAL_LOGGER_NAME).addHandler(handler);
16.
17.        Set<Item> parts = new HashSet<Item>();
18.        parts.add(new Item("Toaster", 1279));
19.        parts.add(new Item("Microwave", 4104));
20.        parts.add(new Item("Toaster", 1279));
21.        System.out.println(parts);
22.    }
23. }
```

## 载入时修改字节码

在前一节中，已经看到了一个用于编辑类文件的工具。不过，在把另一个工具添加到构建处理中的时候，它就会显得笨重不堪。更吸引人的做法是将字节码工程延迟到载入时，即类加载器加载类的时候。

在Java SE5.0之前，必须编写定制的类加载器才能实现这项任务。现在，设备(*instrumentation*) API提供了一个安装字节码转换器的挂钩。不过，必须在程序的main方法调用之前安装这个转换器。通过定义一个代理，即被加载用来按照某种方式监视程序的一个类库，就可以处理这个需求。代理代码可以在premain方法中进行初始化。

下面是构建一个代理所需的步骤：

- 实现一个具有下面这个方法的类：

```
public static void premain(String arg, Instrumentation instr)
```

当加载代理时，调用此方法。代理可以获取一个单一的命令行参数，该参数是通过arg参数传递进来的。instr参数可以用来安装各种各样的挂钩。

- 制作一个清单文件来设置Premain-Class属性。例如：

```
Premain-Class: EntryLoggingAgent
```

- 将代理代码打包，并生成一个JAR文件，例如：

```
javac -classpath .:bcel-version.jar EntryLoggingAgent  
jar cvfm EntryLoggingAgent.jar EntryLoggingAgent.mf Entry*.class
```

为了运行一个具有该代理的Java程序，请使用下面这个命令行选项：

```
java -javaagent:AgentJARFile=agentArgument ...
```

举例来说，运行具有实体日志代理的SetTest程序，需调用：

```
javac SetTest.java  
java -javaagent:EntryLoggingAgent.jar=Item -classpath .:bcel-version.jar SetTest
```

Item参数是代理应该修改的类的名称。

程序清单11-16展示了这个代理的代码。该代理安装了一个类文件转换器，这个转换器首先检验类名是否与代理参数相匹配。如果匹配，那么它会利用上面那个EntryLogger类修改字节码。不过，修改过的字节码并不保存成文件。相反地，转换器只是将它们返回，以加载到虚拟机中（参见图11-4）。换句话说，这项技术实现的是“及时(just in time)”字节码修改。

### 程序清单11-16 EntryLoggingAgent.java

```
1. import java.lang.instrument.*;  
2. import java.io.*;  
3. import java.security.*;  
4. import org.apache.bcel.classfile.*;  
5. import org.apache.bcel.generic.*;  
6.  
7. /**  
8.  * @version 1.00 2004-08-17  
9.  * @author Cay Horstmann  
10. */  
11. public class EntryLoggingAgent  
12. {  
13.     public static void premain(final String arg, Instrumentation instr)
```

```

14. {
15.     instr.addTransformer(new ClassFileTransformer()
16.     {
17.         public byte[] transform(ClassLoader loader, String className, Class<?> c1,
18.             ProtectionDomain pd, byte[] data)
19.         {
20.             if (!className.equals(arg)) return null;
21.             try
22.             {
23.                 ClassParser parser = new ClassParser(new ByteArrayInputStream(data),
24.                     className + ".java");
25.                 JavaClass jc = parser.parse();
26.                 ClassGen cg = new ClassGen(jc);
27.                 EntryLogger el = new EntryLogger(cg);
28.                 el.convert();
29.                 return cg.getJavaClass().getBytes();
30.             }
31.             catch (Exception e)
32.             {
33.                 e.printStackTrace();
34.                 return null;
35.             }
36.         }
37.     });
38. }
39. }

```

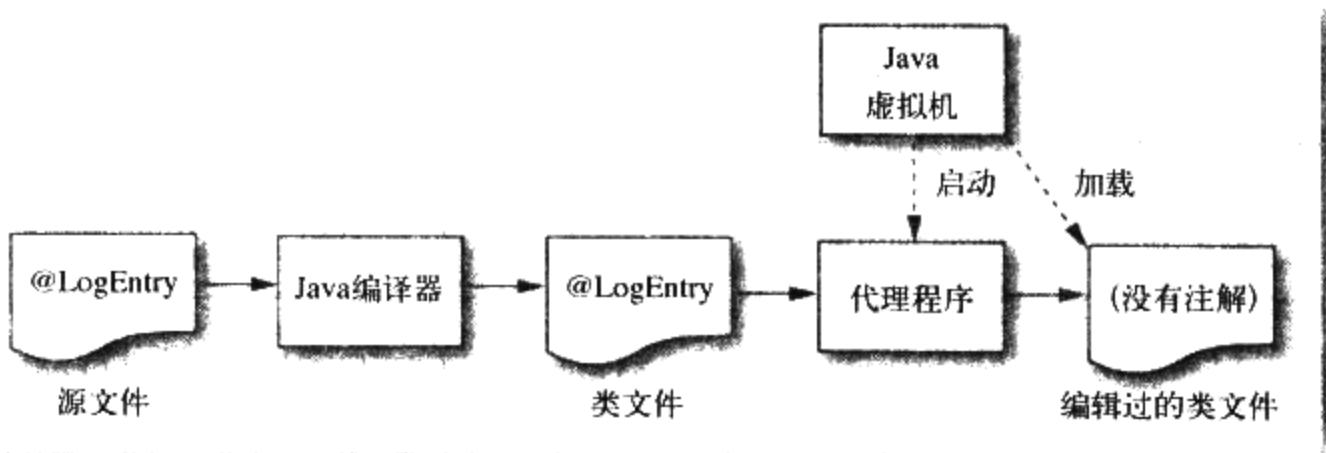


图11-4 在加载时刻修改类

在本章，你已经学习到了以下的知识：

- 怎样向Java程序中添加注解。
- 怎样设计你自己的注解接口。
- 怎样实现可以利用注解的工具。

你已经看到了三种处理代码的技术：编写脚本、编译Java程序和处理注解。前两种技术十分简单。而另一方面，创建注解工具可能会很复杂，但这并非是大多数开发者都需要解决的问题。本章向你介绍了一些背景知识，有助于你去理解可能会碰到的注解工具，但这些背景知识可能会挫伤你自行开发工具的兴趣。

在本书的最后一章，我们将处理用于本地方法的API。这种API允许你将Java和C/C++代码混杂在一起。

# 第12章 本地方法

- ▲ 从Java程序中调用C函数
- ▲ 数值参数与返回值
- ▲ 字符串参数
- ▲ 访问域
- ▲ 编码签名
- ▲ 调用Java方法
- ▲ 访问数组元素
- ▲ 错误处理
- ▲ 使用调用API
- ▲ 完整的示例：访问Windows注册表

原则上说，“100%纯Java”的解决方案是非常好的，但有时你想要编写或使用其他语言的代码（这种代码通常称为本地代码）。

特别是在Java的早期阶段，许多人都认为使用C或C++来加速Java应用中关键部分是个好主意。但是，实际上，这基本上是徒劳的。1996年JavaOne会议上的一个演讲很明确地说明了这一点，来自Sun Microsystems的密码系统类库的实现者们报告说他们的密码功能的纯Java平台实现已臻化境。他们的代码确实没有已有的C实现快，但是事实证明这无关紧要。Java平台实现比网络I/O要快得多，而后者被证明是真正的瓶颈。

当然，求助于本地代码是有缺陷的。如果应用的某个部分是用其他语言编写的，那么就必须为需要支持的每个平台都提供一个单独的本地类库。用C或C++编写的代码没有对通过使用无效指针所造成的内存覆写提供任何保护。编写本地代码很容易破坏你的程序，并感染操作系统。

因此，我们建议只有在必需的时候才使用本地代码。特别是，本地代码之所以可能是正确选择，是因为有如下三个理由：

- 你的应用需要访问系统的各个特性和设备，这些特性和设备通过Java平台是无法访问的。
- 你已经有了大量的测试过和调试过的用另一种语言编写的代码，并且知道如何将其导出到所有的目标平台上。
- 通过基准测试，你已经发现所编写的Java代码比用其他语言编写的等价代码要慢得多。

Java平台有一个用于和本地C代码进行互操作的API，称为Java本地接口（JNI）。我们将在本章讨论JNI编程。



**C++注意：**你可以使用C++代替C来编写本地方法。这样会有一些好处：类型检查会更严格一些，访问JNI函数会更便捷一些。然而，JNI并不支持Java类和C++类之间的任何映射机制。

## 12.1 从Java程序中调用C函数

假设你有一个C函数，它能为你实现某个功能，因为某种原因，你不想费事使用Java编程语言重新实现它。为了方便说明问题，我们从一个很简单的打印问候语的C函数入手。

Java编程语言使用关键字native表示本地方法，而且很显然，你还需要在类中放置一个方法。其结果显示在程序清单12-1中。

关键字native提醒编译器该方法将在外部定义。当然，本地方法不包含Java编程语言的代码，而且方法标题后直接跟着一个表示终结的分号。正如你在上面的例子中看到的，这意味着本地方法声明看上去和抽象方法声明类似。

### 程序清单12-1 HelloNative.java

```
1  /**
2  * @version 1.11 2007-10-26
3  * @author Cay Horstmann
4  */
5 class HelloNative
6 {
7     public static native void greeting();
8 }
```

注意，本地方法也被声明为static。本地方法既可以是静态的也可以是非静态的。我们使用静态方法是因为我们还不需要处理参数传递。

你实际上可以编译这个类，但是在程序中使用它时，虚拟机就会告诉你它不知道如何找到greeting，它会报告一个UnsatisfiedLinkError异常。为了实现本地代码，需要编写一个相应的C函数，你必须完全按照Java运行环境预期的那样来命名这个函数。其规则是：

- 1) 使用完整的Java方法名，比如：HelloNative.greeting。如果类属于某个包，那么在前面添加包名，比如：com.horstmann.HelloNative.greeting。
- 2) 用下划线替换掉所有的句号，并加上Java\_前缀，例如，Java\_HelloNative\_greeting或Java\_com\_horstmann\_HelloNative\_greeting。
- 3) 如果类名含有非ASCII字母或数字，如：'\_'，'\$'或是大于'\u007F'的Unicode字符，用\_0xxxx来替代它们，xxxx是该字符的值的4个十六进制数



**注意：**如果你重载本地方法，也就是说，你用相同的名字提供多个本地方法，那么你必须在名称后附加两个下划线，后面再加上已编码的参数类型。在本章后面，我们将描述参数类型的编码方法。例如，如果你有一个本地方法greeting和另一个本地方法greeting(int repeat)，那么，第一个称为Java\_HelloNative\_greeting\_\_，第二个称为Java\_HelloNative\_greeting\_\_I。

实际上，没人会手工完成这些操作。相反，你应该运行javah实用程序，它能够自动生成函数名。要使用javah，首先要编译程序清单12-1中的源文件。

```
javac HelloNative.java
```

接着，调用javah实用程序，从该类文件中产生一个C的头文件。Javah可执行文件可以在jdk/bin目录下找到。可以用类的名字来调用它，就像调用Java编译器一样。例如，

```
javah HelloNative
```

这条命令会产生一个头文件HelloNative.h，参见程序清单12-2。

**程序清单12-2 HelloNative.h**

```
1. /* DO NOT EDIT THIS FILE - it is machine generated */
2. #include <jni.h>
3. /* Header for class HelloNative */
4.
5. #ifndef _Included_HelloNative
6. #define _Included_HelloNative
7. #ifdef __cplusplus
8. extern "C" {
9. #endif
10. /*
11. * Class:     HelloNative
12. * Method:    greeting
13. * Signature: ()V
14. */
15. JNIEXPORT void JNICALL Java_HelloNative_greeting
16.   (JNIEnv *, jclass);
17.
18. #ifdef __cplusplus
19. }
20. #endif
21. #endif
```

如你所见，这个文件包含了函数Java\_HelloNative\_greeting的声明（字符串JNIEXPORT和JNICALL是在头文件jni.h中定义的。它们为那些来自动态装载库的导出函数标明了依赖于编译器的说明符）。

现在，需要将函数原型从头文件中复制到源文件中，并且给出函数的实现代码，如程序清单12-3所示。

**程序清单12-3 HelloNative.c**

```
1. /*
2.  * @version 1.10 1997-07-01
3.  * @author Cay Horstmann
4. */
5.
6. #include "HelloNative.h"
7. #include <stdio.h>
8.
9. JNIEXPORT void JNICALL Java_HelloNative_greeting(JNIEnv* env, jclass cl)
10. {
11.   printf("Hello Native World!\n");
12. }
```

在这个简单的函数中，我们忽略了env和cl参数。后面你会看到它们的用处。

 **C++注意：**你可以使用C++实现本地方法。然而，那样你必须将实现本地方法的函数声明为extern"C"（阻止C++编译器生成C++特有的代码）。例如：

```
extern "C"
JNIEXPORT void JNICALL Java_HelloNative_greeting(JNIEnv* env, jclass cl)
{
    cout << "Hello, Native World!" << endl;
}
```

将本地C代码编译到一个动态装载库中。具体方法依赖于编译器。

例如，Linux下的Gnu C编译器，使用如下命令：

```
gcc -fPIC -I jdk/include -I jdk/include/linux -shared -o libHelloNative.so HelloNative.c
```

如果是Solaris操作系统的Sun编译器，命令是：

```
cc -G -I jdk/include -I jdk/include/solaris -o libHelloNative.so HelloNative.c
```

用Windows下的微软编译器，命令是：

```
cl -I jdk\include -I jdk\include\win32 -LD HelloNative.c -FeHelloNative.dll
```

这里jdk是含有JDK的目录。

**!** 提示：如果你要从命令shell使用微软的编译器，首先要运行批处理文件vcvars32.net或vsvars32.bat。这个批处理文件设置了编译器需要的路径和环境变量。你可以在目录c:\Program Files\Microsoft Visual Studio .NET 2003\Common7\tools、c:\Program Files\Microsoft Visual Studio 8\VC(或类似位置)找到该文件。

你也可以使用可从<http://www.cygwin.com>自由获取的Cygwin编程环境。它包含了Gnu C编译器和Windows下的UNIX风格编程的库。使用Cygwin时，用以下命令：

```
gcc -mno-cygwin -D __int64="long long" -I jdk/include/ -I jdk/include/win32  
-shared -Wl,--add-stdcall-alias -o HelloNative.dll HelloNative.c
```

整个命令应该键入在同一行中。

**✓ 注意：**Windows版本的头文件jni\_md.h含有如下类型声明：

```
typedef __int64 jlong;
```

它是专门用于微软编译器的。如果你使用的是Gnu编译器，那么你就需要修改这个文件，例如：

```
#ifdef __GNUC__  
    typedef long long jlong;  
#else  
    typedef __int64 jlong;  
#endif
```

或者，如编译器调用的示例那样，使用-D \_\_int64="long long"进行编译。

最后，我们要程序中添加一个对System.loadLibrary方法的调用。为了确保虚拟机在第一次使用该类之前就会装载这个库，需要使用静态初始化代码块，如程序清单12-4所示。

图12-1给出了对本地代码处理的总结。

#### 程序清单12-4 HelloNativeTest.java

```
1. /**  
2. * @version 1.11 2007-10-26  
3. * @author Cay Horstmann  
4. */  
5. class HelloNativeTest  
6. {  
7.     public static void main(String[] args)  
8.     {  
9.         HelloNative.greeting();
```

```
10. }
11.
12. static
13. {
14.     System.loadLibrary("HelloNative");
15. }
16 }
```

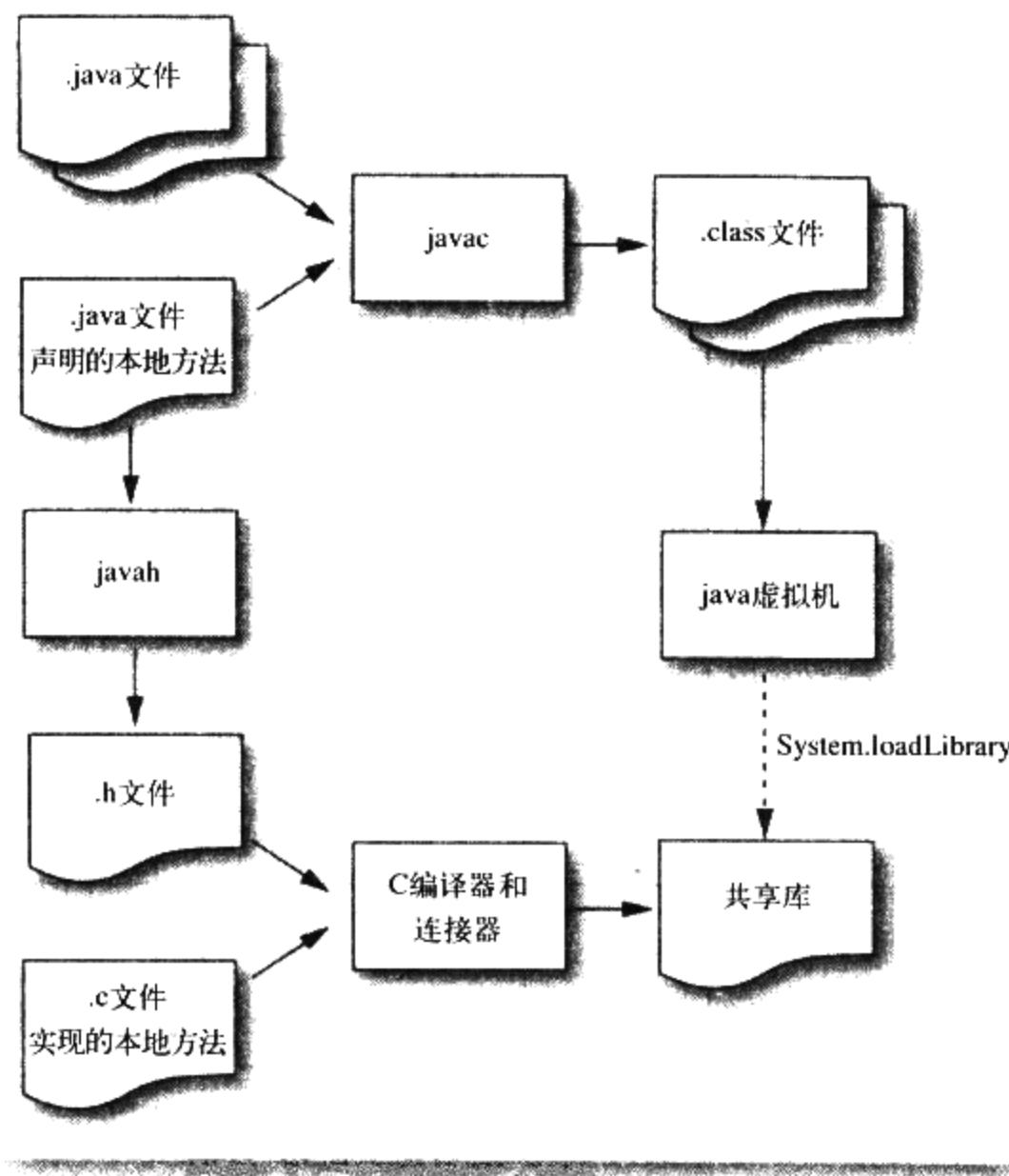


图12-1 处理本地代码

如果编译并运行该程序，终端窗口会显示消息“Hello, Native World!”。

**注意：**如果运行的是Linux，必须把当前目录添加到库路径中。实现方式可以是通过设置**LD\_LIBRARY\_PATH**环境变量：

```
export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
```

或者是设置**java.library.path**系统属性：

```
java -Djava.library.path=. HelloNativeTest
```

当然，这个消息本身并不会给人留下深刻印象。然而，如果你记住这个信息是由C的

`printf`命令产生而不是由任何Java编程语言代码产生的话，你就会明白我们已经在连接两种语言上走出了第一步。

总之，遵循下面的步骤就可以将一个本地方法链接到Java程序中：

- 1) 在Java类中声明一个本地方法。
- 2) 运行javah以获得包含该方法的C声明的头文件。
- 3) 用C实现该本地方法。
- 4) 将代码置于共享类库中。
- 5) 在Java程序中加载该类库。



### java.lang.System 1.0

- `void loadLibrary(String libname)`

装载指定名字的库，该库位于库搜索路径中。定位该库的确切方法依赖于操作系统。



**注意：**一些本地代码的共享库必须先运行初始化代码。你可以把初始化代码放到`JNI_OnLoad`方法中。类似地，如果你提供该方法，当虚拟机关闭时，将会调用`JNI_OnUnload`方法。

它们的原型是：

```
jint JNI_OnLoad(JavaVM* vm, void* reserved);
void JNI_OnUnload(JavaVM* vm, void* reserved);
```

`JNI_OnLoad`方法要返回它所需的虚拟机的最低版本，例如：

```
JNI_VERSION_1_2.
```

## 12.2 数值参数与返回值

当在C和Java之间传递数字时，应该知道它们彼此之间的对应类型。例如，尽管C拥有`int`和`long`的数据类型，但是它们的实现却是取决于平台的。在一些平台上，`int`类型是16位的，在另外一些平台上是32位的。当然，在Java平台上`int`类型总是32位的整数。基于这个原因，Java本地接口定义了`jint`、`jlong`等类型。

表12-1显示了Java数据类型和C数据类型的对应关系。

表12-1 Java数据类型和C数据类型

| Java编程语言             | C编程语言                 | 字 节 | Java编程语言            | C编程语言                | 字 节 |
|----------------------|-----------------------|-----|---------------------|----------------------|-----|
| <code>boolean</code> | <code>jboolean</code> | 1   | <code>int</code>    | <code>jint</code>    | 4   |
| <code>byte</code>    | <code>jbyte</code>    | 1   | <code>long</code>   | <code>jlong</code>   | 8   |
| <code>char</code>    | <code>jchar</code>    | 2   | <code>float</code>  | <code>jfloat</code>  | 4   |
| <code>short</code>   | <code>jshort</code>   | 2   | <code>double</code> | <code>jdouble</code> | 8   |

在头文件`jni.h`中，这些类型被`typedef`语句声明为在目标平台上等价的类型。该头文件也定义了常量`JNI_FALSE = 0`和`JNI_TRUE = 1`。

## 用printf格式化数字

直到Java SE 5.0, Java才有了与C语言的printf函数相类似的方法。在下面的示例中, 我们假设你依然坚持使用古老版本的JDK, 并且决定通过调用本地函数中的C的printf函数来实现同样的功能。

程序清单12-5给出了一个名为Printf1的类, 它使用本地方法来打印给定域宽度和精度的浮点数。

程序清单12-5 Printf1.java

```
1. /**
2. * @version 1.10 1997-07-01
3. * @author Cay Horstmann
4. */
5. class Printf1
6. {
7.     public static native int print(int width, int precision, double x);
8.
9.     static
10.    {
11.        System.loadLibrary("Printf1");
12.    }
13.}
```

注意, 用C实现该方法时, 所有的int和double参数都要转换成jint和jdouble, 如程序清单12-6所示。

程序清单12-6 Printf1.c

```
1. /**
2. * @version 1.10 1997-07-01
3. * @author Cay Horstmann
4. */
5.
6. #include "Printf1.h"
7. #include <stdio.h>
8.
9. JNIEXPORT jint JNICALL Java_Printf1_print(JNIEnv* env, jclass cl,
10.     jint width, jint precision, jdouble x)
11. {
12.     char fmt[30];
13.     jint ret;
14.     sprintf(fmt, "%*d.%df", width, precision);
15.     ret = printf(fmt, x);
16.     fflush(stdout);
17.     return ret;
18. }
```

该函数仅仅装配变量fmt中的格式字符串"%w.pf", 然后调用printf函数, 接着返回打印出的字符的个数。

程序清单12-7给出了验证Printf1类的测试程序。

**程序清单12-7 Printf1Test.java**

```

1. /**
2. * @version 1.10 1997-07-01
3. * @author Cay Horstmann
4. */
5. class Printf1Test
6. {
7.     public static void main(String[] args)
8.     {
9.         int count = Printf1.print(8, 4, 3.14);
10.        count += Printf1.print(8, 4, count);
11.        System.out.println();
12.        for (int i = 0; i < count; i++)
13.            System.out.print("-");
14.        System.out.println();
15.    }
16. }

```

### 12.3 字符串参数

接着，我们要考虑怎样把字符串传入、传出本地方法。如你所知，Java编程语言中的字符串是UTF-16编码点的序列，而C的字符串则是以null结尾的字节序列，所以在这两种语言中的字符串是很不一样的。Java本地接口有两组操作字符串的函数，一组把Java字符串转换成“改良的UTF-8”字节序列，另一组将它们转换成UTF-16数值的数组，也就是说转换成jchar数组。(UTF-8、“改良的UTF-8”和UTF-16格式都已经在卷I第12章中讨论过了，请回忆一下，“改良的UTF-8”编码保持ASCII字符不变，但是其他所有Unicode字符被编码为多字节序列。)

 **注意：**标准UTF-8编码和“改良的UTF-8”编码的差别仅在于编码大于0xFFFF的增补字符。在标准UTF-8编码中，这些字符编码为4字节序列；然而，在改良的编码中，字符首先被编码为一对UTF-16编码的“替代品”，然后再对每个替代品用UTF-8编码，总共产生6字节编码。这有点笨拙，但这是个由历史原因造成的意外，编写Java虚拟机规范的时候Unicode还局限在16位。

如果你的C代码已经使用了Unicode，你可以使用第二组转换函数。另一方面，如果你的字符串都仅限于使用ASCII字符，你可以使用“改良的UTF-8”转换函数。

带有字符串参数的本地方法实际上都要接受一个jstring类型的值。带有字符串参数返回值的本地方法必须返回一个jstring类型的值。JNI函数将读入并构造出这些jstring对象。例如，NewStringUTF函数从包含ASCII字符的字符数组创建一个新的jstring对象，或者是更一般的“改良的UTF-8”编码的字节序列。

JNI函数有一个有些古怪的调用约定。下面是对NewStringUTF函数的一个调用：

```

JNIEXPORT jstring JNICALL Java_HelloNative_getGreeting(JNIEnv* env, jclass c1)
{
    jstring jstr;
    char greeting[] = "Hello, Native World\n";
    jstr = (*env)->NewStringUTF(env, greeting);
    return jstr;
}

```

注意：除非明确地指出，否则本章中的所有代码都是C代码。

所有对JNI函数的调用都使用到了env指针，该指针是每一个本地方法的第一个参数。env指针是函数指针表的指针（参见图12-2）。所以，你必须在每个JNI调用前面加上(\*env)->，以便实际上取消对函数指针的引用。而且，env是每个JNI函数的第一个参数。

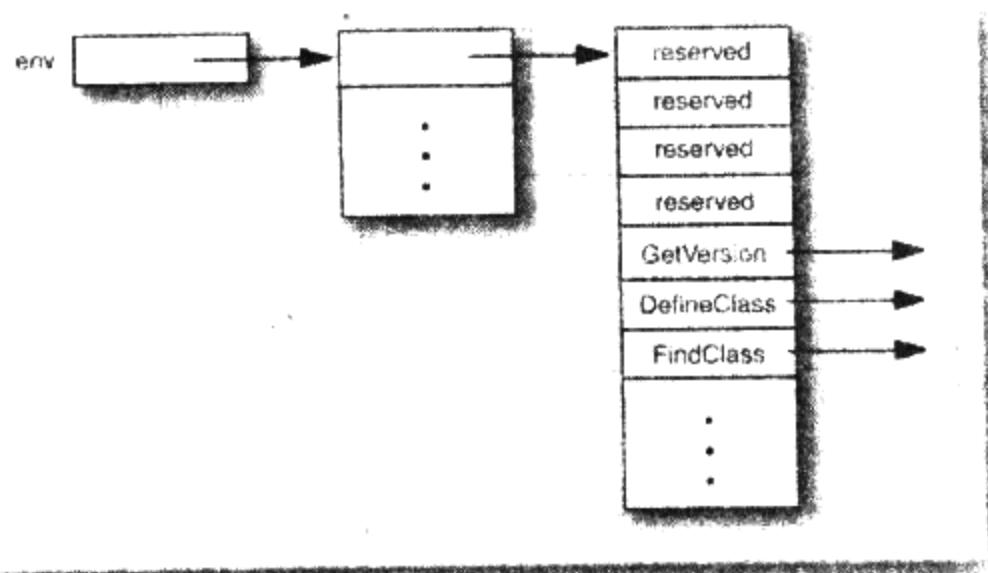


图12-2 env指针

C++注意：C++中对JNI函数的访问要简单一些。JNIEnv类的C++版本有一个内联成员函数，它负责帮你查找函数指针。例如，你可以这样调用NewStringUTF函数：

```
jstr = env->NewStringUTF(greeting);
```

注意，这里从该调用的参数列表里删掉了JNIEnv指针。

NewStringUTF函数可以用来构造一个新的jstring，而读取现有jstring对象的内容，需要使用GetStringUTFChars函数。该函数返回指向描述字符串的“改良UTF-8”字符的const jbyte\*指针。注意，具体的虚拟机可以自由地选择该编码来表示它内部的字符串。所以，你可以得到实际的Java字符串的字符指针。因为Java字符串是不可变的，所以慎重处理const就显得非常重要，不要试图将数据写到该字符数组中。另一方面，如果虚拟机使用UTF-16或UTF-32字符作为其内部字符串的表示，那么该函数会分配一个新的内存块来存储等价的“改良UTF-8”编码字符。

虚拟机必须知道你何时使用完字符串，这样它就能进行垃圾回收（垃圾回收器是在一个独立线程中运行的，它能够中断本地方法的执行）。基于这个原因，你必须调用ReleaseStringUTFChars函数。

另外，可以通过调用GetStringRegion或GetStringUTFRegion方法来提供你自己的缓存，以存放字符串的字符。

最后GetStringUTFLength函数返回编码字符串所需的“改良UTF-8”字符个数。

注意：你可以在<http://java.sun.com/javase/6/docs/technotes/guides/jni/index.html>处找到JNI API。

### API 从C代码访问Java字符串

- `jstring NewStringUTF(JNIEnv* env, const char bytes[])`  
根据以全0字节结尾的“改良UTF-8”字节序列，返回一个新的Java字符串对象，或者当字符串无法构造时，返回NULL。
  - `jsize GetStringUTFLength(JNIEnv* env, jstring string)`  
返回进行UTF-8编码所需的字节个数。(不计算作为终止符的全0字节)
  - `const jbyte* GetStringUTFChars(JNIEnv* env, jstring string, jboolean* isCopy)`  
返回“改良UTF-8”编码的字符串的指针，或者当不能构建字符数组时返回NULL。直到ReleaseStringUTFChars函数调用前，该指针一直有效。`isCopy`指向一个`jboolean`，如果进行了复制，则填入`JNI_TRUE`，否则填入`JNI_FALSE`。
  - `void ReleaseStringUTFChars(JNIEnv* env, jstring string, const jbyte bytes[])`  
通知虚拟机本地代码不再需要通过`bytes`(GetStringUTFChars返回的指针)访问Java字符串。
  - `void GetStringRegion(JNIEnv *env, jstring string, jsize start, jsize length, jchar *buffer)`  
将一个UTF-16双字节序列从字符串复制到用户提供的尺寸至少大于 $2 \times \text{length}$ 的缓存中。
  - `void GetStringUTFRegion(JNIEnv *env, jstring string, jsize start, jsize length, jbyte *buffer)`  
将一个“改良UTF-8”字符序列从字符串复制到用户提供的缓存中。为了存放要复制的字节，该缓存必须足够长。最坏情况下，要复制 $3 \times \text{length}$ 个字节。
  - `jstring NewString(JNIEnv* env, const jchar chars[], jsize length)`  
根据Unicode字符串返回一个新的Java字符串对象，或者在不能创建时返回NULL。  
参数：  
    - `env`      JNI接口指针
    - `chars`    以null结尾的UTF-16字符串
    - `length`   字符串中字符的个数
  - `jsize GetStringLength(JNIEnv* env, jstring string)`  
返回字符串中字符的个数。
  - `const jchar* GetStringChars(JNIEnv* env, jstring string, jboolean* isCopy)`  
返回Unicode编码的字符串的指针，或者当不能构建字符数组时返回NULL。直到ReleaseStringChars函数调用前，该指针一直有效。`isCopy`为NULL或者是指向`JNI_TRUE`填充的`jboolean`；否则，它指向`JNI_FALSE`填充的`jboolean`。
  - `void ReleaseStringChars(JNIEnv* env, jstring string, const jchar chars[])`  
通知虚拟机本地代码不再需要通过`chars`(GetStringChars返回的指针)访问Java字符串。
- 让我们使用这些函数来编写一个调用C函数`sprintf`的类。我们要像程序清单12-8所示那样调用函数。

程序清单12-9给出了带有本地`sprint`方法的类

因此，格式化浮点数的C函数原型如下：

```
JNIEXPORT jstring JNICALL Java_Printf2_sprint(JNIEnv* env, jclass cl, jstring format, jdouble x)
```

**程序清单12-8 Printf2Test.java**

```
1. /**
2. * @version 1.10 1997-07-01
3. * @author Cay Horstmann
4. */
5. class Printf2Test
6. {
7.     public static void main(String[] args)
8.     {
9.         double price = 44.95;
10.        double tax = 7.75;
11.        double amountDue = price * (1 + tax / 100);
12.
13.        String s = Printf2.sprint("Amount due = %8.2f", amountDue);
14.        System.out.println(s);
15.    }
16. }
```

**程序清单12-9 Printf2.java**

```
1. /**
2. * @version 1.10 1997-07-01
3. * @author Cay Horstmann
4. */
5. class Printf2
6. {
7.     public static native String sprint(String format, double x);
8.
9.     static
10.    {
11.        System.loadLibrary("Printf2");
12.    }
13. }
```

程序清单12-10给出了C的实现代码。注意，我们是通过调用GetStringUTFChars来读取格式参数的，通过调用NewStringUTF来产生返回值，通过调用ReleaseStringUTFChars来通知虚拟机不再需要访问该字符串。

**程序清单12-10 Printf2.c**

```
1. /**
2. * @version 1.10 1997-07-01
3. * @author Cay Horstmann
4. */
5.
6. #include "Printf2.h"
7. #include <string.h>
8. #include <stdlib.h>
9. #include <float.h>
10.
11. /**
12. * @param format a string containing a printf format specifier
13. * (such as "%8.2f"). Substrings "%" are skipped.
14. * @return a pointer to the format specifier (skipping the '%')
```

```
15.     or NULL if there wasn't a unique format specifier
16. */
17. char* find_format(const char format[])
18. {
19.     char* p;
20.     char* q;
21.
22.     p = strchr(p + 2, '%');
23.     if (p == NULL) return NULL;
24.     /* now check that % is unique */
25.     p = strchr(format, '%');
26.     while (p != NULL && *(p + 1) == '%') /* skip %% */
27.         p++;
28.     q = strchr(p, '%');
29.     while (q != NULL && *(q + 1) == '%') /* skip %% */
30.         q = strchr(q + 2, '%');
31.     if (q != NULL) return NULL; /* % not unique */
32.     q = p + strspn(p, " -0#"); /* skip past flags */
33.     q += strspn(q, "0123456789"); /* skip past field width */
34.     if (*q == '.') { q++; q += strspn(q, "0123456789"); }
35.     /* skip past precision */
36.     if (strchr("eEfFgG", *q) == NULL) return NULL;
37.     /* not a floating-point format */
38.     return p;
39. }
40.
41. JNIEXPORT jstring JNICALL Java_Printf2_sprint(JNIEnv* env, jclass cl,
42.                                               jstring format, jdouble x)
43. {
44.     const char* cformat;
45.     char* fmt;
46.     jstring ret;
47.
48.     cformat = (*env)->GetStringUTFChars(env, format, NULL);
49.     fmt = find_format(cformat);
50.     if (fmt == NULL)
51.         ret = format;
52.     else
53.     {
54.         char* cret;
55.         int width = atoi(fmt);
56.         if (width == 0) width = DBL_DIG + 10;
57.         cret = (char*) malloc(strlen(cformat) + width);
58.         sprintf(cret, cformat, x);
59.         ret = (*env)->NewStringUTF(env, cret);
60.         free(cret);
61.     }
62.     (*env)->ReleaseStringUTFChars(env, format, cformat);
63.     return ret;
64. }
```

在本函数中，我们选择简化错误处理。如果打印浮点数的格式代码不是%w.pc形式的（其中c是e、E、f、g或G中的一个），那么我们将不对数字进行格式化。后面我们会介绍如何让本地方法抛出异常。

## 12.4 访问域

目前为止你看到的所有本地方法都是带有数字或字符串参数的静态方法。下面，我们考虑操作对象的本地方法。作为一个练习，我们用本地方法实现卷I第4章中的Employee类的一个方法。这不是通常情况下你想要做的，但是这确实演示了当你需要的时候怎样从本地方法访问对象域。

### 12.4.1 访问实例域

为了了解怎样从本地方法访问实例域，我们用Java重新实现了raiseSalary方法。其代码很简单：

```
public void raiseSalary(double byPercent)
{
    salary *= 1 + byPercent / 100;
}
```

让我们重写代码，使其成为一个本地方法。与此前的本地方法不同，它并不是一个静态方法。运行javah给出以下原型：

```
JNIEXPORT void JNICALL Java_Employee_raiseSalary(JNIEnv *, jobject, jdouble);
```

注意，第二个参数不再是jclass类型而是jobject类型。实际上，它和this引用等价。静态方法得到的是类的引用，而非静态方法得到的是隐含的this参数对象的引用。

现在，我们访问隐含参数的salary域。在Java1.0中“原始的”Java到C的绑定中，这很简单，程序员可以直接访问对象数据域。然而，直接访问要求虚拟机暴露它们的内部数据布局。基于这个原因，JNI要求程序员通过调用特殊的JNI函数来获取和设置数据的值。

在我们的例子里，要使用GetdoubleField和SetDoubleField函数，因为salary是double类型的。对于其他类型，可以使用的函数有：GetIntField/SetIntField、GetObjectField/SetObjectField等等。其一般语法是：

```
x = (*env)->GetXxxField(env, this_obj, fieldID);
(*env)->SetXxxField(env, this_obj, fieldID, x);
```

这里，class是代表Class类型Java对象的一个值，fieldID是一个特殊类型的值，jfieldID标识结构中的一个域，Xxx代表Java数据类型（Object、Boolean、Byte或其他）。有两种方法得到class对象。GetObjectClass函数返回任意对象的类。例如：

```
jclass class_Employee = (*env)->GetObjectClass(env, this_obj);
```

FindClass函数可以让你以字符串形式来指定类名（有点奇怪的是，要以/代替句号作为包名之间的分隔符）。

```
jclass class_String = (*env)->FindClass(env, "java/lang/String");
```

使用GetFieldID函数来获得fieldID。必须提供域的名字、它的签名以及它的类型的编码。例如，下面是从salary域得到域ID的代码：

```
jfieldID id_salary = (*env)->GetFieldID(env, class_Employee, "salary", "D");
```

字符串"D"表示类型是double。将在下一节中学习到编码签名的全部规则。

你可能会认为访问数据域相当令人费解。JNI的设计者不想把数据域直接暴露在外，所以他们不得不提供获取和设置数据域值的函数。为了使这些函数的开销最小化，从域名计算域ID

(代价最大的一个步骤)被分解出来作为单独的一步操作。也就是说,如果你反复地获取和设置一个特定的域,你计算域标识符的开销只有一次。

让我们把各部分汇总起来,下面的代码以本地方法形式重新实现了raiseSalary方法。

```
JNIEXPORT void JNICALL Java_Employee_raiseSalary(JNIEnv* env, jobject this_obj, jdouble byPercent)
{
    /* get the class */
    jclass class_Employee = (*env)->GetObjectClass(env, this_obj);

    /* get the field ID */
    jfieldID id_salary = (*env)->GetFieldID(env, class_Employee, "salary", "D");

    /* get the field value */
    jdouble salary = (*env)->GetDoubleField(env, this_obj, id_salary);

    salary *= 1 + byPercent / 100;

    /* set the field value */
    (*env)->SetDoubleField(env, this_obj, id_salary, salary);
}
```

 **警告:**类引用只在本地方法返回之前有效。因此,不能在你的代码中缓存GetObjectClass的返回值。不要将类引用保存下来以供以后的方法调用重复使用。必须在每次执行本地方法时都调用GetObjectClass。如果你无法忍受这一点,必须调用NewGlobalRef来锁定该引用:

```
static jclass class_X = 0;
static jfieldID id_a;

if (class_X == 0)
{
    jclass cx = (*env)->GetObjectClass(env, obj);
    class_X = (*env)->NewGlobalRef(env, cx);
    id_a = (*env)->GetFieldID(env, cls, "a", "...");
```

现在,你可以在后面的调用中使用类引用和域ID了。当你结束对类的使用时,务必调用:  
`(*env)->DeleteGlobalRef(env, class_X);`

程序清单12-11和12-12给出了测试程序和Employee类的Java代码。程序清单12-13包含了本地raiseSalary方法的C代码。

### 程序清单12-11 EmployeeTest.java

```
1. /**
2. * @version 1.10 1999-11-13
3. * @author Cay Horstmann
4. */
5.
6. public class EmployeeTest
7. {
8.     public static void main(String[] args)
9.     {
10.         Employee[] staff = new Employee[3];
11. }
```

```
12.     staff[0] = new Employee("Harry Hacker", 35000);
13.     staff[1] = new Employee("Carl Cracker", 75000);
14.     staff[2] = new Employee("Tony Tester", 38000);
15.
16.     for (Employee e : staff)
17.         e.raiseSalary(5);
18.     for (Employee e : staff)
19.         e.print();
20. }
21 }
```

### 程序清单12-12 Employee.java

```
1. /**
2. * @version 1.10 1999-11-13
3. * @author Cay Horstmann
4. */
5.
6. public class Employee
7. {
8.     public Employee(String n, double s)
9.     {
10.         name = n;
11.         salary = s;
12.     }
13.
14.     public native void raiseSalary(double byPercent);
15.
16.     public void print()
17.     {
18.         System.out.println(name + " " + salary);
19.     }
20.
21.     private String name;
22.     private double salary;
23.
24.     static
25.     {
26.         System.loadLibrary("Employee");
27.     }
28. }
```

### 程序清单12-13 Employee.c

```
1. /**
2. * @version 1.10 1999-11-13
3. * @author Cay Horstmann
4. */
5.
6. #include "Employee.h"
7.
8. #include <stdio.h>
9.
10. JNIEXPORT void JNICALL Java_Employee_raiseSalary(JNIEnv* env, jobject this_obj,
11.         jdouble byPercent)
12. {
```

```

13  /* get the class */
14  jclass class_Employee = (*env)->GetObjectClass(env, this_obj);
15.
16  /* get the field ID */
17  jfieldID id_salary = (*env)->GetFieldID(env, class_Employee, "salary", "D");
18.
19  /* get the field value */
20  jdouble salary = (*env)->GetDoubleField(env, this_obj, id_salary);
21.
22  salary *= 1 + byPercent / 100;
23.
24  /* set the field value */
25  (*env)->SetDoubleField(env, this_obj, id_salary, salary);
26.

```

## 12.4.2 访问静态域

访问静态域和访问非静态域类似。你要使用**GetStaticFieldID**和**GetStaticXxxField/ SetStaticXxxField**函数。它们几乎与非静态的情形一样，只有两个区别：

- 由于没有对象，必须使用**FindClass**代替**GetObjectClass**来获得类引用。
- 访问域时，要提供类而非实例对象。

例如，下面给出的是怎样得到**System.out**的引用的代码：

```

/* get the class */
jclass class_System = (*env)->FindClass(env, "java/lang/System");

/* get the field ID */
jfieldID id_out = (*env)->GetStaticFieldID(env, class_System, "out",
    "Ljava/io/PrintStream;");

/* get the field value */
jobject obj_out = (*env)->GetStaticObjectField(env, class_System, id_out);

```

## API 访问实例域

- **jfieldID GetFieldID(JNIEnv \*env, jclass cl, const char name[], const char fieldSignature[])**  
返回类中一个域的标识符。
- **Xxx GetXxxField(JNIEnv \*env, jobject obj, jfieldID id)**  
返回域的值。域类型Xxx是**Object**、**Boolean**、**Byte**、**Char**、**Short**、**Int**、**Long**、**Float**或**Double**之一。
- **void SetXxxField(JNIEnv \*env, jobject obj, jfieldID id, Xxx value)**  
把某个域设置为一个新值。域类型Xxx是**Object**、**Boolean**、**Byte**、**Char**、**Short**、**Int**、**Long**、**Float**或**Double**之一。
- **jfieldID GetStaticFieldID(JNIEnv \*env, jclass cl, const char name[], const char fieldSignature[])**  
返回某类型的一个静态域的标识符。
- **Xxx GetStaticXxxField(JNIEnv \*env, jclass cl, jfieldID id)**

返回某静态域的值。域类型Xxx是Object、Boolean、Byte、Char、Short、Int、Long、Float或Double之一。

• void SetStaticXxxField(JNIEnv \*env, jclass cl, jfieldID id, Xxx value)

把某个静态域设置为一个新值。域类型Xxx是Object、Boolean、Byte、Char、Short、Int、Long、Float或Double之一。

## 12.5 编码签名

为了访问实例域和调用Java编程语言中定义的方法，你必须学习“编入”数据类型的名称和方法签名的规则（方法签名描述了参数和该方法返回值的类型）。下面是编码方案：

|   |        |
|---|--------|
| B | byte   |
| C | char   |
| D | double |
| F | float  |
| I | int    |
| J | long   |

Lclassname; 类的类型

|   |         |
|---|---------|
| S | short   |
| V | void    |
| Z | boolean |

为了描述数组类型，要使用[。例如，一个字符串数组如下：

[Ljava/lang/String;

一个float[][]可以描述为：

[[F

要建立一个方法的完整签名，需要把括号内的参数类型都列出来，然后列出返回值类型。例如，一个接收两个整型参数并返回一个整数的方法编码为：

(II)I

在前一个例子中，我们使用的print方法的签名是：

(Ljava/lang/String;)V

也就是说，该方法接收一个字符串，返回值是void。

注意，在L表达式结尾处的分号是类型表达式的终止符，而不是参数之间的分隔符。例如，构造器：

Employee(java.lang.String, double, java.util.Date)

具有如下签名：

"(Ljava/lang/String;DLjava/util/Date;)V"

注意，在D和Ljava/util/Date;之间没有分隔符。另外要注意在这个编码方案中，必须用/代替.来分隔包和类名。结尾的V表示返回类型为void。即使对Java的构造器没有指定返回类型，也需要将V添加到虚拟机签名中。

**!** 提示：可以使用带有选项-s的javap命令来从类文件产生方法签名。例如，运行：

javap -s -private Employee

可以得到以下显示所有域和方法的输出：

```
Compiled from "Employee.java"
public class Employee extends java.lang.Object{
    private java.lang.String name;
        Signature: Ljava/lang/String;
    private double salary;
        Signature: D
    public Employee(java.lang.String, double);
        Signature: (Ljava/lang/String;D)V
    public native void raiseSalary(double);
        Signature: (D)V
    public void print();
        Signature: ()V
    static {};
        Signature: ()V
}
```



**注意：**没有任何理由强迫程序员使用这种编入方案来描述签名。本地调用机制的设计者可以非常容易地编写一个函数来读取Java编程语言风格的签名，比如void(int, java.lang.String)，并且将它们编码为他们喜欢的某种内部表示法。再者，使用编入签名使你能够分享接近虚拟机的编程奥秘。

## 12.6 调用Java方法

当然，Java编程语言的函数可以调用C函数，这正是本地方法要做的。我们能不能换一种方式呢？为什么我们要这么做？答案是，本地方法常常需要从传递给它的对象那里得到某种服务。我们首先介绍非静态方法如何进行这种操作，然后介绍静态方法如何进行这种操作。

### 12.6.1 实例方法

作为从本地代码调用Java方法的一个例子，我们先增强Printf类，给它增加一个与C函数fprintf类似的方法。也就是说，它能够在任意PrintWriter对象上打印一个字符串。下面是用Java编写的该方法的定义：

```
class Printf3
{
    public native static void fprintf(PrintWriter out, String s, double x);
    ...
}
```

我们首先把要打印的字符串编成一个String对象str，就像我们在sprint方法中已经实现的那样。然后，我们从实现本地方法的C函数调用PrintWriter类的print方法。

使用如下函数调用，你可以从C调用任何Java方法：

```
(*env)->CallXxxMethod(env, implicit parameter, methodID, explicit parameters)
```

根据方法的返回类型，用Void、Int、Object等来替换Xxx。就像你需要一个fieldID来访问某个对象的一个域，需要一个方法的ID来调用方法。你可以通过调用JNI函数GetMethodID，并且提供该类、方法的名字和方法签名来获得方法ID。

在我们的例子中，我们想要获得PrintWriter类的print方法的ID。如你在卷I第12章中所看

到的，`PrintWriter`类有几个名为print的重载方法。基于这个原因，你还必须提供一个字符串，描述你想要用的特定函数的参数和返回值。例如，我们想要用`void print(java.lang.String)`，正如前一节讲到的那样，我们必须把签名“编”为字符串"(Ljava/lang/String;)V"。

下面是进行方法调用的完整代码，有以下几个步骤：

- 1) 获取隐式参数的类。
- 2) 获取方法ID。
- 3) 进行调用。

```
/* get the class */
class_PrintWriter = (*env)->GetObjectClass(env, out);

/* get the method ID */
id_print = (*env)->GetMethodID(env, class_PrintWriter, "print", "(Ljava/lang/String;)V");

/* call the method */
(*env)->CallVoidMethod(env, out, id_print, str);
```

程序清单12-14和12-15给出了测试程序和`Printf3`类的Java代码。程序清单12-16包含了本地`fprintf`方法的C代码。



**注意：**数值型的方法ID和域ID在概念上和反射API中的Method和Field对象相似。你可以使用以下函数在两者间进行转换：

```
jobject ToReflectedMethod(JNIEnv* env, jclass class, jmethodID methodID);
    // returns Method object
jmethodID FromReflectedMethod(JNIEnv* env, jobject method);
jobject ToReflectedField(JNIEnv* env, jclass class, jfieldID fieldID);
    // returns Field object
jfieldID FromReflectedField(JNIEnv* env, jobject field);
```

## 12.6.2 静态方法

从本地方法调用静态方法与调用非静态方法类似。两者的差别是：

- 要用`GetStaticMethodID`和`CallStaticXxxMethod`函数。
- 当调用方法时，要提供类对象，而不是隐含的参数对象。

作为一个例子，让我们从本地方法调用以下静态方法：

```
System.getProperty("java.class.path")
```

这个调用的返回值是给出了当前类路径的字符串。

首先，我们必须找到要用的类。因为我们没有`System`类的对象可供使用，我们使用`FindClass`而非`GetObjectClass`：

```
jclass class_System = (*env)->FindClass(env, "java/lang/System");
```

接着，我们需要静态`getProperty`方法的ID。该方法的编码签名是：

```
"(Ljava/lang/String;)Ljava/lang/String;"
```

既然参数和返回值都是字符串。因此，我们这样获取方法ID：

```
jmethodID id_getProperty = (*env)->GetStaticMethodID(env, class_System, "getProperty",
    "(Ljava/lang/String;)Ljava/lang/String;");
```

最后，我们进行调用。注意，类对象被传递给了CallStaticObjectMethod函数。

```
jobject obj_ret = (*env)->CallStaticObjectMethod(env, class_System, id_getProperty,
    (*env)->NewStringUTF(env, "java.class.path"));
```

该方法的返回值是 jobject类型的。如果我们想要把它当作字符串操作，必须把它转型为 jstring：

```
jstring str_ret = (jstring) obj_ret;
```

**C++注意：**在C中，jstring和jclass类型同后面将要介绍的数组类型一样，都是与 jobject 等价的类型。因此，在C语言中，前面例子中的转型并不是严格必需的。但是在C++中，这些类型被定义为对拥有正确继承层次关系的“哑类”的指针。例如，将一个jstring不经过转型便赋给 jobject 在C++中是合法的，但是将 jobject 赋给 jstring 必须先转型。

### 12.6.3 构造器

本地方法可以通过调用构造器来创建新的Java对象。可以调用NewObject函数来调用构造器。

```
jobject obj_new = (*env)->NewObject(env, class, methodID, construction parameters);
```

可以通过指定方法名为"<init>"，并指定构造器（返回值为void）的编码签名，从GetMethodID函数中获取该调用必须的方法ID。例如，下面是本地方法创建FileOutputStream 对象的情形：

```
const char[] fileName = "...";
jstring str_fileName = (*env)->NewStringUTF(env, fileName);
jclass class_FileOutputStream = (*env)->FindClass(env, "java/io/FileOutputStream");
jmethodID id_FileOutputStream
    = (*env)->GetMethodID(env, class_FileOutputStream, "<init>", "(Ljava/lang/String;)V");
jobject obj_stream
    = (*env)->NewObject(env, class_FileOutputStream, id_FileOutputStream, str_fileName);
```

注意，构造器的签名接受一个java.lang.String类型的参数，返回类型为void。

### 12.6.4 替代方法调用

有若干种JNI函数的变体都可以从本地代码调用Java方法。它们没有我们已经讨论过的那些函数那么重要，但有偶尔也会很有用。

CallNonvirtualXxxMethod函数接收一个隐含参数、一个方法ID、一个类对象（必须对应于隐含参数的超类）和一个显式参数。这个函数将调用指定的类中的指定版本的方法，而不使用常规的动态调度机制。

所有调用函数都有后缀"A"和"V"的版本，用于接收数组中或va\_list中的显式参数（就像在C头文件中所定义的那样）。

#### 程序清单12-14 Printf3Test.java

```
1. import java.io.*;
2.
3. /**
4. * @version 1.10 1997-07-01
5. * @author Cay Horstmann
```

```
6. */
7. class Printf3Test
8. {
9.     public static void main(String[] args)
10.    {
11.        double price = 44.95;
12.        double tax = 7.75;
13.        double amountDue = price + (tax / 100);
14.        PrintWriter out = new PrintWriter(System.out);
15.        Printf3.fprintf(out, "Amount due = %8.2f\n", amountDue);
16.        out.flush();
17.    }
18. }
```

**程序清单12-15 Printf3.java**

```
1. import java.io.*;
2.
3. /**
4. * @version 1.10 1997-07-01
5. * @author Cay Horstmann
6. */
7. class Printf3
8. {
9.     public static native void fprintf(PrintWriter out, String format, double x);
10.
11.     static
12.     {
13.         System.loadLibrary("Printf3");
14.     }
15. }
```

**程序清单12-16 Printf3.c**

```
1. /**
2. * @version 1.10 1997-07-01
3. * @author Cay Horstmann
4. */
5.
6. #include "Printf3.h"
7. #include <string.h>
8. #include <stdlib.h>
9. #include <float.h>
10.
11. /**
12. * @param format a string containing a printf format specifier
13. * (such as "%8.2f"). Substrings "%" are skipped.
14. * @return a pointer to the format specifier (skipping the '%')
15. * or NULL if there wasn't a unique format specifier
16. */
17. char* find_format(const char format[])
18. {
19.     char* p;
20.     char* q;
21.
22.     p = strchr(format, '%');
```

```
23.     while (p != NULL && *(p + 1) == '%') /* skip %% */
24.         p = strchr(p + 2, '%');
25.     if (p == NULL) return NULL;
26.     /* now check that % is unique */
27.     p++;
28.     q = strchr(p, '%');
29.     while (q != NULL && *(q + 1) == '%') /* skip %% */
30.         q = strchr(q + 2, '%');
31.     if (q != NULL) return NULL; /* % not unique */
32.     q = p + strspn(p, " -0#+"); /* skip past flags */
33.     q += strspn(q, "0123456789"); /* skip past field width */
34.     if (*q == '.') { q++; q += strspn(q, "0123456789"); }
35.         /* skip past precision */
36.     if (strchr("eEfFgG", *q) == NULL) return NULL;
37.         /* not a floating-point format */
38.     return p;
39. }
40.
41. JNIEXPORT void JNICALL Java_Printf3_fprint(JNIEnv* env, jclass c,
42.     jobject out, jstring format, jdouble x)
43. {
44.     const char* cformat;
45.     char* fmt;
46.     jstring str;
47.     jclass class_PrintWriter;
48.     jmethodID id_print;
49.
50.     cformat = (*env)->GetStringUTFChars(env, format, NULL);
51.     fmt = find_format(cformat);
52.     if (fmt == NULL)
53.         str = format;
54.     else
55.     {
56.         char* cstr;
57.         int width = atoi(fmt);
58.         if (width == 0) width = DBL_DIG + 10;
59.         cstr = (char*) malloc(strlen(cformat) + width);
60.         sprintf(cstr, cformat, x);
61.         str = (*env)->NewStringUTF(env, cstr);
62.         free(cstr);
63.     }
64.     (*env)->ReleaseStringUTFChars(env, format, cformat);
65.
66.     /* now call ps.print(str) */
67.
68.     /* get the class */
69.     class_PrintWriter = (*env)->GetObjectClass(env, out);
70.
71.     /* get the method ID */
72.     id_print = (*env)->GetMethodID(env, class_PrintWriter, "print", "(Ljava/lang/String;)V");
73.
74.     /* call the method */
75.     (*env)->CallVoidMethod(env, out, id_print, str);
76. }
```

**API 执行Java方法**

- `jmethodID GetMethodID(JNIEnv *env, jclass cl, const char name[], const char methodSignature[])`

返回类中某个方法的标识符。

- `Xxx CallXxxMethod(JNIEnv *env, jobject obj, jmethodID id, args)`
- `Xxx CallXxxMethodA(JNIEnv *env, jobject obj, jmethodID id, jvalue args[])`
- `Xxx CallXxxMethodV(JNIEnv *env, jobject obj, jmethodID id, va_list args)`

调用一个方法。返回类型Xxx是Object、Boolean、Byte、Char、Short、Int、Long、Float或Double之一。第一个函数有一个可变的参数个数，只要把方法参数附加到方法ID之后即可。第二个函数接受jvalue数组中的方法参数，其中jvalue是一个并集，定义如下：

```
typedef union jvalue
{
    jboolean z;
    jbyte b;
    jchar c;
    jshort s;
    jint i;
    jlong j;
    jfloat f;
    jdouble d;
    jobject l;
} jvalue;
```

第三个函数接收C头文件stdarg.h中定义的va\_list中的方法参数。

- `Xxx CallNonvirtualXxxMethod(JNIEnv *env, jobject obj, jclass cl, jmethodID id, args)`
- `Xxx CallNonvirtualXxxMethodA(JNIEnv *env, jobject obj, jclass cl, jmethodID id, args[])`
- `Xxx CallNonvirtualXxxMethodV(JNIEnv *env, jobject obj, jclass cl, jmethodID id, va_list args)`

调用一个方法，而不需要动态调度。返回类型Xxx是Object、Boolean、Byte、Char、Short、Int、Long、Float或Double之一。第一个函数有一个可变的参数个数，只要把方法参数附加到方法ID之后即可。第二个函数接受jvalue数组中的方法参数。第三个函数接受C头文件stdarg.h中定义的va\_list中的方法参数。

- `jmethodID GetStaticMethodID(JNIEnv *env, jclass cl, const char name[], const char methodSignature[])`

返回类的某个静态方法的标识符。

- `Xxx CallStaticXxxMethod(JNIEnv *env, jclass cl, jmethodID id, args)`
- `Xxx CallStaticXxxMethodA(JNIEnv *env, jclass cl, jmethodID id, jvalue args[])`
- `Xxx CallStaticXxxMethodV(JNIEnv *env, jclass cl, jmethodID id, va_list args)`

调用一个静态方法。返回类型Xxx是Object、Boolean、Byte、Char、Short、Int、Long、

`Float`或`Double`之一。第一个函数有一个可变的参数个数，只要把方法参数附加到方法ID之后即可。第二个函数接受`jvalue`数组中的方法参数。第三个函数接收C头文件`stdarg.h`中定义的`va_list`中的方法参数。

- `jobject NewObject(JNIEnv *env, jclass cl, jmethodID id, args)`
- `jobject NewObjectA(JNIEnv *env, jclass cl, jmethodID id, jvalue args[])`
- `jobject NewObjectV(JNIEnv *env, jclass cl, jmethodID id, va_list args)`

调用构造器。函数ID从带有函数名为"`<init>`"和返回类型为`void`的`GetMethodID`获取。第一个函数有一个可变的参数个数，只要把方法参数附加到方法ID之后即可。第二个函数接收`jvalue`数组中的方法参数。第三个函数接收C头文件`stdarg.h`中定义的`va_list`中的方法参数。

## 12.7 访问数组元素

Java编程语言的所有数组类型都有相对应的C语言类型，见表12-2。

表12-2 Java数组类型和C数组类型之间的对应关系

| Java数组类型               | C数组类型                      | Java数组类型              | C数组类型                     |
|------------------------|----------------------------|-----------------------|---------------------------|
| <code>boolean[]</code> | <code>jbooleanArray</code> | <code>long[]</code>   | <code>jlongArray</code>   |
| <code>byte[]</code>    | <code>jbyteArray</code>    | <code>float[]</code>  | <code>jfloatArray</code>  |
| <code>char[]</code>    | <code>jcharArray</code>    | <code>double[]</code> | <code>jdoubleArray</code> |
| <code>int[]</code>     | <code>jintArray</code>     | <code>Object[]</code> | <code>jobjectArray</code> |
| <code>short[]</code>   | <code>jshortArray</code>   |                       |                           |

**C++注意：**在C中，所有这些数组类型实际上都是`jobject`的同义类型。然而，在C++中它们被安排在如图12-3所示的继承层次结构中。`jarray`类型表示一个通用数组。

`GetArrayLength`函数返回数组的长度。

```
jarray array = . . .;
jsize length = (*env)->GetArrayLength(env, array);
```

怎样访问数组元素取决于数组中存储的是对象还是原始类型的数据（`bool`、`char`或数值类型）。可以通过`GetObjectArrayElement`和`SetObjectArrayElement`方法访问对象数组的元素。

```
jobjectArray array = . . .;
int i, j;
jobject x = (*env)->GetObjectArrayElement(env, array, i);
(*env)->SetObjectArrayElement(env, array, j, x);
```

这个方法虽然简单，但是效率明显低下，你可能想要直接访问数组元素，特别是在进行向量或矩阵计算时更加明显。

`GetXxxArrayElements`函数返回一个指向数组起始元素的C指针。与普通的字符串一样，当你不再需要该指针时，必须记得要调用`ReleaseXxxArrayElements`函数通知虚拟机。这里，类型`Xxx`必须是原始类型，也就是说，不能是`Object`。这样就可以直接读写数组元素了。另一方面，由于指针可能会指向一个副本，只有调用相应的`ReleaseXxxArrayElements`函数时，你

所做的改变才能保证在原始数组里得到反映。

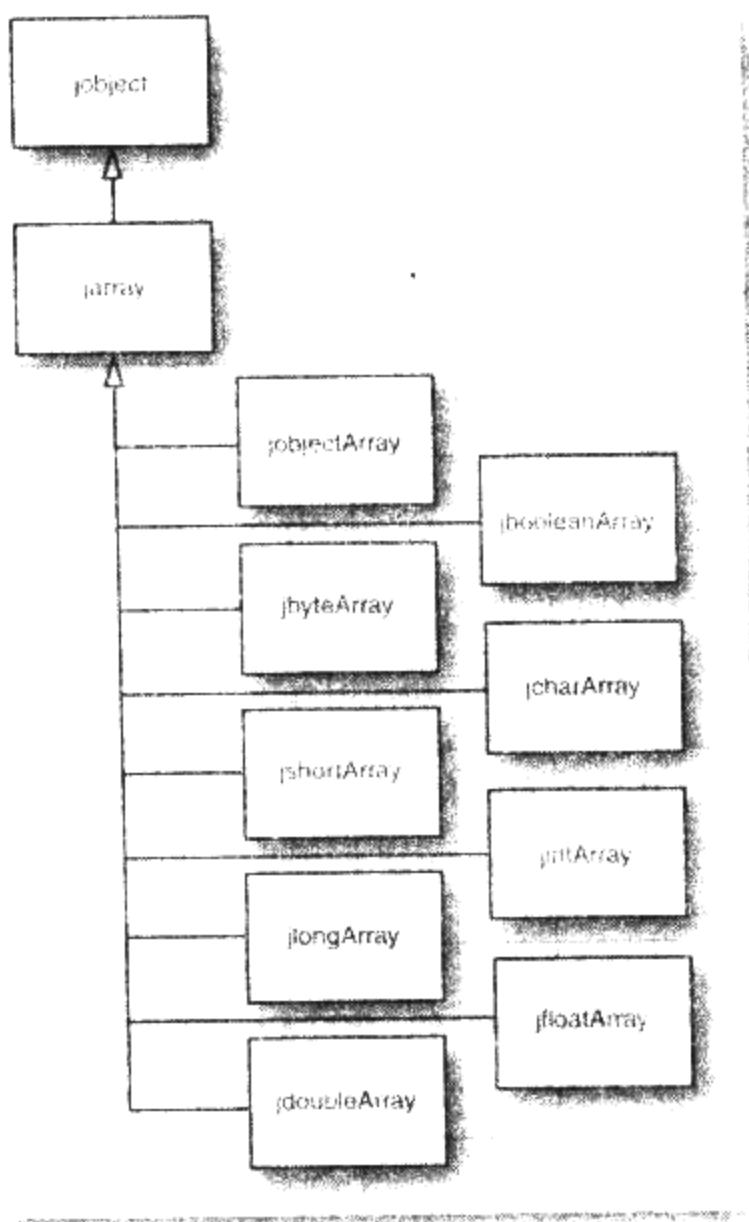


图12-3 数组类型的继承层次结构



**注意：**通过把一个指向jboolean变量的指针作为第三个参数传递给`GetXxxArrayElements`方法，就可以发现一个数组是否是副本了。如果是副本，则该变量被`JNI_TRUE`填充。如果你对这个信息不感兴趣，传一个空指针即可。

下面是对`double`类型数组中的所有元素乘以一个常量的示例代码。我们获取一个Java数组的C指针`a`，并用`a[i]`访问各个元素。

```

jdoubleArray array_a = . . .;
double scaleFactor = . . .;
double* a = (*env)->GetDoubleArrayElements(env, array_a, NULL);
for (i = 0; i < (*env)->GetArrayLength(env, array_a); i++)
    a[i] = a[i] * scaleFactor;
(*env)->ReleaseDoubleArrayElements(env, array_a, a, 0);
  
```

虚拟机是否确实需要对数组进行拷贝取决于它是如何分配数组和如何进行垃圾回收的。有些“拷贝”型的垃圾回收器例行地移动对象，并更新对象引用。该策略与将数组锁定在特定位置是不兼容的，因为回收器不能更新本地代码中的指针值。

注意：Sun公司的JVM实现中，boolean数组是用包装的32位字数组表示的。GetBooleanArrayElements方法能将它们复制到未包装的jboolean值的数组中。

如果要访问一个大数组的多个元素，可以用GetXxxArrayRegion和SetXxxArrayRegion方法，它能把一定范围内的元素从Java数组复制到C数组中或从C数组复制到Java数组中。

可以用NewXxxArray函数在本地方法中创建新的Java数组。要创建新的对象数组，需要指定长度、数组元素的类型和所有元素的初始值（典型的是NULL）。下面是一个例子。

```
jclass class_Employee = (*env)->FindClass(env, "Employee");
jobjectArray array_e = (*env)->NewObjectArray(env, 100, class_Employee, NULL);
```

原始类型的数组要简单一些。只需提供数组长度。

```
jdoubleArray array_d = (*env)->NewDoubleArray(env, 100);
```

该数组被0填充。

注意：Java SE 1.4 在JNI API中增加了3个方法：

```
jobject NewDirectByteBuffer(JNIEnv* env, void* address, jlong capacity)
void* GetDirectBufferAddress(JNIEnv* env, jobject buf)
jlong GetDirectBufferCapacity(JNIEnv* env, jobject buf)
```

java.nio包中使用了直接缓存来支持更高效的输入输出操作，并尽可能减少本地和Java数组之间的复制操作。

## 操作Java数组

- `jsize GetArrayLength(JNIEnv *env, jarray array)`  
返回数组中的元素个数。
- `jobject GetObjectArrayElement(JNIEnv *env, jobjectArray array, jsize index)`  
返回数组元素的值。
- `void SetObjectArrayElement(JNIEnv *env, jobjectArray array, jsize index, jobject value)`  
将数组元素设为新值。
- `Xxx* GetXxxArrayElements(JNIEnv *env, jarray array, jboolean* isCopy)`  
产生一个指向Java数组元素的C指针。域类型Xxx是Boolean、Byte、Char、Short、Int、Long、Float或Double之一。指针不再使用时，该指针必须传递给ReleaseXxxArrayElements。如果进行了复制，isCopy可以是NULL或者指向JNI\_TRUE填充的jboolean；否则，指向JNI\_FALSE填充的jboolean。
- `void ReleaseXxxArrayElements(JNIEnv *env, jarray array, Xxx elems[], jint mode)`  
通知虚拟机通过GetXxxArrayElements获得的一个指针已经不再需要了。Mode是0（更新数组元素后释放elems缓存）、JNI\_COMMIT（更新数组元素后不释放elems缓存）或JNI\_ABORT（不更新数组元素便释放elems缓存）之一。
- `void GetXxxArrayRegion(JNIEnv *env, jarray array, jint start, jint length, Xxx`

```
elems[])
```

将Java数组的元素复制到C数组中。域类型Xxx是Boolean、Byte、Char、Short、Int、Long、Float或Double之一。

```
• void SetXxxArrayRegion(JNIEnv *env, jarray array, jint start, jint length, Xxx  
elems[])
```

将C数组的元素复制到Java数组中。域类型Xxx是Boolean、Byte、Char、Short、Int、Long、Float或Double之一。

## 12.8 错误处理

在Java编程语言中，使用本地方法对于程序来说是要冒很大的安全风险的。C的运行期系统对数组越界错误、不良指针造成的间接错误不提供任何防护。所以，对于本地方法的程序员来说，处理所有的出错条件以保持Java平台的完整性显得格外重要。尤其是，当你的本地方法诊断出一个它无法解决的问题时，那么它应该将此问题报告给Java虚拟机。然后，在这种情况下，很自然地会抛出一个异常。然而，C语言没有异常，必须调用Throw或ThrowNew函数来创建一个新的异常对象。当本地方法退出时，Java虚拟机就会抛出该异常。

要使用Throw函数，需要调用NewObject来创建一个Throwable子类的对象。例如，下面我们分配了一个EOFException对象，然后将它抛出。

```
jclass class_EOFException = (*env)->FindClass(env, "java/io/EOFException");  
jmethodID id_EOFException = (*env)->GetMethodID(env, class_EOFException, "<init>", "()V");  
/* ID of default constructor */  
jthrowable obj_exc = (*env)->NewObject(env, class_EOFException, id_EOFException);  
(*env)->Throw(env, obj_exc);
```

通常调用ThrowNew会更加方便，只需提供一个类和一个“改良UTF-8”字节序列，该函数就会构建一个异常对象。

```
(*env)->ThrowNew(env, (*env)->FindClass(env, "java/io/EOFException"), "Unexpected end of file");
```

Throw和ThrowNew都仅仅登记异常，它们不会中断本地方法的控制流。只有当该方法返回时，Java虚拟机才会抛出异常。所以，每一个对Throw和ThrowNew的调用语句之后总是紧跟着返回语句。



**C++注意：**如果用C++实现本地方法，就无法用你的C++代码抛出Java异常。在C++绑定中，可以实现一个在C++异常和Java异常之间的转换。然而，目前还没有实现。需要在本地方法中使用Throw或ThrowNew函数来抛出Java异常，并且要确保你的本地方法不抛出C++异常。

通常，本地代码不需要考虑捕获Java异常。但是，当本地方法调用Java方法时，该方法可能会抛出异常。而且，一些JNI函数也会抛出异常。例如，如果索引越界，SetObjectArrayElement方法会抛出一个ArrayIndexOutOfBoundsException异常，如果所存储的对象的类不是数组元素类的子类，该方法会抛出一个ArrayStoreException异常。在这种情况下，本地方法应该调用

`ExceptionOccurred`方法来确认是否有异常抛出。如果没有任何异常被挂起，则下面的调用：

```
jthrowable obj_exc = (*env)->ExceptionOccurred(env);
```

将返回NULL。否则，返回一个当前异常对象的引用。如果只要检查是否有异常抛出，而不要获得异常对象的引用，那么使用：

```
jboolean occurred = (*env)->ExceptionCheck(env);
```

通常，有异常出现时，本地方法只需要返回。那样，虚拟机就会将该异常传送给Java代码。但是，本地方法也可以分析异常对象，确定它是否能够处理该异常。如果能够处理，那么必须调用下面的函数来关闭该异常：

```
(*env)->ExceptionClear(env);
```

在我们的例子中，我们实现了`fprint`本地方法，这是基于该方法就适合编写为本地方法的假设而实现的。下面是我们抛出的异常：

- 如果格式字符串是NULL，则抛出`NullPointerException`异常。
- 如果格式字符串不含适合打印`double`所需的%说明符，则抛出`IllegalArgumentException`异常。
- 如果调用`malloc`失败，则抛出`OutOfMemoryError`异常。

最后，为了说明本地方法调用Java方法时，怎样检查异常，我们将一个字符串发送给数据流，一次一个字符，并且在每次调用Java方法后调用`ExceptionOccurred`。程序清单12-17给出了本地方法的代码，程序清单12-18包含了含有本地方法的类的定义。注意，在调用`PrintWriter.printit`出现异常时，本地方法并不会立即终止执行，它会首先释放`cstr`缓存。当本地方法返回时，虚拟机再次抛出异常。程序清单12-19的测试程序说明了当格式字符串不正确时，本地方法是如何抛出异常的。

### 程序清单12-17 Printf4.c

```

1. /**
2.  @version 1.10 1997-07-01
3.  @author Cay Horstmann
4. */
5.
6. #include "Printf4.h"
7. #include <string.h>
8. #include <stdlib.h>
9. #include <float.h>
10.
11. /**
12.  @param format a string containing a printf format specifier
13.  (such as "%8.2f"). Substrings "%%" are skipped.
14.  @return a pointer to the format specifier (skipping the '%')
15.  or NULL if there wasn't a unique format specifier
16. */
17. char* find_format(const char format[])
18. {
19.     char* p;
20.     char* q;
21.
22.     p = strchr(format, '%');

```

```

23.     while (p != NULL && *(p + 1) == '%') /* skip %% */
24.         p = strchr(p + 2, '%');
25.     if (p == NULL) return NULL;
26.     /* now check that % is unique */
27.     p++;
28.     q = strchr(p, '%');
29.     while (q != NULL && *(q + 1) == '%') /* skip %% */
30.         q = strchr(q + 2, '%');
31.     if (q != NULL) return NULL; /* % not unique */
32.     q = p + strspn(p, " -0#+"); /* skip past flags */
33.     q += strspn(q, "0123456789"); /* skip past field width */
34.     if (*q == '.') { q++; q += strspn(q, "0123456789"); }
35.         /* skip past precision */
36.     if (strchr("eEfFgG", *q) == NULL) return NULL;
37.         /* not a floating-point format */
38.     return p;
39. }
40.
41 JNIEXPORT void JNICALL Java_Printf4_fprint(JNIEnv* env, jclass cl,
42     jobject out, jstring format, jdouble x)
43. {
44     const char* cformat;
45     char* fmt;
46     jclass class_PrintWriter;
47     jmethodID id_print;
48     char* cstr;
49     int width;
50     int i;
51.
52     if (format == NULL)
53     {
54         (*env)->ThrowNew(env,
55             (*env)->FindClass(env,
56                 "java/lang/NullPointerException"),
57                 "Printf4.fprint: format is null");
58         return;
59     }
60.
61     cformat = (*env)->GetStringUTFChars(env, format, NULL);
62     fmt = find_format(cformat);
63.
64     if (fmt == NULL)
65     {
66         (*env)->ThrowNew(env,
67             (*env)->FindClass(env,
68                 "java/lang/IllegalArgumentException"),
69                 "Printf4.fprint: format is invalid");
70         return;
71     }
72.
73     width = atoi(fmt);
74     if (width == 0) width = DBL_DIG + 10;
75     cstr = (char*)malloc(strlen(cformat) + width);
76.
77     if (cstr == NULL)
78     {
79         (*env)->ThrowNew(env,

```

```

80.     (*env)->FindClass(env, "java/lang/OutOfMemoryError"),
81.     "Printf4.fprint: malloc failed");
82.     return;
83. }
84.
85. sprintf(cstr, cformat, x);
86.
87. (*env)->ReleaseStringUTFChars(env, format, cformat);
88.
89. /* now call ps.print(str) */
90.
91. /* get the class */
92. class_PrintWriter = (*env)->GetObjectClass(env, out);
93.
94. /* get the method ID */
95. id_print = (*env)->GetMethodID(env, class_PrintWriter, "print", "(C)V");
96.
97. /* call the method */
98. for (i = 0; cstr[i] != 0 && !(*env)->ExceptionOccurred(env); i++)
99.     (*env)->CallVoidMethod(env, out, id_print, cstr[i]);
100.
101. free(cstr);
102. }
```

### 程序清单12-18 Printf4.java

```

1. import java.io.*;
2.
3. /**
4. * @version 1.10 1997-07-01
5. * @author Cay Horstmann
6. */
7. class Printf4
8. {
9.     public static native void fprintf(PrintWriter ps, String format, double x);
10.
11.     static
12.     {
13.         System.loadLibrary("Printf4");
14.     }
15. }
```

### 程序清单12-19 Printf4Test.java

```

1. import java.io.*;
2.
3. /**
4. * @version 1.10 1997-07-01
5. * @author Cay Horstmann
6. */
7. class Printf4Test
8. {
9.     public static void main(String[] args)
10.    {
11.        double price = 44.95;
12.        double tax = 7.75;
13.        double amountDue = price + (1 + tax / 100);
```

```
14     PrintWriter out = new PrintWriter(System.out);
15     /* This call will throw an exception--note the %% */
16     Printf4.fprintf(out, "Amount due = %%8.2f\n", amountDue);
17     out.flush();
18 }
19 }
```

### API 处理Java异常

- **jint Throw(JNIEnv \*env, jthrowable obj)**

准备一个在本地代码退出时抛出的异常。成功时返回0，失败时返回一个负值。

- **jint ThrowNew(JNIEnv \*env, jclass cl, const char msg[])**

准备一个在本地代码退出时抛出的类型为cl的异常。成功时返回0，失败时返回一个负值。  
msg是表示异常对象的String构造参数的“改良UTF-8”字节序列

- **jthrowable ExceptionOccurred(JNIEnv \*env)**

如果有异常挂起，则返回该异常对象，否则返回NULL。

- **jboolean ExceptionCheck(JNIEnv \*env)**

如果有异常挂起，则返回true。

- **void ExceptionClear(JNIEnv \*env)**

清空挂起的异常。

## 12.9 使用调用API

到现在为止，我们主要讨论的都是用Java编程语言编写的进行了一些C调用的程序，大概是因为C的运行速度更快一些，或者允许访问一些Java平台无法访问的功能。假设正处在相反的情况下，你有一个C或者C++的程序，并且想要调用一些Java代码。调用API (invocation API)使你能够把Java虚拟机嵌入到C或者C++程序中。下面是你初始化虚拟机所需的基本代码。

```
JavaVMOption options[1];
JavaVMInitArgs vm_args;
JavaVM *jvm;
JNIEnv *env;

options[0].optionString = "-Djava.class.path=.";

memset(&vm_args, 0, sizeof(vm_args));
vm_args.version = JNI_VERSION_1_2;
vm_args.nOptions = 1;
vm_args.options = options;

JNI_CreateJavaVM(&jvm, (void**) &env, &vm_args);
```

对JNI\_CreateJavaVM的调用将创建虚拟机，并且将指针jvm指向虚拟机，指针env指向执行环境。

可以给虚拟机提供任意数目的选项，这只需增加选项数组的大小和vm\_args.nOptions的值。

例如，

```
options[i].optionString = "-Djava.compiler=NONE";
```

可以钝化实时编译器。

**!** 提示：当你陷入麻烦导致程序崩溃，从而不能初始化JVM或者不能装载你的类时，请打开JNI调试模式。设置一个选项如下：

```
options[i].optionString = "-verbose:jni";
```

你会看到一系列说明JVM初始化进程的消息。如果看不到你装载的类，请检查你的路径和类路径的设置。

一旦设置完虚拟机，就可以如前面章节介绍的那样调用Java方法了。只要按常规方法使用env指针即可。

只有在调用API中的其他函数时，才需要jvm指针。目前，只有四个这样的函数。最重要的一个是终止虚拟机的函数：

```
(*jvm)->DestroyJavaVM(jvm);
```

遗憾的是，在Windows下，动态链接到jre/bin/client/jvm.dll中的JNI\_CreateJavaVM函数变得非常困难，因为Vista改变了链接规则，而Sun的类库仍旧依赖于旧版本的C运行时类库。我们的示范程序通过手工加载该类库解决了这个问题，这种方式与Java程序所使用的方式一样，请参阅JDK中的src.jar文件里的launcher/java\_md.c文件。

程序清单12-20的C程序设置了虚拟机，然后调用Welcome类的main方法，这在卷I第2章中讨论过了（在开始启用测试程序之前，务必编译Welcome.java文件）。

### 程序清单12-20 InvocationTest.c

```

1. /**
2.  * @version 1.20 2007-10-26
3.  * @author Cay Horstmann
4. */
5.
6. #include <jni.h>
7. #include <stdlib.h>
8.
9. #ifdef _WINDOWS
10.
11. #include <windows.h>
12. static HINSTANCE loadJVMLibrary(void);
13. typedef jint (JNICALL *CreateJavaVM_t)(JavaVM **, void **, JavaVMInitArgs *);
14.
15. #endif
16.
17. int main()
18. {
19.     JavaVMOption options[2];
20.     JavaVMInitArgs vm_args;
21.     JavaVM *jvm;
22.     JNIEnv *env;
23.     long status;
24.
```

```
25. jclass class_Welcome;
26. jclass class_String;
27. jobjectArray args;
28. jmethodID id_main;
29.
30. #ifdef _WINDOWS
31.     HINSTANCE hjvmlib;
32.     CreateJavaVM_t createJavaVM;
33. #endif
34.
35.     options[0].optionString = "-Djava.class.path=.";
36.
37.     memset(&vm_args, 0, sizeof(vm_args));
38.     vm_args.version = JNI_VERSION_1_2;
39.     vm_args.nOptions = 1;
40.     vm_args.options = options;
41.
42.
43. #ifdef _WINDOWS
44.     hjvmlib = LoadJVMLibrary();
45.     createJavaVM = (CreateJavaVM_t) GetProcAddress(hjvmlib, "JNI_CreateJavaVM");
46.     status = (*createJavaVM)(&jvm, (void **) &env, &vm_args);
47. #else
48.     status = JNI_CreateJavaVM(&jvm, (void **) &env, &vm_args);
49. #endif
50.
51.     if (status == JNI_ERR)
52.     {
53.         fprintf(stderr, "Error creating VM\n");
54.         return 1;
55.     }
56.
57.     class_Welcome = (*env)->FindClass(env, "Welcome");
58.     id_main = (*env)->GetStaticMethodID(env, class_Welcome, "main", "([Ljava/lang/String;)V");
59.
60.     class_String = (*env)->FindClass(env, "java/lang/String");
61.     args = (*env)->NewObjectArray(env, 0, class_String, NULL);
62.     (*env)->CallStaticVoidMethod(env, class_Welcome, id_main, args);
63.
64.     (*jvm)->DestroyJavaVM(jvm);
65.
66.     return 0;
67. }
68.
69. #ifdef _WINDOWS
70.
71. static int GetStringFromRegistry(HKEY key, const char *name, char *buf, jint bufsize)
72. {
73.     DWORD type, size;
74.
75.     return RegQueryValueEx(key, name, 0, &type, 0, &size) == 0
76.         && type == REG_SZ
77.         && size < (unsigned int) bufsize
78.         && RegQueryValueEx(key, name, 0, 0, buf, &size) == 0;
79. }
80.
81. static void GetPublicJREHome(char *buf, jint bufsize)
```

```
82. {
83.     HKEY key, subkey;
84.     char version[MAX_PATH];
85.
86.     /* Find the current version of the JRE */
87.     char *JRE_KEY = "Software\\JavaSoft\\Java Runtime Environment";
88.     if (RegOpenKeyEx(HKEY_LOCAL_MACHINE, JRE_KEY, 0, KEY_READ, &key) != 0)
89.     {
90.         fprintf(stderr, "Error opening registry key '%s'\n", JRE_KEY);
91.         exit(1);
92.     }
93.
94.     if (!GetStringFromRegistry(key, "CurrentVersion", version, sizeof(version)))
95.     {
96.         fprintf(stderr, "Failed reading value of registry key:\n\t%s\\CurrentVersion\n", JRE_KEY);
97.         RegCloseKey(key);
98.         exit(1);
99.     }
100.
101.    /* Find directory where the current version is installed. */
102.    if (RegOpenKeyEx(key, version, 0, KEY_READ, &subkey) != 0)
103.    {
104.        fprintf(stderr, "Error opening registry key '%s\\%s'\n", JRE_KEY, version);
105.        RegCloseKey(key);
106.        exit(1);
107.    }
108.
109.    if (!GetStringFromRegistry(subkey, "JavaHome", buf, bufsize))
110.    {
111.        fprintf(stderr, "Failed reading value of registry key:\n\t%s\\%s\\JavaHome\n",
112.                JRE_KEY, version);
113.        RegCloseKey(key);
114.        RegCloseKey(subkey);
115.        exit(1);
116.    }
117.
118.    RegCloseKey(key);
119.    RegCloseKey(subkey);
120. }
121.
122. static HINSTANCE loadJVMLibrary(void)
123. {
124.     HINSTANCE h1, h2;
125.     char msวดll[MAX_PATH];
126.     char javadll[MAX_PATH];
127.     GetPublicJREHome(msวดll, MAX_PATH);
128.     strcpy(javadll, msวดll);
129.     strncat(msวดll, "\\bin\\msvcr71.dll", MAX_PATH - strlen(msวดll));
130.     msวดll[MAX_PATH - 1] = '\\0';
131.     strncat(javadll, "\\bin\\client\\jvm.dll", MAX_PATH - strlen(javadll));
132.     javadll[MAX_PATH - 1] = '\\0';
133.
134.     h1 = LoadLibrary(msวดll);
135.     if (h1 == NULL)
136.     {
137.         fprintf(stderr, "Can't load library msvcr71.dll\n");
138.         exit(1);
139.     }
```

```
140.  
141     h2 = LoadLibrary(javadll);  
142     if (h2 == NULL)  
143     {  
144         fprintf(stderr, "Can't load library jvm.dll\n");  
145         exit(1);  
146     }  
147     return h2;  
148 }  
149.  
150 #endif
```

要在Linux下编译该程序，请用：

```
gcc -I jdk/include -I jdk/include/linux -o InvocationTest  
-L jdk/jre/lib/i386/client -ljvm InvocationTest.c
```

在Solaris下，请用：

```
cc -I jdk/include -I jdk/include/solaris -o InvocationTest  
-L jdk/jre/lib/sparc -ljvm InvocationTest.c
```

在Windows下用微软的C编译器时，请用下面的命令行：

```
c1 -D_WINDOWS -I jdk\include -I jdk\include\win32 InvocationTest.c jdk\lib\jvm.lib advapi32.lib
```

需要确保INCLUDE和LIB环境变量包含了Windows API头文件和库文件的路径。

用Cygwin时，用下面的语句进行编译：

```
gcc -D_WINDOWS -mno-cygwin -I jdk\include -I jdk\include\win32 -D_int64="long long"  
-I c:\cygwin\usr\include\w32api -o InvocationTest
```

在Linux/UNIX下运行该程序之前，需要确保LD\_LIBRARY\_PATH包含了共享类库的目录。例如，如果使用Linux上的bash命令行，则需要执行下面的命令：

```
export LD_LIBRARY_PATH=jdk/jre/lib/i386/client:$LD_LIBRARY_PATH
```



### 调用API函数

- **jint JNI\_CreateJavaVM(JavaVM\*\* p\_jvm, void\*\* p\_env, JavaVMInitArgs\* vm\_args)**

初始化Java虚拟机。如果成功，则返回0，否则返回JNI\_ERR。

参数：p\_jvm 填入指向调用API函数表的指针

p\_env 填入指向JNI函数表的指针

vm\_args 虚拟机参数

- **jint DestroyJavaVM(JavaVM\* jvm)**

销毁虚拟机。如果成功，则返回0，否则返回一个负值。该函数必须通过一个虚拟机指针调用。例如，(\*jvm)->DestroyJavaVM(jvm)。

## 12.10 完整的示例：访问Windows注册表

在本节中，我们介绍一个完整的可运行的例子，它涵盖了我们在本章讨论的所有内容：使用带有字符串、数组和对象的本地方法，构造器调用和错误处理。我们将展示如何用Java平台包装器来包装普通的基于C的API子集，用于进行Windows注册表操作。当然，由于Windows的具体特性，使用Windows注册表的程序天生就不可移植。基于这个原因，标准的Java库不支持

注册表，所以使用本地方法访问注册表是有意义的。

### 12.10.1 Windows注册表概述

Windows注册表是一个存放Windows操作系统和应用程序的配置信息的数据仓库。它提供了对系统和应用程序参数的单点管理和备份。

其不足的方面是，注册表的错误也是单点的。如果你弄乱了注册表，你的电脑就会出故障，甚至无法启动。Java配置API（preferences API）是一个更好的解决方案。更多信息请参见第1卷第10章。我们使用注册表只是为了说明怎样把非平凡的本地API包装成Java类。

检查注册表的主要工具是注册表编辑器。由于可能存在幼稚而狂热的用户，所以没有配备任何图标来启动注册表编辑器。你必须启动DOS shell（或打开开始→运行对话框）然后键入regedit。图12-4给出了一个运行中的注册表编辑器。

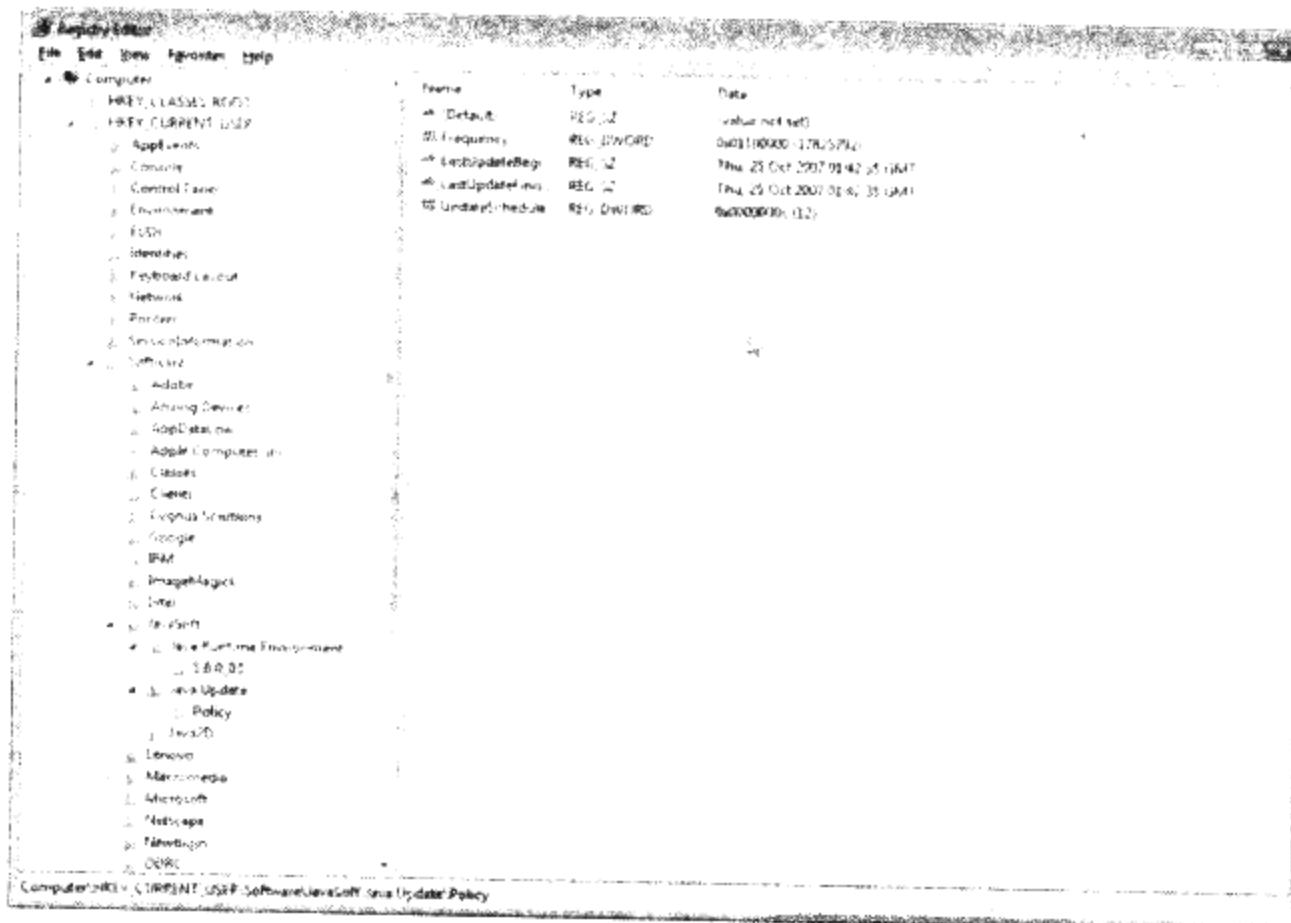


图12-4 注册表编辑器

左边是树形结构排列的注册表键。请注意，每个键都以HKEY节点开始，如：

```
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
```

右边是与特定键关联的名/值对。例如，如果你安装了Java SE 6，那么键：

```
HKEY_LOCAL_MACHINE\Software\JavaSoft\Java Runtime Environment
```

就包含下面这样的名值对：

```
CurrentVersion="1.6.0_03"
```

在本例中，值是字符串。值也可以是整数或字节数组。

### 12.10.2 访问注册表的Java平台接口

我们实现了一个从Java代码访问注册表的简单接口，然后用本地代码实现这个接口。我们的接口只允许几个注册表操作，为了保持较小的代码规模，我们省略了其他重要的操作，如：添加、删除和枚举注册表键（添加剩余的这些注册表API函数是很容易的）。

即使使用我们提供的受限的子集，你也可以：

- 枚举某个键中存储的所有名字；
- 读出用某个名字存储的值；
- 设置用某个名字存储的值。

下面是封装注册表键的Java平台类。

```
public class Win32RegKey
{
    public Win32RegKey(int theRoot, String thePath) { . . . }
    public Enumeration names() { . . . }
    public native Object getValue(String name);
    public native void setValue(String name, Object value);

    public static final int HKEY_CLASSES_ROOT = 0x80000000;
    public static final int HKEY_CURRENT_USER = 0x80000001;
    public static final int HKEY_LOCAL_MACHINE = 0x80000002;

    .
}
```

`names`方法返回与该键存放在一起的所有名字的一个枚举。你可以用你熟悉的`hasMoreElements/nextElement`方法获取它们。`getValue`方法返回一个对象，该对象可以是字符串、`Integer`对象或字节数组。`setValue`方法的`value`参数也必须是上述三种类型之一。

### 12.10.3 以本地方法方式实现注册表访问函数

我们需要实现三个操作：

- 获取某个键的值；
- 设置某个键的值；
- 通过键的名字迭代。

幸运的是，你基本上已经看到了所有必须的工具，如Java字符串和数组到C的字符串和数组的转换。还了解了如何在出错时抛出异常。

有两个问题使得这些本地方法比之前的例子更加复杂。`getValue`和`setValue`方法处理`Object`类型，它可以是`String`、`Integer`或`byte[]`之一。枚举对象需要用来存放连续的`hasMoreElements`和`nextElement`调用之间的状态。

让我们先看一下`getValue`方法。代码（程序清单12-22所示）经历了以下几个步骤：

- 1) 打开注册表键。为了读取它们的值，注册表API要求这些键是开放的。
- 2) 查询与名字关联的值的类型和大小。
- 3) 把数据读到缓存。
- 4) 如果类型是REG\_SZ（字符串），调用`NewStringUTF`，用该值来创建一个新的字符串。
- 5) 如果类型是REG\_DWORD（32位整数），调用`Integer`构造器。

6) 如果类型是REG\_BINARY, 调用NewByteArray来创建一个新的字节数组, 并调用SetByteArrayRegion, 把值数据复制到该字节数组中。

7) 如果不是以上类型或调用API函数时出现错误, 那就抛出异常, 并小心地释放到此为止所获得的所有资源。

8) 关闭键, 并返回创建的对象 (String、Integer或byte[])。

如你所见, 这个例子很好的说明了怎样产生不同类型的Java对象。

在本地方法中, 处理一般的返回类型并不困难。jstring、 jobject或jarray引用仅作为一个jobject返回。但是, setValue方法接受一个对Object的引用, 并且, 为了把该Object保存为字符串、整数或字节数组, 必须确定该Object的确切类型。我们可以通过查询value对象的类, 找出java.lang.String、java.lang.Integer和byte[]的引用, 将其与IsAssignableFrom函数进行比较, 从而确定它的确切类型。

如果, class1和class2是两个类引用, 那么调用:

```
(*env)->IsAssignableFrom(env, class1, class2)
```

当class1和class2是同一个类或class1是class2的子类时, 返回JNI\_TRUE。在这两种情况下, class1对象的引用都可以转型到class2。例如, 当:

```
(*env)->IsAssignableFrom(env, (*env)->GetObjectClass(env, value), (*env)->FindClass(env, "[B"))
```

为true时, 那么我们就知道该值是一个字节数组。

下面是setValue方法代码的概述:

- 1) 打开注册表键以便写入。
- 2) 找出要写入的值的类型。
- 3) 如果类型是String, 调用GetStringUTFChars获取一个指向字符的指针。
- 4) 如果类型是Integer, 调用intValue方法获取该包装对象中存储的整数。
- 5) 如果类型是byte[], 调用GetByteArrayElements获取指向字节的指针。
- 6) 把数据和长度传递给注册表。
- 7) 关闭键。
- 8) 如果类型是String或byte[], 那么还要释放指向数据的指针。

最后, 我们介绍枚举键的本地方法。这些方法属于Win32RegKeyNameEnumeration类 (参见程序清单12-21)。当枚举过程开始时, 我们必须打开键。在枚举过程中, 我们必须保持该键的句柄。也就是说, 该键的句柄必须与枚举对象存放在一起。键的句柄是DWORD类型的, 它是一个32位数, 所以可以存放在一个Java的整数中。它被存放在枚举类的hkey域中。当枚举开始时, SetIntField初始化该域。后续的调用用GetIntField来读取其值。

在这个例子里, 我们用枚举对象存放了另外三个数据项。当枚举一开始, 我们可以从注册表中查询到名/值对的个数和最长的名字的长度, 我们在分配C字符数组保存这些名字时需要这些值。这些值存放在枚举对象的count和maxsize域中。最后, index域被初始化为-1, 表示枚举的开始。一旦其他实例域被初始化, index域就被置为0, 在完成每个枚举步骤之后, 都会进行递增。

让我们简要介绍一下支持枚举的本地方法。hasMoreElements方法很简单。

1) 检索index和count域。

2) 如果index是-1, 调用startNameEnumeration函数打开键, 查询数量和最大长度, 初始化hkey、count、maxsize和index域。

3) 如果index小于count, 则返回JNI\_TRUE, 否则返回JNI\_FALSE。

nextElement方法要复杂一些。

1) 检索index和count域。

2) 如果index是-1, 调用startNameEnumeration函数打开键, 查询数量和最大长度, 初始化hkey、count、maxsize和index域。

3) 如果index等于count, 抛出一个NoSuchElementException异常。

4) 从注册表中读入下一个名字。

5) 递增index。

6) 如果index等于count, 则关闭键。

在编译之前, 记得在Win32RegKey和Win32RegKeyNameEnumeration上都要运行javah。微软C++的完整命令行如下:

```
cl -I jdk\include -I jdk\include\win32 -LD Win32RegKey.c advapi32.lib -FeWin32RegKey.dll
```

Cygwin系统上, 请使用:

```
gcc -mno-cygwin -D __int64="long long" -I jdk\include -I jdk\include\win32  
-I c:\cygwin\usr\include\w32api -shared -Wl,--add-stdcall-alias -o Win32RegKey.dll  
Win32RegKey.c
```

因为注册表API是针对Windows的, 所以这个程序不能在其他操作系统上运行。

程序清单12-23给出了测试我们新的注册表函数的程序。我们在键中添加了三个名值对:一个字符串、一个整数和一个字节数组。

```
HKEY_CURRENT_USER\Software\JavaSoft\Java Runtime Environment
```

然后, 我们枚举该键的所有名字并检索它们的值。该程序应该打印如下信息:

```
Default user=Harry Hacker  
Lucky number=13  
Small primes=2 3 5 7 11 13
```

虽然在该键中添加这些名值对不会有危害, 但是在运行该程序后, 可能需要使用注册表编辑器去移除它们。

### 程序清单12-21 Win32RegKey.java

```
1 import java.util.*;  
2  
3 /**  
4 * A Win32RegKey object can be used to get and set values of a registry key in the Windows  
5 * registry.  
6 * @version 1.00 1997-07-01  
7 * @author Cay Horstmann  
8 */  
9 public class Win32RegKey  
10 {
```

```
11.  /**
12.   * Construct a registry key object.
13.   * @param theRoot one of HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE,
14.   * HKEY_USERS, HKEY_CURRENT_CONFIG, HKEY_DYN_DATA
15.   * @param thePath the registry key path
16.   */
17.  public Win32RegKey(int theRoot, String thePath)
18.  {
19.      root = theRoot;
20.      path = thePath;
21.  }
22.
23.  /**
24.   * Enumerates all names of registry entries under the path that this object describes.
25.   * @return an enumeration listing all entry names
26.   */
27.  public Enumeration<String> names()
28.  {
29.      return new Win32RegKeyNameEnumeration(root, path);
30.  }
31.
32.  /**
33.   * Gets the value of a registry entry.
34.   * @param name the entry name
35.   * @return the associated value
36.   */
37.  public native Object getValue(String name);
38.  /**
39.   * Sets the value of a registry entry.
40.   * @param name the entry name
41.   * @param value the new value
42.   */
43.
44.  public native void setValue(String name, Object value);
45.
46.  public static final int HKEY_CLASSES_ROOT = 0x80000000;
47.  public static final int HKEY_CURRENT_USER = 0x80000001;
48.  public static final int HKEY_LOCAL_MACHINE = 0x80000002;
49.  public static final int HKEY_USERS = 0x80000003;
50.  public static final int HKEY_CURRENT_CONFIG = 0x80000005;
51.  public static final int HKEY_DYN_DATA = 0x80000006;
52.
53.  private int root;
54.  private String path;
55.
56.  static
57.  {
58.      System.loadLibrary("Win32RegKey");
59.  }
60. }
61.
62. class Win32RegKeyNameEnumeration implements Enumeration<String>
63. {
64.     Win32RegKeyNameEnumeration(int theRoot, String thePath)
65.     {
66.         root = theRoot;
67.         path = thePath;
```

```
68.    }
69.
70.   public native String nextElement();
71.
72.   public native boolean hasMoreElements();
73.
74.   private int root;
75.   private String path;
76.   private int index = -1;
77.   private int hkey = 0;
78.   private int maxsize;
79.   private int count;
80. }
81.
82. class Win32RegKeyException extends RuntimeException
83. {
84.   public Win32RegKeyException()
85.   {
86.   }
87.
88.   public Win32RegKeyException(String why)
89.   {
90.     super(why);
91.   }
92. }
```

**程序清单 12-22 Win32RegKey.c**

```
1. /**
2.  * @version 1.00 1997-07-01
3.  * @author Cay Horstmann
4. */
5.
6. #include "Win32RegKey.h"
7. #include "Win32RegKeyNameEnumeration.h"
8. #include <string.h>
9. #include <stdlib.h>
10. #include <windows.h>
11.
12. JNIREPORT jobject JNICALL Java_Win32RegKey_getValue(JNIEnv* env, jobject this_obj, jobject name)
13. {
14.   const char* cname;
15.   jstring path;
16.   const char* cpath;
17.   HKEY hkey;
18.   DWORD type;
19.   DWORD size;
20.   jclass this_class;
21.   jfieldID id_root;
22.   jfieldID id_path;
23.   HKEY root;
24.   jobject ret;
25.   char* cret;
26.
27.   /* get the class */
28.   this_class = (*env)->GetObjectClass(env, this_obj);
29.
```

```
30. /* get the field IDs */
31. id_root = (*env)->GetFieldID(env, this_class, "root", "I");
32. id_path = (*env)->GetFieldID(env, this_class, "path", "Ljava/lang/String;");
33.
34. /* get the fields */
35. root = (HKEY) (*env)->GetIntField(env, this_obj, id_root);
36. path = (jstring)(*env)->GetObjectField(env, this_obj, id_path);
37. cpath = (*env)->GetStringUTFChars(env, path, NULL);
38.
39. /* open the registry key */
40. if (RegOpenKeyEx(root, cpath, 0, KEY_READ, &hkey) != ERROR_SUCCESS)
41. {
42.     (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
43.                         "Open key failed");
44.     (*env)->ReleaseStringUTFChars(env, path, cpath);
45.     return NULL;
46. }
47.
48. (*env)->ReleaseStringUTFChars(env, path, cpath);
49. cname = (*env)->GetStringUTFChars(env, name, NULL);
50.
51. /* find the type and size of the value */
52. if (RegQueryValueEx(hkey, cname, NULL, &type, NULL, &size) != ERROR_SUCCESS)
53. {
54.     (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
55.                         "Query value key failed");
56.     RegCloseKey(hkey);
57.     (*env)->ReleaseStringUTFChars(env, name, cname);
58.     return NULL;
59. }
60.
61. /* get memory to hold the value */
62. cret = (char*)malloc(size);
63.
64. /* read the value */
65. if (RegQueryValueEx(hkey, cname, NULL, &type, cret, &size) != ERROR_SUCCESS)
66. {
67.     (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
68.                         "Query value key failed");
69.     free(cret);
70.     RegCloseKey(hkey);
71.     (*env)->ReleaseStringUTFChars(env, name, cname);
72.     return NULL;
73. }
74.
75. /* depending on the type, store the value in a string,
76.    integer or byte array */
77. if (type == REG_SZ)
78. {
79.     ret = (*env)->NewStringUTF(env, cret);
80. }
81. else if (type == REG_DWORD)
82. {
83.     jclass class_Integer = (*env)->FindClass(env, "java/lang/Integer");
84.     /* get the method ID of the constructor */
85.     jmethodID id_Integer = (*env)->GetMethodID(env, class_Integer, "<init>", "(I)V");
86.     int value = *(int*) cret;
```

```
87.     /* invoke the constructor */
88.     ret = (*env)->NewObject(env, class_Integer, id_Integer, value);
89. }
90. else if (type == REG_BINARY)
91. {
92.     ret = (*env)->NewByteArray(env, size);
93.     (*env)->SetByteArrayRegion(env, (jarray) ret, 0, size, cret);
94. }
95. else
96. {
97.     (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
98.                         "Unsupported value type");
99.     ret = NULL;
100. }
101.
102. free(cret);
103. RegCloseKey(hkey);
104. (*env)->ReleaseStringUTFChars(env, name, cname);
105.
106. return ret;
107. }

108.

109. JNIEXPORT void JNICALL Java_Win32RegKey_setValue(JNIEnv* env, jobject this_obj,
110.         jstring name, jobject value)
111. {
112.     const char* cname;
113.     jstring path;
114.     const char* cpath;
115.     HKEY hkey;
116.     DWORD type;
117.     DWORD size;
118.     jclass this_class;
119.     jclass class_value;
120.     jclass class_Integer;
121.     jfieldID id_root;
122.     jfieldID id_path;
123.     HKEY root;
124.     const char* cvalue;
125.     int ivalue;
126.
127.     /* get the class */
128.     this_class = (*env)->GetObjectClass(env, this_obj);
129.
130.     /* get the field IDs */
131.     id_root = (*env)->GetFieldID(env, this_class, "root", "I");
132.     id_path = (*env)->GetFieldID(env, this_class, "path", "Ljava/lang/String;");
133.
134.     /* get the fields */
135.     root = (HKEY)(*env)->GetIntField(env, this_obj, id_root);
136.     path = (jstring)(*env)->GetObjectField(env, this_obj, id_path);
137.     cpath = (*env)->GetStringUTFChars(env, path, NULL);
138.
139.     /* open the registry key */
140.     if (RegOpenKeyEx(root, cpath, 0, KEY_WRITE, &hkey) != ERROR_SUCCESS)
141.     {
142.         (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
143.                         "Open key failed");
```

```
144.     (*env)->ReleaseStringUTFChars(env, path, cpath);
145.     return;
146. }
147.
148. (*env)->ReleaseStringUTFChars(env, path, cpath);
149. cname = (*env)->GetStringUTFChars(env, name, NULL);
150.
151. class_value = (*env)->GetObjectClass(env, value);
152. class_Integer = (*env)->FindClass(env, "java/lang/Integer");
153. /* determine the type of the value object */
154. if ((*env)->IsAssignableFrom(env, class_value, (*env)->FindClass(env, "java/lang/String")))
155. {
156.     /* it is a string--get a pointer to the characters */
157.     cvalue = (*env)->GetStringUTFChars(env, (jstring) value, NULL);
158.     type = REG_SZ;
159.     size = (*env)->GetStringLength(env, (jstring) value) + 1;
160. }
161. else if ((*env)->IsAssignableFrom(env, class_value, class_Integer))
162. {
163.     /* it is an integer--call intValue to get the value */
164.     jmethodID id_intValue = (*env)->GetMethodID(env, class_Integer, "intValue", "()I");
165.     ivalue = (*env)->CallIntMethod(env, value, id_intValue);
166.     type = REG_DWORD;
167.     cvalue = (char*)&ivalue;
168.     size = 4;
169. }
170. else if ((*env)->IsAssignableFrom(env, class_value, (*env)->FindClass(env, "[B")))
171. {
172.     /* it is a byte array--get a pointer to the bytes */
173.     type = REG_BINARY;
174.     cvalue = (char*)(*env)->GetByteArrayElements(env, (jarray) value, NULL);
175.     size = (*env)->GetArrayLength(env, (jarray) value);
176. }
177. else
178. {
179.     /* we don't know how to handle this type */
180.     (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
181.                         "Unsupported value type");
182.     RegCloseKey(hkey);
183.     (*env)->ReleaseStringUTFChars(env, name, cname);
184.     return;
185. }
186.
187. /* set the value */
188. if (RegSetValueEx(hkey, cname, 0, type, cvalue, size) != ERROR_SUCCESS)
189. {
190.     (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
191.                         "Set value failed");
192. }
193.
194. RegCloseKey(hkey);
195. (*env)->ReleaseStringUTFChars(env, name, cname);
196.
197. /* if the value was a string or byte array, release the pointer */
198. if (type == REG_SZ)
199. {
```

```
200.     (*env)->ReleaseStringUTFChars(env, (jstring) value, cvalue);
201. }
202. else if (type == REG_BINARY)
203. {
204.     (*env)->ReleaseByteArrayElements(env, (jarray) value, (jbyte*) cvalue, 0);
205. }
206 }
207.
208 /* helper function to start enumeration of names */
209 static int startNameEnumeration(JNIEnv* env, jobject this_obj, jclass this_class)
210 {
211.     jfieldID id_index;
212.     jfieldID id_count;
213.     jfieldID id_root;
214.     jfieldID id_path;
215.     jfieldID id_hkey;
216.     jfieldID id_maxsize;
217.
218.     HKEY root;
219.     jstring path;
220.     const char* cpath;
221.     HKEY hkey;
222.     DWORD maxsize = 0;
223.     DWORD count = 0;
224.
225. /* get the field IDs */
226.     id_root = (*env)->GetFieldID(env, this_class, "root", "I");
227.     id_path = (*env)->GetFieldID(env, this_class, "path", "Ljava/lang/String;");
228.     id_hkey = (*env)->GetFieldID(env, this_class, "hkey", "I");
229.     id_maxsize = (*env)->GetFieldID(env, this_class, "maxsize", "I");
230.     id_index = (*env)->GetFieldID(env, this_class, "index", "I");
231.     id_count = (*env)->GetFieldID(env, this_class, "count", "I");
232.
233. /* get the field values */
234.     root = (HKEY)(*env)->GetIntField(env, this_obj, id_root);
235.     path = (jstring)(*env)->GetObjectField(env, this_obj, id_path);
236.     cpath = (*env)->GetStringUTFChars(env, path, NULL);
237.
238. /* open the registry key */
239.     if (RegOpenKeyEx(root, cpath, 0, KEY_READ, &hkey) != ERROR_SUCCESS)
240.     {
241.         (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
242.                           "Open key failed");
243.         (*env)->ReleaseStringUTFChars(env, path, cpath);
244.         return -1;
245.     }
246.     (*env)->ReleaseStringUTFChars(env, path, cpath);
247.
248. /* query count and max length of names */
249.     if (RegQueryInfoKey(hkey, NULL, NULL, NULL, NULL, NULL, NULL, &count, &maxsize,
250.                         NULL, NULL, NULL) != ERROR_SUCCESS)
251.     {
252.         (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
253.                           "Query info key failed");
254.         RegCloseKey(hkey);
255.         return -1;
256.     }
```

```
257
258     /* set the_field values */
259     (*env)->SetIntField(env, this_obj, id_hkey, (DWORD) hkey);
260     (*env)->SetIntField(env, this_obj, id_maxsize, maxsize + 1);
261     (*env)->SetIntField(env, this_obj, id_index, 0);
262     (*env)->SetIntField(env, this_obj, id_count, count);
263     return count;
264 }
265
266 JNIEXPORT jboolean JNICALL Java_Win32RegKeyNameEnumeration_hasMoreElements(JNIEnv* env,
267     jobject this_obj)
268 {
269     jclass this_class;
270     jfieldID id_index;
271     jfieldID id_count;
272     int index;
273     int count;
274     /* get the class */
275     this_class = (*env)->GetObjectClass(env, this_obj);
276
277     /* get the field IDs */
278     id_index = (*env)->GetFieldID(env, this_class, "index", "I");
279     id_count = (*env)->GetFieldID(env, this_class, "count", "I");
280
281     index = (*env)->GetIntField(env, this_obj, id_index);
282     if (index == -1) /* first time */
283     {
284         count = startNameEnumeration(env, this_obj, this_class);
285         index = 0;
286     }
287     else
288         count = (*env)->GetIntField(env, this_obj, id_count);
289     return index < count;
290 }
291
292 JNIEXPORT jobject JNICALL Java_Win32RegKeyNameEnumeration_nextElement(JNIEnv* env,
293     jobject this_obj)
294 {
295     jclass this_class;
296     jfieldID id_index;
297     jfieldID id_hkey;
298     jfieldID id_count;
299     jfieldID id_maxsize;
300
301     HKEY hkey;
302     int index;
303     int count;
304     DWORD maxsize;
305
306     char* cret;
307     jstring ret;
308
309     /* get the class */
310     this_class = (*env)->GetObjectClass(env, this_obj);
311
312     /* get the field IDs */
313     id_index = (*env)->GetFieldID(env, this_class, "index", "I");
314     id_count = (*env)->GetFieldID(env, this_class, "count", "I");
```

```
314.     id_hkey = (*env)->GetFieldID(env, this_class, "hkey", "I");
315.     id_maxsize = (*env)->GetFieldID(env, this_class, "maxsize", "I");
316.
317.     index = (*env)->GetIntField(env, this_obj, id_index);
318.     if (index == -1) /* first time */
319.     {
320.         count = startNameEnumeration(env, this_obj, this_class);
321.         index = 0;
322.     }
323.     else
324.         count = (*env)->GetIntField(env, this_obj, id_count);
325.
326.     if (index >= count) /* already at end */
327.     {
328.         (*env)->ThrowNew(env, (*env)->FindClass(env, "java/util/NoSuchElementException"),
329.                           "past end of enumeration");
330.         return NULL;
331.     }
332.
333.     maxsize = (*env)->GetIntField(env, this_obj, id_maxsize);
334.     hkey = (HKEY)(*env)->GetIntField(env, this_obj, id_hkey);
335.     cret = (char*)malloc(maxsize);
336.
337.     /* find the next name */
338.     if (RegEnumValue(hkey, index, cret, &maxsize, NULL, NULL, NULL, NULL) != ERROR_SUCCESS)
339.     {
340.         (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
341.                           "Enum value failed");
342.         free(cret);
343.         RegCloseKey(hkey);
344.         (*env)->SetIntField(env, this_obj, id_index, count);
345.         return NULL;
346.     }
347.
348.     ret = (*env)->NewStringUTF(env, cret);
349.     free(cret);
350.
351.     /* increment index */
352.     index++;
353.     (*env)->SetIntField(env, this_obj, id_index, *index);
354.
355.     if (index == count) /* at end */
356.     {
357.         RegCloseKey(hkey);
358.     }
359.
360.     return ret;
361. }
```

**程序清单12-23 Win32RegKeyTest.java**

```
1. import java.util.*;
2.
3. /**
4.  * @version 1.02 2007-10-26
5.  * @author Cay Horstmann
```

```

6. */
7. public class Win32RegKeyTest
8. {
9.     public static void main(String[] args)
10.    {
11.        Win32RegKey key = new Win32RegKey(
12.            Win32RegKey.HKEY_CURRENT_USER, "Software\\JavaSoft\\Java Runtime Environment");
13.
14.        key.setValue("Default user", "Harry Hacker");
15.        key.setValue("Lucky number", new Integer(13));
16.        key.setValue("Small primes", new byte[] { 2, 3, 5, 7, 11 });
17.
18.        Enumeration<String> e = key.names();
19.
20.        while (e.hasMoreElements())
21.        {
22.            String name = e.nextElement();
23.            System.out.print(name + "=");
24.
25.            Object value = key.getValue(name);
26.
27.            if (value instanceof byte[])
28.                for (byte b : (byte[]) value) System.out.print((b & 0xFF) + " ");
29.            else
30.                System.out.print(value);
31.
32.            System.out.println();
33.        }
34.    }
35. }

```



## 类型质询函数

- **jboolean IsAssignableFrom(JNIEnv \*env, jclass c11, jclass c12)**

如果第一个类的对象可以赋给第二个类的对象，则返回JNI\_TRUE，否则返回JNI\_FALSE。情况是这样的：两个类相同、c11是c12的子类或c12表示一个由c11或它的一个超类实现的接口。

- **jclass GetSuperClass(JNIEnv \*env, jclass c1)**

返回某个类的超类。如果c1表示Object类或一个接口，则返回NULL。

一路走来，大家已经学习了许多高级API，现在，我们来到了本书的尾声。我们从每位Java程序员都应该了解的主题开始，即：流、XML、网络、数据库和国际化，又用了篇幅很长的三章阐述了图形和GUI编程，最后用非常技术性的几章结尾，即安全、远程方法、注解处理和本地方法。我希望你能够真正享受这些涉及领域广泛的Java API，并能够将这些新知识应用到你的项目中。