

源代码免费下载

商业开发

代码库系列



Java

案例开发集锦

(第二版)

袁 然 郑自国 来为国 编著



电子工业出版社

PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

<http://www.phei.com.cn>

Java

案例开发集锦

(第二版)

本书除了涵盖Java系统内建功能的开发以及与目前新技术相接合的应用开发外，还着重加强了Web应用的开发。在剖析比较案例的同时，也将各种经验性思想及理论性思想寓于其中。同时引入Java常用的设计模式，把软件架构设计理论融于案例讲解之中。

本书的特点包括：

● 全面：10章内容，涉及了J2EE、J2ME、安全、网络、EJB、XML等方面。

● 翔实：每个案例都是根据读者的阅读习惯编写的，案例中不但有详细的实现步骤和源代码，而且还有详尽的代码解释，它能够让你迅速理解和掌握每个技术要点和技巧。

● 专业：每个章节、每个案例均由具有丰富经验的项目开发人员亲手设计和严格测试，确保无技术性漏洞。

● 连贯：保持了第一版的风格和思想，改进了多个案例，案例开发更加贴近实战。

商业开发代码库系列丛书：

Java案例开发集锦
Java案例开发集锦（第二版）
JSP案例开发集锦
JSP案例开发集锦（第二版）
Visual FoxPro案例开发集锦
Visual FoxPro案例开发集锦（第二版）
Visual Basic案例开发集锦
Visual Basic案例开发集锦（第二版）
Visual Basic.NET案例开发集锦
Visual Basic.NET案例开发集锦（第二版）
C++ Builder案例开发集锦
C++ Builder案例开发集锦（第二版）
Visual C++案例开发集锦
Visual C++案例开发集锦（第二版）
Visual C++.NET案例开发集锦
Visual C++.NET案例开发集锦（第二版）
Visual C#.NET案例开发集锦
Visual C#.NET案例开发集锦（第二版）
ASP案例开发集锦
ASP案例开发集锦（第二版）
ASP.NET案例开发集锦
ASP.NET案例开发集锦（第二版）
Delphi案例开发集锦
Delphi案例开发集锦（第二版）
PowerBuilder案例开发集锦
PowerBuilder案例开发集锦（第二版）



责任编辑：戴 新



本书贴有激光防伪标志，凡没有防伪标志者，属盗版图书。

ISBN 978-7-121-07112-6



9 787121 071126 >

定价：40.00元

商业开发代码库系列

Java案例开发集锦 (第二版)

袁 然 郑自国 来为国 编著

電子工業出版社

Publishing House of Electronics Industry

北京 • BEIJING

内 容 简 介

本书沿袭第1版的风格，收录了来自实战中的六十多个完整的Java编程实例，并通过案例讲解Java的开发技巧。本书除了涵盖Java系统内建功能的开发以及与目前新技术相接合的应用开发外，还着重加强了Web应用的开发，涉及了J2EE、J2ME、安全、网络、EJB、XML等。在剖析比较案例的同时，也将各种经验性思想及理论性思想寓于其中。同时引入Java常用的设计模式，把软件架构设计理论融于案例讲解之中。读者不仅可以了解到网络中众多精彩纷呈的网络应用内幕，还可以体会到Java的强大功能，更可以借助这些案例方便地开发出自己的功能强大的应用系统，成为Java的思想者。

本书主要面向Java应用程序开发人员，同时也是Java编程爱好者的参考书，还可作为大中专院校学生学习的辅助材料。

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

图书在版编目(CIP)数据

Java案例开发集锦/袁然，郑自国，来为国编著.—2版.—北京：电子工业出版社，2008.9
(商业开发代码库系列)

ISBN 978-7-121-07112-6

I. J… II. ①袁… ②郑… ③来… III. Java语言—程序设计 IV. TP312

中国版本图书馆CIP数据核字(2008)第106619号

责任编辑：戴 新

印 刷：北京天竺颖华印刷厂

装 订：三河市金马印装有限公司

出版发行：电子工业出版社

北京市海淀区万寿路173信箱 邮编：100036

北京市海淀区翠微东里甲2号 邮编：100036

开 本：787×1092 1/16 印张：22 字数：560千字

印 次：2008年9月第1次印刷

定 价：40.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：(010) 88254888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

服务热线：(010) 88258888。

前 言

从Java诞生以来，许多人意识到Java不仅是一个了不起的概念，而且还蕴藏了了不起的思想。

有人问：“Java会灭亡吗？”

也有人说：“Java是明天的唯一吗？”

或许这些都没有答案，或许答案就在每个Java者的心中。

其实答案是什么并不重要，时间会说明一切。无论是没有可继承应用开发的缺陷，还是开放性和可移植性，都是Java不可阻挡的魅力。这些导致了Java大军越来越多，挥舞着Java的旗帜，在软件开发的高山上奋勇登攀。

本书沿袭了第1版的风格，从实际编程出发，在案例的选取上更倾向于新技术点和Web应用，也更注重知识点的应用与积累，于相同处寻找不同，于不同处寻求共性。加强了知识点的对比，着重于编程中遇到的问题处理，笔者的经验性思想也寓于案例的讲解中，以便读者更加自如地运用语言，提高Java编程的灵活性。本书共收集了62个案例，详细讲解了每个案例的实现步骤和制作要点，并提供了源代码。本书所有案例均通过调试运行，运行环境和配置在书中或者附带源码中都有说明。这些案例涉及的知识点仅仅是Java的冰山一角，笔者希望通过这些案例能使读者尽快投入到Java的实际应用中去。

本书选择的案例名称有几个与第一版中的相同，但开发的角度和内容有较大的差别，侧重的知识点也不同。本书的宗旨是通过案例的学习使读者尽快了解Java的开发思想，所以案例的实现内容并不重要，它仅仅是打开理解Java的一扇窗户，重要的是通过这扇窗，找到Java的应用方法。本书的案例强调制作要点和详细步骤的讲解，由于篇幅关系，案例中的源代码没有完全体现在书中，第10章的源代码更是被省略掉，感兴趣的读者可以到华信教育资源网下载。

全书共分10章，每一章都以Java的一个知识点或者一个应用范围为标题。

第1章 Java与Swing，收集了8个应用Swing技术的案例，从案例实现步骤的讲解中可以了解Swing丰富、灵活的功能和模块化组件，从而创建Java优雅的用户界面。

第2章 Java与线程，通过7个案例说明Java的线程和多线程的处理机制和重要性，通过线程的共享和同步可以增强程序的交互性，提供更好的GUI和更好的服务器功能，大大简化应用程序设计。

第3章 Java与I/O，7个难易程度不同的案例的输入输出流操作有助于读者了解Java的I/O体系概念，通过对字节流、文件、缓冲、管道应用的对比和共性的探讨提高I/O操作的灵活性。

第4章 Java与游戏，游戏的实现是对Java语言的综合应用，侧重于图像、声音、多线程、键盘和鼠标事件的处理，本章提供6个好玩的游戏，在轻松之余学习Java编程的方法和技巧。

第5章 Java与网络, Java是为网络而生的, 它在网络应用上的得心应手不是区区几个案例便能体现的, 希望这7个案例能带给读者一点Java在网络应用和Web上的功能的体会。

第6章 Java与数据库, Java与JDBC的结合使程序员可以只设计一次数据库应用软件后, 就能在各种数据库系统上运行。本章通过5个与不同数据库连接的案例, 突出了Java的跨平台性。

第7章 J2ME技术, 移动开发已经成为开发者社区最为引人注目的新技术, J2ME凭借其开放的特性占据了绝对的市场, 成为了移动开发领域的标准。本章通过5个案例简要介绍了J2ME的特性。

第8章 J2EE, J2EE是一套全然不同于传统应用开发的技术架构, 本章的6个案例只是对庞大的J2EE的简介而已, 希望借此能够使读者了解一点这个企业级应用开发平台的体系结构。

第9章 Web服务与其他, Java在Web服务上的便利性和扩展性, 是Java保持其魅力的原因之一, 本章提供了7个案例, 涉及到Web服务、XML、打印、安全等方面的内容。

第10章 Java综合案例, 分别是在图形界面设计、移动开发、Web应用和数据库方面的综合性的案例, 体现了Java知识在编程中的综合应用。

作为一种程序设计语言, Java有简单、面向对象、不依赖于机器的结构特点, 具有可移植性、安全性等, 并且提供了并发的机制, 具有很高的性能。本书案例中涉及了Java的多线程、异常处理、I/O流、Swing、Servlet、.NET、XML、Security、DB、File、JavaBean、Socket、JVM、MDlet、Web服务等内容, 同时也体现MVC的编程思想。

本书的分类并不严格, 每个案例都涉及到很多知识点, 笔者为了突出某一个知识点而硬性将它们归类, 不妥之处敬请读者谅解, 笔者全力想展现Java的博大精深与无穷魅力, 但这个重担不是意愿所能达到的, 期望与读者共勉, 共同提高。

本书在编写过程中得到了北京美迪亚电子信息有限公司、山东大学、山东建筑大学部分师生的大力协助及许多网友的支持, 在此表示衷心的感谢。参加本书编校的人员还有袁凯、王诚梅、陈圣琳、向小平、汤代禄、刘彬、李志勇等。由于时间仓促, 作者水平有限, 书中的错误与不足之处在所难免, 真诚希望广大读者提出宝贵意见。

谨向阅读本书的读者朋友们表示诚挚的敬意和谢意。

为了方便读者阅读, 本书配套资料请登录“华信教育资源网”(http://www.hxedu.com.cn), 在“资源下载”频道的“图书资源”栏目下载。

目 录

第1章 Java与Swing	1
案例1: 屏幕捕获工具	1
案例2: 文本阅读器	6
案例3: 简单的名片管理系统	11
案例4: 鼠标画线	17
案例5: 鼠标操作	21
案例6: 计算器程序	27
案例7: 数字时钟	32
案例8: 动画效果与颜色的控制	36
本章小结	39
第2章 Java与线程	40
案例1: 一个完整的线程池的实例	40
案例2: 鸭子凫水动画	44
案例3: 生产者-消费者模型的简单实现	49
案例4: 定时关机	53
案例5: 多线程TCP端口扫描程序	57
案例6: 一个简单的年历生成程序	62
案例7: 将GIF和JPG图像转换成VRML格式	67
本章小结	73
第3章 Java与I/O	74
案例1: 使用多线程删除指定目录及子目录下所有指定文件	74
案例2: 压缩文件	79
案例3: 解压缩Zip文件	85
案例4: 批量改名	90
案例5: 文件分割器	98
案例6: 管道流实现线程间的通信	106
案例7: 排序对象	111
本章小结	113

第4章 Java与游戏	114
案例1: Java扫雷	114
案例2: 黑白棋	122
案例3: 象棋游戏	127
案例4: 一个简单的弹球游戏	133
案例5: 找不同	139
案例6: 八皇后问题	146
本章小结	150
第5章 Java与网络	151
案例1: 简单的多线程服务器	151
案例2: 用Java实现的HTTP服务器端例程	155
案例3: 一个简单的HTML浏览器	161
案例4: 用JavaMail发送邮件	167
案例5: Java版MSN	174
案例6: Java实现HTTP队列下载	178
案例7: Java实现HTTP验证	181
本章小结	184
第6章 Java与数据库	185
案例1: Access数据库编程中查询结果的表格式输出	185
案例2: SQL Server数据库编程中查询结果的表格式输出	192
案例3: MySQL数据库编程中查询结果的表格式输出	197
案例4: Oracle OCI数据库编程	203
案例5: 网吧计费系统	206
本章小结	210
第7章 J2ME技术	211
案例1: 九宫格游戏	211
案例2: 五子棋游戏	220
案例3: 手机背单词	228
案例4: 用J2ME与ASP建立数据库连接	235
案例5: 利用J2ME开发联网程序实例	238
本章小结	244

第8章 J2EE技术	245
案例1: 一个用Servlet实现购物车的程序	245
案例2: 连接数据库的JavaBean	251
案例3: 测试安全性代码	257
案例4: 用EJB实现的用户消费信息登记系统	260
案例5: Fibonacci数列	272
案例6: 简单的图书信息管理系统	275
本章小结	285
 第9章 Web服务与其他	 286
案例1: 用Servlet生成图像验证码	286
案例2: 获取JVM系统属性	289
案例3: 密码生成器	293
案例4: 数据库数据转成XML文件	294
案例5: 网页计数器	297
案例6: Java打印程序	299
案例7: 用SunJCE进行文件的加密和解密	306
本章小结	312
 第10章 Java综合案例	 313
综合案例1: 多页面文本编辑器	313
综合案例2: “逃亡者”手机游戏	318
综合案例3: 网上CD销售系统	324
综合案例4: 航空查询订票系统	337
本章小结	343



第1章 Java与Swing

本章内容

- 案例1: 屏幕捕获工具
- 案例2: 文本阅读器
- 案例3: 简单的名片管理系统
- 案例4: 鼠标画线
- 案例5: 鼠标操作
- 案例6: 计算器程序
- 案例7: 数字时钟
- 案例8: 动画效果与颜色的控制
- 本章小结



案例1: 屏幕捕获工具

案例运行效果与操作

本案例的制作涉及Java图像处理方面的内容, 它使用Java扩展包中提供的JPEG编码器, 实现了抓取当前桌面图像并保存到指定文件夹的功能。程序运行后, 界面如图1-1所示。

单击“操作”菜单, 界面如图1-2所示。

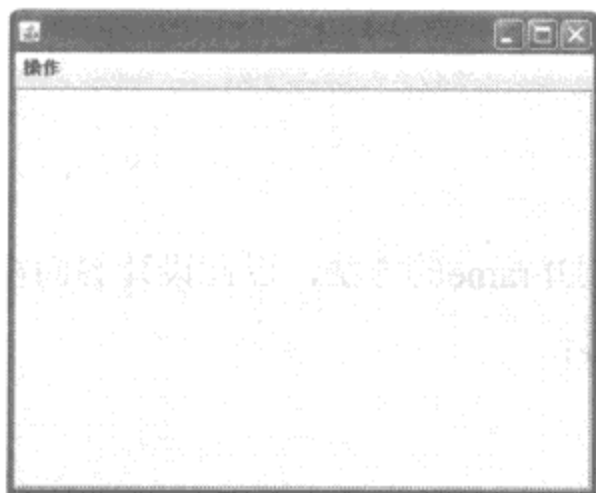


图1-1 运行界面

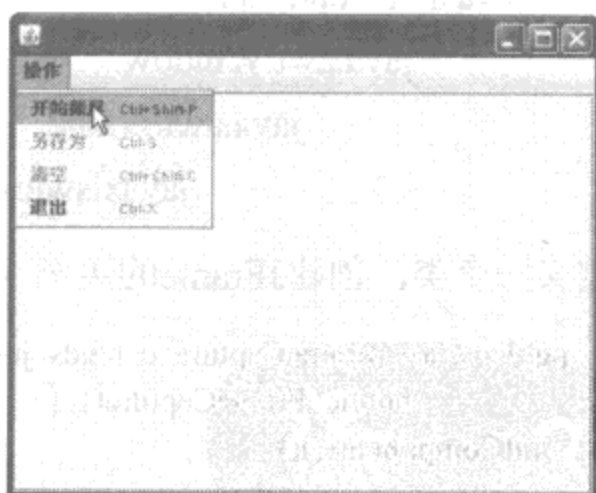


图1-2 “操作”菜单

单击“操作”菜单下的“开始截屏”菜单项, 或者按下抓图组合键Ctrl+Shift+P, 界面如图1-3所示。

按下Ctrl+S组合键保存当前窗口中的图片, 会弹出“保存”对话框, 如图1-4所示。

选择文件保存目录并填写文件名称后, 单击“保存”按钮, 完成抓取图像并保存的操作。按下Ctrl+Shift+C组合键则清空当前窗口中的内容, 返回到图1-1所示界面。按下Ctrl+X组合键则退出程序。

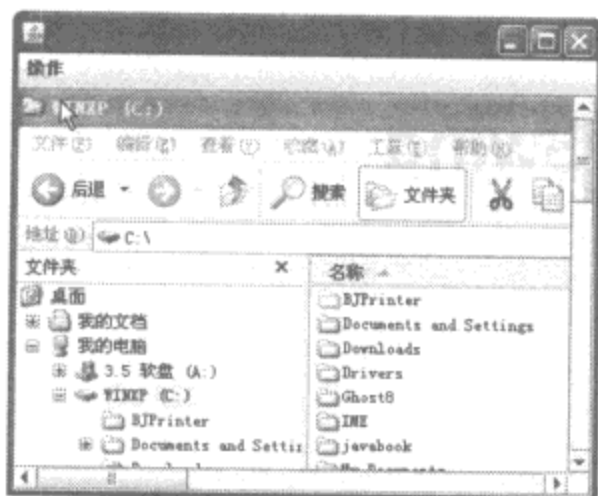


图1-3 抓取屏幕后程序窗口中的内容

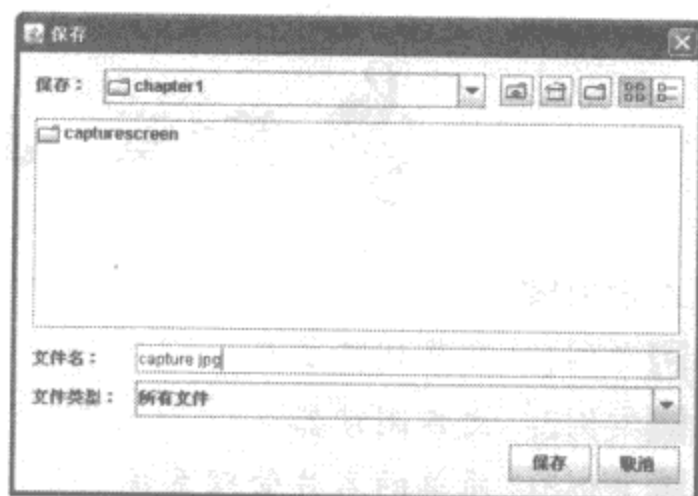


图1-4 保存图像

制作要点

1. Image类的用法。
2. JPEG编码器的使用。
3. Swing菜单的应用技巧。

步骤详解

1. 初始化JFrame窗体：不同的开发环境操作步骤不同，本例使用NetBeans。设计程序的用户界面，笔者偏向于使用Swing，也有人喜欢用AWT，大多时候两者都用。

大多数情况下使用JFrame窗口。JFrame的继承结构如下：

```
java.lang.Object
    java.awt.Component
        java.awt.Container
            java.awt.Window
                java.awt.Frame
                    javax.swing.JFrame
```

自定义一个类，创建JFrame的实例（对象）来调用JFrame的方法，以此设计界面窗口：

```
public class FrameCapture extends javax.swing.JFrame{
    public FrameCapture() {
        initComponents();
    }
}
```

开发环境提供了丰富的界面设计工具，例如NetBeans的GUI设计器，它通过拖拉的方式轻松生成所需的菜单命令等，并添加相应的动作事件，界面如图1-5所示。

2. 添加菜单控件：添加JMenuBar控件并修改其属性，属性修改界面如图1-6所示，也可直接通过代码实现。

```
jMenuBar1 = new javax.swing.JMenuBar();
...
jMenuBar1.add(jMenuItem)
```

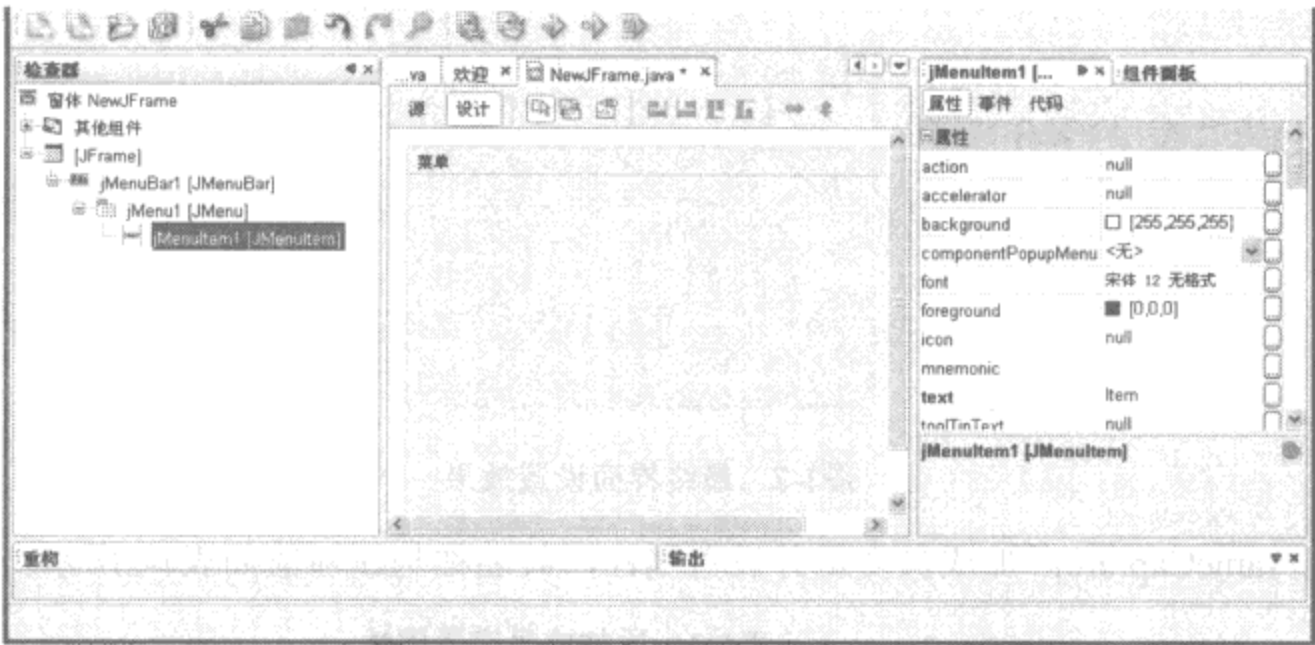



图1-5 GUI设计器

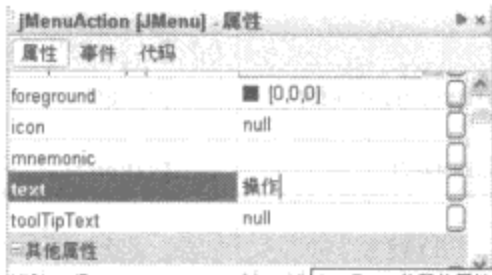


图1-6 在属性窗口中修改属性

添加JMenuItem “开始截屏”，即JMenuItemCapture并设置其属性。

```
jMenuItemCapture = new javax.swing.JMenuItem();
jMenuItemCapture.setText("\u5f00\u59cb\u622a\u5c4f");
```

3. 为JMenuItemCapture设置加速器。

```
jMenuItemCapture.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_P,
java.awt.event.InputEvent.SHIFT_MASK | java.awt.event.InputEvent.CTRL_MASK));
```

4. 用同样的方法再添加3个JMenuItem，并按照如表1-1所示修改属性。

表1-1 属性与值对照表

名称	text属性值	enable属性值	加速键
JMenuItemSaveAs	另存为	未选中	Ctrl+S
JMenuItemClear	清空	未选中	Shift+Ctrl+C
JMenuItemExit	退出	选中	Ctrl+X

5. 添加JScrollPane控件，将其名称修改为JScrollPaneGlobe。

```
jScrollPaneGlobe.setViewportView(jLabelShow)
```

6. 在JScrollPaneGlobe中添加一个JLabel，将其名称修改为JLabelShow，并将其text属性的值设置为空。

```
this.jLabelShow.setIcon(null);
```


7. 调整各个控件实例在窗体上的位置与大小，最终界面设置效果如图1-7所示。

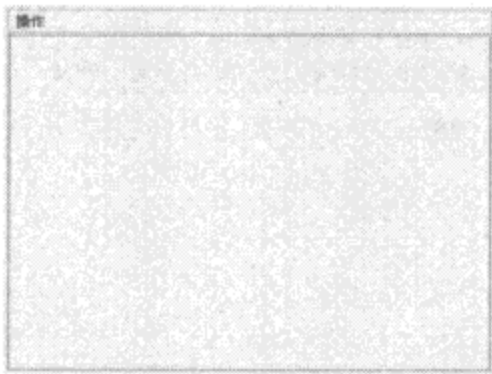


图1-7 最终界面设置效果

8. 向FrameCapture添加成员变量并设置属性，其名称与属性值如表1-2所示。

表1-2 添加成员变量属性

变量名称	变量类型	访问限制修饰符	初始值
tempImage	Image	private	null
encoder	JPEGImageEncoder	private	null
fileChooser	JFileChooser	private	new JFileChooser()

9. 添加createImage()方法及其代码，其返回类型为Image。createImage()是截图的方法，主要功能是获取屏幕大小、截取指定矩形区域内图像并返回。

10. 向FrameCapture添加saveImage()方法。saveImage()实现文件的保存输出，把输出流连接到编码器。

```
String fileName=new String(saveFileName.getPath()+".jpg");    //获取要保存文件的名字
FileOutputStream fileOutPut=new FileOutputStream(fileName);    //首先创建一个输出流
//然后把输出流用JPEG编码器进行包裹，其实就是把输出流连接到编码器
encoder=JPEGCodec.createJPEGEncoder(fileOutPut);
encoder.encode((BufferedImage) tempImage);    //把BufferedImage对象进行编码
```

11. 为JMenuItemCapture菜单项添加ActionEvent事件的处理方法。

```
Image captureImage=this.createImage();    //截取当前屏幕图像
//将图像在JLabelShow中显示
jLabelShow.setIcon(new ImageIcon(tempImage=this.createImage()));
this.jMenuItemSaveAs.setEnabled(true);
this.jMenuItemClear.setEnabled(true);
this.jMenuItemCapture.setEnabled(false);
```

12. 为JMenuItemClear菜单项添加ActionEvent事件的处理方法。

```
this.jLabelShow.setIcon(null);    //将JLabelShow中的图像清空
this.jMenuItemSaveAs.setEnabled(false);
this.jMenuItemClear.setEnabled(false);
this.jMenuItemCapture.setEnabled(true);
```

程序源代码与解释

```
/** FrameCapture.java*/
public class FrameCapture extends javax.swing.JFrame {
```



```

    public FrameCapture() {
        initComponents();
    }
    /**在构造器内调用的方法，用于初始化表格 */
    private void initComponents() {
        //...代码略，请参见本书附带的代码，下同
    }
    private void jMenuItemClearActionPerformed(java.awt.event.ActionEvent evt) {
        //参见步骤详解12...    //将JLabelShow中的图像清空
    }
    private void jMenuItemSaveAsActionPerformed(java.awt.event.ActionEvent evt) {
        this.saveImage();    //调用saveImage()方法保存当前屏幕图像
    }
    private void jMenuItemCaptureActionPerformed(java.awt.event.ActionEvent evt) {
        //截取当前屏幕图像，将图像在JLabelShow中显示
        //参见步骤详解11
    }
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new FrameCapture().setVisible(true);
            }
        });
    }
    private JPEGImageEncoder encoder = null;
    private JFileChooser fileChooser = new JFileChooser();
    private Image tempImage = null;
    private Image createImage() {
        try { //截图代码开始
            Image tempLocalImage=null;
            Robot robot=new Robot();
            Dimension dimension= Toolkit.getDefaultToolkit().getScreenSize();
            Rectangle scrRect=new Rectangle(0,0,dimension.width,dimension.height);
            tempLocalImage=robot.createScreenCapture(scrRect);
            return tempLocalImage;    //截图代码结束
        } catch (AWTException ex) {
            ex.printStackTrace();
        }
        return null;}
    public void saveImage() {
        try{
int saved=fileChooser.showSaveDialog(this);
if(saved==JFileChooser.APPROVE_OPTION) {
            File saveFileName=fileChooser.getSelectedFile();    //获取要设置的文件名（包括
            //参见步骤详解10
            fileOutPut.flush();
            fileOutPut.close();
            }}
            catch(Exception ea) { ea.printStackTrace();}
        }
    }
}

```

路径)



案例2：文本阅读器

案例运行效果与操作

本案例是关于Java复合对话框方面的内容，它使用Swing中提供的各种对话框和文件过滤器等控件，实现针对用户不同操作出现不同的与用户交互对话框的功能。程序运行后，界面如图1-8所示。

单击“浏览”按钮，会弹出“打开”对话框，界面如图1-9所示。

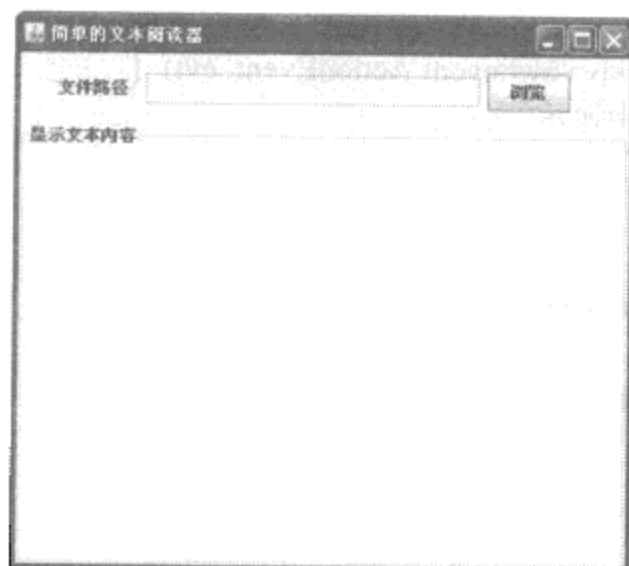


图1-8 运行界面

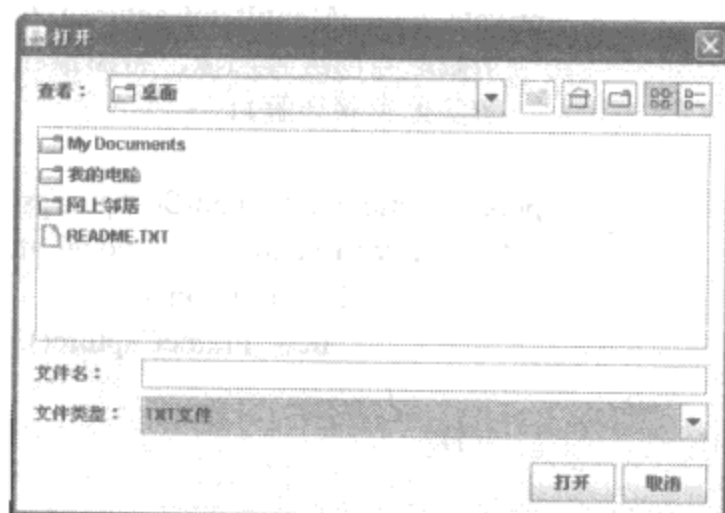


图1-9 “打开”对话框

如果未选择文件而单击“取消”按钮，则弹出提示对话框，界面如图1-10所示。如果选择文件，并单击“打开”按钮，会弹出确认对话框，如图1-11所示。

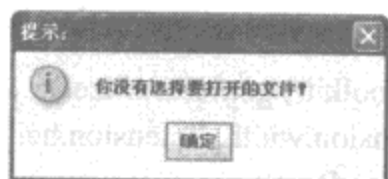


图1-10 未选择文件

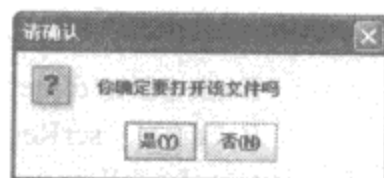


图1-11 确认打开文件

此时如果选择“否”，会弹出确认对话框，如图1-12所示。选择“是”，则显示相应文件内容，如图1-13所示。

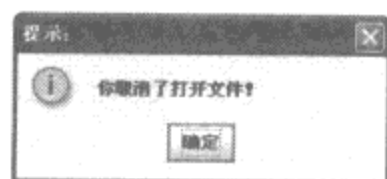


图1-12 取消打开

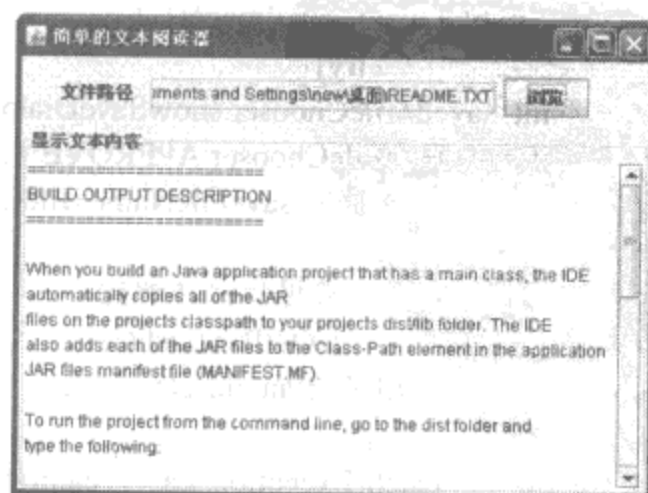


图1-13 显示文件内容

另外，通过文件过滤器的应用，使得该程序只能打开文本文件或者Java文件，如图1-14所示。

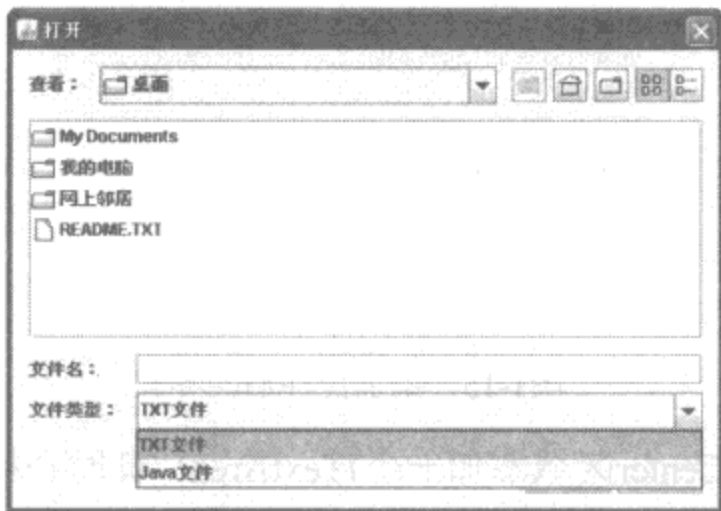


图1-14 只能打开文本文件或者Java文件

制作要点

1. Swing对话框的应用技巧。
2. JOptionPane类的使用方法。
3. 文件过滤器的使用。
4. 输入/输出流的使用。

步骤详解

1. JFrame设置：选择“设置布局” / “BorderLayout”选项，将FrameReader的布局管理器设置为BorderLayout。

```
public class FrameReader extends javax.swing.JFrame{  
    ...  
    initComponents();  
    this.initFileChooser();  
}
```

2. 添加JPanel控件，向窗体中添加一个JPanel控件JPanelNorth，属性设置为North。

```
jPanelNorth = new javax.swing.JPanel();  
...  
getContentPane().add(jPanelNorth, java.awt.BorderLayout.NORTH);
```

3. 向JPanelNorth中分别添加一个JLabel、JTextField和JButton，并按照表1-3所示修改属性。此时GUI设计器中的界面如图1-15所示。

表1-3 控件属性与值对照表

控件类型	控件名称	text属性值
JLabel	JLabelFilePath	文件路径
JTextField	JTextFieldFilePath	空
JButton	JButtonOpen	浏览

4. 在JTextFieldFilePath属性窗口中，将editable属性设置为非选中状态。

```
jTextFieldFilePath.setEditable(false)
```

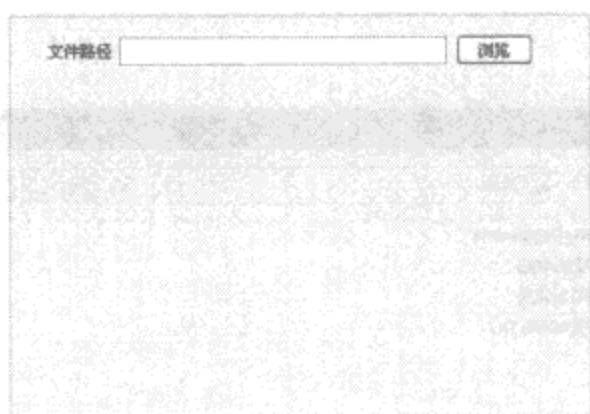



图1-15 添加控件后的界面

5. 在FrameReader的Center区域添加一个JTextArea, 此时在检查器窗口中会自动添加一个JScrollPane节点。

```
jTextAreaContent = new javax.swing.JTextArea();
```

JTextAreaContent和JScrollPaneCenter设置如下:

```
jTextAreaContent = new javax.swing.JTextArea()
jScrollPaneCenter = new javax.swing.JScrollPane()
```

6. 调整各个控件实例在窗体上的位置与大小, 界面最终设置效果如图1-16所示。

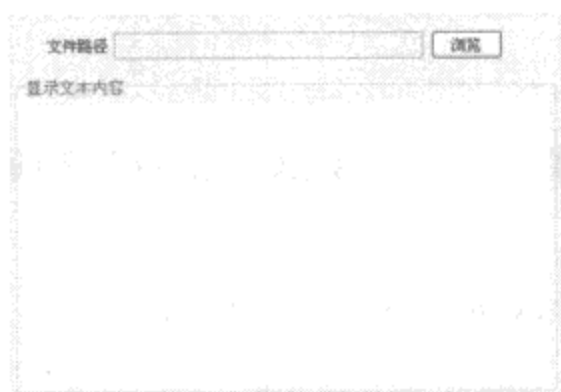


图1-16 界面最终设置效果

7. Swing中提供了JOptionPane类来实现类似Windows平台下的MessageBox的功能, 本例中复合对话框的实现使用了Java中的JOptionPane类。JOptionPane类中的方法可用各种标准的对话框实现显示信息、提出问题、警告、用户输入参数等功能。这些对话框都是模式对话框。其中常用的如下:

- **ConfirmDialog**——确认对话框, 提出问题, 然后由用户自己来确认 (单击 “Yes” 或 “No” 按钮)。
- **InputDialog**——提示输入文本。
- **MessageDialog**——显示信息。
- **OptionDialog**——组合其他三个对话框类型。

这四个对话框可以采用showXXXDialog()来显示, 如showConfirmDialog()显示确认对话框、showInputDialog()显示输入文本对话框、showMessageDialog()显示信息对话框、showOptionDialog()显示选择性的对话框。本例中显示信息对话框:

```
JOptionPane.showMessageDialog(this,"你没有选择要打开的文件!", "提示: ",JOptionPane
.INFORMATION_MESSAGE)
```


选择性对话框:

```
JOptionPane.showConfirmDialog(this,"你确定要打开该文件吗","请确认", JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE)
```

8. 超类javax.swing.JFileChooser.FileFilter。FileFilter是一个抽象类，JFileChooser使用它过滤显示给用户的文件集合。

```
private javax.swing.JFileChooser fileChooser = new javax.swing.JFileChooser()
```

9. 添加文件过滤器initFileChooser()方法:

```
//获取原来已经有的过滤器
FileFilter[] fileFilter= fileChooser.getChoosableFileFilters();
int fSize=fileFilter.length;
for(int i=0;i<fSize;i++) //删除原来方法内已经有的过滤器
{ fileChooser.removeChoosableFileFilter(fileFilter[i]);
}
fileChooser.addChoosableFileFilter(new TextFilter()); //添加TXT过滤器
fileChooser.addChoosableFileFilter(new JavaFilter()); //添加Java过滤器
```

10. 文件输入/输出流:

```
String pathName=selectFile.getPath(); //获取文件路径
//在JTextFieldFilePath中显示路径
this.jTextFieldFilePath.setText(pathName);
try
{
    int size=(int)selectFile.length();
    //创建字节数组对象，用来存储文件内容
    byte[] tempArray=new byte[size];
    //创建输入流对象，获取文件内容
    FileInputStream fin=new FileInputStream(selectFile);
    fin.read(tempArray); //文件内容存储在字节数组中
    //创建字符串对象，用来存储文件内容
    String tempString=new String(tempArray);
    //在JTextAreaContent中显示文件内容
    this.jTextAreaContent.setText(tempString);
}
catch(IOException e)
{
    e.printStackTrace();
}
```

程序源代码与解释

```
/* * FrameReader.java*/
public class FrameReader extends javax.swing.JFrame {
    /**创建 FrameReader窗体 */
    public FrameReader() {
        initComponents();
        this.initFileChooser();
    }
    private void initComponents() { //组件初始化 }
    private void jButtonOpenActionPerformed(java.awt.event.ActionEvent evt) {
```



```

        fileChooser.showOpenDialog(this);
        File selectFile=fileChooser.getSelectedFile();
        if(selectFile==null)
        {
            JOptionPane.showMessageDialog(this,"你没有选择要打开的文件！","提示：",
JOptionPane.INFORMATION_MESSAGE);
            return;
        }
        else {
            int choose=JOptionPane.showConfirmDialog(this,"你确定要打开该文件吗","请确认",
JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE);
            if(choose==JOptionPane.NO_OPTION) {
                JOptionPane.showMessageDialog(this,"你取消了打开文件！","提示：
",JOptionPane.INFORMATION_MESSAGE);
                return;
            }else {
                //参见步骤详解10
            } }
        } }
    }
    private javax.swing.JFileChooser fileChooser = new javax.swing.JFileChooser();
    public void initFileChooser() {
        //获取原来已经有的过滤器
        //参见步骤详解9
    }
}

class TextFilter extends javax.swing.filechooser.FileFilter {
    public boolean accept(File file) {
        if(file.getName().toLowerCase().endsWith(".txt"))
        { flag=true; }
        else if(file.isDirectory()) { flag=true; }
        else { flag=false; }
        return flag;
    }
    public String getDescription(){//该方法返回文件类型的说明信息
        return "TXT文件";
    }
}

class JavaFilter extends javax.swing.filechooser.FileFilter {
    boolean flag;
    public boolean accept(File file) {
        if(file.getName().toLowerCase().endsWith(".java"))
        { flag=true; }
        else if(file.isDirectory())
        { flag=true; }
        else
        { flag=false; }
        return flag;
    }
    public String getDescription(){ //该方法返回文件类型的说明信息
        return "Java文件";
    }
}

```




案例3：简单的名片管理系统

案例运行效果与操作

本案例涉及Java布局管理器方面的内容，综合使用Swing中提供的各种布局管理等控件，实现了简单的名片管理功能，能够查看已有名片信息和添加新的名片信息。程序运行后，界面如图1-17所示。

单击左侧的“添加”单选按钮，此时右侧的“名片详细信息”和“名片附加信息”中的内容变为可编辑状态，用户可以输入名片内容来进行添加，其中单击“名片详细信息”中的“爱好”或“学历”单选按钮，则在“名片附加信息”中显示相应的选项供用户选择，界面如图1-18所示。

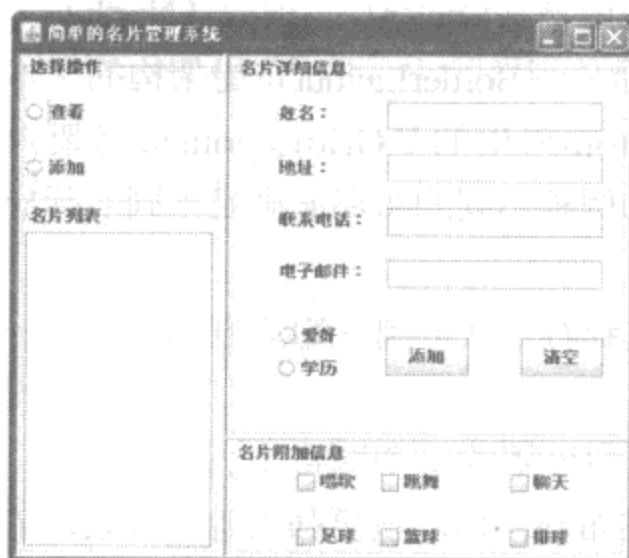


图1-17 运行界面

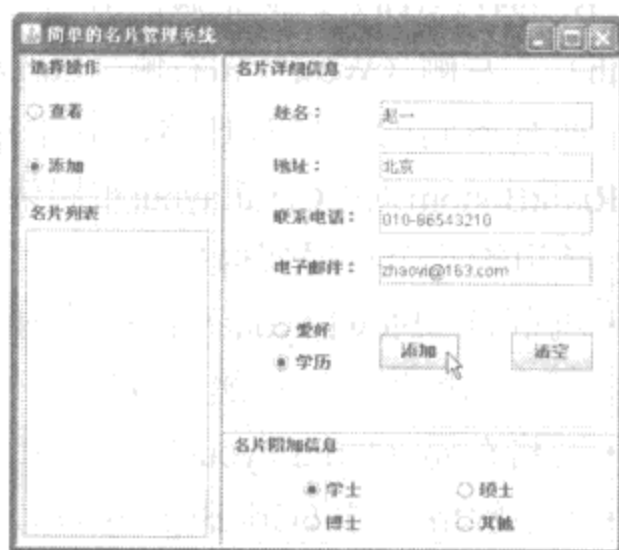


图1-18 输入名片信息

名片信息输入完毕，单击“清空”按钮，则将所有输入信息清空；如果单击“添加”按钮，则在左侧的“名片列表”列表框中会显示相应的名片姓名，表明输入信息得到了保存，同时清空所有的输入信息供下一次的信息添加，如图1-19所示。

单击左侧的“查看”单选按钮，此时右侧的“名片详细信息”和“名片附加信息”中的内容变为不可编辑状态，用户可以查看在左侧的“名片列表”列表框中显示的已有名片的信息。在“名片列表”列表框中选择一个名片名称，则其相应信息会在右侧的“名片详细信息”和“名片附加信息”中显示出来，如图1-20所示。

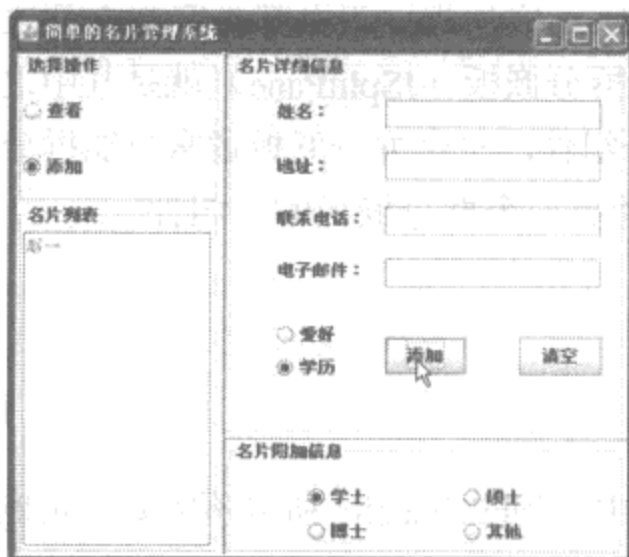


图1-19 添加名片

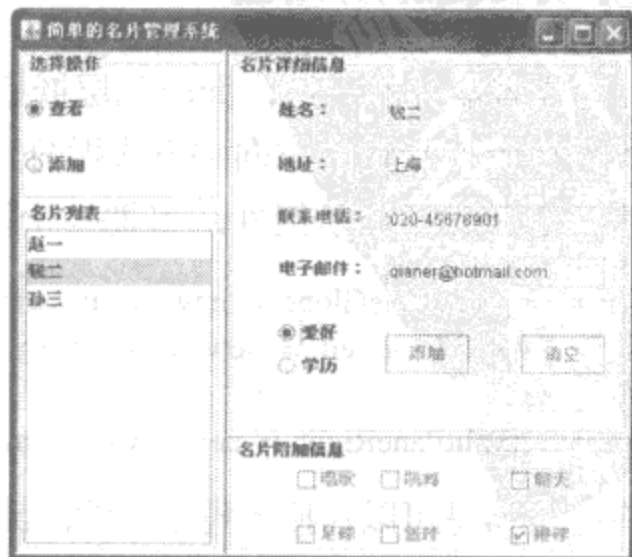


图1-20 查看名片信息

制作要点

1. Swing布局管理器的应用技巧。
2. JSplitPane的使用。
3. Vector类的使用。
4. Hashtable类的使用。

步骤详解

1. Swing布局管理器。

在实际编程中，每设计一个窗体，都要往其中添加若干组件。为了管理好这些组件的布局，就需要使用布局管理器。将加入到容器的组件按照一定的顺序和规则放置，使之看起来更美观，这就是布局。在Java中，布局由布局管理器（LayoutManager）来管理。比如边界管理器（BorderLayout），它把版面分为五大块，即中央区域（Center）、顶端（North）、底部（South）、左侧（West）和右侧（East）。案例2就是使用BorderLayout布局架构的。本例中使用了两种不同的布局管理器，其中JpanelLeftBottom采用的是GridLayout布局架构，而JpanelRightBottom则用CardLayout布局架构，这是因为两种不同的布局能满足不同实际应用的需求。常用的几个布局架构如下：

- 流布局（FlowLayout）： 把其内的组件按从左到右、从上到下的流方式排列，一行放不下则折到下一行继续放置。
- 网格布局（GridLayout）： 把组件放置到布局中设置的每个网格中。
- 无序网格布局（GridBagLayout）： 类似于网格布局，但功能更强大也更复杂，能处理所有的布局。
- CardLayout： 将组件像卡片一样放置在容器中，在某个时刻只有一个组件可见。
- Javax.swing BoxLayout： 就像整齐放置的一行或者一列盒子，每个盒子中一个组件。
- Javax.swing SpringLayout： 根据一组约束条件放置子组件。
- Javax.swing ScrollPaneLayout： 专用于JScrollPane，含一个Viewport、一个行头、一个列头、两个滚动条和四个角组件。
- Javax.swing OverlayLayout： 以彼此覆盖的形式叠置组件。

在本书后面的案例中还会对不同的布局管理器的用法进行讨论。

2. Swing中所有的组件都是放到容器中，主要的容器包括：Jframe、Jpanel、Jwindow、Jdialog、Jpanle。JFrame是Java的主框架，几乎所有的Java应用程序界面都是在主框架之中设计的。有些容器并非一层，而由几层嵌板组成，其中拆分嵌板（JSplitPane）就是其中之一。

拆分嵌板按指定的方向和方式拆分其内的两个子组件，一个拆分嵌板通常只能拆分两个子组件。如拆分三个子组件，可把一个拆分嵌板作为另一个拆分嵌板的子组件。

```
jSplitPaneGlobe = new javax.swing.JSplitPane();
jSplitPaneLeft = new javax.swing.JSplitPane();
...
jSplitPaneRight = new javax.swing.JSplitPane();
```

有些容器是可以直接放置组件的，比如前面案例中用到的JPanel，所以不需要使用嵌板。

3. 在JSplitPaneLeft中添加一个新的JPanel，名为JPanelLeftTop，再添加一个名为JPanelLeftBottom的JPanel，并将其布局管理器设置为GridLayout。

```
jPanelLeftTop = new javax.swing.JPanel();
jPanelLeftBottom = new javax.swing.JPanel();
...
jPanelLeftBottom.setLayout(new java.awt.GridLayout());
```

4. 同理，在JSplitPaneRight中添加JPanelRightTop和ButtonGroup，并按照表1-4所示修改属性。

表1-4 控件属性与值对照表

控件类型	控件名称	text属性值
JLabel	JLabelName	姓名:
JLabel	JLabelAddress	地址:
JLabel	JLabelPhone	联系电话:
JLabel	JLabelEmail	电子邮件:
JTextField	JTextFieldName	空
JTextField	JTextFieldAddress	空
JTextField	JTextFieldPhone	空
JTextField	JTextFieldEmail	空
JRadioButton	JRadioButtonFavor	爱好
JRadioButton	JRadioButtonDegree	学历
JButton	JButtonAdd	添加
JButton	JButtonClear	清空

再添加名为JPanelRightBottom的JPanel，将其布局管理器设置为CardLayout，并将其“标题”属性值设置为“名片附加信息”。

```
jPanelRightBottom = new javax.swing.JPanel();
...
jPanelRightBottom.setLayout(new java.awt.CardLayout());
```

5. 在JPanelRightBottom中添加2个JPanel，在检查器窗口中将其名称分别修改为JPanelFavor和JPanelDegree，向JPanelFavor中添加6个JCheckBox，并按照表1-5所示修改属性。

表1-5 JCheckBox属性与值对照表

控件类型	控件名称	text属性值
JCheckBox	JCheckBoxSing	唱歌
JCheckBox	JCheckBoxDance	跳舞
JCheckBox	JCheckBoxChat	聊天
JCheckBox	JCheckBoxFootBall	足球
JCheckBox	JCheckBoxBasketBall	篮球
JCheckBox	JCheckBoxVolleyBall	排球

6. 向JPanelDegree中添加ButtonGroup，其名称为buttonGroup3。再向JPanelDegree中添加4个JRadioButton，并按照表1-6所示修改属性。

表1-6 JRadioButton属性与值对照表

控件类型	控件名称	text属性值
JRadioButton	JRadioButtonBachelor	学士
JRadioButton	JRadioButtonMaster	硕士
JRadioButton	JRadioButtonDoctor	博士
JRadioButton	JRadioButtonOther	其他

7. 至此，界面设置完毕，结果如图1-21所示。



图1-21 最终界面设置效果

8. Vector类提供了实现可增长数组的功能，随着更多元素加入到其中，数组变得更大，删除一些元素后，数组相应地变小。Vector类提供的访问方法支持类似数组运算和与Vector大小相关的运算。类似数组的运算允许向量中增加、删除和插入元素。他们也允许测试和检索矢量的内容，与大小相关的运算允许判定字节大小和矢量中元素的数目。

```
private java.util.Vector vecListCard = new java.util.Vector();
```

9. Hashtable类。

Hashtables（哈希表）在计算机领域中已经不是一个新概念了。在查询众多数据条目时，它能帮助尽快定位到一个特殊的条目。Java包含两个类，java.util.Hashtable和java.util.HashMap，它们提供了一个多种用途的Hashtable机制。这两个类很相似，通常提供相同的公有接口。本例中使用的是java.util.Hashtable，案例6中将用到HashMap，在其“步骤详解”中将详细介绍两者的区别。

```
private java.util.Hashtable hashCardInfo = new java.util.Hashtable();
```

10. 为JRadioButtonView单选按钮添加ItemEvent事件的处理方法JRadioButtonViewItemStateChanged():

```
this.setState(false); //设置名片信息，显示控件为可编辑状态
this.clearAll();       //清空名片信息，显示控件中的内容
```

11. 为JRadioButtonAdd单选按钮添加ItemEvent事件的处理方法：


```
this.setState(true);
```

12. 为JRadioButtonFavor单选按钮添加ItemEvent事件的处理方法:

```
((java.awt.CardLayout)jPanelRightBottom.getLayout()).show(jPanelRightBottom,"favorcard");
```

13. 为JRadioButtonDegree单选按钮添加ItemEvent事件的处理方法:

```
((java.awt.CardLayout)jPanelRightBottom.getLayout()).show(jPanelRightBottom,"degrecard");
```

14. 类CardInfo用来记录一张名片的信息, 其成员变量按照表1-7进行设置。

表1-7 属性与值对照表

字段名称	类型	字段名称	类型
name	String	address	String
phone	String	email	String
singState	boolean	danceState	boolean
chatState	boolean	footBallState	boolean
basketBallState	boolean	volleyBallState	boolean
bachelorState	boolean	masterState	boolean
doctorState	boolean	otherrState	boolean

15. getCardInfo()方法, 访问限制符为private, 返回值为CardInfo, 该方法将用户所填写的名片信息内容包装成一个CardInfo对象返回:

```
CardInfo cardinfo=new CardInfo();
cardinfo.setName(this.jTextFieldName.getText());
cardinfo.setAddress(this.jTextFieldAddress.getText());
cardinfo.setPhone(this.jTextFieldPhone.getText());
cardinfo.setEmail(this.jTextFieldEmail.getText());
cardinfo.setSingState(this.jCheckBoxSing.isSelected());
cardinfo.setDanceState(this.jCheckBoxDance.isSelected());
cardinfo.setChatState(this.jCheckBoxChat.isSelected());
cardinfo.setFootBallState(this.jCheckBoxFootBall.isSelected());
cardinfo.setBasketBallState(this.jCheckBoxBasketBall.isSelected());
cardinfo.setVolleyBallState(this.jCheckBoxVolleyBall.isSelected());
cardinfo.setBachelorState(this.jRadioButtonBachelor.isSelected());
cardinfo.setMasterState(this.jRadioButtonMaster.isSelected());
cardinfo.setDoctorState(this.jRadioButtonDoctor.isSelected());
cardinfo.setOtherState(this.jRadioButtonOther.isSelected());
return cardinfo;
```

16. setCardInfo()方法, 访问限制符为private, 返回值为void, 入口参数为CardInfo, 该方法用于获取CardInfo中的信息:

```
this.jTextFieldName.setText(cardinfo.getName());
this.jTextFieldPhone.setText(cardinfo.getPhone());
this.jTextFieldAddress.setText(cardinfo.getAddress());
this.jTextFieldEmail.setText(cardinfo.getEmail());
this.jCheckBoxSing.setSelected(cardinfo.getSingState());
this.jCheckBoxDance.setSelected(cardinfo.getDanceState());
```



```

this.jCheckBoxChat.setSelected(cardinfo.getSingState());
this.jCheckBoxFootBall.setSelected(cardinfo.getFootBallState());
this.jCheckBoxBasketBall.setSelected(cardinfo.getBasketBallState());
this.jCheckBoxVolleyBall.setSelected(cardinfo.getVolleyBallState());
this.jRadioButtonBachelor.setSelected(cardinfo.getBachelorState());
this.jRadioButtonMaster.setSelected(cardinfo.getMasterState());
this.jRadioButtonDoctor.setSelected(cardinfo.getDoctorState());
this.jRadioButtonOther.setSelected(cardinfo.getOtherState());

```

17. 为JButtonAdd按钮添加ActionEvent事件的处理方法，JButtonAddActionPerformed()方法将用户所填写的信息包装到CardInfo对象中，用“姓名”字段的值作为键名将Card-Info对象存储到Hashtable中，同时将“姓名”字段的值存储到Vector中，并将该Vector对象传给JListCardList以显示更新后名片的名称列表：

```

CardInfo cardinfo=this.getCardInfo();
hashCardInfo.put(cardinfo.getName(),cardinfo);
vecListCard.add(cardinfo.getName());
this.jListCard.setListData(vecListCard);
this.clearAll();

```

18. 为JListCard列表框添加ListSelectionEvent事件的处理方法jListCardValueChanged(), 该方法将用户所选中名片的详细信息在控件中显示：

```

String selectedCardName=(String)jListCard.getSelectedValue();
CardInfo cardinfo=(CardInfo)hashCardInfo.get(selectedCardName);
this.setCardInfo(cardinfo);

```

程序源代码与解释

```

/** CardManager.java */
package cardmanager;
public class CardManager {
    /** 实例化 CardManager */
    public CardManager() {
    }
    public static void main(String[] args) {
        java.awt.EventQueue.invokeLater( new Runnable( ) {
            public void run()
            {
                new FrameCardManager( ).setVisible(true);
            }
        });
    }
}

/* FrameCardManager.java*/
package cardmanager;
public class FrameCardManager extends javax.swing.JFrame {
    /** Creates new form FrameCardManager */
    public FrameCardManager() {
        initComponents();
    }
    private void initComponents() {
        //代码略
        //界面设计，组件的初始化

```



```

    }
    private void jListCardValueChanged(javax.swing.event.ListSelectionEvent evt) {
        /* *该方法将用户所选中名片的详细信息在控件中显示。*/
        //参见步骤详解18
    }
    private void jButtonClearActionPerformed(java.awt.event.ActionEvent evt) {
        this.clearAll(); //清空名片信息显示控件中的内容
    }
    private void jButtonAddActionPerformed(java.awt.event.ActionEvent evt) {
        /* *该方法将用户所填写的信息包装到CardInfo对象中，用“姓名”字段的值作为键名，
        将CardInfo对象存储到Hashtable中，同时将“姓名”字段的值存储到Vector中，
        并将该Vector对象传给JListCardList以显示更新后名片的名称列表。
        */
        //参见步骤详解17
    }
    private void jRadioButtonDegreeItemStateChanged(java.awt.event.ItemEvent evt) {
        ((java.awt.CardLayout)jPanelRightBottom.getLayout()).show(jPanelRightBottom,"degreecard");
    }
    private void jRadioButtonFavorItemStateChanged(java.awt.event.ItemEvent evt) {
        ((java.awt.CardLayout)jPanelRightBottom.getLayout()).show(jPanelRightBottom,"favorcard");
    }
    private void jRadioButtonAddItemStateChanged(java.awt.event.ItemEvent evt) {
        this.setState(true);
    }
    private void jRadioButtonViewItemStateChanged(java.awt.event.ItemEvent evt) {
        this.setState(false); //设置名片信息显示控件为可编辑状态
        this.clearAll(); //清空名片信息显示控件中的内容
    }
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new FrameCardManager().setVisible(true);
            }
        });
    }
    private CardInfo getCardInfo() {
        //参见步骤详解15
    }
    private void setCardInfo(CardInfo cardinfo) {
        //参见步骤详解16
    }
    private java.util.Vector vecListCard = new java.util.Vector();
    private java.util.Hashtable hashCardInfo = new java.util.Hashtable();
    //变量声明 (略)
}

```



案例4：鼠标画线

案例运行效果与操作

本案例涉及Java鼠标事件方面的内容，使用Swing中提供的各种鼠标事件处理方法，实现了通过拖曳鼠标进行画线的功能。程序运行后，界面如图1-22所示。

此时移动鼠标，鼠标坐标的值会随之变化。按下鼠标不动，就会以当前坐标作为起始坐标，界面如图1-23所示。

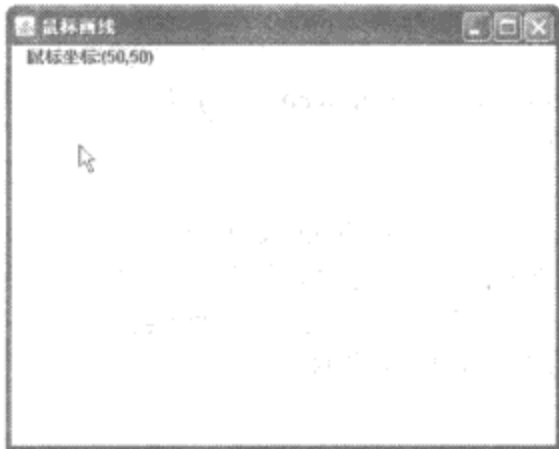


图1-22 鼠标移动时界面

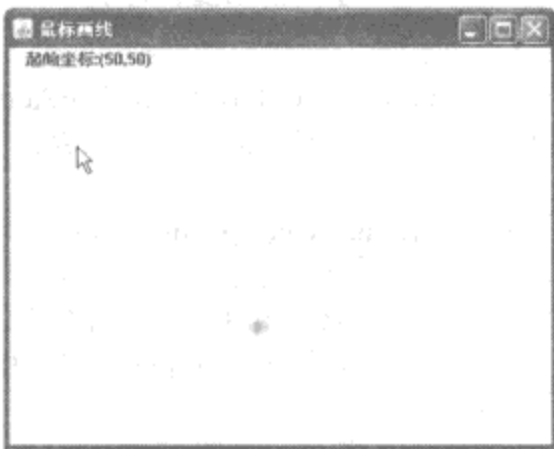


图1-23 鼠标按下时界面

拖曳鼠标（即按下鼠标左键不放并拖动），界面如图1-24所示。此时起始坐标值不变，而拖曳鼠标坐标的值会随鼠标的拖动而变化。

在目标位置松开鼠标，此时会显示起始坐标的值和终点坐标的值，并且在二者之间画一条直线，如图1-25所示。

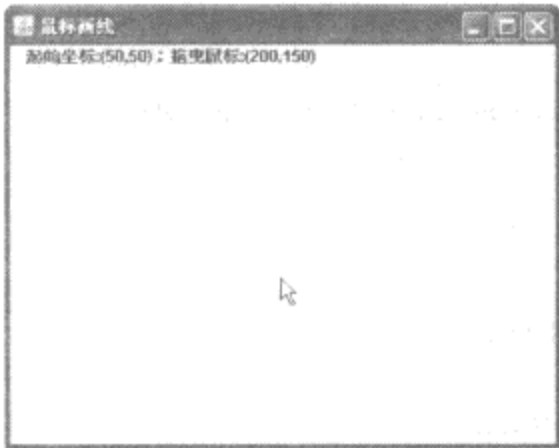


图1-24 鼠标拖曳时界面

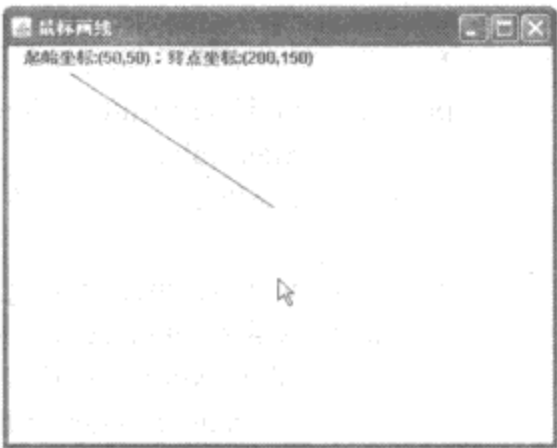


图1-25 鼠标松开时界面

制作要点

- 1. MouseEvent的应用技巧。
- 2. MouseMotionEvent的应用技巧。
- 3. 图形的重载。

步骤详解



图1-26 界面设计效果

- 1. 创建JFrame窗体，添加两个JLabel控件JLabelInfo和JPanelDrawLine，并调整控件实例在窗体上的位置和大小，使JLabelInfo控件处于顶端，而JPanelDrawLine控件覆盖FrameDrawLine窗体的其他位置。界面设计效果如图1-26所示。
- 2. 向FrameDrawLine添加成员变量，其属性按照表1-8进行设置。

表1-8 属性与值对照表

字段名称	类型	初始值
info	String	new String()
flag	int	不设置
x	int	0
y	int	0
startx	int	不设置
starty	int	不设置
endx	int	不设置
endy	int	不设置

3. MouseEvent对象被传递给每一个MouseListener或MouseAdapter对象，这些对象使用组件的addMouseListener方法注册，以接收“令人感兴趣”的鼠标事件（MouseAdapter对象实现MouseListener接口）。所有此类侦听器对象都获得包含鼠标事件的MouseEvent。

```
jPanelDrawLine.addMouseListener(new java.awt.event.MouseAdapter() {  
    public void mousePressed(java.awt.event.MouseEvent evt) {  
        jPanelDrawLineMousePressed(evt); }  
    public void mouseReleased(java.awt.event.MouseEvent evt) {  
        jPanelDrawLineMouseReleased(evt); }  
});
```

MouseEvent对象还传递给每一个使用组件的addMouseMotionListener方法，以注册接收鼠标移动事件的MouseMotionListener或MouseMotionAdapter方法（MouseMotionAdapter对象实现MouseMotionListener接口）。所有此类侦听器对象都获得包含鼠标移动事件的MouseEvent。

```
jPanelDrawLine.addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {  
    public void mouseDragged(java.awt.event.MouseEvent evt) {  
        jPanelDrawLineMouseDragged(evt); }  
    public void mouseMoved(java.awt.event.MouseEvent evt) {  
        jPanelDrawLineMouseMoved(evt); }  
})
```

4. Paint()方法覆盖了类java.awt.Container的paint方法。返回类型为void，参数设置为Graphics g。

```
g.setColor(Color.black);        //设置字体颜色  
if(flag==1)                    //鼠标移动  
{    this.jLabelInfo.setText("鼠标坐标:("+x+","+y+")");    }  
if(flag==2)                    //鼠标拖动  
{    this.jLabelInfo.setText(info+"； 拖曳鼠标:("+x+","+y+")");  
    g.setColor(this.getBackground());  
    g.fillRect(startx,starty,x,y);  
    g.drawLine(startx,starty,x,y);        //画线  
}  
if(flag==3)                    //鼠标释放  
{    this.jLabelInfo.setText(info+"； 终点坐标:("+endx+","+endy+")");  
    g.drawLine(startx,starty,endx,endy);    //画线  
}
```


5. 为JPanelDrawLine添加各种Mouse事件的处理方法，Mouse事件有5种，这里用到了其中的MousePressed和MouseReleased事件。MousePressed()取得鼠标拖曳的起始坐标，MouseReleased()取得鼠标拖曳的终点坐标。

```
//mousePressed
startx=evt.getX();
starty=evt.getY();
info="起始坐标:("+startx+","+starty+")";
this.jLabelInfo.setText(info);
//mouseReleased
flag=3;
endx=evt.getX();
endy=evt.getY();
repaint();
```

6. 添加MouseMotion事件的处理方法。MouseMotion事件有2种，分别是MouseMoved和MouseDragged事件。MouseMoved()取得鼠标移动的每一个坐标，并调用repaint()方法；MouseDragged()取得鼠标移动的每一个坐标，并调用repaint()方法。

```
flag=1;
x=evt.getX();
y=evt.getY();
repaint();
```

程序源代码与解释

```
/** FrameDrawLine.java*/
public class FrameDrawLine extends javax.swing.JFrame {
    /** Creates new form FrameDrawLine */
    public FrameDrawLine() {
        initComponents();
    }
    private void initComponents() {
        //组件初始化，代码略
    }
    //MouseDragged()取得鼠标移动的每一个坐标,并调用repaint()方法
    private void jPanelDrawLineMouseDragged(java.awt.event.MouseEvent evt) {
        flag=2;
        x=evt.getX();
        y=evt.getY();
        repaint();
    }
    //MouseReleased()取得鼠标拖曳的起始与终点坐标
    private void jPanelDrawLineMouseReleased(java.awt.event.MouseEvent evt) {
        //参见步骤详解5
    }
    private void jPanelDrawLineMousePressed(java.awt.event.MouseEvent evt) {
        //参见步骤详解5
    }
    private void jPanelDrawLineMouseMoved(java.awt.event.MouseEvent evt) {
        //参见步骤详解6
    }
    public static void main(String args[]) {
```



```
java.awt.EventQueue.invokeLater(new Runnable() {  
    public void run() {  
        new FrameDrawLine().setVisible(true);  
    }  
});  
}  
int flag;//flag=1代表Mouse Moved,flag=2代表Mouse Dragged  
int startx,starty,endx,endy;//起始坐标和终点坐标  
String info =new String();  
public void paint(Graphics g)  
{  
    //参见步骤详解4  
}}
```



案例5: 鼠标操作

案例运行效果与操作

本案例涉及Java事件处理方面的内容,使用Swing中提供的各种鼠标、键盘事件处理方法,实现了通过鼠标、键盘对颜色框进行控制的功能。程序运行后,界面如图1-27所示。

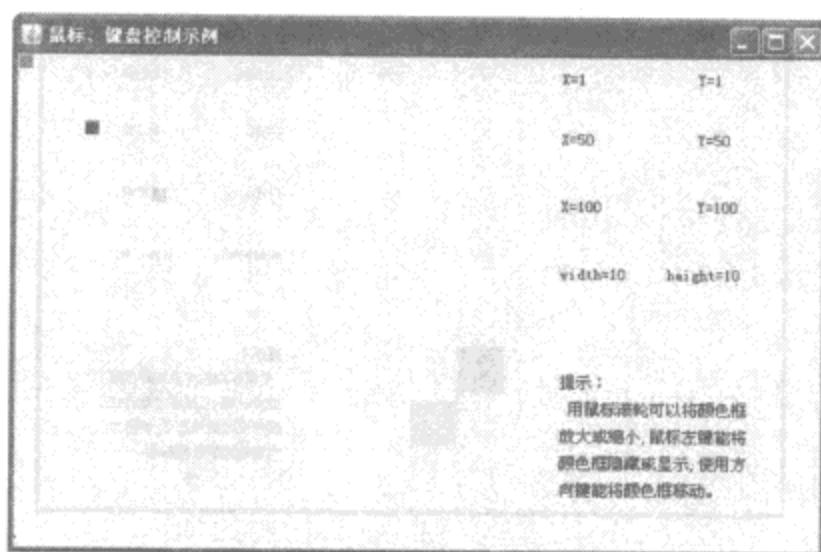


图1-27 运行时界面

此时转动鼠标滚轮,三个颜色框的大小会随之变化,右方width和height的值显示当前颜色框的大小,如图1-28所示。

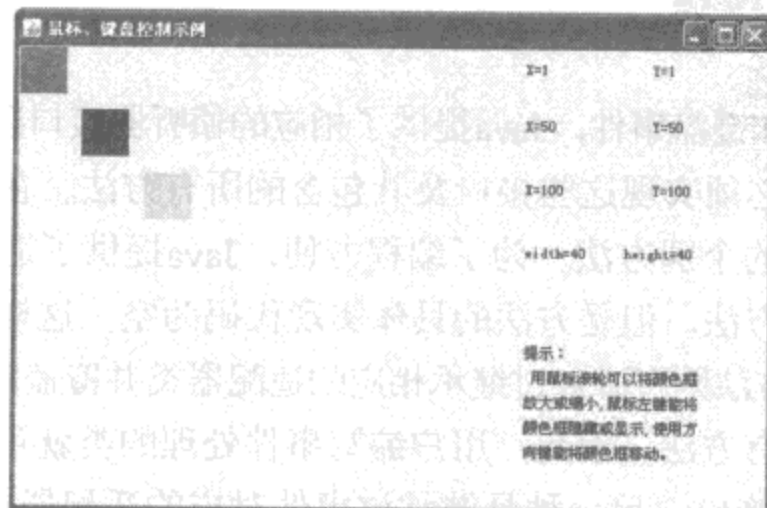


图1-28 转动鼠标滚轮时界面

用鼠标左键单击颜色框会将其隐藏，如图1-29所示为单击了蓝色和绿色颜色框的界面。此时在除红色颜色框外任何位置（单击红色颜色框会使其隐藏）按下鼠标，都会使隐藏的颜色框重新出现。

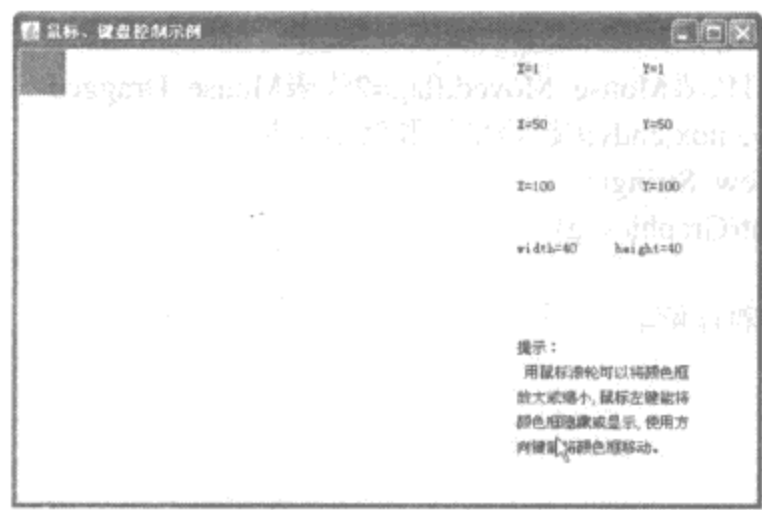


图1-29 单击颜色框时的界面

使用键盘上的方向键，则会使颜色框向相应方向移动，其中红色颜色框移动最快，而绿色颜色框移动最慢。同时，在右方会显示相应颜色框左上角的坐标值，其中最上方的一对X、Y代表红色色框左上角的坐标值，中间的一对X、Y代表蓝色颜色框左上角的坐标值，最下方的一对X、Y代表绿色颜色框左上角的坐标值，如图1-30所示。

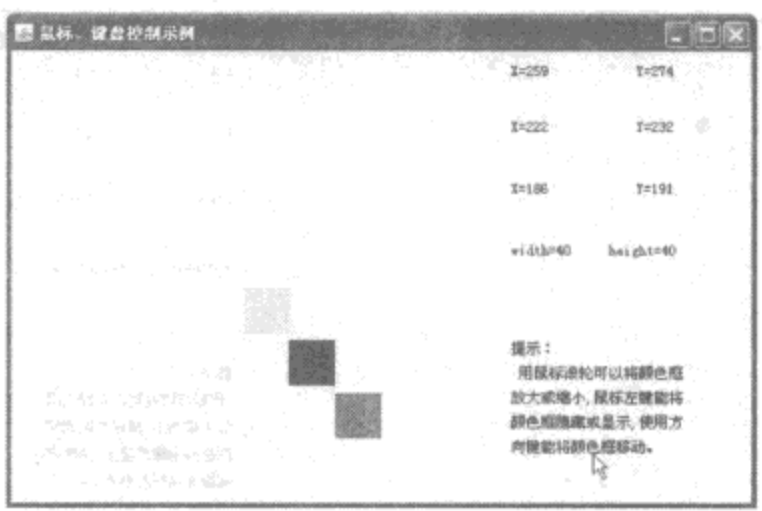


图1-30 键盘控制界面

制作要点

- 1. MouseWheelEvent的应用技巧。
- 2. KeyEvent的应用技巧。
- 3. 适配器类的使用。

对于各种鼠标事件和键盘事件，Java提供了相应的侦听器接口以及相应的事件处理方法，用于处理这些事件的类必须实现这些接口及其包含的所有方法。在许多情况下，程序中实际用到的只是接口中声明的个别方法。为了编程方便，Java提供了适配器类，在适配器类中实现了相应接口中的所有方法，但是方法的具体实现代码为空。这样，开发人员如果不需要用到侦听器接口中的所有方法则可以通过继承相应的适配器类并覆盖需要的事件处理方法既可，而不必实现接口中的所有方法。因此，用户编写事件处理的类就可以有两种选择，一种是实现该事件对应的侦听器接口，另一种是继承该事件对应的适配器类。

步骤详解

1. 本例使用普通的Java类在代码中完成主界面的设计，未用JFrame窗体方法：

```
contentPane = (JPanel) getContentPane();
contentPane.setLayout(null);           //设置布局管理器
setSize(new Dimension(600, 400));      //设置窗口大小
setTitle("鼠标、键盘控制示例");       //设置窗口标题
jPanelRed.setBackground(Color.red);    //设置红色颜色框的颜色
jPanelRed.setBounds(new Rectangle(1, 1, 10, 10)); //设置红色颜色框的大小和初始位置
jPanelBlue.setBackground(Color.blue);  //设置蓝色颜色框的颜色
jPanelBlue.setBounds(new Rectangle(50, 50, 10, 10)); //设置蓝色颜色框的大小和初始位置
jPanelGreen.setBackground(Color.green); //设置绿色颜色框的颜色
```

2. 构造函数：

```
public FrameMouseControl() {
    try {
        setDefaultCloseOperation(EXIT_ON_CLOSE); //设置主界面窗口的关闭操作
        //调用FrameMouseControlInit函数，完成主界面的初始化
        FrameMouseControlInit();
    } catch (Exception exception) {
        exception.printStackTrace();
    }
}
```

3. 编写主界面处理鼠标滚轮事件的类FrameMouseControl_this_mouseWheelAdapter，该类实现了MouseWheelListener接口：

```
class FrameMouseControl_this_mouseWheelAdapter implements MouseWheelListener {
    private FrameMouseControl adaptee;
    FrameMouseControl_this_mouseWheelAdapter(FrameMouseControl adaptee) {
        this.adaptee = adaptee;
    }
    public void mouseWheelMoved(MouseWheelEvent e) {
        adaptee.this_mouseWheelMoved(e);
    }
}
```

4. 编写主界面处理键盘按下事件的类FrameMouseControl_this_keyAdapter，该类必须实现KeyListener接口。由于本案例只用到了其中的keyPressed方法，所以采用了继承适配器类KeyAdapter的方法，这样就不必实现接口中的其他方法：

```
class FrameMouseControl_this_keyAdapter extends KeyAdapter {
    private FrameMouseControl adaptee;
    FrameMouseControl_this_keyAdapter(FrameMouseControl adaptee) {
        this.adaptee = adaptee;
    }
    public void keyPressed(KeyEvent e) {
        adaptee.this_keyPressed(e);
    }
}
```


5. 同样，通过继承适配器类的方法，分别编写处理主界面鼠标按下事件的类FrameMouseControl_this_mouseAdapter、处理红色颜色框鼠标单击事件的类FrameMouseControl_jPanelRed_mouseAdapter、处理蓝色颜色框鼠标单击事件的类FrameMouseControl_jPanelBlue_mouseAdapter、处理绿色颜色框鼠标单击事件的类FrameMouseControl_jPanelGreen_mouseAdapter，FrameMouseControlInit()方法：

```
this.addMouseListener(new FrameMouseControl_this_mouseAdapter(this));
this.addMouseWheelListener(new FrameMouseControl_this_mouseWheelAdapter(this));
this.addKeyListener(new FrameMouseControl_this_keyAdapter(this));
jPanelRed.addMouseListener(new FrameMouseControl_jPanelRed_mouseAdapter(this));
jPanelBlue.addMouseListener(new FrameMouseControl_jPanelBlue_mouseAdapter(this));
jPanelGreen.addMouseListener(new FrameMouseControl_jPanelGreen_mouseAdapter(this));
```

6. 添加主界面键盘按下事件的处理方法，获取当前点的坐标，并判断按键：

```
public void this_keyPressed(KeyEvent e) {
    int x1=jPanelRed.getLocation().x;    //获取红色颜色框的X坐标
    int y1=jPanelRed.getLocation().y;    //获取红色颜色框的Y坐标
    int x2=jPanelBlue.getLocation().x;    //获取蓝色颜色框的X坐标
    int y2=jPanelBlue.getLocation().y;    //获取蓝色颜色框的Y坐标
    int x3=jPanelGreen.getLocation().x;   //获取绿色颜色框的X坐标
    int y3=jPanelGreen.getLocation().y;   //获取绿色颜色框的Y坐标
    //按下向右方向键时
    if(e.getKeyCode()==e.VK_RIGHT & x1<296){
...
    }
    //按下向左方向键时
    else if(e.getKeyCode()==e.VK_LEFT & x1>1){
...
    }
    //按下向上方向键时
    else if(e.getKeyCode()==e.VK_UP & y1>1){
...
    }
    //按下向下方向键时
    else if(e.getKeyCode()==e.VK_DOWN & y1<293){
...
    }
}
```

7. 添加主界面鼠标滚轮事件的处理方法：

```
public void this_mouseWheelMoved(MouseWheelEvent e) {
    int x1=jPanelRed.getSize().width;    //获取红色颜色框的宽度
    int y1=jPanelRed.getSize().height;   //获取红色颜色框的高度
    int x2=jPanelBlue.getSize().width;   //获取蓝色颜色框的宽度
    int y2=jPanelBlue.getSize().height;   //获取蓝色颜色框的高度
    int x3=jPanelGreen.getSize().width;   //获取绿色颜色框的宽度
    int y3=jPanelGreen.getSize().height;   //获取绿色颜色框的高度
    //向下方滚动鼠标滚轮
    if(e.getWheelRotation()==e.WHEEL_BLOCK_SCROLL & x1>10 & y1>10){
...
    }
    //向上方滚动鼠标滚轮
```



```

else if((!(e.getWheelRotation()==e.WHEEL_BLOCK_SCROLL)) & x1<49 & y1<49){
... }
}

```

8. 添加各颜色框鼠标单击事件和主界面鼠标按下事件的处理方法:

```

public void JPanelRed_mouseClicked(MouseEvent e) { //红色颜色框鼠标单击事件处理方法
    if(e.getButton()==e.BUTTON1){
        JPanelRed.setVisible(false); //设为隐藏
    }
}
...

public void this_mousePressed(MouseEvent e) { //主界面鼠标按下事件处理方法
    if(e.getButton()==e.BUTTON1){
        JPanelRed.setVisible(true); //红色颜色框设为显示
        JPanelBlue.setVisible(true); //蓝色颜色框设为显示
        JPanelGreen.setVisible(true); //绿色颜色框设为显示
    }
}

```

9. 总结一下Swing的事件处理。

Swing是目前Java中不可缺少的窗口工具组，是用户建立图形化用户界面（GUI）程序的强大工具。Java Swing组件自动产生各种事件来响应用户行为，如当用户单击按钮或选择菜单项目时，Swing组件会产生一个ActionEvent。Swing组件会产生许多事件，如ActionEvents、ChangeEvents、ItemEvents等，来响应用户的鼠标单击、列表框中值的改变、计时器的开始计时等行为。在Java Swing编程中，通过注册侦听器，可以侦听事件源产生的事件，从而在事件处理程序中处理所需要处理的用户行为。Java Swing中处理各组件事件的一般步骤如下。

- (1) 新建一个组件（如JButton）。
- (2) 将该组件添加到相应的面板（如JPanel）。
- (3) 注册侦听器以侦听事件源产生的事件（如通过ActionListener来响应用户单击按钮）。
- (4) 定义处理事件的方法（如在ActionListener的actionPerformed中定义相应方法）。

以上步骤可以用多种方法实现，但人们通常用两种方法。第一种方法是只利用一个侦听器以及多个if语句来决定是哪个组件产生的事件；第二种方法是使用多个内部类来响应不同组件产生的各种事件，其具体实现又分两种方式，一种是匿名内部类，一种是一般内部类。

程序源代码与解释

```

/** MouseControl.java*/
public class MouseControl {
    boolean packFrame = false;
    /** 构造函数 */
    public MouseControl() {
        FrameMouseControl frame = new FrameMouseControl();
        if (packFrame) {
            frame.pack();
        } else {
            frame.validate();
        }
        //主窗口居中显示
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
    }
}

```



```

        if (frameSize.height > screenSize.height) {
            frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
            frameSize.width = screenSize.width;
        }
        frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height -
frameSize.height) / 2);
        frame.setVisible(true);
    }
    /**程序入口@param args String[] */
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                try {
                    UIManager.setLookAndFeel(UIManager. getSystemLookAndFeelClassName());
                } catch (Exception exception) {
                    exception.printStackTrace();
                }
                new MouseControl();
            }
        });
    }
}

/* * FrameMouseControl.java*/
package mousecontrol;
public class FrameMouseControl extends JFrame {
    /*** 主界面初始化. */
    private void FrameMouseControlInit() throws Exception {
        //代码略
    }
}

//处理红色颜色框鼠标单击事件的类
class FrameMouseControl_jPanelRed_mouseAdapter extends MouseAdapter {
    private FrameMouseControl adaptee;
    FrameMouseControl_jPanelRed_mouseAdapter(FrameMouseControl adaptee) {
        this.adaptee = adaptee;
    }
    public void mouseClicked(MouseEvent e) {
        adaptee.jPanelRed_mouseClicked(e);
    }
}

//处理主界面鼠标按下事件的类
class FrameMouseControl_this_mouseAdapter extends MouseAdapter {
    private FrameMouseControl adaptee;
    FrameMouseControl_this_mouseAdapter(FrameMouseControl adaptee) {
        this.adaptee = adaptee;
    }
    public void mousePressed(MouseEvent e) {
        adaptee.this_mousePressed(e);
    }
}

//处理蓝色颜色框鼠标单击事件的类
class FrameMouseControl_jPanelBlue_mouseAdapter extends MouseAdapter {

```



```
private FrameMouseControl adaptee;
FrameMouseControl_jPanelBlue_mouseAdapter(FrameMouseControl adaptee) {
    this.adaptee = adaptee;
}
public void mouseClicked(MouseEvent e) {
    adaptee.jPanelBlue_mouseClicked(e);
}
}
```



案例6：计算器程序

案例运行效果与操作

本案例讲述如何开发一个具有MVC架构的简单计算器程序。程序运行后，界面如图1-31所示。

该程序实现了简单的计算器功能，能够进行加减乘除四则运算，以及常见的三个三角函数的运算。另外，单击“CE”按钮可以实现清零功能。

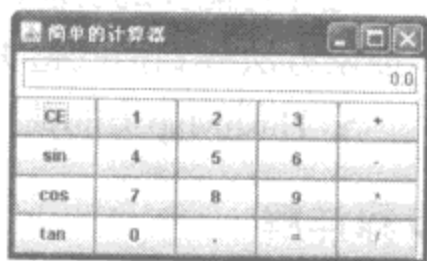


图1-31 运行时界面

制作要点

1. 布局管理器的使用，JButton、JTextfielt的应用技巧。
2. setFocusable屏幕显示是否获取焦点的用法。
3. MVC模式的应用技巧。

MVC（Model-View-Controller）模式，即模型－视图－控制器模式，其核心思想是将整个程序代码分成相对独立而又能协同工作的3个组成部分，具体的功能如下：

- 模型（Model）：业务逻辑层。实现具体的业务逻辑、状态管理的功能。
- 视图（View）：表示层。就是与用户实现交互的页面，通常实现数据的输入和输出功能。
- 控制器（Controller）：控制层。起到控制整个业务流程的作用，实现View层跟Model层的协同工作。

根据MVC模式，本案例设计了表1-9所示的各子模块。

表1-9 各子模块名称及说明

文件名称	说明
CalculateController.java	控制器，控制整个计算流程
CalculatorView.java	视图，实现计算数据的输入和计算结果的输出功能
Calculate.java	实现具体的计算逻辑
CalculateState.java	抽象类，计算状态的管理
InitCalculateState.java	抽象类CalculateState的实现，计算状态的管理
Operator.java	操作符的管理

4. 抽象类的应用技巧。

声明方法存在而不去实现它的类被叫做抽象类（abstract class），它用于创建一个体现某些基本行为的类，并为该类声明方法，但不能在该类中实现该方法的情况。不能创建抽象类的实例，但可以创建一个变量，其类型是一个抽象类，并让它指向具体子类的一个实例。不能有抽象构造函数或抽象静态方法。抽象类的子类为它们父类中的所有抽象方法提供实现，否则它们也是抽象类。知道其行为的其他类可以在类中实现这些方法。

抽象类用abstract来定义，同时，具有一个或多个抽象方法的类本身也必须声明为abstract，也就是说，抽象类可以有具体方法，但是有抽象方法的类一定是抽象类。抽象类不仅可以有具体的方法，还可以有具体数据。即使一个类中不包含任何抽象方法，这个类也可以被定义为抽象类。

把尽可能多的功能，不管是不是抽象的，放到超类中是好的做法，特别是把通用字段和非抽象方法移到抽象超类中。

5. HashMap类的使用。

步骤详解

1. 主界面设计时，调用JFrame类的方法来设置标题、窗体：

```
super(title);    //调用JFrame类的方法来设置标题
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //设置默认窗口关闭操作
setLayout(new BorderLayout());                        //设置主窗口布局
JFrame.setDefaultLookAndFeelDecorated(true);         //设置窗口外观
calController = controller;
mainScreen = new JTextField("0.0", 25);              //设置主显示屏大小和显示初始值
mainScreen.setHorizontalAlignment(JTextField.RIGHT); //设置主显示屏文本右对齐
mainScreen.setFocusable(false);                      //设置主显示屏无法获得焦点
textPanel = new JPanel();                            //定义主显示屏面板
textPanel.setLayout(new FlowLayout(FlowLayout.RIGHT)); //设置主显示屏面板布局及对齐
textPanel.add(mainScreen);                          //将主显示屏文本框添加到主显示屏面板
buttonPanel = new JPanel();                          //定义按钮面板
buttonPanel.setLayout(new GridLayout(4, 5));         //设置按钮面板布局和行列数
buttons = new JButton[NUM];                          //定义按钮数组
for (int i = 0; i < NUM; i++) {
    buttons[i] = new JButton(texts[i]);              //设置各按钮显示文本
    buttons[i].addActionListener(calController);     //为各按钮添加侦听器
    buttons[i].setActionCommand(texts[i]);           //设置各按钮命令文本
    buttonPanel.add(buttons[i]);                    //将各按钮添加到按钮面板
}
this.getContentPane().add(textPanel);                //将主显示屏面板添加到主窗口
add(textPanel, BorderLayout.NORTH);                  //设置主显示屏面板添加位置
add(buttonPanel, BorderLayout.SOUTH);                 //将按钮面板添加到主窗口并设置添加位置
pack();
setVisible(true);
```

2. 重载repaint()方法：

```
public void repaint() {
    super.repaint(); //调用母类的方法
    Calculate cal = calController.getCalculate(); //获取计算对象
```



```
mainScreen.setText(cal.getCurStr()); //设置主显示屏显示文本
```

```
}
```

3. 实现具体的计算逻辑的类Calculate:

```
switch (op) { //根据不同的运算符执行相应的运算
case ADD: //加法运算
case SIN: //正弦运算
return ret; //返回运算结果
public void clear() { //清零运算
    op1 = 0;
    op2 = 0;
    curStr = "0.0";
    op = Operator.ADD;
    calView.repaint();
}
```

4. 实现操作符管理的类Operator:

```
public enum Operator {
    ADD, SUB, MUL, DIV, COS, SIN, TAN, CE, ASSIGN; //程序能够实现的运算
    public String toString() { //该方法用于生成保存在HashMap中的键
        switch (this) {
            case ADD: return "+";
            ...
            case SIN: return "sin";
            case ASSIGN: return "=";
            default: return "Unkown Operator";
        }
    }
}
```

5. 根据不同的运算符进行不同的运算:

```
switch (op) { //根据不同的运算符执行相应的运算
case ADD: //加法运算
    ret = op1 + op2;
    break;
case SUB: //减法运算
    ret = op1 - op2;
    break;
...}
```

6. 创建两个管理计算状态的类CalculateState和InitCalculateState。其中CalculateState是抽象类，而InitCalculateState是它的实现。以下代码是抽象类CalculateState中定义的几种运算状态，其具体实现都在InitCalculateState类中。

```
public void inputA() { } //输入“=”时的运算状态
void inputD(double d) {} //输入数值（0~9以及小数点）时的运算状态
void inputU(Operator op) { } //输入一元操作符时的运算状态
void inputB(Operator op) { } //输入二元操作符时的运算状态
void inputCE(){ //输入“CE”时的运算状态
    cal.clear(); //清零
```

7. HashMap类是Java提供的Hashtable机制之一，下面主要讲解两者的区别。

Hashtable和HashMap类很相似，通常提供相同的公有接口，但它们有三个重要的不同之处。

- 第一个不同主要是历史原因。Hashtable是基于陈旧的Dictionary类的，而HashMap是Java 1.2引进的Map接口的一个实现。
- 第二点是Hashtable的方法是同步的，而HashMap方法不是。也许这是最重要的不同。这就意味着，虽然不用采取任何特殊的行为就可以在一个多线程的应用程序中用一个Hashtable，但必须同样地为一个HashMap提供外同步。一个方便的方法就是利用Collections类的静态的synchronizedMap()方法，它创建一个线程安全的Map对象，并把它作为一个封装的对象来返回。这个对象的方法可以同步访问潜在的HashMap。这么做的结果就是当不需要同步时，不能切断Hashtable中的同步（比如在一个单线程的应用程序中），而且同步增加了很多处理费用。
- 第三点不同是只有HashMap可以将空值作为一个表的条目的key或value。HashMap中只有一条记录可以是一个空的key，但任意数量的条目可以是空的value。这就是说，如果在表中没有发现搜索键，或者如果发现了搜索键，但它是一个空的值，那么get()将返回null。如果有必要，用containsKey()方法来区别这两种情况。

可以说两者的区别不是很大，案例3使用的是Hashtable，本案例使用HashMap。如何区别使用呢？有人建议，需要同步时用Hashtable，反之用HashMap。但是，HashMap可以在需要时被同步，而且HashMap的功能比Hashtable的功能多，加上是比较新的类，因此也有人认为，HashMap优先于Hashtable。

8. 为CalculateController添加静态的HashMap对象，用于保存运算操作符的Key-Value对。在源代码编辑器界面添加下列代码：

```
public static HashMap<String, Operator> operatorMap;
{
    operatorMap = new HashMap<String, Operator>(); //创建HashMap对象
    for (Operator p : Operator.values()) {
        String str = p.toString();                //获取Key
        operatorMap.put(str, p);                  //保存运算操作符的Key-Value对
    }
}
```

程序源代码与解释

```
/** CalculateController.java */
public class CalculateController implements ActionListener {
    public static HashMap<String, Operator> operatorMap;
    {
        //参见步骤详解8
    }
    private void startDigInputListener() {
        digInputListener = true;
        data = 0;
        digStr = "";
    }
    private void endDigInputListener() {
        try {
```



```

        digInputListener = false;
        data = Double.parseDouble(digStr);
    } catch (NumberFormatException e) {
        cal.clear();
        cal.setResult(0.0);
    }
}

private void digInputListener(char d) {
    digStr += d;
    curStr = digStr;
    cal.SetCurStr(digStr);
}

public void actionPerformed(ActionEvent e) {
    String str = e.getActionCommand(); //获取动作命令字符串
    char firstChar = str.charAt(0); //获取命令字符串的首字符
    if ((firstChar <= '9' && firstChar >= '0') || firstChar == '.') { //输入了数值 (0~9及小
数点)
        if (digInputListener == false) {
            startDigInputListener();
        }
        digInputListener(firstChar); //将输入的数值读出并显示
    } else { //如果输入了运算符
        if (digInputListener == true) { //并且存在运算数
            endDigInputListener();
            curCalState.inputD(data);
        }
        Operator op = operatorMap.get(str); //则获取运算符
        passMessage(op); //转passMessage方法进行相应运算
    }
}

public static void main(String[] args) {
    CalculateController calController = new CalculateController(); //创建控制器对象
    CalculatorView mainView = new CalculatorView("简单的计算器", calController); //创建
视图对象

    Calculate cal = new Calculate(mainView); //创建模型对象
    calController.setCalculate(cal);
    calController.addCalculateView(mainView);
}

}

/** CalculateState.java */
public abstract class CalculateState {
    protected CalculateController calController;
    protected Calculate cal;
    public CalculateState(CalculateController controller, Calculate c) {
        calController = controller;
        cal = c;
    }
    //参见步骤详解6
}
}

```




案例7：数字时钟

案例运行效果与操作

本案例是一个简单的时钟显示程序，使用Applet实现。程序运行后，界面如图1-32所示。

时钟代码提供了各种接口，可以在HTML文件中设置，变化出多姿多彩的时钟模型，如图1-33所示是修改HTML文件中的参数后的一个界面。



图1-32 运行时界面



图1-33 设置时区后的界面

制作要点

1. 用传统的Awt.graphics画出基本图形。
2. Applet类的应用技巧。

Applet类是所有Applet应用的基类，所有的Java小应用程序都必须继承该类。Applet类中的四种基本方法用来控制其运行状态：`init()`、`start()`、`stop()`、`destroy()`。

• `init()`方法

这个方法主要是为Applet的正常运行做一些初始化工作。当一个Applet被系统调用时，系统首先调用的就是该方法。通常可以在该方法中完成从网页向Applet传递参数的工作，添加用户界面的基本组件等操作。

• `start()`方法

系统在调用完`init()`方法之后，将自动调用`start()`方法。而且，每当用户离开包含该Applet的主页后又再返回时，系统会再执行一遍`start()`方法。这就意味着`start()`方法可以被多次执行，而不像`init()`方法。因此，可把只希望执行一遍的代码放在`init()`方法中。可以在`start()`方法中开始一个线程，如继续一个动画、声音等。

• `stop()`方法

这个方法在用户离开Applet所在页面时执行，因此，它也是可以被多次执行的。它可以在用户并不注意Applet的时候，停止一些耗用系统资源的工作以免影响系统的运行速度，且

并不需要人为地去调用该方法。如果Applet中不包含动画、声音等程序，通常也不必实现该方法。

- **destroy()方法**

与对象的**finalize()**方法不同，Java在浏览器关闭的时候才调用该方法。Applet是嵌在HTML文件中的，所以**destroy()**方法不关心何时Applet被关闭，它在浏览器关闭的时候自动执行。在**destroy()**方法中一般可以要求收回占用的非内存独立资源。如果在Applet仍在运行时浏览器被关闭，系统将先执行**stop()**方法，再执行**destroy()**方法。

3. 线程的应用技巧。

本案例时钟的不断绘制是用一个线程不断读取系统时间，如果时间有变化即绘制。采用一个线程绘制、一个线程处理网页比较符合Applet设置规范。

4. 双缓冲图形处理机制的应用技巧。

本案例时钟的绘制采用双缓冲图形处理机制，即先在缓冲区内绘制图形，再把图形显示到网页上，这样可以有效地防止闪烁。

5. 抽象类的应用技巧。

步骤详解

1. 解决屏幕闪烁的问题。

Repaint()函数在调用**paint()**函数前会自动清除屏幕，这就在一个毫秒内有一个空白的屏幕，它导致的结果是在快速的变换操作中出现闪烁现象。

解决这种闪烁现象通常有几种方法，下面介绍两种。

第一种：不清除屏幕显示，这个方法会带来副作用，它保留了所有的影像，导致显示结果与期望的大相径庭，所以一般不会采用这种方式。

第二种：使用双缓冲机制（Double buffering）。

双缓冲机制是在显示图像之前，所有的图像已经在后台绘制好并保存在相应的图像变量中，显示时直接复制到前台屏幕。现在大部分的游戏都是采用双缓冲机制来解决屏幕的闪烁现象。

2. 通过一个线程不断读取系统时间，实现时钟的不断绘制，如果时间有变化则绘制。本例采用一个线程绘制、另一个线程处理网页的方法，其中涉及到的线程同步问题请参看第2章中的案例说明。

```
while(null != clockThread){
    cur_time= (localOnly)? new Hms() :new Hms(tzDifference);
    repaint();
    try{
        Thread.sleep(500);
    } catch (InterruptedException e) {}
}
```

3. 创建四个类Hms、ClockHand、SweepHand和HmHand，其中，Hms类是进行时间换算的类；ClockHand是绘制指针的抽象基类，同时提供将角度转换为坐标的方法；而SweepHand和HmHand类继承了ClockHand类，SweepHand是秒针绘制使用的类，HmHand是时针和分针绘制使用的类。


```

abstract class ClockHand//抽象类，提供时针、分针、秒针类使用
{
    protected int baseX[], baseY[];
    protected int transX[],transY[];
    protected int numberOfPoints;
    public ClockHand(int originX, int originY, int length,int thickness,int points){
        baseX= new int[points]; baseY=new int[points];
        transX= new int[points]; transY=new int[points];
        initializePoints(originX,originY,length,thickness);
        numberOfPoints=points;
    }
    abstract protected void initializePoints( int originX, int originY, int length, int thickness);
    abstract public void draw(Color color, double angle, Graphics g);
}

```

4. 通过角度计算出时针、分针、秒针的绘制位置。

```

transX[i]=(int)((baseX[0]-baseX[i]) * Math.cos(angle) - (baseY[0]-baseY[i]) * Math.sin(angle) +
baseX[0]);
transY[i]=(int)((baseX[0]-baseX[i]) * Math.sin(angle) + (baseY[0]-baseY[i]) * Math.cos(angle) +
baseY[0]);

```

5. 添加时钟绘制的处理方法。

画出秒针（如果时与分改变则重绘时针和分针）：

```

sweep.draw(faceColor, angle, g); //画出秒针
if(cur_time.getMinutes() != lastMinute){
    minuteHand.draw(faceColor,MINSEC*lastMinute,g);
    if(cur_time.get_hours() != lastHour)
        hourHand.draw(faceColor,HOUR*lastHour,g);
}

```

程序源代码与解释

```

/* * MyClock.java */
class Hms extends Date
{
    public Hms(double localOffset){ //如果HTML文件中设置了时区，则把时间设为当地时区时间
        super();
        long tzOffset=getTimezoneOffset()*60L*1000L;
        localOffset *= 3600000.0;
        setTime(getTime() + tzOffset + (long)localOffset);
    }
}

public class MyClock extends Applet implements Runnable
{
    public void init()
    {
        URL imagesURL[] = new URL[2];
        String szImagesURL[] = new String[2];
        tracker = new MediaTracker(this);
        //得到HTML页面提供的参数，并把它转换为相应的格式
        String paramString = getParameter( "WIDTH" );
        //...
    }
}

```



```

//构造缓冲区内图形
offScrImage = createImage(width,height);
offScrGC = offScrImage.getGraphics();
System.out.println(getAppletInfo());
}
public void start() //开始启动显示线程
{
    if(clockThread == null){
        clockThread = new Thread(this);
    }
    clockThread.start();
}
public void stop() //停止显示
{clockThread = null; }
public void run()
{
    repaint(); //每次启动时首先重绘一次
    //每隔500ms获得现在时间并重绘一次
    while(null != clockThread){
        cur_time= (localOnly)? new Hms() :new Hms(tzDifference);
        repaint();
        try{
            Thread.sleep(500);
        } catch (InterruptedException e) {}
    }
}
public void paint(Graphics g) //首先绘制缓冲区内图片，再显示出来
{
    //如果没有提供背景图片，则用bgColor绘制背景
    if(images[BACKGROUND] == null){
        offScrGC.setColor(bgColor);
        offScrGC.fillRect(0,0,width,height);
    }
    else //否则直接使用背景图片
        offScrGC.drawImage(images[BACKGROUND], 0, 0, this);
    //绘制外框到表盘间的部分
    offScrGC.setColor(caseColor);
    //将圆形的范围适量缩减（不充满整个区域），防止有些地方被截取
    offScrGC.fillOval( originX+1, originY+1, minDimension-2, minDimension-2);
    //绘制表盘
    offScrGC.setColor(faceColor);
    offScrGC.fillOval( originX + size(5), originY + size(5), minDimension - size(10),
minDimension - size(10));
    //绘制外框线
    offScrGC.setColor(trimColor);
    offScrGC.drawOval( originX+1, originY+1, minDimension-2, minDimension-2);
    //绘制内框线
    offScrGC.drawOval( originX + size(5), originY + size(5), minDimension - size(10),
minDimension - size(10));
    offScrGC.setColor(textColor);
    //画刻度，一共有60个刻度，x0、y0为刻度起始的位置，x1、y1为圆心位置，x2、y2为刻
    度终止位置（x0<x2, y0<y2）
    for(i=0;i<60;i++){
        if(i==0 || (i>=5 && i%5 == 0)){ //每5格绘制一条长线（相对圆心）

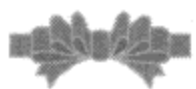
```



```

        x0=(int)(x1+size(40)*Math.sin(MINSEC*i));
        y0=(int)(y1+size(40)*Math.cos(MINSEC*i));
    }
    else{ //其他部分绘制短线
        x0=(int)(x1+size(42)*Math.sin(MINSEC*i));
        y0=(int)(y1+size(42)*Math.cos(MINSEC*i));
    }
    x2=(int)(x1+size(44)*Math.sin(MINSEC*i));
    y2=(int)(y1+size(44)*Math.cos(MINSEC*i));
    offScrGC.drawLine(x0,y0,x2,y2);
}
drawHands(offScrGC); //绘制指针
g.drawImage(offScrImage,0,0,this); //把生成的缓冲区图形绘制到页面上
isPainted=true;          //使下次更新时不绘制表盘
}
public synchronized void update(Graphics g)
{
    if(!isPainted)    //绘制表盘
        paint(g);
    else{              //已经绘制了表盘，只绘制指针即可。首先在缓冲区内绘制，然后显示出来
        drawHands(offScrGC);
        g.drawImage(offScrImage,0,0,this);
    }
}
}
}

```



案例8：动画效果与颜色的控制

案例运行效果与操作

本案例是一个简单的动画效果显示程序，使用Applet实现。程序运行后，界面如图1-34所示。



图1-34 运行界面

两个小球不停地左右运动，运动过程中大小和颜色不断变化，同时下方带“hello!”字样的方框颜色也随之变化。

制作要点

1. 利用Awt.graphics画基本图形。
2. Applet类的应用技巧。

3. 线程的应用技巧。

步骤详解

随机数的产生过程如下。

1. 在J2ME里可以使用`Math.random()`方法来产生一个随机数，产生的这个随机数是0~1之间的一个浮点数（`double`），可以乘以一定的整数，比如说100，就得到一个100以内的随机。本例中便采用这种方法：

```
i=(int) (255*Math.random());  
j=(int) (255*Math.random());  
k=(int) (255*Math.random());
```

2. 在`java.util`包里面提供了一个`Random`的类，可以通过新建一个`Random`的对象来产生随机数，可以产生随机`int`、随机`float`、随机`double`、随机`long`，这是在J2ME的程序里经常使用的一个取随机数的方法。`Math.random()`在J2ME中没有。

其实在`Random`的默认构造方法里是使用方法3进行随机数的产生的。对于方法2中的`Random`类有以下说明。

`java.util.Random`类有两种构建方式：带种子和不带种子。

不带种子：此种方式将会返回随机的数字，每次运行结果不一样：

```
public class RandomTest {  
    public static void main(String[] args) {  
        java.util.Random r=new java.util.Random();  
        for(int i=0;i<10;i++){  
            System.out.println(r.nextInt());  
        }  
    }  
}
```

带种子：此种方式无论程序运行多少次，返回结果都是一样的：

```
public static void main(String[] args) {  
    java.util.Random r=new java.util.Random(10);  
    for(int i=0;i<10;i++){  
        System.out.println(r.nextInt());  
    }  
}
```

`Random`对象的`nextInt()`和`nextInt(int n)`方法的说明如下。

`int nextInt()`：返回下一个伪随机数，它是此随机数生成器的序列中均匀分布的`int`值。

`int nextInt(int n)`：返回一个伪随机数，它是从此随机数生成器的序列中取出的、在0（包括）和指定值（不包括）之间均匀分布的`int`值。

3. `System`类中有一个`currentTimeMillis()`方法，此方法返回一个从1970年1月1号0点0分0秒到目前的一个毫秒数，返回类型是`long`，可以把它作为一个随机数，对其进行取模等进一步操作就可以把它限制在一个范围之内。

`Random`对象的`nextInt()`和`nextInt(int n)`方法的说明如下。

`int nextInt()`：返回下一个伪随机数，它是此随机数生成器的序列中均匀分布的`int`值。

`int nextInt(int n)`: 返回一个伪随机数，它是从此随机数生成器的序列中取出的、在0（包括）和指定值（不包括）之间均匀分布的`int`值。

程序源代码与解释

```
/** AnimatorAndColourControl.java */
import java.applet.Applet;
import java.awt.*;
import java.lang.Math;
import java.awt.event.*;
public class AnimatorAndColourControl extends Applet
{
    int c=50;
    int i,j,k,m;
    int l=10;
    int n=220 ;
    public void paint(Graphics g)
    {
        i=(int) (255*Math.random());
        j=(int) (255*Math.random());
        k=(int) (255*Math.random());
        Color bColor = new Color(i,j,k); //三个随机数产生随机颜色
        if(c<70)
        {
            l+=10;
            m+=1;
            n-=10;
            g.setColor(bColor); //设置当前颜色
            g.drawString("hello!",120,200);
            g.drawRect(100,180,70,30);
            //画两个圆球，其位置和大小随l、m、n的变化而不断变化
            g.fillOval(l,130-m,30+m,30+m);
            g.fillOval(n,130-m,30+m,30+m);
            try
            {
                Thread.sleep(100);
            }catch(InterruptedException e)
            {
                showStatus(e.toString());
            }
            c++;
            repaint(); //图片停留100毫秒后被擦除，重新调用paint()显示下一张图片
        }
        else
        {
            l=10;
            m=1;
            n=220;
            c=50;
            repaint();
        }
    }
}
```


本章小结

有了Swing之后，设计Java界面没有优势的感觉一去不返了。Swing是一个用于开发Java应用程序用户界面的开发工具包。它以抽象窗口工具包（AWT）为基础使跨平台应用程序可以应用任何可插拔的外观风格。Swing在不同的平台上表现一致，并且有能力提供本地窗口系统不支持的其他特性。Swing开发人员只用很少的代码就可以利用Swing丰富、灵活的功能和模块化组件来创建优雅的用户界面。本章介绍了几个难易程度不同的实用案例程序，覆盖了Swing的多个组件，有助于读者加深对Swing诸多特性的理解。



第2章 Java与线程

本章内容

- 案例1：一个完整的线程池的实例
- 案例2：鸭子凫水动画
- 案例3：生产者-消费者模型的简单实现
- 案例4：定时关机
- 案例5：多线程TCP端口扫描程序
- 案例6：一个简单的年历生成程序
- 案例7：将GIF和JPG图像转换成VRML格式
- 本章小结



案例1：一个完整的线程池的实例

案例运行效果与操作

本案例完整地实现了一个线程池。程序没有做任何处理，只是简单地将客户端输入的字符串打印到屏幕上，但它对于读者了解线程池的概念是很有帮助的。本案例创建了一个线程池，它拥有10个线程，程序运行后首先会显示线程池的初始化信息。这时如果从键盘上输入字符串，并按下Enter键，在屏幕上就会显示某个线程正在处理请求。如果快速地输入多行字符串，那么就会发现线程池中不断有线程被唤醒，来处理请求。如果线程池中沒有可用线程，系统会提示相应的警告信息。稍等片刻，就会发现屏幕上陆续提示有线程进入了睡眠状态，这时就又可以发送新的请求了。

程序运行后，输出窗口中出现如图2-1所示的界面。

此时表明线程池已经启动，而且其中的10个线程均处于睡眠状态。在输入框中依次输入一些字符作为请求处理的任务，按Enter键后界面如图2-2所示。

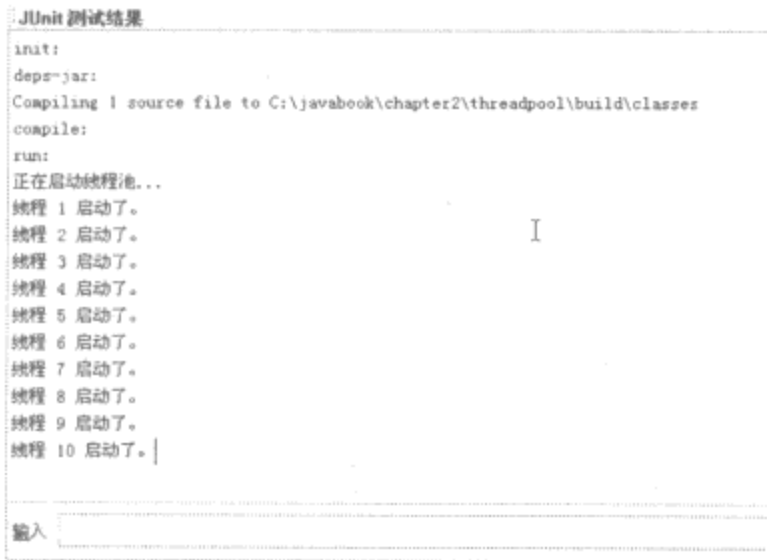


图2-1 运行界面

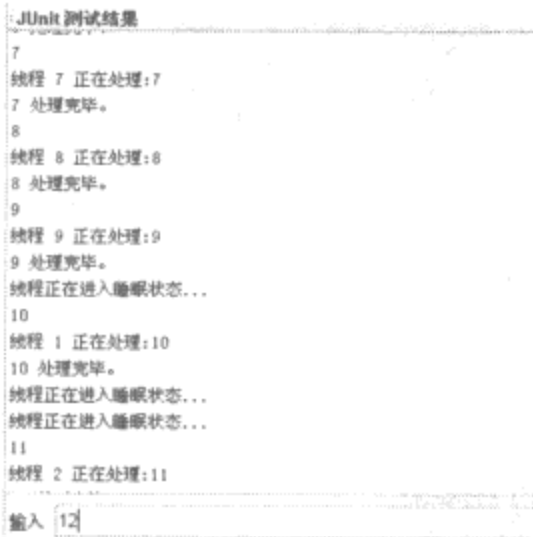


图2-2 多线程运行界面

由图2-2可以看出, 输入10后由于线程1之前已经处于睡眠状态, 可以提供服务, 所以对10的处理由线程1提供, 而不是由线程10提供。如果请求处理的任务过多, 线程池中的10个线程均处于工作状态, 则此时再输入处理任务后会提示“线程池已满, 请稍后再试。”, 而不会对任务进行处理, 界面如图2-3所示。

如果不再有新的任务需要处理, 则线程池中的线程会再次进入睡眠状态, 等待下一次的任務处理, 如图2-4所示。

```
JUnit 测试结果
线程 0 正在处理:H
H 处理完毕。
J
线程 7 正在处理:J
J 处理完毕。
K
线程 8 正在处理:K
K 处理完毕。
L
线程 9 正在处理:L
L 处理完毕。
Q
线程 10 正在处理:Q
Q 处理完毕。
线程池已满, 请稍后再试。
E
线程池已满, 请稍后再试。
输入 10
```

图2-3 线程池已满界面

```
JUnit 测试结果
线程池已满, 请稍后再试。
I
线程池已满, 请稍后再试。
O
线程池已满, 请稍后再试。
P
线程池已满, 请稍后再试。
线程正在进入睡眠状态...
线程正在进入睡眠状态...
线程正在进入睡眠状态...
线程正在进入睡眠状态...
线程正在进入睡眠状态...
线程正在进入睡眠状态...
线程正在进入睡眠状态...
线程正在进入睡眠状态...
```

图2-4 线程进入睡眠状态界面

制作要点

1. 线程池的概念及应用。

在Java中, 如果每当一个请求到达就创建一个新线程, 开销是相当大的。在实际使用中, 每个请求创建新线程的服务器在创建和销毁线程上花费的时间和消耗的系统资源, 甚至可能要比花在处理实际的用户请求的时间和资源要多得多。除了创建和销毁线程的开销之外, 活动的线程也需要消耗系统资源。如果在一个JVM里创建太多的线程, 可能会由于过度消耗内存或“切换过度”而导致系统资源不足。为了防止资源不足, 服务器应用程序需要使用一些办法来限制任何给定时刻处理的请求数目, 尽可能减少创建和销毁线程的次数, 特别是一些资源耗费比较大的线程的创建和销毁, 尽量利用已有对象来进行服务, 这就是“池化资源”技术产生的原因。

线程池的原理类似于操作系统中的缓冲区的概念, 它的流程是: 先启动若干数量的线程, 并让这些线程都处于睡眠状态, 当客户端有一个新请求时, 就会唤醒线程池中的某一个睡眠线程, 让它来处理客户端的这个请求, 当处理完这个请求后, 线程又处于睡眠状态。

线程池主要用来解决线程生命周期开销问题和资源不足问题。通过对多个任务重用线程, 线程创建的开销就被分摊到了多个任务上, 而且由于在请求到达时线程已经存在, 所以消除了线程创建所带来的延迟。这样, 就可以立即为请求服务, 使应用程序响应更快。另外, 通过适当地调整线程池中的线程数目可以防止出现资源不足的情况。

但线程池并不能适用于所有场合。当一个Web服务器接收到大量短小线程的请求时, 使用线程池技术是非常合适的, 它可以大大减少线程的创建和销毁次数, 提高服务器的工作效率。但如果线程要求的运行时间比较长, 此时线程的运行时间比创建时间要长得多, 单靠减少创建时间对系统效率的提高不明显, 此时就不适合应用线程池技术, 需要借助其他的技术来提高服务器的服务效率。

2. Thread类的应用技巧。

创建新线程有两种方法。一种方法是扩展java.lang.Thread类，将类声明为Thread的子类，该子类应重写Thread类的run方法，然后分配并启动该子类的实例。创建线程的另一种方法是通过java.lang.Runnable接口，声明实现Runnable接口的类，该类然后实现run方法，接着可以分配该类的实例，在创建Thread时作为一个参数来传递并启动。本案例使用第一种方法。

要激活一个线程，必须调用它的start()方法，start()方法会自动调用run()接口，因此用户必须在run()接口中写入自己的应用处理逻辑。

线程的睡眠与唤醒控制：Java语言为所有的对象都内置了wait()和notify()方法，当一个线程调用wait()方法时，则线程进入睡眠状态，不会继续执行以下的代码了。当调用notify()方法时，则会从其调用wait()方法的那行代码继续执行以下的代码。如果在一个方法中调用了wait()和notify()函数，需要将此方法置为同步，即synchronized。

3. 程序设计。

本案例由三个类构成，第一个类是TestThreadPool类，它是一个测试程序，用来模拟客户端的请求。

第二个类是ThreadPoolManager类，是一个用于管理线程池的类，主要职责是初始化线程池，并为客户端的请求分配不同的线程来进行处理，如果线程池满了，会发出警告信息。

最后一个类是SimpleThread类，是Thread类的一个子类，用于对客户端的请求进行处理。SimpleThread在示例程序初始化时处于睡眠状态，但如果接收到了ThreadPoolManager类发过来的调度信息，则会将自己唤醒，并对请求进行处理。

步骤详解

1. 创建线程池。

一个比较简单的线程池至少应包含线程池管理器、工作线程、任务队列、任务接口等部分。其中，线程池管理器（ThreadPool Manager）的作用是创建、销毁并管理线程池，将工作线程放入线程池中；工作线程是一个可以循环执行任务的线程，在没有任务时进行等待；任务队列的作用是提供一种缓冲机制，将没有处理的任务放在任务队列中；任务接口是每个任务必须实现的接口，主要用来规定任务的入口、任务执行完成后的收尾工作、任务的执行状态等，工作线程通过该接口调度任务的执行。创建有10个线程的线程池：

```
ThreadPoolManager manager = new ThreadPoolManager(10);
```

创建出所有线程并启动：

```
for(int i = 1; i<= threadCount; i++){           //创建出所有线程并启动它们
    SimpleThread thread = new SimpleThread(i);
    vector.addElement(thread);                    //存放数组中
    thread.start();
}
```

2. 本例通过继承Thread类的方法实现多线程。

将类声明为Thread的子类，该子类应重写Thread类的run方法，然后分配并启动该子类的实例。

```
class SimpleThread extends Thread{}
```


3. 判断每个线程是否为空闲:

```

for(i = 0; i < vector.size(); i++){
    SimpleThread currentThread = (SimpleThread)vector.elementAt(i);
    if(!currentThread.isRunning()){ //有空线程, 处理任务
        System.out.println("线程 "+ (i+1) +" 正在处理:" + argument);
        currentThread.setArgument(argument);
        currentThread.setRunning(true);
        return;
    }
}

```

4. 线程唤醒:

```

public synchronized void setRunning(boolean flag){
    runningFlag = flag; //设置运行标志
    if(flag)
        this.notify(); //唤醒线程, 处理任务
}

```

程序源代码与解释

```

/* * TestThreadPool.java */
public class TestThreadPool {
    /**实例化 TestThreadPool */
    public TestThreadPool() {
    }
    public static void main(String[] args) {
        try{
            //创建一个输入流以便接收键盘输入信息
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            String s;
            //创建一个拥有10个线程的线程池
            ThreadPoolManager manager = new ThreadPoolManager(10);
            while((s = br.readLine()) != null) //获取一个处理任务
            {
                manager.process(s); //调用ThreadPoolManager类的process方法进行处理
            }
        }catch(IOException e){}
    }
}

/* * ThreadPoolManager.java */
class ThreadPoolManager{
    private int maxThread;
    public Vector vector;
    public ThreadPoolManager(int threadCount){ //线程池初始化
        setMaxThread(threadCount); //设置线程池中最大线程数
        System.out.println("正在启动线程池...");
        vector = new Vector();
        //参见步骤详解1
    }
    public void process(String argument){
        int i;
        //参见步骤详解3
    }
}

```



```

    }
    if(i == vector.size()){    //没有空线程，给出提示信息
        System.out.println("线程池已满，请稍后再试。");
    }
}
}

/* * SimpleThread.java */
class SimpleThread extends Thread{    //通过继承Thread类的方法实现多线程
    private boolean runningFlag;
    private String argument;
    public boolean isRunning(){//该方法用来判断线程是否空闲
        return runningFlag;
    }
    //该同步方法让空闲线程处理任务请求
    public synchronized void setRunning(boolean flag){
        //参见步骤详解4
    }
    public String getArgument(){
        return this.argument;
    }
    public void setArgument(String string){
        argument = string;
    }
    public SimpleThread(int threadNumber){//线程初始化
        runningFlag = false; //设置成空闲（睡眠）状态
        System.out.println("线程 " + threadNumber + " 启动了。");
    }
    public synchronized void run(){//继承类必须重写该方法
        try{
            while(true){    //无限循环
                if(!runningFlag){    //如果没有任务
                    this.wait();    //将线程置于睡眠状态
                }
                else{    //唤醒后的线程执行以下代码
                    System.out.println(getArgument() + " 处理完毕。");
                    sleep(5000);    //该语句用于调试程序
                    System.out.println("线程正在进入睡眠状态...");
                    setRunning(false); // //设置成空闲（睡眠）状态
                }
            }
        } catch (InterruptedException e){
            System.out.println("异常，程序中止。");
        }
    }
}
}

```



案例2：鸭子凫水动画

案例运行效果与操作

本案例是一个JApplet动画，它使用Swing中提供的计时器类Timer作为绘制动画帧的触发器，实现了简单的动画效果。程序运行后，界面如图2-5所示。

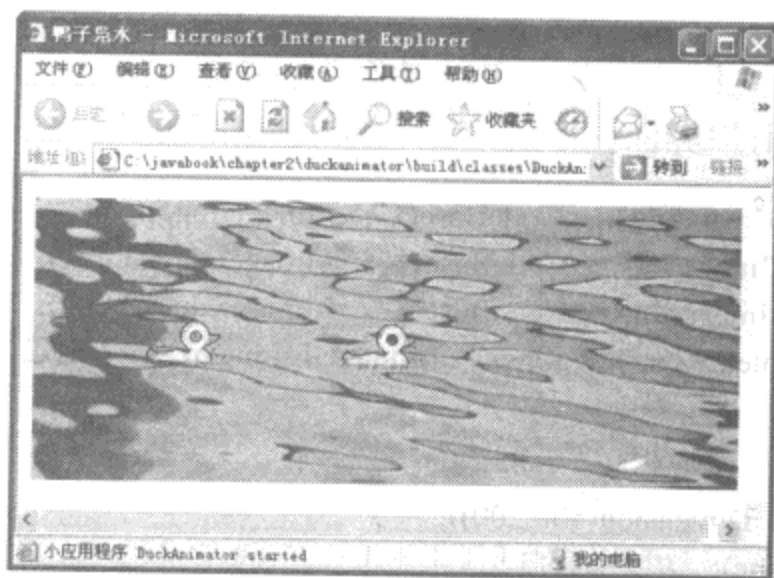


图2-5 运行界面

两只鸭子在水中游动，它们一边前进，还一边凫水。此时用鼠标单击画面，则鸭子停止前进，在原地凫水，界面如图2-6所示。

用鼠标再次单击画面，鸭子继续前进，界面如图2-7所示。

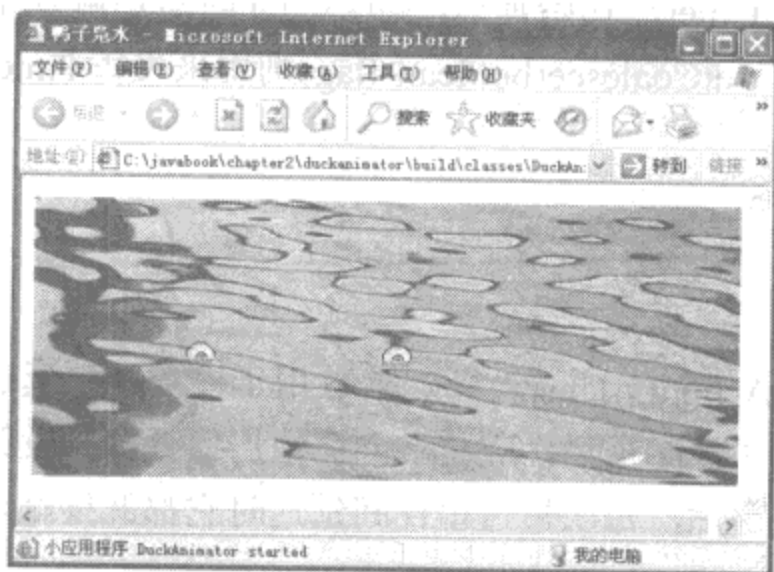


图2-6 鼠标单击画面后的界面

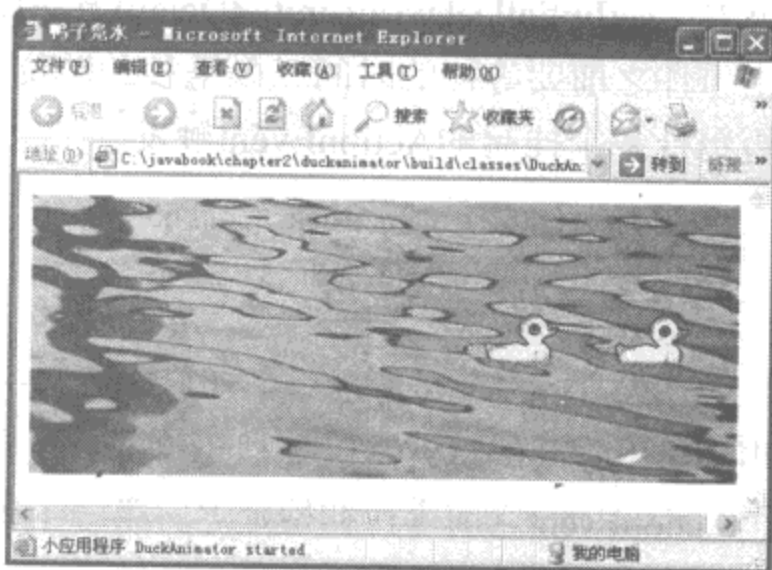


图2-7 用鼠标再次单击画面后的界面

制作要点

1. javax.swing.Timer类的应用技巧。

javax.swing.Timer类在指定时间间隔触发一个或多个ActionEvent事件。一个示例用法是用于动画对象，它将Timer用做绘制其帧的触发器。

设置计时器的过程包括创建一个Timer对象，并在该对象上注册一个或多个动作侦听器，以及使用start()方法启动该计时器。

2. 内部类的使用。

简单的说，内部（inner）类指那些类定义代码被置于其他类定义中的类；而对于一般的且类定义代码不嵌套在其他类定义中的类，称为顶层（top-level）类。对于一个内部类，包含其定义代码的类称为它的外部（outer）类。内部类实例可在其外部类的实例方法中创建。

3. 线程互斥：synchronized关键字的使用。

Java语言内置了synchronized关键字，用于对多线程进行同步，也能够对方法或者代码块进行同步。其原理是：当两个并发线程访问同一个对象时，一个时间内只能有一个线程得到执行，另一个线程必须等待，从而保证了线程安全。

步骤详解

1. 获取背景、建立用户界面：

```
bgImage = this.getImage(this.getCodeBase(), ". //image/bg.jpg");    //获取背景图像
fgImage = this.getImage(this.getCodeBase(), ". //image/duck3.gif");  //第一只鸭子
fgImage1 = this.getImage(this.getCodeBase(), ". //image/duck3.gif"); //第二只鸭子
buildUI(this.getContentPane(), bgImage, fgImage, fgImage1); //创建界面
```

设置窗口大小：

```
jframe.setSize(new Dimension(500,200));
jframe.setVisible(true); //显示窗口
```

2. 创建并启动计时器。

在Applet的init()方法中调用的buildUI方法体内使用以下代码创建了一个每隔一个间隔触发一次动作事件的计时器，这个间隔由Timer构造方法的第一个参数指定。然后在startAnimator()方法内启动该计时器。Timer构造方法的第二个参数指定接收计时器动作事件的侦听器。setInitialDelay(int initialDelay)方法设置Timer的初始延迟，即启动计时器后触发第一个事件之前要等待的时间（以毫秒为单位）。setCoalesce(boolean flag)则用来设置Timer是否组合多个挂起的ActionEvent触发。

```
timer = new Timer(delay, this);
timer.setInitialDelay(0);
timer.setCoalesce(true);
```

3. 构造Timer时要指定一个延迟参数和一个ActionListener。延迟参数用于设置初始延迟和事件触发之间的延迟（以毫秒为单位）。启动了计时器后，它将在向已注册侦听器触发第一个ActionEvent之前等待初始延迟。第一个事件之后，每次超过事件间延迟时它都继续触发事件，直到被停止。

```
animatorPane.addMouseListener(new MouseAdapter(){} )
```

4. 为DuckAnimator添加内部类，用于界面的初始化以及动画的实现：

```
class AnimatorPane extends JPanel
{
    Image background, foreground, foreground1;
    public AnimatorPane(Image background, Image foreground, Image foreground1) //构造
方法
    {...}
...}
```

5. 添加计时器触发的ActionEvent事件的处理方法：

```
public void actionPerformed(ActionEvent e)
{
    frameNumber++;           //鸭子右移
    animatorPane.repaint();  //重新绘制移动后的界面
}
```

6. 线程的互斥。

本案例中的startAnimator和stopAnimator方法分别对计时器对象Timer进行启动和停止操

作，二者是互斥的，所以在这两个方法声明时都加入了synchronized关键字。

```
public synchronized void startAnimator(){}
public synchronized void stopAnimator(){}
```

实际上synchronized是同步的意思，Java为每一个拥有synchronized方法的对象实例提供了一个唯一的管程。为了完成分配资源的功能，线程必须调用管程入口。管程入口就是synchronized方法入口。当调用同步方法时，该线程就获得了该管程。管程边界上实行严格的互斥，在同一时刻，只允许一个线程进入管程；当管程中已有了一个线程时，其他希望进入管程的线程必须等待，这种等待是由管程自动管理的。

线程的同步问题将在案例3中详细讲述。

7. 建立HTML文件，嵌入如下代码：

```
<html>
<head>
<title>鸭子凫水</title>
</head>
<body>
<applet code="DuckAnimator.class" WIDTH="500" HEIGHT="200">
</applet>
</body>
</html>
```

程序源代码与解释

```
/* * DuckAnimator.java */
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
public class DuckAnimator extends JApplet implements ActionListener
{
    int frameNumber = -1;
    boolean frozen = false;
    Timer timer;
    AnimatorPane animatorPane; //定义内部类实例
    Image bgImage,fgImage,fgImage1;
    //init() 方法
    public void init()
    {
        //参见步骤详解1
    }
    //创建用户界面
    void buildUI(Container container, Image bgImage, Image fgImage,Image fgImage1)
    {
        int fps = 10;
        int delay = (fps > 0)?(1000/fps):100;
        //设置计时器t
        timer = new Timer(delay, this);
        timer.setInitialDelay(0);
        timer.setCoalesce(true);
        animatorPane = new AnimatorPane(bgImage, fgImage,fgImage1); //创建内部类实例
        container.add(animatorPane, BorderLayout.CENTER);
        animatorPane.addMouseListener(new MouseAdapter(){ //添加鼠标事件侦听器
```



```

        public void mousePressed(MouseEvent e)
        {
            if(frozen) //鸭子停止前进时
            {
                frozen = false;
                startAnimator(); //启动动画，继续前进
            }
            else
            {
                frozen = true;
                stopAnimator(); //停止动画，原地凫水
            }
        }
    });
}

public void start()
{startAnimator(); //调用计时器对象，启动动画}
public void stop()
{stopAnimator(); //调用计时器对象，停止动画 }
public synchronized void startAnimator()
{
    if(frozen){}
    else
    {
        if(!timer.isRunning())
        {timer.start(); //启动Timer，使它开始向其侦听器发送动作事件 }
        }
    }
}
public synchronized void stopAnimator()
{
    if(timer.isRunning())
    {
        timer.stop(); //停止Timer，使它停止向其侦听器发送动作事件
    }
}
public void actionPerformed(ActionEvent e) //计时器触发的ActionEvent事件的处理方法
{
    //参见步骤详解5
}
//内部类，用于界面的初始化以及动画的实现
class AnimatorPane extends JPanel
{
    Image background, foreground, foreground1;
    public AnimatorPane(Image background, Image foreground, Image foreground1) //构造

    {
        this.background = background;
        this.foreground = foreground;
        this.foreground1 = foreground1;
    }
    //画出界面
    public void paintComponent(Graphics g)
    {
        super.paintComponents(g);
        int compWidth = this.getWidth();
        int compHeight = this.getHeight();
        int imageWidth, imageHeight;
        imageWidth = background.getWidth(this);
        imageHeight = background.getHeight(this);
        //画出背景图像
    }
}

```



```

        if((imageWidth>0) && (imageHeight>0))
        {
            g.drawImage(background,(compWidth-imageWidth)/2,
                (compHeight-imageHeight)/2,this);
        }
        //画出第一只鸭子
        imageWidth = foreground.getWidth(this);
        imageHeight = foreground.getHeight(this);
        if((imageWidth>0) && (imageHeight>0))
        {
            g.drawImage(foreground,((frameNumber*5)%(imageWidth+compWidth))-
                imageWidth, (compHeight-imageHeight)/2,this);

        }
        //画出第二只鸭子，其移动速度是第一只鸭子的2倍
        imageWidth = foreground1.getWidth(this);
        imageHeight = foreground1.getHeight(this);
        if((imageWidth>0) && (imageHeight>0))
        {
            g.drawImage(foreground1,((frameNumber*10)%(imageWidth+compWidth))-
                imageWidth, (compHeight-imageHeight)/2,this);

        }
    }
}
//main()方法
public static void main(String [] args)
{
    //获取背景和鸭子图像
    Image bgImage = Toolkit.getDefaultToolkit().getImage("./image/bg.jpg");
    Image fgImage = Toolkit.getDefaultToolkit().getImage("./image/duck3.gif");
    Image fgImage1 = Toolkit.getDefaultToolkit().getImage("./image/duck3.gif");
    JFrame jframe = new JFrame("鸭子晃水"); //设置标题
    final DuckAnimator controler = new DuckAnimator();
    controler.buildUI(jframe.getContentPane(),bgImage,fgImage,fgImage1); //创建界面
    jframe.addWindowListener(new WindowAdapter(){ //添加窗口事件侦听器
        public void windowIconified(WindowEvent e) //窗口最小化
        { controler.stopAnimator(); //停止动画 }
        public void windowDeiconified(WindowEvent e) //取消窗口最小化
        { controler.startAnimator(); //启动动画 }
        public void windowClosing(WindowEvent e) //关闭窗口
        { System.exit(0); //退出 }
    });
    //设置窗口大小，参见步骤详解1
    controler.startAnimator(); //启动动画
}
}

```



案例3：生产者-消费者模型的简单实现

案例运行效果与操作

生产者-消费者模型是常见的多线程同步互斥模型，本案例用Java对其进行了简单实现。程序运行后，在输出窗口中出现如图2-8所示的界面。


```

JUnit 测试结果
放入0
生产者#101生产0
取出0
消费者#200消费0
放入1
取出1
消费者#201消费1
生产者#101生产1
放入2
取出2
消费者#200消费2
生产者#101生产2
放入3
取出3
消费者#201消费3
生产者#101生产3
放入4
生产者#101生产4
取出4
消费者#200消费4
放入5
取出5
消费者#201消费5
生产者#101生产5

```

制作要点

1. Thread类的应用技巧。

2. 线程的（同步）控制。

一个Java程序的多线程之间可以共享数据。当线程以异步方式访问共享数据时，有时候是不安全的或者不合逻辑的。比如，同一时刻一个线程在读取数据，另外一个线程在处理数据，当处理数据的线程没有等到读取数据的线程读取完毕就去处理数据，必然得到错误的处理结果。如果我们采用多线程同步控制机制，等到第一个线程读取完数据，第二个线程才能处理该数据，就会避免错误。可见，线程同步是多线程编程的一个相当重要的技术。

（1）用Java关键字synchronized同步对共享数据操作的方法。

在一个对象中，用synchronized声明的方法为同步方法。Java中有一个同步模型——监视器，负责管理线程对对象中的同步方法的访问，它的原理是：赋予该对象唯一一把“钥匙”，当多个线程进入对象时，只有取得该对象钥匙的线程才可以访问同步方法，其他线程在该对象中等待，直到该线程用wait()方法放弃这把钥匙，其他等待的线程抢占该钥匙，抢占到钥匙的线程才可得以执行，而没有取得钥匙的线程仍被阻塞在该对象中等待。

（2）利用wait()、notify()及notifyAll()方法发送消息实现线程间的相互联系。

Java程序中多个线程通过消息来实现互动联系，wait()、notify()和notifyAll()方法实现了线程间的消息发送。例如定义一个对象的synchronized方法，同一时刻只能有一个线程访问该对象中的同步方法，其他线程被阻塞，通常可以用notify()或notifyAll()方法唤醒其他一个或所有线程，而使用wait()方法来使该线程处于阻塞状态，等待其他的线程用notify()唤醒。这三个函数由java.lang.Object类提供。

3. 生产者-消费者模型。

生产者-消费者模型是常见的多线程同步互斥模型。在生产者-消费者模型中，生产者（Producer）负责生产数据，而消费者（Consumer）负责使用数据。在生产者-消费者模型中，要保证以下几点。

（1）同一时间内只能有一个生产者生产。

（2）同一时间内只能有一个消费者消费。

（3）生产者生产的同时消费者不能消费。

4. 程序设计。

本案例由四个类构成，第一个类是Producer类，它继承Thread类，封装了生产者的逻辑，用来模拟生产者的生产过程，这个功能要求线程是安全的。

第二个类是Consumer类，也继承Thread类，封装了消费者的逻辑，用来模拟消费者的消费过程，即实现在没有冲突的情况下，从共享区中获取数据。

第三个类是Buffer类，是一个模拟临界缓冲区的类，封装了读出、写入数据的逻辑。

最后一个类是ProducerConsumerModel类，是一个测试程序，定义生产者、消费者线程，同时启动多个线程。

图2-8 运行界面

步骤详解

1. 创建线程:

```
public class Producer extends Thread {}
```

用继承Thread类的方法创建线程。

2. 为Producer类添加线程启动后的运行方法:

```
public void run() {           //线程启动后运行的方法
    for(int i=0;i<6;) {
        buffer.put(i);         //进行6次生产过程
        System.out.println("生产者#"+number+"生产"+(i++));
        try{
            Thread.sleep((int)(Math.random()*2000)); //线程休眠随机毫秒数
        } catch(InterruptedException exc){}
    }
}
```

3. 为Consumer类添加线程启动后的运行方法:

```
public void run() {           //线程启动后运行的方法
    for(int i=0;i<6;i++) {
        int v=buffer.get();     //进行6次消费过程
        System.out.println("消费者#"+number+"消费"+v);
    }
}
```

4. 同步控制写入和取出:

```
public synchronized int get() {};
public synchronized void put(int value) {};
```

Thread类对线程的管理还提供了suspend()、resume()和sleep()函数, 同样实现线程的阻塞、唤醒和停滞。那么它们和wait()、notify()和notifyAll()有什么区别呢?

两组函数的区别如下。

(1) wait()使当前线程进入停滞状态时, 还会释放当前线程所占有的“锁标志”, 从而使线程对象中的synchronized资源可被对象中别的线程使用; 而suspend()和sleep()使当前线程进入停滞状态时不会释放当前线程所占有的“锁标志”。

(2) wait()、notify()、notifyAll()这一组函数必须在synchronized函数或synchronized block中调用, 否则在运行时会产生错误; 而后一组函数可以通过non-synchronized函数和synchronized block中调用。

两组函数的取舍:

Java2已不建议使用suspend()、resume()和sleep()这一组函数, 因为在调用wait()时不会释放当前线程所取得的“锁标志”, 这样很容易造成“死锁”。

5. 不同步时, 线程休眠:

```
while(!available) {
    try{
        this.wait();           //线程休眠
    } catch(InterruptedException exc){}
}
```


6. 完成操作时，唤醒其他线程：

```
this.notifyAll();
```

程序源代码与解释

```
//生产者线程Producer.java
package producerconsumermodel;
public class Producer extends Thread {    //用继承Thread类的方法创建线程
    private Buffer buffer;
    private int number;
    public Producer(Buffer buffer,int number) {    //生产者构造方法
        this.buffer=buffer;
        this.number=number;
    }
    public void run() {                //线程启动后的运行方法
        //参见步骤详解2
    }
}

//消费者线程
package producerconsumermodel;
public class Consumer extends Thread {    //用继承Thread类的方法创建线程
    private Buffer buffer;
    private int number;
    public Consumer(Buffer buffer,int number) {    //消费者构造方法
        this.buffer = buffer;
        this.number = number;
    }
    public void run() {                //线程启动后的运行方法
        //参见步骤详解3
    }
}

//生产者与消费者共享的缓冲区，必须实现读、写的同步
package producerconsumermodel;
public class Buffer {
    private int contents;
    private boolean available=false;    //存取控制变量，实际上由信号量控制
    public synchronized int get() {    //同步方法控制临界区内容取出
        while(!available) {            //无法获得操作句柄时，线程休眠
            //参见步骤详解5
        }
        //获得操作句柄，进行取出操作
        int value=contents;
        available=false;    //消费者取出内容，改变存取控制available
        System.out.println("取出"+contents);
        this.notifyAll();    //取出完成，唤醒其他线程
        return value;
    }
    public synchronized void put(int value) {    //同步方法控制临界区内容写入
        while(available) {                //无法获得操作句柄时，线程休眠
            //参见步骤详解5
        }
        //获得操作句柄，进行写入操作
```



```

        contents=value;
        available=true;    //生产者放入内容, 改变存取控制available
        System.out.println("放入"+contents);
        this.notifyAll();  //写入完成, 唤醒其他线程
    }
}

//演示生产者-消费者问题的主程序
package producerconsumermodel;
public class ProducerConsumerModel {
    /**实例化 ProducerConsumerModel */
    public ProducerConsumerModel() {
    }
    public static void main(String[] args) {
        Buffer buffer=new Buffer();    //创建一个临界区对象
        new Producer(buffer,101).start();    //创建一个生产者对象, 并启动其线程
        new Consumer(buffer,200).start();    //创建一个消费者对象, 并启动其线程
        new Consumer(buffer,201).start();    //创建第二个消费者对象, 并启动其线程
    }
}

```



案例4: 定时关机

案例运行效果与操作

本案例是一个简单的定时关机程序, 在程序线程休眠设定的时间后, 通过Runtime类的exec方法调用关机程序, 实现了根据用户设定时间自动关机的功能。程序运行后, 界面如图2-9所示。

此时依次设置时、分、秒钟, 单击“确定”按钮后界面如图2-10所示。



图2-9 运行时界面

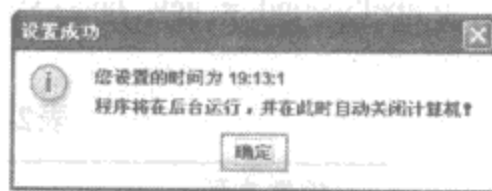


图2-10 设置成功界面

如果设置的时间当天已经过去 (比如在10点钟的时候设置时间为9:30), 则将第二天的相应时间作为关机时间, 界面如图2-11所示。

当设置时间到达时, 会弹出“系统关机”提示框, 并在1分钟内进行关机, 如图2-12所示。

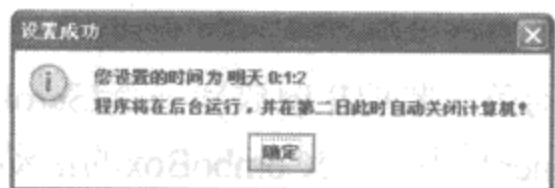


图2-11 设置已过时间的界面

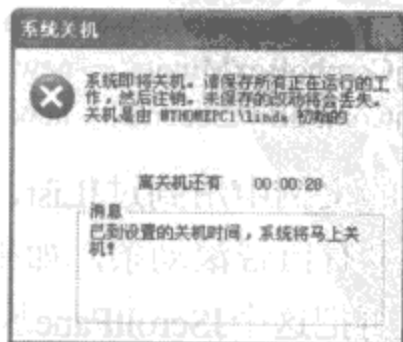


图2-12 关机界面

制作要点

- 1. `ActionEvent`的应用技巧。
- 2. `JComboBox`的应用技巧。
- 3 `Runtime`类的使用。

每个Java应用程序都有一个`Runtime`类实例，使应用程序能够与其运行的环境相连接，可以通过`getRuntime`方法获取与当前Java应用程序相关的`Runtime`对象，然后通过调用该对象的`exec`方法在单独的进程中执行指定的字符串命令。由于`exec`方法可能产生`IOException`异常，所以必须进行异常捕获。

```
Runtime.getRuntime().exec("shutdown.exe -s -c \"已到设置的关机时间，系统将马上关机！\"");
```

步骤详解

- 1. 设置窗体：将大小设置为“180,150”，如图2-13所示。关于`JFrame`的创建和定义在第1章各案例中有详细的讲解，这里不再赘述。

```
setMinimumSize(new java.awt.Dimension(180, 150));
```



图2-13 在属性窗口中修改属性

- 2. 添加`JPanel`控件`JPanelTime`，并向其中添加3个`JLabel`，按照表2-1所示修改属性。

```
jLabelHour = new javax.swing.JLabel();
jLabelMinute = new javax.swing.JLabel();
jLabelSecond = new javax.swing.JLabel();
```

表2-1 JLabel属性与值对照表

控件类型	控件名称	text属性值
JLabel	JLabelHour	设置时钟:
JLabel	JLabelMinute	设置分钟:
JLabel	JLabelSecond	设置秒钟:

- 3. 向`JPanelTime`中添加3个`JComboBox`，并按照表2-2所示修改属性。

```
jComboBoxHour = new javax.swing.JComboBox();
jComboBoxMinute = new javax.swing.JComboBox();
jComboBoxSecond = new javax.swing.JComboBox();
```

在前面的案例中用到过`JList`，下面简单介绍一下`JComboBox`和`JList`的区别。

- `JList`没有自带滚动条，如果想让它自动出现滚动条，要把`JList`放到一个`JScrollPane`之中，再把这个`JScrollPane`放在窗体容器（`Container`）中。而`JComboBox`的滚动条是自动出现的，不用特别设定。

表2-2 JComboBox属性与值对照表

控件类型	控件名称	Model属性值
JComboBox	JComboBoxHour	空
JComboBox	JComboBoxMinute	空
JComboBox	JComboBoxSecond	空

- JList里的Item是用DefaultListModel来操作的，建立JList时，先建立一个DefaultListModel，可以通过DefaultListModel对JList的内容进行操作。而JComboBox相对简单多了。
 - 常用函数有：
 setEditable()设定是否可编辑。
 addItem加入新的Item。
 getItemCount得到所有Item数目。
 getSelectIndex得到选择的Item的Index，以0开始。
4. 调整各个控件实例在窗体上的位置与大小后，界面最终设置效果如图2-14所示。

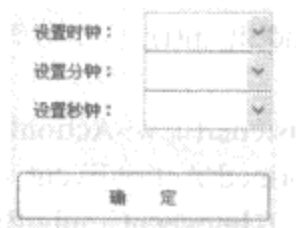


图2-14 最终界面设置效果

5. 隐藏窗口:

```
this.setVisible(false)
```

6. 获取用户设定的时间值:

```
hour = ((Integer)jComboBoxHour.getSelectedItem()).intValue(); //获取用户设定的时钟值
minute = ((Integer)jComboBoxMinute.getSelectedItem()).intValue(); //获取用户设定的分钟值
second = ((Integer)jComboBoxSecond.getSelectedItem()).intValue(); //获取用户设定的秒钟值
totalSeconds = hour * 3600 + minute * 60 + second; //获取用户设定时间对应的总秒数
```

7. 运行关机程序:

```
Runtime.getRuntime().exec("shutdown.exe -s -c \"已到设置的关机时间，系统将马上关机！\"");
```

程序源代码与解释

```
/* * ShutDownPC.java*/
package shutdownpc;
public class ShutDownPC {
    /** 实例化ShutDownPC */
    public ShutDownPC() {
    }
    public static void main(String[] args) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new FrameShutDownPC().setVisible(true);
            }
        });
    }
}
```



```

    });
}
}
/* * FrameShutDownPC.java*/
package shutdownpc;
public class FrameShutDownPC extends javax.swing.JFrame {

    /**实例化FrameShutDownPC */
    public FrameShutDownPC() {
        initComponents();
        for(int i = 0; i < 24; i++) {
            jComboBoxHour.addItem(i);    //时钟复合框初始化
        }
        jComboBoxHour.addActionListener(new ActionListener() { //时钟复合框事件处理
            public void actionPerformed(ActionEvent evt) {
                hour = ((Integer)jComboBoxHour.getSelectedItem()).intValue();    //获取用户
设定的时钟值
            }
        });
        for(int i = 0; i < 60; i++) {
            jComboBoxMinute.addItem(i);    //分钟复合框初始化
        }
        jComboBoxMinute.addActionListener(new ActionListener() {    //分钟复合框事件处理
            public void actionPerformed(ActionEvent evt) {
                minute = ((Integer)jComboBoxMinute.getSelectedItem()).intValue();    //获取
用户设定的分钟值
            }
        });
        for(int i = 0; i < 60; i++) {
            jComboBoxSecond.addItem(i);    //秒钟复合框初始化
        }
        jComboBoxSecond.addActionListener(new ActionListener() {    //秒钟复合框事件处理
            public void actionPerformed(ActionEvent evt) {
                second = ((Integer)jComboBoxSecond.getSelectedItem()).intValue();    //获取
用户设定的秒钟值
            }
        });
        jButtonOK.addActionListener(new ActionListener() {    //“确定”按钮事件处理
            public void actionPerformed(ActionEvent evt) {
                calendar = new GregorianCalendar();
                int currentSeconds = calendar.get(Calendar.HOUR_OF_DAY) * 3600+
calendar.get(Calendar.MINUTE) * 60+ calendar.get(Calendar.SECOND);    //获取当前时间对应的总秒数
                totalSeconds = hour * 3600 + minute * 60 + second;    //获取用户设定
时间对应的总秒数
                if (totalSeconds - currentSeconds < 0) {    //如果用户设定时间在当天已
经过去, 则将关机时间设置为明天的对应时间
                    argue = (24 * 3600 + currentSeconds - totalSeconds)*1000;    //计算距离关机时间相
差的毫秒数
                    String s=new String("您设置的时间为明天 " + hour + ":" + minute +
":" + second + "\n程序将在后台运行, 并在第二日此时自动关闭计算机!");
                    JOptionPane.showMessageDialog(FrameShutDownPC.this,s,"设置成功",JOptionPane
.INFORMATION_MESSAGE);    //弹出提示对话框
                    hideFrame();    //隐藏窗体
                }
            }
        });
    }
}

```



```

else { //否则就设置为当天的对应时间
    argue = (totalSeconds - currentSeconds) * 1000; //计算距离关机时间相差的毫秒数
    String s=new String("您设置的时间为 " + hour + ":" + minute + ":"
+ second + "\n程序将在后台运行, 并在此时自动关闭计算机!");
    JOptionPane.showMessageDialog(FrameShutDownPC.this,s,"设置成功",JOptionPane
.INFORMATION_MESSAGE); //弹出提示对话框
    hideFrame(); //隐藏窗体
    }
    try {
        Thread.sleep(argue); //线程休眠相应的时间
        Runtime.getRuntime().exec("shutdown.exe -s -c \"已到设置的关机时间, 系统将
马上关机!\""); //运行关机程序进行关机
    } catch (Exception e) {
        e.printStackTrace();
    }
}
});
}
private void initComponents() {
//组件初始化, 代码略
}
private void hideFrame() {
    this.setVisible(false);
}
private int hour,minute,second,totalSeconds,currentSeconds;
private long argue;
private GregorianCalendar calendar;
private boolean change = true;
//变量声明略

```



案例5: 多线程TCP端口扫描程序

案例运行效果与操作

本案例使用多线程对目标地址或域名进行TCP端口扫描, 并将扫描得到的开放端口显示出来。程序运行后, 界面如图2-15所示。

首先上面的几个文本框中输入目标地址、端口和同时进行扫描的线程数, 单击“确定”按钮, 就开始进行扫描, 界面如图2-16所示。

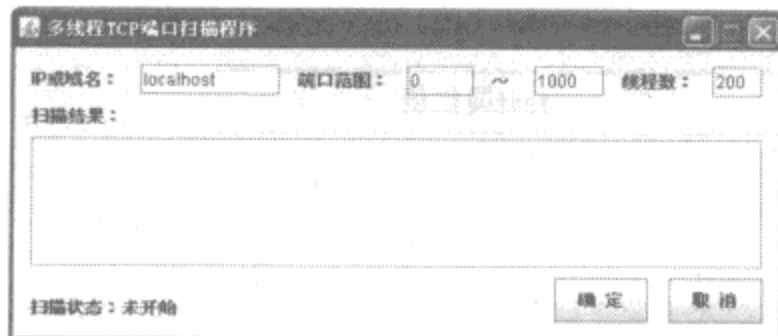


图2-15 运行时界面

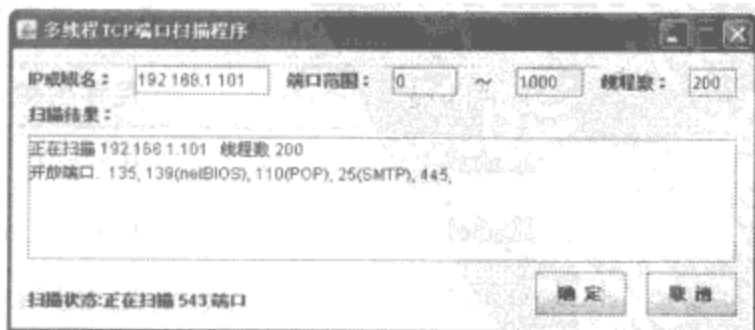


图2-16 扫描界面

扫描得到的结果在中间的文本区显示出来, 同时在界面的左下方实时显示扫描状态。扫描结束后, 界面如图2-17所示。

同时，本程序还具有用户输入合法性检查功能。用户输入不合法，会弹出错误提示对话框。例如，在“IP或域名”文本框中输入非法地址（如localhos等）后，弹出如图2-18所示的对话框。

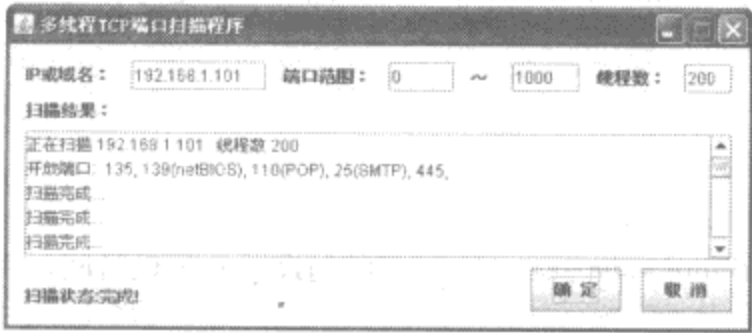


图2-17 扫描结束时界面

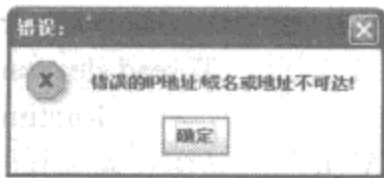


图2-18 错误提示对话框

单击“取消”按钮，或者单击窗口右上方的“关闭”按钮，就会退出程序。

制作要点

- 1. Thread类的用法。
- 2. java.net.Socket类的用法。

Socket类实现客户端套接字（也可以称做“套接字”）。套接字是两台机器间通信的端点。本案例使用Socket（InetAddress address, int port）构造方法，创建一个流套接字并将其连接到指定地址的指定端口号，成功则表示该端口号开放，不成功则抛出IOException异常。接着用多个线程逐个去测试指定端口范围内所有的端口，最后得到所有的开放端口。

```
theTCPsocket=new Socket(hostAddress,i);
```

- 3. java.net.InetAddress类的用法。

InetAddress类表示互联网协议（IP）地址。本案例使用其getByName(String host)方法，在给 定主机名的情况下确定主机的IP地址。主机名可以是机器名（如“java.sun.com”或“localhost”等），也可以是其IP地址的文本表示形式。该方法可能产生UnknownHostException异常，需要进行捕获。

```
TCPThread.hostAddress=InetAddress.getByName(FrameTCPScan.jTextFieldIP.getText());
```

步骤详解

- 1. 界面设计：添加6个JLabel、4个JTextField和2个JButton，并按照表2-3所示修改属性。

表2-3 属性与值对照表

控件类型	控件名称	text属性值
JLabel	JLabelIP	IP或域名:
JLabel	JLabelPort	端口范围:
JLabel	JLabelTo	~
JLabel	JLabelThread	线程数:
JLabel	JLabelResult	扫描结果:
JLabel	JLabelStatus	扫描状态: 未开始

(续表)

控件类型	控件名称	text属性值
TextField	TextFieldIP	localhost
TextField	TextFieldMinPort	0
TextField	TextFieldMaxPort	1000
TextField	TextFieldThread	200
Button	ButtonSubmit	确定
Button	ButtonCancel	取消

调整各个控件实例在窗体上的位置与大小后，界面最终设置效果如图2-19所示。



图2-19 最终界面设置效果

2. 用继承Thread类的方法创建线程:

```
class TCPThread extends Thread {}
```

3. 获取输入内容。

用户输入的要扫描的IP地址:

```
TCPThread.hostAddress=InetAddress.getByName(FrameTCPScan.jTextFieldIP.getText());
```

用户输入的要扫描的端口范围及线程数目:

```
minPort=Integer.parseInt(FrameTCPScan.jTextFieldMinPort.getText());
maxPort=Integer.parseInt(FrameTCPScan.jTextFieldMaxPort.getText());
maxThread=Integer.parseInt(FrameTCPScan.jTextFieldThread.getText());
```

4. 创建一个流套接字并将其连接到指定地址的指定端口号，成功则表示该端口号开放，i表示端口号:

```
theTCPsocket=new Socket(hostAddress,i);
theTCPsocket.close(); //关闭此套接字
```

在Java.net包中，Socket类就是对Socket的具体实现。它通过连接到主机后，返回一个I/O流，实现协议间的信息交换。使用完毕后，Socket必须关闭。

Socket类包含了许多有用的方法，在后面的案例中还会用到。比如:

- getLocalAddress()返回一个包含客户程序IP地址的InetAddress子类对象的引用;
- getLocalPort()返回客户程序的端口号;
- getInetAddress()返回一个包含服务器IP地址的InetAddress子类对象的引用;
- getPort()将返回服务程序的端口号等。

5. 启用多线程扫描，线程数量是用户输入的：

```
for(int i=0;i<maxThread;i++) {
    new TCPThread("T" + i).start();
}
```

6. 为JButtonCancel按钮添加ActionEvent事件的处理方法：

```
System.exit(1);    //单击“取消”按钮，系统退出
```

程序源代码与解释

```
/* * TCPScan.java*/
package tcpscan;
public class TCPScan {
    /** 实例化 TCPScan */
    public TCPScan() {
    }
    public static void main(String[] args) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new FrameTCPScan().setVisible(true);
            }
        });
    }
}

/* * FrameTCPScan.java*/
package tcpscan;
public class FrameTCPScan extends javax.swing.JFrame {
    /** 实例化TCPScan */
    public FrameTCPScan() {
        initComponents();
    }
    private void initComponents() {
        //组件初始化，代码略
    }
    private void jButtonCancelActionPerformed(java.awt.event.ActionEvent evt) {
        System.exit(1);    //单击“取消”按钮系统退出
    }
    private void jButtonSubmitActionPerformed(java.awt.event.ActionEvent evt) {
        int minPort;
        int maxPort;
        int maxThread;
        try { //获取用户输入的端口号和线程数
            //参见步骤详解3
        } catch (NumberFormatException e) { //用户输入错误则弹出对话框
            JOptionPane.showMessageDialog(this,"错误的端口号或线程数!端口号和线程数必须为
            整数!", "错误:", JOptionPane.ERROR_MESSAGE);
            FrameTCPScan.jLabelStatus.setText("扫描状态:未开始");
            return;
        }
        try { //获取用户输入的IP地址或域名
            TCPThread.hostAddress=InetAddress.getByName(FrameTCPScan.jTextFieldIP.getText());
        } catch (UnknownHostException e) { //用户输入错误则弹出对话框
```



```

JOptionPane.showMessageDialog(this,"错误的IP地址/域名或地址不可达!","错误",JOptionPane.ERROR_MESSAGE);
    FrameTCPScan.jLabelStatus.setText("扫描状态:未开始");
    return;
}
if(minPort<0 || minPort>65535 || minPort>maxPort) { //对用户输入的起始端口号进行正确性检查, 错误则弹出对话框
    JOptionPane.showMessageDialog(this,"最小端口必须是0~65535并且小于最大端口的整数!","错误:",JOptionPane.ERROR_MESSAGE);
    FrameTCPScan.jLabelStatus.setText("扫描状态:未开始");
    return;
}
else TCPThread.MIN_port=minPort;
if(maxPort<0 || maxPort>65535 || maxPort<minPort) { //对用户输入的终止端口号进行正确性检查, 错误则弹出对话框
    JOptionPane.showMessageDialog(this,"最大端口必须是0~65535并且大于最小端口的整数!","错误:",JOptionPane.ERROR_MESSAGE);
    FrameTCPScan.jLabelStatus.setText("扫描状态:未开始");
    return;
}
else TCPThread.MAX_port=maxPort;
if(maxThread<1 || maxThread>200) { //对用户输入的线程数进行正确性检查, 错误则弹出对话框
    JOptionPane.showMessageDialog(this,"线程数为1~200的整数!","错误:",JOptionPane.ERROR_MESSAGE);
    FrameTCPScan.jLabelStatus.setText("扫描状态:未开始");
    return;
}
FrameTCPScan.jTextAreaResult.setText("");
FrameTCPScan.jTextAreaResult.append("正在扫描 "+FrameTCPScan.jTextFieldIP.getText()+" 线程数 "+FrameTCPScan.jTextFieldThread.getText()+"\n");
FrameTCPScan.jTextAreaResult.append("开放端口: ");
for(int i=0;i<maxThread;i++) {
    new TCPThread("T" + i,i).start();//启动多线程进行端口扫描, 线程数量为用户输入的线程数
}
}
//变量声明(略)
}
class TCPThread extends Thread { //用继承Thread类的方法创建线程
    public static InetAddress hostAddress;
    public static int MIN_port;
    public static int MAX_port;
    private int threadnum;
    public TCPThread(String name,int threadnum) { //构造方法
        super(name);
        this.threadnum = threadnum;
    }
    public void run() { //线程启动后运行的代码
        int i;
        Socket theTCPsocket;
        for (i = MIN_port+threadnum; i < MAX_port; i += Integer.parseInt(FrameTCPScan.jTextFieldThread.getText())) {
            FrameTCPScan.jLabelStatus.setText("扫描状态:正在扫描 "+i+" 端口");

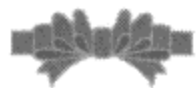
```


开放

```

try {
    //创建一个流套接字并将其连接到指定地址的指定端口号，成功则表示该端口号
    theTCPsocket=new Socket(hostAddress,i);
    theTCPsocket.close();          //关闭此套接字
    FrameTCPScan.jTextAreaResult.append(" "+i);
    switch(i) {                    //对常用的端口进行说明
        case 21:
            FrameTCPScan.jTextAreaResult.append("(FTP)");
            break;
        case 23:
            FrameTCPScan.jTextAreaResult.append("(TELNET)");
            break;
        case 25:
            FrameTCPScan.jTextAreaResult.append("(SMTP)");
            break;
        case 80:
            FrameTCPScan.jTextAreaResult.append("(HTTP)");
            break;
        case 110:
            FrameTCPScan.jTextAreaResult.append("(POP)");
            break;
        case 139:
            FrameTCPScan.jTextAreaResult.append("(netBIOS)");
            break;
    }
    FrameTCPScan.jTextAreaResult.append(",");
} catch (IOException e) {}
}
if (i>=MAX_port) { //扫描完毕给出提示信息
    FrameTCPScan.jTextAreaResult.append("\n"+"扫描完成...");
    FrameTCPScan.jLabelStatus.setText("扫描状态:完成!");
}
}
}

```



案例6：一个简单的年历生成程序

案例运行效果与操作

本案例是一个简单的年历生成程序，它根据用户的输入，能够生成指定年份的年历，包括十二个月的月历等内容。

程序启动后，出现如图2-20所示的界面。

```

G:\javabook\chapter2\mycalendar\dist>java -jar mycalendar.jar
请输入一个数值表示的年份

```

图2-20 DOS窗口运行时界面

此时输入一个数值，比如2000，就会自动生成2000年的年历，它包括12个月的月历，如图2-21所示。

如果在输入年份时，没有输入数值，而是输入了其他字符或者没有输入数值直接按Enter键，则会给出提示，同时自动生成系统当前年份的年历，如图2-22所示。

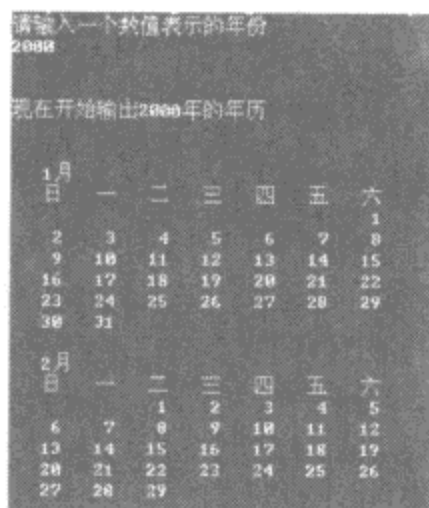


图2-21 生成年历后的部分界面

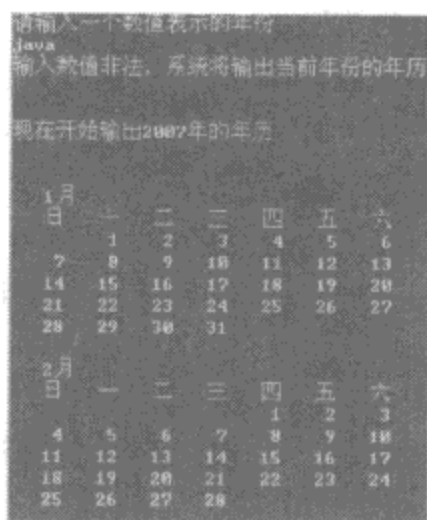


图2-22 默认生成界面

制作要点

1. Calendar类的应用技巧。

Calendar类是一个抽象类，它为特定瞬间与一组诸如YEAR、MONTH、DAY_OF_MONTH、HOUR等日历字段之间的转换提供了一些方法，并为操作日历字段（例如获得下星期的日期）提供了一些方法。直接使用其对象会要求实现所有的抽象方法，所以本案例使用了GregorianCalendar类来创建对象。GregorianCalendar类是Calendar类的一个具体子类，提供了世界上大多数国家和地区使用的标准日历系统。

```
year=new GregorianCalendar().get(Calendar.YEAR);//默认年份为系统当前年份
```

2. 非法输入的输入控制。

本案例要求用户输入一个数值表示的年份，必须考虑非法输入的处理。案例中使用的parseInt静态方法将字符串参数作为有符号的十进制整数进行解析，它要求用户输入的字符可以是用来表示负值的ASCII减号“-”外，字符串中的字符都必须是十进制数字，否则就会抛出NumberFormatException异常，因此使用了try-catch语句对异常进行捕获处理。

步骤详解

1. Java中提供两个抽象类来表示时间及其计算：java.util.Date和Calendar。

Date类实际上只是一个包裹类，它包含的是一个长整型数据，表示的是从GMT(格林尼治标准时间)1970年1月1日00:00:00这一刻之前或者是之后经历的毫秒数。再通过其他构造函数获取系统日期。Date类在许多方面实现复杂而且不尽如人意。

自JDK1.1引入的Calendar类是另一种不同类型的日期处理类。Calendar类的基础即有变量域的概念。每个类元素都是域，并且这些域在Calendar类中表现为静态变量。这些变量域，可以通过get/set类方法来获得或者设置域值。java.util.Calendar类看上去更像是Date类的复杂版本，它还提供额外的功能。

```
year=new GregorianCalendar().get(Calendar.YEAR)
```

2. 创建流对象，取得用户输入：


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

3. 闰年处理:

```
if((year%4==0 && year%100!=0)||year%400==0) //闰年  
    leapyear=1; //闰年时2月份天数加1  
else  
    leapyear=0;
```

实际上, **Calendar**类提供了判断是否为闰年的功能性方法**isLeapYear()**, 如果是非闰年, 返回**false**, 读者感兴趣的话可以试一下:

```
Calendar cal = Calendar.getInstance();  
booleanleapYear = ( (GregorianCalendar)cal ).isLeapYear(year);
```

4. 异常处理:

```
try{  
    year=Integer.parseInt(s); //将用户输入的字符串作为有符号的十进制整数进行解析  
}catch (Exception e){}
```

在前面的案例中, 用到了许多异常抛出, 如**IOException**异常、**ClassNotFoundException**异常、**SQLException**异常、**InterruptedException**异常、**UnknownHostException**异常等。下面简单分析一下Java中异常处理的方式。

(1) 内层方法抛出异常, 外层方法捕捉并处理异常:

```
public void methodA (){  
    try{ //调用methodB  
        methodB();  
    }  
    catch(ExceptionType et){ //相应处理措施  
    }  
}  
  
public void methodB throws ExceptionType{  
    if (condition?is?true)  
    { //相应处理措施  
    }  
    else {  
        throw new ExceptionType(argument);  
    }  
}
```

在这个例子中, 方法B的头部中声明了该方法会抛出一个类型为**ExceptionType**的异常, 在方法体中使用**throw**子句抛出了一个异常, 那么该异常被方法A捕捉。因为异常抛出后, JVM会顺着该方法的调用栈一层一层地往上找, 因为方法A中有一个**catch(ExceptionType et)**, 所以被抛出的异常会被捕捉到并被处理。

(2) 方法中自己捕捉, 处理异常:

```
public methodA() {  
    try  
    { //可能产生异常的语句 }
```



```

catch (ExceptionType et)
{ //相应的处理}
}

```

在这个例子中，方法A使用了try-catch语句块，也就是在方法中所产生的ExceptionType类型的异常都会被捕捉到并在方法内处理。本案例采用的就是这种异常捕捉方法。

(3) 内层方法抛出一个异常，但本身又有try-catch:

```

public?methodB() throws ExceptionType{ try
{ //可能产生异常的语句 }
catch (AnotherExceptionType aet)
{ //相应处理措施}
}

```

在这个例子中有两种异常处理情况，抛出异常和捕捉异常。如果在try语句块中产生ExceptionType类型的异常，会被抛出；如果产生AnotherExceptionType类型的话，则不会被抛出，因为在方法B的头部中并没有声明会抛出该异常。

(4) 内层方法抛出一个异常，但本身有try-finally:

```

public methodB() throws ExceptionType{
try { //可能产生异常的语句}
finally { //一定要执行的语句}
}

```

这个例子与上一个例子很像，不同的是没有catch，但增加了finally。如果方法B的try语句块中产生了异常，则抛出且由外层方法处理，然后方法B继续执行finally中的语句。

程序源代码与解释

```

/* * MyCalendar.java */
public class MyCalendar {
    /**实例化 MyCalendar */
    public MyCalendar() {
    }
    public static void main(String[] args) throws IOException {
        CalendarOutput myCalendar=new CalendarOutput();
        int year,leapyear,y,i;
        BufferedReader in; //创建流对象，用来获取用户输入
        in=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("请输入一个数值表示的年份");
        String s=in.readLine(); //获取用户输入值
        try{
            year=Integer.parseInt(s); //将用户输入的字符串作为有符号的十进制整数进行解析
        }catch (Exception e){
            System.out.println("输入数值非法，系统将输出当前年份的日历");
            year=new GregorianCalendar().get(Calendar.YEAR); //默认年份为系统当前年份
        }
        //闰年处理，参见步骤详解3
        y=year;
        for(i=1;i<year;i++){
            if((i%4==0 && i%100!=0)||i%400==0)
                y++; //计算输入年份之前共有多少个闰年
        }
    }
}

```



```

y=y%7;          //计算输入年份的1月1日是第一周的哪一天
System.out.println("\n");          //加一空行
System.out.println("现在开始输出"+year+"年的日历");
System.out.println("\n");          //加一空行
//调用output方法, 依次输出各月的月历
for(i=1;i<13;i++){
    switch(i){
        case 1: { //输出1月份月历
            myCalendar.output(1,y,31);          //调用output方法, 输出月历, 下同
            y=(y+31)%7;          //计算下月的1日是星期几, 下同
            break;
        }
        case 2: { myCalendar.output(2,y,28+leapyear);y=(y+28+leapyear)%7;break;}
        case 3: { myCalendar.output(3,y,31);y=(y+31)%7;break;}
        case 4: { myCalendar.output(4,y,30);y=(y+30)%7;break;}
        case 5: { myCalendar.output(5,y,31);y=(y+31)%7;break;}
        case 6: { myCalendar.output(6,y,30);y=(y+30)%7;break;}
        case 7: { myCalendar.output(7,y,31);y=(y+31)%7;break;}
        case 8: { myCalendar.output(8,y,31);y=(y+31)%7;break;}
        case 9: { myCalendar.output(9,y,30);y=(y+30)%7;break;}
        case 10: { myCalendar.output(10,y,31);y=(y+31)%7;break;}
        case 11: { myCalendar.output(11,y,30);y=(y+30)%7;break;}
        case 12: { myCalendar.output(12,y,31);y=(y+31)%7;break;}
    }
}
}
}

/* * CalendarOutput.java */
public class CalendarOutput {
    /** Creates a new instance of CalendarOutput */
    public CalendarOutput() {
    }
    public void output(int month,int daysofweek,int dayspermonth){
        int i;
        int a[]= new int[40];
        System.out.println("    "+month+"月");
        System.out.println("    日    一    二    三    四    五    六 ");
        for (i=0;i<daysofweek;i++){
            System.out.print("    ");          //每月1号之前的位置填充空白
        }
        for(i=daysofweek;i<daysofweek+dayspermonth;i++){
            a[i]=i-daysofweek+1;          //a[i]是从1到每月的天数
        }
        for(i=daysofweek;i<daysofweek+dayspermonth;i++){
            if (i%7==0)
                System.out.print("\n");          //每周进行换行
            if (a[i]<10)
                System.out.print("    "+a[i]);          //一位数字的日期, 前面多一个空格
            else
                System.out.print("    "+a[i]);          //二位数字的日期
        }
        System.out.println("\n");          //每月最后加一空行
    }
}
}

```




案例7：将GIF和JPG图像转换成VRML格式

案例运行效果与操作

VRML (Virtual Reality Modeling Language, 虚拟现实建模语言) 被称为第二代Web语言, 是一种基于文本的通用语言, 是在网络上使用的描述三维环境的场景描述语言, 是HTML的3D (三维) 模拟。它定义了3D应用中大多数常见概念, 如光源、视点、动画、雾化、材质属性、纹理映射等。VRML语言的诞生改变了原来WWW上单调、平面的缺点, 将人的行动作为浏览的主体, 所有的表现都将随操作者的行为而改变。

VRML文件是以扩展名.wrl结尾的一种用来描述几何形体的ASCII文本文件, 它不需要任何编译, 直接由浏览器解释执行。当用户打开VRML文件时, 系统首先装入一个内嵌的VRML浏览器, 该浏览器将VRML语言中的信息解释成空间中目标的几何形体描述, 如长方体、球体、不规则的其他三维物等, 同时它将提供实时显示, 一秒显示多次, 这样在用户的计算机上就会有一个活动场景的感觉。

本案例使用Java实现了将GIF和JPG图像转换成VRML格式的功能。程序运行后, 弹出选择文件对话框, 提示用户选择要转换的GIF或者JPG图像文件, 界面如图2-23所示。

选择相应的图像文件, 单击“打开”按钮, 程序自动将其转换成VRML格式。此时打开相应目录, 发现该目录下多出了一个以扩展名.wrl结尾的文件, 如图2-24所示。

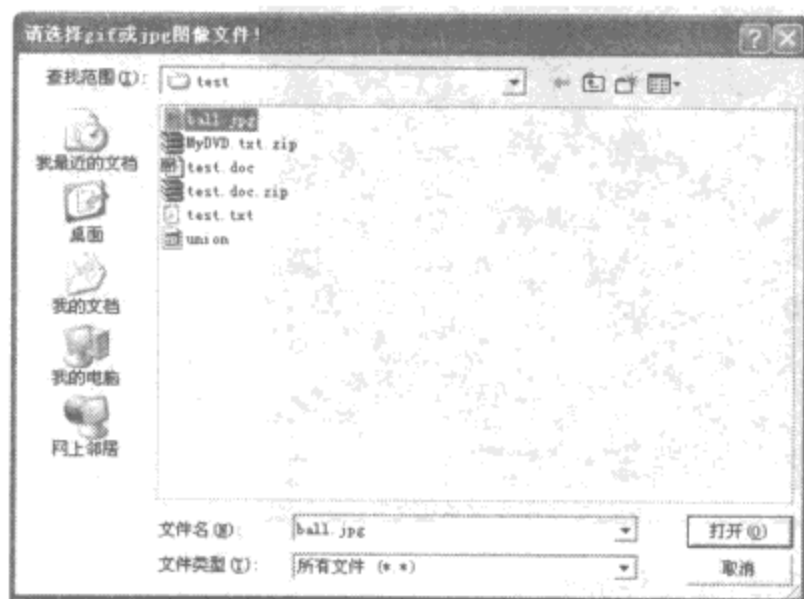


图2-23 运行时界面

名称	大小	类型
ball.jpg	4 KB	JPEG 图像
test.doc	32 KB	Microsoft Word ...
MyDVD.txt.zip	3 KB	WinRAR ZIP 压缩 ...
test.doc.zip	8 KB	WinRAR ZIP 压缩 ...
test.txt	1 KB	文本文档
union	32 KB	文件
ball.jpg.wrl	89 KB	VRML World

图2-24 生成的VRML文件

双击该文件将其打开, 如果浏览器安装了VRML浏览器插件, 则可以正常浏览该文件, 界面如图2-25所示。

VRML的动画比普通动画更吸引人, 因为当用户在一个虚拟世界中漫游时, 他可以从各种角度来观察动画, 而要完成一个复杂的从所有视角来看都成功的动画是很困难的。VRML中的动画产生是变动了任何一个坐标系的位置、方向和形体比例, 从而使物体按需要的方式飞行、平移、旋转或按比例缩放。如图2-26所示即为对生成的文件进行按比例缩放的效果。

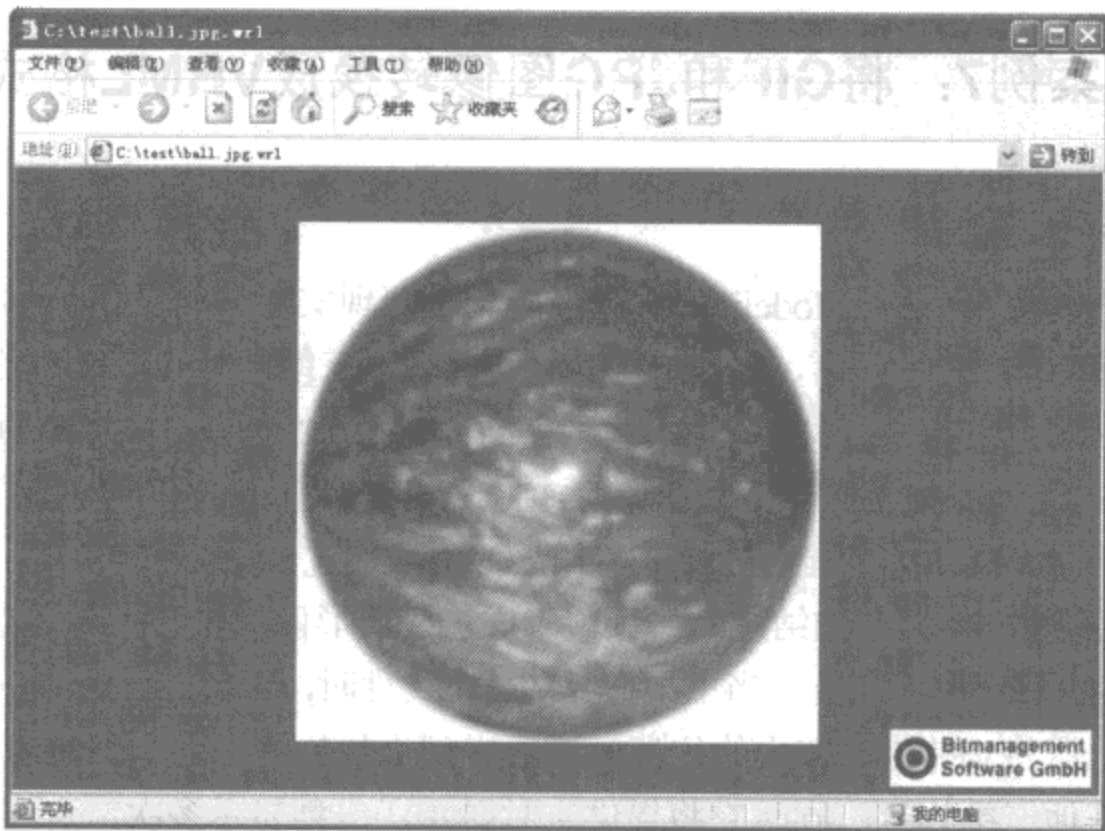


图2-25 浏览文件界面

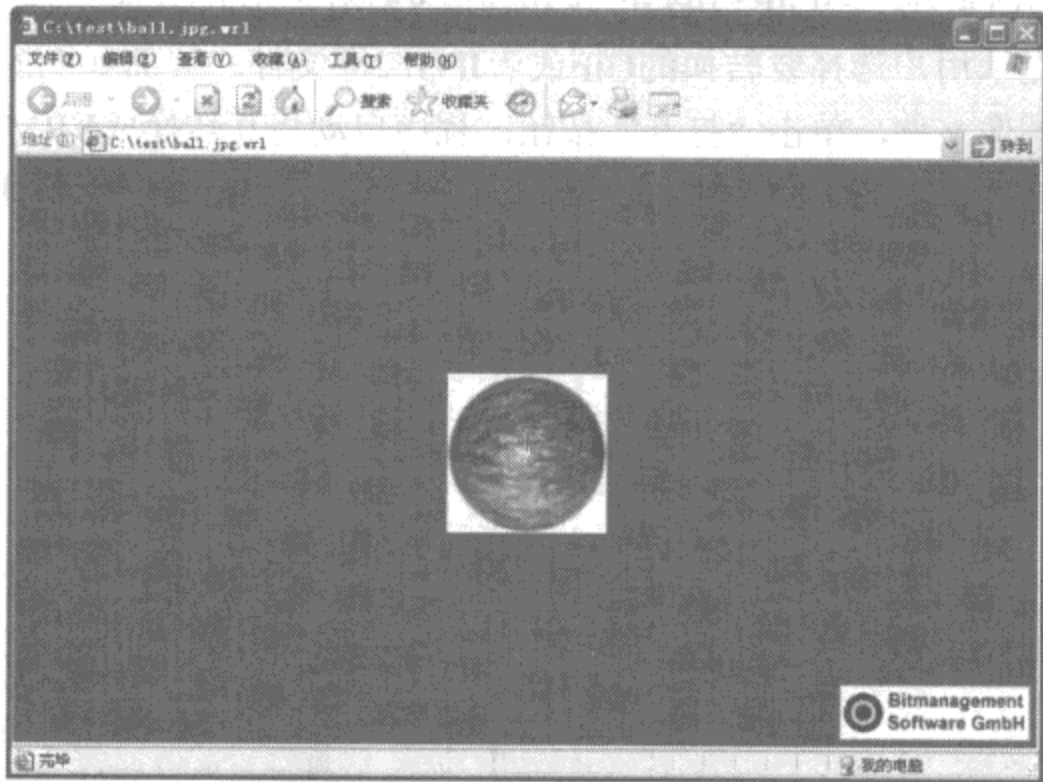


图2-26 按比例缩放文件界面

制作要点

- 1. 线程的应用技巧。
- 2. MediaTracker类的应用技巧。
- 3. 文件操作。
- 4. PixelGrabber类的应用技巧。

步骤详解

- 1. 设置窗口布局:


```
setLayout(null);
setSize(img.getWidth(this),img.getHeight(this));
setVisible(true);
```

2. 为Image2VRML类编写线程的运行方法:

```
public void run() {    //线程的运行方法
    try{
        //检查媒体跟踪器跟踪的ID为0的图像是否已完成加载
        mtrack.waitForID(0);
    }
    catch(Exception e){ }
    repaint(); //重画
}
```

3. 创建对话框对象, 用户通过对话框选择要转换的图像文件:

```
FileDialog fd = new FileDialog(this,"请选择GIF或JPG图像文件!");
fd.setMode(FileDialog.LOAD);
fd.setVisible(true);    //显示对话框
theFile = fd.getFile();
theDir = fd.getDirectory();
if(theDir != null && theDir.length() != 0 && theFile != null && theFile.length() != 0){
    return (theDir+theFile);    //返回选择的文件全路径名
}
```

本案例涉及到的文件操作在第3章中将有详细讲述。**FileDialog**类显示一个对话框窗口, 用户可以选择文件, 由于它是一个模式对话框, 当应用程序调用其**show()**方法来显示对话框时, 它将阻塞其余应用程序, 直到用户选择一个文件。

4. 获取Image文件对象:

```
tk=Toolkit.getDefaultToolkit(); //获取Toolkit对象
img = tk.getImage(path);        //利用Toolkit对象的getImage方法获取Image对象
mtrack = new MediaTracker(this); //创建媒体跟踪器对象
mtrack.addImage(img,0);         //向媒体跟踪器正在跟踪的图像列表添加获取的Image对象
runMe = new Thread(this);       //创建一个线程
runMe.start();                  //启动线程
//查看媒体跟踪器正在跟踪的所有图像是否均已完成加载
while(!mtrack.checkAll()){
    try{//未全部完成加载则线程休眠等待
        runMe.sleep(1000);
    }
    catch(Exception e){ }
}
//完成加载则线程终止
runMe.interrupt();
return ;
}
```

Toolkit使用单例模式创建所谓的**Toolkit**的默认对象, 并且确保这个默认实例在整个系统中是唯一的。**Toolkit**类提供了一个静态的方法**getDefaultToolkit()**来提供这个唯一的实例, 整个方法都是同步化的。**MediaTracker**类也将在第3章中讲述。

5. 抓取给定的图像像素并将Image图像转换成VRML文件:


```

        int width = img2convert.getWidth(this);    //获取图像宽度
        int height = img2convert.getHeight(this); //获取图像高度
        //创建一个 PixelGrabber对象，从指定的图像中抓取全部像素
        PixelGrabber pg = new PixelGrabber(img2convert, 0, 0, width, height, true);

```

创建一个PixelGrabber对象，从指定的图像中抓取全部像素，PixelGrabber类可以实现附加在Image或ImageProducer对象上获得图像像素子集的ImageConsumer。

```
int[] pixels = (int[]) pg.getPixels(); //获取像素缓冲区
```

生成VRML文件:

```
StringBuffer vrml = new StringBuffer();
```

VRML文件头:

```
vrml.append("#VRML V2.0 utf8\n");
```

VRML文件节点:

```
vrml.append("Transform{ scale "+(float) width/100+" "+(float) height/100 +" 1  children[\nShape
{geometry IndexedFaceSet{creaseAngle 3.14 coord Coordinate{ point[-.5 -.5 0,.5 -.5 0,.5 .5 0,-.5 .5 0, ] }
coordIndex[0 1 2 -1, 2 3 0 -1]  } \nappearance Appearance{material Material{diffuseColor 1 1 1}texture ");
```

VRML文件节点数据:

```
vrml.append("\nPixelTexture{image "+width+" "+height+" "+mode+"\n"+pixData+"}");
```

VRML文件尾:

```
vrml.append("\n}}});");
return vrml.toString();
```

6. 以.wrl为扩展名保存生成的VRML文件:

```

File outFile = new File(theDir,(theFile+".wrl"));
FileWriter fw = new FileWriter(outFile);
PrintWriter pw = new PrintWriter(fw);
pw.print(vstring);
pw.flush();
pw.close();

```

程序源代码与解释

```

/* * Image2VRML.java*/
import java.net.MalformedURLException;
/**将GIF或者JPG图像转换成VRML格式*/
public class Image2VRML extends Frame implements Runnable{
    Toolkit tk;
    MediaTracker mtrack;    //创建媒体跟踪器以跟踪图像
    Image img;
    //定义常量
    final int GRAY = 1;
    final int GRAY_WITH_ALPHA = 2;
    final int RGB = 3;
    final int RGB_WITH_ALPHA = 4;
    //定义文件、目录以及线程
    String theFile;

```



```

String theDir;
Thread runMe;
public static void main(String[] args) {    //主方法
    Image2VRML i2v = new Image2VRML();    //调用构造方法生成类的实例
}
public Image2VRML() {    //构造方法
    setLayout(null);        //设置窗口布局
    //调用askForFilePath方法来选择图像文件
    String thePath = askForFilePath();
    if (thePath == "") System.exit(0);    //未选择文件则系统退出
    //调用ImageFromFile方法来获取Image文件
    getImageFromFile(thePath);
    //设置窗口尺寸
    //参见步骤详解1
    //显示窗口
    setVisible(true);
    //调用img2pix方法来生成VRML文件
    String vrml = img2pix(img);
    //调用saveVrmlFile方法来保存VRML文件
    saveVrmlFile(vrml);
    System.exit(0);    //系统退出
}
public void getImageFromFile(String path) {    //获取Image文件
    //参见步骤详解4
}
public void paint(Graphics g) {
    g.drawImage(img,0,0,this);    //画出图像
}
public String img2pix(Image img2convert){    //Image图像转换成VRML文件
    //获取图像，参见步骤详解5
    try {
        boolean worked = pg.grabPixels();
        if(!worked) {
            System.out.println("无法抓取像素！");
        }
    }
    catch(InterruptedException e){
    }
    int[] pixels = (int[]) pg.getPixels();    //获取像素缓冲区
    //Java图形像素起始坐标从左上角开始，而VRML从左下角像素开始
    //因此必须进行转换
    int currentRow = height;
    int firstOfRow = 0;
    int index = 0;
    ColorModel cm = pg.getColorModel();    //获取所存储像素的ColorModel
    int nrOfPixels = width*height;    //像素总数
    String red[] = new String [nrOfPixels];
    String green[] = new String [nrOfPixels];
    String blue[] = new String [nrOfPixels];
    String alpha[] = new String [nrOfPixels];
    int r,g,b,a;
    boolean graytest = true;
    boolean alphatest = false;
    int count = 0;

```



```

for(int h=height;h>0;h--){
    currentRow -= 1; //当前像素行的索引
    firstOfRow = currentRow * width; //当前像素行第一个像素的索引
    for(int w=0;w<width;w++){ //从上一行像素处开始
        index = firstOfRow + w; //像素数组索引
        r = cm.getRed(pixels[index]); //获取R分量值
        g = cm.getGreen(pixels[index]); //获取G分量值
        b = cm.getBlue(pixels[index]); //获取B分量值
        a = cm.getAlpha(pixels[index]); //获取透明度分量值
        //如果一个以上像素具有不同的RGB值，就不可能是单分量（亮度纹理）
        if((r!= g) ||(g != b)||(r!= b)) graytest = false;
        if(a != 255) alphatest = true;
        alpha[count] = Integer.toHexString(a);
        red[count] = Integer.toHexString(r);
        green[count] = Integer.toHexString(g);
        blue[count] = Integer.toHexString(b);
        if(alpha[count].length() == 1) alpha[count] = "0"+alpha[count];
        if(red[count].length() == 1) red[count] = "0"+red[count];
        if(green[count].length() == 1) green[count] = "0"+green[count];
        if(blue[count].length() == 1) blue[count] = "0"+blue[count];
        count++;
    }
}
//按照不同的纹理贴图模式生成像素数据
StringBuffer pixData = new StringBuffer();
//默认值为全RGB纹理（三分量）
int mode = RGB;
//亮度纹理（单分量）
if(graytest && !alphatest) {
    mode = GRAY;
    for(int i=0;i<nrOfPixels;i++){
        pixData.append("0x"+red[i]+"\\n");
    }
}
//亮度加alpha透明度纹理（双分量）
if(graytest && alphatest) {
    mode = GRAY_WITH_ALPHA;
    for(int i=0;i<nrOfPixels;i++){
        pixData.append("0x"+red[i]+alpha[i]+"\\n");
    }
}
//全RGB纹理（三分量）
if(!graytest && !alphatest) {
    mode = RGB;
    for(int i=0;i<nrOfPixels;i++){
        pixData.append("0x"+red[i]+green[i]+blue[i]+"\\n");
    }
}
//全RGB加alpha透明度纹理（四分量）
if(!graytest && alphatest) {
    mode = RGB_WITH_ALPHA;
    for(int i=0;i<nrOfPixels;i++){
        pixData.append("0x"+red[i]+green[i]+blue[i]+alpha[i]+"\\n");
    }
}

```



```
    }  
    //生成VRML文件，参见步骤详解5  
}  
}
```

本章小结

线程（**thread**）是指进程中单一顺序的控制流，又称为轻量级进程。线程共享相同的地址空间并共同构成一个大的进程。线程使得在一个应用程序中，程序的编写更加自由和丰富，一个程序中可以同时使用多个线程来完成不同的任务。多线程可以增进程序的交互性，提供更好的功能、更好的**GUI**和更好的服务器功能。因此，如果很好地利用线程，可以大大简化应用程序设计。本章通过几个不同难易程度的实用案例程序，讲述了在**Java**语言中产生线程的两种方式：一是实现一个**Runnable**界面，二是扩充一个**Thread**类。另外，本章还详细介绍了线程的同步控制，有助于读者对线程概念的全面理解和应用。在实际应用中，线程使用的范围很广，可用于实时数据处理、快速的网络服务，还有更快的图像绘制和打印，以及数据库中的数据的取回和处理等，读者可以在实际应用中仔细体会。



第3章 Java与I/O

本章内容

- 案例1：使用多线程删除指定目录及子目录下所有指定文件
- 案例2：压缩文件
- 案例3：解压缩Zip文件
- 案例4：批量改名
- 案例5：文件分割器
- 案例6：管道流实现线程间的通信
- 案例7：排序对象
- 本章小结



案例1：使用多线程删除指定目录及子目录下所有指定文件

案例运行效果与操作

本案例使用多线程对目标路径中的指定文件以及子目录中的同名文件进行删除。程序运行后，界面如图3-1所示。

可以发现，由于没有指定要删除的文件，所以此时“删除”按钮不可用。单击“浏览”按钮，会弹出“打开”对话框，界面如图3-2所示。

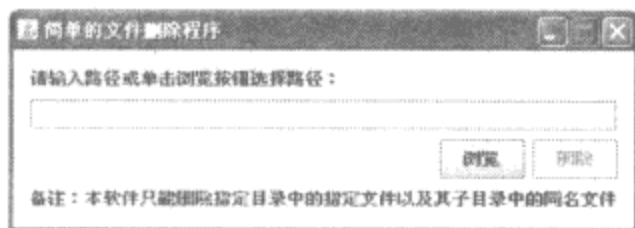


图3-1 运行时界面

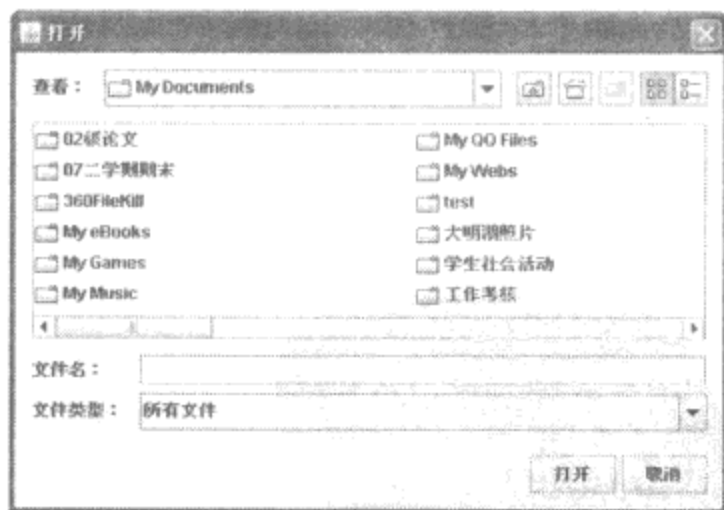


图3-2 “打开”对话框

选择想要删除的文件，单击“打开”按钮返回删除程序界面，如图3-3所示。此时“删除”按钮可用了。也可以在如图3-1所示的界面中，直接在文本框中输入想要删除的文件名，输入后“删除”按钮就变成可用状态。

单击“删除”按钮，会弹出删除文件是否成功以及删除数量的提示，界面如图3-4所示。此时文本框中文字清空，同时“删除”按钮恢复为不可用状态，此时可以进行下一次的删除操作，或者关闭窗口退出。

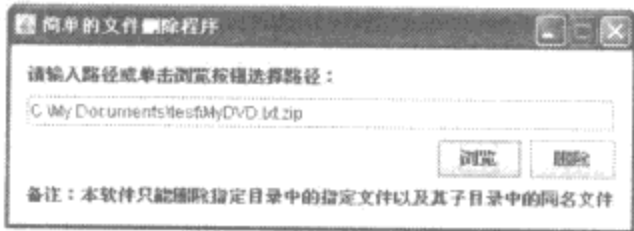


图3-3 选择文件后的程序窗口

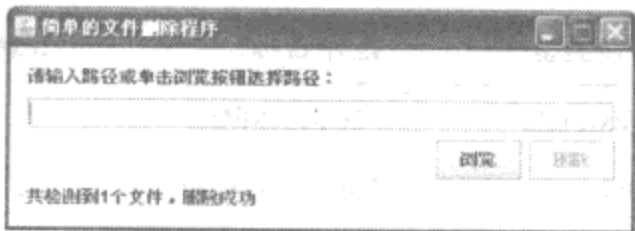


图3-4 删除文件后的程序窗口

制作要点

1. java.io.File类的用法。

File类是文件和目录路径名的抽象表示形式。Java中的File类可以代表文件，也可以代表目录。一般的I/O操作首先都要通过建立File类的对象来表示操作的目录或者文件。File的使用非常的简单，它有四个构造函数：

```
File(String parent,String child)
File(File parent,String child)
File(URI uri)
File(String pathname)
```

其中，前面两个函数可以在某个已知特定的目录下新建文件或者目录，后面两个函数可以通过pathn-ame或者URI新建文件或者目录。

2. javax.swing.JFileChooser类的用法。

当编写应用程序的时候，经常需要打开和保存文件。Swing提供了一个JFileChooser类，该类提供了一个GUI，用来浏览文件系统并提供一个列表选择文件或目录，或者输入文件和目录的名称来选择文件和目录的功能。JFileChooser本身类似于Dialog，是一个重型组件，重型组件是不能添加到JFrame、JDialog等重型组件上面的，只能在使用的时候实例化JFile-Chooser，然后显示。它包含若干常量字段，其中本案例中的APPROVE_OPTI/ON表示在选择确认后返回该值，因此在“打开”对话框中选择了某个文件或目录后再单击“打开”按钮就会返回该值。

```
fc=new JFileChooser();
int returnVal=fc.showOpenDialog(this);/
if(returnVal==JFileChooser.APPROVE_OPTI/ON)
```

3. Thread类的用法。

步骤详解

1. 界面设计，创建JFrame窗体并添加相应的JLabel、JTextField和JButton，其属性值如表3-1所示。

表3-1 属性与值对照表

控件类型	控件名称	text属性值
JLabel	JLabelPath	请输入路径或单击浏览按钮选择路径
JLabel	JLabelDescription	备注：本软件只能删除指定目录中的指定文件以及其子目录中的同名文件

控件类型	控件名称	text属性值
JTextField	JTextFieldPath	
JButton	JButtonBrowse	浏览
JButton	JButtonDelete	删除

调整控件实例在窗体上的位置和大小，界面效果如图3-5所示。



图3-5 最终界面设置效果

2. 添加成员变量，并按照如表3-2所示修改属性。

表3-2 属性与值对照表

变量类型	字段名称	访问属性值
FileDeleteThread	thread	<default>
JFileChooser	fc	private

3. 表示操作的目录或者文件可以通过建立File类的对象来实现。如同本例“制作要点”讲述的，File类的构造函数可以在某个已知特定的目录下新建文件或目录，或者通过pathname或URI新建文件或目录。本案例使用的是后者，即构造File类。

```
File file=new File(jTextFieldPath.getText());
```

4. 文件操作离不开文件的打开、保存等操作，进行GUI编程的时候经常需要用JFileChooser组件构造一个“文件”对话框来为用户提供打开文件、保存文件等操作，然后显示。JFileChooser组件包含若干常量字段，其中本案例中的APPROVE_OPTION表示在选择确认后返回该值，因此，在“打开”对话框中选择了某个文件或目录后再单击“打开”按钮就会返回该值。

通常的做法是调用JFileChooser.showXXX()方法显示“文件”对话框并且选择一个文件后，单击“Approve”按钮（默认情况下标有“Open”或“Save”），当对话框关闭后使用JFileChooser.getSelectedFile()方法得到选取的文件（或使用JFileChooser.getSelectedFiles()取得选取的文件数组），然后对被选取的文件有效性进行验证（例如，文件的文件名是否合法、选取的路径下是否已有同名文件存在等），如果验证不通过，需要再次打开文件选择对话框进行选取。

```
fc=new JFileChooser();
int returnVal=fc.showOpenDialog(this);/
if(returnVal==JFileChooser.APPROVE_OPTION);
```

5. 为JButtonBrowse按钮添加ActionEvent事件的处理方法，弹出一个打开文件选择器，如果用户选择了一个文件，则将文件的绝对路径取出，并将“删除”按钮设为可用：

```
int returnVal=fc.showOpenDialog(this);
```


//如果用户选择了一个文件，则将文件的绝对路径取出，并将“删除”按钮设为可用

```
if(returnVal==JFileChooser.APPROVE_OPTION){
    filename=fc.getSelectedFile().getAbsolutePath();
    jButtonDelete.setEnabled(true);
}
jTextFieldPath.setText(filename);    //文本框中显示可以删除的文件
```

6. 获取用户输入的路径并取得当前路径中的文件和文件夹，存到一个数组中：

```
File file=new File(jTextFieldPath.getText());
File[] files=(new File(file.getParent()).listFiles());
```

7. 删除文件。调用内部类FileDeleteThread，创建一个删除文件线程，然后启动此线程，进行文件删除。

```
thread=new FileDeleteThread(files,filename);
thread.start();
```

8. 为jTextFieldPath文本框添加MouseEvent事件的处理方法，当鼠标指针移出时，如果文本框为空，则使“删除”按钮不可用：

```
if(jTextFieldPath.getText().equals("")){
    jButtonDelete.setEnabled(false);
}else{
    jButtonDelete.setEnabled(true);
}
```

9. 为jTextFieldPath文本框添加KeyEvent事件的处理方法，当键盘按键被按下时，如果文本框为空，则“删除”按钮不可用：

```
if(jTextFieldPath.getText().equals("")){
    jButtonDelete.setEnabled(false);
}else{
    jButtonDelete.setEnabled(true);
}
```

程序源代码与解释

```
/* * FileDeleter.java*/
package filedeleter;
public class FileDeleter {
    /** 实例化FileDeleter */
    public FileDeleter() { }
    public static void main(String[] args) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new FrameFileDeleter().setVisible(true);
            }
        });
    }
}

/** FrameFileDeleter.java*/
package filedeleter;
import java.io.File;
import javax.swing.JFileChooser;
public class FrameFileDeleter extends javax.swing.JFrame {
```



```

int deletefilecount;        //删除文件计数器
private JFileChooser fc;     //文件选择器
FileDeleteThread thread;    //删除文件线程
/** Creates new form FrameFileDeleter */
public FrameFileDeleter() {
    initComponents();
}
private void initComponents() {
    //组件初始化，代码略
}
private void jButtonDeleteActionPerformed(java.awt.event.ActionEvent evt) {
    //构造File类，以获得相关的路径层次
    File file=new File(jTextFieldPath.getText());        //获取用户输入的路径
    //获取当前路径中的文件和文件夹，存到一个数组中
    File[] files=(new File(file.getParent()).listFiles());
    String filename=file.getName();        //获取文件名
    deletefilecount=0;        //初始化文件计数器
    //调用内部类FileDeleteThread，创建一个删除文件线程
    thread=new FileDeleteThread(files,filename);
    thread.start();    //启动删除文件线程，进行文件的删除
}
private void jButtonBrowseActionPerformed(java.awt.event.ActionEvent evt) {
    String filename="";
    fc=new JFileChooser();        //创建一个文件选择器
    //参见步骤详解5
}
private void jTextFieldPathKeyPressed(java.awt.event.KeyEvent evt) {
    //当键盘按键被按下时，如果文本框为空，则“删除”按钮不可用，参见步骤详解9
}
private void jTextFieldPathMouseExited(java.awt.event.MouseEvent evt) {
    //当鼠标指针移出时，如果文本框为空，则“删除”按钮不可用，参见步骤详解8
}
class FileDeleteThread extends Thread { //内部类，完成文件删除操作
    File files[];
    String filename;
    public FileDeleteThread(File files[],String filename){ //构造方法
        this.files=files;        //获取当前路径中的所有文件和文件夹
        this.filename=filename;    //获取文件名
    }
    public void run(){        //删除文件线程启动后的运行方法
        try{        //遍历指定目录下的所有文件夹，对符合条件的文件进行删除
            for(int i=0;i<files.length;i++){
                this.sleep(100);
                //如果是文件且符合删除条件，则进行删除并给出提示信息
                if((files[i].isFile() || files[i].isHidden()) && files[i].getName().equals
(filename)){

                    deletefilecount++;
                    jLabelDescription.setText("共检测到"+deletefilecount+"个文件，");
                    if(files[i].delete()){
                        jLabelDescription.setText(jLabelDescription.getText()+"删除成功");
                    }else{
                        jLabelDescription.setText(jLabelDescription.getText()+"删除失败");
                    }
                }
                //如果是文件夹，则以其为新的根目录启动新的线程（递归），完成文件夹的
                //遍历
            }else if(files[i].isDirectory()){

```



```
File file=new File(files[i].getAbsolutePath());
thread=new FileDeleteThread(file.listFiles(),filename);
thread.start();
}
}
}catch(Exception e){
}
}
}
```



案例2：压缩文件

案例运行效果与操作

本案例可以对目标路径中的指定文件进行压缩，既可以压缩成Windows系统下常见的以.zip为扩展名的Zip文件，也可以压缩成UNIX系统下常见的以.gz为扩展名的GZip文件。程序运行后，界面如图3-6所示。

可以发现，由于没有指定要压缩的文件，所以此时“Zip压缩”和“GZip压缩”按钮不可用。单击“浏览”按钮，会弹出“打开”对话框，界面如图3-7所示。



图3-6 运行时界面

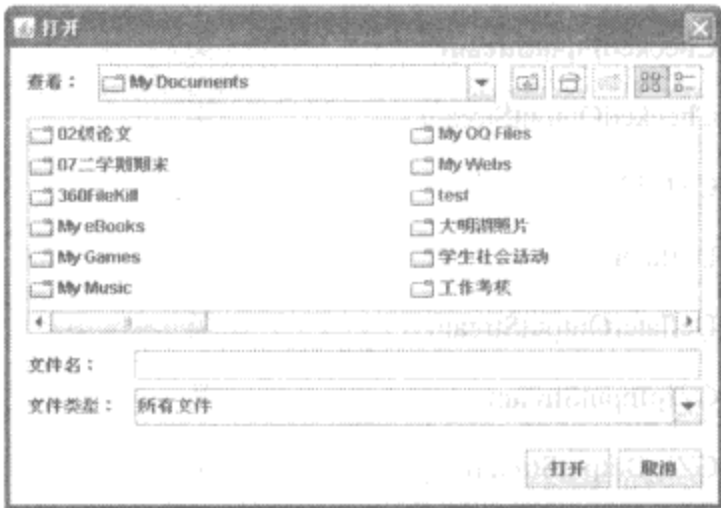


图3-7 “打开”对话框

选择想要压缩的文件，单击“打开”按钮返回压缩文件程序界面，如图3-8所示。可以发现，此时两个压缩按钮可用了。也可以在如图3-6所示的界面中，直接在文本框中输入想要压缩的文件名，输入后压缩按钮就变成可用状态。

如果进行Zip压缩，则单击“Zip压缩”按钮，否则就单击“GZip压缩”按钮，会弹出压缩文件是否成功的提示，界面如图3-9所示。此时压缩按钮恢复为不可用状态，此时可以进行下一次的压缩操作，或者关闭窗口退出。



图3-8 选择文件后的程序窗口



图3-9 压缩文件后的程序窗口

打开资源管理器，查看相应目录，发现确实多了一个压缩文件。

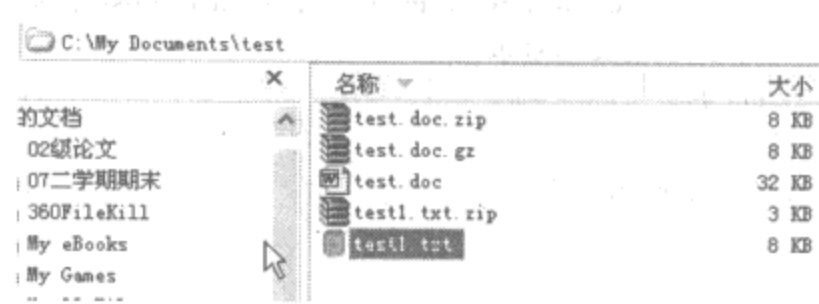


图3-10 资源管理器窗口

制作要点

1. java.util.zip包的用法。

Java为兼容Zip的数据压缩提供java.util.zip包。java.util.zip包提供了数据压缩与解压缩所需要的类。此外，还提供了用于计算任意输入流校验和的应用类，该类可用于确认输入数据。这个包有1个界面、14个类和2个异常类，如表3-3所示。

表3-3 java.util.zip包

条目	类型	描述
Checksum	接口	被类Adler32和CRC32实现的接口
Adler32	类	使用Alder32算法来计算Checksum数目
CheckedInputStream	类	一个输入流，保存着被读取数据的Checksum
CheckedOutputStream	类	一个输出流，保存着被读取数据的Checksum
CRC32	类	使用CRC32算法来计算Checksum数目
Deflater	类	使用ZLIB压缩类，支持通常的压缩方式
DeflaterOutputStream	类	一个输出过滤流，用来压缩Deflater格式数据
GZipInputStream	类	一个输入过滤流，读取GZip格式压缩数据
GZipOutputStream	类	一个输出过滤流，读取GZip格式压缩数据
Inflater	类	使用ZLIB压缩类，支持通常的解压方式
InlfaterInputStream	类	一个输入过滤流，用来解压Inlfater格式的压缩数据
ZipEntry	类	存储Zip条目
ZipFile	类	从Zip文件中读取Zip条目
ZipInputStream	类	一个输入过滤流，用来读取Zip格式文件中的文件
ZipOutputStream	类	一个输出过滤流，用来向Zip格式文件口写入文件
DataFormatException	异常类	抛出一个数据格式错误
ZipException	异常类	抛出一个Zip文件

本案例使用了其中的ZipOutputStream、GZipOutputStream和ZipEntry等类，读者可以从中了解它们的用法，案例3则用到ZipFile类。

2. javax.swing.JFileChooser类的用法。

3. 缓冲输入/输出流（BufferedInputStream/BufferedOutputStream）的用法。

步骤详解

1. 界面设计。新建JFrame窗体，并添加2个JLabel、1个JTextField和3个JButton，按照如表3-4所示修改属性。

表3-4 属性与值对照表

控件类型	控件名称	text属性值
JLabel	JLabelPath	请输入要压缩的文件路径或单击浏览按钮选择文件
JLabel	JLabelDescription	备注：可以将文件压缩成zip文件或gzip文件
JTextField	JTextFieldPath	
JButton	JButtonBrowse	浏览
JButton	JButtonZip	Zip压缩
JButton	JButtonGZip	GZip压缩

然后调整各个控件实例在窗体上的位置与大小，界面最终设置效果如图3-11所示。

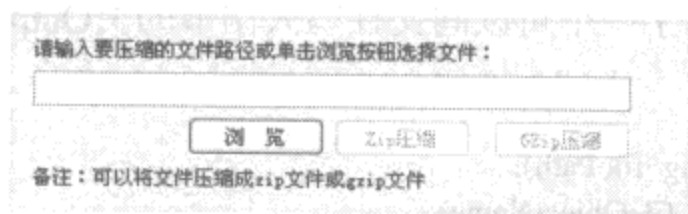


图3-11 界面最终设置效果

2. 添加成员变量，并按照如表3-5所示修改属性。

表3-5 属性与值对照表

变量类型	字段名称	访问属性值	修饰符
String	Filename	<default>	static
ZIPOutputStream	zos	<default>	static
GZIPOutputStream	gzos	<default>	static
JFileChooser	fc	private	

3. 程序设计。添加方法，并按照如表3-6所示修改属性。其中，makeZip方法通过调用recurseFiles方法完成Zip压缩，而makeGZip方法则完成对文件进行GZip压缩的功能。

表3-6 属性与值对照表

方法名称	返回类型	访问属性	修饰符	参数设置	异常设置
recurseFiles	File	private	static	File	IOExceptionF, ileNotFoundException
makeZip	void	public	static	String fileName	IOException, FileNotFoundException
makeGZip	void	public	static	String fileName	IOException, FileNotFoundException

4. 创建文件输入/输出流，更改压缩后的文件名：

```

FileInputStream fin = new FileInputStream(file); //创建文件输入流
BufferedInputStream in = new BufferedInputStream(fin); //创建缓冲输入流
File zipFile=new File(file + ".zip");
FileOutputStream fos = new FileOutputStream(zipFile);
ZipOutputStream zos = new ZipOutputStream(fos); //创建ZipOutputStream对象

```

在《Java案例开发集锦》第一版中，详细讲过文件输入/输出流（FileInputStream/FileOutputStream）的用法，而缓冲输入/输出（BufferedInputStream/BufferedOutputStream）是一个非常普通的性能优化。Java的BufferedInputStream类允许把任何InputStream类“包装”成缓冲流，缓冲流把内在缓冲器连接到输入/输出流，允许Java程序对多个字节同时操作，这样提高了效率，通过BufferedInputStream类与BufferedOutputStream类可实现缓冲流。文件输入操作时将FileInputStream类与BufferedInputStream类结合可大大提高性能。下面简单介绍一下它们的用法：

BufferedInputStream类的构造函数有两种形式：

```

BufferedInputStream(InputStream inputStreamName);
BufferedInputStream(InputStream inputStreamName,int bufferSize);

```

FileOutputStream类创建了一个可以向文件写入字节的类OutputStream。它的构造函数有以下三种形式：

```

FileOutputStream(String filePath);
FileOutputStream(File fileObjectName);
FileOutputStream(String filePath,boolean append);

```

5. 用户选择的文件为目录时进行递归处理：

```

if(file.isDirectory()) { //如果当前文件对象是目录
    String[] fileNames = file.list();
    if(fileNames != null)
        for(int i=0; i<fileNames.length; i++)
            recurseFiles(new File(file, fileNames[i])); //递归调用
    }else { ... }

```

6. GZip压缩处理方法：

```

try { //向GZip文件写入数据
    while((len = in.read(buf)) >= 0) {
        gzos.write(buf, 0, len); //逐行写入
    }
    jLabelDescription.setText(file.toString()+"压缩成功");
} catch (Exception e) {
    jLabelDescription.setText(file.toString()+"未压缩成功");
}finally{ ... }

```

7. 为jTextFieldPath文本框添加MouseEvent事件的处理方法，当鼠标指针移出时，如果文本框为空，则使“压缩”按钮不可用：

```

if(jTextFieldPath.getText().equals("")){
    jButtonZip.setEnabled(false);
    jButtonGZip.setEnabled(false);
}else{
    jButtonZip.setEnabled(true);
}

```



```
jButtonGZip.setEnabled(true);
}
```

8. 为jTextFieldPath文本框添加KeyEvent事件的处理方法，当键盘按键被按下时，如果文本框为空，则使“压缩”按钮不可用：

```
if(jTextFieldPath.getText().equals("")){
    jButtonZip.setEnabled(false);
    jButtonGZip.setEnabled(false);
}else{
    jButtonZip.setEnabled(true);
    jButtonGZip.setEnabled(true);
}
```

9. 总结一下输入/出流（InputStream/OutputStream）的用法。

Java中主要有4个输入/输出的抽象类：InputStream、OutputStream、Reader以及Writer。

InputStream和OutputStream为字节流设计，Reader和Writer为字符流设计。

字节输入流InputStream类的常用方法如下。

- (1) available()得到当前可读的输入字节数。
- (2) close()关闭输入源。
- (3) read()读取数据。

字节输出流OutputStream类常用方法如下。

- (1) close()关闭输出流。
- (2) flush()刷新输出缓冲区。
- (3) write()写入数据。

程序源代码与解释

```
/** CompressFile.java*/
package compressfile;
public class CompressFile {
    /** 实例化 CompressFile */
    public CompressFile() {}
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new FrameCompressFile().setVisible(true);
            }
        });
    }
}

/** FrameCompressFile.java*/
package compressfile;
import javax.swing.JFileChooser;
public class FrameCompressFile extends javax.swing.JFrame {
    static String filename;
    private JFileChooser fc;    //文件选择器
    static GZIPOutputStream gzos;
    static ZipOutputStream zos;
    /** Creates new form FrameCompressFile */
    public FrameCompressFile() {
```



```

        initComponents();
    }
    private void initComponents() {
        //组件初始化，代码略
    }
    private void jTextFieldPathMouseExited(java.awt.event.MouseEvent evt) {
        //当鼠标指针移出时，如果文本框为空，则使“压缩”按钮不可用，参见步骤详解7
    }
    private void jTextFieldPathKeyPressed(java.awt.event.KeyEvent evt) {
        //当键盘按键按下时，如果文本框为空，则使“压缩”按钮不可用，参见步骤详解8
    }
    private void jButtonGZipActionPerformed(java.awt.event.ActionEvent evt) { //构造File类，
//以获得相关的路径层次
        File file=new File(jTextFieldPath.getText()); //获取用户输入的路径
        String filename=file.getAbsolutePath(); //获取文件名
        try {
            makeGZip(filename); //进行GZip压缩
        } catch (Exception e) {
            System.err.println(e);
        }
        //压缩完毕将“压缩”按钮设为不可用
        jButtonZip.setEnabled(false);
        jButtonGZip.setEnabled(false);
    }
    private void jButtonZipActionPerformed(java.awt.event.ActionEvent evt) {
        //构造File类，以获得相关的路径层次
        File file=new File(jTextFieldPath.getText()); //获取用户输入的路径
        String filename=file.getAbsolutePath(); //获取文件名
        try {
            makeZip(filename); //进行Zip压缩
        } catch (Exception e) {
            System.err.println(e);
        }
        //压缩完毕将“压缩”按钮设为不可用
        jButtonZip.setEnabled(false);
        jButtonGZip.setEnabled(false);
    }
    private void jButtonBrowseActionPerformed(java.awt.event.ActionEvent evt) {
        String filename="";
        fc=new JFileChooser(); //创建一个文件选择器
        int returnVal=fc.showOpenDialog(this); //弹出一个打开文件选择器，
        //如果用户选择了一个文件，则将文件的绝对路径取出，并将“压缩”按钮设为可用
        if(returnVal==JFileChooser.APPROVE_OPTION){
            filename=fc.getSelectedFile().getAbsolutePath();
            jButtonZip.setEnabled(true);
            jButtonGZip.setEnabled(true);
        }
        jTextFieldPath.setText(filename); //文本框中显示可以压缩的文件
    }
    public static void makeZip(String fileName) throws IOException, FileNotFoundException {
        File file = new File(fileName); //创建文件对象
        recurseFiles(file); //调用递归方法进行压缩
    }

```



```

private static File recurseFiles(File file) throws IOException, FileNotFoundException {
    FileInputStream fin = new FileInputStream(file);    //创建文件输入流
    BufferedInputStream in = new BufferedInputStream(fin);    //创建缓冲输入流
    File zipFile=new File(file + ".zip"); //压缩后的文件名为源文件名加上.zip
    FileOutputStream fos = new FileOutputStream(zipFile);
    ZipOutputStream zos = new ZipOutputStream(fos); //创建ZipOutputStream对象
    if(file.isDirectory()) { //如果当前文件对象是目录
        String[] fileNames = file.list();
        if(fileNames != null)
            for(int i=0; i<fileNames.length; i++)
                recurseFiles(new File(file, fileNames[i]));    //递归调用
    }else { //如果当前文件不是目录
        byte[] buf = new byte[1024];
        int len;    //存储一行的长度
        ZipEntry zipEntry = new ZipEntry(file.getName());    //为被读取的文件创建
        //在向Zip输出流写入数据之前，必须首先使用putNextEntry(entry)方法安置压缩条目
        zos.putNextEntry(zipEntry);
        try { //向Zip文件写入数据
            while((len = in.read(buf)) >= 0) {
                zos.write(buf, 0, len);//逐行写入
            }
            jLabelDescription.setText(file.toString()+"压缩成功");
        } catch (Exception e) {
            jLabelDescription.setText(file.toString()+"未压缩成功");
        }finally{ //关闭流对象
            in.close();
            zos.close();
        }
        return zipFile;
    }
}

```

压缩条目
对象



案例3：解压缩Zip文件

案例运行效果与操作

本案例可以对目标路径中的指定Zip文件进行解压缩。程序运行后，界面如图3-12所示。

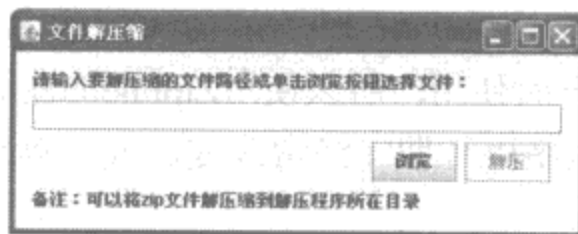


图3-12 运行时界面

可以发现，由于没有指定要解压缩的文件，所以此时“解压”按钮不可用。单击“浏览”按钮，会弹出“打开”对话框，界面如图3-13所示。

此时可以发现，在“文件类型”下拉列表框中，只有“Zip文件”一个选项，表明该程序只支持Zip文件的解压缩。选择想要压缩的文件，单击“打开”按钮返回解压程序界面，如

图3-14所示。此时“解压”按钮可用了。也可以在如图3-12所示的界面中，直接在文本框中输入想要解压缩的文件名，输入后“解压”按钮就变成可用状态了。

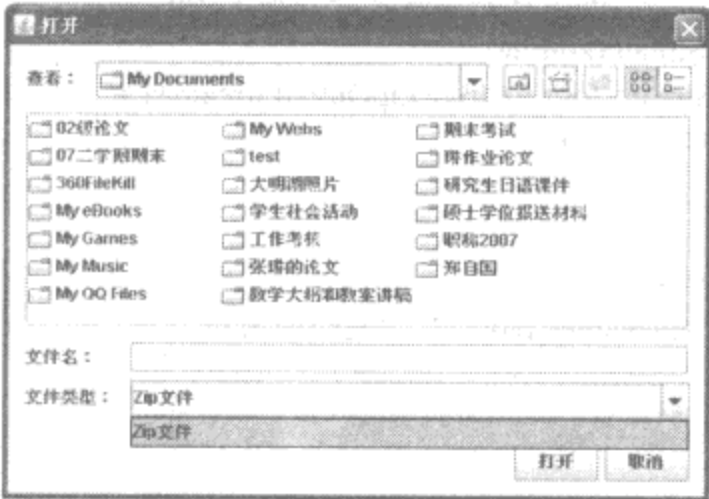


图3-13 “打开”对话框



图3-14 选择文件后的程序窗口

单击“解压”按钮，会给出解压文件是否成功的提示，界面如图3-15所示。此时“解压”按钮恢复为不可用状态，可以进行下一次的解压操作，或者关闭窗口退出。

打开资源管理器，查看解压程序所在目录，发现确实多了一个test.doc文件，如图3-16所示。



图3-15 解压文件后的程序窗口

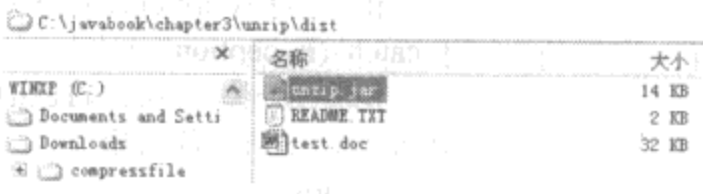


图3-16 资源管理器窗口

制作要点

1. ZipFile类的用法。

Zip文件的解压缩实质上就是从输入流中读取数据。Java.util.zip包提供了ZipInputStream类和ZipFile类来读取Zip文件。ZipInputStream类读出Zip文件序列，而ZipFile类使用内嵌的随机文件访问机制读出其中的文件内容，所以不必顺序读出Zip压缩文件序列。

ZipInputStream和ZipFile另外一个基本的不同点是高速缓冲的使用。当文件使用ZipInputStream和FileInputStream流读出的时候，Zip条目不使用高速缓冲。如果使用ZipFile（文件名）来打开文件，它将使用内嵌的高速缓冲，如果ZipFile（文件名）被重复调用的话，文件只被打开一次。缓冲值在第二次打开时使用，所以使用ZipFile的性能优于Zip-InputStream。然而，如果同一个Zip文件的内容在程序执行期间经常改变，或是重载的话，使用ZipInputStream就成为首选了。

2. FileNameExtensionFilter类的用法。

FileNameExtensionFilter类是FileFilter类的一个实现，它使用指定的扩展名集合进行过滤。文件的扩展名是指文件名最后一个“.”后面的部分。名称不包含“.”的文件没有文件扩展名。文件扩展名不区分大小写。

3. File类的用法。

步骤详解

1. 界面设计。新建JFrame窗体，添加2个JLabel、1个JTextField和2个JButton，并按照如表3-7所示修改属性。

表3-7 属性与值对照表

控件类型	控件名称	text属性值
JLabel	JLabelPath	请输入要压缩的文件路径或单击浏览按钮选择文件
JLabel	JLabelDescription	备注：可以将Zip文件解压缩到解压程序所在目录
JTextField	JTextFieldPath	
JButton	JButtonBrowse	浏览
JButton	JButtonUnZip	解压

再调整各个控件实例在窗体上的位置与大小，界面最终设置效果如图3-17所示。

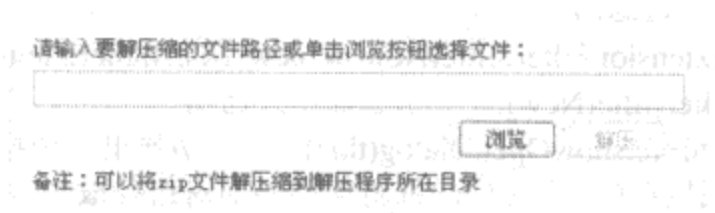


图3-17 界面最终设置效果

2. 解压一个Zip文件的过程如下。

(1) 通过指定一个被读取的Zip文件，或者是文件名，或者是一个文件对象来创建一个ZipFile对象：

```
ZipFile zipfile = new ZipFile(filename);
```

(2) 使用entry方法，返回一个枚举对象，循环获得文件的Zip条目对象：

```
while(e.hasMoreElements()) {
    entry = (ZipEntry) e.nextElement();
}
```

(3) Zip条目作为参数传递给getInputStream方法，可以读取Zip文件中指定条目的内容，其返回的输入流（InputStram）对象可以方便地读出Zip条目的内容：

```
in = new BufferedInputStream(zipfile.getInputStream(entry));
```

(4) 获取Zip条目的文件名，创建输出流，并保存：

```
byte data[] = new byte[BUFFER];
FileOutputStream fos = new FileOutputStream(entry.getName());
dest = new BufferedOutputStream(fos, BUFFER);
while ((count = is.read(data, 0, BUFFER)) != -1) {
    dest.write(data, 0, count);
}
```

(5) 关闭所有的输入输出流：


```
dest.flush();
dest.close();
is.close();
```

3. 使用指定的扩展名集合进行过滤，本案例使用`FileNameExtensionFilter`类来实现在“打开”对话框中只显示Zip文件的功能。

```
//创建一个新过滤器
FileNameExtensionFilter filterNew = new FileNameExtensionFilter("Zip文件", "zip");
fc.setFileFilter(filterNew); //添加新过滤器
```

4. 为`JButtonBrowse`按钮添加`ActionEvent`事件的处理方法：

```
String filename="";
fc=new JFileChooser(); //创建一个文件选择器
FileFilter[] filterOld=fc.getChoosableFileFilters(); //获取原来已有的过滤器
int filterSize=filterOld.length;
for (int i=0;i<filterSize;i++){ //删除原来已有的过滤器
    fc.removeChoosableFileFilter(filterOld[i]);
}
//创建一个新过滤器
FileNameExtensionFilter filterNew = new FileNameExtensionFilter("Zip文件", "zip");
fc.setFileFilter(filterNew); //添加新过滤器
int returnVal=fc.showOpenDialog(this); //弹出一个打开文件选择器，
//如果用户选择了一个文件，则将文件的绝对路径取出，并将“解压”按钮设为可用
if(returnVal==JFileChooser.APPROVE_OPTION){
    filename=fc.getSelectedFile().getAbsolutePath();
    jButtonUnZip.setEnabled(true);
}
jTextFieldPath.setText(filename); //文本框中显示可以解压缩的文件
```

5. 构造`File`类，以获得相关的路径层次：

```
File file=new File(jTextFieldPath.getText()); //获取用户输入的路径
String filename=file.getAbsolutePath(); //获取文件名
```

6. 用同样方法为`jTextFieldPath`文本框添加`MouseEvent`事件的处理方法，当鼠标指针移出时，如果文本框为空，则使“解压”按钮不可用：

```
if(jTextFieldPath.getText().equals("")){
    jButtonUnZip.setEnabled(false);
}else{
    jButtonUnZip.setEnabled(true);
}
```

程序源代码与解释

```
/** FrameUnZip.java */
public class FrameUnZip extends javax.swing.JFrame {
    private JFileChooser fc; //文件选择器
    static final int BUFFER = 2048;
    /** Creates new form FrameUnZip */
    public FrameUnZip() {
        initComponents();
    }
}
```



```

private void initComponents() {
    //组件初始化, 代码略
}
private void jButtonUnZipActionPerformed(java.awt.event.ActionEvent evt) {
    //构造File类, 以获得相关的路径层次
    File file=new File(jTextFieldPath.getText()); //获取用户输入的路径
    String filename=file.getAbsolutePath();      //获取文件名
    try {
        UnZip(filename); //进行解压缩
    } catch (Exception e) {
        System.err.println(e);
    }
    //解压缩完毕将“解压”按钮设为不可用
    jButtonUnZip.setEnabled(false);
}
private void jTextFieldPathMouseExited(java.awt.event.MouseEvent evt) {
    //当鼠标指针移出时, 如果文本框为空, 则让“解压”按钮不可用, 参见步骤详解6
}
private void jTextFieldPathKeyPressed(java.awt.event.KeyEvent evt) {
    //当键盘按键按下时, 如果文本框为空, 则让“解压”按钮不可用
    if(jTextFieldPath.getText().equals("")){
        jButtonUnZip.setEnabled(false);
    }else{
        jButtonUnZip.setEnabled(true);
    }
}
private void UnZip(String filename) {
    try {
        BufferedOutputStream dest = null;
        BufferedInputStream in = null;
        ZipEntry entry;
        ZipFile zipfile = new ZipFile(filename);
        Enumeration e = zipfile.entries();
        while(e.hasMoreElements()) {
            entry = (ZipEntry) e.nextElement();
            System.out.println("Extracting: " +entry);
            in = new BufferedInputStream(zipfile.getInputStream(entry));
            int count;
            byte data[] = new byte[BUFFER];
            FileOutputStream fos = new FileOutputStream(entry.getName());
            dest = new BufferedOutputStream(fos, BUFFER);
            while ((count = in.read(data, 0, BUFFER)) != -1) {
                dest.write(data, 0, count);
            }
            dest.flush();
            dest.close();
            in.close();
            jLabelDescription.setText(filename+"解压缩成功");
        }
    } catch (Exception e) {
        e.printStackTrace();
        jLabelDescription.setText(filename+"解压缩失败");
    }
}
}

```




案例4：批量改名

案例运行效果与操作

本案例可以对目标路径中多个文件按照指定的命名规则进行批量改名，还能利用搜索功能对文件进行筛选，以及对不需要的文件进行删除。程序运行后，界面如图3-18所示。

在上方的文本框中输入文件路径，或者单击“浏览”按钮，在弹出的“打开”对话框中选择文件路径，所选择路径下的文件就会显示在左下方的文件列表中，界面如图3-19所示。可以发现，每个文件前都有一个复选框供用户选择，也可以单击下方的“全选”复选框进行全部文件的选择。用户此时还可以单击“删除”按钮对选中的文件进行删除，删除前会弹出确认对话框，要求用户确认删除。

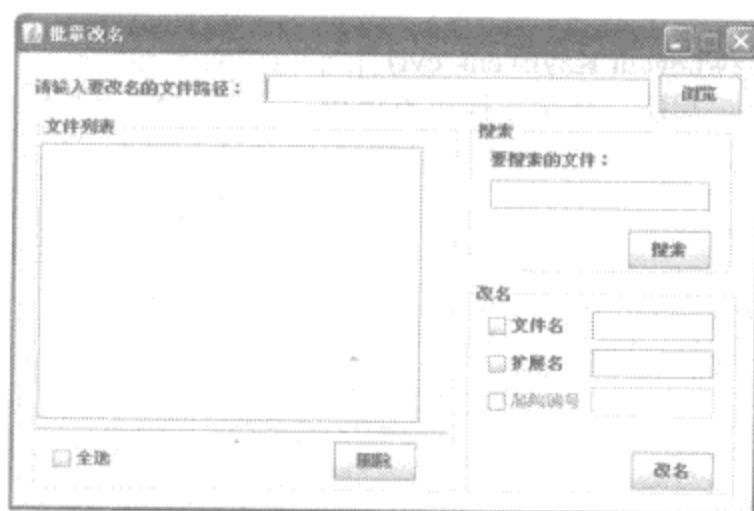


图3-18 运行时界面

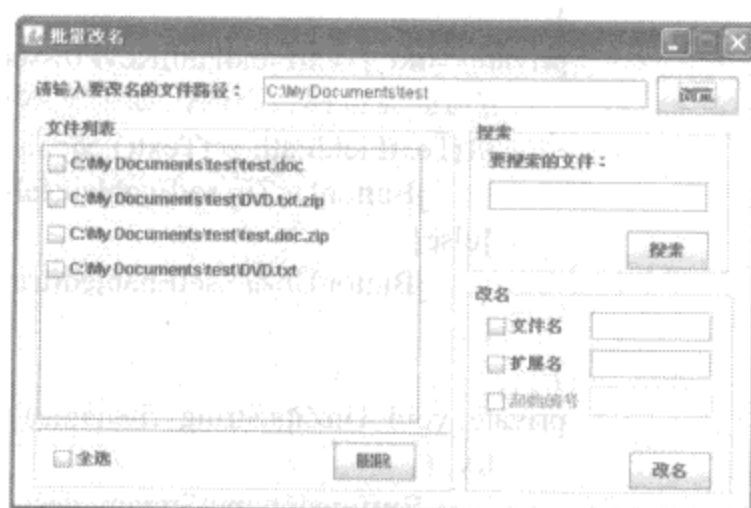


图3-19 选择路径后文件列表中的显示效果

如果文件列表中显示的文件数量非常多，还可以通过搜索功能进行筛选。在“要搜索的文件”文本框中输入筛选条件，单击“搜索”按钮即可。如图3-20所示为选择搜索Zip文件后程序返回的界面。

程序的批量改名操作，可以修改多个文件的文件名和扩展名。在选中“文件名”复选框之前，“起始编号”复选框和文本框是不可用的。选中“文件名”和“起始编号”复选框之后，对文件的改名操作是以统一的文件名加上从起始编号依次递增的编号作为新的文件名来进行的。如图3-21所示是对图3-20所选文件修改扩展名并单击“改名”按钮后的界面。

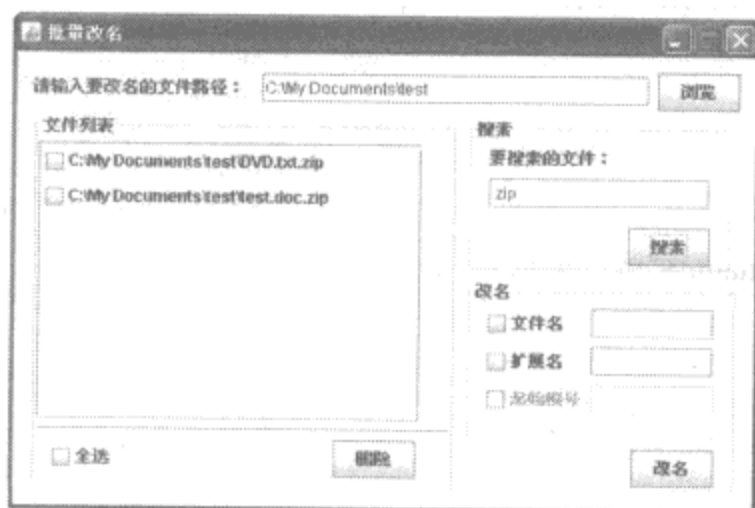


图3-20 搜索Zip文件后程序窗口中的内容

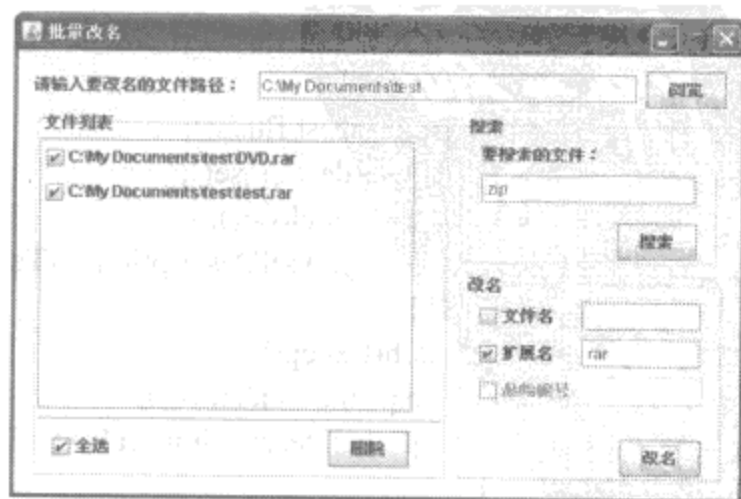


图3-21 修改扩展名后的界面

重新选择一个目录（以c:\My Documents\test为例），单击“全选”按钮选中全部文件，然后选中“文件名”和“起始编号”复选框，具体设置如图3-22所示。

单击“改名”按钮，界面如图3-23所示。程序已经完成批量改名操作。



图3-22 批量改名前的界面



图3-23 批量改名后的界面

制作要点

1. java.io.File类的用法。

File类是文件和目录路径名的抽象表示形式。**Java**中的**File**类可以代表文件也可以代表目录。一般的I/O操作首先都要通过建立**File**类的对象来表示操作的目录或者文件。本案例使用了**File**类的以下几个方法：

exists()：测试此抽象路径名表示的文件或目录是否存在。

getName()：返回由此抽象路径名表示的文件或目录的名称。

getAbsolutePath()：返回此抽象路径名的绝对路径名字符串。

isDirectory()：测试此抽象路径名表示的文件是否是一个目录。

isFile()：测试此抽象路径名表示的文件是否是一个标准文件。

length()：返回由此抽象路径名表示的文件的长度。

listFiles()：返回一个抽象路径名数组，这些路径名表示此抽象路径名所示的目录中的文件。

2. javax.swing.JFileChooser类的用法。

编写应用程序的时候，经常需要打开和保存文件。**Swing**提供了一个**JFileChooser**类，该类提供了一个**GUI**，用来浏览文件系统并提供通过一个列表选择文件或目录，或者输入文件和目录的名称来选择文件和目录的功能。

3. 内部类的用法。

步骤详解

1. 界面设计。创建**JFrame**窗体，在窗体上方添加1个**JLabel**、1个**JTextField**和1个**JButton**，并按照表3-8所示修改属性。

在窗体下方添加3个**JPanel**，按照表3-9所示修改属性。

向**jPanelFile**添加1个**JPanel**、1个**JScrollPane**和1个**JSeparator**，并按照表3-10所示修改属性。

表3-8 属性与值对照表

控件类型	控件名称	text属性值
JLabel	jLabelPath	请输入要改名的文件路径:
TextField	jTextFieldPath	
JButton	jButtonBrowse	浏览

表3-9 属性与值对照表

控件类型	控件名称	border属性值	TitledBorder “标题” 属性值
JPanel	jPanelFile	TitledBorder	文件列表
JPanel	jPanelSearch	TitledBorder	搜索
JPanel	jPanelRename	TitledBorder	改名

表3-10 属性与值对照表

控件类型	控件名称	属性设置
JScrollPane	jScrollPaneFileShow	“可水平调整大小”和“可垂直调整大小”属性设置为未选中状态
JSeparator	jSeparatorFile	
JPanel	jPanelFileOperation	“可水平调整大小”和“可垂直调整大小”属性设置为未选中状态

向jPanelFileOperation添加1个JCheckBox和1个JButton，并按照表3-11所示修改属性。

表3-11 属性与值对照表

控件类型	控件名称	text属性值
JCheckBox	jCheckBoxSelectAll	全选
JButton	jButtonDelete	删除

向jPanelSearch添加1个JLabel、1个JTextField和1个JButton，并按照表3-12所示修改属性。

表3-12 属性与值对照表

控件类型	控件名称	text属性值
JLabel	jLabelSearch	要搜索的文件
TextField	jTextFieldSearch	
JButton	jButtonSearch	搜索

向jPanelRename添加3个JCheckBox、3个JTextField和1个JButton，并按照表3-13所示修改属性。

调整各个控件实例在窗体上的位置与大小，界面最终设置效果如图3-24所示。

表3-13 属性与值对照表

控件类型	控件名称	text属性值
JCheckBox	jCheckBoxFileName	文件名
JCheckBox	jCheckBoxExtension	扩展名
JCheckBox	jCheckBoxNumber	起始编号
JTextField	jTextFieldFileName	
JTextField	jTextFieldExtension	
JTextField	jTextFieldNumber	
JButton	jButtonRename	改名

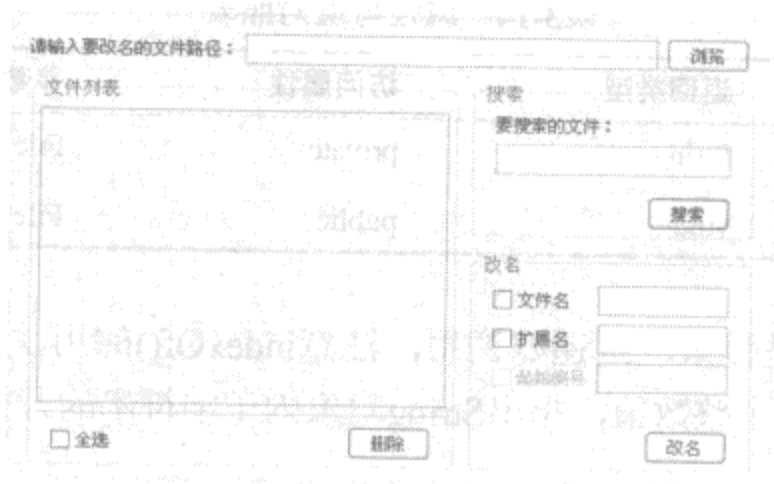


图3-24 界面最终设置效果

2. 用Vector()数组存放JCheckBox组件:

```
Vector checkboxes =new Vector();
```

Vector类在第1章的案例中涉及到了它的具体用法，Vector有三个构造函数：

```
public Vector(int initialCapacity,int capacityIncrement)
public Vector(int initialCapacity)
public Vector()
```

Vector运行时创建一个初始的存储容量initialCapacity，存储容量是以capacityIncrement变量定义的增量增长。初始的存储容量和capacityIncrement可以在Vector的构造函数中定义。第二个构造函数只创建初始存储容量。第三个构造函数既不指定初始的存储容量，也不指定capacityIncrement。

Vector类提供的访问方法支持类似数组运算和与Vector大小相关的运算。类似数组的运算允许向量中增加、删除和插入元素，也允许测试矢量的内容和检索指定的元素，与大小相关的运算允许判定字节大小和矢量中元素的数目。

现针对经常用到的对向量进行的增、删、插操作举例描述。

- (1) `addElement(Object obj)`: 把组件加到向量尾部，同时大小加1，向量容量比以前大1。
- (2) `insertElementAt(Object obj, int index)`: 把组件加到所定索引处，此后的内容向后移动1个单位。
- (3) `setElementAt(Object obj, int index)`: 把组件加到所定索引处，此处的内容被代替。

- (4) `removeElement(Object obj)`：把向量中含有本组件的内容移走。
- (5) `removeAllElements()`：把向量中所有组件移走，向量大小为0。

Vector类的缺点：

Vector类的方法都是被同步的，这会导致当只有一个线程访问矢量时会毫无必要地降低运行速度。能用数组的地方尽量用数组（开销小）。

实在需要使用动态数据的地方，有两种解决问题的方法：

- (1) 另开一个线程，让该线程去访问矢量数据（建议不要这么做，因为同样会遇到同步问题）。
- (2) 使用ArrayList类代替Vector类。ArrayList开销资源要小，功能也要好一些。定义方法的属性与值如表3-14所示。

表3-14 属性与值对照表

方法名称	返回类型	访问属性	参数设置
renameFile	String	private	File tardir, int id
searchFile	void	public	File tardir

3 搜索文件，如果是目录，则递归调用，注意indexOf()的用法。

indexOf()方法返回一个整数值，指出String对象内子字符串的开始位置。如果没有找到子字符串，则返回 - 1。

```
if(list[x].isFile()){           //是文件
    name=jTextFieldSearch.getText().trim();
    strname=list[x].getName().toString().trim();
    if(strname.indexOf(name)!=-1 || strname.endsWith(name) ){           //符合搜索条件
    ...
    }
} else { //是目录
    searchFile(list[x]); //递归调用
    ...
}
```

该方法从左到右进行查找，如果想反过来搜索，可以使用lastIndexOf()方法。

4. 给文件改名：

```
temp=tardir.getName();
str1=temp.substring(0,temp.indexOf("."));           //得到"."前的文件名
str2=temp.substring(temp.indexOf("."));           //得到扩展名
rename=jTextFieldFileName.getText().toString();
reext=jTextFieldExtension.getText().toString();
if(jCheckBoxFileName.isSelected()) {           //改名字
    temp=tardir.getParent()+"\\"+rename+str2;
}
if(jCheckBoxNumber.isSelected()) {           //名字加编号
    temp=tardir.getParent()+"\\"+rename+String.valueOf(id)+str2;
}
if(jCheckBoxExtension.isSelected()){           //改扩展名
    temp=tardir.getParent()+"\\"+str1+reext;
}
```


5. 文件、目录的选择。

本案例使用了JFileChooser类的setFileSelectionMode方法，用来设置允许用户只选择文件、只选择目录，或者选择文件和目录。它包含若干常量字段，本案例中的FILES_AND_DIRECTORIES表示可以选择文件和目录（默认只能选择文件）。

```
JFileChooser fc=new JFileChooser();           //创建一个文件选择器
fc.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
int returnVal=fc.showOpenDialog(this);       //弹出一个打开文件选择器，
//如果用户选择了一个文件，则将文件的绝对路径取出，并将“压缩”按钮设为可用
if(returnVal==JFileChooser.APPROVE_OPTION){
    filename=fc.getSelectedFile().getAbsolutePath();
}
jTextFieldPath.setText(filename);           //在文本框中显示
repaintPanel();                           //调用repaintPanel方法刷新显示文件列表
```

6. 为jButtonDelete按钮添加ActionEvent事件的处理方法，将下列代码添加到jButtonDelete-ActionPerformed方法的方法体内：

```
for(int i=0;i<jPanelFileShow.getComponentCount();++i) {
    //将jPanelFileShow中组件的类型转换成JCheckBox
    JCheckBox tempjcb= (JCheckBox)jPanelFileShow.getComponent(i);
    if (tempjcb.isSelected()) {           //如果被选中
        File tardir=new File(tempjcb.getText().trim()); //得到文件
        delFile(tardir);                 //调用delFile方法删除文件
    }
}
repaintPanel();                         //调用repaintPanel方法刷新显示文件列表
```

7. 为jButtonSearch按钮添加ActionEvent事件的处理方法：

```
File tardir=new File(jTextFieldPath.getText().toString());
if(!tardir.exists()) {
    JOptionPane.showMessageDialog(null,"指定的目录不存在！");
    return;
}
jPanelFileShow.removeAll(); //文件显示列表清空
jPanelFileShow.repaint();   //刷新显示
searchFile(tardir);         //调用searchFile方法搜索文件
```

8. 为jTextFieldPath文本框添加KeyEvent事件的处理方法：

```
switch(evt.getKeyCode()) {
    case KeyEvent.VK_ENTER: { //按了Enter键
        repaintPanel();      //调用repaintPanel方法刷新显示文件列表
        break;
    }
}
```

程序源代码与解释

```
/** FrameAllRename.java*/
package allrename;
public class FrameAllRename extends javax.swing.JFrame {
    public FrameAllRename() {
```



```

        initComponents();
    }
    //布局设计，源代码略
    private void initComponents() { ... }
    private void jButtonBrowseActionPerformed(java.awt.event.ActionEvent evt) {
        String filename="";
        //参见步骤详解5
    }
    private void jTextFieldPathKeyPressed(java.awt.event.KeyEvent evt) {
        //参见步骤详解8
    }
    private void jCheckBoxSelectAllActionPerformed(java.awt.event.ActionEvent evt) {
        AddCheckBox addcb =new AddCheckBox();
        //每单击一次，在全选和不选之间转换
        if(jCheckBoxSelectAll.isSelected()) {
            addcb.selected(true);
        } else {
            addcb.selected(false);
        }
    }
    private void jCheckBoxFileNameActionPerformed(java.awt.event.ActionEvent evt) {
        if(jCheckBoxFileName.isSelected()) { //文件名选中后编号才可用
            jCheckBoxNumber.setEnabled(true);
            jTextFieldNumber.setEnabled(true);
        } else {
            jCheckBoxNumber.setEnabled(false);
            jTextFieldNumber.setEnabled(false);
        }
    }
    private void jButtonRenameActionPerformed(java.awt.event.ActionEvent evt) {
        int dizen=0; //递增参数初始化等于0
        if(jTextFieldNumber.isEnabled()&&(jTextFieldNumber.getText().toString().length()>0)) {
            //如果编号文本框可用并且不为空
            dizen=Integer.parseInt(jTextFieldNumber.getText()); //将编号文本框的值赋给变量
            dizhen
        }
        for(int i=0;i<jPanelFileShow.getComponentCount();++i) {
            String temp="";
            try{
                JCheckBox tempjcb= (JCheckBox)jPanelFileShow.getComponent(i);
                //将jPanelFileShow中组件的类型转换成JCheckBox
                if (tempjcb.isSelected()) { //如果被选中
                    File tardir=new File(tempjcb.getText().trim()); //得到文件
                    if(!tardir.exists()){
                        JOptionPane.showMessageDialog(null,"指定的文件不存在！");
                    } else {
                        try{
                            temp= renameFile(tardir,dizen); //调用renameFile函数将变量
                            dizhen传入函数renameFile
                            dizen++; //变量dizhen进行递增
                        }catch(Exception exp){
                            JOptionPane.showMessageDialog(null,"可能名称有重复。无法改名");
                        }
                    }
                    if (temp!="...") {
                        tempjcb.setText(temp); //将jPanelFileShow中JCheckBox的名称改
                        为改名后的文件名
                    }
                }
            } catch (Exception ex) {
                //异常处理
            }
        }
    }

```



```

        }
    }
    }
    } catch (Exception exp) {
        JOptionPane.showMessageDialog(null, "找不到需要的组件!");
    }
}

class AddCheckBox {    //定义一个增加JCheckBox的类
    Vector checkboxes = new Vector();    //存放JCheckBox组件
    public void AddCheckBox(File tardir) {    //构造方法
        if (tardir.exists()) {
            File[] file = tardir.listFiles();
            if (file != null && file.length > 0) {
                for (int x = 0; x < file.length; x++) {
                    if (!file[x].isDirectory()) {
                        JCheckBox jcb = (JCheckBox) jPanelFileShow.add(new
JCheckBox());
                                checkboxes.addElement(jcb);
                                jcb.setText(file[x].getPath().toString());
                            }
                        }
                    }
                }
            }
        }
    }
    //判断是否被选择
    public void selected(boolean selected) {
        for (int i = 0; i < jPanelFileShow.getComponentCount(); ++i) {    //得到jPanelFileShow
中所有组件的数目
            try {
                JCheckBox tempjcb = (JCheckBox) jPanelFileShow.getComponent(i);
                //将jPanelFileShow中组件的类型转换成JCheckBox
                tempjcb.setSelected(selected);
            } catch (Exception e) {
                JOptionPane.showMessageDialog(null, "找不到组件!");
            }
        }
    }
    }
    private void delFile(File tardir) {
        Object[] options = {"确定", "取消"};
        int choose = JOptionPane.showOptionDialog(this, "单击确定将删除:" + tardir.getName(), "警告:
如果删除将无法恢复", JOptionPane.OK_CANCEL_OPTION, JOptionPane.INFORMATION_MESSAGE, null, op-
tions, options[0]);
        if (choose != 0) {
            return;
        } else {
            tardir.deleteOnExit();
            if (tardir.delete()) {
                JOptionPane.showMessageDialog(null, "成功删除 " + tardir.getName());
            } else {
                JOptionPane.showMessageDialog(null, "删除失败!");
            }
        }
    }
    private void repaintPanel() {
        File tardir;
        jPanelFileShow.removeAll();    //清空显示
        jPanelFileShow.repaint();
    }
}

```



```

        tardir=new File(jTextFieldPath.getText().toString().trim());
        if(tardir.isDirectory()){
            AddCheckBox add =new AddCheckBox();
            add.AddCheckBox(tardir);          //调用AddCheckBox类更新显示
        }
    }

    public void searchFile(File tardir) {      //搜索文件
        String name;
        String strname;
        String str2;
        int count=0;
        File[] list=tardir.listFiles();
        if(list!=null && list.length>0) {
            for(int x=0;x<list.length;++x) {
                if(list[x].isFile()){          //是文件
                    name=jTextFieldSearch.getText().trim();
                    strname=list[x].getName().toString().trim();
                    //System.out.println(strname+name);
                    //System.out.println(strname+name+strname.indexOf(name));
                    //注意indexOf的用法
                    if(strname.indexOf(name)!=-1 || strname.endsWith(name) ){//符合搜索条件
                        System.out.println(list[x].getPath());
                        AddCheckBox addjcb =new AddCheckBox();
                        addjcb.AddCheckBox(list[x]);
                        count++;
                        tardir=new File(list[x].getPath().trim());
                        Vector checkboxes =new Vector();
                        JCheckBox jcb=(JCheckBox)jPanelFileShow.add(new JCheckBox());
                        checkboxes.addElement(jcb);
                        jcb.setText(list[x].getPath().toString());
                    }
                } else {          //是目录
                    searchFile(list[x]);      //递归调用
                }
            }
        }
    }
}

```



案例5：文件分割器

案例运行效果与操作

本案例可以对指定文件进行分割与合并，分割时可以按文件大小以及文件数量进行。程序运行后，界面如图3-25所示。

界面分为三部分，上方用来选择分割操作还是合并操作。根据选择不同，在中间部分分别显示“文件分割”和“文件合并”的操作面板，默认显示“文件分割”操作面板。选择上方的“合并”单选按钮后，出现合并操作界面，如图3-26所示。界面下方是一个文本区，用来显示操作说明以及相应的提示信息。



图3-25 运行时界面

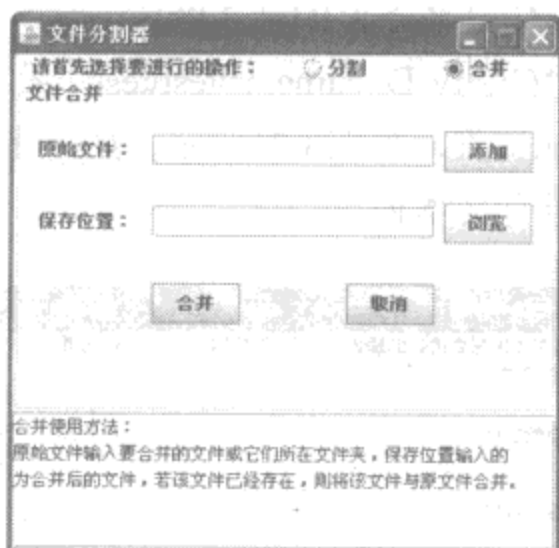


图3-26 合并操作界面

对于分割操作, 在“原始文件”文本框中输入要分割的文件路径(或者单击右侧“浏览”按钮选择文件, 下同), 在“保存位置”文本框中输入分割后的保存文件路径, 此时在下方文本区会显示相应的目录或文件信息。然后根据需要选择分割方式, 若选中“分割尺寸”单选按钮, 右方的尺寸单位下拉列表框可用, 里面有三种单位供选择, 分别是MB、KB和B(字节), 界面如图3-27所示。而选中“分割数量”单选按钮后, 就可以在右方的文本框中直接输入想要分割成的文件数。

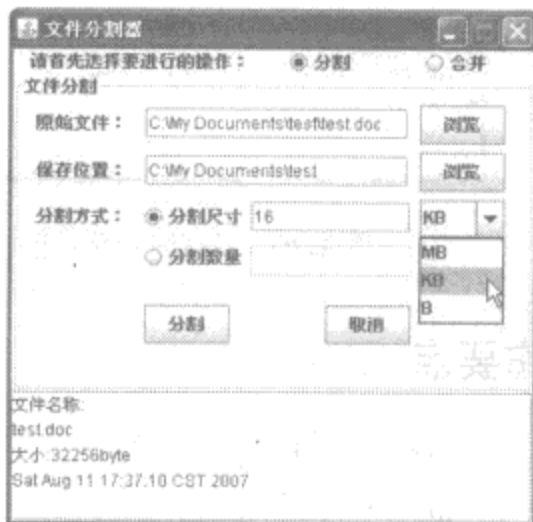


图3-27 分割设置

设置完毕, 单击“分割”按钮, 会在下方的文本区中显示操作信息以及分割是否成功的提示, 如图3-28所示。而此时查看资源管理器中相应目录的信息, 发现确实多了2个分割后形成的文件test.doc(0)和test.doc(1), 如图3-29所示。单击“取消”按钮, 则清空所有文本框。



图3-28 文件分割后的界面

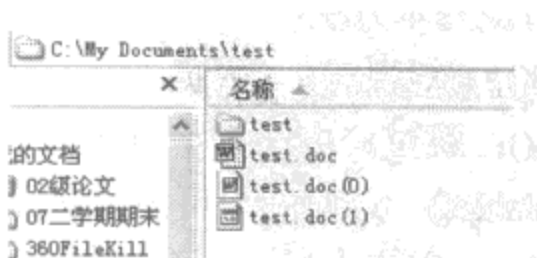


图3-29 资源管理器中的界面

对于合并操作, 在“原始文件”文本框中输入或者单击右侧“添加”按钮添加要合并的文件, 每次操作都会添加一个文件并在下方的文本区显示出来, 界面如图3-30所示。添加完毕, 在“保存位置”文本框中输入合并后的保存文件路径, 单击“合并”按钮, 程序就会进

行合并操作并在下方的文本区中显示操作信息以及合并是否成功的提示。此时由于指定的保存位置“C:\My Documents\test\test”是一个目录而没有指定文件名，所以在该目录下生成了一个union文件，如图3-31所示。如果指定了文件，则合并到该文件中。单击“取消”按钮，则清空所有文本框。



图3-30 合并设置界面



图3-31 资源管理器界面

制作要点

1. java.io.File类的用法。
2. java中字节流的用法。

Java所有的I/O机制都是基于数据流的，这些数据流表示了字符或者字节数据的流动序列。Java的I/O流提供了读写数据的标准方法。任何Java中表示数据源的对象都会提供以数据流的方式读写它的数据的方法。

Java.io是大多数面向数据流的输入/输出类的主要软件包。这个软件包包含了两个抽象类：InputStream和OutputStream。它们是表示字节输入/输出流的所有类的超类，所有其他面向数据流的输入/输出类都要扩展这两个基类。

字节流是从InputStream和OutputStream派生出来的一系列类，这类流以字节（byte）为基本处理单位，包括以下几类：FileInputStream、FileOutputStream、PipedInputStream、PipedOutputStream、ByteArrayInputStream、ByteArrayOutputStream、FilterInputStream、FilterOutputStream、DataInputStream、DataOutputStream、BufferedInputStream、BufferedOutputStream。

字节流基本方法：

read(): 从流中读入数据。

skip(): 跳过流中若干字节数。

available(): 返回流中可用字节数。

mark(): 在流中标记一个位置。

reset(): 返回标记过的位置。

markSupport(): 是否支持标记和复位操作。

write(): 输出到流。

flush(): 刷空输出流。

3. CardLayout布局管理器的用法。

CardLayout对象是容器的布局管理器，它将容器中的每个组件当做一个卡片来处理。在某个时间，只有一个卡片是可见的，容器像一个卡片堆栈一样工作。本案例使用它来根据用户的操作显示“文件分割”和“文件合并”窗口。

步骤详解

1. 界面设计。创建JFrame窗体，添加2个JPanel和1个JScrollPane，并按照表3-15所示修改属性。

表3-15 属性与值对照表

控件类型	控件名称	“方向”属性值
JPanel	jPanelNorth	North
JPanel	jPanelCenter	Center
JScrollPane	jScrollPaneSouth	South

向jPanelNorth添加1个JLabel、1个ButtonGroup和2个JRadioButton，并按照表3-16所示修改属性。

表3-16 属性与值对照表

控件类型	控件名称	属性设置
JLabel	jLabelNorth	“Text”属性设置为“请首先选择要进行的操作：”
ButtonGroup	ButtonGroupNorth	
JRadioButton	jRadioButtonSplitter	“Text”属性设置为“分割”，“ButtonGroup”属性设置为“ButtonGroupNorth”，“Selected”属性设置为选中状态
JRadioButton	jRadioButtonUnion	“Text”属性设置为“合并”，“ButtonGroup”属性设置为“ButtonGroupNorth”，“Selected”属性设置为未选中状态

向jPanelCenter添加2个JPanel，并按照表3-17所示修改属性。

表3-17 属性与值对照表

控件类型	控件名称	border属性值	TitledBorder “标题”属性值	“卡名”属性值
JPanel	jPanelSplitter	TitledBorder	分割	cardSplitter
JPanel	jPanelUnion	TitledBorder	合并	cardUnion

向jPanelSplitter添加3个JLabel、4个JTextField、4个JButton、1个JComboBox、1个Button-Group和2个JRadioButton，并按照表3-18所示修改属性。

表3-18 属性与值对照表

控件类型	控件名称	text属性值
JLabel	jLabelSplitterSource	原始文件:
JLabel	jLabelSplitterTarget	保存位置:
JLabel	jLabelSplitterWay	分割方式:
JTextField	jTextFieldSplitterSource	
JTextField	jTextFieldSplitterTarget	
JTextField	jTextFieldSize	
JTextField	jTextFieldNumber	
JButton	jButtonSplitterSourceBrowse	浏览
JButton	jButtonSplitterTargetBrowse	浏览
JButton	jButtonSplitter	分割
JButton	jButtonSplitterCancel	取消
ButtonGroup	ButtonGroupCenter	
JRadioButton	jRadioButtonSize	分割尺寸:
JRadioButton	jRadioButtonNumber	分割数量:
JComboBox	jComboBoxSize	

向jPanelUnion添加2个JLabel、 2个JTextField、 4个JButton， 并按照表3-19所示修改属性。

表3-19 属性与值对照表

控件类型	控件名称	text属性值
JLabel	jLabelUnionSource	原始文件:
JLabel	jLabelUnionTarget	保存位置:
JTextField	jTextFieldUnionSource	
JTextField	jTextFieldUnionTarget	
JButton	jButtonAdd	添加
JButton	jButtonUnionTargetBrowse	浏览
JButton	jButtonUnion	合并
JButton	jButtonUnionCancel	取消

再调整各个控件实例在窗体上的位置与大小， 界面最终设置效果如图3-32所示。

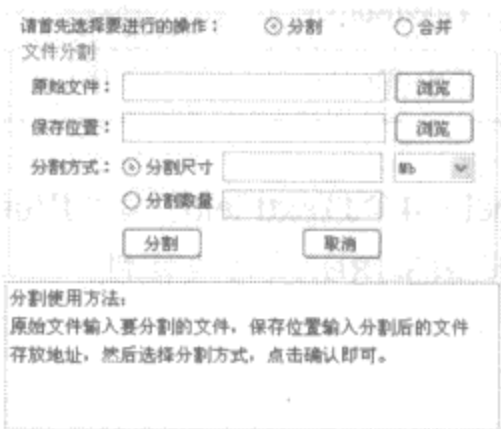


图3-32 界面最终设置效果

2. CardLayout布局设计:

```

jPanelCenter.setLayout(new java.awt.CardLayout());
CardLayout cardUnion=(CardLayout) jPanelCenter.getLayout();
CardLayout cardSplitter=(CardLayout) jPanelCenter.getLayout();

```

3. 创建输入/输出流:

```

FileInputStream in = null;    //输入流
FileOutputStream out;        //输出流
out = new FileOutputStream(outFile, true); //创建输出流
in = new FileInputStream(yuanwenjian);    //创建输入流

```

本案例中使用了FileInputStream和FileOutputStream字节流, 案例6中将涉及到PipedInputStream、PipedOutputStream、PrintStream (FilterOutputStream类的子类) 等字节流的用法。

4. 按文件长度分割:

```

if (jRadioButtonSize.isSelected()) { //按文件长度分割
    //文件尺寸以字节为单位
    if (jComboBoxSize.getSelectedItem() == "MB")
        fileSize = (int) (Float.parseFloat(jTextFieldSize.getText()) * 1024 *
1024);
    if (jComboBoxSize.getSelectedItem() == "KB")
        fileSize = (int) (Float.parseFloat(jTextFieldSize.getText()) * 1024);
    if (jComboBoxSize.getSelectedItem() == "B")
        fileSize = (int) (Float.parseFloat(jTextFieldSize.getText()));
    fileNum = (int) (sourceFile.length() / fileSize); //计算分割成的文件数
    if (sourceFile.length() % fileSize != 0) {
        fileNum++;
    }
}

```

5. 按文件数进行分割:

```

if (jRadioButtonNumber.isSelected()) { //按文件数进行分割
    fileNum = Integer.parseInt(jTextFieldNumber.getText()); //获取输入的文件数
    fileSize = ((int) (sourceFile.length() / fileNum)) + 1; //计算分割文件长度
}

```

6. 创建File对象, 保存分割后的文件:

```

File f = new File(jTextFieldSplitterTarget.getText().replace("\\", "/"));
if (!f.exists())
    f.createNewFile(); //如果文件不存在, 则创建新文件
for (i = 0; i < newFile.length; i++) {
    jTextAreaSouth.append("正在创建第" + (i + 1) + "个文件...\n");
    //依次创建分割文件对象
    newFile[i] = new File(jTextFieldSplitterTarget.getText().replace("\\", "/"),
    sourceFile.getName()+"(" + i + ")");
}

```

7. 合并文件处理:

```

File outFile = new File(jTextFieldUnionTarget.getText()); //输出流的目标文件
File[] unionFiles; //要合并的文件组
File yuanwenjian = new File(jTextFieldUnionSource.getText());
//当源文件与保存的文件都不是文件夹时

```



```

if (yuanwenjian.isFile() && new File(jTextFieldUnionTarget.getText()).isFile() && Number == 0) {
    try {
        out = new FileOutputStream(outFile, true); //创建输出流
        in = new FileInputStream(yuanwenjian);      //创建输入流
        while ((count = in.read(buff, 0, 1024)) != -1) {
            out.write(buff, 0, count);
        }
        in.close();
        out.close();
        jTextAreaSouth.append("\n\n合并成功！！");
    }
}

```

8. 设置文件选择器，用法和前面的案例类似：

```

JFileChooser fc=new JFileChooser();          //创建一个文件选择器
int returnVal=fc.showOpenDialog(this);      //弹出一个打开文件选择器
//如果用户选择了一个文件，则将文件的绝对路径取出
if(returnVal==JFileChooser.APPROVE_OPTION){
    filename=fc.getSelectedFile().getAbsolutePath();
}

```

9. 为jTextFieldUnionSource文本框添加KeyEvent事件的处理方法：

```

switch(evt.getKeyCode()) {
    case KeyEvent.VK_ENTER: {          //按了Enter键
        String filename=jTextFieldUnionSource.getText(); //获取文本
        String str = filename.replace("\\", "/");      //将文件名中"\"换为"/"
        File aimFile = new File(str);
        if (aimFile.exists()) {          //当源文件存在时显示其信息
            if (aimFile.isDirectory()) {    //当为文件夹时提示用户选择文件
                ...
            }
        }
    }
}

```

程序源代码与解释

```

/** FrameFileSplitter.java*/
public class FrameFileSplitter extends javax.swing.JFrame {
    /** Creates new form FrameFileSplitter */
    public FrameFileSplitter() {
        initComponents();
    }
    static String[] unionFileNames=new String[20];    //记录要合并的文件名
    static int Number = 0;        //要合并的文件数组中的文件个数
    private void initComponents() {
        //界面设计，初始化，代码略
        ...
    }
    private void jTextFieldSplitterSourceKeyPressed(java.awt.event.KeyEvent evt) {
        switch(evt.getKeyCode()) {
            case KeyEvent.VK_ENTER: { //按了Enter键
                String filename=jTextFieldSplitterSource.getText();    //获取文本
                String str = filename.replace("\\", "/");    //将文件名中"\"换为"/"
                File aimFile = new File(str);
                if (aimFile.exists()) { //当源文件存在时显示其信息
                    if (aimFile.isDirectory()) { //当为文件夹时提示用户选择文件
                        jTextAreaSouth.setText("警告！！！\n该目标是一个文件夹\n文件夹名称

```

: \n"


```

+ aimFile.getName() + "\n文件数目:"
+ aimFile.listFiles().length + "\n"
+ new Date(aimFile.lastModified()) + "\n请正确选择文
件！！");

    }
    if (aimFile.isFile()) { //当为文件时显示文件信息
        JTextAreaSouth.setText("文件名称:\n" + aimFile.getName()
            + "\n大小:" + aimFile.length() + "byte\n"
            + new Date(aimFile.lastModified()));
    }
    } else {
        JTextAreaSouth.setText("\n\n对不起，找不到源文件！\n");
    }
}
}

private void jButtonAddActionPerformed(java.awt.event.ActionEvent evt) {
    String filename="";
    //参见步骤详解8
    jTextFieldUnionSource.setText(filename); //在“原始文件”文本框中显示
    //待合并文件数加1并将文件名存储到unionFileNames数组中
    unionFileNames[Number++] = jTextFieldUnionSource.getText();
    //显示待合并文件信息
    JTextAreaSouth.setText("将要合并的文件: \n");
    for (int i = 0; i < Number; i++)
        JTextAreaSouth.append(new File(unionFileNames[i]).getName() + "\n");
}

private void jButtonUnionBrowseActionPerformed(java.awt.event.ActionEvent evt) {
    String filename="";
    //参见步骤详解8
    jTextFieldUnionTarget.setText(filename); //在“保存位置”文本框中显示
}

private void jButtonUnionCancelActionPerformed(java.awt.event.ActionEvent evt) {
    //单击“取消”按钮则清空所有文本框
    jTextFieldUnionSource.setText("");
    jTextFieldUnionTarget.setText("");
}

private void jButtonSplitterTargetBrowseActionPerformed(java.awt.event.ActionEvent evt) {
    String filename="";
    //参见步骤详解8
    jTextFieldSplitterTarget.setText(filename); //在“保存位置”文本框中显示
}

private void jButtonSplitterSourceBrowseActionPerformed(java.awt.event.ActionEvent evt) {
    String filename="";
    //参见步骤详解8
    jTextFieldSplitterSource.setText(filename); //在“原始文件”文本框中显示
    String str = filename.replace("\\", "/"); //将文件名中"\"换为"/"
    File aimFile = new File(str);
    if (aimFile.exists()) { //当源文件存在时显示其信息
        if (aimFile.isDirectory()) { //当为文件夹时
            JTextAreaSouth.setText("警告！！\n该目标是一个文件夹\n文件夹名称:\n"+ aimFile
                .getName() + "\n文件数目:" + aimFile.listFiles().length + "\n"+ new Date(aimFile.lastModified()) + "\n请正确选
                择文件！！");
        }
        if (aimFile.isFile()) { //当为文件时
            JTextAreaSouth.setText("文件名称:\n" + aimFile.getName()+ "\n大小:" + aimFile.length()
                + "byte\n"

```



```

+ new Date(aimFile.lastModified()));
    }
} else {
    jTextAreaSouth.setText("\n\n对不起，找不到源文件！\n");
}
}

private void jButtonSplitterCancelActionPerformed(java.awt.event.ActionEvent evt) {
//单击“取消”按钮则清空所有文本框
    jTextFieldNumber.setText("");
    jTextFieldSplitterSource.setText("");
    jTextFieldSplitterTarget.setText("");
    if (jRadioButtonSize.isSelected()) { //按文件长度分割
        jTextFieldSize.setText("");
    } else {
        jTextFieldNumber.setText("");
    }
}

private void jRadioButtonNumberItemStateChanged(java.awt.event.ItemEvent evt) {
//选中“分割数量”单选按钮，则“文件尺寸”文本框和复合框不可用
    jTextFieldSize.setEnabled(false);
    jComboBoxSize.setEnabled(false);
    jTextFieldNumber.setEnabled(true);
}

private void jRadioButtonSizeItemStateChanged(java.awt.event.ItemEvent evt) {
//选中“分割尺寸”单选按钮，则“文件数量”文本框不可用
    jTextFieldSize.setEnabled(true);
    jComboBoxSize.setEnabled(true);
    jTextFieldNumber.setEnabled(false);
}

private void jRadioButtonUnionItemStateChanged(java.awt.event.ItemEvent evt) {
//选中“合并”单选按钮，则显示文件合并JPanel以及合并提示信息
    CardLayout cardUnion=(CardLayout) jPanelCenter.getLayout();
    cardUnion.show(jPanelCenter,"cardUnion");
    jTextAreaSouth.setText("合并使用方法：\n原始文件输入要合并的文件或它们所在文件夹，
保存位置输入的\n为合并后的文件，若该文件已经存在，则将该文件与源文件合并。");
}

private void jRadioButtonSplitterItemStateChanged(java.awt.event.ItemEvent evt) {
//选中“分割”单选按钮，则显示文件分割JPanel以及分割提示信息
    CardLayout cardSplitter=(CardLayout) jPanelCenter.getLayout();
    cardSplitter.show(jPanelCenter,"cardSplitter");
    jTextAreaSouth.setText("分割使用方法：\n原始文件输入要分割的文件，保存位置输入分割
后的文件\n存放地址，然后选择分割方式，单击确认即可。");
}
}
}

```



案例6：管道流实现线程间的通信

案例运行效果与操作

在Java语言中，有各种各样的输入输出流（stream），用户能够方便地对数据进行操作。其中，管道（pipe）流是一种特殊的流，可以在不同线程（threads）间直接传送数据。一个线程发送数据到输出管道，另一个线程从输入管道中读数据。本案例通过使用管道，实现不同线程间的通信，无需求助于类似临时文件之类的东西。

生成项目后在DOS窗口中执行命令`java -jar pipe.jar`，运行后，出现如图3-33所示的界面。

```
G:\javabook\chapter3\pipe\dist>type input.txt
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
G:\javabook\chapter3\pipe\dist>java -jar pipe.jar
转换后结果如下：
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
```

图3-33 运行界面

由图中可以看出，通过`type`命令显示的`input.txt`文件内容全部为“x”，而运行本案例后，显示出的内容全部为“z”，表明程序转换成功。

制作要点

1. java中字符流的使用。

字符流主要是用来处理字符的，是从`Reader`和`Writer`派生出来的一系列类，以16位的Unicode码表示的字符为基本处理单位，包括以下几类：`InputStreamReader`、`OutputStreamWriter`、`FileReader`、`FileWriter`、`CharArrayReader`、`CharArrayWriter`、`PipedReader`、`PipedWriter`、`FilterReader`、`FilterWriter`、`BufferedReader`、`BufferedWriter`、`StringReader`、`StringWriter`。

本案例使用了`BufferedReader`类。

2. 管道的创建与使用。

Java提供了两个特殊、专门的类用于处理管道，它们就是`PipedInputStream`类和`PipedOutputStream`类，属于字节流的高级应用。

`PipedInputStream`代表了数据在管道中的输出端，也就是线程向管道读数据的一端；`PipedOutputStream`代表了数据在管道中的输入端，也就是线程向管道写数据的一端。这两个类一起使用可以提供数据的管道流。

为了创建一个管道流，必须首先创建一个`PipedOutputStream`对象，然后创建`PipedInputStream`对象，一旦创建了一个管道后，就可以像操作文件一样对管道进行数据的读写。

3. 多线程的用法。

4. 程序设计。

本案例由三个类组成：主线程（`Pipe.java`）及由主线程启动的两个二级线程（`yThread.java`和`zThread.java`），它们使用管道来处理数据。程序从内容为一行一行“x”字母的“`input.txt`”文件中读取数据，使用管道传输数据，先是利用线程`yThread`将数据“x”转换为“y”，然后利用线程`zThread`将“y”转换为“z”，最后程序在屏幕上显示修改后的数据。

步骤详解

1. `BufferedReader`。

```
BufferedReader inputStream = new BufferedReader(new InputStreamReader(zInpipe));
```


不同于其他编程语言，Java有一系列流类型，其数量超过60种。类库的设计者声称：“有足够的理由为用户提供丰富的流类型的选择：这样做可以减少程序的错误。”这么多的类型，在使用中如果巧妙利用，会得到事半功倍的效果。在本书和本书第一版中前前后后介绍过多种流类型的用法，下面简单总结一下File、FileInputStream、FileReader、InputStreamReader、BufferedReader的使用和区别。

• File类

File类封装了对用户机器的文件系统进行操作的功能。例如，可以用File类获得文件上次修改的时间，或者对文件进行删除、重命名。换句话说，流类关注的是文件内容，而File类关注的是文件在磁盘上的存储。File类不属于文件流，只能代表一个文件或是目录的路径名。流类关注的是文件内容，而File类关注的是文件在磁盘上的存储。

File类的主要方法有：getName()、getCanonicalFile()、lastModified()、isDerector()、isFile()和getPath()等。

注意：如果处理文件或者目录名，就应该使用File对象，而不是字符串。

• FileInputStream类

FileInputStream类是以字节为单位（非Unicode）的流处理。字节序列即二进制数据，与编码无关，不存在乱码问题。

FileInputStream类的主要方法有：Read()、read(byte[] b)、read(byte[],int off,int len)和available()。

FileInputStream类与FileReader类的构造函数的形式和参数都是相同，它们的区别如下。

FileInputStream类以二进制输入/输出，输入/输出速度快且效率高，但是它的read()方法读到的是一个字节（二进制数据），很不利于人们阅读。而FileReader类可以以文本格式输入/输出，非常方便，比如可以使用BufferedReader的readLine()方法一行一行地读取文本。

InputStreamReader和BufferedReader中最重要的类是InputStreamReader，它是字节转换为字符的桥梁。可以在构造器中指定编码的方式，如果不指定的话将采用底层操作系统的默认编码方式，例如GBK等。FileReader与InputStreamReader涉及编码转换（指定编码方式或者采用操作系统默认编码），可能在不同的平台上出现乱码现象，而FileInputStream以二进制方式处理，不会出现乱码现象。

注意：如果处理纯文本文件，建议使用FileReader，因为更方便，也更适合阅读，但是要注意编码问题；如果处理非纯文本文件，FileInputStream是唯一的选择，FileInputStream在进行Socket通信时会用到很多，如将文件流以Stream的方式传向服务器。

• FileReader类

FileReader类是InputStreamReader类的子类。FileReader类与它的父类InputStreamReader的主要不同在于构造函数，当要指定编码方式时，必须使用InputStreamReader类；而FileReader构造函数的参数与FileInputStream不同，为File对象或表示路径的String，当要根据File对象或者String读取一个文件时，要用FileReader。

• InputStreamReader类

以文本格式输入/输出，可以指定编码格式。

主要方法有：getEncoding()和read()。

• BufferedReader类

BufferedReader由Reader类扩展而来，提供通用的缓冲方式文本读取，而且提供了很实用的readLine，读取分行文本很适合。BufferedReader是针对Reader的，不直接针对文件的，也不是只针对文件读取。

2. 创建输入输出管道:

```
PipedOutputStream pipeOutY = new PipedOutputStream();
PipedInputStream pipeInY = new PipedInputStream(pipeOutY);
```

管道创建之后，对管道的数据读写可以像操作文件一样。管道输入流应该连接到管道输出流，管道输入流提供要写入管道输出流的所有数据字节。通常，数据由某个线程从PipedInputStream对象读取，并由其他线程将其写入到相应的PipedOutputStream，所以一般管道输入流和管道输出流应用多线程的环境下。Pipe的构造方法如下：

- (1) int available(): 返回可以不受阻塞地从此输入流中读取的字节数。
- (2) void close(): 关闭此管道输入流并释放与该流相关的所有系统资源。
- (3) void connect(PipedOutputStream src): 使此管道输入流连接到管道输出流src。
- (4) int read(): 读取此管道输入流中的下一个数据字节。
- (5) int read(byte[] b, int off, int len): 将最多len个数据字节从此管道输入流读入byte数组。
- (6) protected void receive(int b): 接收数据字节。

3. 将管道输入流定位到BufferedReader对象，使程序能够使用readLine()方法逐行读取数据:

```
BufferedReader inputStream=new BufferedReader(new InputStreamReader(zInpipe));
String myString = inputStream.readLine();
```

程序源代码与解释

```
/** Pipe.java*/
public class Pipe {
    public static void main(String[] args) {
        Pipe pipeExample=new Pipe();
        try { //为"input.txt"文件创建了一个输入流xFileIn
            FileInputStream xFileIn =new FileInputStream("input.txt");
            //新的输入流传递给changeToY()方法，让线程ythread能读取该文件
            InputStream yInpipe = pipeExample.changeToY(xFileIn);
            //传递给changeToZ()方法，让线程zthread能读取该文件
            InputStream zInpipe=pipeExample.changeToZ(yInpipe);
            System.out.println();
            System.out.println("转换后结果如下: ");
            System.out.println();
            //将管道输入流定位到BufferedReader对象，使程序能够使用readLine()方法读取数据
            BufferedReader inputStream = new BufferedReader(new InputStreamReader(zInpipe));
            String myString = inputStream.readLine();
            //逐行读取数据并显示在屏幕上
            while (myString!=null) {
                System.out.println(myString);
                myString=inputStream.readLine();
            }
        }
    }
}
```



```

    }
    inputStream.close();    //关闭输入流
} catch(Exception e) {
    System.out.println(e.toString());
} }
//changeToY()方法创建将输入数据“x”改变到“y”的线程ythread，并返回该线程的输入管道
public InputStream changeToY(InputStream inputstream) {
    try {
        //通过传递一个参数InputStream给BufferedReader对象定位资源的输入流
        //使程序能使用readLine()方法从流中读取数据
        BufferedReader xFileIn = new BufferedReader(new InputStreamReader(inputstream));
        //创建输出管道和输入管道
        PipedOutputStream pipeOutY = new PipedOutputStream();
        PipedInputStream pipeInY = new PipedInputStream(pipeOutY);
        //为了能够使用println()方法输出修改后的文本行到管道
        //程序将输出管道定位到PrintStream对象
        PrintStream printstreamY = new PrintStream(pipeOutY);
        //创建将数据从“x”改变到“y”的线程
        yThread ythread = new yThread(xFileIn,printstreamY);
        //启动线程
        ythread.start();
        //返回该线程的输入管道
        return pipeInY;
    } catch(Exception e) { System.out.println(e.toString());}
    return null;
}

```

/*changetoZ()方法启动将数据从“y”改变到“z”的线程zthread，主程序将使用从changeToZ()返回的输入管道，得到已修改的数据。*/

```

public InputStream changeToZ(InputStream inputstream) {
    try {/*通过传递一个参数InputStream给BufferedReader对象定位资源的输入流，使程序能
    使用readLine()方法从流中读取数据*/
        BufferedReader yFileIn = new BufferedReader(new InputStreamReader(inputstream));
        PipedOutputStream pipeOutZ = new PipedOutputStream();    //创建输出管道和输入
管道
        PipedInputStream pipeInZ = new PipedInputStream(pipeOutZ);
        //为了能够使用println()方法输出修改后的文本行到管道，程序将输出管道定位到
PrintStream对象
        PrintStream printstreamZ = new PrintStream(pipeOutZ);
        zThread zthread = new zThread(yFileIn,printstreamZ);    //创建将数据从“y”
改变到“z”的线程
        zthread.start();    //启动线程
        return pipeInZ;    //返回该线程的输入管道
    } catch(Exception e) { System.out.println(e.toString());}
    return null;
}
}

```

/** yThread.java*/

```

class yThread extends Thread {
    //构造方法接收两个参数：输入的文件和第一个管道的输出端
    BufferedReader xFileIn;
    PrintStream printstream;
    yThread(BufferedReader xFileIn,PrintStream printstream) {
        this.xFileIn = xFileIn;
    }
}

```



```

        this.printstream = printstream;
    }
    public void run() {
        try { //读取一行数据，确保xstring不为空的情况下循环执行
            String xstring = xFileIn.readLine();
            while(xstring!=null) {
                String ystring= xstring.replace('x','y'); //每读一行数据，完成一次转换
                printstream.println(ystring); //将修改后的数据输出到管道的输出端
                printstream.flush(); //确保所有缓冲区的数据完全进入管道的输出端
                xstring= xFileIn.readLine();
            }
            printstream.close(); //关闭管道输出流
        } catch(IOException e) { System.out.println(e.toString());}
    } }

/** zThread.java*/
class zThread extends Thread {
    //构造方法接收两个参数：输入的文件和第一个管道的输出端
    BufferedReader yFileIn;
    PrintStream printstream;
    zThread(BufferedReader yFileIn,PrintStream printstream) {
        this.yFileIn = yFileIn;
        this.printstream = printstream;
    }
    public void run() {
        try { //读取一行数据，确保ystring不为空的情况下循环执行
            String ystring = yFileIn.readLine();
            while(ystring!=null) {
                String zstring= ystring.replace('y','z'); //每读一行数据，完成一次转换
                printstream.println(zstring); //将修改后的数据输出到管道的输出端
                printstream.flush(); //确保所有缓冲区的数据完全进入管道的输出端
                ystring= yFileIn.readLine();
            }
            printstream.close(); //关闭管道输出流
        } catch(IOException e) { System.out.println(e.toString());}
    } }

```

案例7：排 序 对 象

案例运行效果与操作

本案例是一个简单的排序程序，可以对字符串和整数进行排序。在DOS窗口运行生成的jar包文件，运行界面如图3-34所示。

界面中输入了Linda、Tom、Jerry三个字符串作为参数，而程序中待排序的整数为11、2、-22、401、6，由运行后的界面可知，字符串和整数都得到了排序。

制作要点

1. ArrayList类的应用技巧。

ArrayList类是List接口的大小可变数组的实现，实现了所有可选列表操作，并允许包括null在内的所有元素，其实ArrayList就是动态数组。

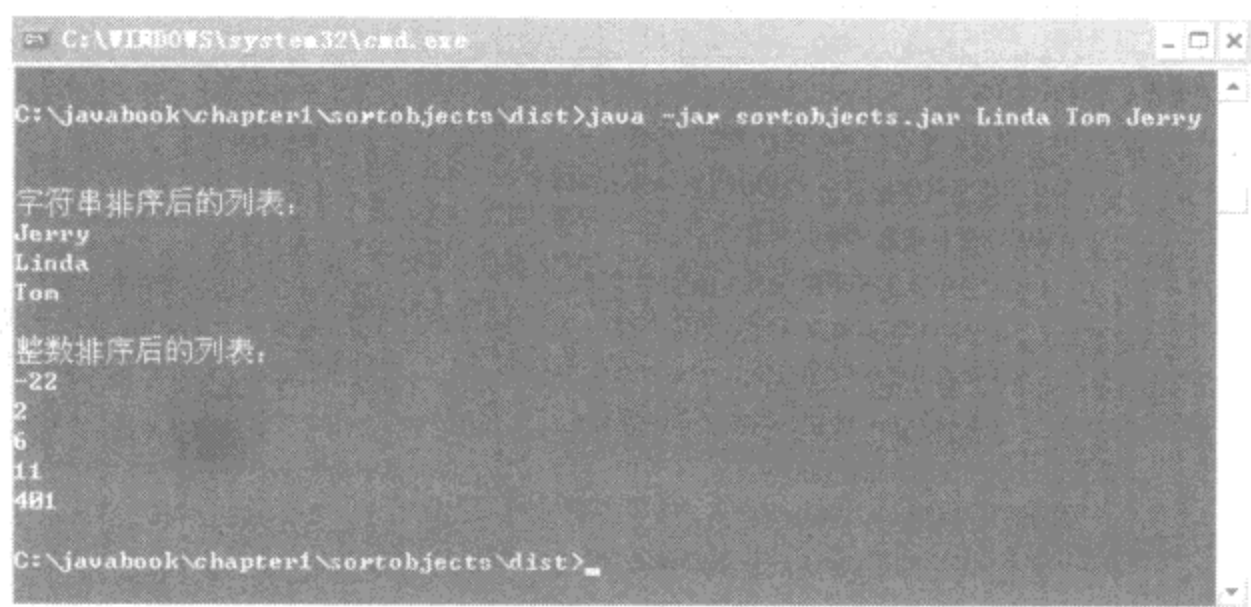


图3-34 运行界面

2. Collections类的应用技巧。

Collections是针对集合（**Collection**）类的一个帮助类，它提供一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作。使用时直接用其静态方法即可，不能进行实例化。本案例使用了其**synchronizedList**方法和**sort**方法。

步骤详解

1. 创建动态数组，并赋值：

```
ArrayList al = new ArrayList();
for (int i = 0; i < o.length; i++)
    al.add(i, o[i]);
```

ArrayList()支持可随需增长的动态数组。但是如果要从中间删除一个对象会影响效率，因为有些未删除的对象要相应地调整位置。它是非线程安全的，但效率比**Vector**要高，如果在单线程下，要选它而不是**Vector**。

2. 获得动态数组的同步拷贝：

```
List list = Collections.synchronizedList(al);
```

synchronizedList()被用来获得各种类集的同步拷贝，它是线程安全的。

3. 排序方法：

```
List myList = sort(args);           //对输入参数中的字符串进行排序
myList = sort(in);                   //对整数序列进行排序
```

Collections类的**sort**方法可以对任何实现了**List**接口的类进行排序。在排序过程中，默认这些类实现了**Comparable**接口，如果想用其他方法排序，可以在调用**sort**方法的时候提供一个**Comparator**对象。读者可以试一下反向排序。

```
Collections.sort(items, itemComparator);
Collections.sort(items, Collections.reverseOrder(itemComparator))
```

程序源代码与解释

```
/* * SortObjects.java*/
```



```
public class SortObjects {
    public static void main (String s[]) {
        List myList = sort(s);
        System.out.println("\nStrings sorted List ...");
        for (int i = 0; i < s.length; i++) {
            System.out.println((String)myList.get(i));
        }
        int myIntegers[] = {11, 2, -22, 401, 6};
        Integer in[] = new Integer[myIntegers.length];
        for (int i = 0; i < in.length; i++) {
            in[i] = new Integer(myIntegers[i]);
        }
        myList = sort(in);
        System.out.println("\nIntegers sorted List ...");
        for (int i = 0; i < in.length; i++) {
            System.out.println((Integer)myList.get(i));
        }
    }
    public static List sort (Object o[]) {
        ArrayList al = new ArrayList(); //创建一个动态数组
        for (int i = 0; i < o.length; i++)
            al.add(i, o[i]);           //动态数组赋值
        List list = Collections.synchronizedList(al); //返回动态数组的同步列表
        Collections.sort(list);           //对同步列表进行排序
        return list;                      //返回
    }
}
```

本章小结

Java的核心库java.io提供了全面的I/O接口，包括文件读写、标准设备输出等。Java中I/O是以流为基础进行输入/输出的，所有数据被串行化写入输出流，或者从输入流读入。Java的I/O体系分Input/Output和Reader/Writer两类，区别是Reader/Writer在读写文本时能自动转换内码。基本上，所有的I/O类都是配对的，即有XxxInput就有一个对应的XxxOutput。本章通过几个不同难易程度的实用案例程序，讲述了在Java语言中具体的各种流操作方法，有助于读者对I/O体系结构概念的全面理解和应用。在实际应用中，由于Java提供的各种I/O类非常广泛，不可能通过几个案例就覆盖到，所以需要读者在实际应用中仔细体会。



第4章 Java与游戏

本章内容

- 案例1: Java扫雷
- 案例2: 黑白棋
- 案例3: 象棋游戏
- 案例4: 一个简单的弹球游戏
- 案例5: 找不同
- 案例6: 八皇后问题
- 本章小结



案例1: Java扫雷

案例运行效果与操作

本案例是用Java实现的一个类似于Windows操作系统中的扫雷游戏的小游戏。运行后界面如图4-1所示。

其操作界面如图4-2所示。在雷区单元中单击左键可以翻开该单元, 或者单击右键为该单元做上地雷标记, 或者单击右键两次为该单元做上疑问标记。翻开的非雷单元会以不同的字体颜色显示相邻的地雷数。

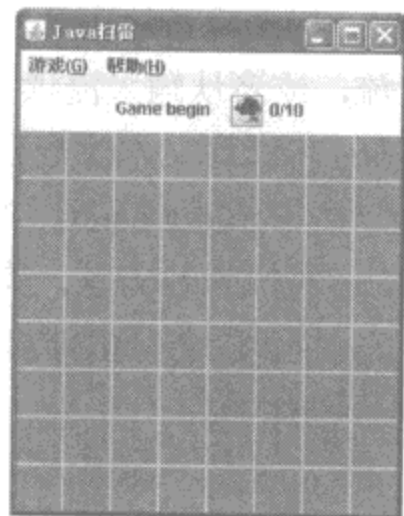


图4-1 运行界面



图4-2 游戏界面

如果游戏过程中翻错雷则游戏结束, 界面如图4-3所示。此时会显示所有的未翻地雷, 同时会在翻错或者标记错的单元上显示叉号。

当所有的单元均正确翻完, 则显示游戏胜利界面, 如图4-4所示。

用户可以通过游戏的“设定”菜单项对游戏参数进行设定, 游戏“设定”对话框如图4-5所示。可以在对话框左侧选择难度, 其中选择“自定义”单选按钮后会激活右侧的“自定义雷区”选项区, 可以指定相应的参数。



图4-3 游戏失败界面

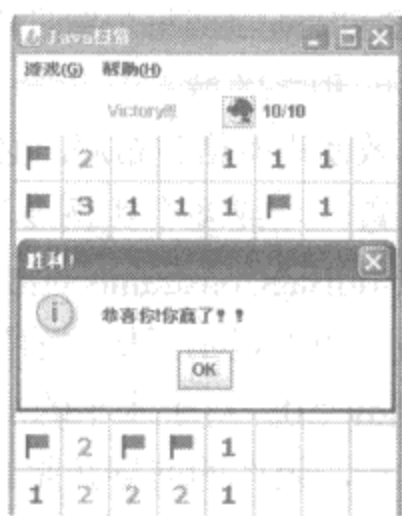


图4-4 游戏胜利界面

单击游戏的“帮助”菜单项，会弹出游戏“帮助”对话框，如图4-6所示。

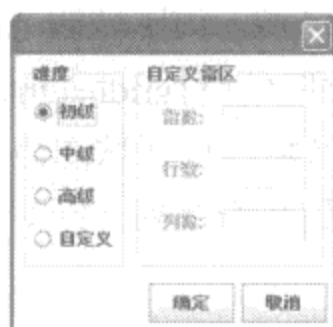


图4-5 游戏“设定”对话框

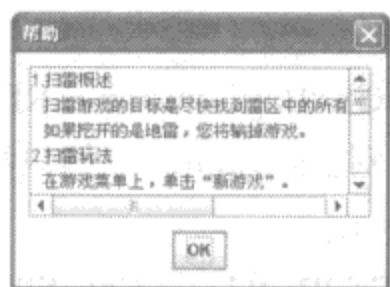


图4-6 游戏“帮助”对话框

制作要点

1. MouseEvent类的应用。

任何组件上都可以发生鼠标事件，如鼠标进入组件、退出组件、在组件上方单击鼠标、拖动鼠标等都属于鼠标事件，也就是说，组件可以成为发生鼠标事件的事件源。

2. GridBagLayout类和GridBagConstraints类的应用技巧。

布局管理器在第1章中有一些的描述，这里重点讲解GridBagLayout类和GridBagConstraints类的应用技巧，使读者熟悉布局管理器的使用。

GridBagLayout类是一个灵活的布局管理器，它不要求组件的大小相同，可以将组件垂直、水平或沿它们的基线对齐。每个GridBagLayout对象维持一个动态的矩形单元网格，每个组件占用一个或多个这样的单元，该单元被称为显示区域。

每个由GridBagLayout管理的组件都与GridBagConstraints的实例相关联。Constraints对象指定组件的显示区域在网格中的具体放置位置，以及组件在其显示区域中的放置方式。除了Constraints对象之外，GridBagLayout还考虑每个组件的最小大小和首选大小，以确定组件的大小。

GridBagLayout是所有AWT布局管理器当中最复杂的，同时它的功能也是最强大的。这种现象源于它所提供的众多的可配置选项，它几乎可以完全地控制容器的布局方式。尽管复杂性很明显，但只要理解了基本思想，就很容易使用GridBagLayout了。

GridBagLayout同GridLayout一样，在容器中以网格形式来管理组件，但GridBagLayout功能却强大得多。

(1) GridBagLayout管理的所有行和列都可以是大小不同的。

(2) **GridLayout**把每个组件限制到一个单元格，而**GridBagLayout**的组件在容器中可以占据任意大小的矩形区域。

为了有效使用网格包布局，必须自定义与组件关联的一个或多个**GridBagConstraints**对象。可以通过设置一个或多个实例变量来自定义**GridBagConstraints**对象。

(1) **GridBagConstraints.gridy**

指定包含组件显示区域的前导角的单元，在此显示区域中，位于网格原点的单元地址是 **gridx = 0, gridy = 0**。对于水平的从左到右的布局，组件的前导角是其左上角。对于水平的从右到左的布局，组件的前导角是其右上角。使用**GridBagConstraints.RELATIVE**（默认值），会将组件直接放置在之前刚添加到容器中的组件的后面（沿X轴向为**gridx**，沿Y轴向为**gridy**）。

(2) **GridBagConstraints.gridwidth**、**GridBagConstraints.gridheight**

指定组件的显示区域中行（针对**gridwidth**）或列（针对**gridheight**）中的单元数。默认值为1。使用**GridBagConstraints.REMAINDER**指定组件的显示区域，该区域的范围是从**gridx**到该行（针对**gridwidth**）中的最后一个单元，或者从**gridy**到该列（针对**gridheight**）中的最后一个单元。使用**GridBagConstraints.RELATIVE**指定组件的显示区域，该区域的范围是从**gridx**到其所在行（针对**gridwidth**）的倒数第二个单元，或者从**gridy**到其所在列（针对**gridheight**）的倒数第二个单元。

(3) **GridBagConstraints.fill**

当组件的显示区域大于组件的所需大小时，用于确定是否（以及如何）调整组件。可能的值为**GridBagConstraints.NONE**（默认值）、**GridBagConstraints.HORIZONTAL**（加宽组件直到它足以在水平方向上填满其显示区域，但不更改其高度）、**GridBagConstraints.VERTICAL**（加高组件直到它足以在垂直方向上填满其显示区域，但不更改其宽度）和**GridBagConstraints.BOTH**（使组件完全填满其显示区域）。

(4) **GridBagConstraints.ipadx**、**GridBagConstraints.ipady**

指定布局中组件的内部填充，即对组件最小大小的添加量。组件的宽度至少为其最小宽度加上**ipadx**像素。类似地，组件的高度至少为其最小高度加上**ipady**像素。

(5) **GridBagConstraints.insets**

指定组件的外部填充，即组件与其显示区域边缘之间间距的最小量。

(6) **GridBagConstraints.anchor**

指定组件应置于其显示区域中何处。可能的值有三种：绝对值、相对于方向的值和相对于基线的值。相对于方向的值是相对于容器的**ComponentOrientation**属性进行解释的，而绝对值则不然。相对于基线的值是相对于基线进行计算的。

(7) **GridBagConstraints.weightx**

用于确定分布空间的方式，这对于指定调整行为至关重要。除非在行（**Weightx**）和列（**Weighty**）中至少指定一个组件的权重，否则所有组件都会聚集在其容器的中央。这是因为，当权重为零（默认值）时，**GridBagLayout**对象会将所有额外空间置于其单元网格和容器边缘之间。

3. 程序设计。

本案例由五个类构成，分别是**MineGame**类、**GamePanel**类、**FieldCell**类、**GraphicsUtil**类和**ConfigDialog**类。其中，**MineGame**类是本程序的主类，它调用**GamePanel**类生成游戏的主界面。

GamePanel类封装了本游戏的所有业务逻辑，主要职责是初始化界面，并对整个游戏的过程进行管理，对用户输入进行响应。它分别调用FieldCell类、GraphicsUtil类和ConfigDialog类来完成对游戏的控制。

FieldCell类用来画出雷区各个单元并对各单元状态信息进行保存，它实际上是调用GraphicsUtil类进行画图，具体的画图操作由GraphicsUtil类完成。而ConfigDialog类则是通过一个对话框来控制游戏的参数，包括雷区的行数、列数以及雷数等，完成游戏的设定功能。

步骤详解

1. 设计游戏面板。

这里用到了多种布局格式，菜单面板放在最上方，使用边界布局管理：

```
getContentPane().add(gamePanel1.panel(), java.awt.BorderLayout.NORTH);
```

状态条面板放在中间，使用边界布局管理：

```
getContentPane().add(gamePanel1.getStatusBar(), java.awt.BorderLayout.CENTER);
```

主面板放在下方，使用边界布局管理：

```
getContentPane().add(gamePanel1, java.awt.BorderLayout.SOUTH);
```

采用流布局管理：

```
statusBar.setLayout(new FlowLayout(FlowLayout.CENTER));
```

组件的显示区域在网格中的具体放置位置以及组件在其显示区域中的放置方式，都可以使用GridBagConstraints来实现：

```
gridBagConstraints = new java.awt.GridBagConstraints(); //创建GridBagConstraints对象
gridBagConstraints.gridx = 0; //组件单元格横坐标
gridBagConstraints.gridy = 4; //组件单元格纵坐标
//用方位进行填充
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
//填充（相对于整个空白区域），横向全部填满
gridBagConstraints.weightx = 1.0;
//填充（相对于整个空白区域），纵向全部填满
gridBagConstraints.weighty = 1.0;
//组件四周的空间（上、左、下、右）
gridBagConstraints.insets = new java.awt.Insets(10, 15, 5, 10);
//按照GridBagConstraints对象的约束将组件添加到“自定义”面板
jPanelCustomization.add(jLabelMine, gridBagConstraints);
```

2. 使用MouseEvent类处理鼠标事件，本案例用到的部分字段和方法如下：

static int BUTTON1: 指示鼠标按键#1；由getButton()方法使用。

static int BUTTON3: 指示鼠标按键#3；由getButton()方法使用。

public int getButton(): 返回更改了状态的鼠标按键，它返回以下常量之一：NOBUTTON、BUTTON1、BUTTON2或BUTTON3。

```
if (e.getButton() == MouseEvent.BUTTON1) { //按下鼠标按键1（左键），进行翻雷
```

本章案例4中将详细讲述如何处理鼠标事件和键盘事件。

3. Pack(): 在JFrame中, 经常会用到pack(), 它的作用是调整此窗口的大小, 以适合其子组件的首选大小和布局。

```
getContentPane().add(gamePanel1.getStatusBar(), java.awt.BorderLayout.CENTER);
pack();
```

当向窗口发送pack()时, 窗口会调整自身大小, 从而提供足够的空间来显示其框架中包含的所有控件。如果组件的对象中没有固定的大小, 使用JFrame.pack()函数会自动裁剪掉组件中所包含的组件的对象的大小, 如果组件对象没有包含任何组件对象, 那么JFrame.pack()函数会被全部裁剪掉, 只剩下标题栏那么大。

4. 设置加速器和快捷键:

```
KeyStroke ks = KeyStroke.getKeyStroke(KeyEvent.VK_X, Event.ALT_MASK);
exit.setAccelerator(ks);
exit.setMnemonic('X');
gameMenu.setMnemonic('G');
KeyStroke ks1 = KeyStroke.getKeyStroke("F2");
newGame.setAccelerator(ks1);
newGame.setMnemonic('N');
setting.setMnemonic('S');
helpMenu.setMnemonic('H');
topHelp.setMnemonic('H');
aboutGame.setMnemonic('A');
```

5. 随机埋雷:

```
while (count < mineNum) {           //随机埋雷
    int s = (int) (Math.random() * totalNum); //产生随机数
    FieldCell fc = cells[s/c][s%c]; //产生随机位置
    if (!fc.isMine()) {
        fc.setMine(true); //埋雷
        count++;
    }
}
```

6. 为FieldCell类添加draw方法。首先定义一个图像信息变量gHint, 其范围为0~14, 其中0~8代表数字(单元相邻雷数), 9代表未扫(雷)单元, 10代表地雷, 11代表已扫标记单元, 12代表已扫标记叉号, 13代表疑问标记单元, 14代表疑问标记叉号。根据各单元图像信息画出单元图像:

```
public void draw(Graphics g, int x, int y) {
    if (gHint < 0 || gHint > 14)
        return;
    switch (gHint) {
        case 9:
            GraphicsUtil.drawUnknown(g, x, y); //画出未扫单元
            return;
        ...
        case 12:
            GraphicsUtil.drawFlag(g, x, y); //画出已扫单元
            GraphicsUtil.drawCross(g, x, y); //画出叉号
            return;
```



```

case 13:
    GraphicsUtil.drawDoubt(g, x, y);    //画出疑问单元
    return;
...

```

7. GraphicsUtil类完成具体的画图操作，它定义了一系列的画图方法，包括drawUnknown、drawMine、drawFlag、drawDoubt、drawCross、drawNumber等。drawDoubt方法为画出疑问单元的方法：

```

public static void drawDoubt(Graphics g, int x, int y) {
    g.clearRect(x, y, 32, 32);    //清空单元
    g.setColor(colorreg[4]);    //设置旗子颜色
    g.fillRect(x+8, y+8, 16, 10); //画出深蓝色旗子
    g.setColor(Color.black);
    g.drawLine(x+8, y+8, x+8, y+24); //画出黑色旗杆
    g.drawLine(x+9, y+8, x+9, y+24); //画出黑色旗杆
    g.setColor(Color.yellow);    //设置问号颜色
    g.setFont(qnmfont);    //设置问号字体
    FontMetrics fm = g.getFontMetrics();
    String s = "?";
    int sx = (14 - fm.stringWidth(s)) / 2;
    int sy = (10 - fm.getHeight()) / 2 + fm.getAscent();
    g.drawString(s, x+sx+10, y+sy+8); //画出问号
}

```

8. 为ConfigDialog类添加ActionEvent事件处理方法。“设定”对话框中“确定”按钮的事件处理代码如下：

```

if (jRadioButtonCustomized.isSelected()) {    //选中“自定义”单选按钮
    try {
        mine = Integer.parseInt(jTextFieldMine.getText()); //获取雷数
        row = Integer.parseInt(jTextFieldRow.getText()); //获取雷区行数
        column = Integer.parseInt(jTextFieldColumn.getText()); //获取雷区列数
    } catch (NumberFormatException nfe) {
        javax.swing.JOptionPane.showMessageDialog(this, "无效参数!",
            "错误", javax.swing.JOptionPane.ERROR_MESSAGE);
        return;
    }
...

```

程序源代码与解释

```

/* GamePanel.java */
public class GamePanel extends JComponent implements ActionListener, MouseListener {
    public void actionPerformed(ActionEvent event) {    //ActionEvent事件处理
        Object temp = event.getSource();
        try{
            if(temp == newGame||temp == newButton){    //单击“新游戏”菜单或者状态条图
                if(gameActive==false){
                    int m=10,c=8,r=8;
                    setGameParam(m, r, c);
                    setActive(true);

```



```

        repaint();
        frame.setExtendedState(JFrame.NORMAL);
        frame.pack();
        frame.setLocationRelativeTo(null);
        gameActive=true;
    }else if(gameActive==true){
        int m = cdialog.getMine(), r = cdialog.getRow(), c = cdialog.getColumn();
        setGameParam(m, r, c);
        setActive(true);
        repaint();
        frame.setExtendedState(JFrame.NORMAL);
        frame.pack();
        frame.setLocationRelativeTo(null);
    }
    }else if (temp == topHelp) { //单击“帮助”菜单
        JOptionPane.showMessageDialog(panel, scrollHelp, “帮助”, JOptionPane.PLAIN
MESSAGE);
    }else if (temp == aboutGame) { //单击“关于扫雷”菜单
        JOptionPane.showMessageDialog(panel, “Java扫雷1.0 版权所有”, “关于”,
JOptionPane.PLAIN_MESSAGE);
    }else if (temp == exit) { //单击“退出”菜单
        System.exit(0);
    }else { //单击“设定”菜单
        if (cdialog == null) { //创建“设定”对话框
            Component c = this.getParent();
            while (!(c instanceof JFrame))
                c = c.getParent();
            frame = (JFrame) c;
            cdialog = new ConfigDialog(frame, true);
            cdialog.setLocationRelativeTo(frame);
        }
        cdialog.setVisible(true);
        cdialog.dispose();
        if (cdialog.getReturnStatus() == ConfigDialog.RET_OK) { //在“设定”对话框中
            int m = cdialog.getMine(), r = cdialog.getRow(), c = cdialog.getColumn();

            setGameParam(m, r, c); //设置参数
            setActive(true);
            repaint();
            frame.setExtendedState(JFrame.NORMAL);
            frame.pack();
            frame.setLocationRelativeTo(null);
        }
    }
}
}catch (Exception e) {}
}

public void mouseClicked(MouseEvent e) { //MouseEvent鼠标事件处理
    int j = (e.getX() + 2) / 34;
    int i = (e.getY() + 2) / 34;
    FieldCell cell = cells[i][j];
    //如果该单元已经翻开，则返回
    if (cell.getState() == FieldCell.REVEALED)

```

单击“确定”按钮

//获取参数


```

        return;
    if (e.getButton() == MouseEvent.BUTTON1) { //按下鼠标按键1 (左键), 进行翻雷
        //如果已经扫雷 (标上红旗), 则返回
        if (cell.getState() == FieldCell.FLAGGED)
            return;
        if (cell.isMine()) { //如果该单元为地雷, 则翻开后游戏结束 (失败)
            cell.setState(FieldCell.REVEALED);
            setActive(false);
            revealAll();
            gameOver();
        } else {
            cell.setState(FieldCell.UNKNOWN); //先设成未扫单元
            reveal(i, j); //再翻雷
            if (checkVictory()) { //检查是否胜利
                setActive(false);
                gameWin();
            }
        }
    } else if (e.getButton() == MouseEvent.BUTTON3) { //按下鼠标按键3 (右键), 进行扫雷
        if (cell.getState() == FieldCell.UNKNOWN) { //对于未扫单元进行扫雷或疑问标记
            if (flagNum < mineNum) { //如果已扫雷数低于总雷数则进行扫雷
                cell.setState(FieldCell.FLAGGED);
                flagNum++;
                statusLabel2.setText(flagNum+"/"+mineNum);
                if (checkVictory()) { //检查是否胜利
                    setActive(false);
                    gameWin();
                }
            } else { //如果已扫雷数不低于总雷数则进行疑问标记
                cell.setState(FieldCell.DOUBTED);
            }
        } else if (cell.getState() == FieldCell.FLAGGED) { //对于已扫单元进行疑问标记
            cell.setState(FieldCell.DOUBTED);
            flagNum--;
            statusLabel2.setText(flagNum+"/"+mineNum);
        } else if (cell.getState() == FieldCell.DOUBTED) { //对于疑问标记单元则再设置
            cell.setState(FieldCell.UNKNOWN);
        }
    }
    repaint(); //重画
}
//用递归的方法显示其他非地雷的位置
private void reveal(int i, int j) {
    if (i < 0 || i >= r || j < 0 || j >= c)
        return;
    FieldCell cell = cells[i][j];
    if (cell.getState() == FieldCell.REVEALED || cell.getState() == FieldCell.FLAGGED
    || cell.getState() == FieldCell.DOUBTED)
        return;
    cell.setState(FieldCell.REVEALED);
    revealNum++;
    if (cell.getNumber() != 0)
        return;
}

```

或标记疑问

成未扫单元


```

        reveal(i-1, j-1);
        reveal(i-1, j);
        reveal(i-1, j+1);
        reveal(i, j-1);
        reveal(i, j+1);
        reveal(i+1, j-1);
        reveal(i+1, j);
        reveal(i+1, j+1);
    }
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
    public void mousePressed(MouseEvent e) {}
    public void mouseReleased(MouseEvent e) {}
}

```



案例2：黑白棋

案例运行效果与操作

黑白棋

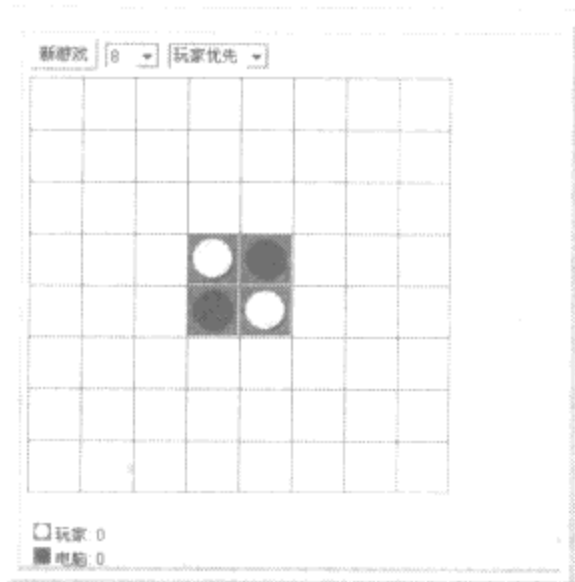


图4-7 运行界面

黑白棋（Othello），也叫苹果棋、翻转棋，是一个经典的策略性游戏。它使用 8×8 的棋盘，由两人分别执黑子和白子下棋，最后子多方为胜方。随着网络的普及，黑白棋作为一种最适合在网上玩的棋类游戏逐渐流行起来。本案例是一个Applet小游戏，实现了黑白棋的功能，而且还可以选择多种棋盘（不限于 8×8 ）以及哪一方先走。程序运行后，界面如图4-7所示。

此时用鼠标单击画面上方中间的下拉列表框可以设置棋盘大小，或者单击右方的下拉列表框设置哪一方先走，界面如图4-8和图4-9所示。

设置完毕，就可以进行游戏了。轮到一方下棋时，必须在与对方棋子相邻的空位放一个己方颜色的棋子，并且要求在放上这个棋子之后，所

下棋子与棋盘上已有的己方棋子之间必须夹住一个或多个对方棋子，然后要把所有被夹住的棋子都翻成自己的颜色。棋子既不能从棋盘上取走，也不能从一个位置移到另一个位置。如果不符合以上规则，则无法走棋，同时会在棋盘下方给出提示信息，界面如图4-10所示。

对局以双方都无棋可下而告终，通常情况下，这时的所有位置都被占满，界面如图4-11所示。此时会在下方显示双方的棋子数（分数）。

制作要点

1. MediaTracker类的应用技巧。

MediaTracker类是一个跟踪多种媒体对象状态的实用工具类。媒体对象可以包括音频剪辑和图像，但目前仅支持图像。



图4-8 选择棋盘大小

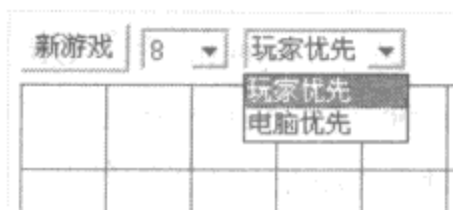


图4-9 设置哪一方先走

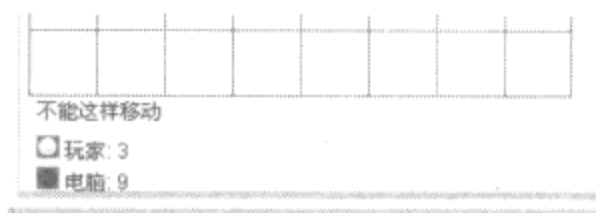


图4-10 无法走棋时给出提示信息

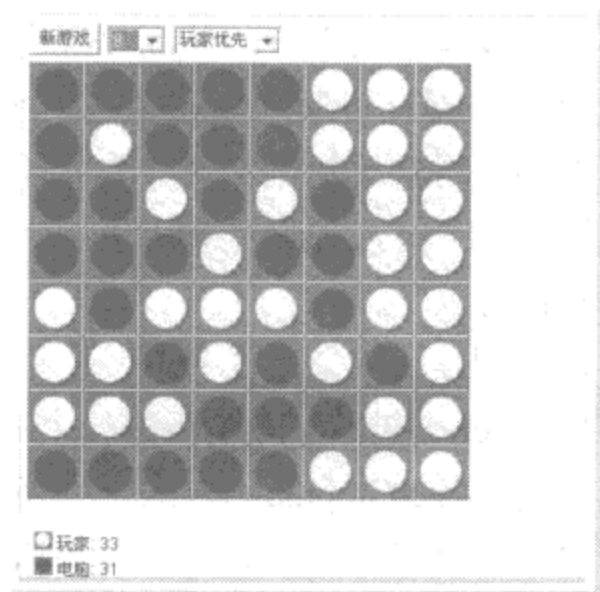


图4-11 游戏结束界面

媒体跟踪器的使用是在调用getImage()方法，且获得对Image的对象引用后。调用跟踪器的addImage()方法就将Image对象传递给媒体跟踪器。addImage方法有两个参数：要跟踪的图像的引用，以及用来跟踪图像的int类型的标识。媒体跟踪器可以同时跟踪多个图像，该标识决定被加载图像的优先级，它还可用于标识可单独等待的唯一图像子集。具有较低ID的图像比具有较高ID的图像优先加载。本案例使用了以下方法：

(1) public void addImage(Image image, int id): 向此媒体跟踪器正在跟踪的图像列表添加一个图像，该图像最终将以它默认的大小（未缩放）呈现。

(2) public boolean checkAll(): 查看此媒体跟踪器正在跟踪的所有图像是否已完成加载，如果图像尚未完成加载，则此方法不能开始加载图像。

(3) public boolean checkID(int id, boolean load): 检查由此媒体跟踪器跟踪且使用指定标识符标记的所有图像是否已完成加载。如果此load标志的值为true，则此方法将开始加载任何尚未加载的图像。

(4) public int statusID(int id, boolean load): 计算或返回由此媒体跟踪器跟踪且具有指定标识符的所有媒体状态的按位或。由MediaTracker类定义的可能标志有LOADING、ABORTED、ERRORED和COMPLETE。尚未开始加载的图像，其状态值为零。如果load值为true，则此方法开始加载任何尚未加载的图像。

(5) public void waitForID(int id) throws InterruptedException: 开始加载由此媒体跟踪器跟踪且具有指定标识符的所有图像。在完成加载具有指定标识符的全部图像之前，此方法一直等待。

2. Image类的使用。

抽象类Image是表示图形图像的所有类的超类，必须以特定于平台的方式获取图像。在实际编程中，往往通过工具类的对象工厂getImage方法来获取实际的Image对象。本案例使用

Applet类的getImage(URL url, String name)方法来获取图像，其中又使用了getDocumentBase()方法来获取嵌入此Applet文档的URL对象。利用getImage方法将图像读取到Java程序中以后，可以使用Graphics类的drawImage方法绘制该图像。

3. synchronized关键字的使用。

Java语言内置了synchronized关键字，用于对多线程进行同步，能够对方法或者代码块进行同步。其原理是，当两个并发线程访问同一个对象时，一个时间内只能有一个线程得到执行，另一个线程必须等待，从而保证了线程安全。

4. 使用双缓冲机制（Double buffering）消除闪烁。

本案例采用双缓冲机制来解决屏幕的闪烁现象，简单的说就是在显示我们想要的图画之前，把所有的图画先在后台绘制好并存放到相应的图像变量中去。当需要显示时直接复制到前台屏幕就可以了。

具体实现：

- (1) 用createImage方法新建一个后台图像类变量。
- (2) 使用getGraphics()方法得到当前图像的图形关联。
- (3) 在后台处理所有相关的工作，如清除屏幕，后台绘画等。

当完成所有的后台工作后，复制已经绘制好的图像到前台，并覆盖前台的存在图像。这样所有的操作都是在后台进行，在屏幕显示新的图像前，这些内容都已经存在于后台了，所以在任何时刻都看不到空屏幕的存在，即代表闪烁消除了。

步骤详解

1. 绘制图像：

```
whiteImage = getImage(getDocumentBase(), "pic2.gif");    //白棋子
blackImage = getImage(getDocumentBase(), "pic1.gif");    //黑棋子
offScrGraphics.drawImage(whiteImage, 0, 0, this);       //画出白棋子
offScrGraphics.drawImage(blackImage, 0, 0, this);       //画出黑棋子
```

2. 向媒体跟踪器添加图像，并检查是否完成加载，图像处理是编程中经常遇到的问题，图像使用的支持文件分布于java.applet、java.awt和java.awt.image包中。每一个图像都用一个java.awt.Image对象表示。除了Image类外，java.awt包提供了其他的基本图像支持，但如果需要更好地图像跟踪和控制，要使用MediaTracker：

```
imageTracker = new MediaTracker(this);
imageTracker.addImage(whiteImage, 0);
imageTracker.addImage(blackImage, 1);
for(int i = 0; i < 2; i++)
try {
    imageTracker.checkID(i, true);
    imageTracker.waitForID(i);
    showStatus("Loading image P" + i + ".gif");
}
catch(Exception exception1) {
    System.out.println(exception1);
}
```


在J2SE 5.0之前的版本中, Java只支持三种格式的图像, 分别是JPG、GIF、PNG, 之后Java将BMP格式文件纳入支持的范围。在处理单色图像和真彩色图像的时候, 无论图像数据多么庞大, BMP文件都不对图像数据进行任何压缩处理, 这就使得它在存储单色图像或者真彩色图像时, 文件体积过于庞大, 所以在网络上一般不采用BMP格式的文件来进行传送。

3. 消除闪烁:

```
offScrImage = createImage(400, 400); //创建一幅用于双缓冲的、可在屏幕外绘制的图像
offScrGraphics = offScrImage.getGraphics(); //获取Graphics类对象
```

使用双缓冲机制消除闪烁, 在前面的案例中已经涉及到这个问题, 这里不再赘述。

4. 创建线程并启动:

```
public void start() { //在init方法后调用
    aThread = new Thread(this); //创建线程对象
    aThread.start(); //启动线程的run方法
    imageTracker.statusID(0, true); //开始加载白棋子 (如果尚未加载的话)
    imageTracker.statusID(1, true); //开始加载黑棋子 (如果尚未加载的话)
}
```

5. 线程互斥:

```
synchronized void newGame() {}
public synchronized void blackMove() {}
public synchronized void paint(Graphics g) {}
public synchronized void init() {}
```

程序源代码与解释

```
/** Othello.java*/
public class Othello extends Applet implements Runnable {
    public synchronized void init() { //初始化
        aThread = null; //创建线程对象
        imageTracker = new MediaTracker(this); //创建图像跟踪器对象
        offScrImage = createImage(400, 400); //创建一幅用于双缓冲的、可在屏幕外绘制的
        //图像

        offScrGraphics = offScrImage.getGraphics(); //获取Graphics类对象
        f = new Font("TimesRoman", 0, 12); //定义字体
        offScrGraphics.setFont(f); //设置字体
        fontHeight = getFontMetrics(f).getHeight(); //获取字体高度
        fontAscent = getFontMetrics(f).getAscent(); //获取字体基线到字符顶部的距离
        loadImages(); //加载图像
        if(button == null) {
            //...
        }
        if(moveFirst == null) {
            moveFirst = new Choice(); //创建弹出式选择菜单对象
            //选择菜单对象, 添加2个菜单项
            moveFirst.addItem("玩家优先");
            moveFirst.addItem("电脑优先");
            add(moveFirst); //选择菜单对象, 添加到游戏窗口
            moveFirst.select(0); //设置默认菜单项
            whiteMoveFirst = true; //白棋可以移动
        }
    }
}
```



```

        newGame();    //开始新游戏
    }
    public synchronized void loadImages() {
        try {
            whiteImage = getImage(getDocumentBase(), "pic2.gif");    //白棋子
            blackImage = getImage(getDocumentBase(), "pic1.gif");    //黑棋子
            offScrGraphics.drawImage(whiteImage, 0, 0, this);    //画出白棋子
            offScrGraphics.drawImage(blackImage, 0, 0, this);    //画出黑棋子
        }
        catch(Exception evt) {
            System.out.println(evt);
        }
        imageTracker.addImage(whiteImage, 0);    //图像跟踪器添加要进行跟踪的图像
        imageTracker.addImage(blackImage, 1);    //图像跟踪器添加要进行跟踪的图像
        for(int i = 0; i < 2; i++)
            try {
                imageTracker.checkID(i, true);    //检查图像是否已完成加载并开始加载尚未加
                imageTracker.waitForID(i);    //开始加载图像并一直等待加载完毕
                showStatus("Loading image Pic" + i + ".gif");    //显示状态信息
            }
            catch(Exception evt) {    //waitForID方法需要捕获异常
                System.out.println(evt);
            }
    }
    public boolean action(Event event, Object obj) { //事件处理
        if(event.target == button) {    //按钮事件
            newGame();    //开始新游戏
            return true;
        }
        if(event.target == choice) {    //选择菜单事件，选择棋盘大小
            Integer integer = new Integer((String)obj);    //获取选择的棋盘大小值
            DIM = integer.intValue();
            newGame();    //开始新游戏
            return true;
        }
        if(event.target == moveFirst) {    //选择菜单事件，选择行棋顺序
            if((String)obj == "电脑优先")
                whiteMoveFirst = false;
            else
                whiteMoveFirst = true;
            return true;
        }
        else {
            return super.action(event, obj);    //调用父类方法处理事件
        }
    }
    //参见步骤详解4

    public void run() {    //线程的运行方法
        Thread thisThread = Thread.currentThread();
        do
            try {

```

载的图像


```
Thread.sleep(50L); //休眠50毫秒
    }
    catch(Exception evt) { }
    while(aThread == thisThread);
}
public void stop() { //线程的停止方法
    aThread = null; }
public void destroy() { }
public Othello() { //构造方法
    boardSize = 400; //设置棋盘大小
    grid = new int[400]; //创建棋盘数组对象
}
}
```



案例3：象棋游戏

案例运行效果与操作

本案例用Java实现了一个简单的中国象棋游戏，运行初始界面如图4-12所示。

游戏时先单击准备走的棋子，然后单击要走到的位置，即可进行走棋或吃棋。必须按照中国象棋的规则进行走棋，否则程序不予响应。如果一方胜利，会弹出提示对话框，如图4-13所示，此时单击“确定”按钮，系统会返回如图4-12所示的初始界面。另外，在游戏过程中单击界面下方的“重新开始”按钮，也会返回如图4-12所示的初始界面。

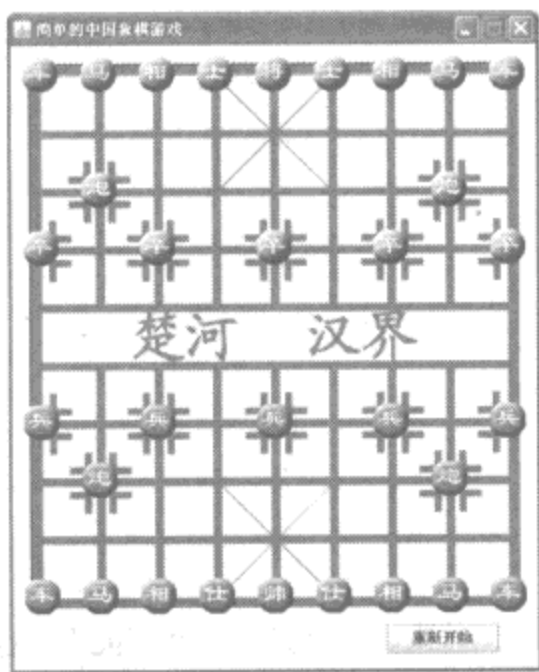


图4-12 初始界面



图4-13 胜利界面

制作要点

1. ImageIcon类的应用技巧。

ImageIcon类是Icon接口的具体实现，它根据Image绘制Icon。可使用MediaTracker预载根据URL、文件名或字节数组创建的图像，以监视该图像的加载状态。ImageIcon可以显示动态图像，并且支持同步图像加载（即在返回之前完全下载图像），这些使得ImageIcon具有非常强大的功能。Image类是一个抽象类（abstract class），所以Applet不能直接创建Image类的对象。但ImageIcon类不是抽象类，它提供了多个构造器，允许程序使用本地计算机上的图像，

或者是存储在Internet中的图像来初始化ImageIcon对象。

```
Icon image=new ImageIcon("../src\\framechess\\images\\Chess.jpg");
```

2. 鼠标事件处理。

3. 程序设计。

本案例由两个类构成，其中FrameChess类是程序的主类，包括所有的初始化过程以及游戏过程中的控制逻辑。它包括以下几个主要方法：myframeInit方法用来进行初始化，redraw方法重画棋子和棋盘，moveChess方法控制走棋，repaintChess方法完成下棋过程中的界面重画以及胜利判断等，restart方法则完成重新开始游戏的工作。另外，还有一些事件处理方法。

第二个类是CenterFrame类，用来对中调整程序窗口和对话框，以改善游戏的显示。

步骤详解

1. 定义数据类型：由于象棋是一个二维图形游戏，所以本案例使用15个整型变量来表示不同的棋子：

```
//定义棋子的整型变量，其中红棋为奇数，黑棋为非0的偶数，空位置为0
private static byte x王=1; //红王
private static byte y王=2; //黑王
```

2. 使用一个二维整型数组来保存象棋的棋盘和棋子位置信息，下列定义是棋盘的初始化：

```
byte cChess[][]={
    {y车,y马,y象,y士,y王,y士,y象,y马,y车}, //1
    {空,空,空,空,空,空,空,空,空}, //2
    {空,y炮,空,空,空,空,空,y炮,空}, //3
    {y兵,空,y兵,空,y兵,空,y兵,空,y兵}, //4
    {空,空,空,空,空,空,空,空,空}, //5
    //-----楚河-----汉界-----
    {空,空,空,空,空,空,空,空,空}, //6
    {x兵,空,x兵,空,x兵,空,x兵,空,x兵}, //7
    {空,x炮,空,空,空,空,空,x炮,空}, //8
    {空,空,空,空,空,空,空,空,空}, //9
    {x车,x马,x象,x士,x王,x士,x象,x马,x车}, //10
};
```

3. 在编程中，图标或者其他的图像文件经常需要显示出来，通过Toolkit的getImage()方法可以完成，它解析了图像文件并返回一个Image对象，如：

```
Toolkit toolkit = Toolkit.getDefaultToolkit();
Image image = toolkit.getImage(fileName);
```

事实上这段代码并不真正加载图像，而是在另一个线程中加载。如果在加载完成前显示图像，只有部分（或者完全没有）图像会被显示。解决的方法是使用MediaTracker对象监视图像，等待加载结束，正如案例2中所讲的一样。有一个更简单的办法，ImageIcon类在加载图像时自动使用MediaTracker。javax.swing包中的ImageIcon类使用Toolkit加载图像，直到加载完毕它才返回。


```
ImageIcon icon = new ImageIcon(fileName);
Image image = icon.getImage();
```

本例中:

```
icon = new ImageIcon("../src\\framechess\\images\\帅2.gif");
```

重画棋子和棋盘, 比如画出黑棋中的“相”:

```
if (cChess[x][y] == y象) {
    icon = new ImageIcon("../src\\framechess\\images\\相1.gif");
    play[arr] = new JLabel(icon);
    play[arr].setBounds(y * 50 + 10, x * 50 + 10, chessW, chessH);
    this.add(play[arr]);
    ++arr;
}
```

而且还要重画一次棋盘, 不然的话棋子就会在棋盘的底层(即棋盘挡住了棋子, 看得见棋盘, 看不见棋子):

```
chessBoard = new JLabel(image);
chessBoard.setBounds(new Rectangle(0, 0, 450, 500));
chessBoard.addMouseListener(this);
this.getContentPane().add(chessBoard, null); //棋盘标签添加到主窗口
this.getContentPane().add(message, null); //“消息”对话框添加到主窗口
this.getContentPane().add(restart, null); //“重新开始”按钮添加到主窗口
chessBoard.setVisible(true); //显示棋盘
```

4. 棋子走动关键的一点是设定走棋逻辑, 对于同样的棋子来说, 黑棋和红棋的规则是一样的。

首先判断是否可以走棋, 分为三种情况:

```
if ( (cChess[oldx][oldy] % 2 == 1 && cChess[newx][newy] % 2 == 0) || //红棋吃黑棋
      (cChess[oldx][oldy] % 2 == 0 && cChess[oldx][oldy] != 空 && (cChess[newx][newy] % 2
== 1 || //黑棋吃红棋
      cChess[newx][newy] == 空))) { //走棋到空位
```

例如“车”的走法:

```
if (cChess[oldx][oldy] == x车 || cChess[oldx][oldy] == y车) {
    if ( (newx - oldx == 0 && newy - oldy != 0) || (newx - oldx != 0 && newy - oldy ==
0)) {
        //横走棋
        if (newx - oldx == 0 && newy - oldy != 0) {
            //左走棋
            if (oldy > newy) {
                o = oldy - newy;
                //查看是否有棋子在中间挡住, 有的话就不能移动
                for (int s = 1; s < o; s++) {
                    if (! (cChess[oldx][--y1] == 空)) {
                        return;
                    }
                } //End for
            }
            //右走棋
```



```

...
        //竖走棋
    else {
        //向上走棋
        if (oldx > newx) {
            o = oldx - newx ;
            for (int s = 1 ; s < o ; s++) {
                //查看是否有棋子在中间挡住，有的话就不能移动
                if (! (cChess[--x1][oldy] == '空')) {
                    return ;
                }
            } //End for
        }
        //向下走棋
    }
...
    -   repaintChess() ; //重绘
    }

```

如果“炮”要吃棋：

```

        //如果要吃棋，查看中间是否有一棋子，如果有就可以吃棋
        //没有棋子的话就不能吃棋....
        else if (cChess[newx][newy] != '空') {
            k = 0 ;
            //向上吃棋
            if (oldx > newx && oldy == newy) {
                o = oldx - newx ;
                for (int s = 1 ; s < o ; s++) {
                    if (cChess[--x1][oldy] != '空') {
                        ++k ;
                    }
                }
                //如果中间没有一个棋子就不可以吃棋
                if (k != 1) {
                    return ;
                }
                if (oldy != newy) {
                    return ;
                }
            }
            ... //向下吃棋

            //左吃棋
            else if (oldy > newy && oldx == newx) {
                o = oldy - newy ;
                for (int s = 1 ; s < o ; s++) {
                    if (cChess[oldx][--y1] != '空') {
                        ++k ;
                    }
                }
                //如果中间没有一个棋子就不可以吃棋
                if (k != 1) {
                    return ;
                }
                if (oldx != newx) {

```



```

        return ;
    }
}
//右吃棋
...
repaintChess() ;
}
}

```

5. 窗体对中处理方法:

```

public void setFrame() { //窗体对中的具体操作
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize(); //获取屏幕尺寸
    Dimension frameSize = myFrame.getSize(); //获取窗体尺寸
    if (frameSize.height > screenSize.height)
        frameSize.height = screenSize.height; //窗体尺寸不能超过屏幕尺寸
    if (frameSize.width > screenSize.width)
        frameSize.width = screenSize.width; //窗体尺寸不能超过屏幕尺寸
    myFrame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height -
frameSize.height) / 2); //对中
}

```

6. 为FrameChess类添加repaintChess方法, 清零后重新根据二维数组重画:

```

public void repaintChess() {
    //如果是红棋走
    if (cChess[oldx][oldy] % 2 == 1) {
        ++xgo ;
    }
    //如果黑棋走
    else if (cChess[oldx][oldy] % 2 == 0 && cChess[oldx][oldy] != 空) {
        ++ygo ;
    }
    else {
        message.showMessageDialog(this, "程序出现错误!!!", "简单的中国象棋游戏",
            message.ERROR_MESSAGE) ;

        System.exit(0) ;
    }
    //如果红棋与黑棋走的步数一样, 就该红棋走, 否则该黑棋走
    if (xgo == ygo) {
        if (cChess[oldx][oldy] % 2 == 1) {
            System.out.println("应该红棋走棋!!!") ;
            return ;
        }
    }
}
... }

```

程序源代码与解释

```

/* * FrameChess.java*/
public class FrameChess extends JFrame implements MouseListener {
    byte Q1[][]=new byte[10][9]; //用来保存棋盘 二维数组初始值, 以便游戏重新开始
    Icon image=new ImageIcon("../src\\framechess\\images\\Chess.jpg");
    private JLabel chessBoard = new JLabel(image);
    private JButton restart = new JButton();
}

```



```

private JOptionPane message = new JOptionPane();
public FrameChess() {    //构造方法
    try {
        myframeInit();    //调用myframeInit方法
    } catch(Exception evt) {
        evt.printStackTrace();
    }
}
void myframeInit() throws Exception {    //主窗口初始化
    for (int i1 = 0; i1 < 10; i1++) {
        for (int i2 = 0; i2 < 9; i2++) {
            Q1[i1][i2]=cChess[i1][i2];    //保存棋盘二维数组初始值
        }
    }
    chessBoard.setBounds(new Rectangle(0, 0, 450, 500));    //设置棋盘位置和大小
    this.getContentPane().setBackground(Color.white);    //设置主窗口背景为白色
    this.setResizable(false); //主窗口不能改变大小
    this.setTitle("简单的中国象棋游戏"); //设置标题
    this.addWindowListener(new java.awt.event.WindowAdapter() { //主窗口添加关闭事件侦听器
        public void windowClosing(WindowEvent e) {
            this.windowClosing(e);
        }
    });
    this.getContentPane().setLayout(null);    //设置布局
    arr=0;
    redraw();    //初始化棋盘
    restart.setBounds(new Rectangle(320, 500, 96, 25));    //设置“重新开始”按钮位置和大小
    restart.setText("重新开始");
    restart.addMouseListener(new java.awt.event.MouseAdapter() { //为“重新开始”按钮添加单击事件侦听器
        public void mouseClicked(MouseEvent e) {
            restart_mouseClicked(e);
        }
    });
    this.getContentPane().add(restart, null);    //“重新开始”按钮添加到主窗口
    this.setSize(456,570);    //设置主窗口大小
    new CenterFrame(this).setFrame();    //主窗口对中
    this.setVisible(true);    //显示主窗口
}
//主程序
public static void main(String[] args) {
    FrameChess FrameChess = new FrameChess();
}
void this_windowClosing(WindowEvent e) {
    System.exit(0);
}
//End主程序
//鼠标按键在组件上单击（按下并释放）时调用
public void mouseClicked(MouseEvent e) {}
//鼠标进入到组件上时调用
public void mouseEntered(MouseEvent e) {}
//鼠标离开组件时调用
public void mouseExited(MouseEvent e) {}
//鼠标按键在组件上按下时调用
public void mousePressed(MouseEvent e) {

```



```

        if(e.getButton()==e.BUTTON1) {
            //把坐标换成数组的下标
            py=e.getX()/g;
            px=e.getY()/g;
            if(!go) {                //单击要走的棋子，确定该棋子走棋
                oldx=px;oldy=py;
                if(cChess[oldx][oldy]!='空') {
                    go=true;        //下一步就可以走棋了
                }
            } else {                //再单击要走到的位置，进行走棋
                newx=px;newy=py;
                go=false;          //这一步走完，下一步就不可以走棋了
                moveChess(); //走棋
            }
        } //End if
    } //End mousePressed
//鼠标按钮在组件上释放时调用
public void mouseReleased(MouseEvent e) {}
public void moveChess() {
    /*所有棋子的移动判断*/
    //代码略
    ...
}
//清零后根据二维数组重画
public void repaintChess() {
    ...
//重画棋盘和棋子，代码略}
void restart_mouseClicked(MouseEvent e) {                //“重新开始”按钮鼠标单击事件处理
    restart();        //调用restart方法重新初始化
}
public void restart() {                //重新开始游戏
    for (int i1 = 0; i1 < 10; i1++) {
        for (int i2 = 0; i2 < 9; i2++) {
            cChess[i1][i2]=Q1[i1][i2];        //棋盘的二维数组重置初始值
        }
    }
    xgo=0;ygo=0;                //走棋步数置初始值
    arr = 0 ;
    for (int i = 0 ; i < 32 ; i++)
        play[i].setVisible(false) ;
    chessBoard.setVisible(false) ;
    redraw();                //重画棋子和棋盘
    this.setVisible(false); //刷新显示
    this.setVisible(true); //刷新显示
}
}
}

```



案例4：一个简单的弹球游戏

案例运行效果与操作

本案例实现了一个简单的弹球游戏，利用线程技术来控制弹球的速度。程序运行后，界面如图4-14所示。

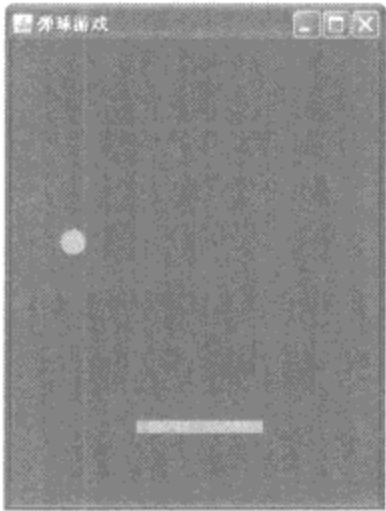


图4-14 运行界面

在第二次接住弹球后，屏幕上会出现4个弹球，界面如图4-15所示。

随着游戏的进行，弹球速度会变快两次。当屏幕上所有的弹球都未接住后，游戏结束，给出提示信息和成绩，界面如图4-16所示。

制作要点

1. Runnable接口的应用技巧。

创建新执行线程有两种方法：一种方法是将类声明为Thread的子类，该子类应重写Thread类的run方法，接下来可以分配并启动该子类的实例；创建线程的另一种方法是声明实现Runnable接口的类，该类然后实现run方法，可以分配该类的实例，在创建Thread时作为一个参数来传递并启动。本案例使用第二种方法。

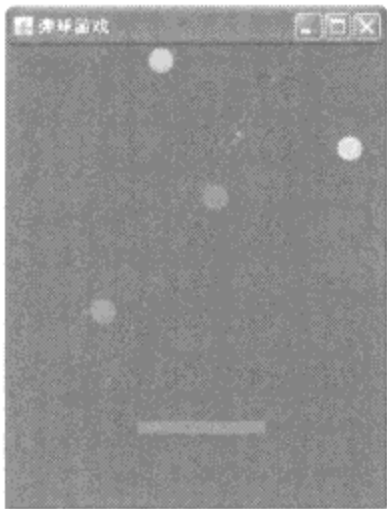


图4-15 第二次接住弹球后的界面



图4-16 游戏结束界面

- 2. KeyEvent的应用技巧。
- 3. 程序设计。

本案例由三个类构成，第一个是BallClass类，用来对弹球对象进行初始化。
第二个类是BoardClass类，是对弹球板对象进行初始化的类，并且封装了弹球板对象在屏幕上移动的逻辑。
最后一个类是MyBall类，是本程序的主类，它负责对整个程序界面的初始化，控制整个程序的流向，提供各种方法对不同的情况进行处理。

步骤详解

1. 创建窗体JFrame:

```
myFrame = new JFrame()
```

2. 随机的弹球位置和颜色，利用random方法产生随机数:

```
vx = -(int)(Math.random()*10 + 5);           //随机步进值
vy = -(int)(Math.random()*10 + 5);           //随机步进值
color = new Color((int)(Math.random()*255),(int)(Math.random()*255),(int)(Math.random()*255));
//随机颜色
```


random()方法的使用参见第1章案例8。

3. 添加关闭窗口的适配器:

```
myFrame.addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent e){
        if (newThread!=null){
            newThread=null;
        }
        myFrame.dispose();
    }
});
```

WindowAdapter()是接收窗口事件的抽象适配器类,它存在的目的是方便创建侦听器对象。WindowAdapter类实现了WindowListener接口,并对接口中的所有方法提供了默认的空方法实现。WindowAdapter类在这里称为WindowListener接口的存根(stub)——存根就是用空方法体实现该接口中所有方法的实现。

WindowEvent()指示窗口状态改变的低级别事件。当打开、关闭、激活、停用、图标化或取消图标化Window对象时,或者焦点转移到Window内或移出Window时,由Window对象生成此低级别事件,该事件被传递给每一个使用窗口的addWindowListener方法注册以接收这种事件的WindowListener或WindowAdapter对象(WindowAdapter对象实现WindowListener接口)。发生事件时,所有此类侦听器对象都将获取此WindowEvent。

4. 各种交互式程序中都会用到鼠标和键盘,下面简单总结一下鼠标事件和键盘事件的处理方法。

(1) 鼠标事件

- 使用MouseListener接口处理鼠标事件。使用MouseListener接口可以处理5种操作发生的鼠标事件:

- 1) 在事件源上按下鼠标键mousePressed(MouseEvent)。
- 2) 在事件源上释放鼠标键mouseReleased(MouseEvent)。
- 3) 在事件源上单击鼠标键mouseClicked(MouseEvent)。
- 4) 鼠标进入事件源mouseEntered(MouseEvent)。
- 5) 鼠标退出事件源mouseExited(MouseEvent)。

鼠标事件的类型是MouseEvent,即当发生鼠标事件时,MouseEvent类自动创建一个事件对象。事件源获得监视器的方法是addMouseListener(监视器)。MouseEvent类中有以下几个重要的方法:

- 1) getX(): 获取鼠标在事件源坐标系中的x坐标。
 - 2) getY(): 获取鼠标在事件源坐标系中的y坐标。
 - 3) getModifiers(): 获取鼠标的左键或右键。鼠标的左键和右键分别使用InputEvent类中的常量BUTTON1_MASK和BUTTON3_MASK来表示。
 - 4) getClickCount(): 获取鼠标被单击的次数。
 - 5) getSource(): 获取发生鼠标事件的事件源。
- 使用MouseMotionListener接口处理鼠标事件。使用MouseMotionListener接口可以处理两种操作发生的鼠标事件:

- 1) 在事件源上拖动鼠标mouseDragged(MouseEvent)。
- 2) 在事件源上移动鼠标mouseMoved(MouseEvent)。

鼠标事件的类型是MouseEvent，即当发生鼠标事件时，MouseEvent类自动创建一个事件对象。事件源获得监视器的方法是addMouseMotionListener（监视器）。

（2）键盘事件

当按下、释放或敲击键盘上一个键时就发生了键盘事件，在Java 1.2事件模式中，必须要有发生事件的事件源。当一个组件处于激活状态时，敲击键盘上一个键就导致这个组件上发生了键盘事件。事件源使用addKeyListener（监视器）方法获得监视器。监视器是一个对象，创建该对象的类必须实现接口KeyListener。接口KeyListener中有3个方法：

- 1) public void keyPressed(KeyEvent e)。
- 2) public void keyTyped(KeyEvent e)。
- 3) public void KeyReleased(KeyEvent e)。

当按下键盘上某个键时，监视器就会侦听到，然后keyPressed方法会自动执行，并且KeyEvet类自动创建一个对象传递给方法keyPressed中的参数e。方法keyTyped是keyPressed和keyReleased方法的组合，当键被按下又释放时，keyTyped方法被调用。用KeyEvent类的public int getKeyCode()方法，可以判断哪个键被按下、敲击或释放，getKeyCode方法返回一个键码值。也可以用KeyEvent类的public char getKeyChar()判断哪个键被按下、敲击或释放，getKeyChar方法返回键的字符。

本案例中的键盘响应事件：

```
public void keyTyped(KeyEvent e){}
public void keyPressed(KeyEvent e){
    key=e.getKeyCode();
    if(key==KeyEvent.VK_LEFT){ //按下左方向键
        bd.leftmove(mv);
    }
    if(key==KeyEvent.VK_RIGHT){ //按下右方向键
        bd.rightmove(mv);
    }
}
```

程序源代码与解释

```
/* * MyBall.java*/
public class MyBall extends Panel implements Runnable,KeyListener{ //主界面类，实现了Runnable
接口
    public MyBall(){ //构造函数
        time = 50;
        b = new BallClass[cont]; //创建具有一个弹球对象的数组
        for(int i=0;i<cont;i++){
            b[i] = new BallClass(100,10); //初始化弹球对象
        }
        bd = new BoardClass(100,300); //初始化弹球板对象
        this.addKeyListener(this);
        newThread = new Thread(this); //创建线程
        newThread.start(); //启动线程，调用run方法
    }
    public void paint(Graphics g){
```



```

        this.requestFocus();
        for(int i=0;i<cont;i++){ //画出所有的弹球
            g.setColor(b[i].color);
            g.fillOval(b[i].ballX,b[i].ballY,20,20);
        }
        g.fillRect(bd.boardX,bd.boardY,bd.width,bd.height); //画出弹球板
        if(over == 0){ //游戏结束时
            Font f = new Font("Courier",Font.BOLD,28);
            g.setFont(f);
            g.setColor(Color.orange);
            g.drawString("GAME OVER",50,150); //给出提示信息
            Font f1 = new Font("Courier",Font.BOLD,16);
            g.setFont(f1);
            g.drawString("你一共接住了 "+total+" 个球。",55,180); //给出成绩信
            //给出成绩信

            newThread = null;
            control = false;
            System.gc();
        }
    }

    public static void main(String[] args){ //主方法，对系统初始化，显示界面
        myFrame = new JFrame();
        MyBall bl = new MyBall();
        bl.setBackground(Color.black); //设置窗口背景颜色
        bl.setSize(300,400); //设置窗口大小
        myFrame.setSize(300,400);
        myFrame.setLocation(300,100);
        myFrame.setTitle("弹球游戏");
        //添加关闭窗口的适配器
        myFrame.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                if (newThread!=null){
                    newThread=null;
                }
                myFrame.dispose();
            }
        });
        myFrame.add(bl);
        myFrame.setVisible(true); //显示主窗口
    }

    //线程启动后执行
    public void run(){
        while(control){
            repaint();
            for(int i=0;i<cont;i++){
                if(b[i].ballX > 280||b[i].ballX < 0){ //弹球碰到左右边界时
                    b[i].vx = -b[i].vx; //向相反方向移动
                }
                if(b[i].ballY < 0){ //弹球碰到上边界时
                    b[i].vy = -b[i].vy; //向相反方向移动
                }
                if(b[i].ballY > 400){ //弹球未接住落下时
                    b[i].state = false; //设为隐藏
                }
            }
        }
    }
}

```



```

    }
    //弹球碰到弹球板时
    if((Math.abs(b[i].ballY-bd.boardY)<=20)&&
        ((b[i].ballX+20>=bd.boardX)&&(b[i].ballX<=bd.boardX+bd.width)))){
        b[i].vy = -b[i].vy; //向相反方向移动
        total++;           //接球数加1
        switch(total){     //根据接球数设置弹球及弹球板速度
        case 3:             //在接到第3个球时
            time = 100;
            break;
        case 5: //在接到第5个球时
            time = 50;
            mv = 40;
            break;
        case 20: //在接到第20个球时
            time = 20;
            mv = 50;
            break;
        default:break;
        }
        if(total == 2){ //在接到第2个球时
            cont = 4;
            x = b[0].ballX;
            y = b[0].ballY-20;
            b = new BallClass[cont];
            for(int j=0;j<cont;j++){ //产生4 个弹球
                b[j] = new BallClass(x,y);
            }
        }
        b[i].ballX+=b[i].vx;
        b[i].ballY+=b[i].vy;
    }
    over = 0;
    for(int i=0;i<cont;i++){
        if(b[i].state){
            over = 1;    //只要有弹球，游戏就不结束
            break;
        }
    }
}

try{
    Thread.sleep(time);    //线程睡眠的时间
}catch(InterruptedException e){}
}

}

public void keyTyped(KeyEvent e){}
public void keyPressed(KeyEvent e){    //键盘事件响应，用于控制弹球板的移动
    int key;
    key=e.getKeyCode();
    if(key==KeyEvent.VK_LEFT){ //按下左方向键
        bd.leftmove(mv);
    }
    if(key==KeyEvent.VK_RIGHT){ //按下右方向键

```



```
bd.rightmove(mv);  
    }  
}  
public void keyReleased(KeyEvent e){}  
}
```



案例5：找 不 同

案例运行效果与操作

本案例简单实现了常见的找不同游戏的功能。程序运行后，游戏主界面如图4-17所示。



图4-17 运行时界面

单击“开始游戏”按钮后出现游戏界面，如图4-18所示。界面主体是两幅基本一样的图片，图片中有5处不同点，玩家的任务就是将这5个不同点找出。找出不同点后在任意一幅图片相应位置单击鼠标，就会在该不同位置画出一个绿色圆圈。界面上方是一些游戏信息，中间是一列在轨道上不停左移的火车，用来完成对游戏的时间控制，如果火车到最左端仍然未找出所有不同则游戏结束。左上角蘑菇右侧的数值代表当前关卡数，右上角苹果右侧的数值代表玩家当前得分，单击左下方的放大镜可以直接找到一个不同，但不得分，最多只能用5次。单击沙漏则可以使火车复位到最右端重新开始左移，最多只能用2次。右下方的火车图标代表玩家当前的生命值，如果未找对不同（玩家在非不同位置单击鼠标一次），生命值减1。如果生命值为0则游戏结束。

如果生命值为0或者火车到轨道最左端仍未找出所有不同，则游戏结束，界面如图4-19所示。

如果完成所有关卡，会出现游戏胜利界面，如图4-20所示。

当游戏结束或者已经完成所有关卡时，如果玩家成绩能够进入玩家英雄排行榜的前10名，则会弹出输入姓名窗口，如图4-21所示。此时玩家可输入自己的姓名（限制在3个英文字母以内，也可不输），单击“OK”按钮返回并将成绩写入排行榜。

在如图4-17所示的主界面单击“英雄榜”按钮，会弹出“排行榜”窗口，如图4-22所示。单击“返回”按钮可回到主界面。

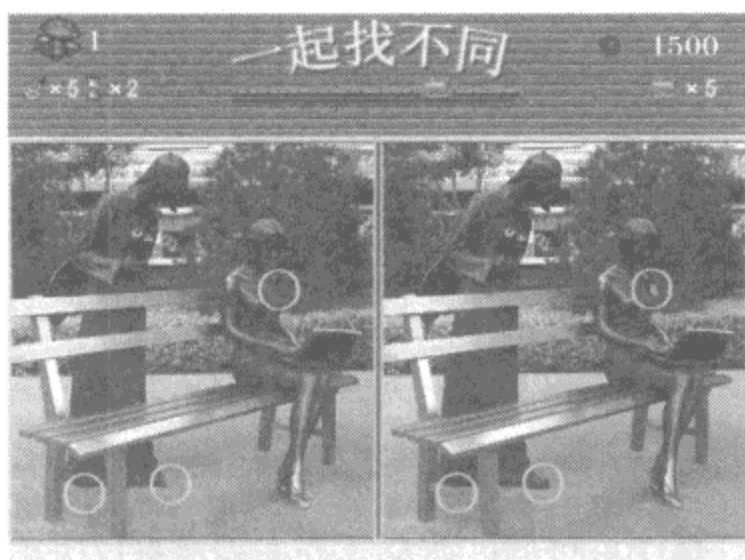


图4-18 游戏界面



图4-19 游戏结束界面

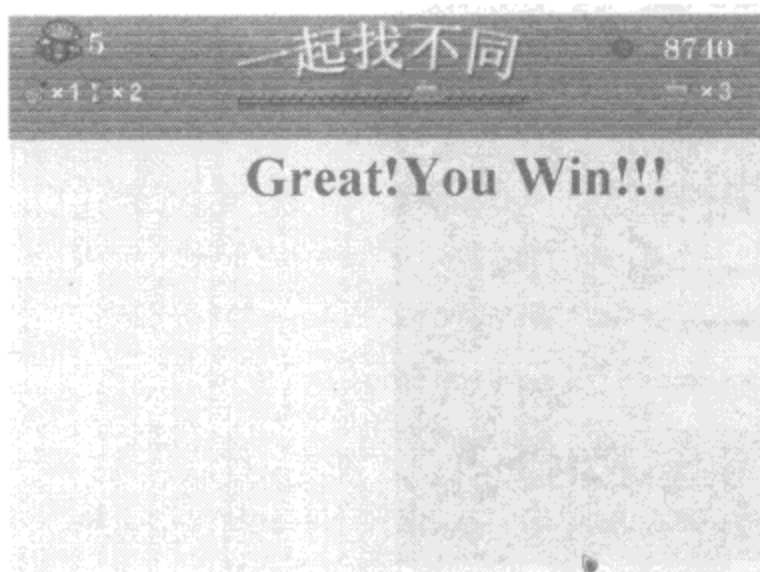


图4-20 游戏胜利界面

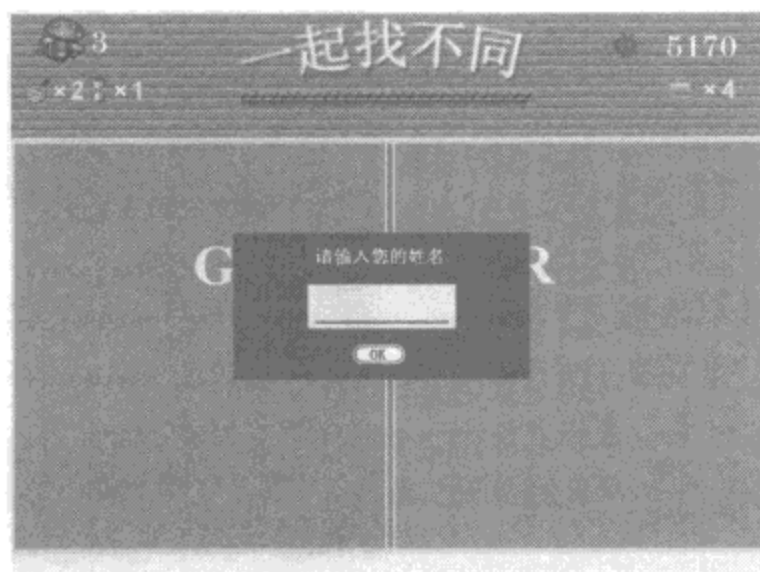


图4-21 输入姓名界面

在主界面单击“游戏设置”按钮，会弹出“游戏设置”窗口，如图4-23所示。此时可以设置游戏的音效、背景音乐、游戏难度等。单击“返回”按钮可回到主界面。

在主界面单击“退出游戏”按钮，可退出游戏。



图4-22 “排行榜”窗口



图4-23 “游戏设置”窗口

制作要点

1. 在JLabel中使用图像。

JLabel主要用于短文本字符串或图像的显示。JLabel对象可以显示文本、图像或同时显示

二者。在JLabel中使用图像，既可以使用其构造方法来创建，也可以使用setIcon方法，具体介绍如下。

- **public JLabel(Icon image):** 创建具有指定图像的实例。该标签在其显示区内垂直和水平居中对齐。
- **public JLabel(String text, Icon icon, int horizontalAlignment):** 创建具有指定文本、图像和水平对齐方式的实例。该标签在其显示区内垂直居中对齐。文本位于图像的结尾边上。
- **public void setIcon(Icon icon):** 定义此组件将要显示的图标。

2. Java游戏编程中声音处理的应用技巧。

Java中的AudioClip接口提供了对声音或者音乐的简单播放控制，Clip接口表示一个特殊的通道，它可以在播放之前先装入音频数据。由于数据是预装入的，不是流入的，所以Clip可以支持持续查询、循环播放以及重新定位。它有play、stop、loop等几个方法，因为它是接口，所以不能直接创建对象。但是，Applet类的新AudioClip和getAudioClip方法均可以创建AudioClip对象。方法具体介绍如下。

- **public static final AudioClip newAudioClip(URL url):** 从给定URL处获取音频剪辑。
- **public AudioClip getAudioClip(URL url):** 返回URL参数指定的AudioClip对象。

可见这两个方法都需要URL对象作为参数，所以必须将加载的File对象转换成URL对象。

3. 多线程的应用。

4. 程序设计。

本案例由五个类构成，分别是FindDifference类、FrameFindDifference类、paneOption类、paneName类和paneScores类。其中FindDifference类是本程序的主类，它调用FrameFindDifference类生成游戏的主界面。

FrameFindDifference类封装了本游戏的所有业务逻辑，主要职责是初始化界面，并对整个游戏的过程进行管理，对用户输入进行响应。它分别调用paneOption类、paneName类和paneScores类来完成对游戏的控制。

paneOption类用来显示“游戏设置”窗口，可以对音效、背景音乐和游戏难度进行设置。

paneName类用来在游戏结束时显示“输入姓名”窗口，可以将玩家输入的姓名以及获得的分数写入排行榜文件。

paneScores类用来显示“排行榜”窗口，它在硬盘游戏目录中创建排行榜文本文件，将10个最好的游戏成绩写入该文件，并在“排行榜”窗口中显示。

步骤详解

1. 声明实现Runnable接口的类：

```
class FrameFindDifference extends JFrame implements KeyListener, ActionListener, MouseListener, Runnable {}
```

Runnable接口由那些实例倾向于由线程执行的类来实现。类必须定义一个名为run的无参数的方法。设计该接口的目的是为执行代码的对象在其活动时提供一个公共协议，即在整个游戏过程中KeyListener、ActionListener、MouseListener启动并且不停止。

```
public void run() {}
```


2. 定义JLabel对象要显示的图像:

```

JLabel boom = new JLabel(new ImageIcon(""));
JLabel left = new JLabel(new ImageIcon(""));
JLabel apple = new JLabel(new ImageIcon("../src\\finddifference\\images\\apple.gif"));
JLabel mg = new JLabel(new ImageIcon("../src\\finddifference\\images\\mg.gif"));
JLabel stage = new JLabel("1",new ImageIcon("../src\\finddifference\\images\\mg.gif"),JLabel
.RIGHT);
JLabel train = new JLabel(new ImageIcon("../src\\finddifference\\images\\train.gif"));
JLabel life = new JLabel("×",new ImageIcon("../src\\finddifference\\images\\train.gif"),JLabel
.LEFT);

```

使用setIcon()方法:

```

left.setIcon(new ImageIcon("../src\\finddifference\\images\\1.jpg"));    //左右使用同一幅图片
right.setIcon(new ImageIcon("../src\\finddifference\\images\\1.jpg"));    //左右使用同一幅图片

```

设置控件位置和大小:

```

left1.setBounds(sint[s_array[1]][0],sint[s_array[1]][1],80,80);

```

3. 设置关卡方法, 5个关卡方法 (stage1 ~ stage5) 有5幅不同的游戏画面, 它们在逻辑上是完全一样的。

定义10个错误位置:

```

int sint[][] = {{0,0},{17,19},{76,470},{133,388},{397,411},
                {369,203},{390,103},{254,0},{103,80},{65,240},{260,166}};
left.setIcon(new ImageIcon("../src\\finddifference\\images\\1.jpg"));    //左右使用同一幅图片
right.setIcon(new ImageIcon("../src\\finddifference\\images\\1.jpg"));    //左右使用同一幅图片

```

随机选取10个错误位置中的5个放置到左边图片相应位置:

```

left1.setBounds(sint[s_array[1]][0],sint[s_array[1]][1],80,80);
left1.setIcon(new ImageIcon("../src\\finddifference\\images\\1_"+String.valueOf(s_array[1])+
".jpg"));
left2.setBounds(sint[s_array[2]][0],sint[s_array[2]][1],80,80);
left2.setIcon(new ImageIcon("../src\\finddifference\\images\\1_"+String.valueOf(s_array[2])+
".jpg"));
left3.setBounds(sint[s_array[3]][0],sint[s_array[3]][1],80,80);
left3.setIcon(new ImageIcon("../src\\finddifference\\images\\1_"+String.valueOf(s_array[3])+
".jpg"));
left4.setBounds(sint[s_array[4]][0],sint[s_array[4]][1],80,80);
left4.setIcon(new ImageIcon("../src\\finddifference\\images\\1_"+String.valueOf(s_array[4])+
".jpg"));
left5.setBounds(sint[s_array[5]][0],sint[s_array[5]][1],80,80);
left5.setIcon(new ImageIcon("../src\\finddifference\\images\\1_"+String.valueOf(s_array[5])+
".jpg"));

```

设置本关卡声音文件, 使用AudioClip:

```

Sstage=Applet.newAudioClip(new File("../src\\finddifference\\sound\\S1.au").toURI().toURL());

```

在Java 2平台出现之前, Java仅支持AU文件。Java 2平台增加了对AIFF、WAV以及三种MIDI文件类型的支持。所支持的三种MIDI文件格式为MIDI文件类型0、MIDI文件类型1、以及RMF。如果getAudioClip或newAudioClip方法不能找到指定的声音文件, AudioClip对象的

值将是空的。试着播放空对象会导致出错，所以标准的过程首先是对该条件进行检测。

```
try {
    Sstage=Applet.newAudioClip(new File("../src\\finddifference\\sound\\S1.au").toURI().toURL());
}
catch (MalformedURLException he) {
    System.out.println("err");
}
```

4. 为FrameFindDifference类添加线程的运行方法:

```
public void run() {
    while(booend) {...}
} //run
```

5. 为FrameFindDifference类添加鼠标单击事件处理方法:

```
public void mouseClicked(MouseEvent e) { //鼠标单击事件处理
    //生命值
    int i=0;
    //单击“开始游戏”按钮，根据玩家设置初始化火车移动速度、难度分数和游戏界面
    if (e.getSource()==font1) {
        Sinit.stop();
        if (paneOption.b3.getText().equals("✓")) { //难度低
            numspeed=600;
            speeds=20;
            numleave=300;
        }
        else
            ...

        ready.start(); //启动ready线程
    }
    else //单击“英雄榜”按钮，读取相应文件并在“排行榜”窗口显示
        if (e.getSource()==font2) {
            Smove.play();
            paneScores.readScores();
            panescores.setVisible(true);
        }
    else //单击“游戏设置”按钮，显示“游戏设置”窗口
        ...
}
```

6. 声音对象的初始化:

```
Scom = Applet.newAudioClip(new File("../src\\finddifference\\sound\\complete.au").toURI()
.toURL());
Sinit = Applet.newAudioClip(new File("../src\\finddifference\\sound\\init.wav").toURI().toURL());
Sscores = Applet.newAudioClip(new File("../src\\finddifference\\sound\\scores.wav").toURI()
.toURL());
...
```

程序源代码与解释

```
/* * FrameFindDifference.java*/
class FrameFindDifference extends JFrame implements KeyListener,ActionListener,MouseListener,
Runnable {
    /*构造方法*/
    public FrameFindDifference() {
        numtotal=0;
```



```

timer = new Timer(40,this);
setSize(1024, 768);
Image img= Toolkit.getDefaultToolkit().getImage("../src\\finddifference\\images\\cur.png");
setCursor(this,img);
addWindowListener(new WindowAdapter() {
    public void windowClosing( WindowEvent e) {
        System.exit(0);
    }
});
setUndecorated(true);
//生成SNUM个不相等的随机数来选关
for(int i=1;i<=SNUM;i++) {
    array[i]=(int)(Math.random()*SNUM)+1;
    for (int j=1;j<i ; j++) {
        if (array[i]==array[j])
            i--;
    }
}
System.out.print("关数: ");
for (int i=1;i<=SNUM ;i++ ) {
    System.out.print(array[i]+"," );
}
System.out.println("");
//...
Sinit.play();
}
//设置光标
public static void setCursor(Component c, Image image) {
    Cursor cursor = toolkit.createCustomCursor(image, new Point(10, 10), null);
    c.setCursor(cursor);
}
/*从10处错误中随机选择5处*/
public void selectWrong() {
//生成5个不相等的随机数(1~10)
    for(int i=1;i<6;i++) {
        s_array[i]=(int)(Math.random()*10)+1;
        for (int j=1;j<i ; j++) {
            if (s_array[i]==s_array[j])
                i--;
        }
    }
    System.out.print("错误值: ");
    for (int i=1;i<6 ;i++ ) {
        System.out.print(s_array[i]+"," );
    }
    System.out.println("");
}
/*统计找到的不同数*/
public void pass() {
    pass=pass+1;//找到的不同数加1
total.setText(String.valueOf(numtotal));//显示当前总分数
    if (pass==5) { //5个不同全部找到则火车停止移动
        boo=false;
    }
}
}

```



```

public void mouseEntered(MouseEvent e2) {    //鼠标移入事件处理
    if (e2.getSource()==font1) {
        font1.setForeground(Color.red);
    }
    if (e2.getSource()==font2) {
        font2.setForeground(Color.red);
    }
    if (e2.getSource()==font3) {
        font3.setForeground(Color.red);
    }
    if (e2.getSource()==font4) {
        font4.setForeground(Color.red);
    }
}

public void mouseReleased(MouseEvent e){}
public void mousePressed(MouseEvent e){}
public void keyTyped(KeyEvent ke){}
public void keyReleased(KeyEvent ke){}
public void keyPressed(KeyEvent ke) {    //键盘按键事件处理
    if (panename.isVisible()) {
        if (ke.getKeyCode()==8) {    //按下空格键
            switch (playname.length()) {
                case 1:playname="";break;
                case 2:playname=playname.substring(0,1);break;
                case 3:playname=playname.substring(0,2);break;
            }
        }
        if ((ke.getKeyCode()<=90)&&(ke.getKeyCode()>=65)&&(playname.length()<3)) {
//按下字母键且低于3个
            String s=String.valueOf(ke.getKeyChar());
            playname=playname+s.toUpperCase();
        }
        panename.repaint();
    }
    if (ke.getKeyCode()==27) {    //按下Esc键
        System.exit(0);
    }
}

public void actionPerformed(ActionEvent ae) {    //Action事件处理
    if (numtotal<int1) {
        numtotal=numtotal+10;
        numbonus=numbonus-10;
        total.setText(String.valueOf(numtotal));
        snum.setText(String.valueOf(numbonus));
    }
    else {
        timer.stop();
        Sscores.stop();
    }
}
}

```




案例6：八皇后问题

案例运行效果与操作

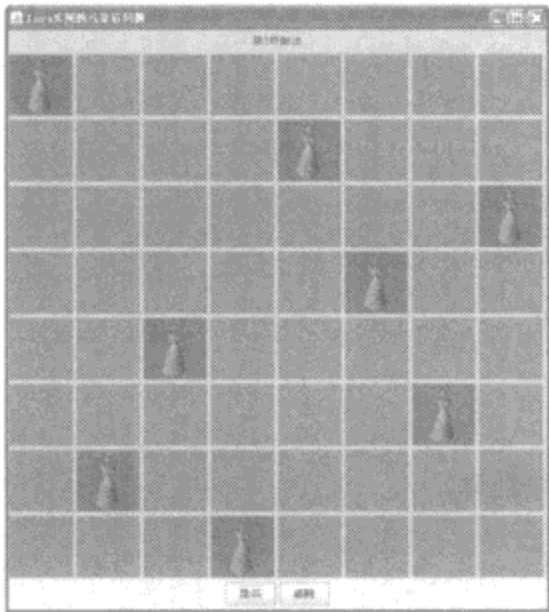


图4-24 运行界面

八皇后问题是一个古老而著名的问题，是回溯算法的典型例题。八皇后问题是十九世纪著名的数学家高斯提出的：在 8×8 格的国际象棋上摆放八个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上，问有多少种摆法。本案例是Java实现的八皇后问题，它使用递归算法，用图形界面显示八皇后问题的92种解法。程序运行后，首先显示八皇后问题的第1种解法，界面如图4-24所示。

此时单击下方的“显示”按钮，则循环显示八皇后问题的92种解法，即每次单击就显示当前解法的下一种解法，同时在窗口上方显示当前是哪一种解法。当显示第92种解法时，界面如图4-25所示。此时再单击

“显示”按钮，又会显示第1种解法。

单击下方的“清除”按钮，则清除所有棋子，显示空棋盘，如图4-26所示。

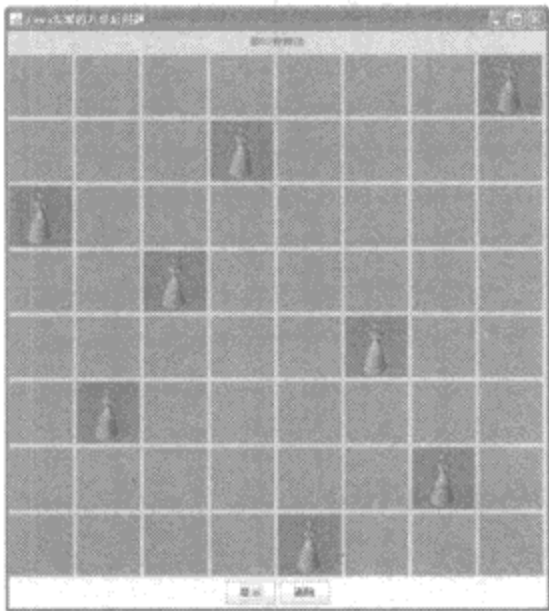


图4-25 第92种解法

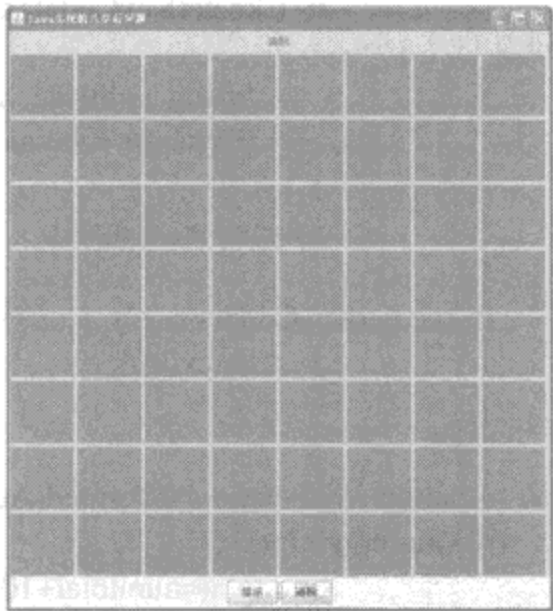


图4-26 清除界面

制作要点

- 1. JFrame的用法。
- 2. 递归算法的应用技巧。

递归函数即自调用函数，在函数体内直接或间接地调用自己，即函数的嵌套是函数本身。递归的目的是简化程序设计，使程序易读。当一个方法调用它自身的时候，堆栈就会给新的局部变量和自变量分配内存，方法代码就带着这些新的变量从头执行。递归调用并不产生方法新的拷贝，只有参数是新的。每当递归调用返回时，旧的局部变量和自变量就从堆栈中清除，从方法中的调用点重新开始运行。递归方法像“望远镜”一样，可以自由伸缩。

许多子程序的递归版本执行时会比它们的迭代版本要慢一点，因为它们增加了额外的方法调用的消耗。对一个方法太多的递归调用会引起堆栈崩溃。因为自变量和局部变量的存储都在堆栈中，每次调用都创建这些变量新的拷贝，堆栈有可能被耗尽。如果发生这种情况，Java在运行时系统会产生异常。但是，除非递归子程序疯狂运行，否则不必担心这种情况。

递归的主要优点在于：某些类型的算法采用递归比采用迭代算法要更加清晰和简单。例如快速排序算法按照迭代方法是很难实现的。还有其他一些问题，特别是人工智能问题，就依赖于递归提供解决方案。另外，有些人认为递归要比迭代简单。

步骤详解

1. 为FrameQueen编写构造方法，首先定义好相应的成员变量，然后在源代码编辑器中添加代码如下：

```
FrameQueen() {    //构造方法
    super("Java实现的八皇后问题");    //窗口标题
    addWindowListener(new WindowAdapter(){    //窗口关闭事件侦听器
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
    initialize();    //调用initialize方法进行窗口初始化
}
```

2. 设置棋盘布局

设置棋盘面板布局：

```
jPanelChess.setLayout(new GridLayout(EightQueen.QueenMax,EightQueen.QueenMax,2,2));
```

设置棋盘面板尺寸：

```
jPanelChess.setSize(76*EightQueen.QueenMax,86*EightQueen.QueenMax);
```

设置按钮面板尺寸：

```
jPanelButton.setSize(50,50);
```

设置信息标签前景色（字体颜色）为红色：

```
jLabelMessage.setForeground(Color.red);
```

添加按钮事件侦听器：

```
jButtonDisplay.addActionListener(new Listener());
jButtonClear.addActionListener(new Listener());
```

按钮添加到按钮面板：

```
jPanelButton.add(jButtonDisplay);
jPanelButton.add(jButtonClear);
```

信息标签添加到标签面板：

```
jPanelLabel.add(jLabelMessage);
```

将各个面板添加到主窗口：


```

getContentPane().add(jPanelLabel,BorderLayout.NORTH);
getContentPane().add(jPanelChess,BorderLayout.CENTER);
getContentPane().add(jPanelButton,BorderLayout.SOUTH);

```

设置主窗口位置:

```
setLocation(300,0);
```

设置主窗口尺寸:

```
setSize(76*EightQueen.QueenMax+50,86*EightQueen.QueenMax+50);
```

设置各个面板背景色:

```

jPanelChess.setBackground(Color.ORANGE);
jPanelButton.setBackground(Color.YELLOW);
jPanelLabel.setBackground(Color.green);

```

设置主窗口背景色:

```
setBackground(Color.MAGENTA);
```

3. 为FrameQueen类编写事件处理的内部类Listener, 该类实现了ActionListener接口, 用于主窗口的按钮事件处理。在源代码编辑器中添加代码如下:

```

class Listener implements ActionListener {    //侦听器类
    public void actionPerformed(ActionEvent e) {    //事件处理
        if(e.getSource()==jButtonDisplay)    //“显示”按钮事件处理
            if(i< EightQueen.oktimes) {    //第1~91种解法
                ...
            }
            else { //第92种解法
                ...
            }
        if(e.getSource()==jButtonClear) {    //“清除”按钮事件处理
            for(int j=0;j< size;j++) {
                jLabelImage[j].setIcon(IconNoQueen); //清除棋盘
            }
        }
    }
}

```

4. 为EightQueen类编写placequeen方法, 该方法是本案例的核心算法, 其功能就是获得每一种解法的字符串并保存在rows数组中, 用的是递归方法。八皇后问题递归算法逻辑是: 如果有一个皇后位置为chess[i]=j, 则不安全的地方是k行的j位置、j+k-i位置和j-k+i位置。

```

public static void placequeen(int num) { //num为现在要放置的行数
    int i=0;
    boolean qsave[] = new boolean[QueenMax];
    for(;i< QueenMax;i++)
        qsave[i]=true;
    i=0;
    //下面先把安全位数组完成
    //i是现在要检查的数组值
    while (i< num) {
        qsave[chess[i]]=false;
        int k=num-i;
        if ( (chess[i]+k >= 0) && (chess[i]+k < QueenMax) )
            qsave[chess[i]+k]=false;
    }
}

```



```

        if ( (chess[i]-k >= 0) && (chess[i]-k < QueenMax) )
            qsave[chess[i]-k]=false;
        i++;
    }
    //下面遍历安全位
    for(i=0;i< QueenMax;i++) {
        if (qsave[i]==false)
            continue;
        if (num< QueenMax-1) {
            chess[num]=i;
            placequeen(num+1);//递归调用
        }
        else { //产生每一种解法的解法字符串, 其中+代表皇后位置
            chess[num]=i;
            oktimes++;
            String row="";
            for (i=0;i< QueenMax;i++) {
                if (chess[i]==0);
                else
                    for(int j=0;j< chess[i];j++) row+="-";
                row+=" ";
                int j = chess[i];
                while(j< QueenMax-1) {
                    row+="-";
                    j++;
                }
                rows[oktimes-1]=row;
            }
        }
    }
    //遍历完成就停止
}

```

5. 主方法的代码:

```

for (int i=0;i< QueenMax;i++)
    chess[i]=-1;
placequeen(0);    //递归调用placequeen方法
FrameQueen queen=new FrameQueen(); //显示主窗口
System.out.println("\n八皇后问题共有"+oktimes+"个解法。");

```

程序源代码与解释

```

/* * FrameQueen.java */
public class FrameQueen extends JFrame {
    //非皇后位置图标
    Icon IconNoQueen=new ImageIcon("../src\\eightqueen\\imageicons\\noqueen.jpg");
    //皇后位置图标
    Icon IconQueen=new ImageIcon("../src\\eightqueen\\imageicons\\queen.jpg");
    int size=EightQueen.QueenMax*EightQueen.QueenMax; //棋盘格子总数
    public JLabel[] jLabelImage=new JLabel[size]; //棋盘格子标签数组
    int i=0;
    FrameQueen() { //构造方法
        super("Java实现的八皇后问题"); //窗口标题
    }
}

```



```

addWindowListener(new WindowAdapter(){ //窗口关闭事件侦听器
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
initialize(); //调用initialize方法进行窗口初始化
}
public void initialize() { //进行窗口初始化
jPanelChess.setLayout(new GridLayout(EightQueen.QueenMax,EightQueen.QueenMax,2,2));
jPanelChess.setSize(76*EightQueen.QueenMax,86*EightQueen.QueenMax);
jPanelButton.setSize(50,50);
jLabelMessage.setForeground(Color.red);
for(int j=0;j< size;j++) { //画出第一种解法棋盘
    String str;
    str=(EightQueen.rows[0].substring(j,j+1)).trim();
    if(str.equals("+")) //皇后位置
        JLabelImage[j]=new JLabel(IconQueen); //设置皇后位置图标
    else
        JLabelImage[j]=new JLabel(IconNoQueen); //设置非皇后位置图标
}
i++;
//添加按钮事件侦听器
jButtonDisplay.addActionListener(new Listener());
jButtonClear.addActionListener(new Listener());
for(int i=0;i< size;i++) //棋子图标添加到棋盘面板
    jPanelChess.add(jLabelImage[i]);
jPanelButton.add(jButtonDisplay);
jPanelButton.add(jButtonClear);
jPanelLabel.add(jLabelMessage);
getContentPane().add(jPanelLabel,BorderLayout.NORTH);
getContentPane().add(jPanelChess,BorderLayout.CENTER);
getContentPane().add(jPanelButton,BorderLayout.SOUTH);
setLocation(300,0);
setSize(76*EightQueen.QueenMax+50,86*EightQueen.QueenMax+50);
jPanelChess.setBackground(Color.ORANGE);
jPanelButton.setBackground(Color.YELLOW);
jPanelLabel.setBackground(Color.green);
setBackground(Color.MAGENTA);
//显示主窗口
setVisible(true);
}

```

本章小结

Java游戏编程已经在游戏开发中成为重头戏。使用Java的好处是明显的：代码易读易写，而且可以在不同平台上方便、快捷地开发。本章通过几个不同难易程度的实用范例程序，讲述了在Java语言中进行游戏编程的基本方法，主要包括多线程的应用、图像的处理、声音的处理等。在实际应用中，J2ME平台上的游戏开发范围更广，我们将在第7章进行讲解。



第5章 Java与网络

本章内容

- 案例1: 简单的多线程服务器
- 案例2: 用Java实现的HTTP服务器端例程
- 案例3: 一个简单的HTML浏览器
- 案例4: 用JavaMail发送邮件
- 案例5: Java版MSN
- 案例6: Java实现HTTP队列下载
- 案例7: Java实现HTTP验证
- 本章小结



案例1: 简单的多线程服务器

案例运行效果与操作

本案例展示了一个简单的多线程服务器程序的架构, 可以在此基础上对命令进行扩充。本案例未涉及数据库, 用户执行命令后只显示相应信息。生成项目后在DOS窗口中执行`java -jar receiveserver.jar`命令运行程序, 界面如图5-1所示, 表明服务器已经启动, 并给出当前时间和服务的端口号。此时服务器会一直运行, 只有按`Ctrl+C`组合键或关闭DOS窗口才能退出服务器程序。

```
C:\javabook\chapter5\receiveserver\dist>java -jar receiveserver.jar
Welcome to the server!
Tue Sep 18 22:13:43 CST 2007
The server is ready!
Port: 9090
```

图5-1 服务器运行时界面

启动服务器程序后, 再打开另一个DOS窗口, 用`telnet machine port`命令连接服务器, 其中`machine`为本机名或地址(如192.168.1.101), `port`为程序中指定的端口(如9090)。如果连接成功, 服务器会给出欢迎信息和提示信息, 此时界面如图5-2所示。

```
C> Telnet 192.168.1.101
Welcome to the server!
Now is: Tue Sep 18 22:16:53 CST 2007 Port:9090
What can I do for you?
```

图5-2 连接服务器成功界面

由于本案例是一个简单的多线程服务器示例程序, 所以只提供了三个命令: **HELP**、**QUERY**和**QUIT**, 而且这三个命令只是显示一些相应信息, 不进行任何其他操作。在如图5-2所示的界面中, 输入**HELP**命令, 会显示当前服务器支持的命令名称, 如图5-3所示。

输入QUERY命令，会给出提示信息，如图5-4所示。

```
Telnet 192.168.1.101
Welcome to the server!
Now is: Tue Sep 18 22:16:53 CST 2007 Port:9090
What can I do for you?
HELP
query
quit
help
```

图5-3 执行HELP命令

```
QUERY
OK to query something!
```

图5-4 执行QUERY命令

如果输入了除HELP、QUERY和QUIT之外服务器不支持的命令，会提示当前命令未发现，请参考HELP命令，如图5-5所示。

输入QUIT命令，则telnet命令终止，失去与服务器的连接，界面如图5-6所示。

```
exit
Command not Found! Please refer to the HELP!
he
Command not Found! Please refer to the HELP!
```

图5-5 执行其他命令

```
quit
失去了跟主机的连接。
C:\Documents and Settings\neu>
```

图5-6 执行QUIT命令

制作要点

1. ServerSocket类的用法。

Java软件包内支持的网络协议为TCP/IP，也是当今最流行的广域网/局域网协议。Java有关网络的类及接口定义在java.net包中。客户端软件通常使用java.net包中的核心类Socket与服务器的某个端口建立连接，而服务器程序不同于客户机，它需要初始化一个端口进行监听，遇到连接呼叫，才与相应的客户机建立连接。Java.net包的ServerSocket类包含了编写服务器系统所需的一切。ServerSocket类的使用非常简单，下面给出其部分定义：

```
public class ServerSocket {
    public ServerSocket(int port) throws IOException ;
    public Socket accept() throws IOException ;
    public InetAddress getInetAddress() ;
    public int getLocalPort() ;
    public void close() throws IOException ;
    public synchronized void setSoTimeout (int timeout) throws SocketException ;
    public synchronized int getSoTimeout() throws IOException ;
}
```

其中，ServerSocket构造方法是服务器程序运行的基础，它将参数port指定的端口初始化作为该服务器的端口，监听客户机连接请求。Port的范围是0~65536，其中0~1023是标准Internet协议保留端口，而且在UNIX主机上，这些端口只有root用户可以使用。一般自定义的端口号在8000~16000之间。仅初始化了ServerSocket还是远远不够的，它没有同客户机交互的套接字（Socket），因此需要调用该类的accept方法接受客户呼叫。accept()方法直到有连接请求才返回通信套接字（Socket）的实例。通过这个实例的输入、输出流，服务器可以接收用户指令，并将相应结果返回给客户机。ServerSocket类的getInetAddress和getLocalPort方法可得到该服务器的IP地址和端口。setSoTimeout和getSoTimeout方法分别是设置和得到服务器超时设置，如果服务器在timeout设定时间内还未得到accept方法返回的套接字实例，则抛出IOException的异常。

2. 服务器程序多线程的使用。

Java的多线程可谓是Java编程的精华之一，运用得当可以极大地改善程序的响应时间，提高程序的并行性。在服务器程序中，由于往往要接收不同客户机的同时请求或命令，因此可以对每个客户机的请求生成一个命令处理线程，同时对各用户的指令做出反应。在一些较复杂的系统中，还可以为每个数据库查询指令生成单独的线程，并行对数据库进行操作。实践证明，采用多线程设计可以很好地改善系统的响应，并保证用户指令执行的独立性。由于Java本身是“线程安全”的，因此有一条编程原则，即能够独立在一个线程中完成的操作就应该开辟一个新的线程。

3. 程序设计。

本案例展示了一个多线程服务器程序的架构，可以在此基础上对命令进行扩充。本例未涉及数据库，用户执行命令后只显示相应信息。如果在线程运行中需要根据用户指令对数据库进行更新操作，则应注意线程间的同步问题，使同一个更新方法一次只能由一个线程调用。这里共有两个类：**ReceiveServer**类和**ServerThread**类，其中**ReceiveServer**包含启动代码（**main()**），并初始化**ServerSocket**的实例，在**accept**方法返回用户请求后，将返回的套接字（**Socket**）交给生成的线程类**ServerThread**的实例，直到该用户结束连接。

步骤详解

1. 设置服务器的端口号：

```
final int RECEIVE_PORT=9090;
```

2. 初始化**ServerSocket**：

```
rServer=new ServerSocket(RECEIVE_PORT);
```

3. 接收客户机连接请求：

```
request=rServer.accept();           //接收客户机连接请求
receiveThread=new ServerThread(request); //生成ServerThread的实例
receiveThread.start();               //启动ServerThread线程
```

4. 初始化输入/输出流：

```
reader=new InputStreamReader (clientRequest.getInputStream());
writer=new OutputStreamWriter (clientRequest.getOutputStream());
input=new BufferedReader(reader);
output=new PrintWriter(writer,true);
```

5. 为**ServerThread**类添加线程的运行方法，包括接收客户机指令及各指令的处理方式：

```
public void run(){
...
    str=input.readLine();           //接收客户机指令
...
    if(str==null || command.equals("QUIT")) //命令QUIT结束本次连接
        done=true;
    else if(command.equals("HELP")){      //命令HELP查询本服务器可接受的命令
        output.println("");
        output.println("query");
        output.println("quit");
        output.println("help");
    }
...
}
```


6. 关闭套接字:

```
clientRequest.close();
```

程序源代码与解释

```
/** ReceiveServer.java*/
package receiveserver;
import java.io.*;
import java.util.*;
import java.net.*;
public class ReceiveServer{
    final int RECEIVE_PORT=9090; //该服务器的端口号
    public ReceiveServer() { //构造方法
        ServerSocket rServer=null; //ServerSocket的实例
        Socket request=null; //用户请求的套接字
        Thread receiveThread=null;
        try{
            rServer=new ServerSocket(RECEIVE_PORT);
            //初始化ServerSocket
            System.out.println("Welcome to the server!");
            System.out.println(new Date());
            System.out.println("The server is ready!");
            System.out.println("Port: "+RECEIVE_PORT);
            while(true){ //等待用户请求
                //接收客户机连接请求, 参见步骤详解3
            }
        }catch(IOException e){
            System.out.println(e.getMessage());
        }
        public static void main(String args[]){
            new ReceiveServer();
        }
    }

/** ServerThread.java*/
class ServerThread extends Thread {
    Socket clientRequest; //用户连接的通信套接字
    BufferedReader input; //输入流
    PrintWriter output; //输出流
    public ServerThread(Socket s) { //构造方法
        this.clientRequest=s; //接收ReceiveServer传来的套接字
        InputStreamReader reader;
        OutputStreamWriter writer;
        try{ //初始化输入/输出流
            //参见步骤详解4
        }catch(IOException e){
            System.out.println(e.getMessage());
        }
        output.println("Welcome to the server!"); //客户机连接欢迎词
        output.println("Now is: "+new java.util.Date()+" "+ "Port:"+clientRequest.getLocalPort());
        output.println("What can I do for you?");
    }
    public void run(){ //线程的执行方法
        String command=null; //用户指令
```



```

String str=null;
boolean done=false;
while(!done){
    //参见步骤详解5
    //else if ..... //在此可加入服务器的其他指令
    else if(!command.startsWith("HELP") &&
        !command.startsWith("QUIT") &&
        !command.startsWith("QUERY")){
        output.println("");
        output.println("Command not Found! Please refer to the HELP!");
    }
}
try{
    clientRequest.close(); //关闭套接字
}catch(IOException e){
    System.out.println(e.getMessage());
}
command=null;
}
}

```



案例2：用Java实现的HTTP服务器端例程

案例运行效果与操作

本案例同样是一个简单的多线程服务器程序，与上一个案例不同的是，它实现了基本的HTTP服务器功能，即可以读取并显示Web浏览器提交的请求信息。

生成项目后在DOS窗口中执行`java-jar httpserver.jar`命令运行程序，界面如图5-7所示，表明服务器已经启动，并给出当前服务的端口号。此后服务器会一直运行，只有按`Ctrl+C`组合键或关闭DOS窗口才能退出服务器程序。

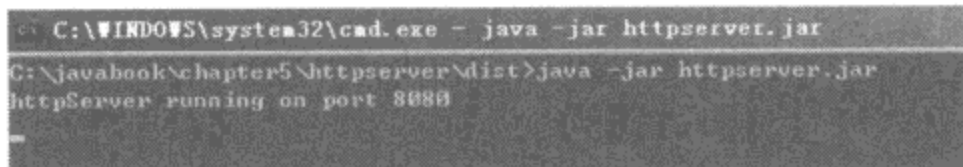


图5-7 服务器运行时界面

启动服务器程序后，打开浏览器窗口，在地址栏中输入`http://localhost:8080/compress.html`，其中`compress.html`为本案例程序目录中存在的一个HTML文件，8080为程序中指定的服务器端口默认值。按`Enter`键后在浏览器中显示该文件内容，界面如图5-8所示。此时服务器运行窗口中的显示如图5-9所示。

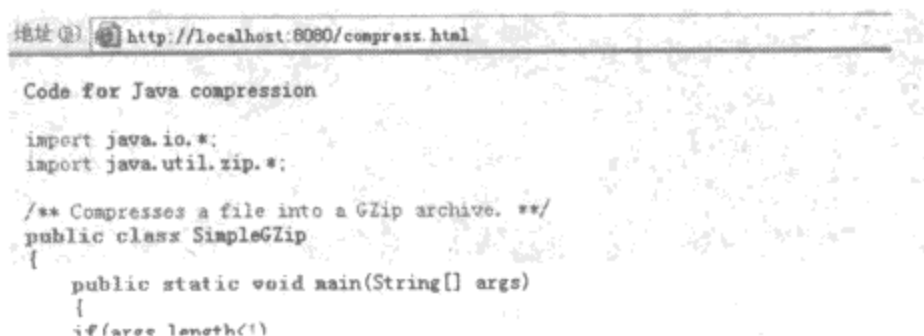


图5-8 打开HTML文件界面

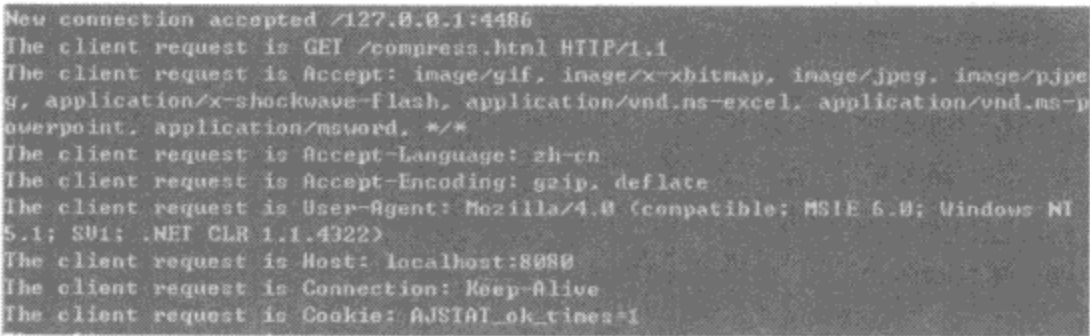


图5-9 打开HTML文件时的服务器界面

除了HTML文件，还可以打开文本文件等，如图5-10和图5-11所示。

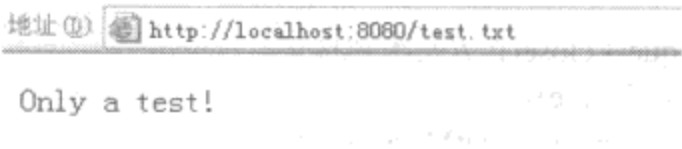


图5-10 打开文本文件界面

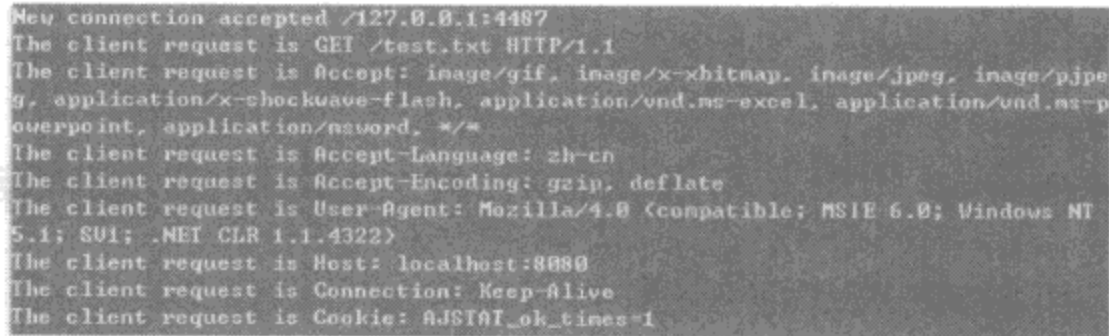


图5-11 打开文本文件时的服务器界面

如果输入的文件路径在本案例程序目录中不存在，则显示找不到网页信息，界面如图5-12所示。此时服务器运行窗口中的显示如图5-13所示。



图5-12 “找不到网页”界面

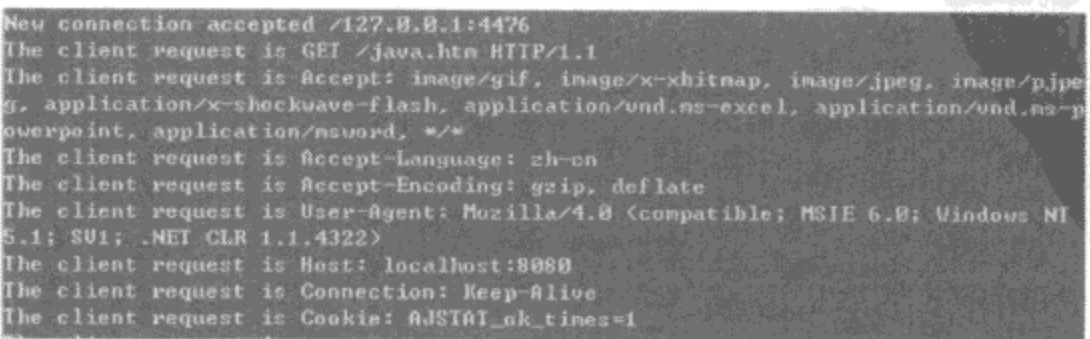


图5-13 “找不到网页”时的服务器界面

另外，在服务器程序运行时还可以指定端口，不指定则默认是8080端口，如图5-14所示是指定端口号9090时的情形。

```
C:\WINDOWS\system32\cmd.exe - java -jar httpserver.jar 9090
C:\java\book\chapter5\httpserver\dist>java -jar httpserver.jar 9090
httpServer running on port 9090
```

图5-14 程序运行时指定端口

制作要点

1. `ServerSocket`类的用法。
2. `StringTokenizer`类的使用。

`StringTokenizer`类允许应用程序将字符串分解为标记。`StringTokenizer`方法不区分标识符、数和带引号的字符串，它们也不识别注释并跳过注释。

可以在创建时指定，也可以根据每个标记来指定分隔符（分隔标记的字符）集。

`StringTokenizer`的实例有两种行为方式，这取决于它在创建时使用的`returnDelims`标志的值是`true`还是`false`。

（1）如果标志为`false`，则分隔符用来分隔标记。标记是连续字符（不是分隔符）的最大序列。

（2）如果标志为`true`，则认为那些分隔符本身即为标记。因此，标记要么是一个分隔符，要么是那些连续字符（不是分隔符）的最大序列。

`StringTokenizer`对象在内部维护字符串中要被标记的当前位置。某些操作将当前位置移至已处理的字符后。

通过截取字符串的一个子串来返回标记，该字符串用于创建`StringTokenizer`对象。

3. 程序设计。

本案例有两个类：`httpServer`类和`httpRequestHandler`类。其中，`httpServer`类包含启动代码（`main()`），并初始化`ServerSocket`的实例，在`accept`方法返回用户请求后，将返回的套接字（`Socket`）交给生成的线程类`httpRequestHandler`的实例，直到该用户结束连接。

步骤详解

1. 监听服务器端口号，启动`Socket`。

简单来讲，**HTTP**服务或者说整个网络编程的基本模型就是客户机到服务器模型，就是两个进程之间相互通信，其中一个必须提供一个固定的位置，而另一个则只需要知道这个固定的位置，并去建立两者之间的联系，完成数据的通信。提供固定位置的通常称为服务器。端口是唯一标识每台计算机唯一服务的，所以创建`Socket`类的一个实例对象并提供一个端口资源就建立了一个固定位置，可以让其他计算机来访问了。

```
port = Integer.parseInt(args[0]);
server_socket = new ServerSocket(port);
```

2. 用`ServerSocket`中的`accept()`方法可以使服务器检测到指定端口的活动：

```
Socket socket = server_socket.accept();
```

3. 服务器接收客户机请求后，创建分线程，实现了**HTTP**协议的处理并启动线程：


```
httpRequestHandler request = new httpRequestHandler(socket);
Thread thread = new Thread(request);
thread.start();
```

得到输入/输出流:

```
this.socket = socket;
this.input = socket.getInputStream();
this.output = socket.getOutputStream();
this.br = new BufferedReader(new InputStreamReader(socket.getInputStream()));
```

4. 处理信息的接收和发送。

读取并显示Web浏览器提交的请求信息:

```
String headerLine = br.readLine();
System.out.println("The client request is "+headerLine);
```

根据请求字符串中的空格拆分客户请求, 利用StringTokenizer类完成字符串的拆分, 这个类可以实现根据字符串中指定的分隔符(缺省为空格)将字符串拆分成为字串的功能。利用nextToken()方法依次得到这些字串。

```
StringTokenizer s = new StringTokenizer(headerLine);
String temp = s.nextToken();
```

contentType()方法用于判断文件的类型:

```
if (fileName.endsWith(".htm") || fileName.endsWith(".html")) {
    return "text/html";
...
statusLine = "HTTP/1.0 200 OK" + CRLF ;
contentTypeLine = "Content-type: " +contentType( fileName ) + CRLF ;
contentLengthLine = "Content-Length: " + (new Integer(fis.available())).toString() + CRLF;
```

将文件输出到套接字输出流中:

```
while ((bytes = fis.read(buffer)) != -1 ){
    os.write(buffer, 0, bytes);
```

5. 关闭套接字和流:

```
output.close();
br.close();
socket.close();
```

程序源代码与解释

```
/** httpServer.java*/
public class httpServer{
    public static void main(String args[]) {
        int port;
        ServerSocket server_socket;
        //读取服务器端口号
        try {
            port = Integer.parseInt(args[0]);
        }
        catch (Exception e) {
```



```

        port = 8080;
    }
    try { //监听服务器端口, 等待连接请求
        server_socket = new ServerSocket(port);
        System.out.println("httpServer running on port " + server_socket.getLocalPort());
        //显示启动信息
        while(true) {
            Socket socket = server_socket.accept();
            System.out.println("New connection accepted " +
                socket.getInetAddress() + ":" + socket.getPort());
            //创建分线程
            try {
                httpRequestHandler request = new httpRequestHandler(socket);
                Thread thread = new Thread(request);
                //启动线程
                thread.start();
            }
            catch(Exception e) {
                System.out.println(e);
            }
        }
    }
    catch (IOException e) {
        System.out.println(e);
    }
}

class httpRequestHandler implements Runnable {
    final static String CRLF = "\r\n";
    Socket socket;
    InputStream input;
    OutputStream output;
    BufferedReader br;
    //构造方法
    public httpRequestHandler(Socket socket) throws Exception{
        this.socket = socket;
        this.input = socket.getInputStream();
        this.output = socket.getOutputStream();
        this.br = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    }
    //实现Runnable接口的run()方法
    public void run() {
        try {
            processRequest();
        }
        catch(Exception e) {
            System.out.println(e);
        }
    }
    private void processRequest() throws Exception {
        while(true) {
            //读取并显示Web浏览器提交的请求信息
            String headerLine = br.readLine();
            System.out.println("The client request is "+headerLine);
            if(headerLine.equals(CRLF) || headerLine.equals(""))

```



```

        break;
StringTokenizer s = new StringTokenizer(headerLine);
String temp = s.nextToken();
if(temp.equals("GET")) {
    String fileName = s.nextToken();
    fileName = "." + fileName ;
    //打开所请求的文件
    FileInputStream fis = null ;
    boolean fileExists = true ;
    try{
        fis = new FileInputStream( fileName ) ;
    }
    catch ( FileNotFoundException e ){
        fileExists = false ;
    }
    //完成回应消息
    String serverLine = "Server: a simple java httpServer";
    String statusLine = null;
    String contentTypeLine = null;
    String entityBody = null;
    String contentLengthLine = "error";
    if ( fileExists ){
        statusLine = "HTTP/1.0 200 OK" + CRLF ;
        contentTypeLine = "Content-type: " +contentType( fileName ) + CRLF ;
        contentLengthLine = "Content-Length: " + (new Integer(fis.available())).toString() +
CRLF;
    }
    else {
        statusLine = "HTTP/1.0 404 Not Found" + CRLF ;
        contentTypeLine = "text/html" ;
        entityBody = "<HTML>" + "<HEAD><TITLE>404 Not Found</TITLE></
HEAD>" +
        "<BODY>404 Not Found"+ "<br>usage:http://yourHostName:port/" + "fileName.html</
BODY></HTML>" ;
    }
    //发送到服务器信息
    output.write(statusLine.getBytes());
    output.write(serverLine.getBytes());
    output.write(contentTypeLine.getBytes());
    output.write(contentLengthLine.getBytes());
    output.write(CRLF.getBytes());
    //发送信息内容
    if (fileExists){
        sendBytes(fis, output) ;
        fis.close();
    }
    else{ output.write(entityBody.getBytes());}
}
}
//关闭套接字和流
try {
    output.close();
    br.close();
    socket.close();

```



```
}  
catch(Exception e) {}  
}  
private static void sendBytes(FileInputStream fis, OutputStream os) throws Exception{  
    //创建一个1KB的buffer  
    byte[] buffer = new byte[1024] ;  
    int bytes = 0 ;  
    //将文件输出到套接字输出流中  
    while ((bytes = fis.read(buffer)) != -1 ){  
        os.write(buffer, 0, bytes);  
    }  
    private static String contentType(String fileName){  
        if (fileName.endsWith(".htm") || fileName.endsWith(".html")) {  
            return "text/html";  
        }  
        return "fileName";  
    }  
}
```



案例3：一个简单的HTML浏览器

案例运行效果与操作

本案例实现了简单的HTML浏览器功能。程序运行后，界面如图5-15所示。

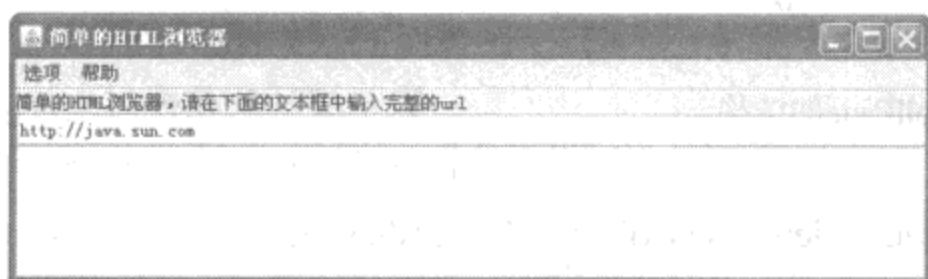


图5-15 运行时界面

在文本框中输入要浏览的网页地址，按Enter键后就会在下方显示出网页内容，界面如图5-16所示。此时，单击网页中的超级链接，就会显示相应链接的网页内容，实现了基本的浏览器功能。

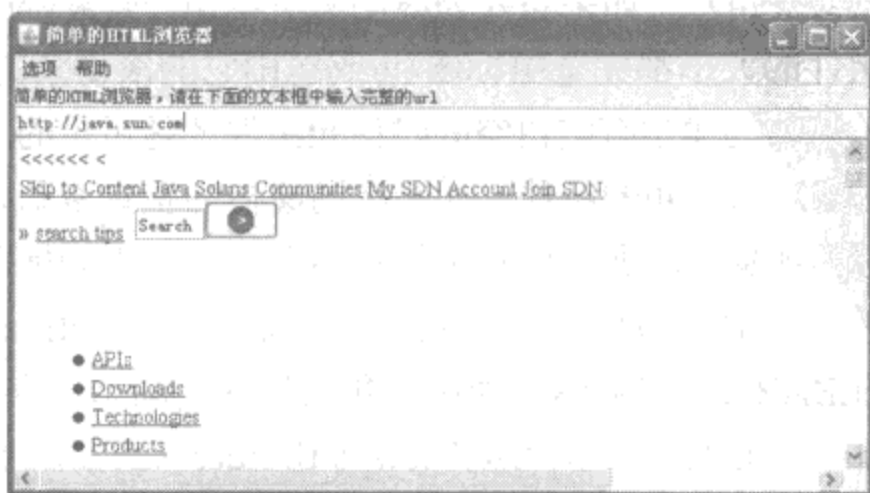


图5-16 显示网页内容

本案例程序包含一个菜单条，其中包括“选项”和“帮助”两个菜单，每个菜单下又分别有一个菜单项，如图5-17所示。单击“退出”菜单项会退出程序，单击“关于”菜单项则弹出“关于”对话框，如图5-18所示。



图5-17 程序中的菜单

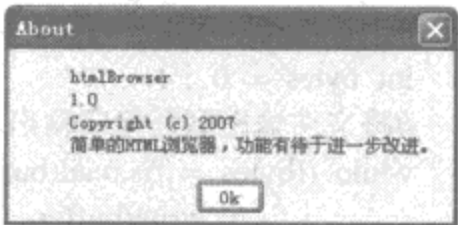


图5-18 “关于”对话框

制作要点

1. JScrollPane类的用法。

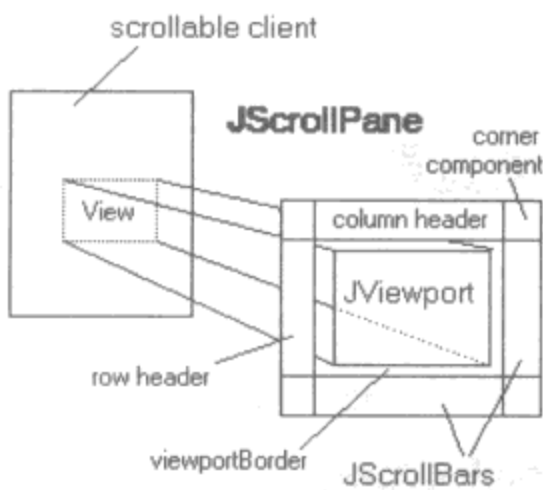


图5-19 JScrollPane的结构

JScrollPane类提供轻量级组件的scrollable视图，JScrollPane管理视口（Viewport）、可选的垂直和水平滚动条以及可选的行和列标题视口。注意，JScrollPane不支持重量级组件。

JViewport为数据源提供一个窗口或视口，例如一个文本文件。该数据源为由JViewport视图显示的“scrollable客户端”（即数据模型）。JScrollPane基本上由JScrollBars、一个JViewport以及它们之间的连线组成，如图5-19所示。

除了滚动条和视口之外，JScrollPane也可以有一个列标题和一个行标题。这二者都是JViewport

对象，可用setRowHeaderView和setColumnHeaderView指定。列标题视口自动左右滚动，跟踪主视口的左右滚动（但是它不会垂直滚动）。行标题的滚动方式与此类似。

在两个滚动条的交汇处、行标题与列标题的交汇处，或者滚动条与其中一个标题的交汇处，两个组件在接近角的地方停止，留下一个默认情况下为空的矩形空间。四个角都有可能存在这些空间。在图5-19中，右上角存在该空间，由标签“角组件”标识。

可使用setCorner方法替换许多的这些空白空间，以便将组件添加到一个特定角（注意，相同的组件不能添加到多个角）。如果想要为滚动窗格增加一些额外的装饰或功能，那么此方法很有用。每个角组件的大小都完全由标题和/或包围它的滚动条的大小确定。

只有角组件存在于其中的角中有空白空间时该角组件才是可见的。例如，设想一个设置在滚动窗格（带有列标题）右上角的组件，如果滚动窗格的垂直滚动条不存在（可能因为视图组件尚未大到需要它的地步），那么该角组件将不会显示（因为标题和垂直滚动条的交汇点所创建的角中没有空白空间）。要强制使滚动条始终显示，可使用 setVerticalScrollBarPolicy（VERTICAL_SCROLLBAR_ALWAYS）确保该角组件的空间始终存在。

要围绕主视口添加一个边界，可使用setViewportBorder（当然，也可以使用setBorder 围绕整个滚动窗格添加一个边界）。

应该执行的一个常见操作是设置背景颜色，此颜色可在主视口小于视口或透明时使用。使用scrollPane.getViewport().setBackground()可以设置视口的背景色。设置视口而不是滚动窗

格的颜色的原因是，默认情况下，JViewport为不透明，还有一些其他属性，这意味着它将其背景色完全填充背景。因此，当JScrollPane绘制其背景时，视口通常将在它上面绘制。

默认情况下，JScrollPane使用ScrollPaneLayout处理其子组件的布局。ScrollPaneLayout使用两个方法之一确定视口视图的大小：如果视图实现了Scrollable，将使用getPreferredScrollableViewportSize、getScrollableTracksViewportWidth和getScrollableTracksViewportHeight的组合，否则使用getPreferredSize。

```
pane.getViewport().add(jEditorPaneBrowser);
```

2. 事件处理。
3. 常量字段的用法。

步骤详解

1. 设计自己的“皮肤”。

```
UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
```

本案例将自己的Look And Feel设置为本地系统的Look And Feel。

从JDK 1.1.3开始，Sun提供了三个Look And Feel的子类javax.swing.plaf.metal.MetalLookAndFeel、com.sun.java.swing.plaf.motif.MotifLookAndFeel和com.sun.java.swing.plaf.windows.WindowsLookAndFeel，分别提供了“Metal”、“Motif”与“Windows”的界面式样。也就是说，任何基于Swing的界面程序本身都可以使用三种系统提供的皮肤。感兴趣的读者也可以直接或间接继承Look And Feel类，自己编写一种“皮肤”。Skin Look And Feel是开放源代码的产品，它的源代码可以在<http://www.l2fprod.com/>上全部找到。Skin Look And Feel本身还可以更换<http://www.l2fprod.com/>提供的各种“皮肤”。

UIManager类是Swing界面管理的核心，用于管理Swing的小应用程序以及应用程序样式的状态。UIManager类提供了许多静态方法用于更换与管理Look and Feel。

本案例中通过static String getSystemLookAndFeelClassName()，返回与当前系统相关的本地系统Look and Feel，如果没有实现本地Look and Feel则返回默认的跨平台的Look and Feel。再使用static void setLookAndFeel(LookAndFeel newLookAndFeel)设置当前使用的Look And Feel。

其他常用的方法如下：

- static void addAuxiliaryLookAndFeel(LookAndFeel laf)：增加一个“Look And Feel”到辅助的“look and feels”列表；
- static LookAndFeel[] getAuxiliaryLookAndFeels()：返回辅助的“look and feels”列表（可能为空）；
- static String getCrossPlatformLookAndFeelClassName()：返回默认的实现了跨平台的Look and Feel——即Java Look and Feel (JLF)；
- static UIManager.LookAndFeelInfo[] getInstalledLookAndFeels()：返回了在目前已经安装的Look And Feel的信息；
- static LookAndFeel getLookAndFeel()：返回当前使用的Look and Feel；

- `static void installLookAndFeel(String name, String className)`: 创建一个新的Look and Feel并安装到当前系统;
- `static void installLookAndFeel(UIManager.LookAndFeelInfo info)`: 创建一个新的Look and Feel并安装到当前系统;
- `static boolean removeAuxiliaryLookAndFeel(LookAndFeel laf)`: 从辅助的“look and feels”列表删除一个“Look And Feel”;
- `static void setInstalledLookAndFeels(UIManager.LookAndFeelInfo[] infos)`: 设置当前的已安装Look and Feel信息。

2. 显示主界面。

前面的案例中多次用到布局设计，有时候布局需要能够将控件按照X轴（从左到右）或者Y轴（从上到下）方向来摆放，而且沿着主轴能够设置不同尺寸。`setAlignmentX()`和`setAlignmentY()`方法能实现这个功能：

```
jLabelDescription.setAlignmentX(0.5F);
```

添加事件侦听器，打开输入的URL地址页面：

```
jTextFieldInput.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            //打开
            jEditorPaneBrowser.setPage(jTextFieldInput.getText());
        }
        catch(IOException ex) { ... }
    }
});
```

在编辑器面板添加超链接事件侦听器：

```
public void hyperlinkUpdate(final HyperlinkEvent e) {
    if(e.getEventType() == javax.swing.event.HyperlinkEvent.EventType.ACTIVATED)
        SwingUtilities.invokeLater(new Runnable() { //创建线程
            public void run() { //线程的运行方法
                Document doc = jEditorPaneBrowser.getDocument();
                try {
                    URL url = e.getURL();
                    jEditorPaneBrowser.setPage(url); //打开链接页面
                    jTextFieldInput.setText(url.toString());
                }
                catch(IOException io) { ... }
            }
        });
}
```

3. 常量字段的用法。

```
super.processWindowEvent(e);
if(e.getID() == 201) //用户要求窗口管理程序关掉窗口（常量字段）
    System.exit(0); //系统退出
```

指示窗口状态改变的低级别事件。当打开、关闭、激活、停用、图标化或取消图标化Window对象时，或者焦点转移到Window内或移出Window时，由Window对象生成此低级别

事件。常量值为201，表示的窗口事件为“Window_Closing”。在Java标准文档中从常量字段值中都可以查找出常量值对应的动作。如本案例中的“enableEvents(64L);”，常量64L表示用于选择窗口事件的事件掩码WINDOW_EVENT_MASK。

程序源代码与解释

```

/* * HtmlBrowser.java*/
package htmlbrowser;
import java.awt.*;
import javax.swing.UIManager;
public class HtmlBrowser {    //主类
    boolean packFrame;
    public HtmlBrowser() {    //构造方法
        packFrame = false;
        FrameBrowser frame = new FrameBrowser(); //创建FrameBrowser类对象显示主界面
        if(packFrame)
            frame.pack();    //调整主窗口的大小，以适合其子组件的首选大小和布局
        else
            frame.validate(); //验证主窗口及其所有子组件
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        //主窗口居中显示
        if(frameSize.height > screenSize.height)
            frameSize.height = screenSize.height;
        if(frameSize.width > screenSize.width)
            frameSize.width = screenSize.width;
        frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height - frameSize
.height) / 2);
        frame.setVisible(true);
    }
    public static void main(String args[]) {    //主方法
        try {    //设置窗口外观
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
        catch(Exception e) {
            e.printStackTrace();
        }
        new HtmlBrowser();
    }
}

/* * FrameBrowser.java*/
package htmlbrowser;
import java.awt.*;
import java.awt.event.*;
import java.io.IOException;
import java.net.URL;
import javax.swing.*;
import javax.swing.event.HyperlinkEvent;
import javax.swing.event.HyperlinkListener;
import javax.swing.text.Document;
import javax.swing.text.JTextComponent;

public class FrameBrowser extends JFrame { //程序主界面类

    public FrameBrowser() {    //构造方法

```



```

borderLayout1 = new BorderLayout();
jPanel2 = new JPanel();
jLabelDescription = new JLabel();    //程序说明标签
borderLayout2 = new BorderLayout();
jTextFieldInput = new JTextField(); //URL地址文本框
jEditorPaneBrowser = new JEditorPane(); //显示网页内容的编辑器面板
jMenuBar1 = new JMenuBar();    //定义一个菜单条
jMenuOption = new JMenu();    //“选项”菜单
jMenuItemExit = new JMenuItem(); //“退出”菜单项
jMenuHelp = new JMenu();    //“帮助”菜单
jMenuAbout = new JMenuItem(); //“关于”菜单项
enableEvents(64L);    //能够选择窗口事件（常量字段）
try {
    FrameBrowserInit();    //调用FrameBrowserInit方法，完成窗口初始化
}
catch(Exception e) {
    e.printStackTrace();
}
}
private void FrameBrowserInit() throws Exception {    //窗口初始化
protected void processWindowEvent(WindowEvent e) { //处理窗口事件
    super.processWindowEvent(e);
    if(e.getID() == 201) //用户要求窗口管理程序关掉窗口（常量字段）
        System.exit(0); //系统退出
}
void jTextFieldInput_actionPerformed(ActionEvent actionevent) {
}
void jMenuAbout_actionPerformed(ActionEvent e) {    //处理“关于”菜单项Action事件
    AboutDialog about = new AboutDialog(this);    //创建“关于”对话框
    about.setVisible(true);    //显示“关于”对话框
}
void jMenuItemExit_actionPerformed(ActionEvent e) { //处理“退出”菜单项Action事件
    System.exit(0);    //系统退出
}
}

/* * AboutDialog.java*/
package htmlbrowser;
import java.awt.*;
import java.awt.event.*;
import java.util.EventObject;
import javax.swing.*;
//“关于”对话框类
public class AboutDialog extends JDialog implements ActionListener {
    public AboutDialog(Frame parent) {    //带参数构造方法，窗口初始化
        super(parent);
        //定义窗口组件
        panel1 = new JPanel();
        panel2 = new JPanel();
        insetsPanel1 = new JPanel();
        insetsPanel2 = new JPanel();
        insetsPanel3 = new JPanel();
        jButtonOK = new JButton();
        imageLabel = new JLabel();
        jLabelProduct = new JLabel();
        jLabelVersion = new JLabel();
    }
}

```



```

jLabelCopyright = new JLabel();
jLabelDescription = new JLabel();
borderLayout1 = new BorderLayout();
borderLayout2 = new BorderLayout();
flowLayout1 = new FlowLayout();
gridLayout1 = new GridLayout();
product = "htmlBrowser";
version = "1.0";
copyright = "Copyright (c) 2007";
Description = "简单的HTML浏览器，功能有待于进一步改进。";
enableEvents(64L);           //用于选择窗口事件的事件掩码WINDOW_EVENT_MASK
(常量字段)

try {    //窗口组件初始化
    //...
}
catch(Exception e) {
    e.printStackTrace();
}
pack();
Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
Dimension dialogSize = this.getSize();
//对话框居中显示
if(dialogSize.height > screenSize.height)
    dialogSize.height = screenSize.height;
if(dialogSize.width > screenSize.width)
    dialogSize.width = screenSize.width;
this.setLocation((screenSize.width - dialogSize.width) / 2, (screenSize.height - dialogSize
.height) / 2);
}
protected void processWindowEvent(WindowEvent e) { //处理窗口事件
    if(e.getID() == 201) //用户要求窗口管理程序关掉窗口（常量字段）
        dispose();      //关闭窗口，释放资源
    super.processWindowEvent(e);
}
public void actionPerformed(ActionEvent e) { //处理Action事件
    if(e.getSource() == jButtonOK) //用户单击“OK”按钮（常量字段）
        dispose(); //关闭窗口，释放资源
}
}


```



案例4：用JavaMail发送邮件

案例运行效果与操作

本案例利用Sun公司发布的用来处理E-mail的JavaMail API发送邮件。这是一个Web应用程序，需要Web服务器进行服务。为简便起见，本案例使用了NetBeans自带的Tomcat 5.5作为Web服务器。另外，在运行前，申请了两个免费邮箱作为测试邮箱，其中申请mailtest_java@sohu.com邮箱是为了在程序中使用搜狐的smtp服务器smtp.sohu.com发送信件，申请mailtest_java@126.com邮箱则是作为收信邮箱，程序运行完毕后要到该邮箱查看是否发送成功。本案例的开发运行环境为NetBeans。

在NetBeans窗口中单击  按钮，运行程序。运行后，在NetBeans窗口的输出窗口中出现如下代码：

```
2007-9-22 16:29:36 org.apache.catalina.core.StandardContext reload
信息: Reloading this Context has started
DEBUG: setDebug: JavaMail version 1.4ea
DEBUG: getProvider() returning javax.mail.Provider[TRANSPORT,smtp,com.sun.mail.smtp
.SMTPTransport,Sun Microsystems, Inc]
DEBUG SMTP: useEhlo true, useAuth true
DEBUG SMTP: useEhlo true, useAuth true
DEBUG SMTP: trying to connect to host "smtp.sohu.com", port 25, isSSL false
220 smtp.sohu.com ESMTP Postfix
DEBUG SMTP: connected to host "smtp.sohu.com", port: 25

EHLO Myhomepc1
250-smtp.sohu.com
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-AUTH LOGIN
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
DEBUG SMTP: Found extension "PIPELINING", arg ""
DEBUG SMTP: Found extension "SIZE", arg "10240000"
DEBUG SMTP: Found extension "VRFY", arg ""
DEBUG SMTP: Found extension "ETRN", arg ""
DEBUG SMTP: Found extension "AUTH", arg "LOGIN"
DEBUG SMTP: Found extension "ENHANCEDSTATUSCODES", arg ""
DEBUG SMTP: Found extension "8BITMIME", arg ""
DEBUG SMTP: Found extension "DSN", arg ""
DEBUG SMTP: Attempt to authenticate
AUTH LOGIN
334 VXNlcm5hbWU6
bWFpbHRlc3RfamF2YQ==
334 UGFzc3dvcmQ6
amF2YW1haWw=
235 2.0.0 Authentication successful
DEBUG SMTP: use8bit false
MAIL FROM:<mailto:java@sohu.com>
250 2.1.0 Ok
RCPT TO:<mailto:java@126.com>
250 2.1.5 Ok
DEBUG SMTP: Verified Addresses
DEBUG SMTP: mailtest_java@126.com
DATA
354 Send from Rising mail proxy
Date: Sat, 22 Sep 2007 16:29:37 +0800 (CST)
From: mailtest_java@sohu.com
To: mailtest_java@126.com
Message-ID: <1419599.01190449777156.JavaMail.linda@Myhomepc1>
Subject: =?GBK?B?t6LLzdPKvP6y4srU?=
MIME-Version: 1.0
```



```

Content-Type: multipart/mixed;
    boundary="-----=_Part_0_24480977.1190449777140"

-----=_Part_0_24480977.1190449777140
Content-Type: text/plain; charset=GBK
Content-Transfer-Encoding: base64

08NKYXZhbWFpbLeiy821xNPKvP4=
-----=_Part_0_24480977.1190449777140
Content-Type: text/plain; charset=GBK; name=test.txt
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=test.txt

suLK1Li9vP4=
-----=_Part_0_24480977.1190449777140--

250
QUIT
221 2.0.0 Bye

```

程序运行完毕后登录mailtest_java@126.com邮箱，可以发现收到了一封邮件，如图5-20所示。打开该邮件查看其内容及其附件，如图5-21所示，可以看到确实是从mailtest_java@sohu.com发送的测试邮件，表明程序运行成功。



图5-20 收到新邮件



图5-21 查看新邮件的内容

制作要点

1. JavaMail的用法。

JavaMail是Sun公司发布的用来处理E-mail的API。它可以方便地执行一些常用的邮件传输，支持POP3、IMAP、SMTP协议。虽然JavaMail是Sun公司发布的API之一，但它目前还没有被加入标准的Java开发工具包中（Java Development Kit），因此在使用前必须另外下载JavaMail文件。除此以外，JavaMail的运行必须得依赖于Sun公司JavaBeans Activation Frame-

work (JAF) 的支持，可以从java.sun.com网站上下载javamail.zip和jaf.zip两个压缩文件。

(1) JavaMail需要创建一个格式为“mail.smtp.host”的文件用来发送信息。

```
Properties props = new Properties ();  
props.put("mail.smtp.host", "smtp.jspinsider.com");
```

(2) 所有的基于JavaMail的程序都至少需要一个或全部的对话目标。

```
Session sendMailSession;  
sendMailSession = Session.getInstance(props, null);
```

(3) 传输。

邮件的传输只有送出或收到两种状态。JavaMail将这两种不同状态描述为传输和储存。传输将送出邮件，而储存将收取邮件。

```
Transport transport;  
transport = sendMailSession.getTransport("smtp");
```

(4) 信息对象。

信息对象将把所发送的邮件真实地反映出来。

```
Message newMessage = new MimeMessage(sendMailSession);
```

这就是我们所需要的全部四个对象，下一步是如何将对象加入到JSP中。

2. Authenticator类的用法。

当用浏览器在网上冲浪时，遇到要求代理服务器认证或HTTP服务器认证的URL，会出现要求输入用户名及口令的窗口；在收发E-mail时，也会要求输入用户名和密码。Java提供了一个Authenticator类，可以用来进行身份验证。应用程序通过重写子类中的getPasswordAuthentication()方法使用此类，此方法通常使用各种getXXX()访问器方法获取关于请求验证的实体的信息，然后通过用户交互或者其他非交互手段获取用户名和密码，之后凭据将以PasswordAuthentication返回值的形式返回。

接下来通过调用setDefaultAuthenticator()向系统注册此具体子类的实例。需要进行验证时，系统将调用其中一个requestPasswordAuthentication()方法，这些方法将依次调用注册对象的getPasswordAuthentication()方法。

请求验证的所有方法都有一个失败的默认实现。

```
public PasswordAuthentication getPasswordAuthentication() {  
    return new PasswordAuthentication(this.username, this.password);  
}
```

步骤详解

1. 选择Web类别，新建Web应用程序，使用NetBeans捆绑的Tomcat 5.5作为Web服务器，界面如图5-22所示。

2. 安装、配置JavaMail和JAF。

从java.sun.com网站下载两个Zip包（JavaMail和Jaf），将两个ZIP文件解压到硬盘某个位置。添加JAR文件夹，选择刚才解压的文件位置，选中【javamail-1.4】文件夹下的mail.jar文件，将JavaMail API导入。同样，选中【jaf-1.1】文件夹下的activation.jar文件，将JavaBeans Activation

Framework导入。这样就完成了JavaMail和JAF的安装和配置。

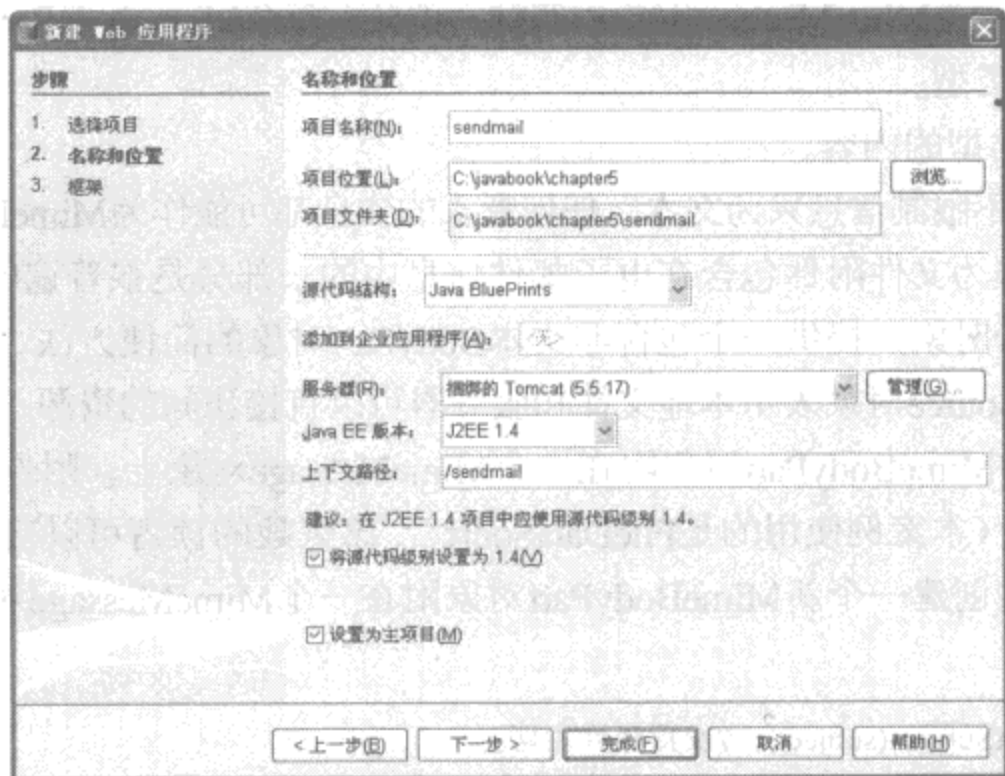


图5-22 “新建Web应用程序”对话框

3. 创建文件，用来发送消息。

JavaMail需要Properties来创建一个session对象。它将寻找字符串“mail.smtp.host”，属性值就是发送邮件的主机，本案例的smtp定义为smtp.sohu.com。

```
Properties props=System.getProperties();
props.put("mail.smtp.host",smtp);
```

需要身份验证，设置身份验证为真：

```
props.put("mail.smtp.auth","true");
```

这个Session类代表JavaMail中的一个邮件session。每一个基于JavaMail的应用程序至少有一个session，也可以有任意多的session。获得邮件会话对象：

```
Session sess=Session.getInstance(props,sa);
```

设置发信人：

```
msg.setFrom(new InternetAddress(from));
```

设置邮件，Message对象将存储实际发送的电子邮件信息，Message对象被作为一个Mime-Message对象来创建，并且需要知道应当选择哪一个JavaMail session，也因为Mime类型太多，编写收邮件的程序时最重要和最难的要数解析附件和邮件正文的部分。本案例非常简单，将Mime类型限制在“text/html; charset=gb2312”。

```
Message msg = new MimeMessage(sess);
msg.setDataHandler(new DataHandler(body,"text/html; charset=gb2312"));
```

电子邮件内容的容器是Multipart抽象类，它定义了增加和删除及获得电子邮件不同部分内容的方法。由于Multipart是抽象类，必须为它使用一个具体的子类，JavaMail API提供javax.mail.internet.MimeMultipart类来使用MimeMessage对象。使用MimeMultipart对象的一个方法是addBodyPart()，它在电子邮件内容里添加BodyPart对象。消息可以有很多部分，一个Body-

Part可以代表一个部分。MimeBodyPart是BodyPart具体用于MimeMessage的一个子类。MimeBodyPart对象代表一个MimeMessage对象内容的一部分，每个MimeBodyPart被认为有两部分：

- 一个Mime类型。
- 匹配这个类型的内容。

JavaMail API不限制信息只为文本，任何形式的信息都可能作为MimeMessage的一部分。除了文本信息，作为文件附件包含在电子邮件信息中的一部分是很普遍的。JavaMail API通过使用DataHandler对象，提供一个包含非文本BodyPart对象的简便方法。

一个FileDataSource对象表示本地文件和服务器可以直接访问的资源。一个本地文件可以通过创建一个新的MimeBodyPart对象附在一个MimeMessage对象上。附件还可以使用远程资源URLDataSource（本案例使用的是FileDataSource，感兴趣的读者可以自己试一试），一个远程资源可以通过创建一个新MimeBodyPart对象附在一个MimeMessage对象上（同FileDataSource差不多）。

```
msg.setSubject(subject); //设置邮件主题
MimeBodyPart mbp1 = new MimeBodyPart(); //正文
mbp1.setText(body);
MimeBodyPart mbp2 = new MimeBodyPart(); //附件
FileDataSource fds = new FileDataSource(filename);
mbp2.setDataHandler(new DataHandler(fds));
mbp2.setFileName(fds.getName());
//整个邮件（正文 + 附件）
Multipart mp = new MimeMultipart();
mp.addBodyPart(mbp1);
mp.addBodyPart(mbp2);
msg.setContent(mp); //添加Multipart到Message中
```

添加收信人，创建了Session和Message，并将内容填入消息后，就可以用Address确定信件地址了。和Message一样，Address也是个抽象类，使用的是Javax.mail.internet.InternetAddress类：

```
msg.setRecipients(Message.RecipientType.TO,InternetAddress.parse(to,false));
```

发送邮件，邮件既可以被发送也可以被收到。JavaMail使用了两个不同的类来完成这两个功能：Transport和Store。Transport是用来发送信息的：

```
Transport.send(msg);
```

4. SMTP身份认证，这是防止垃圾邮件中继转发的办法，虽然SMTP本身有缺陷，但这是最基本的身份验证。

```
public SmtplibAuthenticator(String username, String password) {
    this.username = username;
    this.password = password;
}
public PasswordAuthentication getPasswordAuthentication() {
    return new PasswordAuthentication(this.username, this.password);
}
String username = null;
String password = null;
```


5. 编辑缺省的JSP文件，展开sendmail项目节点和“Web页”节点。请注意，IDE已创建了缺省JSP文件index.jsp。创建项目时，IDE在源代码编辑器中已打开index.jsp文件，选择index.jsp源代码编辑器标签。现在，index.jsp文件在源代码编辑器中具有焦点。在源代码编辑器右侧的组件面板（如图5-23所示）中，展开“JSP”并将一个“使用Bean”项拖到源代码编辑器中紧靠<body>标记的下面。此时会弹出【插入使用Bean】对话框，设置ID为send，类为sendmail.SendMail，范围为page，单击“确定”按钮将在<body>标记下面添加“使用Bean”。JSP代码如下：

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
    <title>用JavaMail发送邮件</title>
  </head>
  <body>
    <h1>用Javamail发送邮件测试</h1>
    <jsp:useBean id="send" scope="page" class="sendmail.SendMail" />
    <%
      if(send.SendMail("发送邮件测试","用JavaMail发送的邮件","mailto:java@126.com",
"C:\\test.txt"))
        out.println("发送成功！");
      else
        out.println("发送失败！");
    %>
  </body>
</html>
```

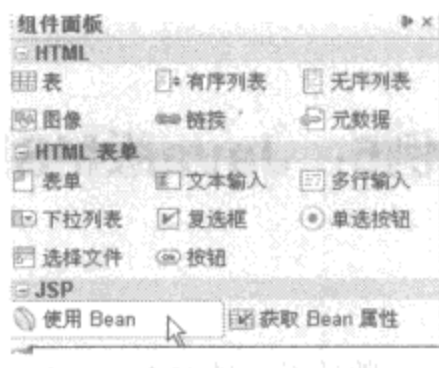


图5-23 组件面板

程序源代码与解释

```
/* * SendMail.java */
public class SendMail {
    private String smtp="smtp.sohu.com",from="mailto:java@sohu.com";
    //发送一个邮件
    public boolean SendMail(String subject, String body, String to, String filename){
        try {
            Smtputhicator sa=new Smtputhicator("mailto:java","javamailtest");
            Properties props=System.getProperties(); //获得系统属性
            props.put("mail.smtp.auth","true"); //设置身份验证为真，若需身份验证则必须设为真
            props.put("mail.smtp.host",smtp); //设置SMTP主机
            Session sess=Session.getInstance(props,sa); //获得邮件会话对象
            sess.setDebug(true);
```



```

Message msg = new MimeMessage(sess); //创建Mime邮件对象
msg.setDataHandler(new DataHandler(body,"text/html; charset=gb2312")); //设置邮
件内容

if(filename.equals("") || filename == null || filename.equals("null")) {
    msg.setFrom(new InternetAddress(from)); //设置发信人
    msg.setRecipients(Message.RecipientType.TO,InternetAddress.parse(to,false));
//设置收信人

    msg.setSubject(subject);//设置邮件主题
}else{
    msg.setFrom(new InternetAddress(from)); //设置发信人
    //添加收信人
    msg.addRecipient(javax.mail.Message.RecipientType.TO, new InternetAddress
(to));

    //具体邮件设置，参见步骤详解3
}
msg.setSentDate(new Date()); //发送日期
Transport.send(msg); //发送邮件
return true; //发送成功
}
catch (Exception e) {
    return false; //发送失败
}
}

/* * Smtput authenticator.java*/
class Smtput authenticator extends Authenticator {
    //SMTP身份验证，参见步骤详解4
}

```



案例5: Java版MSN

案例运行效果与操作

即时通信软件有腾讯公司的QQ，微软公司的MSN Messenger、Skype等。MSN Messenger是由微软公司开发的即时通信软件，凭借其Windows操作系统和整个微软产品家族的紧密结合，具有简单实用、性能稳定、世界通用等特点，而且让开发者雀跃的是该软件同时也提供了开放的API以及开放的通信协议。jMSN就是一个韩国人开发的开放源码的API，可以从<http://sourceforge.net/projects/jmsn/>站点上下载，它是封装了MSN Messenger开放的通信协议的Java API，通过这个API，开发者完全可以使用Java语言模拟出MSN Messenger软件。

本案例使用jMSN实现了简单的MSN文字聊天功能。生成项目后在DOS窗口运行java -jar javamsn.jar后，JavaMSN自动进行登录，界面如图5-24所示。

本案例实现的功能比较简单，它可以显示好友的上、下线信息；当有人把它加入好友时，程序自动将之加为好友；当有人给它发送信息时，程序自动回复一条相同的信息。如图5-25所示为程序运行过程中的部分截图界面。


```
C:\javabook\chapter5\javamsn\dist>java -jar javamsn.jar
正在登录....
java_test@hotmail.com 登录成功!
```

图5-24 运行时界面

```
C:\javabook\chapter5\javamsn\dist>java -jar javamsn.jar
正在登录....
java_test@hotmail.com 登录成功!
zhziguohotmail.com:zhziguohotmail.com <FLN> Add me.
用户 zhziguohotmail.com 加我为好友。
用户 zhziguohotmail.com 下线了。
用户 Ziguoh 上线了。
接收到消息, Ziguoh说: 来测试一下JavaMSN吧!
Ziguoh正在输入信息...
接收到消息, Ziguoh说: 还可以了。
zhziguohotmail.com remove me.
用户 zhziguohotmail.com 把我从好友列表中删除。
JavaMSN下线成功!
```

图5-25 运行中的截图界面

制作要点

1. jMSN的使用。

jMSN是一个韩国人开发的开放源码的API, 可以跨平台使用。msnm-lib、jMSN和MSN系统之间的关系如图5-26所示。可以发现, 通过msnm-lib来完成与MSN服务器之间的通信而不需要去操心具体的通信协议的细节。事实上, msnm-lib做了更多的事情, 使得使用msnm-lib来开发一个MSN应用程序变得非常简单。

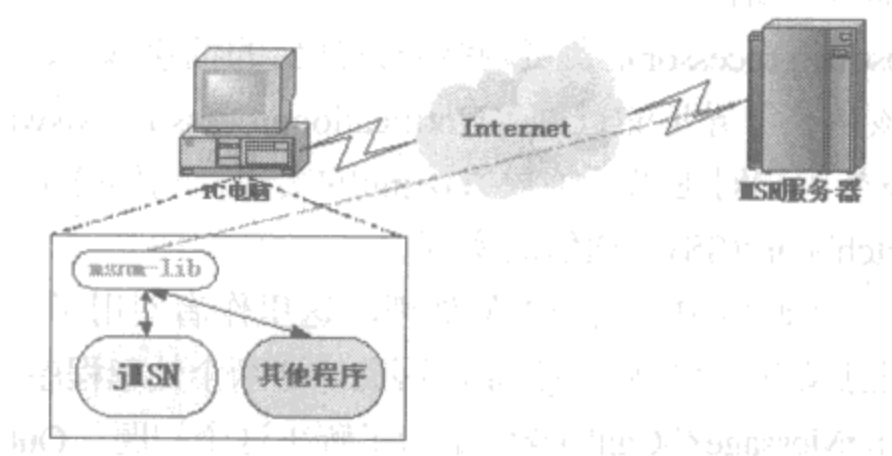


图5-26 msnm-lib、jMSN和MSN系统之间的关系

除了两个常用的对象MsnFriend和MimeMessage分别用来表示我的好友以及MSN信息外, 其他需要了解的也就是MSNMessenger以及MsnAdapter了。当然, 前提是不需要除了聊天以外的其他功能, 例如文件传输等。MSNMessenger类对应着一个账号的一次登录会话, 只需告诉MSNMessenger类登录所用的账号、密码、登录后的初始状态以及怎么来处理从MSN服务器上接收到的任何信息。在msnm-lib中, 处理MSN信息是通过一个叫MsnAdapter类来处理的, 这个类定义了如何处理例如收到消息、有人加我为好友等的事件, 开发人员可以重载这些方法以便自行处理。自行扩展MsnAdapter的类必须告诉MSNMessenger实例, 所以需要添加侦听器msn.addMsnListener(new MsnAdapter(msn))。自行扩展MsnAdapter的类是用来处理被动消息的, 例如有人给我发消息等。当我们要发送消息给别人时就需要用到MSNMessenger的实例, 这也就是我们为什么要把MSNMessenger的实例传递给MsnAdapter的原因, 因为当我们接收到任何消息时要给发送人回复一条相同的信息。

```
msn = new MSNMessenger("java_test@hotmail.com", "123456"); //用户名和密码
msn.setInitialStatus(UserStatus.ONLINE); //初始状态为联机
```



```

msn.addMsnListener(new MSNAdapter(msn)); //添加侦听器
msn.login(); //登录
//发送相同的回复信息给发送者
messenger.sendMessage(friend.getLoginName(), mime);
messenger.addFriend(friend.getLoginName()); //添加好友
messenger.removeFriend(friend.getLoginName()); //删除好友

```

2. 多线程的使用。

在正常线程之外，增加一个线程来捕捉Ctrl+C的输入以便注销MSN的登录。

```
Runtime.getRuntime().addShutdownHook(new JavaMSN());
```

步骤详解

1. 初始状态：

```

msn = new MSNMessenger("java_test@hotmail.com", "123456"); //用户名和密码
msn.setInitialStatus(UserStatus.ONLINE); //初始状态为联机
msn.addMsnListener(new MSNAdapter(msn)); //添加侦听器
msn.login(); //登录

```

2. 增加一个线程来捕捉Ctrl+C的输入以便注销MSN的登录：

```
Runtime.getRuntime().addShutdownHook(new JavaMSN());
```

下面对JMSn类库简单介绍一下。

在JMSn lib中，abstractprocessor是负责网络协议通信的抽象基类，同时继承了thread。同服务器对话的过程应该是一个异步的过程。Notificationprocesso类和switchboardsession类分别继承了abstractprocessor类，同时提供了抽象方法init的实现。它们分别对应MSN协议中notification server(DS)和switchboard(SS)的通信的实现。

Callback类实现了网络通信中一问一答的处理，这里作者使用了回调技术。在MSN通信中，发送消息和接收消息没有必然的先后关系，因此，这两个处理程序必须要异步。JMSn Lib的OutMessage、IncomingMessage和Callback就是为了解决这个问题。OutMessage存储了消息的组成部分，另外存放了回调的处理函数（针对这个消息，服务器回答消息由哪个函数来处理）。服务器回答的消息，通过IncommingMessage来解析。还有认证过程中，数据信息处理准备的标准算法TWIN和MD5。

程序源代码与解释

```

/** JavaMSN.java */
package javamsn;
import rath.msnm.MSNMessenger;
import rath.msnm.SwitchboardSession;
import rath.msnm.UserStatus;
import rath.msnm.entity.MsnFriend;
import rath.msnm.event.MsnAdapter;
import rath.msnm.msg.MimeMessage;
public class JavaMSN extends Thread {
    private static MSNMessenger msn;
    public static void main(String[] args) { //主方法
        msn = new MSNMessenger("java_test@hotmail.com", "123456"); //用户名和密码
    }
}

```



```

        msn.setInitialStatus(UserStatus.ONLINE);           //初始状态为联机
        msn.addMsnListener(new MSNAdapter(msn));          //添加侦听器
        msn.login();   //登录
        System.out.println("正在登录....");
        //增加一个线程来捕捉Ctrl+C的输入以便注销MSN的登录
        Runtime.getRuntime().addShutdownHook(new JavaMSN());
    }
    public void run() { //用户中止程序执行的线程运行方法
        msn.logout(); //下线
        System.out.println("JavaMSN下线成功!");
    }
}

class MSNAdapter extends MsnAdapter { //MSN消息事件处理类
    MSNMessenger messenger;
    public MSNAdapter(MSNMessenger messenger) { //构造方法
        this.messenger = messenger;
    }
    //某人正在输入信息
    public void progressTyping(SwitchboardSession ss,MsnFriend friend,String typingUser) {
        System.out.println(friend.getFriendlyName() + "正在输入信息...");
    }
    //收到消息的时候执行该方法
    public void instantMessageReceived(SwitchboardSession ss,MsnFriend friend,MimeMessage
mime) {
        System.out.print("接收到消息: " + friend.getFriendlyName() + "说: ");
        System.out.println(mime.getMessage());
        try {
            //发送相同的回复信息给发送者
            messenger.sendMessage(friend.getLoginName(), mime);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    //登录成功后执行该方法
    public void loginComplete(MsnFriend own) {
        System.out.println(own.getFriendlyName() + " 登录成功!");
    }
    //登录失败后执行该方法
    public void loginError(String header) {
        System.out.println("登录失败: " + header);
    }
    //好友离线时执行该方法
    public void userOffline(String loginName) {
        System.out.println("用户 " + loginName + " 下线了。");
    }
    //好友上线时执行该方法
    public void userOnline(MsnFriend friend) {
        System.out.println("用户 "+friend.getFriendlyName()+" 上线了。");
    }
    //有人加我为好友时执行
    public void whoAddedMe(MsnFriend friend) {
        System.out.println("用户 " + friend.getFriendlyName() + " 加我为好友。");
        try {
            messenger.addFriend(friend.getLoginName()); //添加好友

```



```
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}  
//有人把我从好友列表中删除时执行  
public void whoRemovedMe(MsnFriend friend) {  
    System.out.println("用户 "+friend.getFriendlyName()+" 把我从好友列表中删除。");  
    try {  
        messenger.removeFriend(friend.getLoginName()); //删除好友  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}  
}
```

案例6: Java实现HTTP队列下载

案例运行效果与操作

本案例是用Java实现的简单下载工具，可以批量下载HTTP资源文件。程序运行后，输出窗口如图5-27所示。下载的文件存放在当前目录下，如图5-28所示。

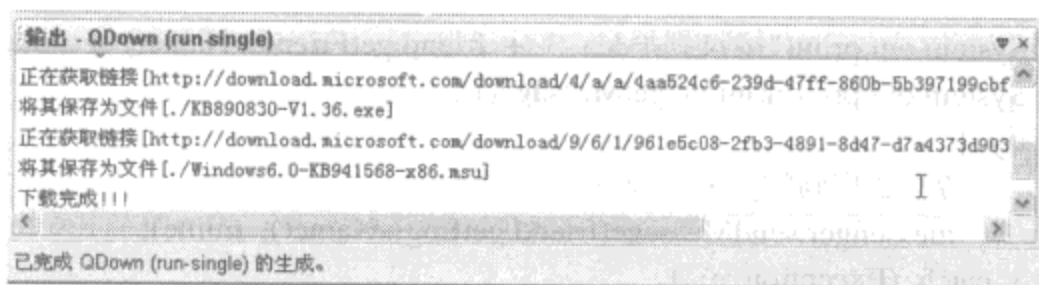


图5-27 程序输出窗口



图5-28 下载的文件

制作要点

1. HttpURLConnection的使用。

Java中可以使用HttpURLConnection来请求Web资源。HttpURLConnection是基于HTTP协议的，其底层通过socket通信实现。

2. 代理服务器设置。

许多上网方式是通过代理服务器实现的，有的网络中也使用了防火墙，不允许直接连接，

这样就得把代理服务器以及端口号的详细信息告诉Java，才能访问到防火墙外的主机。可以通过定义一些HTTP或者FTP属性来做到：

```
http.proxyHost (default: )
http.proxyPort (default: 80 if http.proxyHost specified)
http.nonProxyHosts (default: )
```

http.proxyHost和http.proxyPort用来指定HTTP协议处理器需要使用的代理服务器和端口号。http.nonProxyHosts用来指定哪些主机是直接连接的（即不通过代理服务器来连接）。http.nonProxyHosts属性的值是一个由“|”分隔开的主机列表，它可以使用正则表达式来表示所匹配的主机。

```
ftp.proxyHost (default: )
ftp.proxyPort (default: 80 if ftp.proxyHost specified)
ftp.nonProxyHosts (default: )
```

ftp.proxyHost和ftp.proxyPort用来指定FTP协议处理器需要使用的代理服务器和端口号。ftp.nonProxyHosts用来指定哪些主机是直接连接的，指定的方法与http.nonProxyHosts类似。

步骤详解

1. 建立网络连接，客户端发出连接一个URL请求。

```
url = new URL(destUrl);
httpUrl = (HttpURLConnection) url.openConnection();
//连接指定的资源
httpUrl.connect();
```

2. 服务器解析URL，并将指定的资源返回一个输入流给客户。

```
bis = new BufferedInputStream(httpUrl.getInputStream());
```

3. 客户端接收输入流，将流中的内容存到文件。

```
fos = new FileOutputStream(fileName);
if (this.DEBUG)
    System.out.println("正在获取链接[" + destUrl + "]的内容...\n将其保存为文件[" + fileName + "]);
//保存文件
while ( (size = bis.read(buf)) != -1)
    fos.write(buf, 0, size); .....
```

4. 关闭连接。操作完成后，应该调用disconnect()方法关闭连接，否则在大量访问后，会出现打开太多文件的错误。

```
httpUrl.disconnect();
```

程序源代码与解释

```
import java.io.*;
import java.net.*;
import java.util.*;
public class Quene {
    public final static boolean DEBUG = true; //调试用
    private static int BUFFER_SIZE = 8096; //缓冲区大小
    private Vector vDownload = new Vector(); //URL列表
```



```

private Vector vFileList = new Vector();    //下载后的保存文件名列表
//构造方法
public Queue() { }
//清除下载列表
public void resetList() {
    vDownload.clear();
    vFileList.clear();
}
//增加下载列表项
public void addItem(String url, String filename) {
    vDownload.add(url);
    vFileList.add(filename);
}
//根据列表下载资源
public void downLoadByList() {
    String url = null;
    String filename = null;
    //按列表顺序保存资源
    for (int i = 0; i < vDownload.size(); i++) {
        url = (String) vDownload.get(i);
        filename = (String) vFileList.get(i);
        try { saveToFile(url, filename); }
        catch (IOException err) {
            if (DEBUG) { System.out.println("资源[" + url + "]下载失败!!!"); }
        }
    }
    if (DEBUG) { System.out.println("下载完成!!!"); }
}
//将HTTP资源另存为文件
public void saveToFile(String destUrl, String fileName) throws IOException {
    FileOutputStream fos = null;
    BufferedInputStream bis = null;
    HttpURLConnection httpUrl = null;
    URL url = null;
    byte[] buf = new byte[BUFFER_SIZE];
    int size = 0;
    //建立连接
    url = new URL(destUrl);
    httpUrl = (HttpURLConnection) url.openConnection();
    //连接指定的资源
    httpUrl.connect();
    //获取网络输入流
    bis = new BufferedInputStream(httpUrl.getInputStream());
    //建立文件
    //参见步骤详解4
    fos.close();
    bis.close();
    httpUrl.disconnect();
}
//设置代理服务器
public void setProxyServer(String proxy, String proxyPort) {
    //设置代理服务器
    System.getProperties().put("proxySet", "true");
    System.getProperties().put("proxyHost", proxy);
}

```



```

System.getProperties().put("proxyPort", proxyPort);
}
//用于测试的主方法
public static void main(String argv[]) {
    Queue oInstance = new Queue();
    try {
        //增加下载列表（此处用户可以写入自己代码来增加下载列表）
        oInstance.addItem("http://download.microsoft.com/download/4/a/a/4aa524c6-239d-47ff-860b-5b397199cbf8/Windows-KB890830-V1.36.exe", "/KB890830-V1.36.exe");
        oInstance.addItem("http://download.microsoft.com/download/9/6/1/961e5c08-2fb3-4891-8d47-d7a4373d9032/Windows6.0-KB941568-x86.msu", "/Windows6.0-KB941568-x86.msu");
        //开始下载
        oInstance.downloadByList();
    }
    catch (Exception err) {System.out.println(err.getMessage());}
}
}

```



案例7: Java实现HTTP验证

案例运行效果与操作

通过验证访问资源以达到保护资源的目的是网站经常使用的手法。本案例通过Tomcat的访问认证,说明了Java支持HTTP验证。

在Tomcat主目录下,建立/test/wait目录,把Axis的index.html文件拷贝至此(任意HTML文件均可)。验证之前发送http://localhost:8080/test/wait的请求,会有如图5-29所示的返回界面。

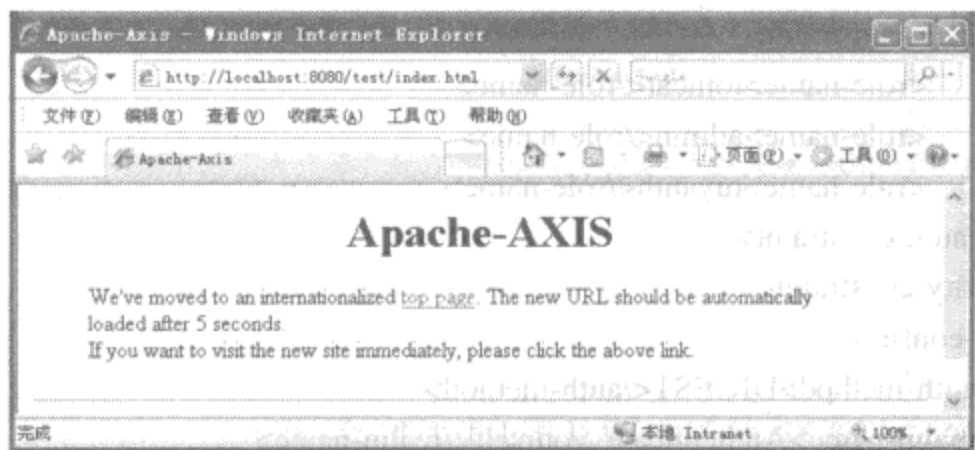


图5-29 返回界面

在DOS窗口运行命令java myAuth http://localhost:8080/test/wait,程序运行后,弹出如图5-30所示的窗口。输入正确的用户名和密码后,又可以访问如图5-29所示的内容了,不过显示方式有所不同,如图5-31所示。



图5-30 验证窗口

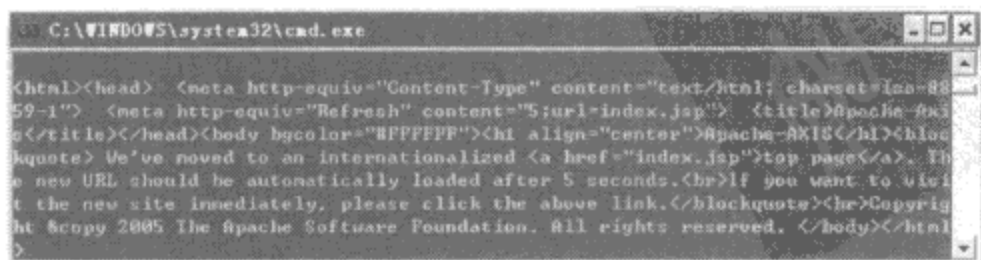


图5-31 访问内容

制作要点

1. HttpURLConnection的使用。
2. Authenticator类的用法。

通过Authenticator类为验证提供了本地支持，方法很简单，继承这个类并实现它的getPasswordAuthentication方法，这个方法取得用户名和密码并用它们生成一个PasswordAuthentication对象后返回。完成之后，使用Authenticator.setDefault方法注册Authenticator实例。现在，只要访问受保护的资源，就会调用getPasswordAuthentication。Authenticator类管理着所有低层的详细资料，它不受HTTP的限制，可以应用于所有网络连接。

步骤详解

1. tomcat-users.xml文件中增加用户及其角色的列表：

```
<user username="role1" password="tomcat" roles="role1"/>
<user username="admin" password="admin" roles="author"/>
<user username="myauth" password="admin" roles="reader"/>
```

2. 修改web.xml：

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>
            Restricted Area
        </web-resource-name>
        <url-pattern>/wait/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>tomcat</role-name>
        <role-name>admin</role-name>
        <role-name>myauth</role-name>
    </auth-constraint>
</security-constraint>
<login-config>
    <auth-method>DIGEST</auth-method>
    <realm-name>Authenticate yourself</realm-name>
</login-config>
```

其中，<security-constraint>元素限制对一个或者多个资源的访问，<login-config>元素用于指定验证方法。

3. 继承Authenticator类，实现getPasswordAuthentication方法：

```
public class AuthJava1 extends Authenticator {
    protected PasswordAuthentication getPasswordAuthentication() {}
}
```

4. 建立网络连接：

```
URLConnection connection = url.openConnection();
```


程序源代码与解释

```

//myAuth.java
import java.io.*;
import java.net.*;
public class myAuth {
    public static void main(String argv[]) throws Exception {
        Authenticator.setDefault(new AuthJava ());
        if (argv.length != 1) {
            System.err.println("Usage: java BasicAuth <site>");
            System.exit(1);
        }
        URL url = new URL("http://localhost:8080/test/wait");
        URLConnection connection = url.openConnection();
        BufferedReader in= new BufferedReader(new InputStreamReader(connection
.getInputStream()));
        String line;
        StringBuffer sb = new StringBuffer();
        while ((line = in.readLine()) != null) {
            sb.append(line);
        }
        in.close();
        System.out.println(sb.toString());
        System.exit(0);
    }
}

//AuthJava.java
import java.net.*;
import java.awt.*;
import javax.swing.*;
public class AuthJava1 extends Authenticator {
    protected PasswordAuthentication getPasswordAuthentication() {
        JTextField username = new JTextField();
        JPasswordField password = new JPasswordField();
        JPanel panel = new JPanel(new GridLayout(2,2));
        panel.add(new JLabel("User Name"));
        panel.add(username);
        panel.add(new JLabel("Password") );
        panel.add(password);
        int option= JOptionPane.showConfirmDialog(null, new Object[] {"Site:
"+getRequestingHost(),"Realm:
"+getRequestingPrompt(),panel}, "Enter Network Password", JOptionPane.OK_
CANCEL_OPTION, JOptionPane.PLAIN_MESSAGE);
        if ( option == JOptionPane.OK_OPTION ) {
            String user = username.getText();
            char pass[] = password.getText().toCharArray();
            return new PasswordAuthentication(user, pass);
        } else {
            return null;
        }
    }
}

```


本章小结

Java从诞生之日起就与网络密不可分，Java的特性是它能够最大限度地利用网络。Java在网络中的应用大致分为两种，即Web浏览和网络应用系统。Applet、Servlet和JSP三大技术应用在Web浏览中，进一步增强了Web服务的能力。而由于Java语言本身是与平台无关的，因此用Java语言开发的网络应用系统可以在各种平台上运行，大大增强了代码的重复使用效率，提高了系统开发效率，减少了程序员的重复劳动。而且，Java继承的网络功能有利于开发网络应用程序。本章通过几个不同难易程度的实用案例程序，讲述了在Java语言中网络应用的一些概念和方法。在实际应用中，由于Java在网络方面的应用非常广博，不可能通过几个案例覆盖，所以需要读者在实际应用中仔细体会。



第6章 Java与数据库

本章内容

- 案例1: Access数据库编程中查询结果的表格式输出
- 案例2: SQL Server数据库编程中查询结果的表格式输出
- 案例3: MySQL数据库编程中查询结果的表格式输出
- 案例4: Oracle OCI数据库编程
- 案例5: 网吧计费系统
- 本章小结



案例1: Access数据库编程中查询结果的表格式输出

案例运行效果与操作

本案例展示了一个简单的利用JDBC-ODBC桥访问Access数据库的例子，并将数据库查询结果用表格的形式进行输出。程序运行后，界面如图6-1所示。



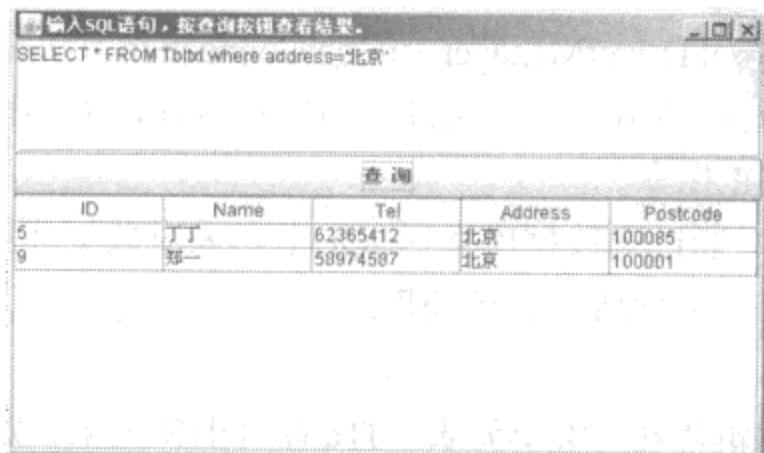
输入SQL语句，按查询按钮查看结果。

SELECT * FROM Tbltbl

ID	Name	Tel	Address	Postcode
1	张三	83854123	大连	210000
2	李四	26321456	沈阳	120000
3	王二	25412365	哈尔滨	110564
4	马五	54123654	长春	256487
5	丁丁	62365412	北京	100085
6	齐整	71236541	上海	324058
7	冯泉	88654325	济南	250011
8	齐鲁	42985647	济南	250101
9	郑一	58974587	北京	100001

图6-1 运行界面

在窗口上方的文本区中输入查询语句即可对数据库进行查询，如图6-2所示为查询通讯录中地址为“北京”的记录后的输出界面。



输入SQL语句，按查询按钮查看结果。

SELECT * FROM Tbltbl where address='北京'

ID	Name	Tel	Address	Postcode
5	丁丁	62365412	北京	100085
9	郑一	58974587	北京	100001

图6-2 查询输出界面

如果查询的结果集中没有记录，会弹出消息对话框进行提示，界面如图6-3所示。

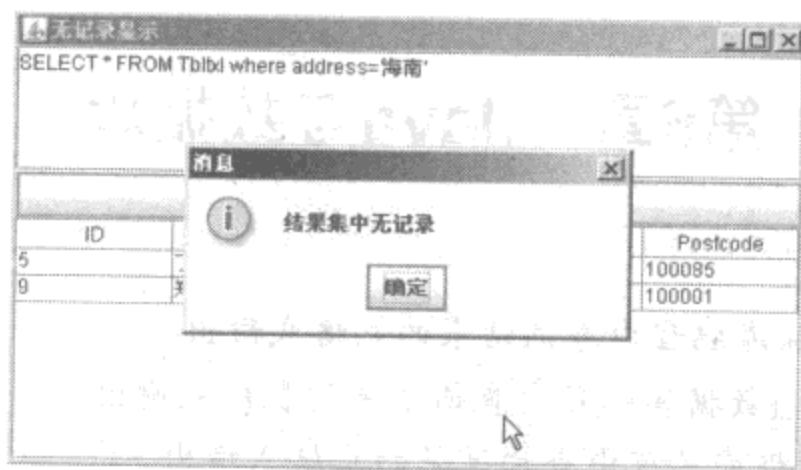


图6-3 无记录提示界面

制作要点

1. 基于JDBC-ODBC桥的数据库访问方法。

用Java连接数据库主要有两种方式，一是用JDBC-ODBC桥来连接，二是用相关厂商提供的相应驱动程序来连接。首先谈谈第一种连接。

JDBC-ODBC桥接器是用JdbcOdbc.Class和一个用于访问ODBC驱动程序的本地库实现的。对于Windows平台，该本地库是一个动态连接库DLL(JDBCODBC.DLL)。

JDBC在设计上与ODBC很接近。在内部，这个驱动程序把JDBC的方法映射到ODBC调用上，这样，JDBC就可以和任何可用的ODBC驱动程序进行交互了。这种桥接器的优点是，它使JDBC目前有能力访问几乎所有的数据库。通行方式如下：

应用程序→JDBC API→JDBC-ODBC→ODBC API→ODBC层→数据源

根据JDBC的任务，在Java应用程序中实现基于JDBC-ODBC桥的数据库访问方法可描述如下：

- (1) 设置ODBC数据源。
- (2) 代码设计。

按如下步骤设计Java应用程序代码：

1) 注册数据库驱动程序。

使用JDBC-ODBC桥驱动程序，必须先注册。通常采用如下加载数据库驱动程序类的方法进行JDBC-ODBC桥驱动程序的隐式注册：

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

2) 连接数据库。

数据库的连接由java.sql.DriverManager类的getConnection方法完成，该方法自动从数据库驱动程序注册表中搜索适合的驱动程序连接到指定数据库，并返回一个Connection对象。

3) 向数据库发送SQL指令。

数据库连接成功后，通过向数据库发送SQL指令来对数据库进行查询、更新等操作。

4) 对返回的SQL指令执行结果进行处理。

2. JTable类的使用。

JTable用来显示和编辑常规二维单元表。JTable有很多用来自定义其呈现和编辑的工具，同时提供了这些功能的默认设置，从而可以轻松地设置简单表。

JTable构造方法如下:

- (1) JTable(): 建立一个新的JTables, 并使用系统默认的Model。
- (2) JTable(int numRows,int numColumns): 建立一个具有numRows行、numColumns列的空表格, 使用的是DefaultTableModel。
- (3) JTable(Object[][] rowData,Object[][] columnNames): 建立一个显示二维数组数据的表格, 且可以显示列的名称。
- (4) JTable(TableModel dm): 建立一个JTable, 有默认的字段模式以及选择模式, 并设置数据模式。
- (5) JTable(TableModel dm,TableColumnModel cm): 建立一个JTable, 设置数据模式与字段模式, 并有默认的选择模式。
- (6) JTable(TableModel dm,TableColumnModel cm,ListSelectionModel sm): 建立一个JTable, 设置数据模式、字段模式与选择模式。
- (7) JTable(Vector rowData,Vector columnNames): 建立一个以Vector为输入来源的数据表格, 可显示行的名称。

表格由两部分组成, 分别是行标题 (Column Header) 与行对象 (Column Object), 利用JTable所提供的getTableHeader()方法取得行标题。本案例将JTable放在JScrollPane中, 这种做法可以将Column Header与Colmn Object完整地显示出来, 因为JScrollPane会自动取得Column Header。

```
table = new JTable( rows, columnHeads );
JScrollPane scroller = new JScrollPane( table );
```

步骤详解

1. 准备工作: 创建数据库和表, 创建数据源等。

(1) 新建Access数据库, 名称设置为“DBtxl”, 注意数据库的保存位置, 如果程序中不使用绝对位置的话, 则把新建数据库保存在程序运行的相应目录中。

- (2) 在数据库DBtxl中创建Tbltxl表, 并按照表6-1向表中添加字段。

表6-1 添加字段的名称及相应的设置选项

字段名称	数据类型	是否主键
ID	数字	是
Name	文本	否
Tel	文本	否
Address	文本	否
Postcode	文本	否

- (3) 在表Tbltxl中输入若干记录。

2. 创建数据源。

(1) 在“创建新数据源”对话框中选择驱动程序“Driver do Microsoft Acess (*.mdb)”新建数据源, 如图6-4所示。

(2) 单击“完成”按钮，会弹出“ODBC Microsoft Access安装”对话框，如图6-5所示，输入数据源名称（accessstxl）和说明文字（Access通讯录）。



图6-4 “创建新数据源”对话框

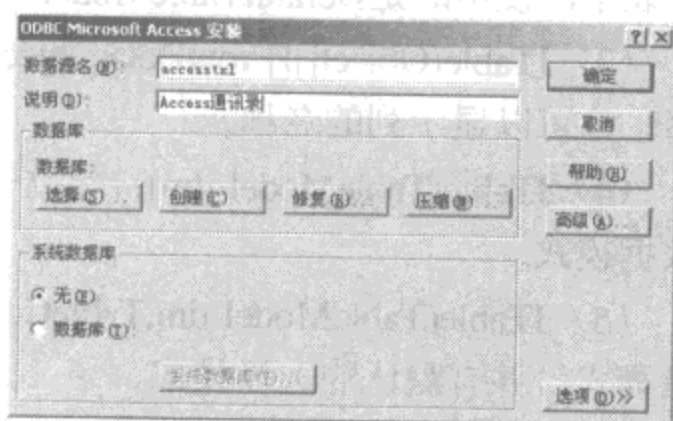


图6-5 “ODBC Microsoft Access安装”对话框

(3) 单击“选择”按钮，选择要连接的数据库，如图6-6所示。

(4) 选择刚才创建的数据库DBtxl.mdb，单击“确定”按钮，此时“ODBC数据源管理器”对话框界面如图6-7所示，表明accessstxl数据源创建成功。

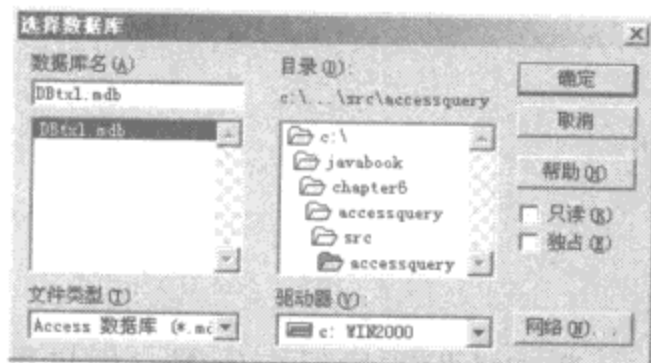


图6-6 “选择数据库”对话框

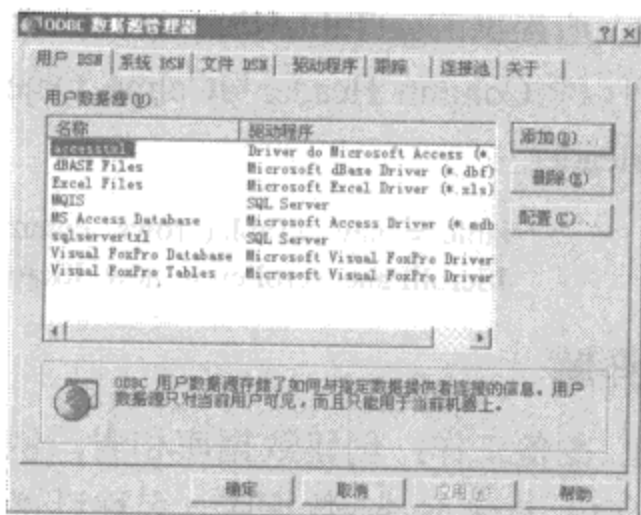


图6-7 创建数据源后的“ODBC数据源管理器”对话框

3. 加载数据库驱动程序，连接数据库：

```
String url = "jdbc:odbc:accessstxl"
Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
connection = DriverManager.getConnection( url, username, password );
```

4. 数据库连接成功后，设计界面布局并取得表格：

```
JPanel topPanel = new JPanel();
topPanel.setLayout( new BorderLayout() );
topPanel.add( new JScrollPane( inputQuery), BorderLayout.CENTER );
//将“提交查询”按钮布置到“SOUTH”
topPanel.add( submitQuery, BorderLayout.SOUTH );
table = new JTable();
Container c = getContentPane();
c.setLayout( new BorderLayout() );
//将“topPanel”编辑框布置到“NORTH”
c.add( topPanel, BorderLayout.NORTH );
//将“table”编辑框布置到“CENTER”
```



```
c.add( table, BorderLayout.CENTER );
getTable();
setSize( 500, 300 );
```

5. 执行SQL语句:

```
String query = inputQuery.getText();
statement = connection.createStatement();
resultSet = statement.executeQuery( query );
```

6. 将查询结果生成表格并显示出来。

定义两个Vector数组, 分别存放表格行、表格列内容:

```
Vector columnHeads = new Vector();
Vector rows = new Vector();
```

获取记录:

```
ResultSetMetaData rsmd = rs.getMetaData();
for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
    columnHeads.addElement( rsmd.getColumnName( i ) );
do {
    rows.addElement( getNextRow( rs, rsmd ) );
} while ( rs.next() );
```

建立一个显示二维数组数据的表格, 且可以显示列的名称, 显示查询结果

```
table = new JTable( rows, columnHeads );
JScrollPane scroller = new JScrollPane( table );
Container c = getContentPane();
c.remove(1);
c.add( scroller, BorderLayout.CENTER );
```

7. 不要忘记断开数据库的连接:

```
connection.close();
```

程序源代码与解释

```
/** AccessQuery.java*/
public class AccessQuery extends JFrame {
    //数据库变量定义
    private Connection connection;
    private Statement statement;
    private ResultSet resultSet;
    private ResultSetMetaData rsMetaData;
    //GUI变量定义
    private JTable table;
    private JTextArea inputQuery;
    private JButton submitQuery;
    public AccessQuery() {
        //Form的标题
        super( "输入SQL语句, 按查询按钮查看结果。" );
        //URL中指定ODBC中设置的DSN名称
        String url = "jdbc:odbc:accesstxl";
        String username = "";
```



```

String password = "";
//加载驱动程序以连接数据库
try {
    Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
    connection = DriverManager.getConnection( url, username, password );
}
//捕获加载驱动程序异常
catch ( ClassNotFoundException cnfex ) {
    System.err.println( "装载JDBC/ODBC驱动程序失败。" );
    cnfex.printStackTrace();
    System.exit( 1 ); //terminate program
}
//捕获连接数据库异常
catch ( SQLException sqlx ) {
    System.err.println( "无法连接数据库" );
    sqlx.printStackTrace();
    System.exit( 1 ); //终止程序
}
//如果数据库连接成功, 则建立GUI
//SQL语句
String test="SELECT * FROM Tbltxl";
inputQuery = new JTextArea( test, 4, 30 );
submitQuery = new JButton( "查 询" );
//Button事件
submitQuery.addActionListener(
    new ActionListener() {
        public void actionPerformed((ActionEvent e) )
        {
            getTable();
        }
    }
);
//设计界面布局, 参见步骤详解4
//显示Form
this.setVisible(true);
}
private void getTable() {
    try {
        //执行SQL语句, 参见步骤详解5
        //在表格中显示查询结果
        displayResultSet( resultSet );
    }
    catch ( SQLException sqlx ) {
        sqlx.printStackTrace();
    }
}
private void displayResultSet( ResultSet rs ) throws SQLException {
    //定位到达第一条记录
    boolean moreRecords = rs.next();
    //如果没有记录, 则提示一条消息
    if ( ! moreRecords ) {
        JOptionPane.showMessageDialog( this,
            "结果集中无记录" );
    }
}

```



```

        setTitle( "无记录显示" );
        return;
    }
    Vector columnHeads = new Vector();
    Vector rows = new Vector();
    try {
        //获取字段的名称
        ResultSetMetaData rsmd = rs.getMetaData();
        for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
            columnHeads.addElement( rsmd.getColumnName( i ) );
        //获取记录集
        do {
            rows.addElement( getNextRow( rs, rsmd ) );
        } while ( rs.next() );
        //在表格中显示查询结果, 参见步骤详解6
        //刷新Table
        c.validate();
    }
    catch ( SQLException sqlex ) {
        sqlex.printStackTrace();
    }
}

private Vector getNextRow( ResultSet rs, ResultSetMetaData rsmd ) throws SQLException {
    Vector currentRow = new Vector();
    for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
        currentRow.addElement( rs.getString( i ) );
    //返回一条记录
    return currentRow;
}

public void shutDown() {
    try {
        //断开数据库连接
        connection.close();
    }
    catch ( SQLException sqlex ) {
        System.err.println( "Unable to disconnect" );
        sqlex.printStackTrace();
    }
}

public static void main( String args[] ) {
    final AccessQuery app = new AccessQuery();
    app.addWindowListener(
        new WindowAdapter() {
            public void windowClosing( WindowEvent e ) {
                app.shutDown();
                System.exit( 0 );
            }
        }
    );
}
}

```




案例2: SQL Server数据库编程中 查询结果的表格式输出

案例运行效果与操作

本案例展示了一个简单的利用JDBC-ODBC桥访问SQL Server数据库的例子，并将数据库查询结果用表格的形式进行输出。程序运行后，界面如图6-8所示。本案例与案例1非常相似，最大的不同是数据源的不同。读者稍微改动一下源码即可完成数据库的移植，当然前提是已经完成数据的迁移。

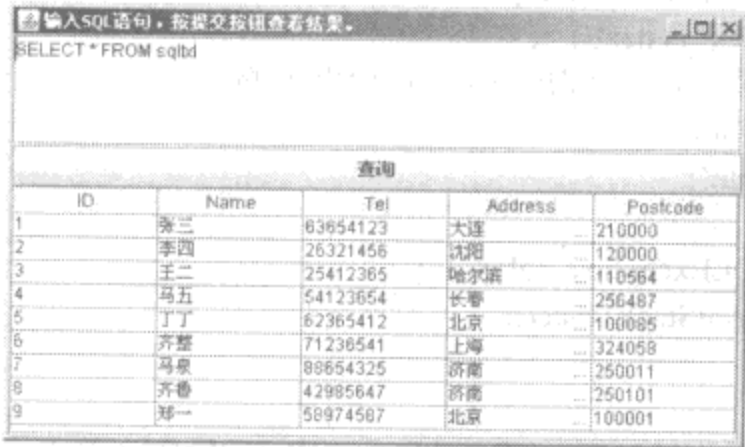


图6-8 运行界面

此时在窗口上方的文本区中输入查询语句即可对数据库进行查询，如图6-9所示为查询通讯录中地址为“济南”的记录后的输出界面。

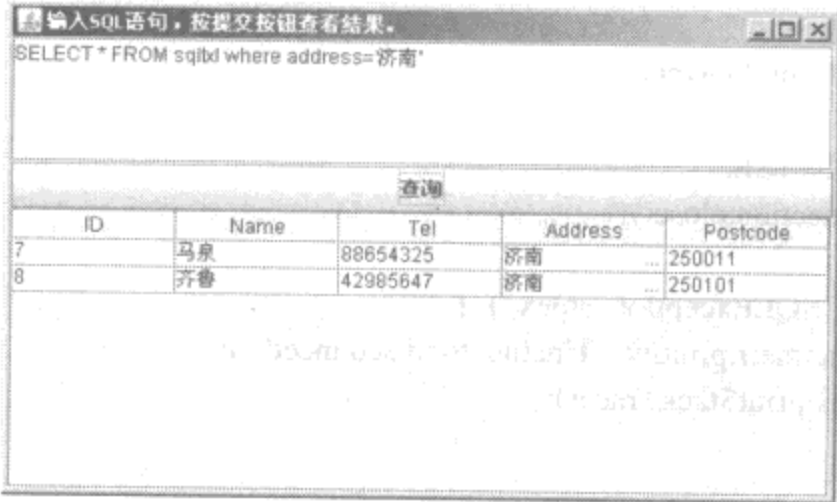


图6-9 查询输出界面

如果查询的结果集中没有记录，会弹出消息对话框进行提示，界面如图6-10所示。

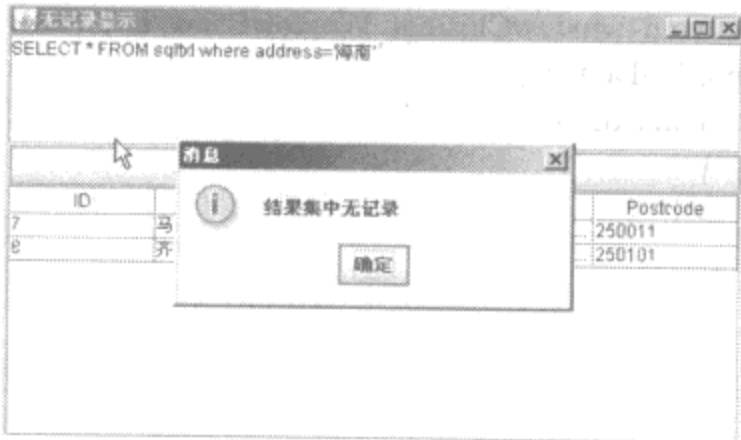


图6-10 无记录提示界面

制作要点

1. 基于JDBC-ODBC桥的数据库访问方法。
2. JTable类的使用。

步骤详解

1. 准备工作：创建数据库和表，创建数据源等。

- (1) 打开“企业管理器”控制台。新建名为“txl”的数据库，保存到默认目录中。
- (2) 在数据库txl中创建sqltxl表，并按照表6-2向表中添加字段。

表6-2 添加字段的名称及相应的设置选项

字段名称	数据类型	是否主键	长度	允许空
ID	bigint	是	8	否
Name	char	否	10	是
Tel	char	否	10	是
Address	char	否	40	是
Postcode	char	否	6	是

- (3) 在sqltxl表中输入若干记录。

2. 创建数据源。

- (1) 在“ODBC数据源管理器”中创建新数据源，选择数据库的ODBC驱动程序SQL Server，本案例数据源名称为sqlservertxl，连接的服务器实例名称为myhomepc2，步骤如同案例1。

- (2) SQL Server要求验证登录，需要选择如何进行身份验证，如图6-11所示。

- (3) 继续单击“下一步”按钮，界面如图6-12所示，在该界面中选中“更改默认的数据库为：”复选框，将数据库设定为刚才建立的txl。

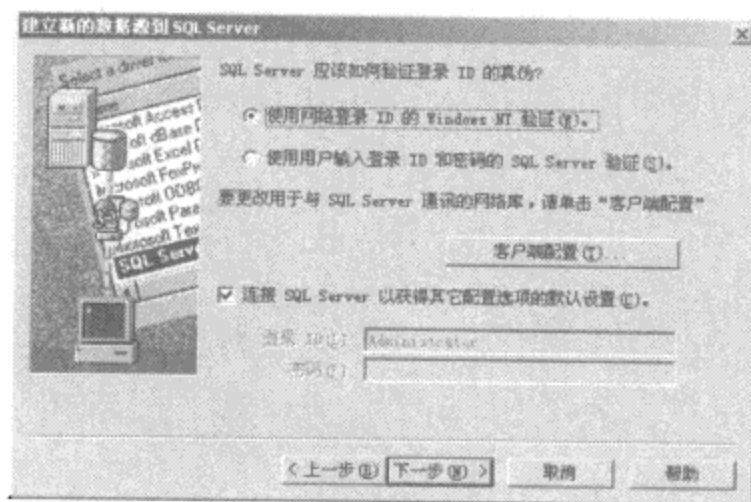


图6-11 “建立新的数据源到 SQL Server”对话框

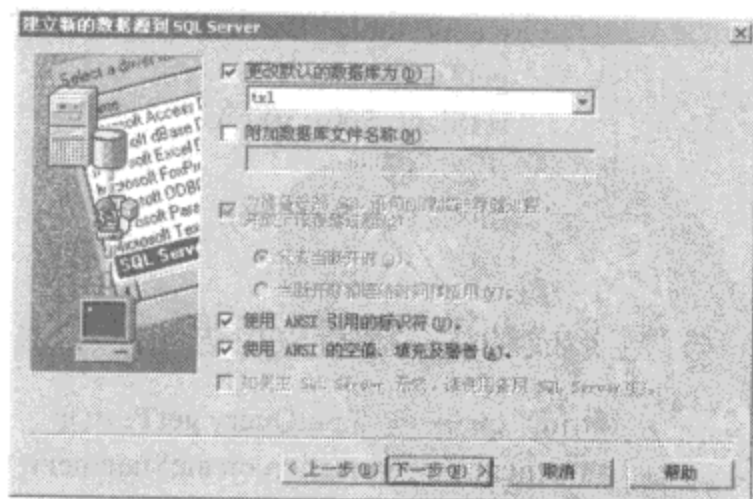


图6-12 “建立新的数据源到 SQL Server”对话框

- (4) 完成建立后的“ODBC Microsoft SQL Server安装”对话框如图6-13所示。

- (5) 可以单击“测试数据源”按钮选择要连接的数据库，如果测试成功则可以继续，否则检查数据库服务器或者网络状态。成功建立数据源后，在ODBC数据源管理器中会增加

一条用户数据源的记录。

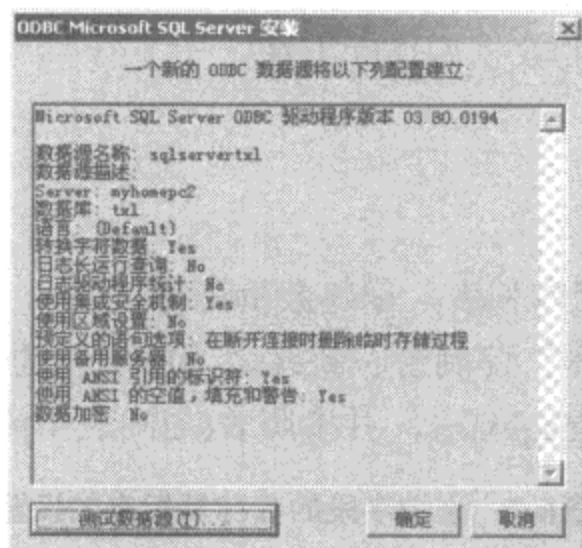


图6-13 “ODBC Microsoft SQL Server安装”对话框

3. 准备工作完成后，加载数据库驱动，连接数据库：

```
String url = "jdbc:odbc:sqlservrtxl";
Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
connection = DriverManager.getConnection(url, username, password );
```

4. 设计界面：

```
JPanel topPanel = new JPanel();
topPanel.setLayout( new BorderLayout() );
//将“输入查询”编辑框布置到“CENTER”
topPanel.add( new JScrollPane( inputQuery), BorderLayout.CENTER );
//将“提交查询”按钮布置到“SOUTH”
topPanel.add( submitQuery, BorderLayout.SOUTH );
table = new JTable();
Container c = getContentPane();
c.setLayout( new BorderLayout() );
//将“topPanel”编辑框布置到“NORTH”
c.add( topPanel, BorderLayout.NORTH );
//将“table”编辑框布置到“CENTER”
c.add( table, BorderLayout.CENTER );
getTable();
setSize( 500, 300 );
//显示Form
this.setVisible(true);
}
```

5. 连接成功后，执行SQL语句：

```
String query = inputQuery.getText();
statement = connection.createStatement();
resultSet = statement.executeQuery( query );
```

6. 显示查询结果：

```
Vector columnHeads = new Vector();
Vector rows = new Vector();
try {
    //获取字段的名称
```



```

ResultSetMetaData rsmd = rs.getMetaData();
for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
    columnHeads.addElement( rsmd.getColumnName( i ) );
//获取记录集
do {
    rows.addElement( getNextRow( rs, rsmd ) );
} while ( rs.next() );
//在表格中显示查询结果
table = new JTable( rows, columnHeads );
JScrollPane scroller = new JScrollPane( table );
Container c = getContentPane();
c.remove(1);
c.add( scroller, BorderLayout.CENTER );
//刷新Table
c.validate();

```

7. 断开连接:

```
connection.close();
```

程序源代码与解释

```

/* * SQLServerQuery.java */
public class SQLServerQuery extends JFrame {
    //数据库变量定义
    private Connection connection;
    private Statement statement;
    private ResultSet resultSet;
    private ResultSetMetaData rsMetaData;
    //GUI变量定义
    private JTable table;
    private JTextArea inputQuery;
    private JButton submitQuery;
    public SQLServerQuery() {
        //Form的标题
        super( "输入SQL语句, 按提交按钮查看结果。" );
        //url中指定ODBC中设置的DSN名称
        String url = "jdbc:odbc:sqlservertxl";
        String username = "";
        String password = "";
        //加载驱动程序以连接数据库
        try {
            Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" ); //加载驱动
            connection = DriverManager.getConnection(url, username, password );
        }
        //捕获加载驱动程序异常
        catch ( ClassNotFoundException cnfex ) {
            System.err.println(
                "装载JDBC/ODBC驱动程序失败。" );
            cnfex.printStackTrace();
            System.exit( 1 ); //terminate program
        }
        //捕获连接数据库异常
        catch ( SQLException sqlex ) {

```



```

        System.err.println( "无法连接数据库" );
        sqllex.printStackTrace();
        System.exit( 1 ); //terminate program
    }
    //如果数据库连接成功，则建立GUI
    //SQL语句
    String test="SELECT * FROM sqltxl";
    inputQuery = new JTextArea( test, 4, 30 );
    submitQuery = new JButton( "查询" );
    //Button事件
    submitQuery.addActionListener(
        new ActionListener() {
            public void actionPerformed((ActionEvent e) )
            {
                getTable();
            }
        }
    );
    //界面设计，参见步骤详解4
}

private void getTable() {
    try {
        //执行SQL语句
        String query = inputQuery.getText();
        statement = connection.createStatement(); //创建Statement对象
        resultSet = statement.executeQuery( query ); //执行查询
        //在表格中显示查询结果
        displayResultSet( resultSet );
    }
    catch ( SQLException sqllex ) {
        sqllex.printStackTrace();
    }
}

private void displayResultSet( ResultSet rs ) throws SQLException {
    //定位到达第一条记录
    boolean moreRecords = rs.next();
    //如果没有记录，则提示一条消息
    if ( ! moreRecords ) {
        JOptionPane.showMessageDialog( this,
            "结果集中无记录" );
        setTitle( "无记录显示" );
        return;
    }
    //参见步骤详解6
}
catch ( SQLException sqllex ) {
    sqllex.printStackTrace();
}
}

private Vector getNextRow( ResultSet rs, ResultSetMetaData rsmd ) throws SQLException {
    Vector currentRow = new Vector();
    for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
        currentRow.addElement( rs.getString( i ) );
}

```



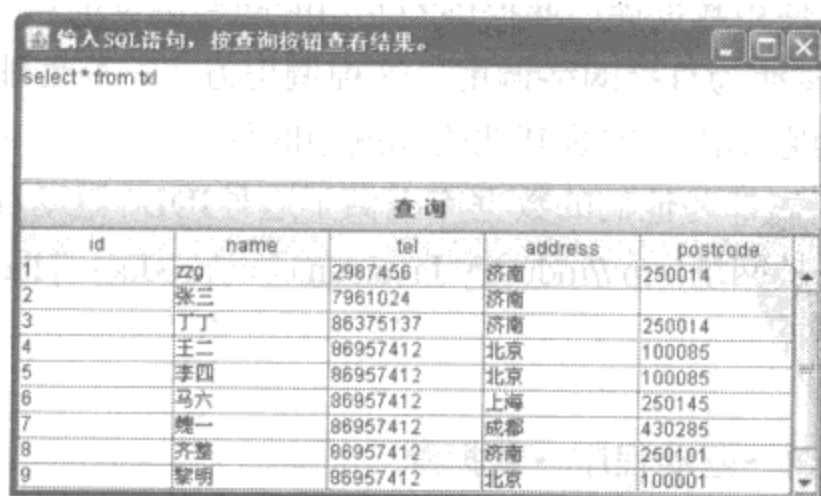
```
//返回一条记录
return currentRow;
}
public void shutDown() {
    try {
        //断开数据库连接
        connection.close();
    }
    catch ( SQLException sqllex ) {
        System.err.println( "Unable to disconnect" );
        sqllex.printStackTrace();
    }
}
public static void main( String args[] ) {
    final SQLServerQuery app = new SQLServerQuery();
    app.addWindowListener(
        new WindowAdapter() {
            public void windowClosing( WindowEvent e ) {
                app.shutDown();
                System.exit( 0 );
            }
        }
    );
}
```



案例3: MySQL数据库编程中查询结果的表格式输出

案例运行效果与操作

本案例展示了一个简单的由厂商提供的相应驱动程序来连接MySQL数据库的例子，并将数据库查询结果用表格的形式进行输出。程序运行后，界面如图6-14所示。



The screenshot shows a window titled "输入SQL语句，按查询按钮查看结果。". Inside, there is a text area with the query "select * from bd". Below the text area is a table with the following data:

查询				
id	name	tel	address	postcode
1	zzg	2987456	济南	250014
2	张三	7961024	济南	
3	丁丁	86375137	济南	250014
4	王二	86957412	北京	100085
5	李四	86957412	北京	100085
6	马六	86957412	上海	250145
7	魏一	86957412	成都	430285
8	齐整	86957412	济南	250101
9	黎明	86957412	北京	100001

图6-14 运行界面

此时在窗口上方的文本区中输入查询语句即可对数据库进行查询，如图6-15所示为查询通讯录中地址为“北京”的记录后的输出界面。

如果查询的结果集中没有记录，会弹出消息对话框进行提示，界面如图6-16所示。

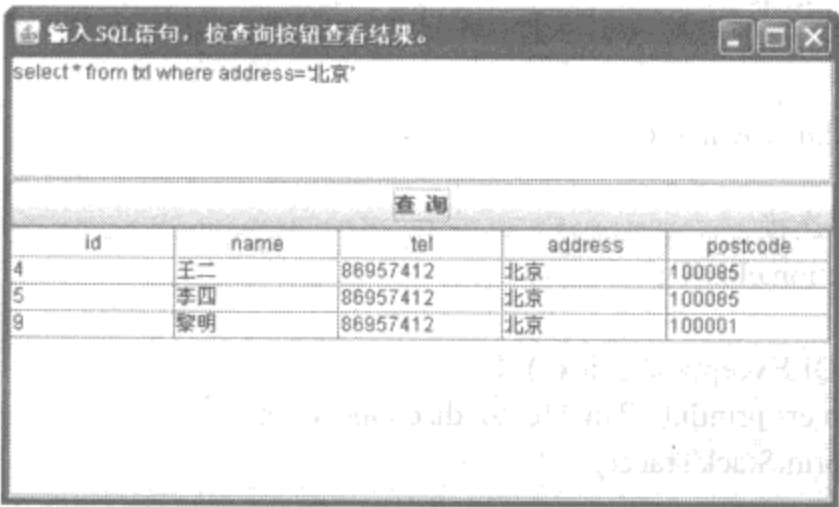


图6-15 查询输出界面

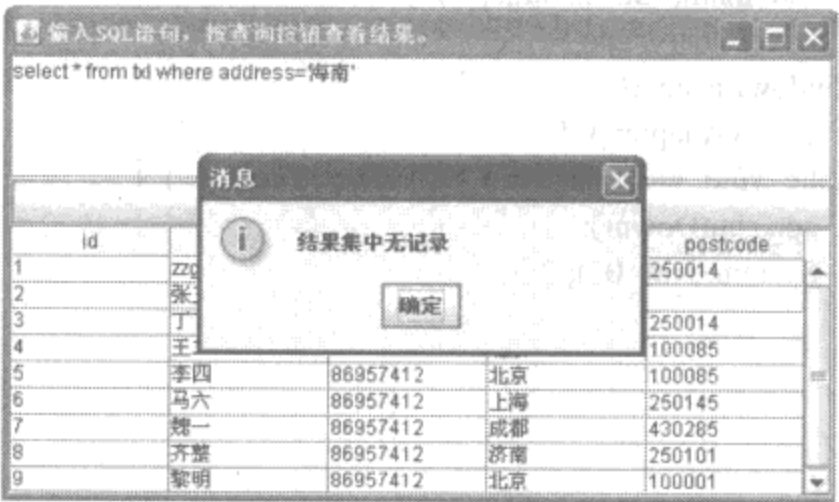


图6-16 无记录提示界面

制作要点

1. 基于JDBC驱动程序的数据库访问方法。
- 用Java连接数据库主要有两种方式，一是用JDBC-ODBC桥来连接，二是用相关厂商提供的相应JDBC驱动程序来连接。前面两个案例用的是第一种连接方法，现在来探讨一下第二种连接方法。

这种实现方法是直接使用数据库厂商提供的专用的网络协议创建的驱动程序，通过它可以直接将JDBC API调用转换为直接网络调用。这种调用方式一般性能比较好，而且也是实际中最简单的方法。因为它不需要安装其他的库或中间件。

几乎所有的数据库厂商都为他们的数据库提供了这种JDBC驱动程序，也可以从第三方厂商获得这些驱动程序。从网址<http://industry.java.sun.com/products/jdbc/drivers/>可以看到所有可用的驱动程序的清单。

其运行方式如下：



应用程序→JDBC API→驱动程序→数据源

在Java应用程序中实现这种数据库访问方法也非常简单，只要在Java应用程序中导入驱动程序的相关JAR文件即可。
2. JTable类的使用。

步骤详解

1. 准备工作：创建数据库和表等。安装MySQL 5.0和MySQL GUI Tools 5.0，前者是MySQL

数据库服务器的主程序，后者则是其图形界面管理工具，包括MySQL Administrator和MySQL QueryBrowser等。然后按照如下步骤完成数据库和表的创建。

(1) 启动服务。双击安装的MySQL Administrator目录下的MySQLSystemTrayMonitor.exe，这时会在任务栏右侧出现一个图标。用右键单击该图标，出现如图6-17所示的菜单，选择“Start Instance”，运行MySQL。MySQL成功启动后，任务栏右侧图标变成.

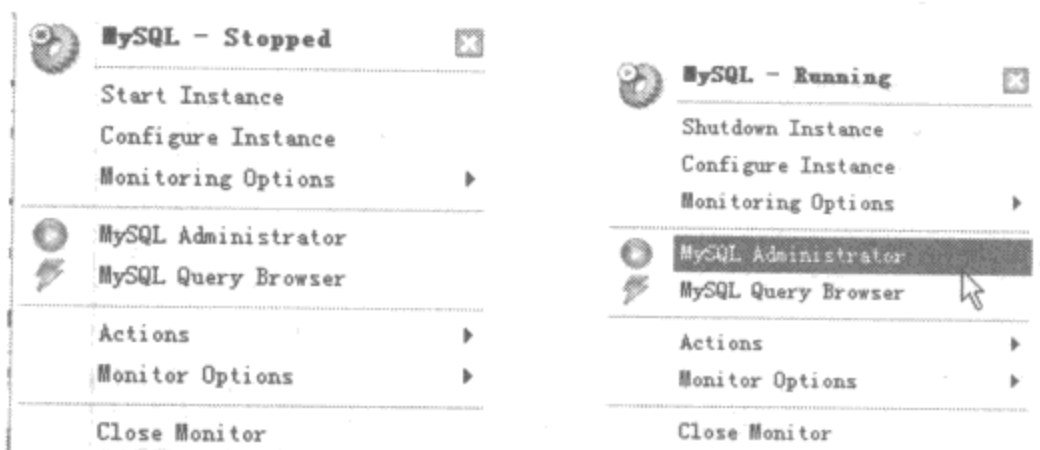


图6-17 MySQLSystemTrayMonitor右键菜单（左为MySQL启动前，右为启动后）

(2) 创建数据库和表。MySQL初始有三个数据库：information_schema、mysql和test库。其中前两者含有各种配置信息，而test库初始是空的，本案里就以test库作为测试数据库。右击右下角“MySQL System Tray Monitor”图标，选择“MySQL Query Browser”进入查询浏览器登录界面，如图6-18所示。此处将数据库名称设置为“localhost”，用户名为“root”，密码为空，Default_schema（默认数据库）为test。单击“OK”按钮进入查询浏览器界面。

(3) 在test数据库中创建tbl表。在查询浏览器界面中用右键单击test数据库，出现如图6-19所示的菜单，选择“Create New Table”，创建一个新表。

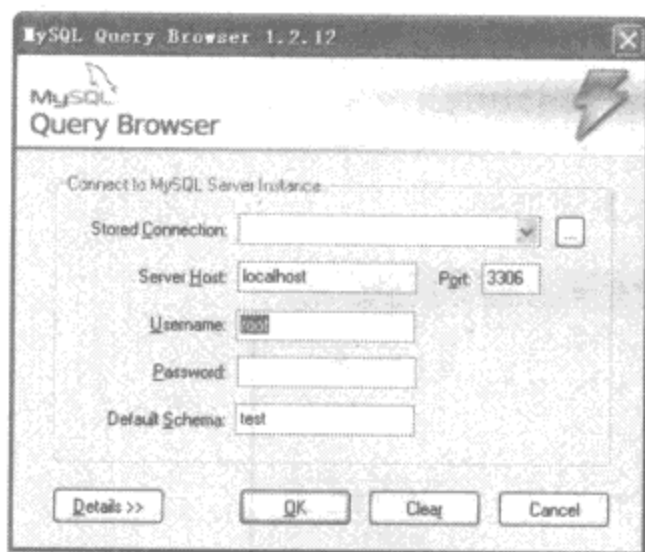


图6-18 查询浏览器登录界面

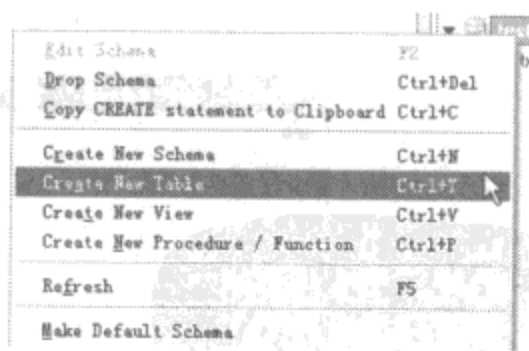


图6-19 创建一个新表

(4) 进行表设计。在弹出的“表设计器”窗口进行表格设计，如图6-20所示。

(5) 在tbl表中输入若干记录。

2. 准备工作完成后，安装、配置数据库厂商提供的JDBC驱动程序。从<http://dev.mysql.com/downloads/>网址下载mysql-connector-java-5.0.7，将下载得到的Zip文件解压到硬盘某个位置。添加JAR/文件夹mysql-connector-java-5.0.7-bin.jar到开发环境，如图6-21和图6-22所示，将MySQL的JDBC驱动程序导入，完成JDBC驱动程序的安装和配置。

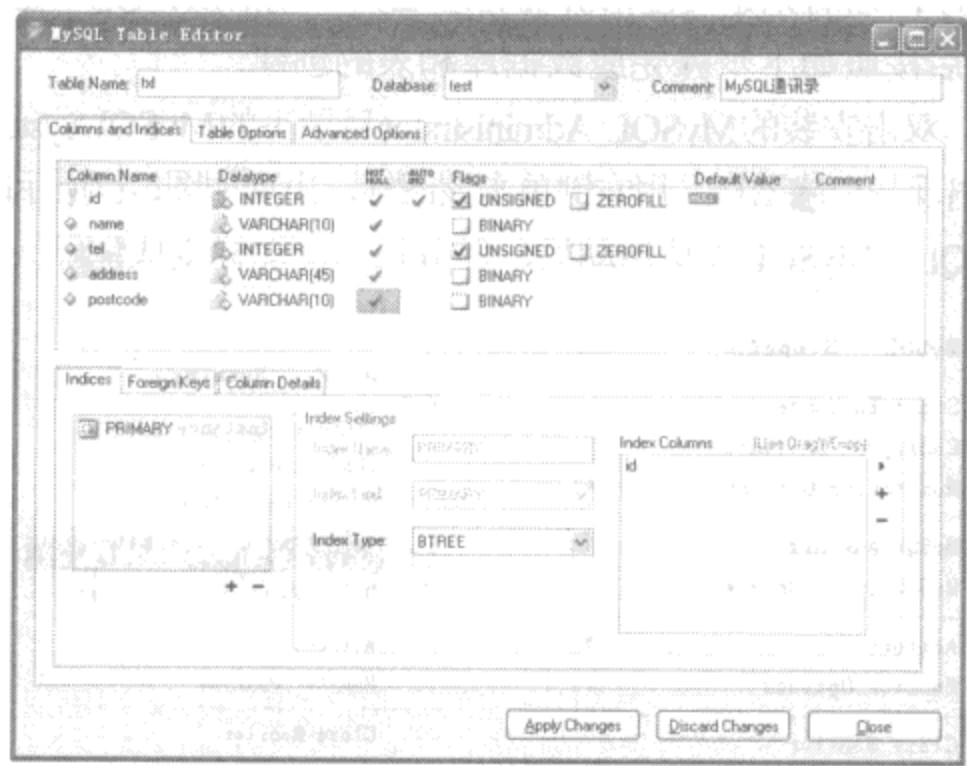


图6-20 “表设计器”窗口

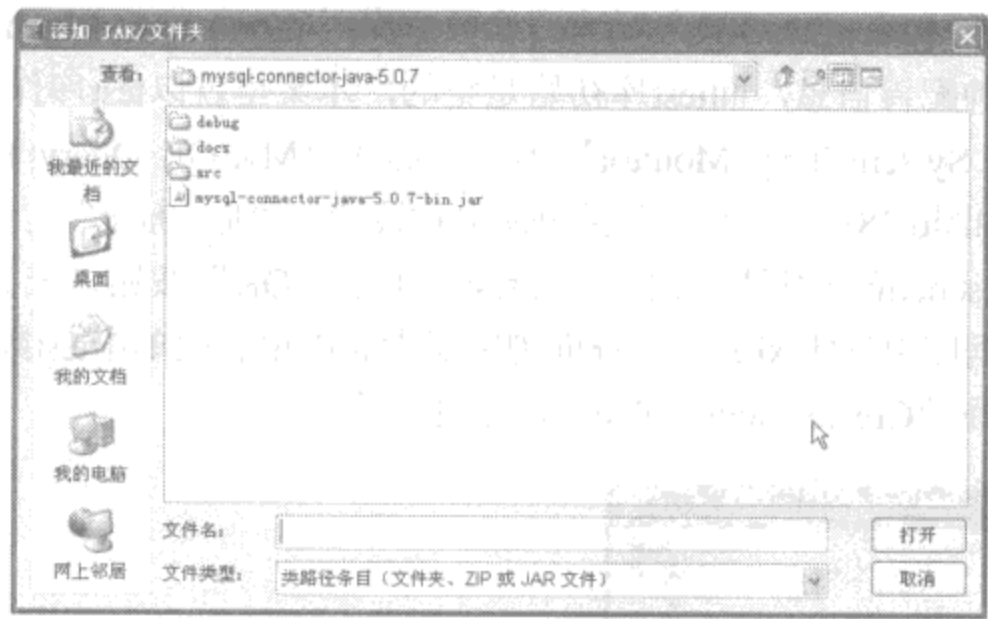


图6-21 【添加JAR/文件夹】对话框

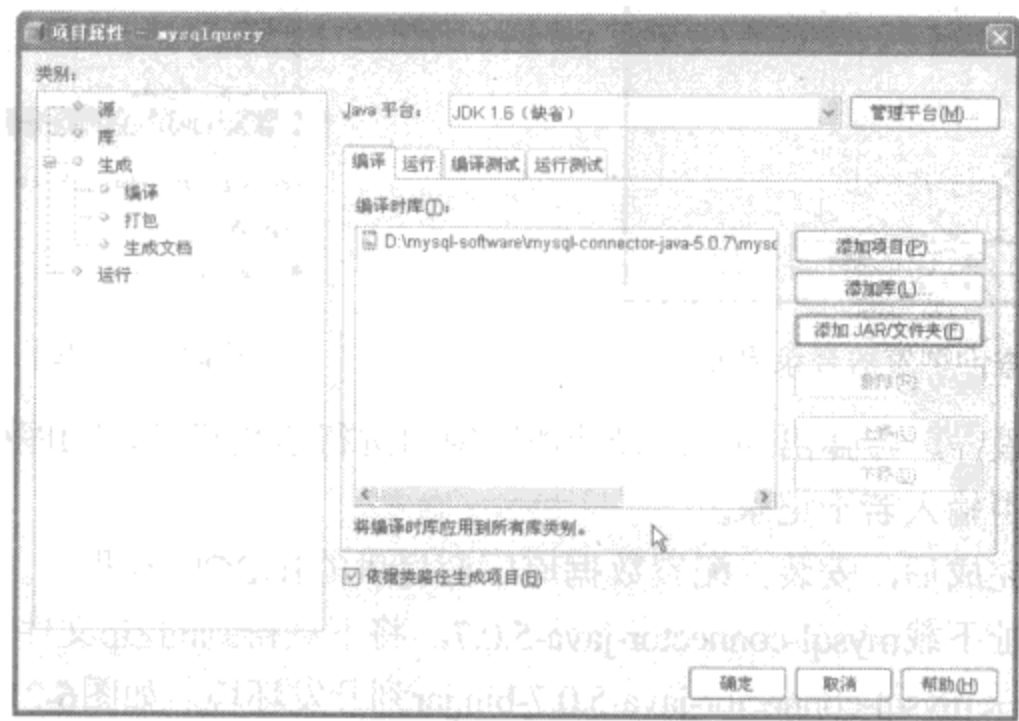


图6-22 【项目属性】对话框

3. 界面设计:

```

JPanel topPanel = new JPanel();
topPanel.setLayout( new BorderLayout() );
//将“输入查询”编辑框布置到“CENTER”
topPanel.add( new JScrollPane( inputQuery), BorderLayout.CENTER );
//将“提交查询”按钮布置到“SOUTH”
topPanel.add( submitQuery, BorderLayout.SOUTH );
table = new JTable();
Container c = getContentPane();
c.setLayout( new BorderLayout() );
//将“topPanel”编辑框布置到“NORTH”
c.add( topPanel, BorderLayout.NORTH );
//将“table”编辑框布置到“CENTER”
c.add( table, BorderLayout.CENTER );
getTable();

```

4. 加载数据库驱动, 连接数据库, 执行SQL语句:

```

Class.forName("org.gjt.mm.mysql.Driver").newInstance();    //加载驱动
//URL中指定JDBC中设置的数据库名称
String url = "jdbc:mysql://localhost:3306/test";
String username = "root";
String password = "";
query = inputQuery.getText();
//执行SQL语句
Connection connection = DriverManager.getConnection(url, username, password);
Statement statement = connection.createStatement();
resultSet = statement.executeQuery( query );

```

需要增加一个关于驱动程序加载的异常捕获:

```

catch (IllegalAccessException ex1) { }
catch (InstantiationException ex1) { }
catch ( ClassNotFoundException cnfex ) {
    system.err.println( "装载JDBC/ODBC驱动程序失败。" );
    cnfex.printStackTrace();
    System.exit( 1 ); //终止程序
}

```

5. 将查询结果生成表格并显示出来:

```

private void displayResultSet( ResultSet rs ) throws SQLException {
    Vector columnHeads = new Vector();
    Vector rows = new Vector();
    try {
        //获取字段的名称
        ResultSetMetaData rsmd = rs.getMetaData();
        for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
            columnHeads.addElement( rsmd.getColumnName( i ) );
        //获取记录集
        do {
            rows.addElement( getNextRow( rs, rsmd ) );
        } while ( rs.next() );
        //在表格中显示查询结果
        table = new JTable( rows, columnHeads );
        JScrollPane scroller = new JScrollPane( table );
    }
}

```



```

        Container c = getContentPane();
        c.remove(1);
        c.add( scroller, BorderLayout.CENTER );
    }
}

```

程序源代码与解释

```

/* * MySQLQuery.java */
public class MySQLQuery extends JFrame {
    //数据库变量定义
    private Connection connection;
    private Statement statement;
    private ResultSet resultSet;
    private ResultSetMetaData rsMetaData;
    //GUI变量定义
    private JTable table;
    private JTextArea inputQuery;
    private JButton submitQuery;
    public MySQLQuery() {
        //Form的标题
        super( "输入SQL语句，按查询按钮查看结果。" );
        //如果数据库连接成功，则建立GUI
        String test = "select * from txl";
        inputQuery = new JTextArea( test, 4, 30 );
        submitQuery = new JButton( "查  询" );
        //Button事件
        submitQuery.addActionListener( new ActionListener() {
            public void actionPerformed((ActionEvent e) {
                getTable();
            }
        });
        //...
        this.setVisible(true);
    }
    private void getTable() {
        String query;
        try {
            //加载驱动程序以连接数据库
            try {
                //参见步骤详解4
                //捕获连接数据库异常
                catch (SQLException sqlex) {
                    System.err.println( "无法连接数据库" );
                    sqlex.printStackTrace();
                    System.exit( 1 ); //终止程序
                }
                //在表格中显示查询结果
                displayResultSet( resultSet );
                resultSet.close();
            }
            catch (SQLException sqlex) {
                sqlex.printStackTrace();
            }
        }
    }
}

```



```

private Vector getNextRow( ResultSet rs, ResultSetMetaData rsmd ) throws SQLException {
    Vector currentRow = new Vector();
    for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
        currentRow.addElement( rs.getString( i ) );
    //返回一条记录
    return currentRow;
}

public static void main( String args[] ) {
    final MySQLQuery app = new MySQLQuery();
    app.addWindowListener( new WindowAdapter() {
        public void windowClosing( WindowEvent e ) {
            System.exit( 0 );
        }
    });
}
}

```

案例4: Oracle OCI数据库编程

案例运行效果与操作

本案例展示了一个简单的使用Oracle OCI JDBC驱动程序来连接Oracle数据库的例子，使用SCOTT模式中的dept数据库提供数据。运行程序后，在开发环境（本例为NetBeans）的输出窗口中出现如图6-23所示的界面。

而使用Oracle的SQL*Plus工具查询获得的输出界面如图6-24所示，与图6-23所示结果完全相同，表明连接Oracle数据库成功。

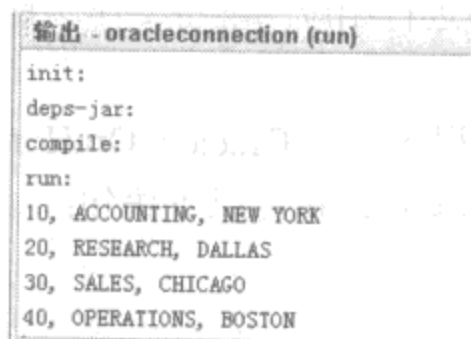


图6-23 运行界面

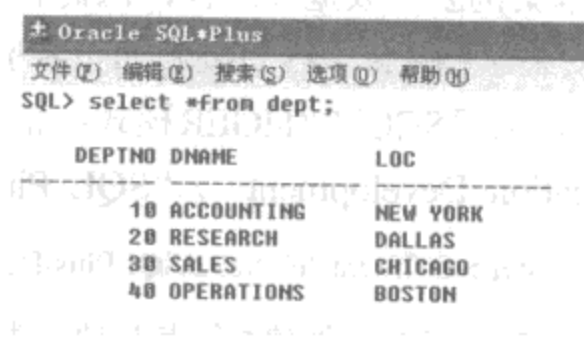


图6-24 SQL*Plus查询界面

制作要点

1. 基于JDBC驱动程序的数据库访问方法。

在Oracle中有三种类型的JDBC驱动，它们都使用相同的语法、API以及Oracle extensions，以使Java代码在客户端程序、基于Web的Java Applet小程序以及Java存储过程之间保持轻便灵活。三种类型如下。

(1) JDBC OCI: 此驱动类似于传统的ODBC驱动。因为需要OCI (Oracle Call Interface) 和Net8，所以要使用此驱动的Java程序的机器上安装客户端软件。

(2) JDBC Thin: 这种驱动一般用在运行于Web浏览器中的Java程序。它不是通过OCI或Net8，而是通过Java socket进行通信，因此不需要在使用JDBC Thin的客户端机器上安装客户端软件。

(3) JDBC KPRB: 这种驱动由直接存储在数据库中的Java程序使用, 如Java存储过程、触发器、数据库JSP等。它使用默认的或者当前的数据库会话, 因此不需要另外的数据库用户名、口令或URL。

Java程序连接Oracle数据库时, 用OCI驱动要比用Thin驱动性能好些。

Thin驱动连接字符串:

```
Connection conn= DriverManager.getConnection
                ("jdbc:oracle:thin:@dlsun511:1521:ora1","scott","tiger");
                machine(ip@) : port# : sid
```

OCI驱动连接字符串:

```
Connection conn= DriverManager.getConnection
                ("jdbc:oracle:oci8[9]:@RAC","scott","tiger");
                Net Service
```

本案例使用OCI驱动。

2. JTable类的使用。

3. SCOTT模式的使用。

SCOTT是Oracle内部的一个示例用户, 默认口令为tiger, 其下面有表emp和表dept等, 这些表和表间的关系演示了关系型数据库的一些基本原理, Oracle举例说明时一般都用这个用户, 一些关于Oracle的书、教材上一般也都用这个用户来讲解。它对于Oracle本身不是必须的, 如果不想用可以删除。

步骤详解

1. 本案例涉及数据库操作, 因此在开发Java程序之前应进行相应的准备工作。首先要确保系统安装了Oracle, 然后按照如下步骤完成测试环境的配置。

(1) 创建SCOTT/TIGER模式: 依次单击“开始”/“程序”/“Oracle - OraHome92”/“Application Development”/“SQL Plus”, 进入SQL*Plus窗口, 执行以下语句:

```
SQL> @C:\Oracle\Ora92\SQLPlus\Demo\DemoBld.SQL;
```

DemoBld.SQL会创建5个表并填入数据。执行结束后, 它会自动退出SQL*Plus, 所以运行完这个脚本后SQL*Plus窗口将消失。

(2) 定义引用完整性: 上一步创建的5个标准演示表上没有定义任何引用完整性, 因此在运行完DemoBld.SQL后, 再执行以下语句:

```
SQL> ALTER TABLE Emp ADD CONSTRAINT Emp_PK PRIMARY KEY(EmpNo);
表已更改。
```

```
SQL> ALTER TABLE Dept ADD CONSTRAINT Dept_PK PRIMARY KEY(DeptNo);
表已更改。
```

```
SQL> ALTER TABLE Emp ADD CONSTRAINT Emp_FK_Dept FOREIGN KEY(DeptNo) REF-
ERENCES Dept;
表已更改。
```

```
SQL> ALTER TABLE Emp ADD CONSTRAINT Emp_FK_Emp FOREIGN KEY(Mgr) REFER-
ENCES Emp;
表已更改。
```


2. 准备工作完成后, 创建基于Java应用程序的项目oracleconnection。
3. 安装、配置Oracle JDBC驱动程序。操作步骤与案例3类似, 选择的JAR文件是oracle\ora92\jdbc\lib\ojdbc14.jar。
4. 为OracleConnection类添加带参数构造方法:

```
public OracleConnection(String db,String id,String pwd) {
    dbName=db;
    userID=id;
    userPWD=pwd;
    beginConnect();      //连接数据库
}
```

5. 加载Oracle驱动, 使用OCI8连接数据库:

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
conn=DriverManager.getConnection("jdbc:oracle:oci8:@" +dbName,userID,userPWD);
```

程序源代码与解释

```
/* * OracleConnection.java */
public class OracleConnection {
    private Connection conn;    //连接对象
    private String dbName;     //实例
    private String userID;     //用户名
    private String userPWD;    //口令
    /** Creates a new instance of OracleConnection */
    public OracleConnection() { }
    public OracleConnection(String db,String id,String pwd) {
        dbName=db;
        userID=id;
        userPWD=pwd;
        beginConnect();        //连接数据库
    }
    /*- *返回一个Connection对象 */
    public Connection getConnection(){return conn;}
    /*- *连接数据库, 成功后返回1, 否则返回0 */
    public int beginConnect() {
        try {    //加载一个Oracle驱动
            DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
            //使用OCI8连接到数据库
            conn=DriverManager.getConnection("jdbc:oracle:oci8:@" +dbName,userID,userPWD);
            Statement stmt = conn.createStatement();
            String query = "select * from dept";
            ResultSet rs = stmt.executeQuery(query);
            ResultSetMetaData rsmd = rs.getMetaData();
            int columnCount = rsmd.getColumnCount();
            while (rs.next()) {
                for (int i = 1; i <= columnCount; i++) {
                    if (i > 1) System.out.print(", ");
                    System.out.print(rs.getString(i));
                }
                System.out.println();
            }
        }
    }
}
```



```
        rs.close();
        stmt.close();
        conn.close();
        return 1;
    }
    catch(SQLException e) {    //捕捉SQL违例
        System.out.println("Ora9iConnect在连接oracle9数据库时捕获");
        while (e!=null) {
            System.out.println("SQLState:"+e.getSQLState());
            System.out.println("Message :"+e.getMessage());
            System.out.println("Vendor  :"+e.getErrorCode());
            e=e.getNextException();
            System.out.println(" ");
        }
        conn=null;
        return 0;
    }
}

public static void main(String[] args) {
    OracleConnection myconn=new OracleConnection("", "scott", "tiger");
}
}
```



案例5：网吧计费系统

案例运行效果与操作

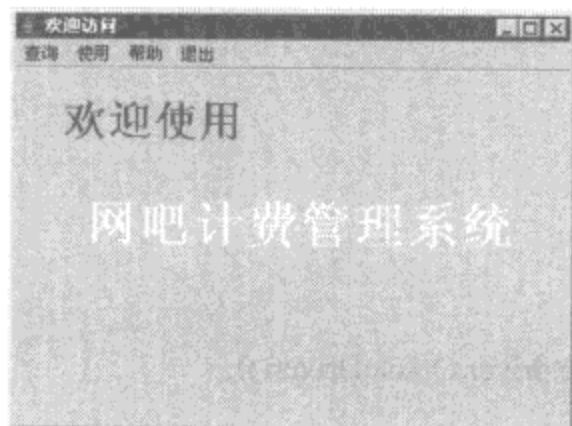


图6-25 欢迎界面

本案例是简单的网吧计费管理应用，主要实现了计算机上机、下机管理，账号使用记录和账号余额查询，应用到数据库的查询、更新、检索等操作。本例使用的数据库为MS SQL Server 2000。程序运行后，界面如图6-25所示。

本案例系统功能有“查询”、“使用”、“帮助”和“退出”。

“查询”功能中有余额查询和记录查询，如图6-26所示。

单击余额查询，其界面如图6-27所示。输入卡号和密码，验证通过后，显示余额查询结果，如图6-28所示。若单击记录查询，结果显示如图6-29所示，结束时间没有标明是因为本卡持有人正在上机。

“使用”功能中有上机和下机，其中上机界面如图6-30所示。在下拉菜单中选择机器号码，输入卡号和密码，就可开始上机，如图6-31所示。

下机界面如图6-32所示。

单击“确定”按钮后，显示下机状态，提示使用时间及卡内余额，如图6-33所示。



图6-26 功能子菜单

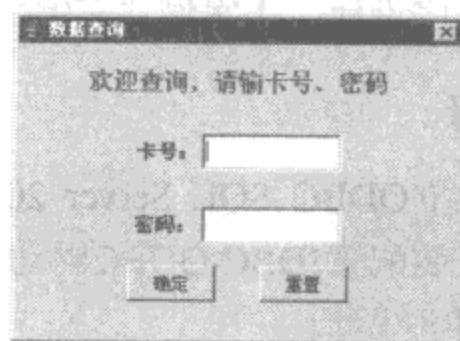


图6-27 数据查询界面

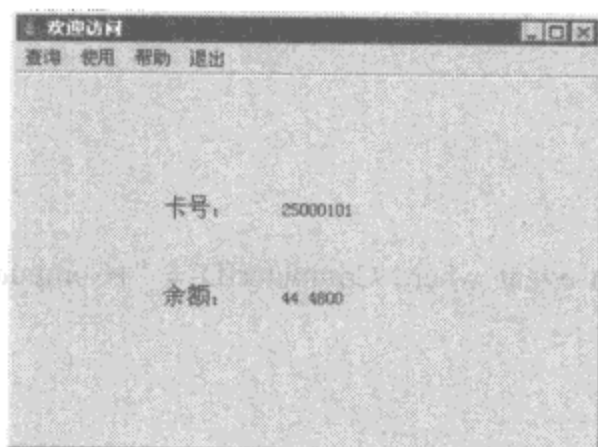


图6-28 余额查询结果

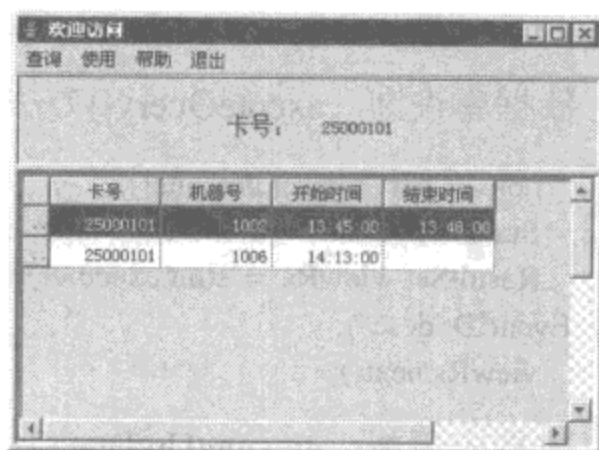


图6-29 记录查询结果

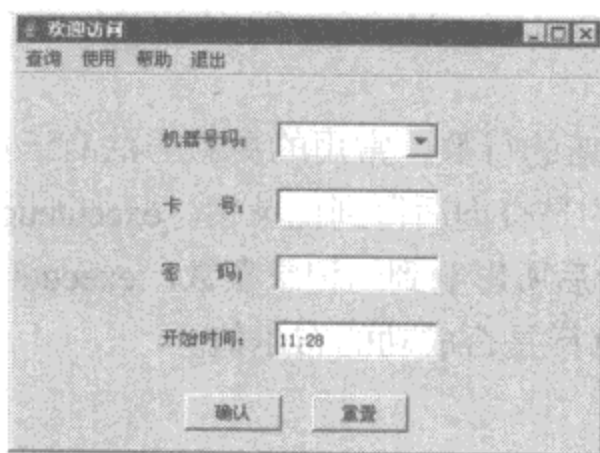


图6-30 上机界面

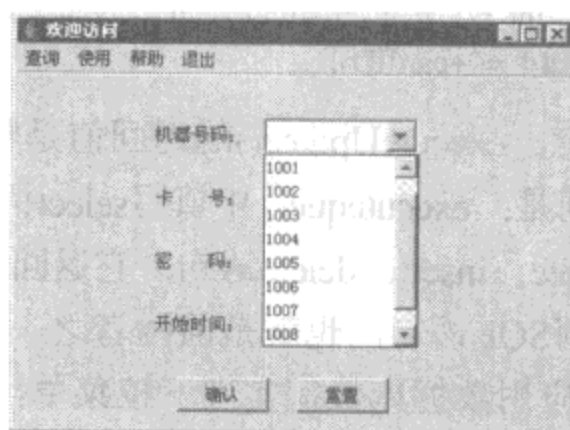


图6-31 选择上机

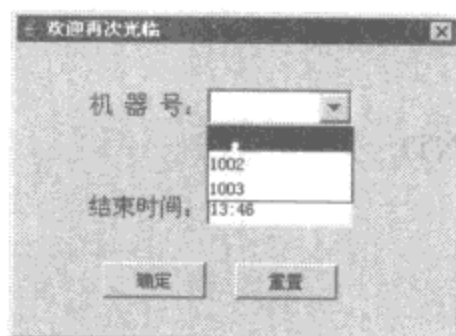


图6-32 下机界面

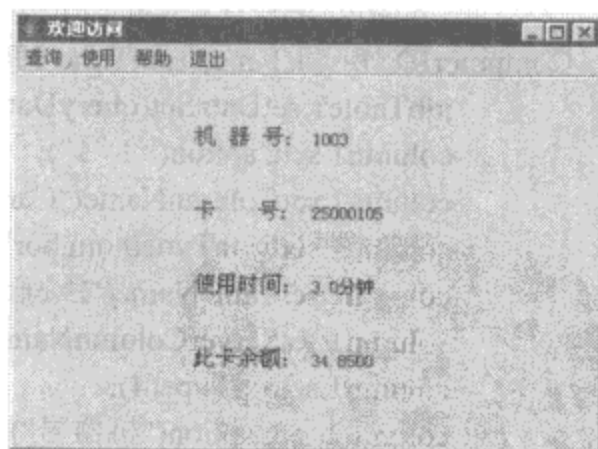


图6-33 下机状态

制作要点

1. JFrame的应用技巧。
2. ResultSet的用法。
3. TableScrollPane()的用法。
4. 数据库的查询、更新等操作。

5. 程序设计：本案例共有9个类，其中barApp是主类。

步骤详解

1. 建立ODBC SQL Server 2000数据源dbbar。
2. 安装配置JDBC-ODBC驱动环境。
3. 界面设计。
4. 数据库连接：

```
return DriverManager.getConnection(url,username,password);
```

5. 数据库查询，executeQuery()方法：

```
con = barConnect.getconn();
Statement stmt = con.createStatement();
ResultSet viewRs = stmt.executeQuery("select * from event where ComputerID = "+ComputerID+"
order by EventID desc");
viewRs.next();
```

6. 数据库更新，executeUpdate()方法：

```
Statement insertstmt = con.createStatement();
int rs1 = insertstmt.executeUpdate("update Card set Card.LeaveMoney = Card.LeaveMoney-"+cost+"
where CardId =" +cardID);
```

注意，executeUpdate()的返回值是整数，表示更新的行数。常用的更新方法有三种，它们的区别是：executequery中填写select语句，它返回的是查询后得到记录集；executeupdate中填写update、insert、delete语句，它返回的是语句执行后所影响到的记录条数；execute中可以填写任何SQL语句，也就是前两者之一，返回的是执行是否成功的布尔值。

7. 应用数据库数据实现下拉菜单：

```
database1.setConnection(new com.borland.dx.sql.dataset.ConnectionDescriptor("jdbc:odbc:dbbar",
"sa", "", false, "sun.jdbc.odbc.JdbcOdbcDriver"));
queryDataSet1.setQuery(new com.borland.dx.sql.dataset.QueryDescriptor(database1, "SELECT
CardID, ComputerID, BeginTime, EndTime FROM Event WHERE CardID = "+CardID, null, true, Load.ALL));
jdbTable1.setDataSet(queryDataSet1);
column1.setCaption("卡号");
column1.setColumnName("CardID");
column1.setDataType(com.borland.dx.dataset.Variant.INT);
column1.setTableName("Event");
column1.setServerColumnName("CardID");
column1.setSqlType(4);
column2.setCaption("机器号");
column2.setColumnName("ComputerID");
column2.setDataType(com.borland.dx.dataset.Variant.INT);
column2.setTableName("Event");
column2.setServerColumnName("ComputerID");
column2.setSqlType(4);
column3.setCaption("开始时间");
column3.setColumnName("BeginTime");
column3.setDataType(com.borland.dx.dataset.Variant.TIME);
column3.setTableName("Event");
column3.setServerColumnName("BeginTime");
```



```

column3.setSqlType(93);
column4.setCaption("结束时间");
column4.setColumnName("EndTime");
column4.setDataType(com.borland.dx.dataset.Variant.TIME);
column4.setTableName("Event");
column4.setServerColumnName("EndTime");
column4.setSqlType(93);
this.add(jSplitPane1, BorderLayout.CENTER);
jSplitPane1.add(jPanel1, JSplitPane.LEFT);
jPanel1.add(jLabel1, null);
jPanel1.add(jLabel2, null);
jSplitPane1.add(jScrollPane1, JSplitPane.RIGHT);
jScrollPane1.getViewport().add(tableScrollPane1, null);
tableScrollPane1.getViewport().add(jdbTable1, null);
jSplitPane1.setDividerLocation(60);
queryDataSet1.setColumns(new Column[] {column1, column2, column3, column4});
Frame1.login_false();
Frame1.SetCardID(0);

```

程序源代码与解释

```

barApp.java
public class barApp {
    boolean packFrame = false;
    //构造方法
    public barApp() {
        mainFrame frame = new mainFrame();
        if (packFrame) {
            frame.pack();
        }
        else {
            frame.validate();
        }
        //窗口居中
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height) {
            frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
            frameSize.width = screenSize.width;
        }
        frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height - frameSize.height) /
2);

        frame.setVisible(true);
        try {
            jbInit();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    //Main方法
    public static void main(String[] args) {

```



```
try {
    UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
}
catch(Exception e) {
    e.printStackTrace();
}
new barApp();
}
private void jbInit() throws Exception {
}
}
```

本章小结

由于Java语言安全、易使用、易理解和能够自动下载到网络等优点，因此它是数据库应用的一个极好的基础语言。Java与JDBC的结合，使程序员可以只设计一次数据库应用软件后，就能在各种数据库系统上运行，使用JDBC可以很容易地把SQL语句传送到任何关系型数据库中。用Java连接数据库主要有两种方式，一是用JDBC-ODBC桥来连接，二是用相关厂商提供的相应驱动程序来连接。本章通过几个Java连接不同数据库的简单案例程序，讲述了在Java语言中数据库应用的一些基本概念和方法。数据库在J2EE中得到了广泛的应用，因此在第8章中我们将继续给大家提供一些实际应用的例子。



第7章 J2ME技术

本章内容

- 案例1：九宫格游戏
- 案例2：五子棋游戏
- 案例3：手机背单词
- 案例4：用J2ME与ASP建立数据库连接
- 案例5：利用J2ME开发联网程序实例
- 本章小结



案例1：九宫格游戏

案例运行效果与操作

本案例是一个简单的MIDlet，允许游戏者与MIDlet程序之间玩一种称为九宫格的人机游戏。它通过对高级和低级用户界面组件的使用，实现了在多显示屏幕之间进行切换、处理简单的命令、动态适配显示尺寸等功能，并且能够处理键盘事件。游戏者首先选择使用哪种棋子（用圆和叉表示），然后开始游戏。游戏者和MIDlet谁先游戏是随机决定的。每走一步棋，程序都要检查游戏状态，判断游戏是否已经结束。游戏可能的几种结果是：游戏者赢，MIDlet程序赢，或者平局。在应用程序运行期间，双方的得分都能显示出来。游戏者可以随时开始新游戏或者退出游戏。程序运行后，界面如图7-1所示。

按“Launch”软键运行程序，此时首先出现一个闪烁屏幕，显示游戏名称，如图7-2所示。闪烁屏幕显示4秒之后，第一个游戏屏幕开始显示，该屏幕让游戏者选择使用哪种棋子（圆圈还是叉号），界面如图7-3所示。

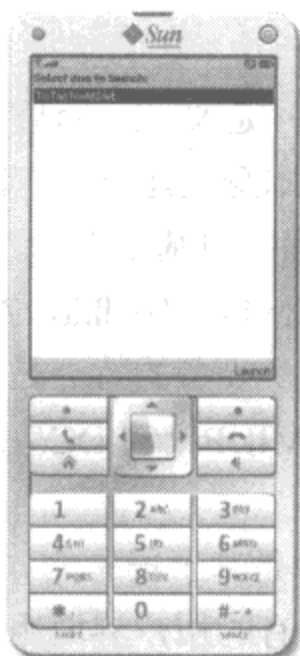


图7-1 运行界面



图7-2 闪烁屏幕

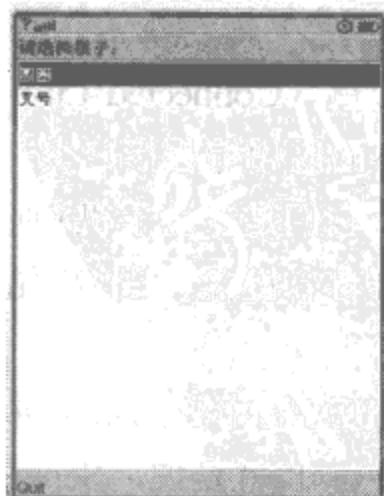


图7-3 选择棋子屏幕

游戏者做出选择后，使用“OK”键（导航键）确认。此时在屏幕右下方出现New菜单，如图7-4所示。按“New”软键运行程序，进入游戏主屏幕，如图7-5所示。

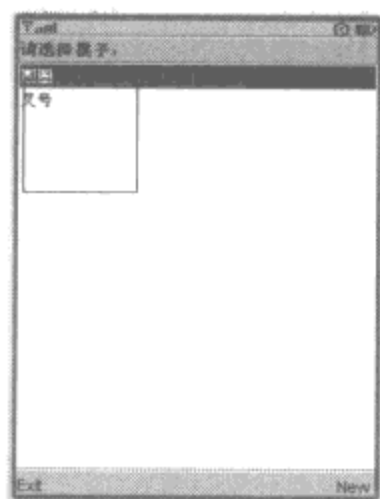


图7-4 选择棋子后的屏幕

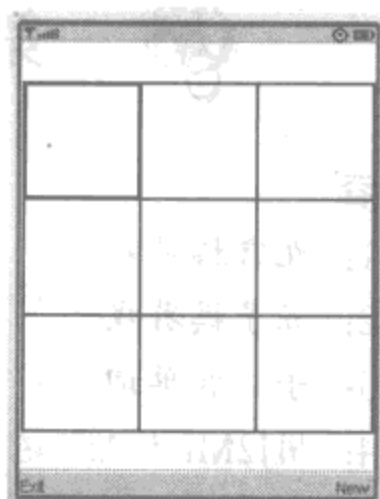


图7-5 游戏主屏幕

每当轮到游戏者下棋的时候，使用箭头键和“OK”键（导航键）来选择想要走的空格，如图7-6所示。每一回合之后，应用程序都会检查游戏的状态，检查其是否符合游戏结束条件并显示游戏结果，如图7-7所示。在游戏过程中，游戏者还可以通过按“Exit”软键结束游戏，或按“New”软键开始新一轮游戏。

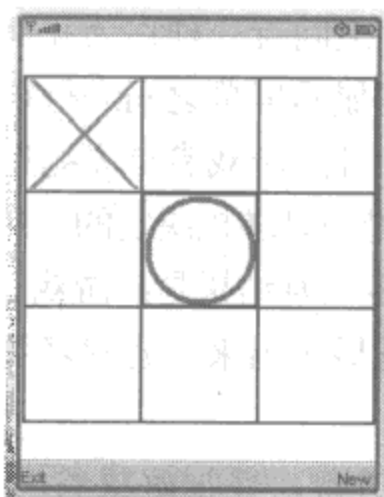


图7-6 游戏界面



图7-7 游戏结束界面

制作要点

1. MIDP图形设计。

移动信息设备描述（Mobile Information Device Profile, MIDP）定义了一套应用编程接口（API），用于运行在MIDP容器中的MIDlet应用程序。这套API本身是建立在有限连接设备配置（Connected Limited Device Configuration, CLDC）应用编程接口基础上的。MIDP用户界面应用编程接口类并不是基于Java抽象窗口工具包（Abstract Window Toolkit, AWT）设计，它们是专为手机和呼机这样的小型移动信息设备而设计的，这类设备的特点是只有很小的屏幕和键盘。当编写MIDP图形应用程序的时候，可能只能使用MIDP或CLDC应用编程接口。

MIDP的中心抽象是屏幕，这句话的含义是MIDP的用户界面设计是基于屏幕的（screen-based）。也就是说，Screen类封装了设备特定的图形和用户交互，所有的用户界面组件都位于屏幕上，并且一次只显示一个屏幕，而且只能浏览或使用这个屏幕上的条目。由屏幕来处理所有的用户界面事件，并只把高级事件传送给应用。采取这种面向屏幕（screenoriented）

的方式，主要是因为移动设备的显示屏幕和键盘种类太多了，几乎每个厂家都有所不同。

2. MIDP应用编程接口的高级用户界面类和低级用户界面类。

MIDP应用编程接口具有高级用户界面类和低级用户界面类。高级用户界面类（例如Form、List、TextBox、TextField、Alert及Ticker）可被适配到设备上，支持图像、文本、文本输入域、单选按钮等。低级用户界面类（Canvas类）允许开发者根据需要绘制任意图形。MIDlet可以运行在各种不同尺寸的彩色、不同灰度等级或黑白屏幕的手机上。高级用户界面类是通用用户界面元素的抽象，它的用途在于提高MIDlet跨不同设备的移植性，并且可以使用本地设备的外观表现。低级应用编程接口则能够更直接地控制显示内容，但是MIDlet设计者应该确保其在不同设备（显示尺寸、键盘、色彩等）上的可移植性。本案例既用到了高级应用编程接口又用到了低级应用编程接口。

所有的MIDP图形用户界面类都是javax.microedition.lcdui程序包的一部分。

3. 程序设计。

本案例是一个简单的MIDlet，它通过对高级和低级用户界面组件的使用，实现了在多显示屏幕之间进行切换、处理简单的命令、动态适配显示尺寸等功能，并且能够处理键盘事件。本案例结构如图7-8所示。

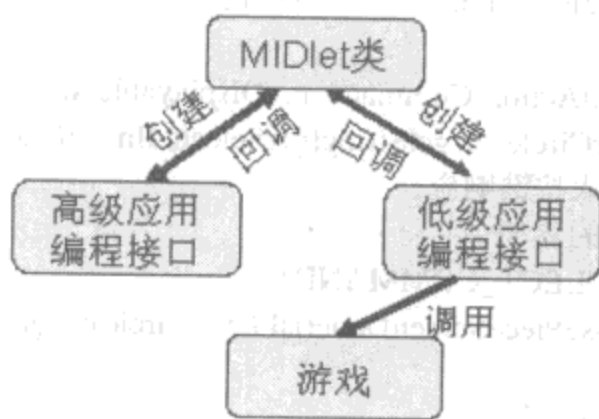


图7-8 九宫格MIDlet的类关系图

步骤详解

1. 在标准的IDE项目中创建新的J2ME应用程序，即Mobile应用程序。
2. 创建继承MIDlet的类，处理MIDlet的生命周期事件。它根据需要创建屏幕对象并且处理来自屏幕的回调：

```

public void startApp() {
    Displayable current = Display.getDisplay(this).getCurrent();
    if (current == null) {
        //首次运行时获取图标（LOGO）图像
        Image logo = null;
        try { logo = Image.createImage("/tictactoe.png"); }
        catch (IOException e) { //获取图像不成功则使用空图像(例如文本等)代替 }
        Alert splashScreen = new Alert(null, "Tic-Tac-Toe\n 九宫格", logo, AlertType.INFO);
        splashScreen.setTimeout(4000); // 延迟4秒
        //在持续显示4秒后进入角色选择界面
        choosePieceScreen = new ChoosePieceScreen(this);
        Display.getDisplay(this).setCurrent(splashScreen, choosePieceScreen);
    }
    else { Display.getDisplay(this).setCurrent(current); }
}

```


3. **ChoosePieceScreen**是一个基于高级应用编程接口窗体的屏幕, 允许游戏者选择圆圈或叉号作为棋子。当游戏者按下“OK”键时, 它使用**MIDlet**的回调方法**choosePieceScreenDone**来处理游戏者的选择。在其构造函数中首先指定当前界面为列表选择方式, 然后通过**append()**将装载的图像与相应的列表文字建立关联, 最后为了响应用户的输入选择还必须调用**setCommandListener()**来检测按键事件的发生, 并在**commandAction()**方法中实现对选定角色的确认。建立**ChoosePieceScreen.java**文件, 其代码如下:

```
public class ChoosePieceScreen extends List implements CommandListener {
    private static final String CIRCLE_TEXT = "圆圈";
    private static final String CROSS_TEXT = "叉号";
    private final TicTacToeMIDlet midlet;
    private final Command quitCommand;
    public ChoosePieceScreen(TicTacToeMIDlet midlet) {
        super("请选择棋子: ", List.IMPLICIT);    //设置列表选择
        this.midlet = midlet;
        append(CIRCLE_TEXT, loadImage("/circle.png"));    //添加图像选项到列表
        append(CROSS_TEXT, loadImage("/cross.png"));
        quitCommand = new Command("Quit", Command.EXIT, 2);
        addCommand(quitCommand);
        setCommandListener(this);    //侦听按键响应
    }
    public void commandAction(Command c, Displayable d) {
        boolean isPlayerCircle = getString(getSelectedIndex()).equals(CIRCLE_TEXT);
        //检测是否为列表按键响应
        //检测用户选中的选项
        if (c == List.SELECT_COMMAND) {
            midlet.choosePieceScreenDone(isPlayerCircle);    //进入游戏画面
        }
        else { //选中quit命令
            midlet.quit(); //退出
        }
    }
    private Image loadImage(String imageFile) {
        Image image = null;
        try {    //装载图像
            image = Image.createImage(imageFile);
        }
        catch (Exception e) { //获取图像不成功则使用空图像(例如文本等)代替}
            return image;
        }
    }
}
```

4. **GameScreen**类负责游戏界面的绘制, 如对棋盘和双方棋子的绘制以及对光标移动的处理等工作。**GameScreen**使用了一个低级应用编程接口**Canvas**屏幕和**Image**、**Graphics**类来绘制游戏面板、棋子以及游戏的最终结果状态。此屏幕适应各种可用显示性能(高、宽、色彩等)。它封装了主游戏程序逻辑的**Game**类。至于对移动光标的处理, 可以先在将要移动到的网格内侧绘制一个新的、四边与棋盘网格紧密相连的黑色矩形框, 然后在原网格位置用原网格背景进行重绘以擦除上次绘制的光标痕迹。在擦除旧光标痕迹时首先需要判断该位置是空白还是绘制有棋子图案, 并根据判断结果绘制白色矩形或是重新装载当前显示的棋子图像。只

要游戏没有结束，上述绘制模块将会多次反复调用执行。如果程序的智能控制部分判断出游戏已经结束并给出胜负结果，则不再显示棋盘界面而是通过下面这段代码以特定的字体在白色画布上绘制出当前战绩。

```
private void paintGameOver(Graphics g) {
    String statusMsg = null;
    if(game.isComputerWinner()) {
        statusMsg = "我赢了!";
        computerGamesWonTally++;
    }
    else if (game.isPlayerWinner()) {
        statusMsg = "你赢了!";
        playerGamesWonTally++;
    }
    else {
        statusMsg = "平局!";
    }

    String tallyMsg = "你的总分:" + playerGamesWonTally + "我的总分:" +
computerGamesWonTally;

    Font font = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_PLAIN, Font.SIZE_MEDIUM);
    int strHeight = font.getHeight();
    int statusMsgWidth = font.stringWidth(statusMsg);
    int tallyMsgWidth = font.stringWidth(tallyMsg);
    int strWidth = tallyMsgWidth;
    if (statusMsgWidth > tallyMsgWidth) {
        strWidth = statusMsgWidth;
    }
    int x = (screenWidth - strWidth) / 2;    //计算字符绘制位置
    x = x < 0 ? 0 : x;
    int y = (screenHeight - 2 * strHeight) / 2; //计算字符绘制位置
    y = y < 0 ? 0 : y;
    //白色清空画布
    g.setColor(WHITE);
    g.fillRect(0, 0, screenWidth, screenHeight);
    //黑色显示信息
    g.setColor(BLACK);
    g.drawString(statusMsg, x, y, (Graphics.TOP | Graphics.LEFT));
    g.drawString(tallyMsg, x, (y + 1 + strHeight), (Graphics.TOP | Graphics.LEFT));
}
```

5. **Game**类封装了九宫格游戏的主要游戏程序逻辑，是整个程序的灵魂。它完成对弈双方落子的合法性检测、计算机行棋的智能计算、游戏结束检测以及对胜负结果的判定等工作。考虑到游戏规则始终是围绕双方棋子的排列形状来进行的，因此可以把棋盘网格作为主要因素进行设计。按从左到右，从上到下的次序从0开始依次对棋盘的9个网格进行编号，可以得出如下几组获胜条件：0, 1, 2; 3, 4, 5; 6, 7, 8; 0, 3, 6; 1, 4, 7; 2, 5, 8; 0, 4, 8; 2, 4, 6。只要一方有三颗棋子的位置符合其中任何一组即可认定该方获胜。在程序实现过程中以**WINS**数组记录上述几种获胜条件，并在每一次行棋完毕后进行比对，以判断游戏是否有获胜方产生。在项目窗口中双击**Game.java**文件，在源代码编辑器中添加代码如下：

```
class Game {
    private static final int[] WINS = {    //WINS数组记录获胜条件
        //水平方向获胜条件
```



```

        bit(0) | bit(1) | bit(2),
        bit(3) | bit(4) | bit(5),
        bit(6) | bit(7) | bit(8),
        //垂直方向获胜条件
        bit(0) | bit(3) | bit(6),
        bit(1) | bit(4) | bit(7),
        bit(2) | bit(5) | bit(8),
        //对角方向获胜条件
        bit(0) | bit(4) | bit(8),
        bit(2) | bit(4) | bit(6) };
private static final int DRAWN_GAME = bit(0) | bit(1) | bit(2) | bit(3) | bit(4) | bit(5) | bit(6)
| bit(7) | bit(8);
private int playerState;
private int computerState;
private Random random;
Game(Random random) {
    this.random = random;
    initialize();
}
void initialize() {
    playerState = 0;
    computerState = 0;
}
boolean isFree(int position) {    //检查是否空位
    int bit = bit(position);
    return (((playerState & bit) == 0) && ((computerState & bit) == 0));
}
void makePlayerMove(int position) {    //游戏者走棋
    playerState |= bit(position);
}
int makeComputerMove() { //计算机走棋（人工智能逻辑）
    int move = getWinningComputerMove();    //如能立即获胜则在获胜位置下子
    if (move == -1) {
        //如游戏者即将获胜则在游戏者将获胜的位置下子
        move = getRequiredBlockingComputerMove();
        if (move == -1) {
            //如双方均暂时无法获胜则下随手棋
            move = getRandomComputerMove();
        }
    }
    computerState |= bit(move);    //当前计算机占用的所有位置
    return move;
}
boolean isGameOver() {    //判断游戏是否结束
    return isPlayerWinner() | isComputerWinner() | isGameDrawn();
}
boolean isPlayerWinner() { //判断是否游戏者胜利
    return isWin(playerState);
}
boolean isComputerWinner() {    //判断是否计算机胜利
    return isWin(computerState);
}
boolean isGameDrawn() {
    return (playerState | computerState) == DRAWN_GAME;
}

```



```

    }
    //如能立即获胜则返回获胜位置, 否则返回 -1
    private int getWinningComputerMove() {
        int move = -1;
        for (int i = 0; i < 9; ++i) {
            if (isFree(i) && isWin(computerState | bit(i))) {
                move = i; //找到获胜位置时中断
                break;
            }
        }
        return move;
    }
    //如游戏者即将获胜则返回游戏者将获胜的位置, 否则返回 -1
    private int getRequiredBlockingComputerMove() {
        int move = -1;
        for (int i = 0; i < 9; ++i) {
            if (isFree(i) && isWin(playerState | bit(i))) {
                move = i;
                break;
            }
        }
        return move;
    }
    //如双方均暂时无法获胜则返回一个随机位置, 如果无位置可下则返回 -1
    private int getRandomComputerMove() {
        int move = -1;
        //计算还有多少步棋可走
        int numFreeSquares = 0;
        for (int i=0; i < 9; ++i) {
            if (isFree(i)) {
                numFreeSquares++;
            }
        }
        //如果有位置可下则返回一个随机位置
        if (numFreeSquares > 0) {
            int pick = ((random.nextInt() << 1) >>> 1) % numFreeSquares;
            for (int i = 0; i < 9; ++i) {
                if (isFree(i)) {
                    if (pick == 0) {
                        move = i;
                        break;
                    }
                    pick--;
                }
            }
        }
        return move;
    }
    private static boolean isWin(int state) { //判断哪一方取胜
        boolean isWinner = false;
        for (int i = 0; i < WINS.length; ++i) {
            if ((state & WINS[i]) == WINS[i]) {
                isWinner = true;
                break;
            }
        }
    }

```



```

    }
    }
    return isWinner;
}
private static int bit(int i) {
    return 1 << i;
}
}

```

程序源代码与解释

```

/** GameScreen.java*/
import java.util.Random;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
class GameScreen extends Canvas implements CommandListener {
    //定义颜色常量（略）
    public GameScreen(TicTacToeMIDlet midlet, boolean playerIsCircle) {
        //代码略
    }
    private void initialize() {
        //初始化游戏界面，代码略
    }
    private void paintGame(Graphics g) {
        if (isRestart) {
            //清空画布
            g.setColor(WHITE);
            g.fillRect(0, 0, screenWidth, screenHeight);
            drawBoard(g);
            isRestart = false;
        }
        drawCursor(g);
        if (playerMove != NO_MOVE) {
            drawPiece(g, playerIsCircle, playerMove);
        }
        if (computerMove != NO_MOVE) {
            drawPiece(g, computerIsCircle, computerMove);
        }
    }
    private void initializeBoard() { //根据屏幕大小计算棋盘网格间距和棋子的大小
        //获取屏幕大小
        screenWidth = getWidth();
        screenHeight = getHeight();
        //计算网格大小
        if (screenWidth > screenHeight) {
            boardCellSize = (screenHeight - 2) / 3;
            boardLeft = (screenWidth - (boardCellSize * 3)) / 2;
            boardTop = 1;
        }
        else {
            boardCellSize = (screenWidth - 2) / 3;
            boardLeft = 1;
            boardTop = (screenHeight - boardCellSize * 3) / 2;
        }
    }
}

```



```

    }
    protected void keyPressed(int keyCode) {
        //游戏者重新启动才能继续玩
        if (game.isGameOver()) {
            return;
        }
        int gameAction = getGameAction(keyCode);
        switch (gameAction) {
            case FIRE: doPlayerMove();
                        break;
            case RIGHT: doMoveCursor(1, 0);
                        break;
            case DOWN: doMoveCursor(0, 1);
                        break;
            case LEFT: doMoveCursor(-1, 0);
                        break;
            case UP: doMoveCursor(0, -1);
                     break;
            default: break;
        }
    }
    private void doPlayerMove() {
        if (game.isFree(cursorPosition)) {
            //游戏者走棋
            game.makePlayerMove(cursorPosition);
            playerMove = cursorPosition;
            //计算机走棋
            if (!game.isGameOver()) {
                computerMove = game.makeComputerMove();
            }
            repaint();
        }
    }
    private void doMoveCursor(int dx, int dy) {
        int newCursorPosition = cursorPosition + dx + 3 * dy;
        if ((newCursorPosition >= 0) && (newCursorPosition < 9)) {
            preCursorPosition = cursorPosition;
            cursorPosition = newCursorPosition;
            repaint();
        }
    }
    //在棋盘上画出棋子（圆圈或者叉号）
    private void drawPiece(Graphics g, boolean isCircle, int pos) {
        int x = ((pos % 3) * boardCellSize) + 3;
        int y = ((pos / 3) * boardCellSize) + 3;
        if (isCircle) {
            drawCircle(g, x, y);
        }
        else {
            drawCross(g, x, y);
        }
    }
    //在棋盘上画出蓝色圆圈
    private void drawCircle(Graphics g, int x, int y) {

```



```

        g.setColor(BLUE);
        g.fillArc(x + boardLeft, y + boardTop, boardCellSize - 4, boardCellSize - 4, 0, 360);
        g.setColor(WHITE);
        g.fillArc(x + 4 + boardLeft, y + 4 + boardTop, boardCellSize - 4 - 8, boardCellSize - 4
- 8, 0, 360);
    }
    // 在棋盘上画出红色叉号
    private void drawCross(Graphics g, int x, int y) {
        g.setColor(RED);
        for (int i = 0; i < 4; i++) {
            g.drawLine(x + 1 + i + boardLeft, y + boardTop, x + boardCellSize - 4 - 4 + i +
boardLeft, y + boardCellSize - 5 + boardTop);
            g.drawLine(x + 1 + i + boardLeft, y + boardCellSize - 5 + boardTop, x +
boardCellSize - 4 - 4 + i + boardLeft, y + boardTop);
        }
    }
    //在棋盘上显示游戏者选中的网格（对移动光标的处理）
    private void drawCursor(Graphics g) {
        //在游戏者选中的网格中画出白色光标
        g.setColor(WHITE);
        g.drawRect(((preCursorPosition % 3) * boardCellSize) + 2 + boardLeft, ((preCursorPosition/
3) * boardCellSize) + 2 + boardTop, boardCellSize - 3, boardCellSize - 3);
        //在游戏者选中的网格中画出黑色光标
        g.setColor(BLACK);
        g.drawRect(((cursorPosition % 3) * boardCellSize) + 2 + boardLeft, ((cursorPosition/3) *
boardCellSize) + 2 + boardTop, boardCellSize - 3, boardCellSize - 3);
    }
    private void drawBoard(Graphics g) {    //绘制棋盘
        //用背景色清空整个画布
        g.setColor(WHITE);
        g.fillRect(0, 0, screenWidth, screenHeight);
        //分别按行列绘制出黑色网格
        g.setColor(BLACK);
        for (int i = 0; i < 4; i++) {
            g.fillRect(boardLeft, boardCellSize * i + boardTop, (boardCellSize * 3) + 2, 2);
            g.fillRect(boardCellSize * i + boardLeft, boardTop, 2, boardCellSize * 3);
        }
    }
}

```



案例2：五子棋游戏

案例运行效果与操作

本案例是一个简单的MIDlet，简单实现了两个人对战的五子棋游戏功能。程序运行后，界面如图7-9所示。

按“Launch”软键运行程序，进入游戏开始界面，如图7-10所示。此时棋盘外框为一个3个像素宽的白框，表明当前下棋方为白棋方。棋盘是一个15×15的网格，中间为一个蓝色的选择框。按下方向键可以移动选择框，按下确认键则在当前选择框所在位置下棋，界面如图7-11和图7-12所示。此时棋盘外框变为红色框，表明当前下棋方切换为红棋方。这样双方依

次下棋，直到游戏结束。每一回合之后，应用程序都会检查游戏的状态，检查其是否符合游戏结束条件并显示游戏结果。在游戏过程中，游戏者还可以通过按“退出”软键结束游戏，或使用“重新开始”软键开始新一轮游戏。

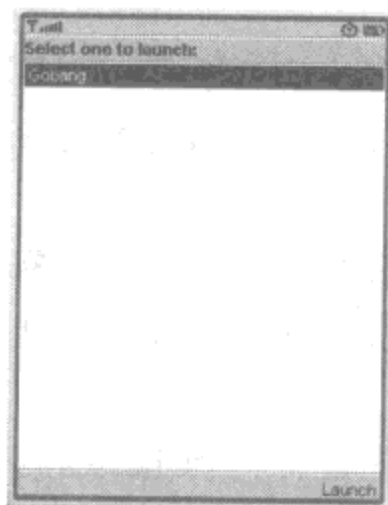


图7-9 初始界面

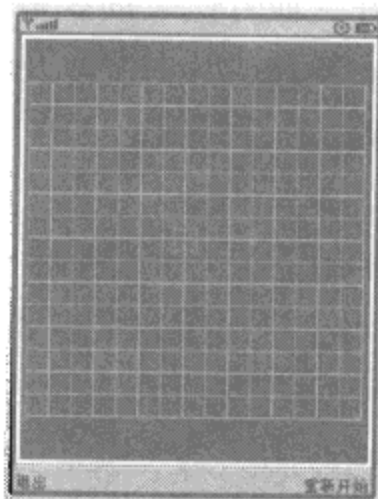


图7-10 游戏开始界面

当任意一方满足胜利条件时，会出现游戏结束界面，如图7-13所示。此时按“Done”软键即可返回到如图7-10所示的界面，开始新一轮游戏。

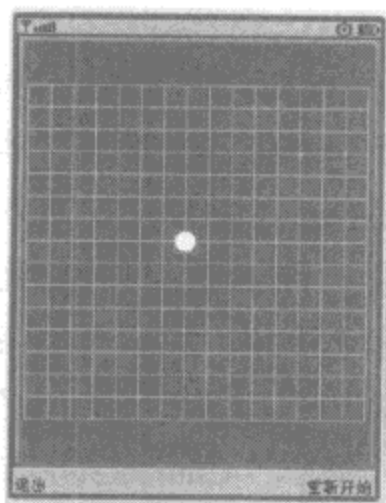


图7-11 白棋下棋界面

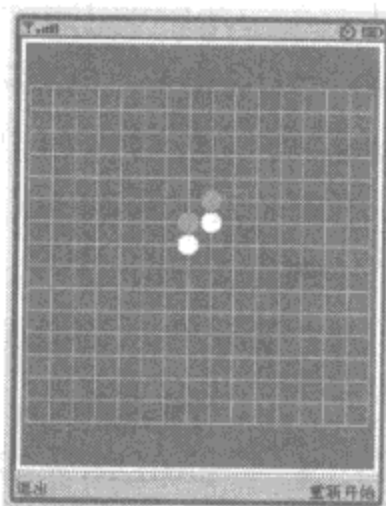


图7-12 红棋下棋界面

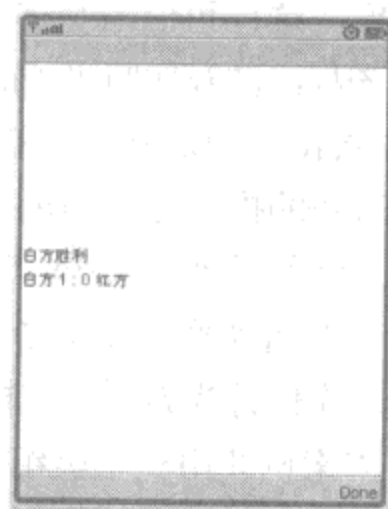


图7-13 游戏结束界面

制作要点

1. MIDlet高级屏幕。

包括简单的高级屏幕类，例如List、TextBox、TextField、Alert等。用户不能添加额外的图形用户界面组件到这种类型的屏幕中。

一般的Form屏幕类和List类很相似，但是它允许使用额外的图形元素，例如图像、只读文本域、可编辑文本域、可编辑数据域、标尺和选项组。Form条目可以任意地被添加或删除。

高级屏幕的使用保证了基于高级应用编程接口类的用户界面对象的可移植性和适用性。本案例使用一个Alert显示棋局输赢以及当前的比分：

```
Alert winAlert=new Alert("",winner+"\n白方 "+player1win+" : "+player2win+" 红方",null,AlertType.INFO);
```

2. MIDP低级屏幕。

Canvas（画布）屏幕（和Graphics、Image类）可以用来编写基于低级应用编程接口的用户界面。这些类给予MIDlet程序员很大程度的绘画灵活性。程序员可以绘制各种类型的图形

元素，例如线、弧、矩形、圆角矩形、圆、文字（不同颜色、字体、大小）、位图剪辑等。大部分的游戏MIDlet是使用基于画布屏幕类的主图形用户界面元素编写的。

像Canvas和Graphics这样的低级类为程序员提供了更大的自由空间让其控制用户界面的视觉表现，并且侦听低级键盘事件。这种情况下，程序员需要确保应用程序在不同特性（例如显示尺寸、彩色或黑白，以及不同键盘类型）的移动设备上的可移植性。比如说，有可能需要使用getWidth()和getHeight()方法调节用户界面外观使其适应一个或更多设备的可用Canvas尺寸。

一个MIDlet用户界面通常包含一个或多个屏幕。因为每次只能显示一个屏幕，因此MIDlet具有良好设计的结构是非常重要的，这样就能更加容易地处理屏幕之间内容的切换。

3. 程序设计。

数据结构：由于五子棋是一个二维棋类游戏，所以首先定义一个Chesses类来表示棋子，Chesses有一个boolean型的变量isPlayer1来区分该棋子是哪方下的，然后用一个Chess类型的二维数组来包含棋盘上的所有棋子。考虑到移动设备的资源有限，尽可能减少系统资源占用，考虑不在数组建立后直接生成数组的每一个对象，而是把每一个棋子对象（Chesses）放在游戏的进行中生成分，也就是说在游戏进行时，游戏者每下一步棋，在数组相应位置生成该棋子的对象，这样可以避免还没有下的棋子在一开始就占用了系统内存。

流程：游戏按照棋子的二维数组进行绘制棋子，游戏者下棋后，程序修改数组相应位置，设置isPlayer1值，然后重新绘制（repaint），就更新了棋盘界面。由于游戏的功能简单，也为了使游戏的操作尽可能的简便，所以不在游戏进入时设计菜单，而是直接开始对战，在对战界面，设置了“重新开始”和“退出”按钮。即运行即玩，一键开始，一键重来，一键退出。

游戏者切换：棋类游戏有一个问题需要注意，就是提示当前由哪方下棋，为了节省界面空间，简化游戏界面，在棋盘外围加一个3个像素宽的框，框的颜色就是当前下棋方的颜色。

本案例共由三个类组成，分别是：继承了MIDlet类的Gobang类、继承了Canvas类的游戏核心类GobangCanvas类以及棋子类Chesses类。

步骤详解

1. 创建基于Mobile的Mobile应用程序项目，新建MIDlet。

2. Gobang类继承自MIDlet类，用于连接设备的应用程序管理器（Application Manager），通过方法startApp、pauseApp、destroyApp来通知游戏的开始、暂停和结束：

```
public class Gobang extends MIDlet {
    GobangCanvas gobang;//定义游戏界面的Canvas类GobangCanvas的对象gobang
    public Gobang() {
        super();
        gobang=new GobangCanvas(this);    //生成GobangCanvas类的对象gobang
    }
    protected void startApp(){
        //在屏幕上绘出游戏界面gobang
        Display.getDisplay(this).setCurrent(gobang);
    }
}
```

3. 棋子类Chesses.java定义了一个棋子，棋盘上的每一个棋子都对应着一个Chesses的对象，整个棋盘是一个Chesses类型的二维数组：


```

public class Chesses {
    boolean isPlayer1;
    public Chesses() {}
    public Chesses(boolean isPlayer1) {
        this.isPlayer1=isPlayer1;
    }
}

```

4. 游戏界面类GobangCanvas是游戏的核心类，继承自Canvas，此类完成游戏的逻辑、绘图、控制、互动等所有功能。首先为GobangCanvas类添加构造方法和绘制棋盘方法，棋盘是正方形，但屏幕是矩形的，所以棋盘边长要按短边计算，但短边未必是棋盘格子数的整数倍，因此棋盘边长=短边-短边%格子数。另外，因为棋盘要居中，所以在计算左上角坐标时，也要把留空（Empty）除以2。代码如下：

```

public GobangCanvas(Gobang gobang){ //带参数构造方法
    this.gobang=gobang;
    newGame=true;
    empty=10; //游戏界面到屏幕边缘的留空
    canvasW=getWidth()-empty; //画布的长
    canvasH=getHeight()-empty; //画布的宽
    chessMapGrid=15;
    chesses=new Chesses[chessMapGrid+1][chessMapGrid+1];
    if(canvasW>canvasH){ //棋盘边长要按短边计
        //棋盘边长=短边 - 短边 % 格子数
        chessMapLength=canvasH-canvasH%chessMapGrid;
        chessMapX=(canvasW-chessMapLength)/2+empty/2; //棋盘左上角X坐标
        chessMapY=(canvasH%chessMapGrid)/2+empty/2; //棋盘左上角Y坐标
    }
    else{
        //棋盘边长=短边 - 短边 % 格子数
        chessMapLength=canvasW-canvasW%chessMapGrid;
        chessMapX=(canvasW%chessMapGrid)/2+empty/2; //棋盘左上角X坐标
        chessMapY=(canvasH-chessMapLength)/2+empty/2; //棋盘左上角Y坐标
    }
    chessGridLength=chessMapLength/chessMapGrid;
    chessLength=chessGridLength-1;
    selectedX=selectedY=chessMapGrid/2;
    isPlayer1=true;
    restartCmd = new Command("重新开始", Command.SCREEN, 0);
    exitCmd = new Command("退出", Command.EXIT, 0);
    addCommand(restartCmd); //添加“开始”按钮
    addCommand(exitCmd); //添加“退出”按钮
    setCommandListener(this); //添加侦听器
}

protected void paintMap(Graphics g){ //根据屏幕大小绘制棋盘
    for(int i=0;i<chessMapGrid;i++){
        for(int j=0;j<chessMapGrid;j++){
            g.setColor(128,128,128);
            g.drawRect(chessMapX+j*chessGridLength,
                chessMapY+i*chessGridLength,
                chessGridLength,chessGridLength);
        }
    }
}

```


5. 为GobangCanvas类添加命令按钮事件处理方法。在游戏中需要有两个按钮，即“重新开始”和“退出”按钮，在源代码编辑器中添加代码如下：

```
public void commandAction(Command c, Displayable d) {    //命令按钮事件处理
    if (c == exitCmd) {    //按下“退出”按钮
        gobang.destroyApp(false);
        gobang.notifyDestroyed();
    } else if (c == restartCmd) {    //按下“重新开始”按钮
        init();    //调用init方法
        repaint();    //重新绘制
    }
}
```

6. 为GobangCanvas类添加按键事件处理方法。在游戏中需要记录5种按键动作，即玩家控制选择框的上下左右操作和走棋，代码如下：

```
protected synchronized void keyPressed(int keyCode) {    //按键事件处理
    int action = getGameAction(keyCode);
    if (action == Canvas.LEFT) {    //按下左方向键
        selectedX=(--selectedX+chessMapGrid+1)%(chessMapGrid+1);
    }
    else if (action == Canvas.RIGHT) {    //按下右方向键
        selectedX=(++selectedX)%(chessMapGrid+1);
    }
    else if (action == Canvas.UP) {    //按下上方向键
        selectedY=(--selectedY+chessMapGrid+1)%(chessMapGrid+1);
    }
    else if (action == Canvas.DOWN) {    //按下下方向键
        selectedY=(++selectedY)%(chessMapGrid+1);
    }
    else if (action == Canvas.FIRE) {    //按下确认（OK）键
        if(chesses[selectedY][selectedX]==null){    //走棋
            chesses[selectedY][selectedX]=new Chesses(this.isPlayer1);
            if(checkWin()){    //检查棋局的输赢
                String winner;
                if(isPlayer1){
                    winner="白方胜利";
                    player1win++;
                }
                else{
                    winner="红方胜利";
                    player2win++;
                }
            }
            try{
                Thread.sleep(3000);    //线程停止3秒，然后显示棋局的输赢
            } catch (Exception e) {
            }
            //用一个Alert显示哪一方赢以及当前的比分
            Alert winAlert=new Alert("",winner+"\n白方 "+player1win+" : "+player2win
            +" 红方",null,AlertType.INFO);

            winAlert.setTimeout(Alert.FOREVER);
            Display.getDisplay(gobang).setCurrent(winAlert,this);
            //返回后开始新的一局
            init();
            repaint();
        }
    }
}
```



```

        this.isPlayer1=!this.isPlayer1;//切换下棋方
    }
    repaint();
}

```

7. 为GobangCanvas类添加判断游戏输赢的checkWin方法。每次在游戏者走棋(按下确认键)时, 首先判断棋局的输赢, 然后用一个Alert显示哪一方获胜以及当前的比分, 返回后开始新的一局。判断的逻辑是, 在当前所下的棋子的0度/180度、90度/270度、45度/225度、135度/315度四个方向上分别按照由近至远的顺序往两头判断各5个棋子是否是当前下棋方的棋子, 如果是则累加到一个变量上, 如果在到达5之前出现“否”的情况, 则中止在这一方向或这一角度的判断, 变量归1 (因为当前棋子肯定是当前下棋方下完的棋子) 并进行下一个方向或角度的判断。在判断是否是当前方时, 用当前isPlayer1变量和棋子对象的isPlayer1变量进行比较。在源代码编辑器中添加代码如下:

```

private boolean checkWin(){//检查棋局的输赢
    int num=1;
    if(num<5){
        num=1;
        //在当前所下的棋子的180度方向上按照由近至远的顺序判断各5个棋子是否是当前下棋方的棋子
        for(int i=1;i<=4;i++){
            if(isPlayer1(selectedX-i,selectedY)){
                num++;
            }
            else break;
        }
        //在当前所下的棋子的0度方向上按照由近至远的顺序判断各5个棋子是否是当前下棋方的棋子
        for(int i=1;i<=4;i++){
            if(isPlayer1(selectedX+i,selectedY)){
                num++;
            }
            else break;
        }
    }
    if(num<5){
        num=1;
        //在当前所下的棋子的270度方向上按照由近至远的顺序判断各5个棋子是否是当前下棋方的棋子
        for(int i=1;i<=4;i++){
            if(isPlayer1(selectedX,selectedY-i)){
                num++;
            }
            else break;
        }
        //在当前所下的棋子的90度方向上按照由近至远的顺序判断各5个棋子是否是当前下棋方的棋子
        for(int i=1;i<=4;i++){
            if(isPlayer1(selectedX,selectedY+i)){
                num++;
            }
            else break;
        }
    }
    if(num<5){
        num=1;
        //在当前所下的棋子的225度方向上按照由近至远的顺序判断各5个棋子是否是当前下棋方的棋子
        for(int i=1;i<=4;i++){

```



```

        if(isPlayer1(selectedX-i,selectedY-i))
            num++;
        else break;
    }
    //在当前所下的棋子的45度方向上按照由近至远的顺序判断各5个棋子是否是当前下棋方的棋子
    for(int i=1;i<=4;i++){
        if(isPlayer1(selectedX+i,selectedY+i))
            num++;
        else break;
    }
}
if(num<5){
    num=1;
    //在当前所下的棋子的315度方向上按照由近至远的顺序判断各5个棋子是否是当前下棋方的棋子
    for(int i=1;i<=4;i++){
        if(isPlayer1(selectedX+i,selectedY-i))
            num++;
        else break;
    }
    //在当前所下的棋子的135度方向上按照由近至远的顺序判断各5个棋子是否是当前下棋方的棋子
    for(int i=1;i<=4;i++){
        if(isPlayer1(selectedX-i,selectedY+i))
            num++;
        else break;
    }
}
if(num>=5)
    return true;
else
    return false;
}

private boolean isPlayer1(int y,int x){    //判断是否是玩家1
    if(x<=15 && x>=0 && y<=15 && y>=0 && chesses[x][y]!=null){
        if(chesses[x][y].isPlayer1==this.isPlayer1)
            return true;
        else
            return false;
    }
    else return false;
}
}

```

程序源代码与解释

```

/* * GobangCanvas.java*/
public class GobangCanvas extends Canvas implements CommandListener{
    protected Gobang gobang;
    int empty;    //游戏界面到屏幕边缘的留空
    int canvasW,canvasH;    //画布的长和宽
    int chessLength;    //棋子的直径
    int chessMapLength,chessMapGrid,chessGridLength;
    //棋盘的边长, 棋盘一边格子数, 每格宽度
    int chessMapX,chessMapY;    //棋盘左上角x,y坐标
    int selectedX,selectedY;    //选择框在棋盘格局上的x,y位置
    boolean isPlayer1;    //是否是游戏者1
    Chesses[][] chesses;    //棋子数组
}

```



```

boolean newGame;           //是否是新的游戏
Command exitCmd;           //“退出”命令按钮
Command restartCmd;        //“开始”命令按钮
int player1win,player2win;
public GobangCanvas(Gobang gobang){ //带参数构造方法
//代码略}
private void init(){ //初始化棋盘，把棋盘清空，重新开始游戏
    if(newGame){
        chesses=new Chesses[chessMapGrid+1][chessMapGrid+1];
        isPlayer1=true;
        selectedX=selectedY=chessMapGrid/2;
    }
}
public void commandAction(Command c, Displayable d) { //命令按钮事件处理
    if (c == exitCmd) { //按下“退出”按钮
        gobang.destroyApp(false);
        gobang.notifyDestroyed();
    }else if(c==restartCmd){ //按下“重新开始”按钮
        init(); //调用init方法
        repaint(); //重新绘制
    }
}
private boolean isPlayer1(int y,int x){ //判断是否是游戏者1
    if(x<=15 && x>=0 && y<=15 && y>=0 && chesses[x][y]!=null){
        if(chesses[x][y].isPlayer1==this.isPlayer1)
            return true;
        else
            return false;
    }
    else return false;
}
protected void paintSelected(Graphics g){ //绘制选择框
    g.setColor(0,0,255); //蓝色选择框
    g.drawRect(chessMapX+selectedX*chessGridLength-chessGridLength/2,
        chessMapY+selectedY*chessGridLength-chessGridLength/2,
        chessGridLength,chessGridLength);
}
protected void paintChesses(Graphics g){ //按照棋子二维数组绘制已经下了的棋子
    for(int i=0;i<=chessMapGrid;i++){
        for(int j=0;j<=chessMapGrid;j++){
            if(chesses[i][j]!=null){
                if(chesses[i][j].isPlayer1)
                    g.setColor(255,255,255); //白棋
                else
                    g.setColor(255,0,0); //红棋
                g.fillArc(chessMapX+j*chessGridLength-chessLength/2,
                    chessMapY+i*chessGridLength-chessLength/2,
                    chessLength,chessLength,0,360);
            }
        }
    }
}
protected void paintPlayer(Graphics g,boolean isPlayer1){ //绘制游戏者提示框
    if(isPlayer1)
        g.setColor(255,255,255); //白色外框

```



```

else
    g.setColor(255,0,0);    //红色外框
    //根据设定的颜色画出3个像素宽的外框
    g.drawRect(1,1,getWidth()-2,getHeight()-2);
    g.drawRect(2,2,getWidth()-4,getHeight()-4);
    g.drawRect(3,3,getWidth()-6,getHeight()-6);
}
protected void paint(Graphics g) {    //把所有部件的绘制汇总在paint()方法中
    g.setColor(0x00000000);    //设置背景色为黑色
    g.fillRect(0, 0, getWidth(), getHeight());    //用背景色填充整个屏幕
    paintPlayer(g,isPlayer1);    //绘制提示框
    paintMap(g);    //绘制棋盘
    paintSelected(g);    //绘制选择框
    paintChesses(g);    //绘制棋子
}
}

```



案例3：手机背单词

案例运行效果与操作

本案例是一个简单的MIDlet，实现了背单词软件的基本功能。程序运行后，界面如图7-14所示。

按“Launch”软键运行程序，此时首先出现主屏幕，显示一个选择列表，共有三个项目，如图7-15所示。屏幕下方是一些提示信息。



图7-14 运行界面

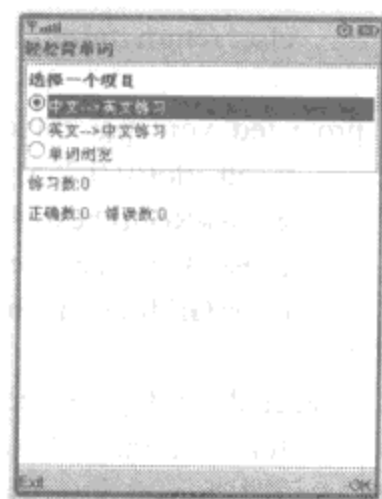


图7-15 选择列表

选择第一个项目“中文→英文练习”后，使用“OK”键确认，此时界面如图7-16所示。屏幕上方列出当前文件包含的单词总数，然后给出一个中文题干，可以选择相应的英文。使用上下方向键选择合适的选项，使用“OK”键确认后，如果选择正确则给出正确提示信息，如图7-17所示；否则给出错误提示信息，如图7-18所示。按下“Continue”（继续）键会返回如图7-16所示界面接着做题。按下“BACK”（退出）键则返回如图7-15所示的界面。

在如图7-19所示界面选择第二个项目“英文→中文练习”后，使用“OK”键确认，此时界面如图7-20所示。屏幕上方列出当前文件包含的单词总数，然后给出一个英文题干，可以选择相应的中文。使用上下方向键选择合适的选项，使用“OK”键确认后，如果选择正确则给出正确提示信息，如图7-17所示；否则给出错误提示信息，如图7-18所示。

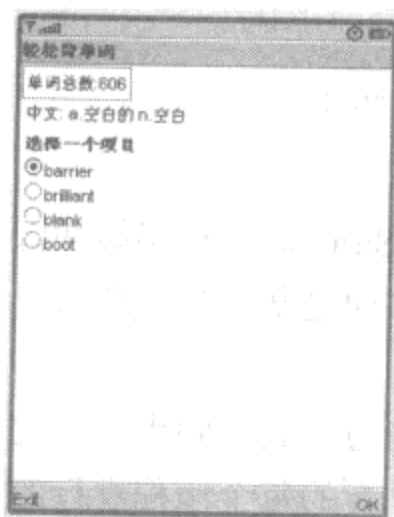


图7-16 第一个项目界面



图7-17 正确选择界面

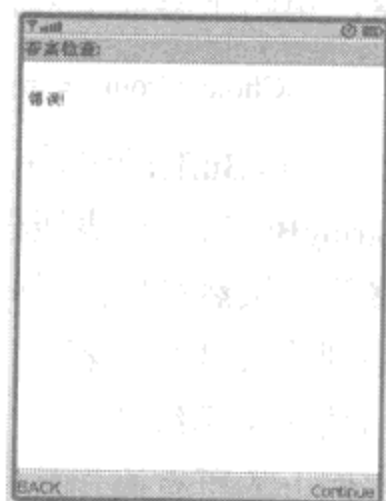


图7-18 错误选择界面

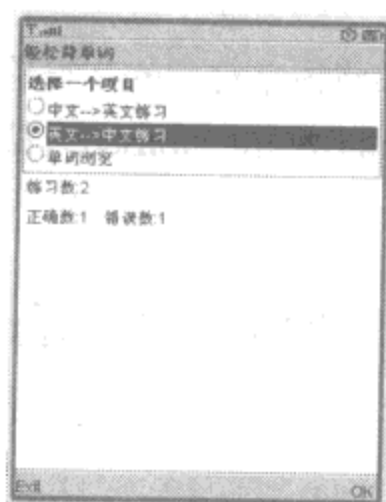


图7-19 选择第二个项目

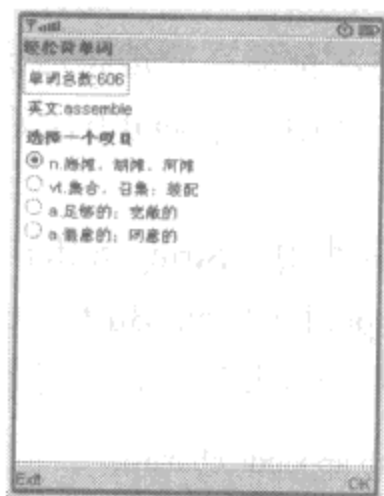


图7-20 第二个项目界面

在如图7-21所示界面选择第三个项目“单词浏览”后，使用“OK”键确认，此时界面如图7-22所示。此时会列出所有词条的内容。

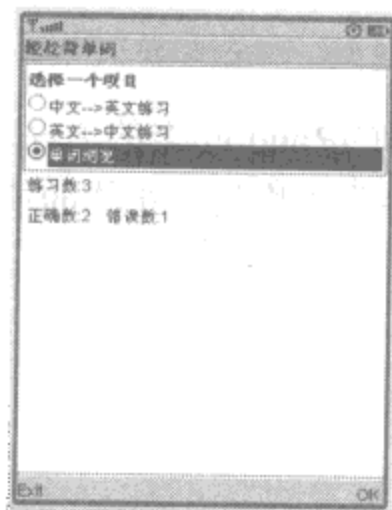


图7-21 选择第三个项目

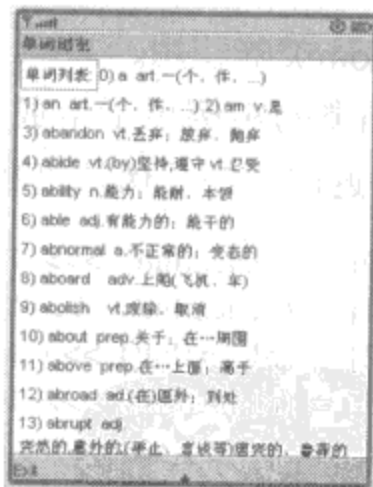


图7-22 第三个项目界面

制作要点

1. J2ME中ChoiceGroup类的用法。

J2ME中的ChoiceGroup类可以实现单选和多选，同时，它的类型又可以分为显式、隐式或者弹出式。ChoiceGroup也是一个项目类型的对象，代表一个选择列表，它的作用和List对象类似，不过后者是一个容器，而前者是一个项目。

需要特别注意ChoiceGroup类的构造函数。它有四个参数，第一个参数是标签；第二个参数是此选择列表的类型，例如多选还是单选；第三个参数是一个字符串数组，代表每个选项

的标签；第四个参数是一个Image类型的数组，代表每个选项前面的小图标。

```
mChoiceGroup = new ChoiceGroup("选择一个项目", 1, run_game, null);
```

2. StringBuffer类的使用。

StringBuffer类代表线程安全的可变字符序列，它是一个类似于String的字符串缓冲区，但不能修改。虽然在任意时间点上它都包含某种特定的字符序列，但通过某些方法调用可以改变该序列的长度和内容。

可将字符串缓冲区安全地用于多个线程，可以在必要时对这些方法进行同步，因此任意特定实例上的所有操作就好像是以串行顺序发生的，该顺序与所涉及的每个线程进行的方法调用顺序一致。

StringBuffer上的主要操作是append和insert方法，可重载这些方法，以接受任意类型的数据。每个方法都能有效地将给定的数据转换成字符串，然后将该字符串的字符追加或插入到字符串缓冲区中。append方法始终将这些字符添加到缓冲区的末端，而insert方法则在指定的点添加字符。

本案例使用了StringBuffer类的以下两种方法。

(1) public StringBuffer append(char c): 将char参数的字符串表示形式追加到此序列。参数将被追加到此序列，此序列的长度将增加1。

```
sb.append((char)b);
```

(2) public StringBuffer delete(int?start, int?end): 移除此序列的子字符串中的字符。该子字符串从指定的start处开始，一直到索引end-1处的字符，如果不存在这种字符，则一直到序列尾部。如果start等于end，则不发生任何更改。

```
sb.delete(0, sb.length()); //清空sb的内容以便读入下一个词条
```

3. Random类的使用。

Random类的实例用于生成伪随机数流。本案例使用的Random类的方法是public int nextInt(), 它返回下一个伪随机数，是此随机数生成器的序列中均匀分布的int值。

```
w[0] = (rd.nextInt() >>> 1) % vl;  
w[1] = (rd.nextInt() >>> 1) % vl;  
w[2] = (rd.nextInt() >>> 1) % vl;  
w[3] = (rd.nextInt() >>> 1) % vl;
```

4. 程序设计。

本案例是一个简单的MIDlet，由两个类组成，其中RememberWords类是本案例的核心类，封装了本案例的逻辑并对程序界面进行初始化；E_Word类是词条类，用于保存输入和输出的词条。

步骤详解

1. 新建基于Mobile的Mobile应用程序项目，创建MIDlet类。

2. 添加词条类，代码如下：

```
import java.io.*;  
public class E_Word {           //词条类，每个词条包括一个单词及其汉语解释
```



```

private String e_word;    //定义英文单词变量
private String C_Version; //定义汉语解释变量
public E_Word() {
}
public E_Word(byte rec[]) {    //带参数构造方法
    initE_Word(rec);          //调用initE_Word方法完成初始化
}
public void initE_Word(byte rec[]) {
    //定义输入流
    ByteArrayInputStream bais = new ByteArrayInputStream(rec);
    DataInputStream dis = new DataInputStream(bais);
    try {
        e_word = dis.readUTF();    //读出英文单词
        C_Version = dis.readUTF(); //读出汉语解释
    }
    catch(Exception e) { e.printStackTrace();}
}
public byte[] toBytes() {
    byte data[] = (byte[])null;
    try {
        //定义输出流
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        DataOutputStream dos = new DataOutputStream(baos);
        dos.writeUTF(e_word);    //写入英文单词
        dos.writeUTF(C_Version); //写入汉语解释
        data = baos.toByteArray(); //写入的内容转换成字节数组储存到data中
        baos.close();
        dos.close();
    }
    catch(Exception e) { e.printStackTrace();}
    return data; //返回data
}
public void set_Eng_word(String s) {    //设定英语单词
    e_word = s; }
public void set_Chinese_Version(String s) {    //设定汉语解释
    C_Version = s; }
public String get_Eng_Word() {    //获取英语单词
    return e_word; }
public String get_Chinese_Version() {    //获取汉语解释
    return C_Version; }
}

```

3. RememberWords类是程序的核心类，此类完成游戏的逻辑、绘图、控制、互动等所有功能。在源代码编辑器中添加其startApp方法代码如下：

```

protected void startApp() throws MIDletStateChangeException {
    StringBuffer sb = new StringBuffer();    //创建存放词条的StringBuffer类对象
    //创建输入流以便读入词条
    InputStream inputstream = getClass().getResourceAsStream("/ab.txt");
    try {
        InputStreamReader dis = new InputStreamReader(inputstream, "UTF-8");
        int b;
        while((b = dis.read()) != -1)    //未到文件末尾

```



```

        if((char)b != '\n') {           //未到行尾 (一个词条尚未结束)
            sb.append((char)b);         //词条内容追加到sb
        }
        else {                          //已经读入一个词条的内容
            st = split(sb.toString(), " ");           //拆分成两个字符串
            E_Word EW = new E_Word();                //创建一个词条类对象
            EW.set_Chinese_Version(st[1]);            //设置该词条的汉语解释
            EW.set_Eng_word(st[0]);                   //设置该词条的英语单词
            v_Eng.addElement(EW);                     //添加该词条到v_Eng向量
            sb.delete(0, sb.length());                //清空sb的内容以便读入下一个词条
        }
    }
    catch(Exception e) {
        System.out.println(e.toString());
    }
    run(); //调用运行方法, 显示主屏幕
}

```

4. 为RememberWords类添加运行方法。在源代码编辑器中添加代码如下:

```

protected void run() throws MIDletStateChangeException { //运行方法, 显示主屏幕
    mainform = new Form("轻松背单词"); //主屏幕标题
    mainform.addCommand(CMD_EXIT);      //为主屏幕添加“退出”按钮
    mainform.addCommand(CMD_Sele);      //为主屏幕添加“选定”按钮
    String run_game[] = new String[3];  //可运行项目数组
    run_game[0] = "中文-->英文练习";   //可运行项目1
    run_game[1] = "英文-->中文练习";   //可运行项目2
    run_game[2] = "单词浏览";           //可运行项目3
    //创建选择列表
    mChoiceGroup = new ChoiceGroup("选择一个项目", 1, run_game, null);
    mainform.append(mChoiceGroup);      //主屏幕添加选择列表
    int z = right + error;               //已练习的词条总数为正确数、错误数之和
    mainform.append("练习数:" + z + "\n"); //主屏幕添加练习数提示信息
    mainform.append("正确数:" + right);  //主屏幕添加正确数提示信息
    mainform.append(" 错误数:" + error); //主屏幕添加错误数提示信息
    mainform.setCommandListener(this);   //主屏幕添加侦听器
    display.setCurrent(mainform);        //设置显示主屏幕
}

```

5. 为RememberWords类添加英文单词选择屏幕方法和汉语解释选择屏幕方法。这两个方法是本案例的主要逻辑, 而且二者在逻辑上是完全一样的, 下面以英文单词选择屏幕方法为例, 添加如下代码:

```

protected void run_Sele_Eng() throws MIDletStateChangeException { //英文单词选择屏幕
    mainform = new Form("轻松背单词"); //英文单词选择屏幕标题
    mainform.addCommand(CMD_Ex_Game);  //为英文单词选择屏幕添加“退出”按钮
    mainform.addCommand(CMD_OK);       //为英文单词选择屏幕添加“确认”按钮
    int vl = v_Eng.size();              //获取词条总数
    mainform.append("单词总数:" + vl + "\n");
    EWA = new E_Word();
    //随机抽取4个词条
    Random rd = new Random();
    w[0] = (rd.nextInt() >>> 1) % vl;
    w[1] = (rd.nextInt() >>> 1) % vl;
}

```



```

w[2] = (rd.nextInt() >>> 1) % vl;
w[3] = (rd.nextInt() >>> 1) % vl;
for(int i = 0; i < 4; i++)
    System.out.println(w[i]);
int t = (rd.nextInt() >>> 1) % 4;    //随机选取其中一个词条作为题干
EWA = (E_Word)v_Eng.elementAt(w[t]);
A_Str = EWA.get_Chinese_Version();
E_Str = EWA.get_Eng_Word();
mainform.append("中文:" + A_Str + "\n"); //显示汉语解释题干
String stringArray[] = new String[4];
for(int i = 0; i < 4; i++) {        //初始化选择列表显示项目
    EWA = (E_Word)v_Eng.elementAt(w[i]);
    stringArray[i] = EWA.get_Eng_Word();
}
//创建选择列表
mChoiceGroup = new ChoiceGroup("选择一个项目", 1, stringArray, null);
mainform.append(mChoiceGroup);    //英文单词选择屏幕显示选择列表
mainform.setCommandListener(this); //英文单词选择屏幕添加侦听器
display.setCurrent(mainform);    //设置显示单词选择屏幕
ItemStateListener listener = new ItemStateListener() {    //创建Item状态侦听器
    public void itemStateChanged(Item item) {
        int i = mChoiceGroup.getSelectedIndex();
        E_Word EW_A = (E_Word)v_Eng.elementAt(w[i]);
        if(E_Str == EW_A.get_Eng_Word()) {    //正确选择
            showScreen("    \t\n 正确!");    //显示提示信息
            right++; //正确数加1
        }
        else {    //错误选择
            showScreen("    \t\n 错误!");    //显示提示信息
            error++; //错误数加1
        }
    }
};
mainform.setItemStateListener(listener);    //英文单词选择屏幕添加Item状态侦听器
}

```

6. 为RememberWords类添加事件处理方法代码:

```

public void commandAction(Command c, Displayable d) {    //命令按钮事件处理
    if(c == CMD_EXIT) {    //“退出程序”按钮
    if(c == CMD_Ex_Game)    //“退出项目”按钮
    if(c == CMD_BACK) {    //“返回”按钮
    if(c == CMD_Conn) //“继续”按钮
    if(c == CMD_OK) { //“确认”按钮
    if(c == CMD_Se) { //“选择”按钮

```

程序源代码与解释

```

/** RememberWords.java*/
import java.io.*;
import java.util.Random;
import java.util.Vector;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

```



```

import javax.microedition.midlet.MIDletStateChangeException;
public class RememberWords extends MIDlet implements CommandListener {
    public RememberWords() {        //构造方法，完成变量初始化
        A_Str = null;
        E_Str = null;
        right = 0;
        error = 0;
        w = new int[4];
        display = Display.getDisplay(this);    //创建设备显示画面的Display对象
        v_Eng = new Vector();
        st = new String[2];
    }
    protected void startApp() throws MIDletStateChangeException {
        //参见步骤详解3
    }
    protected void run() throws MIDletStateChangeException {    //运行方法，显示主屏幕
        //参见步骤详解4
    }
    protected void run_brow() {        //单词浏览屏幕
        mainform = new Form("单词浏览\n");    //单词浏览屏幕标题
        mainform.addCommand(CMD_Ex_Game);    //为单词浏览屏幕添加“退出”按钮
        mainform.append("单词列表:");    //为单词浏览屏幕添加提示信息
        int vl = v_Eng.size();    //获取词条总数
        EWA = new E_Word();
        //循环显示所有词条内容
        for(int i = 0; i < vl; i++) {
            EWA = (E_Word)v_Eng.elementAt(i);    //获取向量中的元素
            //显示词条内容
            mainform.append(i + ") " + EWA.get_Eng_Word() + " " + EWA.get_Chinese_ Ver-
sion());
        }
        mainform.setCommandListener(this);    //单词浏览屏幕添加侦听器
        display.setCurrent(mainform);    //设置显示单词浏览屏幕
    }
    protected void run_Sele_Chi() throws MIDletStateChangeException {    //汉语解释选择屏幕
        //类似于步骤详解5的“英文解释选择屏幕”
    }
    protected void pauseApp() { }
    protected void destroyApp(boolean flag) throws MIDletStateChangeException { }
    public String[] split(String original, String regex) {    //分割字符串
        int startIndex = 0;
        String str[] = new String[2];
        startIndex = original.indexOf(regex);
        str[0] = original.substring(0, startIndex);
        str[1] = original.substring(startIndex + 1, original.length() - 1);
        return str;
    }
    public void showScreen(String cmd) {    //答案检查屏幕
        Form form = new Form("答案检查:");
        form.append(cmd);
        form.addCommand(CMD_BACK);
        form.addCommand(CMD_Conn);
        form.setCommandListener(this);
        display.setCurrent(form);
    }
}

```




案例4：用J2ME与ASP建立数据库连接

案例运行效果与操作

J2ME是利用HttpConnection建立HTTP连接，然后获取数据的。ASP也是利用HTTP协议，因而可以利用J2ME与ASP建立连接，从而访问数据库。本案例给出一个简单的例子，说明MIDP如何与ASP进行交互，完成数据库操作。程序运行后，界面如图7-23所示。

按“Launch”软键运行程序，此时首先出现联网确认界面，如图7-24所示，提示用户上网需要费用，要求用户确认上网。按“YES”软键确认上网，则程序会与IIS服务器进行连接，通过ASP向数据库写入数据。此时打开数据库查看表Message，会发现一条记录已经成功写入，如图7-25所示。

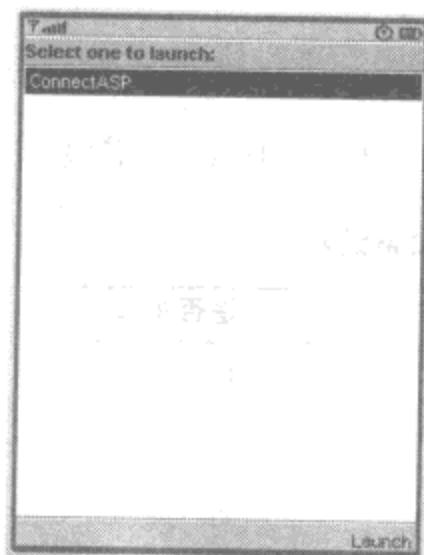


图7-23 初始界面

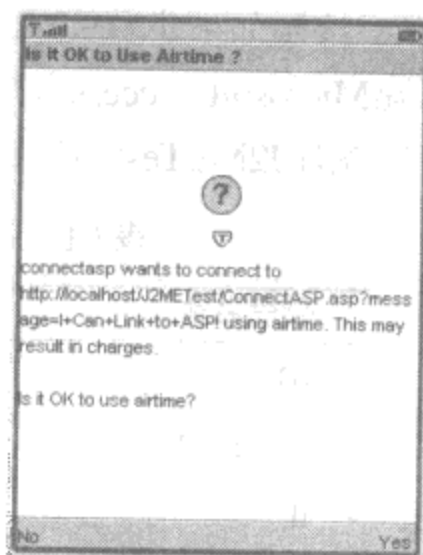


图7-24 运行后确认界面

Message : 表				
	ID	Message	IP	Date
	1	Hello J2ME!	127.0.0.1	2007-9-10
	2	Hello J2ME!	127.0.0.1	2007-9-10
	3	I Can Link to ASP!	127.0.0.1	2007-9-10
*	(自动编号)			

图7-25 数据表中添加了一条记录

制作要点

HTTP连接是MIDP的一个需求。一个MIDP可用的设备必须能够跟一个Web服务器通过HTTP请求进行交互。如果网络不直接支持HTTP，设备必须通过网关路由这个请求。但是这些对于应用开发人员都是透明的。

定义在javax.microedition.io类中的抽象网络 and 文件输入/输出框架称为通用连接框架（Generic Connection Framework，简称GCF）。这个框架是一组由CLDC定义的类和接口，它替换了有J2SE定义的java.io和java.net中的大部分类。GCF定义了一套有关抽象化的内容来描述不同的通信方法。最高级的抽象被称做连接（Connection），还声明了六个接口（四个是直接的，两个是间接的）。这七个接口就构成了J2ME的CLDC的一部分，CLDC是大多数能使用Java的无线设备使用的配置。设计这个配置的目的就是为所有的CLDC设备（手提电话、双向传呼机、低档的PDA等）提供公用的网络和文件输入/输出能力。虽然GCF的目的是公用网络和文件输

入/输出框架，但是生产商并不要求实现GCF中声明的所有的接口。有的厂家可以决定只支持socket连接，而其他的厂家可以选择只支持基于数据报的通信。为了促进跨越类似装置的可移植性，MIDP规范要求所有的MIDP设备实现HttpConnection接口。HttpConnection不是GCF的一部分，但是它是从GCF的一个接口ContentConnection衍生出来的。

建立一个HTTP请求，使用Connector.open方法和相应的URL，投放结果到一个HttpConnection。HttpConnection接口定义了所有的方法，这些方法是建立HTTP请求和处理回复过程中所需要的。它包括setRequestMethod、getHeaderField和openInputStream。HttpConnection接口使与互联网上的任何Web站点进行交互的工作变得简单：

```
HttpConnection conn=(HttpConnection)Connector.open(uri);
```

步骤详解

- 1. 数据库准备工作包括创建数据库和表、编写ASP代码等。首先要确保IIS Web服务器正常工作，然后按照如下步骤完成数据库与表的创建。
 - (1) 新建Microsoft Access数据库J2METest。
 - (2) 在数据库J2METest中创建Message表，并按照表7-1向表中添加字段。

表7-1 添加字段的名称及相应的设置选项

字段名称	数据类型	是否主键
ID	自动编号	是
MESSAGE	文本	否
IP	文本	否
DATE	日期	否

- 2. 创建ASP代码，用记事本等编辑工具编写一个ConnectASP.asp文件，添加下列代码：

```
<% @LANGUAGE="VBScript" %>
<%
strDBLocation = Server.MapPath("/Database/J2METest.mdb")
strConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=" & strDBLocation
Set cnn= Server.CreateObject("ADODB.Connection")
cnn.Open(strConnectionString)
Set rs=Server.CreateObject("ADODB.Recordset")
ip=Request.ServerVariables("REMOTE_ADDR")
message=Request.QueryString("message")
sql = "Select * from Message"
rs.Open sql,cnn,1,2
rs.addNew
rs("Message")=message
rs("IP")=ip
rs("Date")=Date
rs.update()
rs.close()
cnn.close()
Response.Write("The message writed successfully.")
%>
```


在IIS服务器中创建一个虚拟目录，名称设置为“J2METest”，目录指向“C:\javabook\chapter7\connectasp\IIS\ASP”。此时在浏览器中输入以下URL “http://localhost/J2METest/ConnectASP.asp message=Hello+J2ME!”，如果运行成功，浏览器会输出语句“The message writed successfully.”，如图7-26所示。然后打开数据库文件查看表Message，会发现一条记录已经成功写入，其ID为1，Message为“Hello J2ME !”，IP为“127.0.0.1”，Date为当前日期。

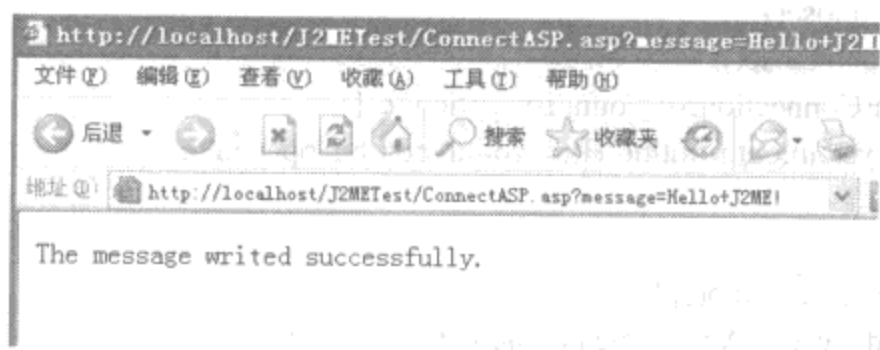


图7-26 在本地机器上运行ASP的结果

3. 准备工作完成后，新建项目Mobile应用项目connectasp，创建MIDlet类ConnectASP。
4. 建立一个HTTP连接，其URI指向测试用ASP文件，并附带message参数：

```
String uri="http://localhost/J2METest/ConnectASP.asp?message=I+Can+Link+to+ASP!";
HttpConnection conn=(HttpConnection)Connector.open(uri);
```

5. 创建输入流：

```
InputStream in=conn.openInputStream();
int ch=in.read();
while(ch!=-1){
    System.out.print((char)ch);
}
```

程序源代码与解释

```
/* * ConnectASP.java*/
import javax.microedition.io.ConnectionNotFoundException;
import javax.microedition.io.Connector;
import javax.microedition.io.HttpConnection;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.io.*;

public class ConnectASP extends MIDlet{
    public void startApp() {
        try{
            testASP();          //调用testASP方法测试ASP连接
        }catch(IOException e){
            System.out.println("error");
        }
        notifyDestroyed();//运行完毕释放资源
    }
    void testASP()throws IOException{//测试ASP连接
        try{
            //建立一个HTTP连接，其URI指向测试用ASP文件，并附带message参数
            String uri="http://localhost/J2METest/ConnectASP.asp?message=I+Can+Link+
to+ASP!";
```



```
HttpConnection conn=(HttpConnection)Connector.open(uri);
//创建输入流
InputStream in=conn.openInputStream();
int ch=in.read();
while(ch!=-1){
    System.out.print((char)ch);
}
in.close();
conn.close();
}catch(ConnectionNotFoundException e){
    System.out.println("Http could not be opened");
}
}
public void pauseApp(){}
public void destroyApp(boolean unconditional) {}
}
```



案例5： 利用J2ME开发联网程序实例

案例运行效果与操作

本案例是一个简单的MIDlet，用Socket简单实现了基于TCP/IP协议的J2ME联网功能。程序运行后，界面如图7-27和图7-28所示。



图7-27 运行界面



图7-28 第二个项目界面

按“Launch”软键运行程序，此时首先出现SocketConnection示例主屏幕，它显示一个选择列表，让用户选择角色（服务器或者客户端），如图7-29所示。

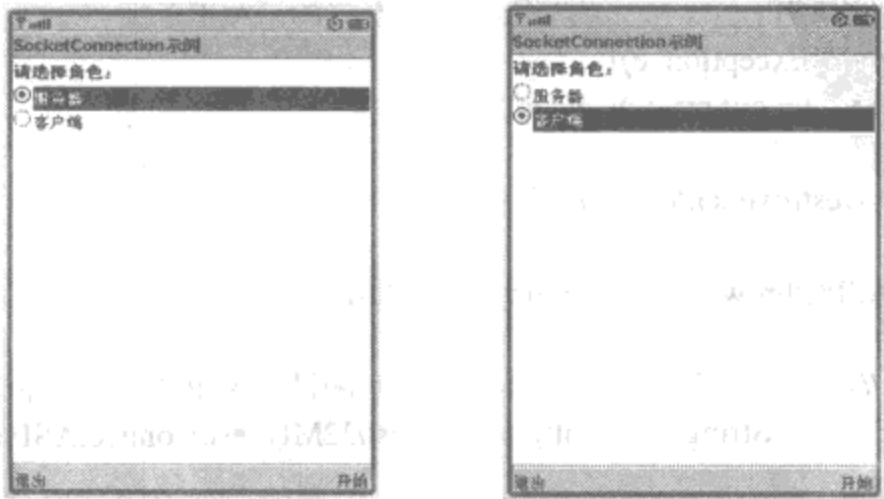


图7-29 选择角色界面

首先选择“服务器”角色，按下“开始”按钮，出现联网确认界面，如图7-30所示。按“YES”软键确认上网，会出现Socket服务器屏幕，如图7-31所示。

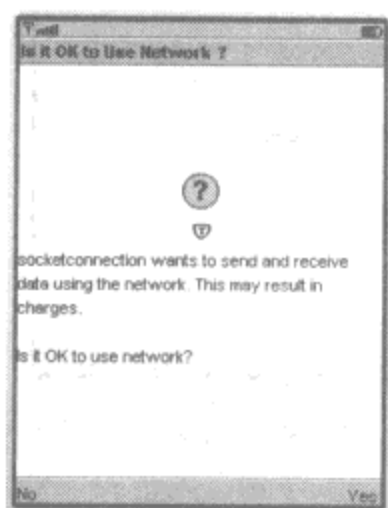


图7-30 联网确认界面



图7-31 服务器等待连接界面

再运行一个窗口，选择“客户端”角色，按下“开始”按钮，出现联网确认界面，如图7-30所示，按“YES”软键确认上网，会出现Socket客户端屏幕，如图7-31所示。此时状态变为“已建立连接”。服务器连接成功双方就可以进行通信了。

在Socket客户端屏幕的“发送”文本框中输入要发送给服务器的信息，例如“Hello! I am a client.”，按“发送”键，会在Socket服务器屏幕上方的状态信息栏中显示：“接收到信息：Hello! I am a client.”，如图7-32所示。同样，在Socket服务器屏幕的“发送”文本框中输入要发送给客户端的信息，例如“Hi! I am server.”，按“发送”键，会在Socket客户端屏幕上方的状态信息栏中显示：“接收到信息：Hi! I am server.”，如图7-33所示。



图7-32 Socket服务器接收和发送信息



图7-33 Socket客户端接收和发送信息

如果在服务器启动后再次启动一个服务器，会出现服务器错误界面，如图7-34所示。如果在服务器尚未启动时就启动了客户端，会出现客户端错误界面，如图7-35所示。

制作要点

1. J2ME中Connection的层次结构。

通用连接框架（Generic Connection Framework, GCF）是在J2ME平台中广泛使用的用于联网和I/O处理的类。在使用GCF的时候，首先应该根据项目需求选择适当的Connection。

上一个案例简单介绍了HttpConnection的应用，本案例简单介绍ServerSocketConnection和SocketConnection的应用。它们都属于MIDP 2.0的内容。由于它们都属于GCF，因此在程序编写方面也非常相似。



图7-34 Socket服务器错误界面

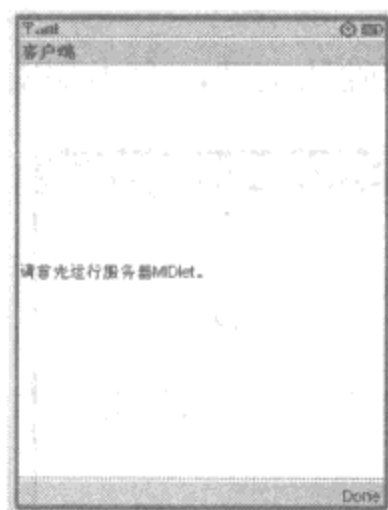


图7-35 Socket客户端错误界面

2. ServerSocketConnection类和SocketConnection类的使用。

通常在服务器端的某个端口监听，当客户端连接服务器的时候，可以得到一个Socket-Connection的实例。通过两端的SocketConnection则可以实现C/S结构的通信。ServerSocket-Connection类的一个非常重要的方法就是acceptAndOpen()，这个方法返回一个Socket-Connection实例，从而使得客户端和服务端可以通过Socket进行通信。

```
sc = (SocketConnection) scn.acceptAndOpen();
```

SocketConnection的使用也非常简单，通过Connector的open方法可以得到一个SocketConnection的实例。

```
sc = (SocketConnection) Connector.open("socket://localhost:5000");
```

在用Socket编写程序时只需遵循这样的一种规则：服务器端建立监听端口，等待连接，客户端通过open()方法与服务器端建立连接，两端通过建立的Socket传输数据，关闭连接。

3. 程序设计。

本案例是一个简单的MIDlet，共由四个类组成，其中继承了Midlet的SocketConnection-Midlet类用于初始化界面以及程序控制，Sender类实现具体传送信息的工作，而两个对等类Server和Client则分别模拟服务器和客户端的操作，它们都调用Sender类来发送和接收信息。

步骤详解

1. 创建基于Mobile应用程序的项目socketconnection，新建MIDlet类SocketConnectionMidlet。
2. 建立SocketConnectionMidlet.java文件，主要实现界面的初始化和程序的控制，如程序的启动、暂停、终止及其判断，以下是部分代码：

```
public class SocketConnectionMidlet extends MIDlet implements CommandListener {
    ...

    public SocketConnectionMidlet() {          //构造方法，完成初始化
        display = Display.getDisplay(this);
        f = new Form("SocketConnection示例");
        cg = new ChoiceGroup("请选择角色: ",Choice.EXCLUSIVE, names, null);
        f.append(cg);
        f.addCommand(exitCommand);
        f.addCommand(startCommand);
        f.setCommandListener(this);
        display.setCurrent(f);
    }
}
```



```

    }
    public void commandAction(Command c, Displayable s) { //命令按钮事件处理
        if (c == exitCommand) { //“退出”按钮
            destroyApp(true);
            notifyDestroyed();
        } else if (c == startCommand) { //“开始”按钮
            String name = cg.getString(cg.getSelectedIndex()); //获取用户角色
            if (name.equals(SERVER)) { //选择了服务器角色
                server = new Server(this); //创建Server对象并启动
                server.start();
            } else { //选择了客户端角色
                client = new Client(this); //创建Client对象并启动
                client.start();
            }
        }
    }
}

```

3. 考虑到网络带宽因素，为了避免操作堵塞，必须把联网动作放到另外一个线程去运行，而不能在主线程运行。因此让**Server**、**Client**、**Sender**三个类实现多线程，其中**Server**类和**Client**类二者的逻辑完全一致，它们使用实现**Runnable**接口的方法，而**Sender**类则使用继承**Thread**类的方法。另外，联网操作逻辑按照如下的流程进行：

- (1) 建立连接；
- (2) 打开输出流，传输数据给服务器；
- (3) 判断相应的状态码，进入不同流程控制，注意错误处理；
- (4) 关闭连接和流。

基于上述原因，为**Server**类添加**run**方法和**stop**方法。

线程运行方法：

```

    public void run() {
        try {
            si.setText("等待连接...");
            //创建ServerSocketConnection对象
            ServerSocketConnection scn = (ServerSocketConnection)
            Connector.open("socket://:5000");
            //等待连接
            sc = (SocketConnection) scn.acceptAndOpen();
            si.setText("已建立连接。");
            //创建输入/输出流
            is = sc.openInputStream();
            os = sc.openOutputStream();
            //启动Sender线程传输数据
            sender = new Sender(os);
            //只有在Sender对象创建后才能发送信息
            f.addCommand(sendCommand);
            f.setCommandListener(this);
            //接收数据的处理，始终处在循环中，代码略
        } catch (Exception e) {
            //线程停止方法
        }
        public void stop() {
            try {
                stop = true;
            }
        }
    }

```



```

        sender.stop();
        if (is != null) {
            is.close();
        }
        if (os != null) {
            os.close();
        }
        if (sc != null) {
            sc.close();
        }
    } catch (IOException ioe) {}
}

```

4. 为Sender类添加运行方法:

```

public synchronized void run() { //线程运行方法
    while(true) { //始终在发送数据的死循环
        //无客户端连接时则一直等待
        if (message == null) {
            try { wait();
            } catch (InterruptedException e) {}
        }
        if (message == null) {
            break; }
        try {
            os.write(message.getBytes());
            os.write("\r\n".getBytes());
        } catch (IOException ioe) { ioe.printStackTrace();}
        //客户端连接处理完成后重置
        message = null;
    }
}

```

程序源代码与解释

```

/* * Server.java */
public class Server implements Runnable, CommandListener {
    public Server(SocketConnectionMidlet m) { //带参数构造方法，初始化界面
        //显示服务器端屏幕
        parent = m;
        display = Display.getDisplay(parent);
        f = new Form("Socket服务器");
        si = new StringItem("状态: ", " ");
        tf = new TextField("发送: ", "", 30, TextField.ANY);
        f.append(si);
        f.append(tf);
        display.setCurrent(f);
    }
    //启动客户端线程
    public void start() { //线程启动方法
        Thread t = new Thread(this); //创建线程
        t.start();
    }
    public void commandAction(Command c, Displayable s) { //命令按钮事件处理

```



```

        if (c == sendCommand && !parent.isPaused()) {
            sender.send(tf.getString());
        }
        if (c == Alert.DISMISS_COMMAND) {
            parent.notifyDestroyed();
            parent.destroyApp(true);
        }
    }
}

/* * Client.java */
public class Client implements Runnable, CommandListener {
    public Client(SocketConnectionMidlet m) {          //带参数构造方法，初始化界面
        //显示客户端屏幕
        parent = m;
        display = Display.getDisplay(parent);
        f = new Form("Socket客户端");
        si = new StringItem("状态: ", " ");
        tf = new TextField("发送: ", "", 30, TextField.ANY);
        f.append(si);
        f.append(tf);
        f.addCommand(sendCommand);
        f.setCommandListener(this);
        display.setCurrent(f);
    }
    /*** 启动客户端线程 */
    public void start() {          //线程启动方法
        Thread t = new Thread(this);          //创建线程
        t.start();
    }
    public void run() {          //线程运行方法
        try {
            //类似于步骤详解3
        }
    }
    public void commandAction(Command c, Displayable s) {          //命令按钮事件处理
        if (c == sendCommand && !parent.isPaused()) {
            sender.send(tf.getString());
        }
        if (c == Alert.DISMISS_COMMAND) {
            parent.notifyDestroyed();
            parent.destroyApp(true);
        }
    }
}

/*** 关闭所有连接和流 */
public void stop() {          //线程停止方法
    //参见步骤详解3
}

}

/* * Sender.java */
public class Sender extends Thread {
    private OutputStream os;
    private String message;
    public Sender(OutputStream os) {          //带参数构造方法
        this.os = os;
        start();          //启动Sender线程
    }
}

```



```
    }  
    public synchronized void send(String msg) {    //发送信息  
        message = msg;  
        notify();  
    }  
    public synchronized void stop() {            //线程停止方法  
        message = null;  
        notify();  
    }  
}
```

本章小结

移动开发已经成为开发者社区最为引人注目的新技术。移动互联网蕴藏的巨大商机以及嵌入式开发的神秘色彩使得越来越多的程序员开始学习和研究移动开发技术。J2ME、Symbian、Windows Mobile、BREW是目前主流的技术平台，其中J2ME凭借其开放的特性占据了绝对的市场，成为了移动开发领域的标准。相比其他平台，J2ME的参考资料也更丰富，更适合初次涉足此领域的开发者入门。Java技术最早的时候就是为了嵌入式系统而设计的一项产品，虽然Java已经被用到许多企业级软件上，其实还是非常适合用在嵌入式系统之中。本章通过几个不同难易程度的实用案例程序，覆盖了J2ME在游戏、应用程序、网络等多个方面的应用，有助于读者加深对J2ME诸多特性的理解。

在本章的最后简单讲述一下目前主流的用于开发J2ME应用程序的开发工具，以供读者参考。开发工具主要有SUN Wireless Toolkit 2.2、Netbeans IDE、Eclipse和JBuilder。通常，开发应用程序的时候都会针对具体的目标平台，比如Nokia Series60系列。如果可以把第三方的SDK集成到我们熟悉的开发工具中，必将方便调试和测试，缩短软件开发周期。当然，开发工具固然重要，它可以提高软件开发效率，缩短周期，甚至可以方便地进行单元测试和团队协作。但是，掌握MIDP应用程序模型，熟悉开发流程比单纯地掌握开发工具更加重要。



第8章 J2EE技术

本章内容

- 案例1：一个用Servlet实现购物车的程序
- 案例2：连接数据库的JavaBean
- 案例3：测试安全性代码
- 案例4：用EJB实现的用户消费信息登记系统
- 案例5：Fibonacci数列
- 案例6：简单的图书信息管理系统
- 本章小结



案例1：一个用Servlet实现购物车的程序

案例运行效果与操作

本案例利用Servlet技术，简单模拟了一个网上书店购物车程序。这是一个Web应用程序，需要Web服务器进行服务，本案例使用了Tomcat 5.5.25作为Web服务器。

程序运行后，会打开一个IE浏览器窗口，如图8-1所示。

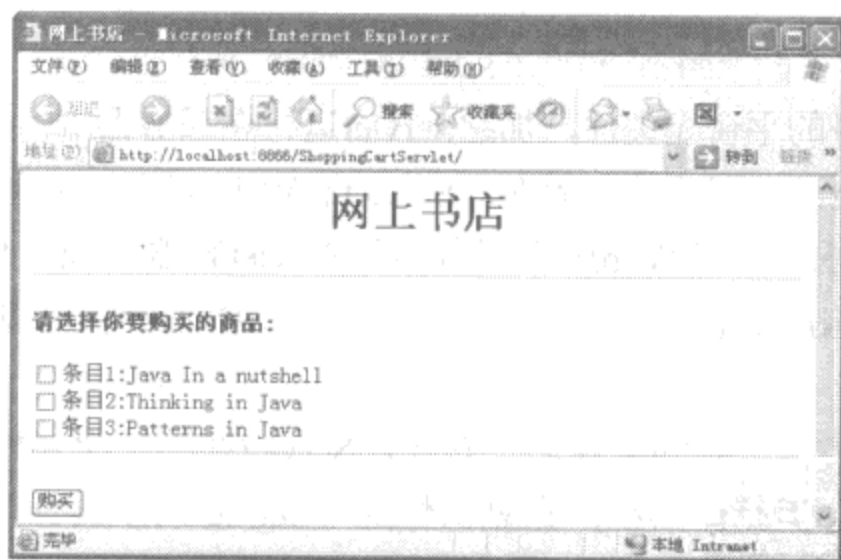


图8-1 运行界面

此时可以在这个页面选择商品，只要选中相应商品左方的复选框即可，如图8-2所示选中了两本书。

单击“购买”按钮，Servlet会生成相应的购物单界面，如图8-3所示。

制作要点

1. Servlet技术及应用。

Servlet是Java平台上的CGI技术。Servlet在服务器端运行，动态地生成Web页面。与传统的CGI和许多其他类似CGI的技术相比，Java Servlet具有更高的效率并更容易使用。对于Servlet，

重复的请求不会导致同一程序的多次转载，它是依靠线程的方式来支持并发访问的。

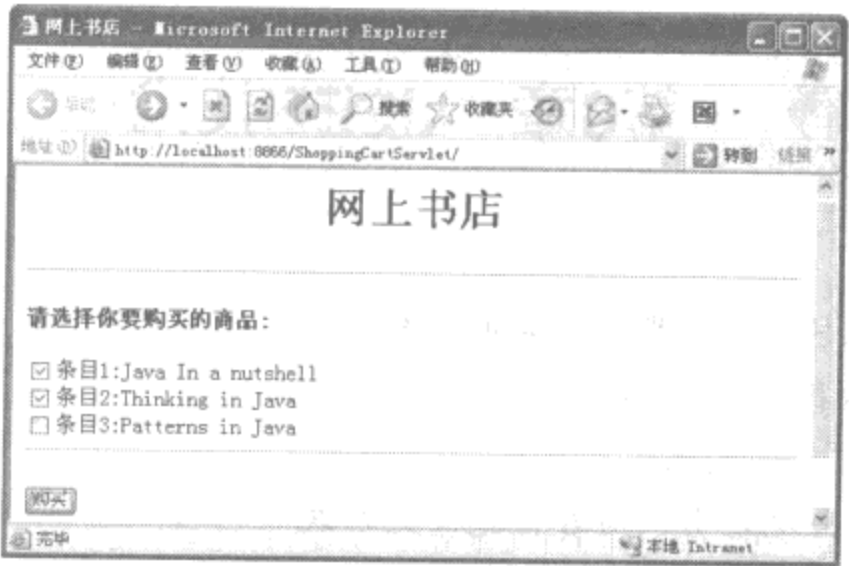


图8-2 选择商品界面

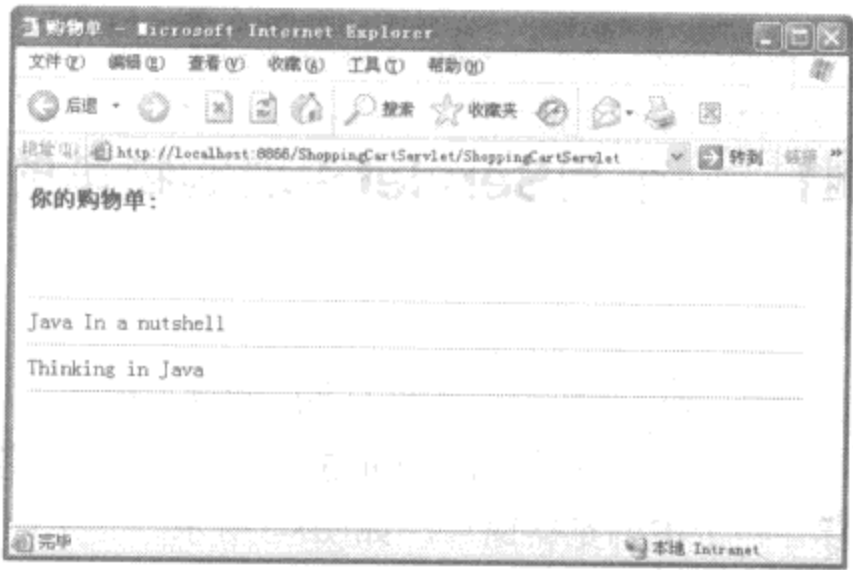


图8-3 生成购物单界面

由于Java语言本身的平台无关性，加之Servlet运行在服务器端，所以Servlet的运行对用户是完全透明的。Servlet没有用户图形界面，Servlet和Web应用服务器的Servlet容器交互以接收请求、返回响应。请求最先由Web应用服务器的Servlet容器处理并传给Servlet，Servlet通过Web应用服务器返回响应给客户端。客户端程序可以使用任何可向Web应用服务器发送请求的语言开发。

Servlet最大的优势在于它的高性能。首先，Servlet在第一次初始化时装载并驻留在内存中，以后直接从内存中运行；其次，在默认情况下Servlet以单实例多线程的方式工作，一个新请求到达后，Servlet实例开启一个新的线程服务这个请求。

所有的Servlet都直接或间接地实现javax.servlet.Servlet接口，这些接口规定了Servlet如何与Servlet容器进行通信的方法，此外还定义了Servlet的生命周期。GenericServlet是和协议无关的通用Servlet，HttpServlet是专门针对HTTP协议开发的Servlet，Web应用程序的Servlet都直接继承HttpServlet。其类的继承体系如图8-4所示。

javax.servlet.Servlet接口包括了3个控制Servlet生命周期的方法，它们分别如下。

(1) init(ServletConfig config)方法。

当Servlet初始化时，init()方法被调用执行初始化Servlet的工作，init()方法只被调用一次。Servlet初始化后就进入就绪状态，随时准备响应客户端的请求。



图8-4 Servlet的类继承体系

(2) `service(ServletRequest req, ServletResponse resp)`方法。

Servlet容器调用`service()`方法处理请求并返回响应。`ServletRequest`和`ServletResponse`作为参数传给`service()`，`ServletRequest`封装了请求的信息而`ServletResponse`封装了响应的信息。

(3) `destroy()`方法。

Servlet容器可以在任何时候卸载Servlet，此时`destroy()`被调用，可以在这里释放Servlet所占用的资源。

而`javax.servlet.http`包中的类用于支持HTTP协议、创建HTML网页。HTTP协议基于请求/响应工作模式，这些HTTP的请求方式包括：

- GET;
- POST;
- PUT;
- DELETE;
- HEAD;
- TRACE;
- CONNECT;
- OPTIONS.

`javax.servlet.http.HttpServlet`定义了多个服务HTTP协议的方法，这些方法名以`doXxx()`的样式命名，和HTTP请求方式名相呼应，例如HTTP GET请求方式对应`doGet()`，而HTTP POST对应`doPost()`等。`HttpServlet`最初以`service(HttpServletRequest req, HttpServletResponse resp)`响应客户端请求，并依据HTTP的请求方式调用相应的`doXxx()`方法来处理。

一般的，仅需要覆盖`doGet()`或`doPost()`方法，如果希望得到更多的控制，也可以覆盖`doPut()`和`doDelete()`方法，其他的方法一般很少使用。

特别需要指出的是Servlet是以多线程的方式工作的，Servlet可以同时处理多个请求。作为开发人员，需要注意Servlet成员变量的线程安全，在`doGet()`、`doPost()`中的局部域变量是线程安全的，而Servlet的成员变量则有线程安全的隐患。所以除非有意需要应用这种特性，在一般情况下，不宜将一些可改写的变量定义成Servlet的成员变量，否则一定要采取线程同步的措施确保线程安全。

2. HttpSession接口的用法。

HttpSession接口提供了一种方式来标识跨越多个页面请求或访问网站的用户，存储用户的相关信息。容器利用该接口创建位于HTTP客户端和HTTP服务器之间的会话。会话将保持指定的时间段，跨越多个连接或者来自用户的页面请求。会话通常相当于一个多次访问某个

网址的用户。服务器可以使用多种方式来维护会话，比如使用Cookies或重写URL。

其主要方法如下：

- (1) Object `getAttribute(String name)`: 返回session中指定名称的绑定对象，如果没有对象被绑定，返回null。
- (2) Enumeration `getAttributeNames()`: 以String对象的枚举形式返回绑定给该session的所有对象名称。
- (3) ServletContext `getServletContext()`: 返回session属于的ServletContext。
- (4) void `removeAttribute(String name)`: 从session中删除指定名称绑定的对象。
- (5) void `setAttribute(String name, Object value)`: 使用指定名称，向session绑定一个对象。

步骤详解

1. 配置Web应用服务器，本例使用的是Tomcat 5.5.25，端口号为8866，在NetBeans开发环境中建立Web应用程序，如图8-5所示。

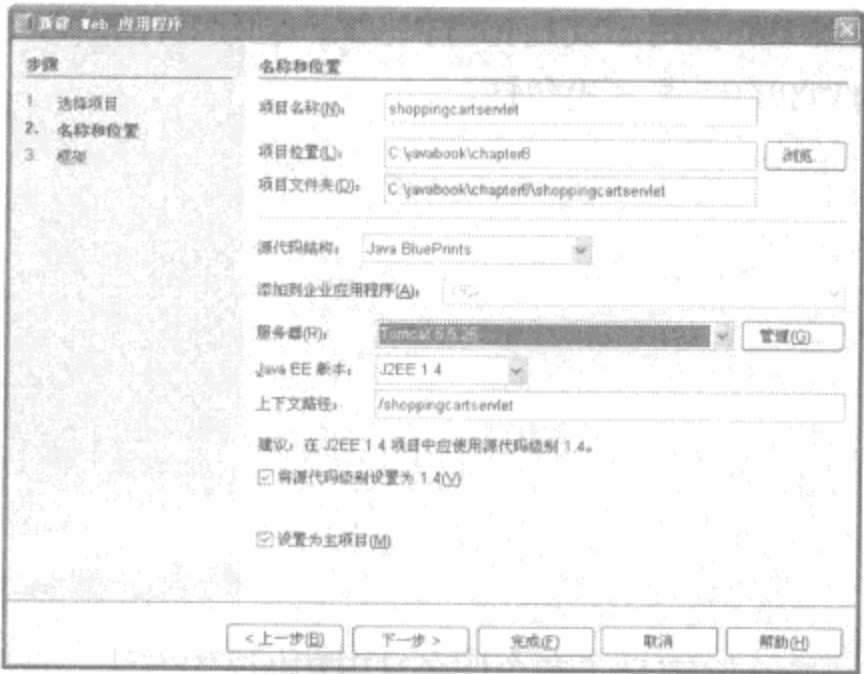


图8-5 “新建Web应用程序”对话框

配置Serverlet部署，可以设置Servlet的名称以及使用的URL映射，并设置初始化Servlet时使用的参数，其界面如图8-6所示。

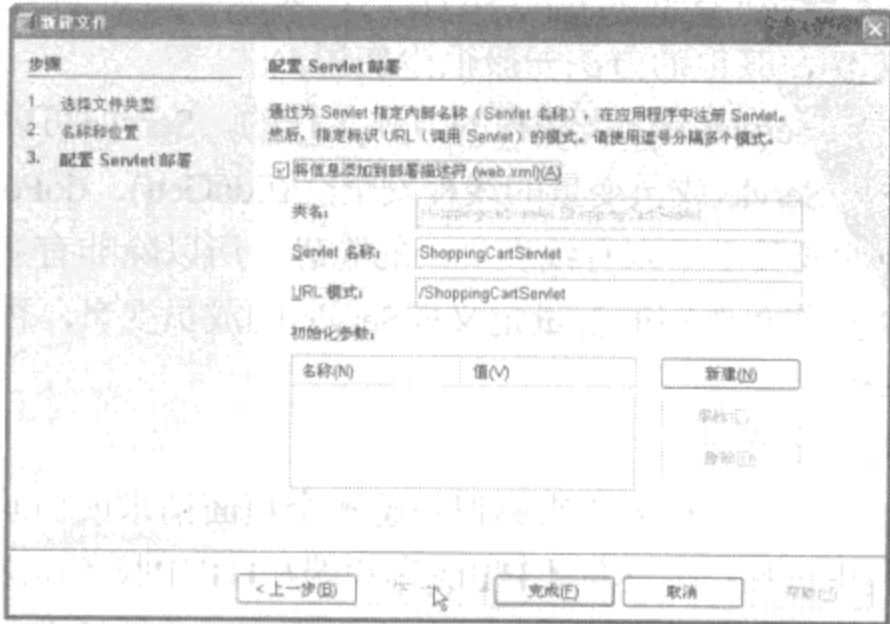


图8-6 “新建文件”对话框

2. Http请求头和应答头。

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
```

HTTP请求头概述 (HttpServletRequest)： HTTP客户程序（例如浏览器）向服务器发送请求的时候必须指明请求类型（一般是GET或者POST）。

HTTP应答头概述 (HttpServletResponse)： Web服务器的HTTP应答一般由以下几项构成：一个状态行，一个或多个应答头，一个空行，内容文档。设置HTTP应答头往往和设置状态行中的状态代码结合起来。

3. 取得session对象并接收传来的参数：

```
HttpSession session =request.getSession(true);
Integer itemCount=(Integer)session.getAttribute("itemCount");
itemsSelected=request.getParameterValues("item");
```

4. 生成网页文件。

```
out.println("<html>");
out.println("<title>");
out.println("购物单");
out.println("</title>");
out.println("<body><h4>你的购物单:</h4><br><br><hr>");
for(int i=1;i<=itemsSelected.length;i++){
    out.println((String)session.getAttribute("item"+i)+"<hr>");
}
out.println("</body>");
out.println("</html>");
```

5. 建立JSP文件shoppingcart.jsp:

```
<%@page contentType="text/html"%>
<%@page pageEncoding="gb2312"%>
<html>
<head>
<title>网上书店</title>
</head>
<body bgcolor="#FFFFFF">
<h1 align="center">网上书店</h1>
<hr>
<p><b>请选择你要购买的商品:</b></p>
<form action="ShoppingCartServlet" method="post">
<table width="500" border="0" cellspacing="0" cellpadding="0">
<tr>
<td width="20">
<input type="checkbox" name="item" value="Java In a nutshell">
</td>
<td width="431">条目1:Java In a nutshell</td>
</tr>
<tr>
<td width="20">
<input type="checkbox" name="item" value="Thinking in Java">
</td>
<td width="431">条目2:Thinking in Java</td>
</tr>
```



```
</tr>
<tr>
  <td width="20">
    <input type="checkbox" name="item" value="Patterns in Java ">
  </td>
  <td width="431">条目3:Patterns in Java </td>
</tr>
</table>
<hr>
<p>
  <input type="submit" name="btn_submit" value="购买"/>
</p>
</form>
<p>&nbsp;</p>
</body>
</html>
```

6. 修改web.xml文件，设置欢迎文件为“shoppingcart.jsp”，如图8-7所示。



图8-7 设置欢迎文件

```
- <welcome-file-list>
  <welcome-file>shoppingcart.jsp</welcome-file>
</welcome-file-list>
```

程序源代码与解释

```
/* * ShoppingCartServlet.java*/
package shoppingcartervlet;
public class ShoppingCartServlet extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=gb2312");
        //取得session对象
        //如果session不存在，为本次会话创建此对象
        HttpSession session =request.getSession(true);
        Integer itemCount=(Integer)session.getAttribute("itemCount");
        //如果session是新的
        if (itemCount==null)
            itemCount=new Integer(0);
        PrintWriter out=response.getWriter();
        //接收传来的参数
        String[] itemsSelected;
        String itemName;
        itemsSelected=request.getParameterValues("item");
        if(itemsSelected !=null){
            for(int i=0;i<itemsSelected.length;i++){
```



```

        itemName=itemsSelected[i];
        System.out.println(itemName);
        itemCount=new Integer(itemCount.intValue()+1);
        //购买的条目
        session.setAttribute("item"+itemCount,itemName);
        //总条目
        session.setAttribute("itemCount",itemCount);
    }
}
//生成网页文件
}
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    processRequest(request, response);
}
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    processRequest(request, response);
}
public String getServletInfo() {
    return "Short description";
}
}

```



案例2：连接数据库的JavaBean

案例运行效果与操作

本案例使用JavaBean封装数据库连接，这样JavaBean组件执行后台的数据库操作并获得执行结果，再通过JSP显示这些结果。程序运行后，首先进入填写用户信息界面，如图8-8所示。

在用户信息界面依次填写一些数据，如图8-9所示。

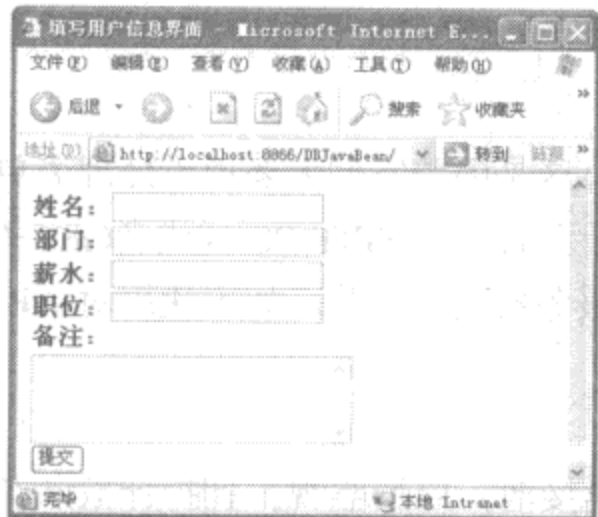


图8-8 初始界面

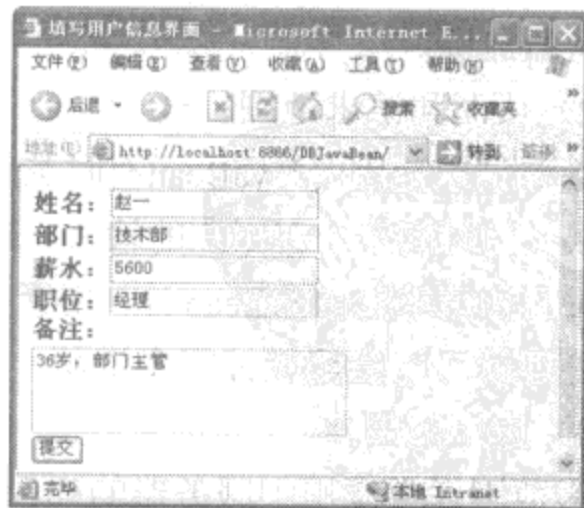


图8-9 填写用户信息

单击“提交”按钮，JavaBean会与数据库进行连接，将相应信息添加到数据库表中，同时出现一个JSP页面显示执行结果，如图8-10所示。

查看数据库，会发现有一条相应的记录被添加到userinfo表中。

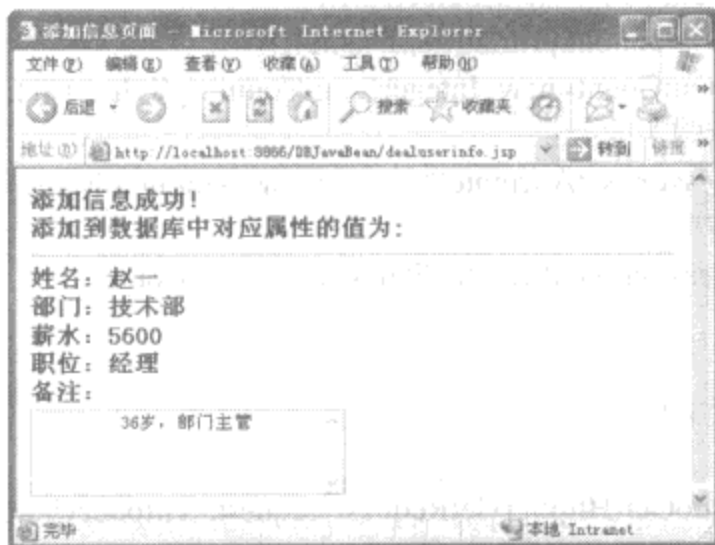


图8-10 添加信息界面

制作要点

1. JavaBean的应用。

JavaBean是Sun微系统的一个面向对象的编程接口，是Java语言写成的一种可重用组件。像Java applet一样，JavaBean组件（或“Beans”）能够给予万维网页面交互的能力。JavaBean中定义了一系列属性，并通过提供符合一致性设计模式的公共方法来访问和设置这些属性。其他Java类可以通过自省机制发现和操作这些JavaBean属性。

用户可以使用JavaBean将功能、处理、值、数据库访问和其他任何可以用Java代码创造的对象进行打包，并且其他的开发者可以通过内部的JSP页面、Servlet、其他JavaBean、Applet程序或者应用来使用这些对象。可以认为JavaBean提供了一种随时随地的复制和粘贴的功能，而不用关心任何改变。

JavaBean存在下面四种范围：页面、请求、对话、应用。

（1）对话范围。

对话范围的JavaBean主要应用于跨多个页面和时间段，例如填充用户信息。添加信息并且接受回馈，保存用户最近执行页面的轨迹。对话范围JavaBean保留一些和用户对话ID相关的信息，这些信息来自临时的对话Cookie，并在当用户关闭浏览器时，这个Cookie将从客户端和服务端删除。

（2）页面和请求范围。

页面和请求范围的JavaBean有时类似表单的Bean，这是因为它们大多用于处理表单。表单需要很长的时间来处理用户的输入，通常情况下用于页面接受HTTP/POST或者GET请求。另外，页面和请求范围的Bean可以用于减少大型站点服务器上的负载，如果使用对话Bean，耽搁的处理就可能会消耗掉很多资源。

（3）应用范围。

应用范围通常应用于服务器的部件，例如JDBC连接池、应用监视、拥护计数和其他参与用户行为的类。

2. JavaBean的编写方法：JavaBean是一个Java类，但这个类必须是具体的和公共的，并且具有无参数的构造器。

3. 程序设计。

数据库：本案例是一个在JavaBean中连接数据库的Web项目，需要在后台数据库中创建

相应的表。本案例使用MySQL数据库作为后台数据库，在其中建立一个userinfo数据库，在这个数据库中再建立一个userinfo表，JavaBean连接数据库后，向这个表中读写数据。

JavaBean：本案例包括两个JavaBean，其中UserInfo是用来表示用户信息的JavaBean组件，而UserRegister则是用来封装数据库操作的JavaBean组件。

用户界面：本案例包括两个页面，分别是用来提交数据的HTML页面文件createuserinfo.html和用来注册用户信息的JSP页面文件deluserinfo.jsp。

步骤详解

1. 创建数据库和表。本案例涉及数据库操作，因此在开发Java程序之前应进行相应的准备工作。创建MySQL数据库userinfo，在userinfo数据库中创建userinfo表，按如图8-11所示进行表设计。

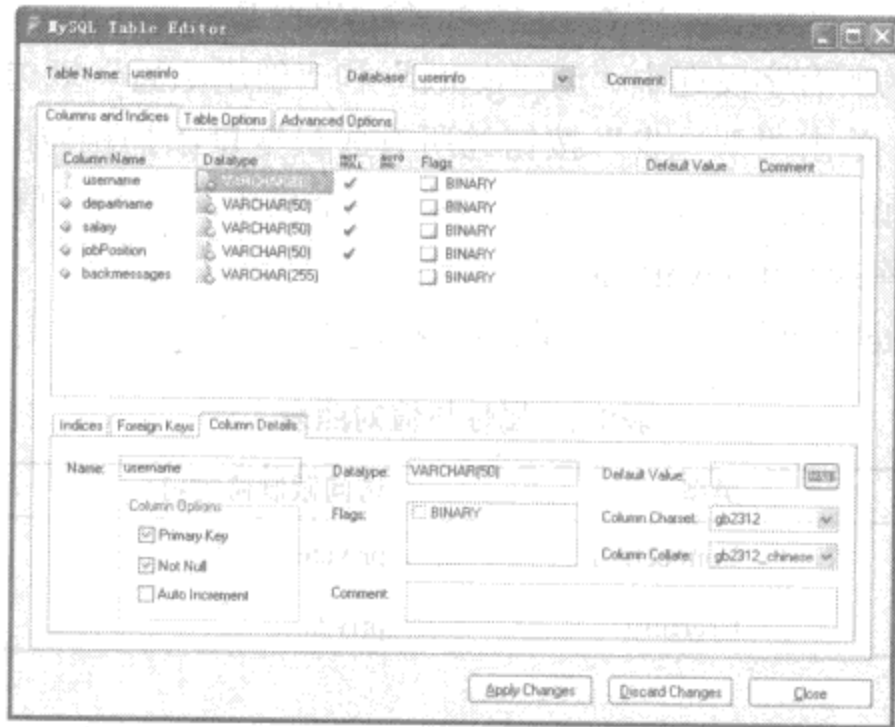


图8-11 “表设计器”界面

2. 创建Web项目，设置Web服务器，本例为“Tomcat 5.5.25”。

3. 安装、配置MySQL的JDBC驱动程序。

添加JAR/文件夹，找到mysql-connector-java-5.0.7-bin.jar文件，将MySQL的JDBC驱动程序导入，完成JDBC驱动程序的安装和配置。

4. 开发表示用户信息的JavaBean。

(1) 添加成员变量，按表8-1设置属性。

表8-1 属性与值对照表

字段名称	类型	访问限制修饰符	初始值
userName	String	private	null
departName	String	private	null
salary	String	private	null
jobPosition	String	private	null
backMessages	String	private	null

(2) 对字段进行封装，如图8-12所示。



图8-12 “封装字段”对话框

(3) 在图8-12中为所有字段选中“创建getter”和“创建setter”复选框，并取消选中“预览所有更改”复选框，单击“下一步”按钮为所有成员变量添加相应的get与set方法。

(4) 保存并编译该文件，此时就成功开发了一个用于表示用户信息的JavaBean。

5. 开发封装数据库操作的JavaBean。

(1) 为UserRegiste添加两个成员变量，按表8-2设置属性。

表8-2 属性与值对照表

字段名称	类型	访问限制修饰符	初始值
conn	Connection	private	null
userInfo	UserInfo	private	null

(2) 按照前述方法封装userInfo字段，为其添加set方法。

(3) 初始化数据库连接，连接数据库方法connectTODB:

```
Class.forName("org.gjt.mm.mysql.Driver");  
//url中指定JDBC中设置的数据库名称  
String url = "jdbc:mysql://localhost:3306/userinfo";  
conn = DriverManager.getConnection(url , username , password);
```

(4) 添加数据到数据库，使用register方法:

```
pstmt=conn.prepareStatement(sql);  
pstmt.setString(1,this.userInfo.getUserName());  
pstmt.setString(2,this.userInfo.getDepartName());  
pstmt.setString(3,this.userInfo.getSalary());  
pstmt.setString(4,this.userInfo.getJobPosition());  
pstmt.setString(5,this.userInfo.getDepartName());  
pstmt.executeUpdate();
```

(5) 保存并编译该文件，此时就成功开发了一个用于执行数据库操作的JavaBean。

6. 开发HTML页面:

```
<html>  
<head>  
  <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
```



```

<title>填写用户信息界面</title>
</head>
<body>
<h3>
<form action="dealuserinfo.jsp" name="myform" method="post">
    姓名: <input type="text" name="userName"><br>
    部门: <input type="text" name="departName"><br>
    薪水: <input type="text" name="salary"><br>
    职位: <input type="text" name="jobPosition"><br>
    备注: <br><textarea cols="30" rows="4" name="backMessages"></textarea><br>
        <input type="submit" value="提交">
</h3>
</form>
</body>
</html>

```

7. 建立JSP文件:

```

<%@page contentType="text/html; charset=gb2312"%>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
    <title>添加信息页面</title>
</head>
<body><h3>
<%
    request.setCharacterEncoding("gb2312");
%>
<jsp:useBean id="userinfo" class="dbjavabean.UserInfo" scope="page">
    <jsp:setProperty name="userinfo" property="*" />
</jsp:useBean>
<jsp:useBean id="userregister" class="dbjavabean.UserRegister"/>
<%
    userregister.setUserInfo(userinfo);    //设置userregister的userinfo属性
    userregister.register();                //调用register方法向数据库中添加数据
    out.println("添加信息成功！<br>");
%>
    添加到数据库中对应属性的值为: <br>
    <!--使用jsp:getProperty标签获取userinfo中的属性并显示-->
    姓名: <jsp:getProperty name="userinfo" property="userName"/><br>
    部门: <jsp:getProperty name="userinfo" property="departName"/><br>
    薪水: <jsp:getProperty name="userinfo" property="salary"/><br>
    职位: <jsp:getProperty name="userinfo" property="jobPosition"/><br>
    备注: <br><textarea cols="30" rows="4" name="backMessages">
        <jsp:getProperty name="userinfo" property="backMessages"/>
    </textarea><br>
</h3></body>
</html>

```

8. 修改web.xml, 将欢迎文件设置为“createuserinfo.html”:

```

- <welcome-file-list>
  <welcome-file>createuserinfo.html</welcome-file>
</welcome-file-list>

```


程序源代码与解释

```
/* * UserRegister.java */
public class UserRegister {
    /** 实例化 UserRegister */
    public UserRegister() {
    }
    private Connection conn;
    private UserInfo userInfo;
    public void setUserInfo(UserInfo userInfo) {
        this.userInfo = userInfo;
    }
    private void connectTODB() {    //初始化数据库连接，并将获取的连接赋值给conn
        try {
            Class.forName("org.gjt.mm.mysql.Driver");
            //url中指定JDBC中设置的数据库名称
            String url = "jdbc:mysql://localhost:3306/userinfo";
            String username = "root";
            String password = "";
            //将获取的连接赋值给conn
            conn = DriverManager.getConnection(url , username , password);
        }
        catch (ClassNotFoundException e1){e1.printStackTrace();}
        catch (SQLException e2) {e2.printStackTrace();}
    }
    public void register() {
        String sql="insert into userinfo values(?,?,?,?)";
        PreparedStatement pstmt=null;    //声明一个预处理对象
        try{
            this.connectTODB();
            //使用conn创建一个预处理对象，并设置其参数
            pstmt=conn.prepareStatement(sql);
            pstmt.setString(1,this.userInfo.getUserName());
            pstmt.setString(2,this.userInfo.getDepartName());
            pstmt.setString(3,this.userInfo.getSalary());
            pstmt.setString(4,this.userInfo.getJobPosition());
            pstmt.setString(5,this.userInfo.getDepartName());
            pstmt.executeUpdate();    //将数据添加到数据库
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(pstmt!=null){
                    pstmt.close();
                }
                if(conn!=null){
                    conn.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```




案例3：测试安全性代码

案例运行效果与操作

本案例是一个简单的Servlet，实现了显示系统信息的基本功能，对文件写入、文件读取、网络访问、命令执行、系统退出等涉及安全性的操作进行了测试，并对系统是否允许上述操作给出了提示。程序运行后，会打开一个IE浏览器窗口，如图8-13所示。

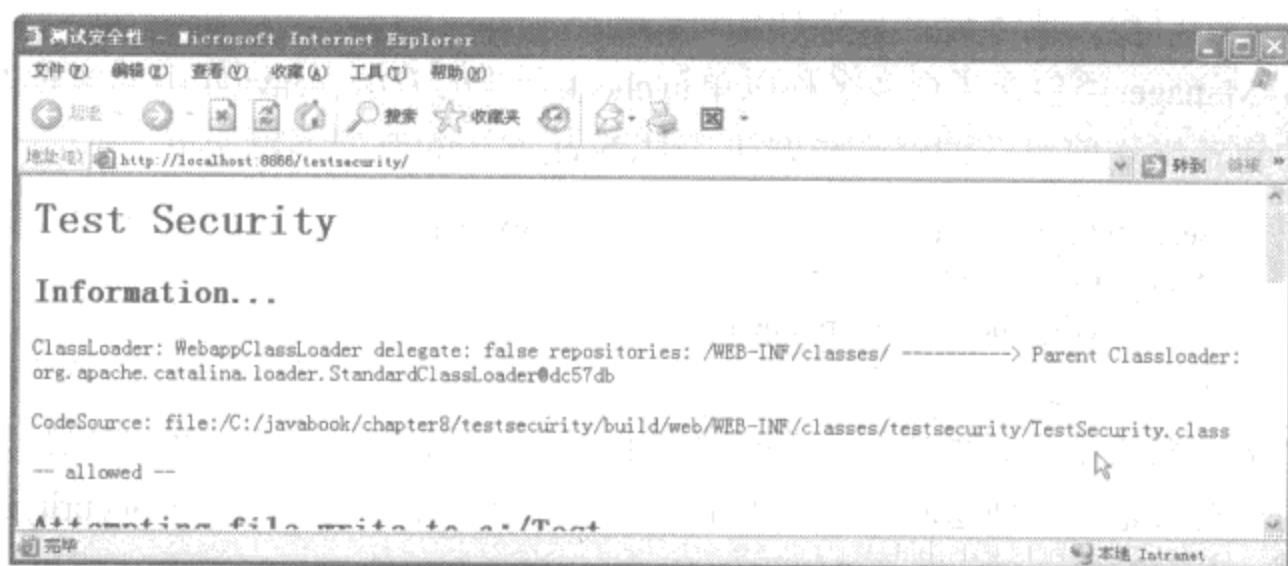


图8-13 运行界面

图8-13中部分条目显示不全，下面列出显示在浏览器窗口中的全部信息：

```
Test Security
Information...
ClassLoader: WebappClassLoader delegate: false repositories: /WEB-INF/classes/ -----> Parent Classloader:
org.apache.catalina.loader.StandardClassLoader@dc57db
CodeSource: file:/C:/javabook/chapter8/testsecurity/build/web/WEB-INF/classes/testsecurity/TestSecurity.class
-- allowed --
Attempting file write to c:/Test...
-- allowed --
Attempting file write to C:/javabook/chapter8/testsecurity/src/java/testsecurity...
-- allowed --
Attempting file read to c:/Ntdetect...
-- allowed --
Attempting file read to C:/Program Files/Apache Software Foundation/Tomcat 5.5/conf/
catalina.properties...
-- allowed --
Attempting to connect to www.baidu.com...
-- rejected -- Connection timed out: connect
Attempting to listen on port 37337...
-- rejected -- Address already in use: JVM_Bind
Attempting to listen on port 8866...
-- rejected -- Address already in use: JVM_Bind
Attempting exec...
```



```
-- rejected -- Cannot run program "dir": CreateProcess error=2, ??????????
Attempting system exit...
-- allowed --
```

制作要点

1. SecurityManager类的用法。

安全管理器是一个允许应用程序实现安全策略的类。它允许应用程序在执行一个可能不安全或敏感的操作前确定该操作是什么，以及是否是在允许执行该操作的安全上下文中执行它。应用程序可以允许或不允许该操作。

SecurityManager类包含了很多名称以单词check开头的方法。Java库中的各种方法在执行某些潜在的敏感操作前可以调用这些方法。对check方法的典型调用如下：

```
SecurityManager security = System.getSecurityManager();
if (security != null) {
    security.checkXXX(argument, . . . );
}
```

因此，安全管理器通过抛出异常来提供阻止操作完成的机会。如果允许执行该操作，则安全管理器例程只是简单地返回，但如果不允许执行该操作，则抛出一个SecurityException。该约定的唯一例外是checkTopLevelWindow，它返回boolean值。

当前的安全管理器由System类中的setSecurityManager方法设置。当前的安全管理器由getSecurityManager方法获得。

2. Servlet的使用。

步骤详解

1. 创建基于Web应用程序的项目，服务器设置为“Tomcat 5.5.25”。
2. 获得系统安全性接口：

```
response.setContentType("text/html;charset=gb2312");
PrintWriter out = response.getWriter();
out.println("<HTML>");
out.println("<HEAD><TITLE>测试安全性</TITLE></HEAD>");
out.println("<BODY>");
out.println(h2o + "<Big>Test Security</Big>" + h2c);
try { //显示系统信息
    out.println(h2o + "Information..." + h2c);
    SecurityManager security = System.getSecurityManager();
    if (security != null) {
        out.println("  Security Manager: " + security.getClass().getName() + p);
    }
    out.println("  ClassLoader: " + this.getClass().getClassLoader() + p);
    out.println("  CodeSource: " + this.getClass().getProtectionDomain().getCodeSource()
        .getLocation()
        + p);
    out.println(" -- allowed -- " + p);
}
```

3. 文件写入安全性测试。


```

out.println(h2o + "Attempting file write to c:/Test..." + h2c);
File f = new File("c:/Test/test.txt");
FileWriter fw = new FileWriter(f);
fw.write("test\n");
fw.close();
out.println(" -- allowed -- " + p);

```

4. 文件读取安全性测试。

```

out.println(h2o + "Attempting file read to c:/Ntdetect..." + h2c);
File f = new File("c:/Ntdetect.com");
FileReader fr = new FileReader(f);
int c = fr.read();
out.println(" -- allowed -- " + p);

```

5. 网络访问安全性测试。

```

out.println(h2o + "Attempting to connect to www.baidu.com..." + h2c);
Socket s = new Socket("www.baidu.com.com", 8080);
out.println(" -- allowed -- " + p);

```

6. 端口监听安全性测试。

```

out.println(h2o + "Attempting to listen on port 8866..." + h2c);
ServerSocket s = new ServerSocket(8866);
Socket c = s.accept();
out.println(" -- allowed -- " + p);

```

7. 命令执行安全性测试。

```

out.println(h2o + "Attempting exec..." + h2c);
Runtime.getRuntime().exec("dir");
out.println(" -- allowed -- " + p);

```

8. 修改web.xml，将欢迎文件设置为“TestSecurity”。

```

- <welcome-file-list>
  <welcome-file>TestSecurity</welcome-file>
</welcome-file-list>

```

程序源代码与解释

```

/* * TestSecurity.java */
package testsecurity;
public class TestSecurity extends HttpServlet {
    /** Processes requests for both HTTP <code>GET</code> and <code>POST</code> methods. */
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=gb2312");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD><TITLE>测试安全性</TITLE></HEAD>");
        out.println("<BODY>");
        out.println(h2o + "<Big>Test Security</Big>" + h2c);
        try { //显示系统信息
            out.println(h2o + "Information..." + h2c);
            SecurityManager security = System.getSecurityManager();

```



```

        if (security != null) {
            out.println(" Security Manager: " + security.getClass().getName()+ p);
        }
        out.println(" ClassLoader: " + this.getClass().getClassLoader() + p);
        out.println(" CodeSource: " + this.getClass().getProtectionDomain().getCodeSource()
.getLocation()
        + p);
        out.println(" -- allowed -- " + p);
    } catch (Exception e) {
        out.println(" -- rejected -- " + e.getMessage() + p);
    }
    //各种安全性测试
    out.println("</BODY></HTML>");
    out.close();
}
/** Handles the HTTP <code>GET</code> method. */
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    processRequest(request, response);
}
/** Handles the HTTP <code>POST</code> method. */
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    processRequest(request, response);
}
/**返回Servlet "short description" */
public String getServletInfo() {
    return "Short description";
}
}

```



案例4：用EJB实现的用户消费信息登记系统

案例运行效果与操作

EJB（Enterprise Java Bean）是Java EE技术的基础和核心内容，在EJB 2.0中定义了三种不同类别的EJB：Session Bean（会话Bean）、Entity Bean（实体Bean）和Message-Driven Bean（消息驱动Bean）。会话Bean根据其是否保存客户的状态，又分为有状态会话Bean和无状态会话Bean。无状态会话Bean不保存与特定客户的对话状态，可以通过EJB容器自由地在客户机之间交换，从而少量的会话Bean就可以服务于大量的客户机。本案例给出一个简单的无状态会话Bean例子，说明Java EE应用程序开发的基本步骤和操作。程序运行后，在浏览器窗口出现用户消费信息的输入界面，如图8-14所示。

在该界面中输入用户姓名、支付方式、信用卡或支票号以及消费的总价格等信息，如图8-15所示。信息输入完毕后，单击“提交”按钮，数据将被写进数据库，写入成功后，浏览器界面如图8-16所示。

如果写入失败，也就是输入的信息有错误，则浏览器会显示信息输入失败的页面，如图8-17所示。

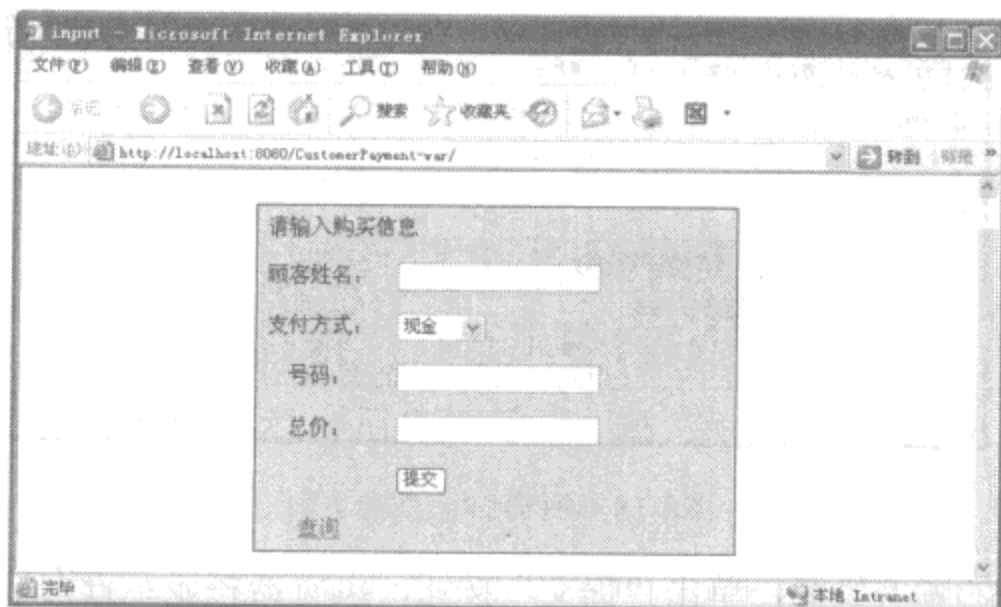


图8-14 初始界面

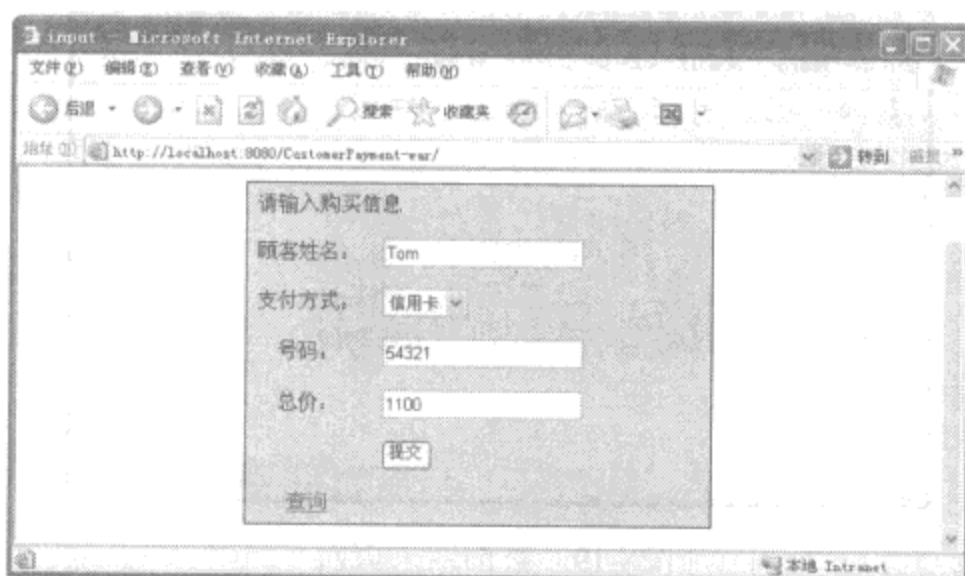


图8-15 输入用户消费信息

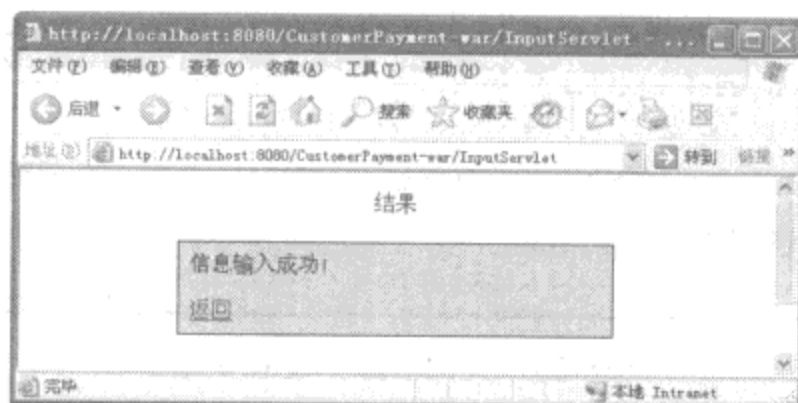


图8-16 信息输入成功提示页面

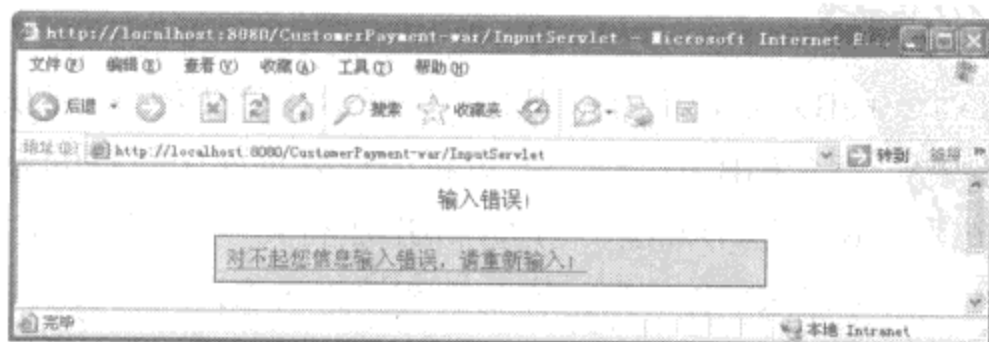


图8-17 信息输入失败提示页面

在如图8-16所示的页面中单击“返回”按钮可以返回消费信息输入界面。如果在如图8-15所示的页面中单击“查询”按钮，可以进入消费信息查询页面，如图8-18所示。

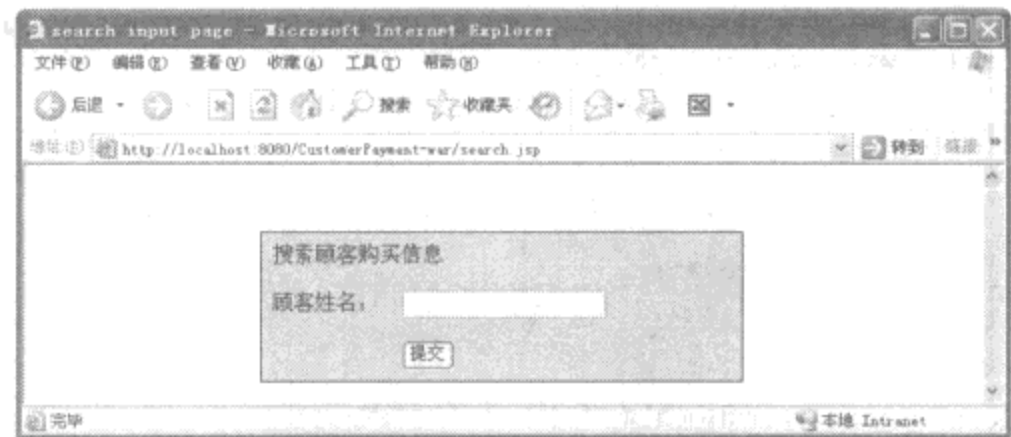


图8-18 消费信息查询页面

输入需要查询的顾客姓名，单击“提交”按钮，浏览器将显示查询结果页面。如果查询成功，浏览器中将显示用户的消费信息。如输入“Tom”进行查询，结果如图8-19所示。

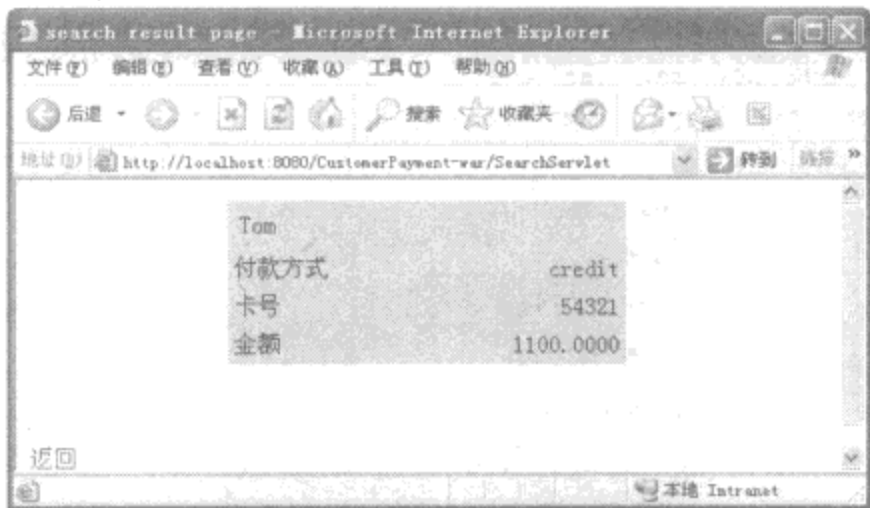


图8-19 查询结果页面

如果查询的信息不存在，则显示如图8-20所示的页面。

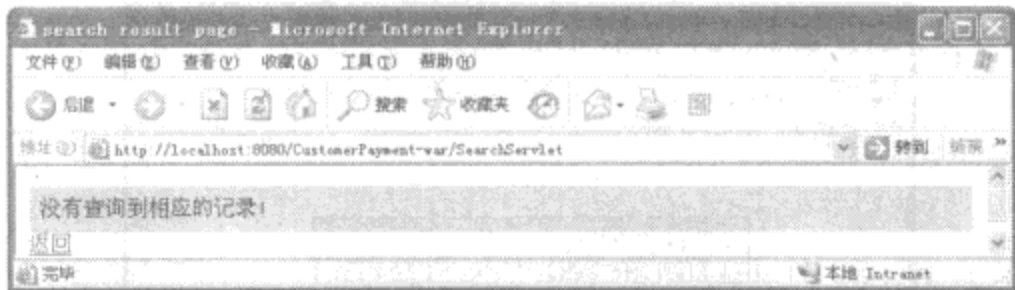


图8-20 查询信息不存在页面

制作要点

1. EJB的概念和分类。

一个企业JavaBean（EJB）是一个可重用的、可移植的J2EE组件。EJB由封装了业务逻辑的多个方法组成。例如，一个EJB可以有一个更新客户数据库中数据的方法的业务逻辑。多个远程和本地客户端可以调用这个方法。另外，EJB运行在一个容器里，允许开发者只关注Bean中的业务逻辑而不用考虑像事务支持、安全性和远程对象访问等复杂和容易出错的事情。EJB以POJO或者普通旧的Java对象形式开发，开发者可以用源数据注释来定义容器如何管理这些Bean。

EJB的三种类型中，会话Bean完成一个清晰的解耦的任务，例如检查客户账户历史记录；实体Bean是一个代表存在于数据库中业务对象的复杂业务实体；消息驱动Bean用于接收异步JMS消息。

当编写一个Enterprise JavaBean组件时，所创建的类必须实现一个EJB接口，并且它必须包含一个名为ejbCreate()的方法。一个EJB接口，例如SessionBean接口指定了一些方法，它们包括以下各项：

- ejbActivate();
- ejbPassivate();
- ejbRemove();
- setSessionContext()。

ejbActivate()和ejbPassivate()方法通知一个Bean，管理该Bean的容器组件正在主动和被动之间切换Bean的状态（这通常是指是在内存中或是交换到磁盘）。ejbRemove()方法使Bean知道它已被从容器中删除。setSessionContext()方法使Bean与一个上下文对象相关联，此上下文对象是为了便于Bean与其容器进行通信。

ejbCreate()方法并不是从零做起创建EJB的。当客户机想要创建新的EJB时，Bean的容器将调用这个Bean类的newInstance()方法，来实例化新的Bean对象。然后容器调用setSessionContext()方法来建立上下文对象，用于与Bean进行通信。最后，容器调用新Bean中的ejbCreate()方法。像ejbCreate()、ejbActivate()和ejbPassivate()这样的方法有时称为对象生存周期方法，以区别于业务逻辑方法。

当开发人员设计一个新的EJB组件时，编写组成EJB类的代码本身是不够的。EJB程序员还必须编写两个将由helper类使用的Java接口。这些强制性接口必须扩展标准的EJBObject和EJBHome接口，而这两个接口都是java.rmi.Remote marker接口的扩展。扩展标准EJBObject接口的接口被称为EJB的远程接口，它指定在Bean自身中定义的业务方法。当应用程序调用EJB中的业务方法时，应用程序并不访问Bean本身。实际上，方法调用被传递给实现EJBObject接口扩展的那个对象。这种做法称为委托，它是EJB体系结构中的一个设计要点。

Bean对EJBObject接口的扩展称为其远程接口，而实现远程接口的对象则称为EJB对象。

EJB还必须具有本地接口。此接口是标准EJBHome接口的扩展。实现Bean的本地接口的对象称为本地对象。本地对象包含一个create()方法，此方法由应用程序调用，而应用程序则必须创建一个Bean实例。本地对象中的create()方法创建一个新的EJB对象。它并不直接创建新的EJB实例，因为不允许直接访问Bean。

2. 无状态会话Bean的使用。

会话Bean之所以被称为会话Bean，是因为它代表的是一个动作、一个过程，它的生存期就是调用它的客户端与它进行会话的过程。

会话Bean根据其是否保存客户的状态，又分为状态会话Bean和无状态会话Bean。状态会话Bean是一种保持会话状态的服务，每个实例都与特定的客户机相关联，在与客户机的方法调用之间维持对话状态。与之相反，无状态会话Bean不保存与特定客户的对话状态。因此状态会话Bean比无状态会话Bean具有更多的功能，而无状态会话Bean实例可以通过EJB容器自由地在客户机之间交换，从而少量的会话Bean就可以服务于大量的客户机。

无状态会话Bean是可以模仿业务过程的组件，它可以在单独的方法调用中被执行。Stateless Session Bean不能够维持一个调用客户的状态，在一个方法调用中，Stateless Session Bean可以维持调用客户的状态，当方法执行完，状态不会被保持。在调用完成后，Stateless Session Bean被立即释放到缓冲池中，所以Stateless Session Bean具有很好的伸缩性，可以支持大量用户的调用。

无状态会话Bean的特点：

- 没有对话状态。

无状态会话Bean可以拥有内部状态，它们的状态不能为特殊的客户端定制。这意味着所有的无状态Bean对于客户端是无差别的，客户端也不能分离它们。客户端必须将所有的必需的客户端数据作为业务逻辑方法的参数传给无状态Bean，无状态Bean可以从外部资源（例如数据库）获得所需的数据。

- 初始化无状态Bean只有一种方法。

我们知道会话Bean的初始化调用ejbCreate()方法，因为无状态会话Bean不能够在方法调用之间保留状态，因此它也不能在客户端给ejbCreate()调用传递数据以后保留状态。它调用不带参数的ejbCreate()或create()。

- 容器可以聚集和重用无状态会话Bean。

3. 程序设计。

本案例是使用NetBeans结合JBoss开发一个无状态会话Bean，并通过Web应用调用这个无状态会话Bean。在本例中，无状态会话Bean，主要负责向数据库中添加用户进行消费的数据，以及在数据库中进行查询。

在本案例中有一个无状态会话Bean——“PaymentBean”，其负责用户与数据库的交互。另外，还有5个JSP页面负责表示层、两个Servlet负责控制器的功能，这些组件的功能如表8-3所示。

表8-3 各个JSP页面的功能

页面	功能
input.jsp	负责显示用户消费信息输入的界面
result.jsp	负责显示用户信息输入成功提示界面
error.jsp	负责显示用户信息输入失败提示界面
search.jsp	负责显示查询用户消费信息的界面
searchresult.jsp	负责显示查询结果的界面
InputServlet	负责处理用户的输入请求
SearchServlet	负责处理用户的查询请求

步骤详解

1. 创建数据库和表：本案例选用开源数据库MySQL作为项目的后台数据库，使用其Test数据库作为测试数据库，在数据库Test中创建一个名为payment的表，表中各个字段的名称和类型如图8-21所示。

2. 在JBoss中配置相应的数据源：本案例是使用NetBeans结合JBoss开发的，需要事先安装和配置好JBoss。JBoss是一个开源的应用服务器，在JBoss中配置数据源的方式就是编写相应的XML配置文件。在本项目中编写一个名为mysql-ds.xml的XML文件，并把该文件存放到JBoss安装目录的server\default\deploy中。mysql-ds.xml配置文件如下：

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



```
<local-tx-datasource>
<jndi-name>mysqls</jndi-name>
<connection-url>jdbc:mysql://localhost/test</connection-url>
<driver-class>com.mysql.jdbc.Driver</driver-class>
<user-name>root</user-name>
<password></password>
<metadata>
<type-mapping>mySQL</type-mapping>
</metadata>
</local-tx-datasource>
</datasources>
```

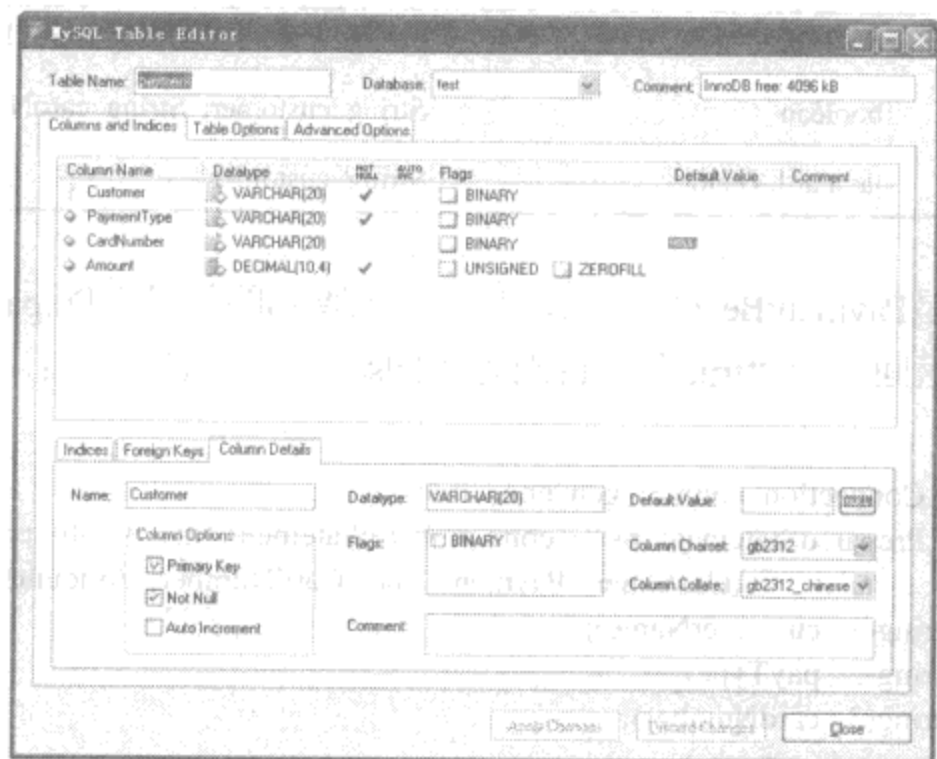


图8-21 payment “表设计器” 界面

3. 配置JBoss类路径：为了使用JBoss和MySQL，首先要把驱动程序类.jar文件（mysql-connector-java-5.0.7-bin.jar）复制到JBoss安装目录的server/default/lib目录中。lib目录的.jar和.zip文件都包含在JBoss服务器的Classpath（类路径）中。

4. 创建企业应用程序项目并设置窗体的属性，将服务器设置为“JBoss Application Server 4.0.5”。

5. 开发EJB模块。

（1）展开项目的EJB模块节点（CustomerPayment-ejb节点），新建“会话Bean”为“PaymentBean”，选择会话类型为“无态”，创建接口选择“远程”选项。创建成功后在项目窗口中Enterprise Bean节点下将出现PaymentSB节点，如图8-22所示。

（2）用鼠标右键单击PaymentSB节点下“远程方法”子节点，依次选择右键菜单中的“添加”/“Business方法”选项，如图8-23所示。按照创建业务方法向导创建业务方法名称为“payByCash”，返回值类型为“boolean”，并添加两个入口参数，分别为String类型的“customer”和Double类型的“amount”。

（3）单击“确定”按钮，完成“payByCash”业务方法的创建。创建成功后会发现“项目”窗口中“远程方法”节点下出现了以图标表示的payByCash节点，即新创建的业务方法。

（4）按照上面的步骤再添加3个业务方法，并按照表8-4进行设置。



图8-22 PaymentSB节点

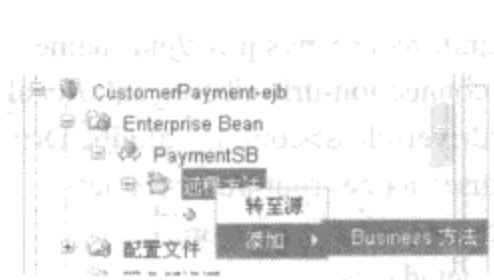


图8-23 创建业务方法

表8-4 3个业务方法的详细信息

方法名称	返回值类型	参数序列
payByCheck	boolean	String customer, String checkNumber, double amount
payByCreditCard	boolean	String customer, String cardNumber, double amount
search	java.util.Vector	String customer

（5）Process()是PaymentBean的方法，功能是把收到的信息用PreparedStatement写入数据库。如果写入成功返回值为“true”，否则为“false”。其代码如下：

```
try{
    java.sql.Connection conn = getConnection();
    java.sql.PreparedStatement ps = conn.prepareStatement("insert into payment
        (Customer, PaymentType, CardNumber, Amount) values (?, ?, ?, ?)");
    ps.setString(1, customerName);
    ps.setString(2, payType);
    ps.setString(3, cardNumber);
    ps.setDouble(4, amount);
    int value = ps.executeUpdate();
    if(value == 1) retType = true;
    ps.close();
    conn.close();
}catch(Exception e) {
    System.out.println(e.getMessage());
    e.printStackTrace();
}
```

（6）Connection方法的作用是通过JNDI技术获取数据库链接。如果成功获取链接则返回连接对象的句柄，否则返回“null”。在Connection方法中加入如下代码：

```
javax.naming.Context ctx = null;
javax.sql.DataSource ds = null;
String url = "localhost:1099";
try {
    java.util.Hashtable h = new java.util.Hashtable( );
    h.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces
.NamingContextFactory");
    h.put(javax.naming.Context.PROVIDER_URL, url);
    ctx = new javax.naming.InitialContext(h);
    ds = (javax.sql.DataSource) ctx.lookup("java:mysqlds");
} catch(Exception ne) {
    System.out.println("UNABLE to get a connection!");
}
return ds.getConnection();
```


(7) 在payByCash方法中添加如下代码:

```
return process(customer, "cash", "", amount); //调用process方法处理数据
```

(8) 在payByCheck方法中添加如下代码:

```
return process(customer, "check", checkNumber, amount); //调用process方法处理数据
```

(9) 在payByCredit方法中添加如下代码:

```
return process(customer, "credit", cardNumber, amount); //调用process方法处理数据
```

(10) search方法的主要功能是根据输入的用户名查找相关用户的消费信息, 并把信息封装在Vector对象中返回:

```
java.util.Vector vec = new java.util.Vector();
try{
    java.sql.Connection conn = getConnection();
    java.sql.PreparedStatement ps = conn.prepareStatement("select * from payment where Customer = ?");

    ps.setString(1, customer);
    java.sql.ResultSet rs = ps.executeQuery();
    while (rs.next()) {
        String temp[] = new String[4];
        temp[0]=rs.getString("Customer");
        temp[1]=rs.getString("PaymentType");
        temp[2]=rs.getString("CardNumber");
        temp[3]=rs.getString("Amount");
        vec.add(temp);
        temp = null;
    }
    rs.close();
    ps.close();
    conn.close();
}
return vec;
```

(11) 修改配置文件jboss.xml的内容:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss PUBLIC
    "-//JBoss//DTD JBOSS 4.0//EN"
    "http://www.jboss.org/j2ee/dtd/jboss_4_0.dtd">
<jboss>
    <enterprise-beans>
        <session>
            <ejb-name>PaymentBean</ejb-name>
            <jndi-name>PaymentBeanNetBeans</jndi-name>
        </session>
    </enterprise-beans>
</jboss>
```

(12) 用鼠标右键单击项目窗口中的CustomerPayment-ejb节点, 选择右键菜单中的“清理并生成项目”选项, 编译并打包EJB模块。如果编译打包成功, 在输出窗口中将显示成功的信息, 如图8-24所示。

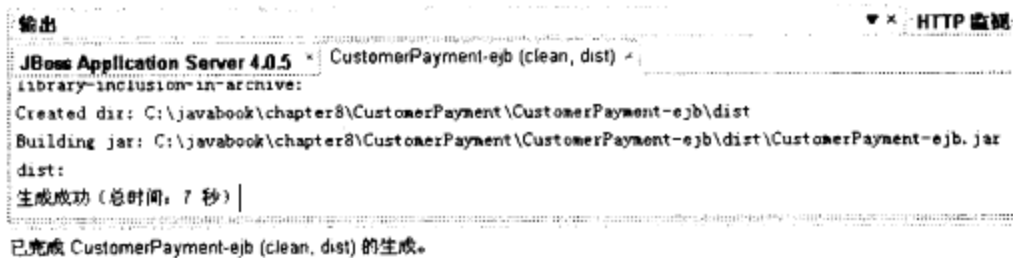


图8-24 输出编译打包成功信息的窗口

6. 开发Web模块。

(1) 展开项目窗口中的“CustomerPayment-war”节点，用鼠标右键单击“Web页”子节点，新建JSP文件input.jsp。input.jsp页的功能是把用户输入的消费信息，如姓名、支付方式、号码、总价等打包到一个HTTP请求中，发送给InputServlet。

(2) 添加input.jsp文件的内容（部分）：

```
<form name="form1" method="post" action="<%=request.getContextPath()%>/InputServlet">
  <p align="center">&nbsp;</p>
  <table width="52%" border="0" align="center" cellpadding="1" cellspacing="0" bgcolor=
"#000000">
    <tr>
      <td> <table width="100%" height="100%" border="0" cellpadding="8" cellspacing="0"
bgcolor="#CCCCCC">
        <tr>
          <td colspan="2" bgcolor="#CCCCCC"><font color="#000000">请输入购买信息</
font></td>
        </tr>
        <tr>
          <td width="27%" align="center">顾客姓名: </td>
          <td width="73%" align="left">
            <input type="text" name="customerName"></td>
          </tr>
      </td>
    </tr>
  </table>
</form>
```

(3) 把“input.jsp”文件设置为本Web模块的欢迎页。

```
- <welcome-file-list>
<welcome-file>input</welcome-file>
</welcome-file-list>
```

(4) 按照同样的步骤在Web模块中添加信息输入成功页面文件result.jsp和信息输入错误页面文件error.jsp，添加查询页面文件search.jsp和查询结果页面文件searchresult.jsp。

(5) 新建Servlet，其名称为InputServlet。添加“lookupHome”的方法，其主要功能是获取“PaymentBean” EJB的Home句柄。lookupHome方法的代码如下：

```
try {
    Hashtable ht = new Hashtable();
    ht.put(Context.INITIAL_CONTEXT_FACTORY,"org.jnp.interfaces.NamingContextFactory");
    ht.put(Context.PROVIDER_URL, "localhost:1099");
    Context ctx = new InitialContext(ht);
    Object home = ctx.lookup("PaymentBeanNetBeans");
    //把远程对象引用转换成Home句柄
    myHome = (PaymentRemoteHome) PortableRemoteObject.narrow(home,
PaymentRemoteHome.class);
} catch (Exception e) {
```



```

        System.out.println("The client was unable to lookup the EJBHome.");
    }
    return myHome;

```

(6) `InputServlet`的`forward`负责把请求进行转发:

```

RequestDispatcher dispatcher = request.getRequestDispatcher(url);
dispatcher.forward(request, response);

```

(7) `InputServlet`的`processRequest`方法是处理请求, 用来被`doGet`和`doPost`方法调用:

```

try {
    double cartValue = Double.parseDouble(request.getParameter("cartValues"));
    home = lookupHome();           //获取EJB的Home句柄
    spayment = home.create();       //创建EJB对象
    if(paymentType.equals("cash")) {
        pay = spayment.payByCash(cusname, cartValue);
    }else if (paymentType.equals("check")){
        pay = spayment.payByCheck(cusname, cardNumber, cartValue);
    }else {
        pay = spayment.payByCreditCard(cusname, cardNumber, cartValue);
    }
    } catch(Exception ce){
        ce.printStackTrace();
    }
    if (pay){           //输入成功后转到result.jsp页
        forward (request,response,"/result.jsp");
    }else {             //输入失败后转到error.jsp页
        forward (request,response,"/error.jsp");
    }
}

```

(8) 再按同样方法新建一个`Servlet`, 其名称为`SearchServlet`, URL映射为`/SearchServlet`。给`Search-Servlet`添加两个方法, 名称分别为“`lookupHome`”和“`forward`”, 这两个方法和`InputServlet`的完全相同, 这里不再赘述。

(9) `SearchServlet`的`processRequest`方法的功能是根据请求中用户名称的信息调用EJB在数据库中进行查询, 把查询的结果封装到`Vector`对象中并转发给`searchresult.jsp`页面以显示查询结果, 其代码如下:

```

Vector result = new Vector( );
request.setCharacterEncoding("GBK");
String cusname = (String)request.getParameter("customerName");
cusname=new String(cusname.getBytes(),"ISO-8859-1");
home = lookupHome();
System.out.println("Home===== "+home);
try{
    spayment = home.create();
    }catch(CreateException ce){
        ce.printStackTrace();
    }
    result = spayment.search(cusname);
    request.setAttribute("result",result);
    forward (request,response,"/searchresult.jsp");
}

```


1) 向类路径中添加并配置EJB模块：本案例开发的两个Servlet都通过JNDI技术调用了EJB模块的远程方法或create方法，并返回了相应的值（对象类型或远程接口类型）。因此如果要使开发的Servlet能够成功地编译，需要将包含EJB模块的JAR文件添加到Servlet的类路径中。在开发环境中选择“添加JAR/文件夹”选项，找到生成的EJB模块JAR文件（本案例为C:\java-book\chapter8\CustomerPayment\CustomerPayment-ejb\dist\CustomerPayment-ejb.jar），完成添加。添加成功后会在“库”节点出现一个“CustomerPayment-ejb.jar”子节点，如图8-25所示。



图8-25 添加成功后的项目窗口

2) 在项目窗口中选中“库”节点，单击鼠标右键，在弹出的菜单中选择“属性”选项，系统将弹出模块属性配置对话框，如图8-26所示。

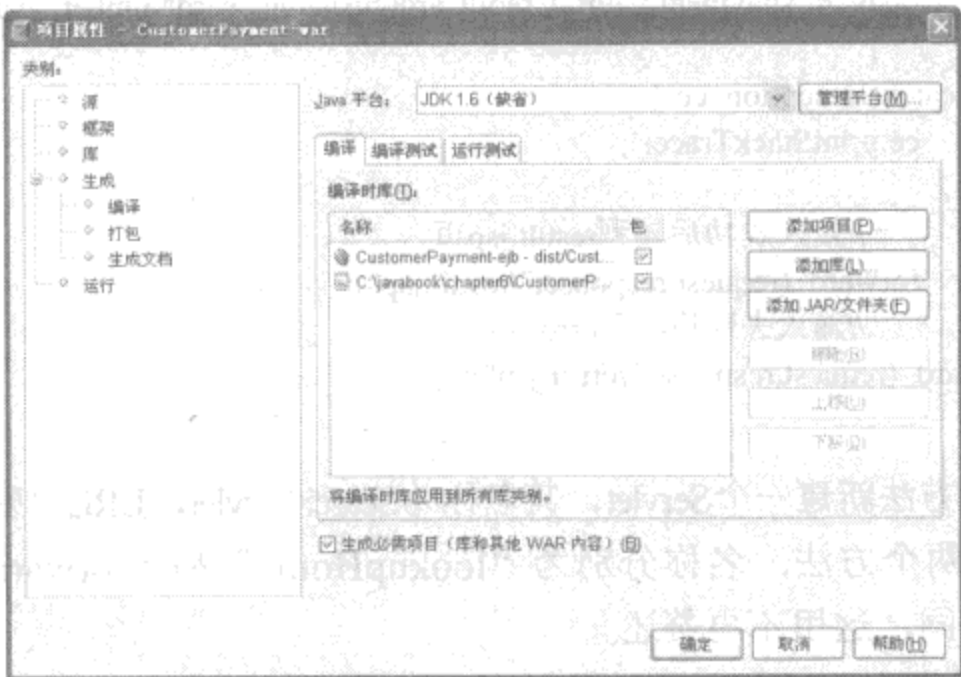


图8-26 模块属性配置对话框

3) 选择“类别”列表中的“库”选项，在窗体的右侧选择“编译”标签，设置列表中“../CustomerPayment-ejb.jar”选项后的“包”复选框为选中状态，单击“确定”按钮确定。

程序源代码与解释

```
/* * SearchServlet.java */
import CustmerPayment.PaymentRemote;
import CustmerPayment.PaymentRemoteHome;
import java.io.*;
import java.net.*;
import java.util.Hashtable;
import java.util.Vector;
import javax.ejb.CreateException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
import javax.servlet.*;
```



```

import javax.servlet.http.*;
public class SearchServlet extends HttpServlet {
    protected PaymentRemoteHome home;
    protected PaymentRemote spayment;
    /**处理HTTP请求中的GET和POST. */
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        Vector result = new Vector( );
        request.setCharacterEncoding("GBK");
        String cusname = (String)request.getParameter("customerName");
        cusname=new String(cusname.getBytes(),"ISO-8859-1");
        home = lookupHome();
        System.out.println("Home====="+home);
        try{
            spayment = home.create();
        }catch(CreateException ce){
            ce.printStackTrace();
        }
        result = spayment.search(cusname);
        request.setAttribute("result",result);
        forward (request,response,"/searchresult.jsp");
    }

    private PaymentRemoteHome lookupHome( ) {
        //利用JNDI查找bean的home路径
        PaymentRemoteHome myHome = null;
        try {
            Hashtable ht = new Hashtable();
            ht.put(Context.INITIAL_CONTEXT_FACTORY,"org.jnp.interfaces.NamingContextFactory");
            ht.put(Context.PROVIDER_URL, "localhost:1099");
            Context ctx = new InitialContext(ht);
            Object home = ctx.lookup("PaymentBeanNetBeans");
            myHome = (PaymentRemoteHome) PortableRemoteObject.narrow(home,
PaymentRemoteHome.class);
        } catch (Exception e) {
            System.out.println("The client was unable to lookup the EJBHome.");
        }
        return myHome;
    }

    protected void forward(HttpServletRequest request,HttpServletResponse response,String url)
throws ServletException,IOException {
        RequestDispatcher dispatcher = request.getRequestDispatcher(url);
        dispatcher.forward(request, response);
    }
    /**处理HTTP的GET方法. */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        processRequest(request, response);
    }
    /**处理HTTP的POST方法. */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        processRequest(request, response);
    }
}

```



```
/** 返回servlet信息描述. */
public String getServletInfo() {
    return "Short description";
}
}
```



案例5: Fibonacci数列

案例运行效果与操作

斐波纳契数列是中世纪意大利数学家斐波纳契在《算盘全书》中提出的。这个数列前两项为1，从第三项开始，前面相邻两项之和，构成了后一项。本案例给出一个简单的有状态会话Bean例子，实现了一个Fibonacci数列的程序。程序运行后，输出界面如图8-27所示。

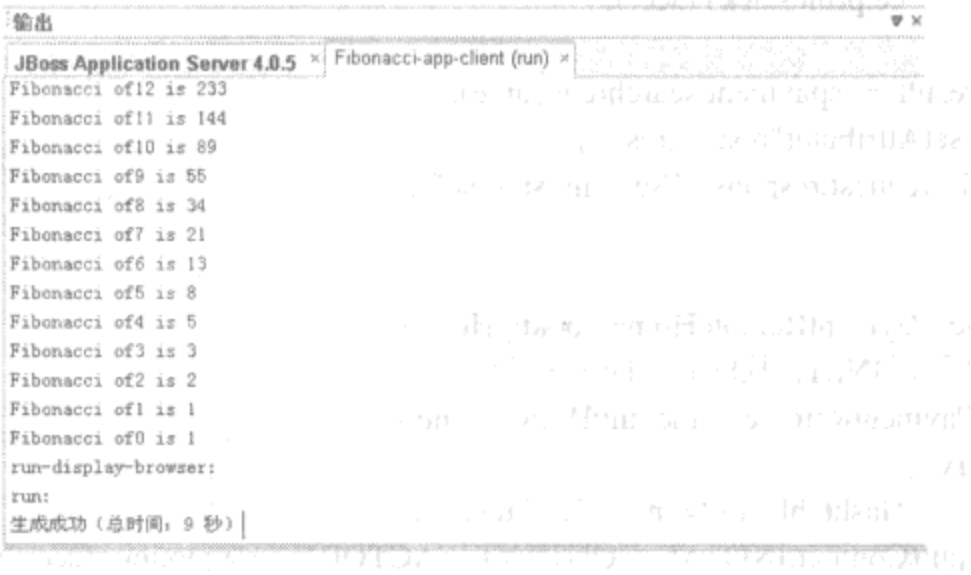


图8-27 运行界面

制作要点

1. 有状态会话Bean的使用。

有状态会话Bean每个用户有自己特有的一个实例，在用户的生存期内，Bean保持了用户的信息，即“有状态”；一旦用户灭亡（调用结束或实例结束），Bean的生命期也告结束。即每个用户最初都会得到一个初始的Bean。

所有的有状态会话Bean都需要一个会话Bean类，它实现了SessionBean接口。另外，所有允许远程访问的企业Bean都必须有一个Home接口和一个Remote接口，分别介绍如下：

• SessionBean接口

SessionBean接口继承EnterpriseBean接口，后者又继承Serializable接口。SessionBean接口声明了ejbRemove、ejbActivate、ejbPassivate和setSessionContext方法。该接口没有声明ejbCreate方法，一般ejbCreate方法初始化企业Bean的状态。企业Bean必须至少实现一个ejbCreate方法。

• Home接口

企业Bean的Home接口继承javax.ejb.EJBHome接口。对于会话Bean，Home接口的目标是要定义客户端可访问create方法。

• Remote接口

Remote接口继承EJBObject接口，它定义客户端调用的业务方法。

2. 程序设计。

本案例中，客户端通过JNDI利用Home接口创建出一个Remote远程对象，再通过调用Remote远程对象中的getFibonacci方法与FibonacciBean进行交互，处理逻辑（计算Fibonacci数列的值）。客户端并不直接与Bean进行交互。

步骤详解

1. 创建基于企业应用程序的项目，服务器设置为“JBoss Application Server 4.0.5”。

2. 开发EJB模块。

(1) 展开项目的EJB模块节点（Fibonacci-ejb节点），新建会话Bean，设置EJB的名称为“FibonacciBean”，指定其所在的包名为“Fibonacci”，选择会话类型为“有态”，创建接口选中“远程”选项，完成有状态会话Bean的创建。创建成功后在项目窗口中Enterprise Beans节点下将出现FibonacciSB节点。

(2) 建立FibonacciBean.java:

```
package Fibonacci;
import javax.ejb.*;
public class FibonacciBean implements SessionBean, FibonacciRemoteBusiness {
    private SessionContext context;
    public void setSessionContext(SessionContext aContext) { context = aContext; }
    public void ejbActivate() { }
    public void ejbPassivate() {}
    public void ejbRemove() {}
    public void ejbCreate() {
        System.out.println("Bean created");
    }
    利用递归算法计算数列值:
    public long getFibonacci(int n) {
        if (n==0) return 1;
        else if (n==1) return 1;
        else
            return getFibonacci(n - 1) + getFibonacci(n - 2);
    }
}
```

(3) 用鼠标双击FibonacciRemote.java节点，在代码编辑器中添加如下代码:

```
package Fibonacci;
import java.rmi.RemoteException;
import javax.ejb.EJBObject;
public interface FibonacciRemote extends EJBObject, FibonacciRemoteBusiness {
    public long getFibonacci(int n) throws RemoteException;
}
```

(4) 建立jboss.xml文件:

```
<?xml version="1.0"?>
<jboss>
  <enterprise-beans>
    <session>
      <ejb-name>FibonacciBean</ejb-name>
      <jndi-name>FibonacciRemoteHome</jndi-name>
```



```

</session>
</enterprise-beans>
</jboss>

```

(5) 用鼠标右键单击项目窗口中 **Fibonacci-ejb** 节点，选择右键菜单中的“清理并生成项目”选项，编译并打包EJB模块。如果编译打包成功，在输出窗口中将显示成功的信息，如图8-28所示。

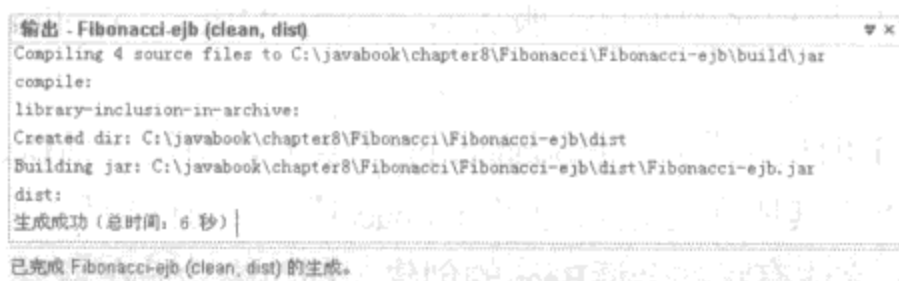


图8-28 输出编译打包成功信息的窗口

3. 添加FibonacciClient.java的代码:

```

import Fibonacci.FibonacciRemote;
import Fibonacci.FibonacciRemoteHome;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
public class FibonacciClient {
    /** Creates a new instance of FibonacciClient */
    public FibonacciClient() {}
    public static void main(String[] args) {
        try {
            //设置属性
            java.util.Properties p = new java.util.Properties();
            p.put(Context.INITIAL_CONTEXT_FACTORY,
                "org.jnp.interfaces.NamingContextFactory");
            p.put(Context.URL_PKG_PREFIXES, "jboss.naming:org.jnp.interfaces");
            p.put(Context.PROVIDER_URL, "localhost:1099");
            Context jndiContext = new InitialContext(p);           //产生JNDI上下文对象
            Object ref = jndiContext.lookup("FibonacciRemoteHome"); //产生Home接口对象
            // 将此对象重构为远程接口对象即可重新调用它的方法
            FibonacciRemoteHome home = (FibonacciRemoteHome)
                PortableRemoteObject.narrow( ref, FibonacciRemoteHome.class);
            FibonacciRemote ff = home.create(); //产生Remote接口对象
            long result;
            for (int i = 40; i >= 0; i--) {
                //调用远程对象的getFibonacci方法计算数列值
                result = ff.getFibonacci(i);
                System.out.println("Fibonacci of" + i + " is " + result);
            }
        }
    }
}

```

4. 向类路径中添加并配置EJB模块：步骤同案例4类似，本案例的JAR文件为Fibonacci-ejb.jar。

程序源代码与解释

```
/*FibonacciBean.java*/
import javax.ejb.*;
public class FibonacciBean implements SessionBean, FibonacciRemoteBusiness {
    private SessionContext context;
    public void setSessionContext(SessionContext aContext) {
        context = aContext;
    }
    public void ejbActivate() { }
    public void ejbPassivate() {}
    public void ejbRemove() { }
    public void ejbCreate() {
        System.out.println("Bean created");
    }
    public long getFibonacci(int n) { //计算数列值
        if (n==0) return 1;
        else if (n==1) return 1;
        else
            return getFibonacci(n - 1) + getFibonacci(n - 2); //Fibonacci数列递归计算
    }
}
```



案例6：简单的图书信息管理系统

案例运行效果与操作

本案例给出一个简单的图书信息管理系统，使用了EJB中的容器管理的持久存储实体Bean（Container-Managed Persistence Bean, CMP）。程序运行后，在浏览器窗口出现添加书籍的输入界面，如图8-29所示。

在该界面中用户输入图书编号、名称、价格等信息，如图8-30所示。信息输入完毕后，单击“添加”按钮，数据将被写进数据库，写入成功后，浏览器界面如图8-31所示。

依次添加几本书，然后单击“查看所有图书”超级链接，则浏览器会显示当前库中的所有图书，如图8-32所示。

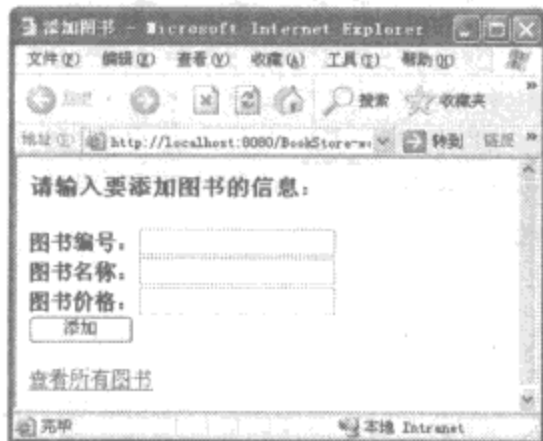


图8-29 初始界面



图8-30 添加图书窗口

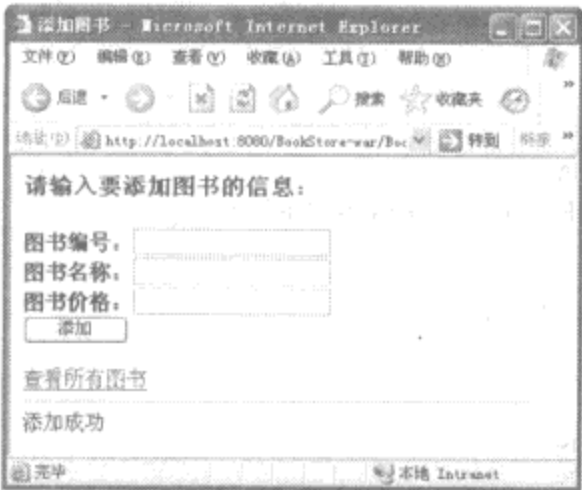


图8-31 添加成功窗口

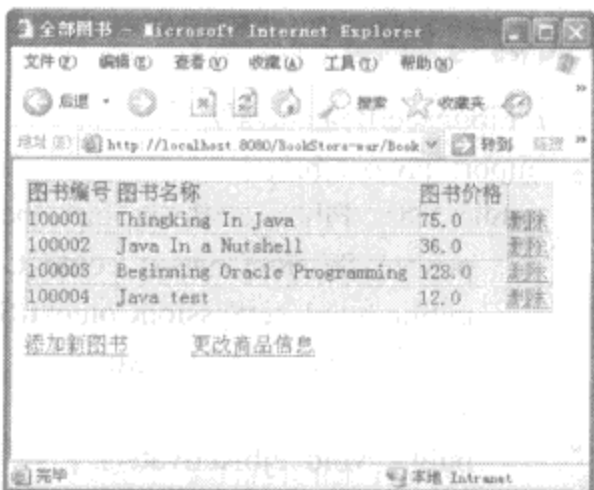


图8-32 查看图书信息

在如图8-32所示的页面中单击对应记录右侧的“删除”链接，可以将该记录从数据库中删除。如单击编号为100004的图书对应的“删除”链接后，浏览器界面如图8-33所示。可以看出，对应的图书确实被删除了。

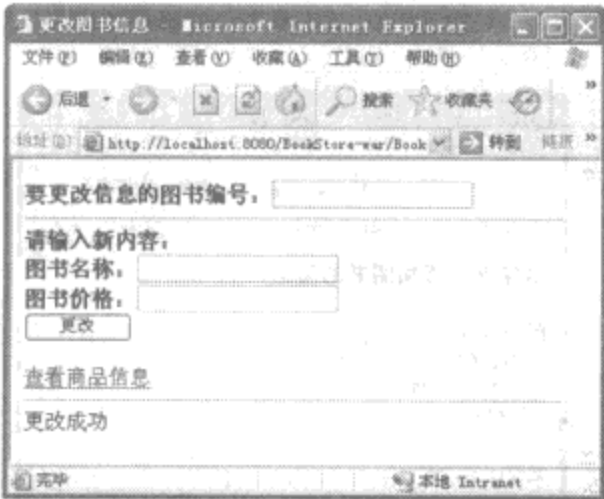
单击“添加新图书”链接，会转到如图8-29所示界面。单击“更改商品信息”链接，浏览器将显示如图8-34所示界面。在如图8-34所示界面输入修改的信息，如修改编号为100002的图书信息，其名称不变，价格改为36.5，单击“更改”按钮后，浏览器界面如图8-35所示。此时单击“查看商品信息”链接，浏览器界面如图8-36所示，可以看出，图书信息确实被更改了。



图8-33 删除图书后的界面



图8-34 更改图书信息窗口



8-35 更改成功窗口

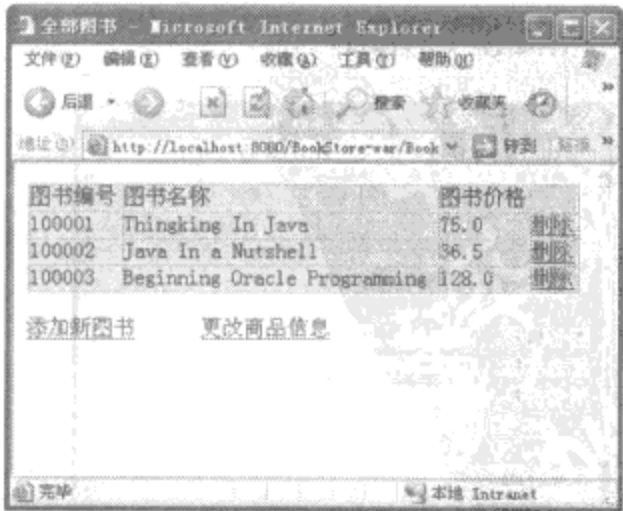


图8-36 更改信息后的窗口

制作要点

1. 实体Bean的使用。

实体Bean是管理持久化数据的一个对象，潜在使用一些相关的Java对象并且可以依靠主键被唯一识别。实体Bean表示来自数据库的持久化数据，例如客户表中的一个记录，或者一个员工表中的一个员工记录。实体Bean也可以被多个客户端共享。实体Bean对象特定变量能够保持持久化。根据管理实体Bean持久存储方式的不同，存在以下两种类型的实体Bean：容器管理的持久存储实体Bean（Container-Managed Persistence Bean, CMP）和Bean管理的持久存储实体Bean（Bean-Managed Persistence Bean, BMP）。

以下情况需要使用实体Bean：

（1）Bean代表一个商务实体而不是一个过程。例如表示信用卡的CreditCardEJB要做成EntityBean，而信用卡核实的VerifierEJB就只能做成会话Bean。

（2）Bean的状态是需要持久存储的。如果Bean的实例结束了或者J2EE服务器关闭，它的状态依然存在，只是回到向数据库这样的存储设备中了。

虽然可能存在一些很合理的例外，但是一般情况下使用CMP这种设计。至于为什么要选择CMP，而不是BMP，这里有三大主要原因。

（1）与BMP相比，CMP具有跨多种不同数据库的可移植性，因为CMP实体Bean不包含任何特定于数据库的持久性代码。CMP易于设计、实现和维护。

（2）通常，CMP拥有好于BMP的性能，因为EJB容器将自动生成特定于数据库的代码，并且这些代码将为目标数据库而优化。

（3）CMP通过使用本地接口，使得在相关EJB的网络中程序性地(programmatically)进行导航变得非常容易。

但以下情况需要使用BMP：

（1）在系统的业务对象模型中表示持久性数据。

（2）利用EJB部署模型，允许通过修改XML做出行为的改变，而非代码的改变。

实体Bean的特点如下：

• 持久性

为了实体Bean的状态保存在存储设备中，所以它具有持久性。持久性是指实体Bean的状态跨越应用程序和J2EE服务器处理过程的生存期，就是说应用程序结束或者服务器终止，实体Bean的状态仍然保留。数据库中的数据是持久性的，就算关闭数据库服务器或者相应的应用程序，它们仍然是存在的。而会话Bean不是持久的，受服务器关机影响。

• 共享访问

实体Bean可以被多客户端所共享。由于多个客户端可能同时去修改同一个数据，所以在调用过程中事务机制非常重要。典型情况下，EJB容器都支持事务机制。在这种情况下，可以在Bean的部署描述符中确定它的事务属性。开发者不必为事务界限编码，容器会自动划分事务界限。而会话Bean通常一个用户使用一个。

• 主键

每一个实体Bean实例都有一个唯一对象标识。例如一个特定的实体Bean实例可能用一个特定的数字来标识。这个唯一标识就是主键，可以让客户端找到对应的实体Bean实例。

• 关系

像关系数据库中的一个表一样，实体Bean之间也会有关系。例如在一个学校登记系统中，表示学生的StudentEJB和表示课程的CourseEJB，因为学生必须登记上课而产生关系。

实体Bean关系的实现方法对于BMP和CMP是不同的。BMP需要编码来实现关系，而CMP是由容器来处理关系的（开发者必须在部署描述符中定义关系）。

2. 程序设计。

本案例是使用NetBeans结合JBoss开发一个实体Bean，并通过Web应用调用这个实体Bean。在本例中实体Bean负责与数据库交互，主要是向数据库中添加、修改、删除图书数据，以及在数据库中进行查询。

在本案例中有3个JSP页面负责表示层、一个Servlet负责控制器的功能，这些组件的功能如表8-5所示。

表8-5 各个JSP页面的功能

页面	功能
addbook.jsp	添加图书界面
changebook.jsp	修改图书信息界面
showanddelete.jsp	负责显示图书信息，也可以删除图书信息界面
BookStoreServlet	负责处理用户的输入请求

步骤详解

1. 创建数据库和表：本案例涉及数据库操作，因此在开发Java EE程序之前应进行相应的准备工作，包括创建数据库和表等。在这里仍然选用开源数据库MySQL作为项目的后台数据库，使用其Test数据库作为测试数据库，在数据库Test中创建一个名为bookstore的表，表中各个字段的名称和类型如图8-37所示。具体的创建过程读者可以参照案例2。

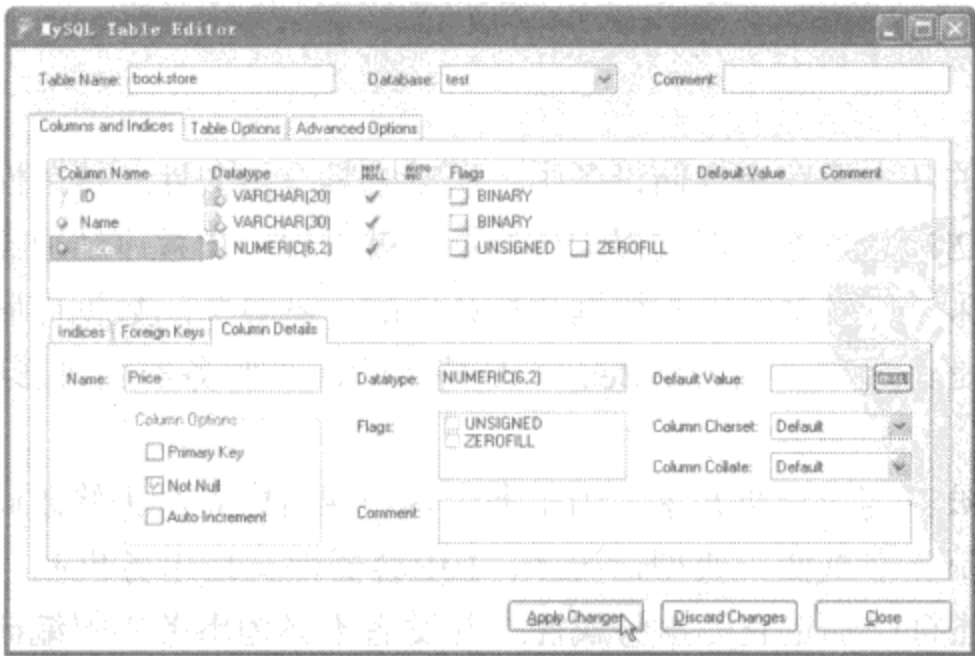


图8-37 “表设计器”界面

2. 在JBoss中配置相应的数据源：本案例是使用NetBeans结合JBoss开发的，需要事先安装和配置好JBoss。JBoss是一个开源的应用服务器，在JBoss中配置数据源的方式就是编写相

应的XML配置文件。在本项目中编写一个名为mysql-ds.xml的XML文件，并把该文件存放到JBoss安装目录的server\default\deploy中。mysql-ds.xml配置文件如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
<local-tx-datasource>
<jndi-name>cmpbook</jndi-name>
<connection-url>jdbc:mysql://localhost/test</connection-url>
<driver-class>com.mysql.jdbc.Driver</driver-class>
<user-name>root</user-name>
<password></password>
<metadata>
<type-mapping>mySQL</type-mapping>
</metadata>
</local-tx-datasource>
</datasources>
```

在编写完配置文件后，在NetBeans中启动JBoss服务器，测试数据源是否创建成功。如果数据源配置正确，在启动信息窗口中，就会看到数据源创建成功的信息，如图8-38所示。

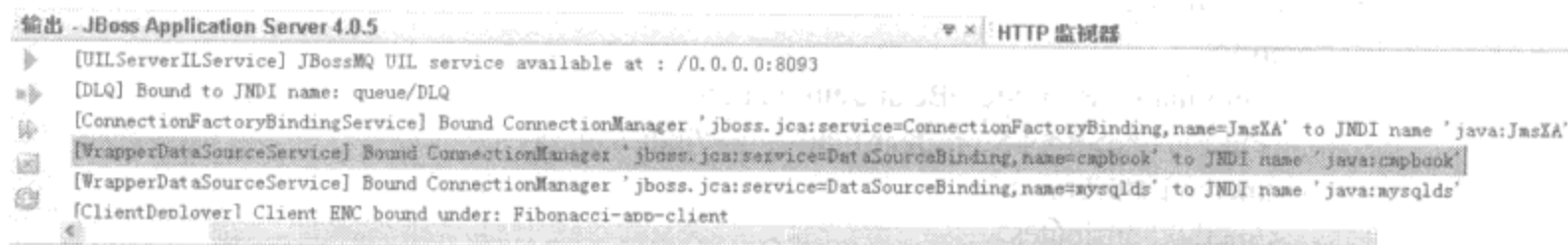


图8-38 数据源创建成功

3. 创建项目：建立基于企业应用程序的项目，服务器设置为“JBoss Application Server 4.0.5”。

4. 开发EJB模块。

(1) 选中项目的EJB模块节点（BookStore-ejb节点），建立Java包BookStore。

(2) 展开项目的EJB模块节点（BookStore-ejb节点），新建实体Bean，名称设为BookStoreBean，选择持久性类型为“容器”，创建接口仅选中“远程”选项。

(3) 完成实体Bean的创建后，在项目窗口中Enterprise Beans节点下将出现BookStoreEB节点。可以看出，NetBeans自动添加了一个名称为key的关系字段，并且创建了3个远程方法，即findByPrimaryKey、create和getKey。此时还自动生成了相应的Home接口、Remote接口以及一个业务接口（对应“源包”/“BookStore”下的三个Java文件：BookStoreRemoteHome.java、BookStoreRemote.java和BookStoreRemoteBusiness.java）。

(4) 添加三个CMP字段，字段名称分别为id、name和price，类型分别为String、String和Double，向远程接口中添加获取方法和设置方法。

(5) 删除自动生成的create方法，添加新的create方法，把三个CMP字段添加为入口参数，并在ejbCreate方法体中添加如下代码：

```
this.setId(id);
this.setName(name);
this.setPrice(price);
```


- (6) 添加finder方法，指定名称为findAll。
- (7) 配置ejb-jar.xml，设置BookStoreEB的主键字段为id，并删除CMP字段中的key，如图8-39所示。

CMP 字段						
字段名称	类型	本地 g...	本地 s...	远程 g...	远程 s...	描述
id	java.lang.String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
key	java.lang.String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
name	java.lang.String	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
price	double	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
<div>添加(A) 编辑(E) 删除(R)</div>						

图8-39 设置CMP字段

- (8) 编辑jboss.xml文件的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss PUBLIC
    "-//JBoss//DTD JBOSS 4.0//EN"
    "http://www.jboss.org/j2ee/dtd/jboss_4_0.dtd">
<jboss>
  <enterprise-beans>
    <entity>
      <ejb-name>BookStoreBean</ejb-name>
      <jndi-name>BookStoreBeanJndiName</jndi-name>
      <method-attributes>
      </method-attributes>
    </entity>
  </enterprise-beans>
</jboss>
```
- (9) 下面添加一个定义CMP各个选项与数据库表之间映射关系的配置文件jbossCMP-jdbc.XML，并添加如下代码。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jbossCMP-jdbc PUBLIC "-//JBoss//DTD JBOSSCMP-JDBC 3.2//EN" "http://www.jboss.org/j2ee/dtd/jbossCMP-jdbc_3_2.dtd">
<jbossCMP-jdbc>
  <defaults>
    <datasource>java:cmpbook</datasource>
    <datasource-mapping>Hypersonic SQL</datasource-mapping>
  </defaults>
  <enterprise-beans>
    <entity>
      <ejb-name>BookStoreBean</ejb-name>
      <create-table>true</create-table>
      <remove-table>false</remove-table>
      <table-name>bookstore</table-name>
      <cmp-field>
        <field-name>name</field-name>
        <column-name>name</column-name>
      </cmp-field>
      <cmp-field>
        <field-name>price</field-name>
        <column-name>price</column-name>
```



```

</cmp-field>
<cmp-field>
    <field-name>id</field-name>
    <column-name>id</column-name>
</cmp-field>
</entity>
</enterprise-beans>
</jbosscmp-jdbc>

```

(10) 编译并打包EJB模块，步骤参见案例4。

5. 开发Web模块。

(1) 新建JSP文件addbook，这个JSP页的功能是把用户输入的图书信息打包到一个HTTP请求中，并发送给BookStoreServlet。编辑addbook.jsp页面：

```

<%@page contentType="text/html"%>
<%@page pageEncoding="gb2312"%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
        <title>添加图书</title>
    </head>
    <body> <h4>
        <h3>请输入要添加图书的信息： </h3>
        <form action="BookStoreServlet" method="post">
            图书编号： <input type="text" name="id"><br>
            图书名称： <input type="text" name="name"><br>
            图书价格： <input type="text" name="price"><br>
            <input type="hidden" value="add" name="action">
            <input type="submit" value=" 添加 " >
        </form></h4>
        <a href="BookStoreServlet?action=showall">查看所有图书</a>
    <%
        Object obj=session.getAttribute("add");
        if(session.getAttribute("add")!=null){
            if(((String)obj).equals("ok")){
                out.println("<hr>添加成功");
            }else{
                out.println("<hr>添加失败");
            }
        }
        session.setAttribute("add",null);
    %>
    </body>
</html>

```

(2) 把“addbook.jsp”文件设置为Web模块的欢迎页。

(3) 按照同样的步骤在Web模块中添加“修改图书信息”页面文件changebook.jsp和“显示图书信息”页面文件showanddelete.jsp。

(4) 向BookStore-war中添加一个名称为BookStore的包。新建一个Servlet，名称为“BookStoreServlet”，添加lookupBookStoreBeanIndiName方法，方法的签名为“private BookStoreRemoteHome lookupBookStoreBeanIndiName()”。这个方法的主要功能是获取“BookStoreBean”EJB的Home句柄。添加如下代码：


```

Hashtable ht = new Hashtable();
ht.put(Context.INITIAL_CONTEXT_FACTORY,"org.jnp.interfaces.NamingContextFactory");
ht.put(Context.PROVIDER_URL, "localhost:1099");
Context ctx = new InitialContext(ht);
Object remote = ctx.lookup("BookStoreBeanJndiName");
rv = (BookStoreRemoteHome) javax.rmi.PortableRemoteObject.narrow (remote,
BookStoreRemoteHome.class);
return rv;

```

(5) 为BookStoreServlet添加一个名称为“init”的方法，其功能是调用lookupBookStoreBeanJndiName方法的返回值初始化remoteHome变量：

```
this.remoteHome=this.lookupBookStoreBeanJndiName();
```

(6) 为BookStoreServlet添加名为“forward”的方法，此方法的功能是把页面跳转到由参数URL指定的页面）。在方法体中添加如下代码：

```

RequestDispatcher dispatcher = request.getRequestDispatcher(url);
dispatcher.forward(request, response);

```

(7) 为BookStoreServlet添加名为“showAllBook”的方法，此方法的功能是获取数据库中的记录，并将这些记录的相应值添加到名称为vec的Vector对象中，再将vec添加到名称为all的session对象中：

```

//调用EJB中的远程方法，其返回值为Collection
Collection allBook = remoteHome.findAll();
Iterator ite=allBook.iterator();
Vector vec=new Vector();
while(ite.hasNext()){//遍历该Iterator
    Object obj=ite.next();
    //将obj转换为BookStoreRemote对象
    BookStoreRemote temp=(BookStoreRemote) PortableRemoteObject.narrow(obj,
BookStoreRemote.class);
    //获取id值
    String tempId=new String(temp.getId().getBytes("ISO-8859-1"),"GB2312");
    //获取name值
    String tempName=new String (temp.getName().getBytes("ISO-8859-1"),"GB2312");
    //获取price的值
    String tempPrice=new String (temp.getPrice().toString().getBytes("ISO-8859-
1"),"GB2312");

    vec.add(tempId);          //将id值添加到Vector中
    vec.add(tempName);        //将name值添加到Vector中
    vec.add(tempPrice);       //将price值添加到Vector中
}
session.setAttribute("all",vec);    //将vec对象添加到名称为all的session对象中
this.forward(request,response,"showanddelete.jsp");//转到showanddelete.jsp页面

```

(8) 在BookStoreServlet的processRequest方法中添加如下代码（此方法的功能是处理请求，用来被doGet和doPost方法调用）：

```

//如果action的值为添加或add，则向数据库中添加数据
if(action.equals("add")){
    if(id!=null&&!id.trim().equals("")){ //id值不为空
        //调用create方法
    }
}

```



```

        remoteHome.create(new String(id.getBytes(),"ISO-8859-1"),new
            String(name.getBytes(),"ISO-8859-1"),Double.valueOf(price));
        session.setAttribute("add","ok");        //将add属性的值设置为ok
    }
    this.forward(request,response,"addbook.jsp");        //转到addbook.jsp页面
} else if(action.equals("delete")){
    //如果action属性的值为delete, 则删除指定id的记录
    BookStoreRemote remote=remoteHome.findByPrimaryKey(id);
    if(remote!=null){
        remote.remove();
        this.showAllBook(session,request,response);
    }
} else if(action.equals("change")){
    //如果action的值为更改或delete, 则删除指定id的记录
    if(id!=null&&!id.trim().equals("")){
        BookStoreRemote remote=remoteHome.findByPrimaryKey(id);
        if(remote!=null){
            remote.setName(new String(name.getBytes(),"ISO-8859-1"));
            remote.setPrice(Double.valueOf(price));
            session.setAttribute("change","ok");        //将change属性的值设置为ok
        } else { //删除不成功
            session.setAttribute("change","no");        //将change属性的值设置为no
        }
    }
    this.forward(request,response,"changebook.jsp");        //转到changebook.jsp页面
} else if(action.equals("showall")){
    //如果action的值为showall, 则获取数据库中的所有记录
    this.showAllBook(session,request,response);
}

```

6. 向类路径中添加并配置EJB模块：本案例开发的Servlet通过JNDI技术调用了EJB模块的远程方法或create方法，并返回了相应的值（对象类型或远程接口类型）。因此如果要使开发的Servlet能够成功地编译，需要将包含EJB模块的JAR文件添加到Servlet的类路径中。完成JAR文件的添加与配置的步骤参见前面案例，本案例JAR文件为BookStore-ejb.jar。

程序源代码与解释

```

/* * BookStoreServlet.java*/
public class BookStoreServlet extends HttpServlet {
    private BookStoreRemoteHome remoteHome = null;
    /**处理HTTP的GETThePOST请求 */
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=gb2312");
        request.setCharacterEncoding("gb2312");
        HttpSession session=request.getSession(true);
        //获取action的值, 根据action值的不同, 进行相应的操作
        String action=request.getParameter("action").trim();
        String id=request.getParameter("id");        //获取id的值
        String name=request.getParameter("name");    //获取name的值
        String price=request.getParameter("price");  //获取price的值
        try {
            //参见步骤详解5中的(8)
            //.....

```



```

        } catch (javax.ejb.RemoveException ex) {
            ex.printStackTrace();
        } catch (javax.ejb.FinderException ex) {
            ex.printStackTrace();
        } catch (RemoteException ex) {
            ex.printStackTrace();
        } catch (UnsupportedEncodingException ex) {
            ex.printStackTrace();
        } catch (CreateException ex) {
            ex.printStackTrace();
        }
    }
    /**处理HTTP句柄GET方法*/
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }
    /**处理HTTP句柄POST方法*/
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }
    /**返回Servlet信息*/
    public String getServletInfo() {
        return "Short description";
    }
    private BookStoreRemoteHome lookupBookStoreBeanIndiName() {
        BookStoreRemoteHome rv=null;
        //参见步骤详解5中的 (4)
        //...
    }
    public void init()throws ServletException {
        this.remoteHome=this.lookupBookStoreBeanIndiName();
    }
    private void forward(HttpServletRequest request,HttpServletResponse response,String url) throws
ServletException,IOException {
        RequestDispatcher dispatcher = request.getRequestDispatcher(url);
        dispatcher.forward(request, response);
    }
    public void showAllBook(HttpSession session,HttpServletRequest request, HttpServletResponse
response){
        try {
            //参见步骤详解5中的 (7)
        } catch (ServletException ex) {
            ex.printStackTrace();
        } catch (RemoteException ex) {
            ex.printStackTrace();
        } catch (UnsupportedEncodingException ex) {
            ex.printStackTrace();
        } catch (FinderException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}

```


本章小结

J2EE是一套全然不同于传统应用开发的技术架构，它以Java 2平台为技术基础，包含许多组件，可简化且规范应用系统的开发与部署，进而提高可移植性、安全性与再用价值。它还提供了对EJB、Servlet、JSP和XML等技术的全面支持，其最终目标是成为一个支持企业级应用开发的体系结构，简化企业解决方案的开发，部署和管理等复杂问题。事实上，J2EE已经成为企业级开发的工业标准和首选平台。J2EE核心是一组技术规范与指南，其中所包含的各类组件、服务架构及技术层次，均有共同的标准及规格，让各种依循J2EE架构的不同平台之间存在良好的兼容性，解决过去企业后端使用的信息产品彼此之间无法兼容、导致企业内部或外部难以互通的窘境。本章通过几个不同难易程度的实用案例程序，覆盖了J2EE在JSP、Servlet、EJB等多项技术方面的应用，有助于读者加深对J2EE诸多特性的理解。

第9章 Web服务与其他

本章内容

- 案例1: 用Servlet生成图像验证码
- 案例2: 获取Java虚拟机的系统属性
- 案例3: 密码生成器
- 案例4: 数据库数据转成XML文件
- 案例5: 网页计数器
- 案例6: Java打印程序
- 案例7: 用SunJCE进行文件的加密和解密
- 本章小结



案例1: 用Servlet生成图像验证码

案例运行效果与操作

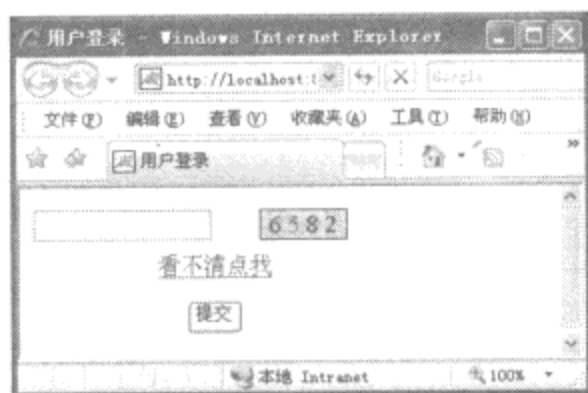


图9-1 验证码输入界面

验证码的目的之一是防范一些恶意的网站下载软件, 这些软件能通过遍历链接而将网站的所有网页下载; 还可以防止用户不经过本网站的页面而使用网站的资源。所以现在很多网站在用户登录或者发布信息时, 都要求用户输入验证码。验证码通常是以一幅图片的形式显示的, 用户按照图片中显示的数字或者字母依次输入, 服务端将用户输入和验证码进行比较, 以判断用户是否经过检验。由于验证码是Web服务器上

生成的随机字符串, 所示自动发布信息的软件无法知道生成的验证码。本案例便是使用Servlet生成这种图像验证码。案例运行后, 界面如图9-1所示。

当输入的验证码不正确时, 提示窗口如图9-2所示。

输入验证码正确时, 如图9-3所示。



图9-2 提示窗口

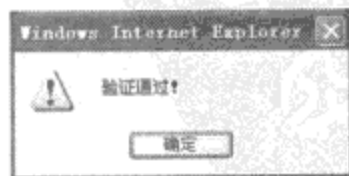


图9-3 验证通过提示窗口

制作要点

1. Servlet的用法。
2. 随机函数Random的使用。
3. HttpSession接口的用法。

步骤详解

1. 配置Web应用服务器，本例使用Tomcat 5.5，并配置Servlet部署。
2. 创建Servlet类。

```
public class ValidImage extends HttpServlet
```

doGet()方法对应于HTTP Get请求。

3. 生成随机类。

```
Random random = new Random();
```

生成随机产生的验证码：

```
String rand=String.valueOf(random.nextInt(10))
```

4. 获取session。

```
session.setAttribute("randcode",sRand)
```

5. 用户登录的JSP文件，用于测试验证码：

```
Login.jsp
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>用户登录</title>
</head>
<body>
<table width="256" border="0" cellpadding="0" cellspacing="0">
  <form action="auth.jsp" method="post" name="input">
    <tr>
      <td width="118" height="22" valign="middle" align="center"><input type="text" name=
"randcode" size="15"></td>
      <td width="138" valign="middle" align="center"></td>
    </tr>
    <tr>
      <td height="36" colspan="2" align="center" valign="middle"><a href= "javascript :loadimage();
"><font class=pt95>看不清点我</font></a></td>
    </tr>
    <tr>
      <td height="36" colspan="2" align="center" valign="middle"><input type="submit" name= "in-
put" value="提交"></td>
    </tr>
  </form>
</table>
</body>
</html>
```

6. 简单的验证JSP文件，用来检查Login.jsp输入的验证码与ValidImage类生成的验证码是否匹配。

```
<%
String rand = (String)session.getAttribute("randcode");
```



```
String input = request.getParameter("randcode");
if(rand.equals(input)){
out.print("<script>alert('验证通过！');</script>");
} else{
out.print("<script>alert('请输入正确的验证码！');location.href='login.jsp';</script>");
}
%>
```

程序源代码与解释

```
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import java.awt.image.*;
import java.util.*;
import javax.imageio.*;
import com.sun.image.codec.jpeg.*;
public class ValidImage extends HttpServlet
{
    private static final String CONTENT_TYPE = "image/jpeg; charset=GB2312";
    //初始化全局变量
    public void init() throws ServletException{ }
    //处理HTTP Get请求
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException
    {
        response.setContentType(CONTENT_TYPE);
        //设置页面不缓存
        HttpSession session=request.getSession();
        response.setHeader("Pragma","No-cache");
        response.setHeader("Cache-Control","no-cache");
        response.setDateHeader("Expires", 0);
        //在内存中创建图像
        int width=60, height=20;
        BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
        //获取图形上下文
        Graphics g = image.getGraphics();
        //设定背景色
        g.setColor(getRandColor(155,254));
        //g.setColor(new Color(255,255,255));
        g.fillRect(0, 0, width, height);
        //设定字体
        g.setFont(new Font("Times New Roman",Font.PLAIN,18));
        //画边框
        //g.setColor(new Color());
        //g.drawRect(0,0,width-1,height-1);
        //随机产生155条干扰线，使图像中的认证码不易被其他程序探测到
        g.setColor(getRandColor(160,220));
        //生成随机类
```



```

Random random = new Random();
for (int i=0;i<155;i++)
{
    int x = random.nextInt(width);
    int y = random.nextInt(height);
    int xl = random.nextInt(12);
    int yl = random.nextInt(12);
    g.drawLine(x,y,x+xl,y+yl);
}
//取随机产生的验证码（4位数字）
String sRand="";
for (int i=0;i<4;i++)
{
    String rand=String.valueOf(random.nextInt(10));
    sRand+=rand;
    //将验证码显示到图像中
    g.setColor(new Color(20+random.nextInt(110),20+random.nextInt(110),20+random.nextInt(110)));
    g.drawString(rand,13*i+6,16);
}
//将验证码存入session
session.setAttribute("randcode",sRand);
//图像生效
g.dispose();
//输出图像到页面
ImageIO.write(image, "JPEG", response.getOutputStream());
}

```



案例2：获取JVM系统属性

案例运行效果与操作

JVM，即Java虚拟机，在机器和编译程序之间加入了一个抽象的虚拟的机器。在任何平台上都提供给编译程序一个共同的接口。编译程序只需要面向虚拟机，生成虚拟机能够理解的代码，然后由解释器来将虚拟机代码转换为特定系统的机器码执行。

本案例是一个可以获得当前JVM属性的类，通过它可以获知正在使用的Java虚拟机的系统属性。在开发环境中，运行或者生成项目后在DOS窗口中执行命令`java -jar systeminfo.jar`，都可以启动程序。运行后，开发环境在输出窗口中出现如图9-4所示的界面。

制作要点

1. Properties类的应用技巧。

在Java应用程序运行时，特别是需要在跨平台工作环境下运行时，需要确定操作系统类型、用户JDK版本和用户工作目录等随工作平台变化的信息，来保证程序正确运行。一般情况下，可以利用JDK提供的系统属性类（Properties）中的方法，快速地获取工作环境信息。

Java Properties类的继承关系如下：

```

java.lang.Object
+ -- java.util.Dictionary
+ -- java.util.Hashtable
+ -- java.util.Properties

```

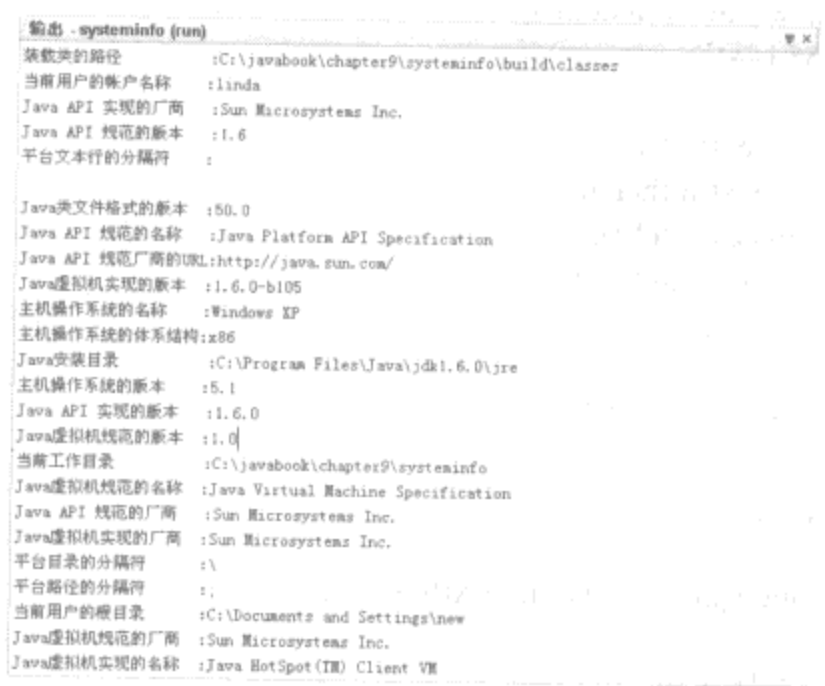



图9-4 运行界面

在应用程序开始执行时，程序首先读取系统的默认属性。如果定义了用户属性文件，则应用程序加载该属性文件。程序运行过程中可以根据执行情况动态地修改属性定义，并在程序结束运行前，保存属性文件。

获取属性的方法：

- (1) `contains(Object value)`、`containsKey(Object key)`：如果给定的参数或属性关键字在属性表中有定义，该方法返回True，否则返回False。
- (2) `getProperty(String key)`、`getProperty(String key, String default)`：根据给定的属性关键字获取关键字值。
- (3) `list(PrintStream s)`、`list(PrintWriter w)`：在输出流中输出属性表内容。
- (4) `size()`：返回当前属性表中定义的属性关键字个数。

设置属性的方法：

- (1) `put(Object key, Object value)`：向属性表中追加属性关键字和关键字的值。
- (2) `remove(Object key)`：从属性表中删除关键字。
- (3) `setProperty(String key, String value)`：调用Hashtable的方法put。

2. System类的getProperties方法的使用。

System类不能被实例化，它包含一些有用的类字段和方法，其中静态方法static Properties getProperties()用来确定当前的系统属性。系统属性是指与用户程序相关的操作系统配置信息以及软件信息。该方法将当前系统属性集合作为Properties对象返回。如果没有当前系统属性集合，则先创建并初始化一个系统属性集合。这个系统属性集合包含的键-值对如表9-1所示。

表9-1 系统属性集合包含的键-值对

键	相关值的描述
java.version	Java运行时环境版本
java.vendor	Java运行时环境供应商
java.vendor.url	Java供应商的URL
java.home	Java安装目录
java.vm.specification.version	Java虚拟机规范版本

(续表)

键	相关值的描述
java.vm.specification.vendor	Java虚拟机规范供应商
java.vm.specification.name	Java虚拟机规范名称
java.vm.version	Java虚拟机实现版本
java.vm.vendor	Java虚拟机实现供应商
java.vm.name	Java虚拟机实现名称
java.specification.version	Java运行时环境规范版本
java.specification.vendor	Java运行时环境规范供应商
java.specification.name	Java运行时环境规范名称
java.class.version	Java类格式版本号
java.class.path	Java类路径
java.library.path	加载库时搜索的路径列表
java.io.tmpdir	默认的临时文件路径
java.compiler	要使用的JIT编译器的名称
java.ext.dirs	一个或多个扩展目录的路径
os.name	操作系统的名称
os.arch	操作系统的架构
os.version	操作系统的版本
file.separator	文件分隔符（在UNIX系统中是“/”）
path.separator	路径分隔符（在UNIX系统中是“:”）
line.separator	行分隔符（在UNIX系统中是“\n”）
user.name	用户的账户名称
user.home	用户的主目录
user.dir	用户的当前工作目录

步骤详解

- 1. 新建基于Java类库的项目systeminfo。
- 2. 装载Java核心包。本案例使用了Java核心包中的Properties、Hashtable、Enumeration等类，添加代码如下：

```
import java.util.Properties;
import java.util.Hashtable;
import java.util.Enumeration;
```

- 3. getSystemProperty方法调用System.getProperties方法来获取当前系统属性集合。

```
static public Properties getSystemProperty(){
    _property=System.getProperties();//调用getProperties方法来获取当前系统属性集合
    return _property;
}
```


4. 主方法首先调用`getSystemProperty`方法获取当前系统属性集合，然后将获取的系统属性集合的内容保存到一个`Hashtable`对象，最后用一个`Enumeration`对象将`Hashtable`对象内容输出。添加下列代码：

```
public static void main(String[] args){
    getSystemProperty();
    Hashtable hashKey;
    hashKey=new Hashtable();    //创建一个Hashtable对象
    //将系统信息的关键字和标题放到Hashtable
    hashKey.put("java.home",           "Java安装目录           ");
    hashKey.put("java.class.path",     "装载类的路径         ");
    hashKey.put("java.specification.version", "Java API规范的版本   ");
    hashKey.put("java.specification.vendor", "Java API规范的厂商   ");
    hashKey.put("java.specification.name", "Java API规范的名称   ");
    hashKey.put("java.version",        "Java API实现的版本   ");
    hashKey.put("java.vendor",         "Java API实现的厂商   ");
    hashKey.put("java.vendor.url",     "Java API规范厂商的URL");
    hashKey.put("java.vm.specification.version", "Java虚拟机规范的版本 ");
    hashKey.put("java.vm.specification.vendor", "Java虚拟机规范的厂商 ");
    hashKey.put("java.vm.specification.name", "Java虚拟机规范的名称 ");
    hashKey.put("java.vm.version",      "Java虚拟机实现的版本 ");
    hashKey.put("java.vm.vendor",       "Java虚拟机实现的厂商 ");
    hashKey.put("java.vm.name",         "Java虚拟机实现的名称 ");
    hashKey.put("java.class.version",   "Java类文件格式的版本 ");
    hashKey.put("os.name",              "主机操作系统的名称   ");
    hashKey.put("os.arch",              "主机操作系统的体系结构");
    hashKey.put("os.version",           "主机操作系统的版本   ");
    hashKey.put("file.separator",       "平台目录的分隔符     ");
    hashKey.put("path.separator",       "平台路径的分隔符     ");
    hashKey.put("line.separator",      "平台文本行的分隔符   ");
    hashKey.put("user.name",            "当前用户的账户名称   ");
    hashKey.put("user.home",            "当前用户的根目录     ");
    hashKey.put("user.dir",             "当前工作目录         ");
    Enumeration myenum; //创建一个Enumeration对象
    String propertyKey;
    myenum=hashKey.keys();
    while(myenum.hasMoreElements()){
        propertyKey=(String)myenum.nextElement();
        System.out.println((String)hashKey.get(propertyKey)+"_"+
                           _property.getProperty(propertyKey));
    }
}
```

程序源代码与解释

```
/* * SystemInfo.java*/
package systeminfo;
import java.util.Properties;
import java.util.Hashtable;
import java.util.Enumeration;
public class SystemInfo{
    //存放JVM获得的系统属性
    static private Properties _property;
```



```

//获得系统属性列表 @return Properties
static public Properties getSystemProperty(){
    _property=System.getProperties();    //调用getProperties方法来获取当前系统属性集合
    return _property;
}
}

```



案例3：密码生成器

案例运行效果与操作

为了安全，很多程序在用户注册时生成一个密码并返回给用户，通常这个密码是随机的。本案例主要讲解如何使用JavaBean生成随机密码。程序运行后，界面如图9-5所示。

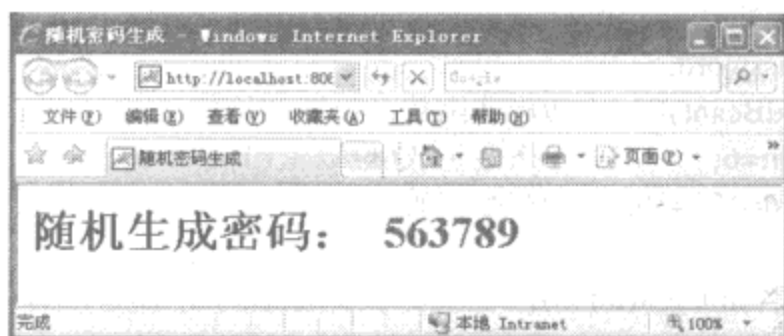


图9-5 运行界面

每次运行，都会得到一个随机产生的不同密码。

制作要点

1. 编写JavaBean。
2. Math对象的random()方法。
3. 调用Bean方法PasswordBean.createPassword()。

步骤详解

1. 建立JavaBean程序PasswordBean.java，结构如同“第8章案例2”的说明。添加方法createPassword()，用于产生6位的随机数，并将其赋给password变量作为密码。

2. 产生随机数，添加代码如下：

```

public void createPassword(){
    double d=Math.random();
    int n=1;
    for(int i=0;i<numDigit;i++){
        n=n*10;
        password=(long)(d*n)+1;
    }
}

```

3. 新建JSP文件，代码如下，用户调用刚建立的Bean文件：

```

<html>
<head>
<title>密码生成器</title>

```



```
</head>
<body>
<jsp:useBean id="pass" scope="session" class=" PasswordBean" />
<jsp:setProperty name="pass" property="numDigit" value="6"/> //设置密码位数
<% PasswordBean.createPassword();%> //执行脚本程序，生成密码
随机生成密码: <font color="#663366">
<jsp:getProperty name="pass" property="password"/>/font><br> //取出并显示密码
</body>
</html>
```

程序源代码与解释

```
//PasswordBean.java
import java.util.*;           //载入用到的Java包
import java.io.Serializable;
public class PasswordBean implements Serializable{
    private int numDigit;
    private long password;
    public PasswordBean(){      //构造函数，设置初始值
        numDigit=6;           //指定生成6位数密码
        password=123456;      //初始密码
    }
    public void setNumDigit(int n){
        if(n<6)
            numDigit=n;
        else
            numDigit=6;
    }
    public void createPassword(){ //利用随机数产生密码
        double d=Math.random();
        int n=1;
        for(int i=0;i<numDigit;i++)
            n=n*10;
        password=(long)(d*n)+1;
    }
    public int getNumDigit() {
        return numDigit;
    }
    public long getPassword() {
        return password;
    }
}
```



案例4：数据库数据转成XML文件

案例运行效果与操作

本案例读取数据库中的数据，将其输出转换成XML格式文件。程序运行后，生成的flight.xml文件如图9-6所示，数据来源是“第10章案例4”的数据库航班信息的表单flight。



图9-6 生成XML文件

制作要点

1. getConnection()的用法。
2. Vector数组的应用。
3. XML解析。

XML作为全球通用的结构化语言，越来越受人们青睐。对XML文件的读写是编程中经常遇到的操作，XML语义比较严格，各种平台提供不同的解析的工具。用Java解析XML文档，最常用的有两种方法：使用基于事件的XML简单API（Simple API for XML，称为SAX）和基于树和节点的文档对象模型（Document Object Module，称为DOM）。有4种技术可选择：DOM、SAX、JDOM、DOM4J。它们各有千秋，应用场合也不尽相同，其中，DOM是许多其他与XML相关的标准的基础，广泛应用于多种编程语言；JDOM仅应用于Java；SAX有特定的解析方式；许多开源项目中大量采用DOM4J，但可移植性较差。本例采用JDOM。

JDOM是在Apache许可证变体下发布的开放源码，目的是成为Java特定文档模型，它简化与XML的交互并且比使用DOM实现更快。JDOM仅使用具体类而不使用接口。这在某些方面简化了API，API大量使用了Collections类，方便了那些已经熟悉这些类的Java开发者的使用。

JDOM自身不包含解析器，它通常使用SAX2解析器来解析和验证输入XML文档，它包含一些转换器以将JDOM表示输出成SAX2事件流、DOM模型或XML文本文档。

步骤详解

1. 建立数据库连接，读取数据：

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");//数据驱动
String url="jdbc:odbc:dsStudent";//数据源
Connection con=DriverManager.getConnection(url,"sa","");
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from flight");
```

2. 通过javax.xml.parsers.DocumentBuilderFactory实例的静态方法newDocumentBuilder()得到DOM解析器，并得到一个Document：


```

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.newDocument();

```

3. 获取输入流，即数据库的数据：

```

while(rs.next()) {
    Element row = doc.createElement("row");
    root.appendChild(row);
    for(int i =0; i < columns.size(); i++) {
        String temp = rs.getString(i+1);
        if(temp == null) {
            temp = "";
        }
        Element colum = doc.createElement(columns.get(i).toString());
        row.appendChild(colum);
        Text t = doc.createTextNode(temp);
        colum.appendChild(t);
    }
}

```

4. 实例化一个转换工厂：

```

DOMSource source = new DOMSource(doc);
StreamResult result = new StreamResult(xmlFile);
TransformerFactory tfactory = TransformerFactory.newInstance();
Transformer trans = tfactory.newTransformer();

```

程序源代码与解释

```

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamSource;
import org.w3c.dom.*;
import java.sql.*;
import java.util.*;
import javax.xml.transform.stream.StreamResult;
public class DBxml {
    public static void convertDBToXml (ResultSet rs, String xmlFile) {
        try {
            ResultSetMetaData rsmd = rs.getMetaData();
            Vector columns = new Vector();
            for(int i =1; i < rsmd.getColumnCount(); i++) {
                columns.addElement(rsmd.getColumnName(i));
            }
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document doc = builder.newDocument();
            Element root = doc.createElement("dataTable");
            doc.appendChild(root);
            while(rs.next()) {
                Element row = doc.createElement("row");
                root.appendChild(row);
            }
        }
    }
}

```



```

        for(int i =0; i < columns.size(); i++) {
            String temp = rs.getString(i+1);
            if(temp == null) {
                temp = "";
            }
            Element column = doc.createElement(columns.get(i).toString());
            row.appendChild(column);
            Text t = doc.createTextNode(temp);
            column.appendChild(t);
        }
    }
    DOMSource source = new DOMSource(doc);
    StreamResult result = new StreamResult(xmlFile);
    TransformerFactory tfactory = TransformerFactory.newInstance();
    Transformer trans = tfactory.newTransformer();
    trans.transform(source,result);
    System.out.println("complete");
} catch (Exception ex) {
    ex.printStackTrace();
}
}

public static void main(String[] args) {
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //数据驱动
        String url="jdbc:odbc:dsStudent"; //数据源
        Connection con=DriverManager.getConnection(url,"sa","");
        Statement stmt=con.createStatement();
        ResultSet rs=stmt.executeQuery("select * from flight");
        convertDBToXml(rs,"D:\\flight.xml");
        con.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
}
}

```



案例5: 网页计数器

案例运行效果与操作

网站浏览量的统计一般基于用户访问的次数,很多网页都内嵌一段统计访问量的代码。本案例主要讲解如何使用Bean实现网页计数功能,用户每次运行程序或刷新网页,计数器都会加1。程序counter.jsp运行后,显示界面如图9-7所示。

关闭浏览器,重新运行程序,或单击“刷新”按钮,则界面如图9-8所示。

制作要点

1. 在Bean程序中建立方法。

ReadFile(String filePath): 读取指定文件内容。

WriteFile(String filePath): 将内容写入指定文件。

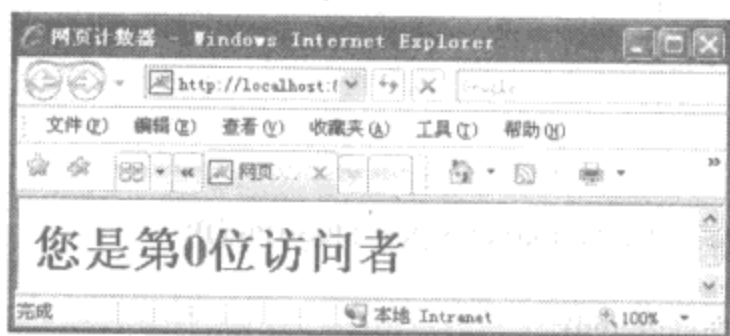


图9-7 运行界面



图9-8 再次运行界面

2. BufferedReader对象的文件操作方法。

步骤详解

1. 建立Bean文件counter.java，设置初始值，建立ReadFile()方法来读取指定文件内容，创建BufferedReader对象，使用readLine()方法读入数据，进行错误处理，然后返回读取内容。
2. 建立WriteFile()方法来将读取的数据加1，创建PrintWriter对象，使用println()方法将加1的数据写入到文件中。
3. 建立文件counter.jsp， counter.txt文件初始值设为零。

```
<%@ page contentType="text/html;charset=gb2312"%>
<HTML>
<HEAD>
  <TITLE>Bean计数器程序</TITLE>
</HEAD>
<BODY>
  <jsp:useBean id="counter" class="counter" scope="request"/>
  <%
    //调用counter对象的ReadFile方法来读取文件count.txt中的计数
    String cont=counter.ReadFile("c:/count.txt");
    //调用counter对象的WriteFile方法来将计数器加1后写入到文件count.txt中
    counter.WriteFile("c:/count.txt",cont);
  %>
  您是第<font color="red"><%=cont%></font>位访问者
</BODY>
</HTML>
```

4. 读取指定文件内容，添加代码如下：

```
public String ReadFile(String filePath) throws FileNotFoundException{
    path = filePath;
    //创建新的BufferedReader对象
    file = new BufferedReader(new FileReader(path));
    String returnStr =null;
    try{
        //读取一行数据并保存到currentRecord变量中
        currentRecord = file.readLine();
    }
    catch (IOException e){           //错误处理
        System.out.println("读取数据错误.");
    }
    if (currentRecord == null)      //如果文件为空
        returnStr = "没有任何记录";
}
```



```

        else{    //文件不为空
            returnStr =currentRecord;
        }
    }
    return returnStr;    //返回读取文件的数据
}

```

5. 写入文件，代码如下：

```

public void WriteFile(String filePath,String counter) throws FileNotFoundException{
    path = filePath;
    //将counter转换为int类型并加1
    int Writestr = Integer.parseInt(counter)+1;
    try {    //创建PrintWriter对象，用于写入数据到文件中
        PrintWriter pw = new PrintWriter(new FileOutputStream(filePath));
        pw.println(Writestr);    //用文本格式打印整数Writestr
        pw.close();    //清除PrintWriter对象
    }
    catch(IOException e) {    //错误处理
        System.out.println("写入文件错误"+e.getMessage());
    }
}

```

程序源代码与解释

```

//counter.java Bean程序
import java.io.*;
public class counter extends Object {
    private String currentRecord = null;    //保存文本的变量
    private BufferedReader file;    //BufferedReader对象，用于读取文件数据
    private String path;    //文件完整路径名
    public counter() {    //空构造函数
    }
    //ReadFile方法用来读取文件filePath中的数据，并返回这个数据
    public String ReadFile(String filePath) throws FileNotFoundException{
        //参见步骤详解4}
    //WriteFile方法用来将数据counter加1后写入到文本文件filePath中，实现计数增长
    public void WriteFile(String filePath,String counter) throws FileNotFoundException{
        //参见步骤详解5
    }
}

```



案例6: Java打印程序

案例运行效果与操作

在实际工作中，经常需要实现打印功能。从JDK 1.4开始，Java提供了一套完整的“Java打印服务API（Java Print Service API）”，利用它可以实现大部分实际应用需求，包括打印文字、图形、文件及打印预览等。本案例讲述如何利用“Java打印服务API”开发打印程序。程序运行后，界面如图9-9所示。

窗口主体是一个文本区，用户可以在此输入要打印的文本。下方是一排按钮，分别用于完成不同的打印工作。

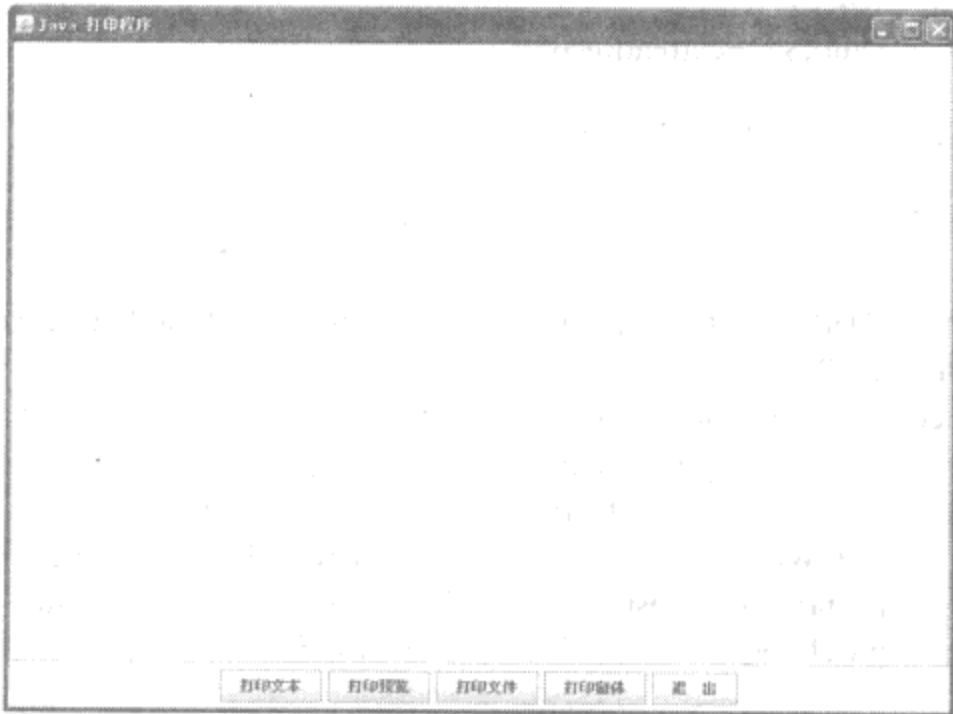


图9-9 运行界面

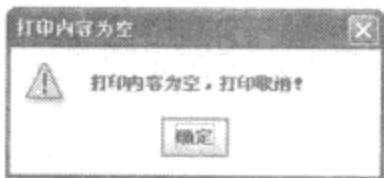


图9-10 打印内容为空时的界面

若不输入文本，直接单击“打印文本”按钮，会提示打印内容为空，界面如图9-10所示。

在文本区输入要打印的文本，或者通过剪贴板粘贴文本，如图9-11所示，然后单击“打印文本”按钮即可开始打印。

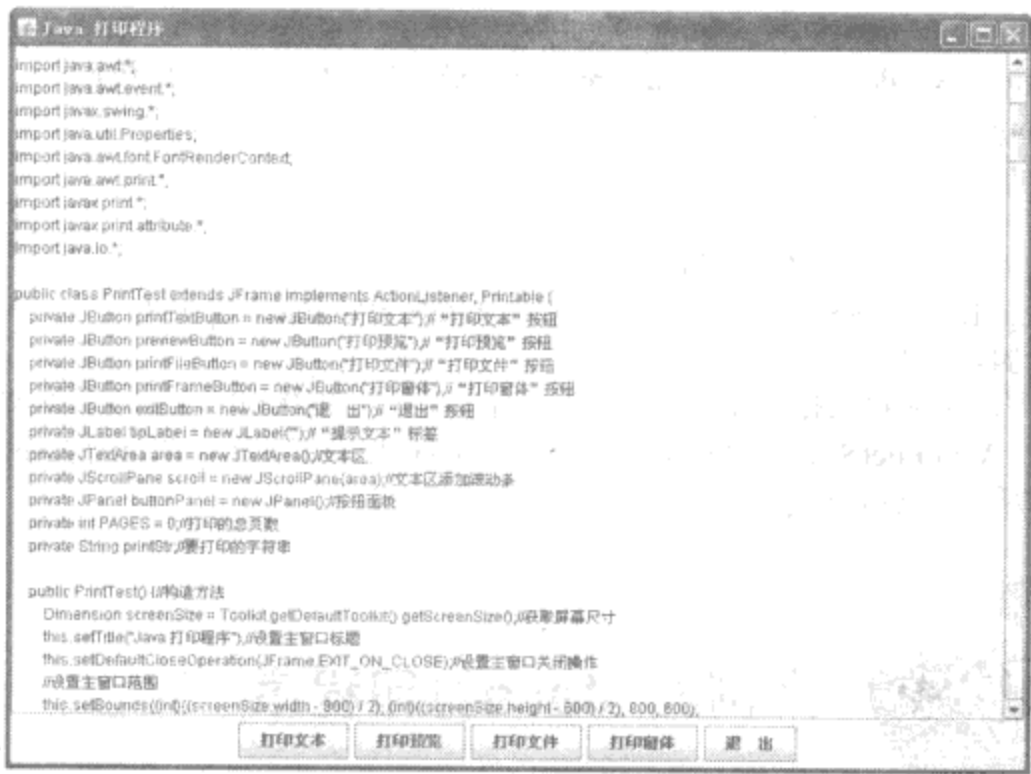


图9-11 在文本区输入或粘贴要打印的文本

在开始打印之前，还可以单击“打印预览”按钮，对所打印内容的打印效果进行预览，此时界面如图9-12所示。通过单击“下一页”按钮和“上一页”按钮，可以对打印内容翻页查看，单击“关闭”按钮则可退出预览界面。

在主界面单击“打印文件”按钮，会弹出“打开”对话框，要求用户选择需要进行打印的文件，默认打印Java文件，如图9-13所示。



图9-12 打印预览

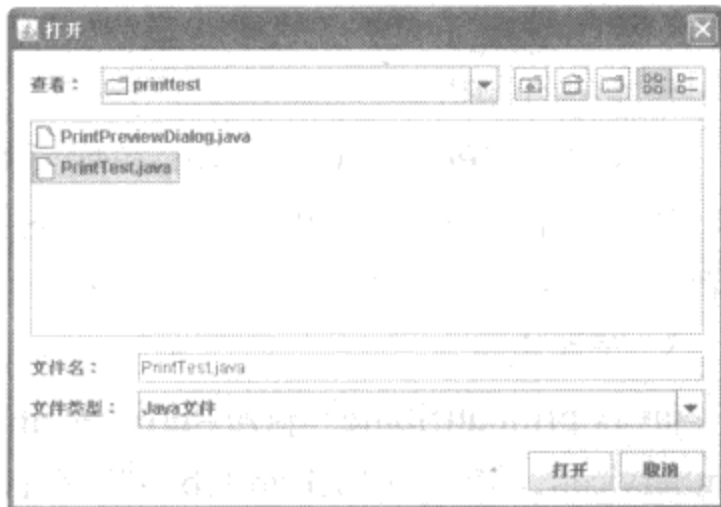


图9-13 “打开”对话框

选择文件后，则弹出“打印”对话框，如图9-14所示。此处的设置与我们平常所用的打印设置类似，在此不再赘述。单击“打印”按钮即可开始文件的打印。

在主界面单击“打印窗体”按钮，会弹出系统自带的“打印”对话框，如图9-15所示。单击“确定”按钮即可将当前窗体打印出来。



图9-14 Java中的“打印”对话框

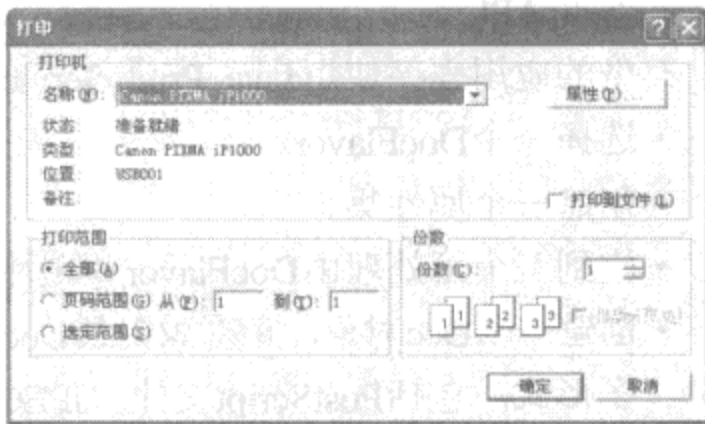


图9-15 系统自带的“打印”对话框

制作要点

1. 布局管理器的应用技巧。
2. FileFilter类的使用。
3. Java打印API的应用技巧。

Java的打印API主要存在于`javax.print`包及其相应的子包`javax.print.event`和`javax.print.attribute`中。其中，`javax.print`包中主要包含打印服务的相关类，`javax.print.event`包中则包含打印事件的相关定义，`javax.print.attribute`包中则包括打印服务的可用属性列表等。软件包`javax.print`为Java Print Service API提供了主要类和接口。Java Print Service API允许客户端和服务端应用程序具备如下功能。

- 根据其性能发现并选择PrintService。
- 指定打印数据的格式。
- 向支持所打印文档类型的服务提交PrintJob。

4. PrintService发现。

应用程序可调用抽象类PrintServiceLookup的静态方法来查找PrintService，这些PrintService具有满足应用程序打印要求的功能。例如，要打印双面文档，应用程序首先需要找到具有双面打印功能的打印机。

JDK包括的PrintServiceLookup实现可查找标准的平台打印机。要查找其他类型的打印机（如IPP打印机或JINI打印机），PrintService提供者可写入PrintServiceLookup实现。PrintService提供者可使用SPI JAR文件规范动态地安装这些PrintServiceLookup实现。

5. 属性定义。

javax.print.attribute和javax.print.attribute.standard包定义了打印属性，这些属性描述了PrintService的功能、指定PrintJob的要求并跟踪PrintJob的进度。

javax.print.attribute包描述了属性类型和属性分类方法。javax.print.attribute.standard包枚举了API所支持的所有标准属性。javax.print.attribute.standard包中指定的属性有分辨率、份数、介质大小、作业优先级和页面范围。

6. 文档类型规范。

DocFlavor类表示了打印数据的格式，如JPEG或PostScript。DocFlavor对象由MIME类型（描述了格式）和文档表示形式类名（指示如何将文档发送到打印机或输出流）所组成。应用程序使用DocFlavor和属性集来查找某些打印机，这些打印机可打印由DocFlavor所指定的文档类型且具有属性集所指定的功能。

7. 使用API。

典型的应用程序使用Java Print Service API执行以下步骤来处理打印请求。

- 选择一个DocFlavor。
- 创建一个属性集。
- 找到一个可处理由DocFlavor和属性集所指定的打印请求的PrintService。
- 创建一个Doc对象，该对象封装DocFlavor和实际的打印数据，这些打印数据可采用很多形式，包括PostScript文件、JPEG图像、URL或纯文本。
- 从PrintService获得一个由DocPrintJob表示的PrintJob。
- 调用PrintJob的print方法。

8. 程序设计。

打印文本：设定每页最多打印54行，首先需要实现Printable接口，然后按照每页最多54行的格式计算共需要打印多少页，当“打印文本”按钮被单击时，执行相应的打印动作。打印文本的具体操作可通过Graphics2D的drawString方法来实现。

打印预览：正常情况下，print方法将页面环境绘制到一个打印机图形环境上，从而实现打印。而事实上，print方法并不能真正产生打印页面，它只是将待打印内容绘制到图形环境上。所以，可以忽略掉屏幕图形环境，经过适当的缩放比例，使整个打印页容纳在一个屏幕矩形里，从而实现精确的打印预览。

打印文件：JDK的打印服务API提供了一整套的打印文件流的类和方法，利用它们，可以非常方便、快捷地实现各式各样不同类型文件的打印功能。

打印窗体：在Java的Component类及其派生类中都提供了print和printAll方法，只要设置好属性就可以直接调用这两个方法，从而实现对组件及图形的打印。

步骤详解

1. 新建Java应用项目printtest。
2. 设计主界面。
3. 为PrintTest类编写构造方法，添加代码如下：

```
public PrintTest() { //构造方法
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize(); //获取屏幕尺寸
    this.setTitle("Java打印程序"); //设置主窗口标题
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //设置主窗口关闭操作
    this.setBounds((int)((screenSize.width - 800) / 2), (int)((screenSize.height - 600) / 2), 800,
600); //设置主窗口范围
    initLayout(); //调用initLayout方法进行主窗口界面初始化
}
```

4. 实现Printable接口的代码：

```
public int print(Graphics g, PageFormat pf, int page) throws PrinterException {
    Graphics2D g2 = (Graphics2D)g; //转换成Graphics2D
    g2.setPaint(Color.black); //设置打印颜色为黑色
    if (page >= PAGES) //当打印页号大于需要打印的总页数时，打印工作结束
        return Printable.NO_SUCH_PAGE;
    g2.translate(pf.getImageableX(), pf.getImageableY()); //转换坐标，确定打印边界
    drawCurrentPageText(g2, pf, page); //打印当前页文本
    return Printable.PAGE_EXISTS; //存在打印页时，继续打印工作
}
/*打印指定页号的具体文本内容*/
private void drawCurrentPageText(Graphics2D g2, PageFormat pf, int page) {
    Font f = area.getFont(); //获取打印字体
    String s = getDrawText(printStr)[page]; //获取当前页的待打印文本内容
    String drawText;
    float ascent = 16; //给定字符点阵
    int k, i = f.getSize(), lines = 0;
    while(s.length() > 0 && lines < 54) { //每页限定在54行以内
        k = s.indexOf('\n'); //获取每一个回车符的位置
        if (k != -1) { //存在回车符
            lines += 1; //计算行数
            drawText = s.substring(0, k); //获取每一行文本
            g2.drawString(drawText, 0, ascent); //具体打印每一行文本，同时走纸移位
            if (s.substring(k + 1).length() > 0) {
                s = s.substring(k + 1); //截取尚未打印的文本
                ascent += i;
            }
        }
    }
}
```

5. 为PrintTest类编写getPagesCount方法，该方法计算需要打印的总页数：

```
public int getPagesCount(String curStr) { //计算需要打印的总页数
    while(str.length() > 0) { //文本尚未计算完毕
        position = str.indexOf('\n'); //计算回车符的位置
        count += 1; //统计行数
        if (position != -1)
            str = str.substring(position + 1); //截取尚未计算的文本
        else
            str = ""; //文本已计算完毕
    }
    if (count > 0)
```



```

        page = count / 54 + 1;        //以总行数除以54获取总页数
        return page; //返回需打印的总页数
    }

```

6. 为PrintTest类编写printTextAction方法，该方法用来处理“打印文本”按钮事件：

```

private void printTextAction() {        //“打印文本”按钮事件处理
    printStr = area.getText().trim();    //获取需要打印的目标文本
    if (printStr != null && printStr.length() > 0) {    //当打印内容不为空时
        PAGES = getPagesCount(printStr);    //获取打印总页数
        //指定打印输出格式
        DocFlavor flavor = DocFlavor.SERVICE_FORMATTED.PRINTABLE;
    }
    ...
}

```

7. 为PrintPreviewDialog类编写内部类PreviewCanvas，该内部类用来显示“打印预览”对话框以及预览的内容：

```

class PreviewCanvas extends JPanel {    //预览面板类
    private String printStr;
    private int currentPage = 0;
    private PrintTest preview;
    public PreviewCanvas(PrintTest pt, String str) {    //构造方法
        printStr = str;
        preview = pt;
    }
    /*将待打印内容按比例绘制到屏幕*/
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
    }
    ...

    /*预览指定的页面*/
    public void viewPage(int pos) {
        int newPage = currentPage + pos;
        //指定页面在实际的范围内
        if (0 <= newPage && newPage < preview.getPagesCount(printStr)) {
            currentPage = newPage;    //将指定页面赋值为当前页
            repaint();
        }
    }
    ...
}

```

程序源代码与解释

```

/* * PrintTest.java*/
public class PrintTest extends JFrame implements ActionListener, Printable {
    /*将打印目标文本按页存放为字符串数组*/
    public String[] getDrawText(String s) {
        String[] drawText = new String[PAGES];    //根据页数初始化数组
        for (int i = 0; i < PAGES; i++)
            drawText[i] = "";    //数组元素初始化为空字符串
        int k, suffix = 0, lines = 0;
        while(s.length() > 0) {
            if(lines < 54) {    //不够一页时
                k = s.indexOf("\n");
                if (k != -1) {    //存在回车符
                    lines += 1;    //行数累加
                    //计算该页的具体文本内容，存放相应下标的数组元素
                    drawText[suffix] = drawText[suffix] + s.substring(0, k + 1);
                    if (s.substring(k + 1).length() > 0)

```



```

        s = s.substring(k + 1);
    }
    else {
        lines += 1;    //行数累加
        //将文本内容存放到相应的数组元素
        drawText[suffix] = drawText[suffix] + s;
        s = "";
    }
}
else { //已满一页时
    lines = 0;    //行数统计清零
    suffix++;    //数组下标加1
}
}
return drawText;
}
private void previewAction() {    //“打印预览”按钮事件处理
    printStr = area.getText().trim();
    PAGES = getPagesCount(printStr);
    //显示“打印预览”对话框
    (new PrintPreviewDialog(this, "打印预览", true, this, printStr)).setVisible(true);
}
/*打印指定的文件*/
private void printFileAction() {    //“打印文件”按钮事件处理
    //构造一个文件选择器，默认为当前目录
    JFileChooser fileChooser = new JFileChooser(System.getProperties().getProperty
("user.dir"));

    fileChooser.setFileFilter(new JavaFilter());    //设置文件过滤器
    int state = fileChooser.showOpenDialog(this);    //弹出“文件”对话框
    if (state == fileChooser.APPROVE_OPTION) {    //如果用户选定了文件
        File file = fileChooser.getSelectedFile();    //获取选择的文件
        //构建打印请求属性集
        PrintRequestAttributeSet pras = new HashPrintRequestAttributeSet();
        //设置打印格式，因为未确定文件类型，这里选择AUTODENSE
        DocFlavor flavor = DocFlavor.INPUT_STREAM.AUTODENSE;
        //查找所有的可用打印服务
        PrintService printService[] = PrintServiceLookup.lookupPrintServices(flavor, pras);
        //定位默认的打印服务
        PrintService defaultService = PrintServiceLookup.lookupDefaultPrintService();
        //显示“打印”对话框
        PrintService service = ServiceUI.printDialog(null, 200, 200, printService ,
defaultService, flavor, pras);
        if (service != null) {
            try {
                DocPrintJob job = service.createPrintJob();    //创建打印作业
                FileInputStream fis = new FileInputStream(file);    //构造待打印的文件流
                DocAttributeSet das = new HashDocAttributeSet();
                Doc doc = new SimpleDoc(fis, flavor, das);    //建立打印文件格式
                job.print(doc, pras);    //进行文件的打印
            }
            catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
}
}

```




案例7：用SunJCE进行文件的加密和解密

案例运行效果与操作

安全的范围很广，因为Java平台是开放的，所以在网络环境中安全对于Java开发者来说尤其重要。本案例使用Sun公司提供的JCE应用程序编程接口（API）进行文件的加密和解密，能够对各种文档进行加密保存，以及对加密文档进行解密显示。加密方法可以应用到代码保护、文件传输等方面。程序运行后，界面如图9-16所示。

在“文件名”文本框中输入要保存的文件路径和文件名（本案例为c:\test.txt），在“密码”文本框中输入设定的密码（本案例为test），在“文件内容”编辑器面板中输入未加密的文档内容，如图9-17所示。



图9-16 运行界面



图9-17 输入未加密的文档内容

文档内容输入完毕后，单击下方的“加密并写入文件”按钮，则将所有输入信息进行加密处理，并将加密后的内容输出到“文件名”文本框指定的文件中（本案例为c:\test.txt）。此时“文件内容”编辑器面板中显示加密后的内容，如图9-18所示。

此时打开保存的文件（本案例为c:\test.txt），会发现其内容已经过加密处理，文件内容全是乱码，并且与本案例“文件内容”编辑器面板中显示的内容一致，如图9-19所示。

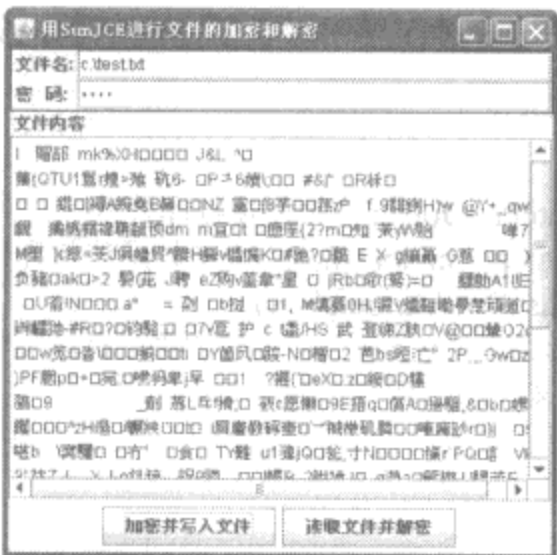


图9-18 加密处理后的文件



图9-19 保存文件的内容

文件解密的过程与加密类似，也是在“文件名”文本框中输入要解密的文件路径和文件名（本案例为刚才加密的c:\test.txt），在“密码”文本框中输入设定的密码（本案例为test），然后单击下方的“读取文件并解密”按钮，则将加密的文件进行解密处理，并将解密后的内容显示在“文件内容”编辑器面板中，如图9-20所示。

制作要点

1. JFrame的使用。
2. SunJCE的应用技巧。

Java体系结构提供了三类主要的安全API：Java认证和授权服务（Java Authentication and Authorization Service, JAAS）、Java安全套接字扩展（Java Secure Socket Extension, JSSE）和Java加密扩展（Java Cryptography Extension, JCE）。其中，JCE是Sun公司的加密服务软件，包含了加密和密钥生成功能。JCE是JCA（Java Cryptography Architecture）的一种扩展。

Java加密扩展包JCE提供的是基于密钥的加密，它通过javax.crypto.Cipher类来实现数据加密和解密，加密和解密的对象可以是程序中的数组对象，也可以是通过Java流接口读出或者写入的数据。使用Cipher类加密可以选择多种加密算法、加密模式和填补机制。DES是很多机构组织采用的数据加密标准；而多重DES使用多个DES密码进行DES加密，加大了攻击的难度，但是也增加了加密和解密过程所花费的时间；PBEWithMD5AndDES主要是计算散列，然后对散列进行DES加密，来实现签名认证；RSA算法是1978年公布的一种分组加密算法，也是现在应用得最广泛的公钥密钥算法；Blowfish是由Bruce Schneier公布的一种加密算法，没有申请专利，并且公布了实现的代码，它适合不需要经常更换密钥的情况。JCE没有规定具体的加密算法，但提供了一个框架，加密算法的具体实现可以让服务提供者加入。除了JCE框架之外，JCE软件包还包含了SunJCE服务提供者，其中包括许多有用的加密算法，比如DES、多重DES、PBEWithMD5AndDES、RSA和Blowfish等。本案例使用PBEWithMD5And-DES算法。

加密/解密的步骤如下。

- （1）利用密码生成一个安全密钥（key）：在加密或解密任何数据之前需要有一个密钥。密钥是随同被加密的应用一起发布的一小段数据。
- （2）调用getInstance方法产生一个Cipher对象。
- （3）调用init方法设定加密或解密。
- （4）进行加密/解密。

步骤详解

1. 装入JCE包：

```
import com.sun.crypto.provider.SunJCE;
```

JCE（Java Cryptography Extension）是Java安全体系的重要组成部分。它提供了加密、密钥产生、密钥协议、验证算法的开发基础，以及常用加密算法的实现方案。JCE共有3个包，

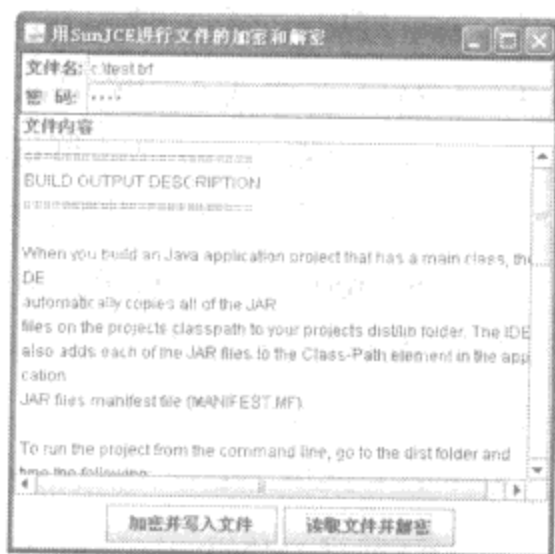


图9-20 解密处理文件

26个类、4个接口和4个异常。其中，`Javax.crypto`包是JCE的核心，提供其中的14个类、1个接口和4个异常，支持基本加密算法。重要的类如下。

- (1) `SecretKey`: 接口扩展`java.security.Key`，提供对称和秘密密钥的加密支持。
- (2) `Cipher`类: 加密引擎，提供了一个实现数据加密和解密的接口。
- (3) `SecretKeyFactory`类: 在秘密密钥与密钥规范之间进行转换的密钥工厂。
- (4) `KeyGenerator`类: 密钥产生器。

2. 程序界面设置初始化。

设置安全提供者:

```
Security.addProvider( new SunJCE() );
```

主窗口初始化:

```
setSize( new Dimension( 400, 400 ) );  
setTitle( "用SunJCE进行文件的加密和解密" );
```

主界面顶部面板:

```
JPanel topPanel = new JPanel();  
topPanel.setBorder( BorderFactory.createLineBorder( Color.black ) );  
topPanel.setLayout( new BorderLayout() );  
...
```

主窗口:

```
JPanel contentPane = ( JPanel ) this.getContentPane();  
contentPane.setLayout( new BorderLayout() );  
contentPane.add( topPanel, BorderLayout.NORTH );  
contentPane.add( middlePanel, BorderLayout.CENTER );  
contentPane.add( bottomPanel, BorderLayout.SOUTH );  
setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
```

界面设计在应用程序与开发中是最常遇到的问题，在前面的案例中已经讲过不少设计方法，这里不再赘述。

3. 从原始密匙数据创建密码（`PBEKeySpec`对象）:

```
PBEKeySpec keySpec = new PBEKeySpec( password.toCharArray() );
```

`PBEKeySpec()`带有一个密码的构造方法。可随同基于密码的加密法（PBE）使用的供用户选择的密码。可以将密码视为某种原始密钥内容，由此使用加密机制导出一个密钥。不同的PBE机制可能对每一个密码字符使用不同的位数。例如，在PKCS #5中定义的PBE机制仅仅关注每一个字符的低8位，而PKCS #12关注每一个字符的全16位。通过创建一个合适的密钥工厂实例来将密码转换成为一个PBE密钥。例如，PKCS #5的密钥工厂仅根据每个密码字符的低8位来构造PBE密钥，而PKCS #12的密钥工厂将使用每个字符的全16位。还要注意，此类以`char`数组而不是`String`对象的形式存储密码（这可能更具逻辑性），因为`String`类是不可变的，当不再需要存储在其中的密码时没有任何途径来重写其内部值。因此，此类需要以`char`数组作为密码，以便在完成后进行重写。

4. 加密过程：创建密钥和`Cipher`对象，建立密钥工厂。

获取一个密匙工厂实例:


```
SecretKeyFactory keyFactory = SecretKeyFactory.getInstance( "PBEWithMD5AndDES" );
```

SecretKeyFactory类是在秘密密钥与密钥规范之间进行转换的密钥工厂。此方法返回转换指定算法的秘密密钥的**SecretKeyFactory**对象，本案例的算法是**PBEWithMD5AndDES**，这种加密方法是基于口令的。它用到了口令、盐值、迭代计数以及**MD5**消息摘要，以生成**DES**秘密密钥，此密钥将用于**DES**加密和解密。其中，口令是一个字符串，需要用户在系统界面中输入；盐值（salt）认为是随机数种子，表现为一个字节数组；迭代计数以及**MD5**消息摘要在程序中生成并制定。

生成加密密钥：

```
SecretKey secretKey = keyFactory.generateSecret( keySpec );
```

指定加密参数：

```
PBEParameterSpec parameterSpec = new PBEParameterSpec( salt, iterationCount );
```

获取**Cipher**对象实例：

```
cipher = Cipher.getInstance( "PBEWithMD5AndDES" );
```

在加密模式中用密钥初始化**Cipher**对象（加密）：

```
cipher.init( Cipher.ENCRYPT_MODE, secretKey, parameterSpec );
```

5. 将加密后的内容保存到指定文件。

创建文件输出流：

```
fileOutputStream = new FileOutputStream( file );
```

创建**CipherOutputStream**对象：

```
CipherOutputStream out = new CipherOutputStream( fileOutputStream, cipher );
```

写入文件：

```
out.write( outputArray );  
out.flush();
```

创建**Vector**对象以存储读取的文件内容：

```
Vector fileBytes = new Vector();
```

读取文件：

```
while ( in.available() > 0 ) {  
    contents = ( byte )in.read();  
    fileBytes.add( new Byte( contents ) );  
}
```

将**Vector**对象内容存储到字节数组：

```
byte[] encryptedText = new byte[ fileBytes.size() ];  
for ( int i = 0; i < fileBytes.size(); i++ ) {  
    encryptedText[ i ] = ( ( Byte ) fileBytes.elementAt( i ) ).byteValue();  
}
```

更新“文件内容”编辑器面板中的内容：


```
fileContentsEditorPane.setText( new String( encryptedText ) );
```

6. 处理异常，注意这里异常的种类比较多。

异常：如果没有任何Provider支持指定算法的SecretKeyFactorySpi实现。

```
catch ( NoSuchAlgorithmException exception ) {}
```

异常：如果给定密钥规范不适合生成秘密密钥的秘密密钥工厂。

```
catch ( InvalidKeySpecException exception ) {}
```

异常：如果此秘密密钥工厂无法处理给定的密钥。

```
catch ( InvalidKeyException exception ) {}
```

异常：当请求特定填充机制但该环境中未提供时抛出。

```
catch ( NoSuchPaddingException exception ) {}
```

异常：无效或不合适的算法参数的异常。

```
catch ( InvalidAlgorithmParameterException exception ) {}
```

异常：不支持字符编码。

```
catch ( UnsupportedEncodingException exception ) {}
```

7. 为FrameEncipherDecipher类添加readFromFileAndDecrypt方法，该方法读取指定加密文件的内容并解密后在“文件内容”编辑器面板中显示出来。该方法代码与encryptAndWriteToFile方法基本一致，只是在用密钥初始化Cipher对象时使用了解密模式：

```
cipher.init( Cipher.DECRYPT_MODE, secretKey, parameterSpec );
```

程序源代码与解释

```
/* * FrameEncipherDecipher.java */
/** 创建一个Frame,可以创建一个文件,对文件进行加密 */
import com.sun.crypto.provider.SunJCE;          //装载JCE包
//装入Java扩展包
import javax.swing.*;
import javax.crypto.*;
import javax.crypto.spec.*;
public class FrameEncipherDecipher extends JFrame {
    //用基于密码的加密/解密算法获取数据
    private static final byte[] salt = {
        ( byte )0xf5, ( byte )0x33, ( byte )0x01, ( byte )0x2a,
        ( byte )0xb2, ( byte )0xcc, ( byte )0xe4, ( byte )0x7f
    };
    private int iterationCount = 100;    //迭代次数
    //用户输入组件
    private JTextField passwordTextField;
    private JTextField fileNameTextField;
    private JEditorPane fileContentsEditorPane;
    //构造方法
    public FrameEncipherDecipher() {
        //设置安全提供者
```



```

        Security.addProvider( new SunJCE() );
        //代码略，下同。设置界面及按钮、窗口初始化
    }
    //获取“文件内容”编辑器面板中的内容并加密
    private void encryptAndWriteToFile() {
        //获取用户输入
        String originalText = fileContentsEditorPane.getText();
        String password = passwordTextField.getText();
        String fileName = fileNameTextField.getText();
        // 创建密钥和Cipher对象
        Cipher cipher = null;
        try {
            //加密过程， 参见步骤详解3和步骤详解4
        }
        //异常处理，参见步骤详解6
        byte[] outputArray = null;    //创建字节数组
        try { outputArray = originalText.getBytes( "ISO-8859-1" );//GB2312 }
        catch ( UnsupportedEncodingException exception ) { //处理UnsupportedEncodingException

            exception.printStackTrace();
            System.exit( 1 );
        }
        //创建FileOutputStream对象
        File file = new File( fileName );
        //将加密后的内容保存至指定文件，参见步骤详解5
    }
    private void readFromFileAndDecrypt() { //读取文件并解密
        Vector fileBytes = new Vector();    //创建Vector对象
        //获取用户输入
        String password = passwordTextField.getText();
        String fileName = fileNameTextField.getText();
        Cipher cipher = null;    //创建Cipher对象
        try {
            //从原始密匙数据创建密码（PBEKeySpec对象）
            PBEKeySpec keySpec =new PBEKeySpec( password.toCharArray() );
            SecretKeyFactory keyFactory =
                SecretKeyFactory.getInstance( "PBEWithMD5AndDES" );    //获取一个密匙工厂实

            SecretKey secretKey = keyFactory.generateSecret( keySpec );    //生成加密密钥
            //指定加密参数
            PBEPParameterSpec parameterSpec = new PBEPParameterSpec( salt, iterationCount );
            cipher = Cipher.getInstance( "PBEWithMD5AndDES" ); //获取Cipher对象实例
            //在解密模式中用密匙初始化Cipher对象
            cipher.init( Cipher.DECRYPT_MODE, secretKey, parameterSpec );
        }
        //异常处理，参见步骤详解6
    }
    try { //读取文件并解密
        File file = new File( fileName );
        FileInputStream fileInputStream = new FileInputStream( file );
        CipherInputStream in = new CipherInputStream( fileInputStream, cipher );
        byte contents = ( byte ) in.read(); //读取文件
        while ( contents != -1 ) {

```

异常

例


```
        fileBytes.add( new Byte( contents ) );
        contents = ( byte ) in.read();
    }
    in.close();
}
catch ( IOException exception ) {
    exception.printStackTrace();
    System.exit( 1 );
}
byte[] decryptedText = new byte[ fileBytes.size() ]; //Vector对象内容存储到字节数组
for ( int i = 0; i < fileBytes.size(); i++ ) {
    decryptedText[ i ] = ( ( Byte )fileBytes.elementAt( i ) ).byteValue();
}
fileContentsEditorPane.setText( new String( decryptedText ) ); //更新“文件内容”编辑器面板中的内容
}
}
```

本章小结

自Sun公司在Internet上分发Java以来，Java语言、Java平台、Java技术的发展与应用取得了令世人瞩目的成就。随着技术的进步，Sun公司除了进一步改进JDK开发工具外，还宣布了使Java用于多媒体、企业、商业、安全、服务器、管理及嵌入式应用等不同领域的Java标准扩充API。本章列举了几个不同难易程度的实用案例程序，覆盖了Java在安全、打印等方面的API相应用法。Java在Web服务上的便利性和扩展性，是Java保持其魅力的原因之一。

第10章 Java综合案例

本章内容

- 综合案例1: 多页面文本编辑器
- 综合案例2: “逃亡者”手机游戏
- 综合案例3: 网上CD销售系统
- 综合案例4: 航空查询订票系统
- 本章小结



综合案例1: 多页面文本编辑器

案例运行效果与操作

本案例为Java Swing应用方面的综合案例,讲述如何开发一个具有标准Windows界面风格的多页面文本编辑器。程序运行后,界面如图10-1所示。

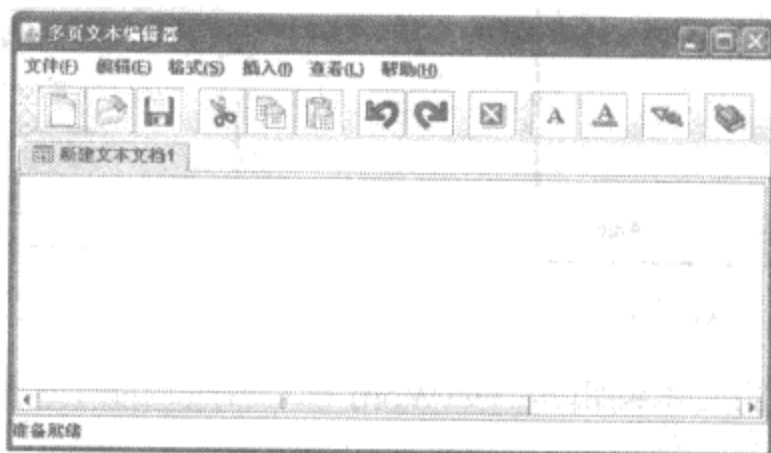


图10-1 运行时界面

由图10-1中可以看出,该程序界面具有标准的Windows界面风格,它由标题栏、菜单栏、工具栏、编辑区及状态栏组成,编辑区由多个Tab页完成多页面编辑的功能。其各个菜单包含的内容如图10-2和图10-3所示。



图10-2 “文件”菜单、“编辑”菜单和“格式”菜单

由菜单命令可以看出,该程序实现了基本的文本编辑功能,如图10-4所示为打开两个Tab页面的情形。



图10-3 “插入”菜单、“查看”菜单和“帮助”菜单

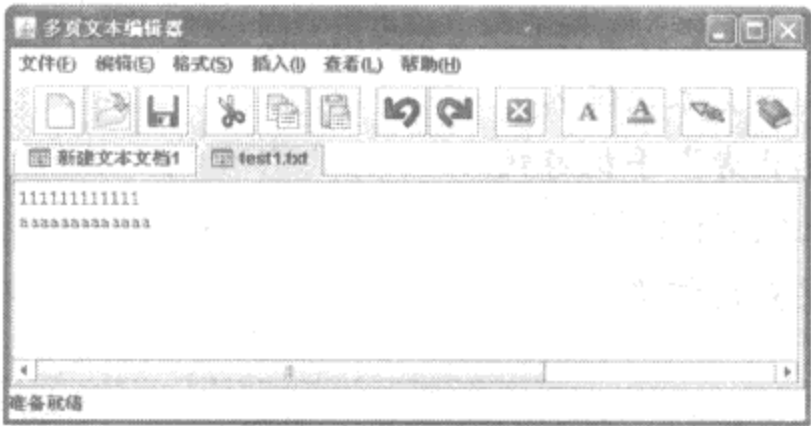


图10-4 打开两个Tab页面

与常见的文本编辑器（如Word等）类似，在“格式”菜单中可以设置字体大小、颜色以及背景颜色等，此时会弹出相应的对话框，例如，如图10-5所示即为字体设置对话框。在“帮助”菜单下单击“关于”命令，会弹出“关于”对话框，如图10-6所示。

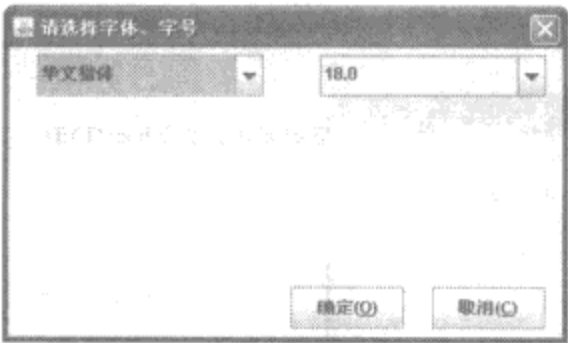


图10-5 字体设置对话框



图10-6 “关于”对话框

其他编辑功能与其他的文本编辑器（如Word等，当然功能上差很远）类似，不再赘述。

制作要点

- 1. Swing各种控件类的综合应用。
它包括JLabel、JButton、JTextArea、JCheckBox、JComboBox、JEditorPane、JScrollPane、JTabbedPane等各种控件类的使用。
- 2. Swing容器类JFrame的应用。
- 3. Swing对话框的应用。
它包括对话框JDialog类、文件选择器JFileChooser类和颜色选择器JColorChooser类的应用。
- 4. 布局管理器的应用。
- 5. Swing菜单的应用。
- 6. ImageIcon类的应用（Java图像处理）。
- 7. 界面外观LookAndFell类的应用。
- 8. Swing线程的应用。
- 9. 程序设计。

本案例由四个类构成，第一个类是MultiPageEditor类，它主要负责完成主界面初始化、主界面各种事件的侦听和处理以及界面设置的功能。

第二个类是EditorTabPage类，它负责对各个Tab编辑页面进行管理，包括文本编辑处理的主要逻辑、各个Tab编辑页面线程的管理等，还包含了进行相应文本编辑处理使用的内部类。

另外两个辅助类分别是FontDialog类和FrameAbout类；它们分别用来弹出字体设置对话框和“关于”对话框。

步骤详解

1. 在相应\chapter10\multipageeditor\src\multipageeditor目录下新建一个名为ICON的图标文件夹，将准备好的所有图标文件复制到该文件夹。

2. 为MultiPageEditor类编写构造方法，在源代码编辑器中添加代码如下：

```
public MultiPageEditor(){//构造方法
    try {
        mainFrameInit();    //主窗口初始化
        lblStatus.setText("正在初始化图标.....");
        initIcon();    //初始化图标
        lblStatus.setText("正在初始化Tab编辑页");
        initPage();    //初始化Tab编辑页
        this.setSize(800, 600);
        this.setLocationByPlatform(true);
        this.setVisible(true);    //显示主窗口
        lblStatus.setText("准备就绪");
    }
    catch (Exception ex) { ex.printStackTrace();}
}
```

3. 依次为MultiPageEditor类编写各种事件侦听器内部类，包括窗口事件、菜单事件、按钮事件等，下面以菜单栏中的“字体颜色”按钮事件侦听器内部类为例进行讲解，其余事件侦听器内部类代码与此类似，不再赘述。添加代码如下：

```
//侦听“字体颜色”菜单项动作事件
class MultiPageEditor_mnuFontColor_actionAdapter implements ActionListener {
    private MultiPageEditor adaptee;
    MultiPageEditor_mnuFontColor_actionAdapter(MultiPageEditor adaptee) {
        this.adaptee = adaptee;
    }
    public void actionPerformed(ActionEvent e) {
        adaptee.mnuFontColor_actionPerformed(e);
    }
}
```

4. 依次为MultiPageEditor类编写各种事件处理方法，包括窗口事件、菜单事件、按钮事件等，下面以工具栏中的“设置字体颜色”按钮事件处理方法为例进行讲解，其余事件处理方法代码与此类似，不再赘述。添加代码如下：

```
public void btnFontColor_actionPerformed(ActionEvent e) {    //工具栏中的“设置字体颜色”按钮事件处理
    EditorTabPage page = getCurrentPage();
```



```

        if (page == null) { return;}
        Color color = getChooseColor(page.getFontColor());
        page.setFontColor(color);
    }

```

5. 为MultiPageEditor类编写创建新的Tab编辑页方法NewPage, 添加代码如下:

```

private void NewPage() {    //创建新的Tab编辑页
    EditorTabPage page = new EditorTabPage();    //创建EditorTabPage类实例
    this.TabControl.addTab("新建文本文档" + PageCount, new ImageIcon(
        this.getClass().getResource("ICON/TabPage.png")), page);    //设置Tab编辑页图标
    PageCount += 1;    //打开的Tab编辑页数加1
    setTabPage();    //调用setTabPage方法, 将其设置为当前活动Tab编辑页
    this.repaint();    //重画
}

```

6. 为MultiPageEditor类编写其他方法, 如打开一个或多个文档方法OpenDoc等。

7. 编写EditorTabPage类代码。首先为EditorTabPage类编写构造方法, 定义好相应的成员变量, 添加代码如下:

```

public EditorTabPage() {    //无参数构造方法
    super();    //调用基类构造方法
    editor = new JEditorPane();
    editor.setFont(font);
    editordocument = editor.getDocument();    //设置文档模型
    editordocument.addDocumentListener(new
        editordocument_documentAdapter(this));    //为文档模型添加事件处理
    editordocument.addUndoableEditListener(undoHandler);
    this.getViewPort().add(editor);    //为editor提供滚动支持
    resetUndoManager();
    ischange=false;
}

public EditorTabPage(String filename) {    //带参数构造方法
    this();    //调用无参数构造方法
    _filename = filename;
    if (_filename.endsWith(".rtf") || _filename.endsWith(".html")
        || _filename.endsWith(".htm")) {
        OpenFileName="file:\\\\" + _filename;
        OpenRTFHtmlFile();
    }
    else {
        OpenTXTAndElseFile();
    }
    editordocument = editor.getDocument();
    editordocument.addDocumentListener(new
        editordocument_documentAdapter(this));    //为文档模型添加事件处理
    editordocument.addUndoableEditListener(undoHandler);
    resetUndoManager();
    ischange=false;
}

```

8. 为EditorTabPage类编写打开文本文档的OpenTXTAndElseFile()方法, 该方法创建一个继承了Thread类的内部类OpenTXTThread的实例, 并启动该线程的运行方法。添加代码如下:


```

private void OpenTXTAndElseFile() { //打开文本文档
    OpenTXTThread open = new OpenTXTThread();
    open.start();
}

class OpenTXTThread extends Thread { //内部类，用于创建打开文本文档的线程
    public void run() { //打开文本文档的线程运行方法
        FileReader file = null;
        BufferedReader br = null;
        try {
            file = new FileReader(_filename);
            br = new BufferedReader(file);
            String line;
            StringBuffer buffer = new StringBuffer();
            while ((line = br.readLine()) != null) {
                buffer.append(line + "\n");
            }
            file.close();
            br.close();
            editor.setText(buffer.toString());
        }
        catch (IOException ex) {
            JOptionPane.showMessageDialog(null, _filename
                + " 文件打开时发生了IO异常,或许该文件不存在!\n" + ex.toString());
        }
        ischange=false;
    }
}

```

9. 依次为EditorTabPage类编写实现各种编辑功能的方法，包括剪切、复制、粘贴、全选、删除、清空、另存为、隐藏、插入日期和时间、设置字体和背景颜色等功能，它们都是调用JEditorPane类的相应方法。下面以实现插入日期和时间功能为例进行讲解，其余方法代码与此类似，不再赘述。添加代码如下：

```

//插入日期和时间
public void InsertDateTime() {
    //获取当前日期和时间
    Date d = new Date();
    String s;
    //格式化输出
    s = DateFormat.getDateInstance(DateFormat.FULL).format(d)+
        DateFormat.getTimeInstance(DateFormat.FULL).format(d);
    editor.replaceSelection(s);
}

```

程序源代码与解释

可在华信教育资源网上下载此案例源代码。



综合案例2：“逃亡者”手机游戏

案例运行效果与操作

本案例是Java ME应用方面的综合案例，讲述如何开发一个2D的单屏幕飞机游戏。程序运行后，界面如图10-7所示。

单击“Launch”软键运行程序，进入闪屏界面，如图10-8所示。画面为一架战斗机，以及游戏的名称和版本信息。三秒钟后或第一次按键后进入游戏菜单，此时界面如图10-9所示。另外，若在游戏中用户按下一个非游戏键从而暂停了游戏时，在菜单列表的顶部还有一个额外的菜单项“继续游戏”，如图10-10所示。



图10-7 初始界面



图10-8 闪屏界面

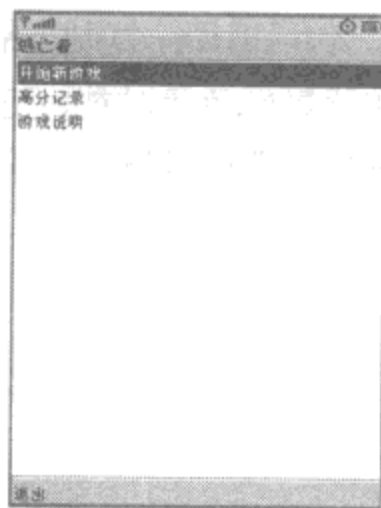


图10-9 游戏菜单界面



图10-10 暂停后的游戏菜单界面

选择“开始新游戏”菜单命令，会进入游戏界面，如图10-11所示。当游戏结束时则进入游戏结束界面，如图10-12和图10-13所示。屏幕上显示了玩家的成绩和等级，以及游戏的最好成绩，如果当前成绩是最好成绩，则手机震动并播放音乐庆祝成功。

在游戏菜单界面中选择“高分记录”菜单命令，则进入最好成绩界面，如图10-14和图10-15所示，其中，图10-14是首次进入游戏尚没有高分记录时的界面。可按“后退”软键返回菜单。

在游戏菜单屏幕中选择“游戏说明”菜单命令，则进入游戏说明界面，如图10-16所示，可按“后退”软键返回菜单。

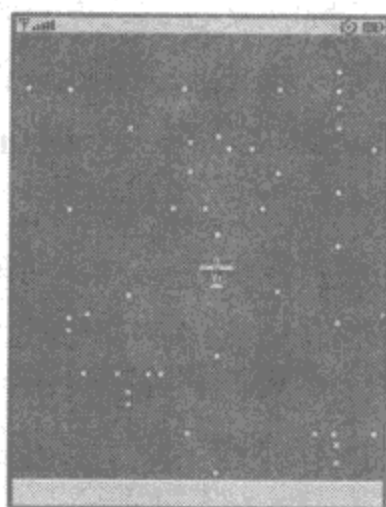


图10-11 游戏界面



图10-12 打破记录界面



图10-13 未打破记录界面



图10-14 最好成绩初始界面



图10-15 最好成绩界面

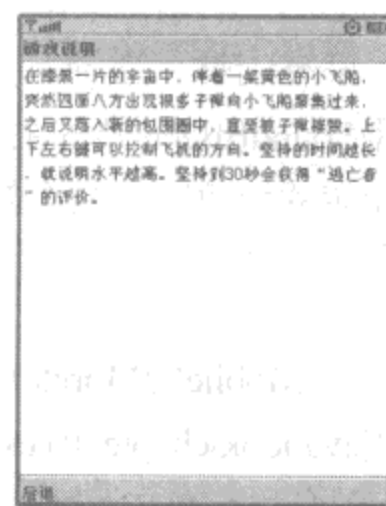


图10-16 游戏说明界面

制作要点

1. MIDlet高级屏幕的综合应用。
2. MIDP低级屏幕的综合应用。
3. 精灵Sprite类的应用。
4. 程序设计。

“逃亡者”游戏一共实现了10个类，分别是：继承了MIDlet类的主类escapeeMIDlet类，游戏核心类包括继承了Canvas类的escapeeCanvas类以及子弹类Bullets、逃亡飞机类Escapee和声音效果类SoundEffects，游戏外部的辅助类包括闪屏类SplashScreen、菜单类MenuList、高分屏幕类HighScoreScreen、简介屏幕类InstructionsScreen、结束屏幕类GameOverScreen。

其中，`escapeeMIDlet`类被用做一个状态机来管理各种屏幕以及它们之间的转换。例如，当显示`splash`屏幕时，该类和`SplashScreen`类通过方法`splashScreenPainted`和`splashScreenDone`共同在方法`init`中完成背景初始化。而使用方法`readRecordStore`和`writeRecordStore`在一个名为“BESTTIME”的记录存储区中保存最高得分等。

闪屏类`SplashScreen`在屏幕中央显示图像。在第一次绘制屏幕后，它将图像释放作为垃圾回收（把图片设为`null`）并回调`MIDlet`进行初始化工作。三秒钟后或第一次按键后，它再次回调`MIDlet`显示游戏菜单。通过这种方法，`MIDlet`可在显示`Splash`屏幕的同时进行初始化。

菜单列表类`MenuList`则是在`Splash`屏幕之后，或者游戏结束时，或者游戏中用户按下一个非游戏键时（从而暂停了游戏）显示的屏幕。

`HighScoreScreen`类用来显示最高得分屏幕，它显示到目前为止的最好成绩，是`Form`的子类，并且实现了`CommandListener`接口。

简介屏幕`InstructionsScreen`类向用户显示说明文本，也是`Form`的子类，并且实现了`CommandListener`接口。

子弹`Bullets`类和逃亡飞机`Escapee`类分别实现了在屏幕上显示和控制子弹和逃亡飞机的功能，它们都是精灵`Sprite`类的子类，继承了`Sprite`类的很多方法。

游戏画布`escapeeCanvas`类负责处理游戏的大部分工作，包括检测用户输入、处理游戏逻辑和绘制游戏画面，所有这些工作都放置在游戏线程中完成。`escapeeCanvas`类是`GameCanvas`类的子类，并且实现了`Runnable`接口。

结束屏幕`GameOverScreen`类实现了当飞机和子弹碰撞后显示结束屏幕的功能。该屏幕显示游戏时间和等级，并指出这是否是新的游戏最长时间，或者给游戏者显示出当前最好记录。当它出现时，屏幕闪烁一秒钟，并播放两个短MIDI曲调之一。

声音效果`SoundEffects`类的主要功能为使用MIDP 2.0 Media API播放三种声响效果：碰撞爆炸声音和游戏结束时的两个短MIDI曲调。

步骤详解

1. 新建基于Mobile的Mobile应用程序项目`escapee`。
2. 在C:\javabook\chapter10\escapee\src\escapee目录下新建一个名为`res`的资源文件夹，将准备好的所有图像和声音文件复制到该文件夹，并在开发环境中的“库和资源”中添加上`res`。
3. 在`escapee`中新建`MIDlet`类`escapeeMIDlet`，新建Java类`escapeeCanvas`，再用同样的方法依次添加其他8个类。
4. `escapeeMIDlet`类继承自`MIDlet`类，用于连接设备的应用程序管理器（`Application Manager`），通过方法`startApp`、`pauseApp`、`destroyApp`来通知游戏的开始、暂停和结束。为`escapeeMIDlet.java`文件添加代码如下：

```
public void startApp() {  
    //获得关于显示的设备上下文  
    Displayable current = Display.getDisplay(this).getCurrent();  
    if (current == null) {  
        //第一次将显示闪屏  
        Display.getDisplay(this).setCurrent(new SplashScreen(this));  
    }  
    else {
```



```

        if (current == myCanvas) {
            myCanvas.start();    //重新开始游戏线程
        }
        //显示游戏画布内容
        Display.getDisplay(this).setCurrent(current);
    }
}

public void pauseApp() {
    Displayable current = Display.getDisplay(this).getCurrent(); //获得当前设备的显示上下文
    if (current == myCanvas) {    //如果屏幕上的显示内容为游戏画布
        myCanvas.stop();        //中止游戏线程
    }
}

public void destroyApp(boolean unconditional) {
    if (myCanvas != null) {
        myCanvas.stop();        //在销毁程序之前先停止游戏线程
    }
}
}

```

5. 为escapeeMIDlet类添加splashScreenPainted方法，当闪屏绘制完毕时，将会调用该方法，表示程序可以利用剩余的间歇来完成一些工作，这里启动了MIDlet实现的一个线程，它会自动去调用run方法。在run方法中将会进行一些初始化工作，包括读取游戏记录、初始化游戏声音、初始化游戏菜单、初始化游戏画布。当所有工作都完成时，将一个initDone标志位设置为true，表明已经初始化完毕（用来避免重复初始化）。当闪屏结束后，将会调用splashScreenDone方法，在这个方法里将检测初始化是否完毕，如果还没有完成则继续初始化，如果已经完毕则显示游戏菜单。添加代码如下：

```

void splashScreenPainted() {
    new Thread(this).start();    //启动幕后的初始化线程
}

public void run() {            //线程运行方法
    init();                    //调用初始化方法
}

private synchronized void init() { //初始化方法
    if (!initDone) {            //如果还没有初始化完毕
        readRecordStore();        //读取游戏记录
        SoundEffects.getInstance(); //获得声音效果类的唯一实例
        menuList = new MenuList(this); //初始化菜单
        myCanvas = new escapeeCanvas(this); //初始化游戏画布
        initDone = true;        //标志位为true表示初始化完毕
    }
}

void splashScreenDone() {
    init();                    //检测是否初始化完毕，如果没有继续初始化
    Display.getDisplay(this).setCurrent(menuList); //在屏幕上显示游戏菜单
}

```

6. 为escapeeMIDlet类添加根据字符串类型的文件名来加载图片资源的通用方法createImage，这样其他各个类都可以使用这个方法加载图片，在源代码编辑器中添加代码如下：

```

static Image createImage(String filename) {    //加载图片资源
    Image image = null;

```



```

    try {
        image = Image.createImage(filename);    //根据文件名创建Image对象
    }
    catch (java.io.IOException ex) {
    }
    return image;        //返回图片对象image
}

```

7. 为escapeeMIDlet类添加游戏特效方法，添加代码如下：

```

void flashBacklight(int millis) {
    //手机背景光闪烁，持续时间为millis
    Display.getDisplay(this).flashBacklight(millis);
}
//手机震动，持续时间为millis
void vibrate(int millis) {
    Display.getDisplay(this).vibrate(millis);
}

```

8. 为escapeeCanvas类添加tick方法，该方法用来实现游戏逻辑。在源代码编辑器中添加代码如下：

```

private void tick() {        //游戏逻辑
    escapee.tick();        //飞机的运动
    if (!isCollidesWith) bullets.tick();    //如果未碰撞，子弹运动
    //用explosionCnt来计数，当爆炸动画播放完，游戏结束
    if (isCollidesWith && (explosionCnt!=0)) {
        explosion.setFrame((explosionCnt-1));
        explosionCnt--;
        if(explosionCnt == 0)
            isGameOver = true;
    }
    if(bullets.collidesWith(escapee)){        //如果子弹和飞机发生碰撞
        isCollidesWith = true;        //置标志位，发生碰撞
        escapee.setAlive(false);        //置标志位，飞机不存活
        SoundEffects.getInstance().startBlastSound();    //播放爆炸声音
        //定位爆炸点
        explosion.setPosition(escapee.getRefPixelX()-explosion.getWidth()/2,
                                escapee.getRefPixelY()-explosion.getHeight()/2);
        explosion.setVisible(true);        //爆炸精灵可见
    }
    if(isGameOver){        //如果游戏结束
        long time = (System.currentTimeMillis() - startTime);    //记录玩家的持续时间
        midlet.GameCanvasGameOver(time,BULLETS_NUM);    //绘制游戏结束画面
    }
}

```

9. 为Bullets类添加碰撞检测方法，因为每个子弹都是Sprite类对象，所以这里使用Sprite类的collidesWith方法来检测精灵与精灵之间的碰撞。在源代码编辑器中添加代码如下：

```

public boolean collidesWith(Sprite s){        //碰撞检测方法
    for (int i = 0; i < bullets.length; i++) {    //遍历子弹数组
        if(bullets[i][5]!= ALIVE){        //如果子弹处于非激活状态则进行下一轮循环
            continue;
        }
    }
}

```



```

    }
    setPosition(bullets[i][1],bullets[i][2]); //将子弹设置到子弹数组所指定位置
    //和逃亡飞机进行像素级的碰撞检测，如果发生碰撞则返回
    if(collidesWith(s,true)){
        return true;
    }
}
return false; //如果未发生碰撞，则返回false
}

```

10. 为Escapee类添加飞机移动move方法，按照用户键控获得飞机移动的方向，然后根据移动的方向和当前的速度计算出下一帧的位置，再调用精灵类的setPosition方法将飞机放置到那个位置上，也可以直接调用move方法向那个方向移动若干个像素。需要注意的是，飞机的移动不可能超出屏幕边界，因此应该在各个移动方向上设置边界检测。添加代码如下：

```

public void move(int direction){ //飞机移动方法
    if(direction == UP){ //如果方向向上
        //调用move方法向上移动坐标，每次移动幅度为SPEED个像素
        move(0,-SPEED);
        //如果到达上边界（y坐标为0）则停止不动
        if(getY()<0) setPosition(getX(),0);
        setFrame(0);
    }
    if(direction == DOWN){ //如果方向向下
        //调用move方法向下移动坐标，每次移动幅度为SPEED个像素
        move(0,SPEED);
        //如果到达下边界（y坐标靠近CanvasHeight）则停止不动
        if(getY()>CanvasHeight-frameHeight) setPosition(getX(),CanvasHeight-frameHeight);
        setFrame(0);
    }
    if(direction == LEFT){ //如果方向向左
        //调用move方法向左移动坐标，每次移动幅度为SPEED个像素
        move(-SPEED,0);
        //如果到达左边界（x坐标靠近0）则停止不动
        if(getX()<0) setPosition(0,getY());
        setFrame(1);
    }
    if(direction == RIGHT){ //如果方向向右
        //调用move方法向右移动坐标，每次移动幅度为SPEED个像素
        move(SPEED,0);
        //如果到达右边界（x坐标靠近CanvasWidth）则停止不动
        if(getX()>CanvasWidth-frameWidth) setPosition(CanvasWidth-frameWidth,getY());
        setFrame(2);
    }
    isMove = true;
}

```

11. 为SoundEffects类添加创建播放器方法：

```

private Player createPlayer(String filename, String format) { //创建播放器方法
    Player p = null;
    try {
        InputStream is = getClass().getResourceAsStream(filename); //以流的方式读取资源
    }
}

```



```
        if (is!=null){
            p = Manager.createPlayer(is, format);    //根据流和文件格式创建播放器
            p.prefetch();
        }
    }
    catch (IOException ioe) {}    //捕获IO异常
    catch (MediaException me) {}    //捕获媒体异常
    return p;    //返回Player对象
}
```

12. 为SplashScreen类添加构造方法和获取实例方法:

```
SplashScreen( escapeeMIDlet midlet) {    //构造方法
    this.midlet = midlet;
    setFullScreenMode(true);
    splashImage = escapeeMIDlet.createImage("/splash.png");
    new Thread(this).start();
}
```

13. 为HighScoreScreen类添加构造方法:

```
HighScoreScreen(escapeeMIDlet midlet) {    //构造方法
    super("最好成绩");    //实现父类的构造函数, 指定Form的名称
    this.midlet = midlet;
    long bestTime = midlet.getBestTime();    //从midlet中获取最好成绩
    float BestTime_Second = (float)bestTime/1000;    //将成绩转换为以秒为单位 (原来是毫秒)

    String text = (bestTime == -1) ? "目前还没有, 期待您的表现哦!"
        : (Float.toString(BestTime_Second) + "s");    //判断是否存在这个成绩
    append(new StringItem("最长坚持时间: ", text));
    backCommand = new Command("后退", Command.BACK, 1);    //添加后退软键
    addCommand(backCommand);
    setCommandListener(this);    //绑定侦听器
}
```

14. 其他代码添加与解释就不再多讲, 详见程序源代码。

程序源代码与解释

可在华信教育资源网上下载此案例源代码。



综合案例3: 网上CD销售系统

案例运行效果与操作

本案例是Java Web应用方面的综合案例, 综合利用JSP、Servlet和JavaBean技术, 模拟实现了一个基于MVC架构的网上CD销售系统。程序运行后, 会打开一个IE浏览器窗口, 如图10-17所示。页面上部为网站的Logo和一排导航链接; 页面中间左部列出CD分类的超级链接以及具有查询功能的一些组件, 支持三种查询关键字和两种查询方式, 页面中间右部以分页方式列出了当前的所有商品, 单击“下一页”链接可以查看其他页的商品, 还可以选择商品后单击“添加到购物车”按钮进行购买操作, 商品信息最右侧还有一个商品“详细信息”的链接; 页面底部显示网站的相关信息。



图10-17 欢迎页面

单击CD分类的超级链接, 则会显示相应类别的所有商品, 例如单击“华语流行”类别后, 页面显示如图10-18所示。



图10-18 显示某一类别的商品

本系统查询功能支持三种查询关键字和两种查询方式, 其中关键字支持CD名称、歌手名和歌曲名查询, 查询方式支持精确查询和模糊查询。精确查询会从数据库中查找与用户输入的查询关键字完全匹配的商品, 而模糊查询从数据库中查找包含用户输入的查询关键字的商品。在CD查询“关键字”文本框中输入相应的关键字, 例如输入“王力宏”, 选择关键字类型为“CD名称”, 查询方式选择“模糊查询”, 单击“查询”按钮后, 查询结果如图10-19所示。而如果选择“精确查询”方式, 则查询结果如图10-20所示。

单击商品信息最右侧的“详细信息”链接, 会打开该商品的详细信息页面, 如图10-21所示。



图10-19 模糊查询



图10-20 精确查询

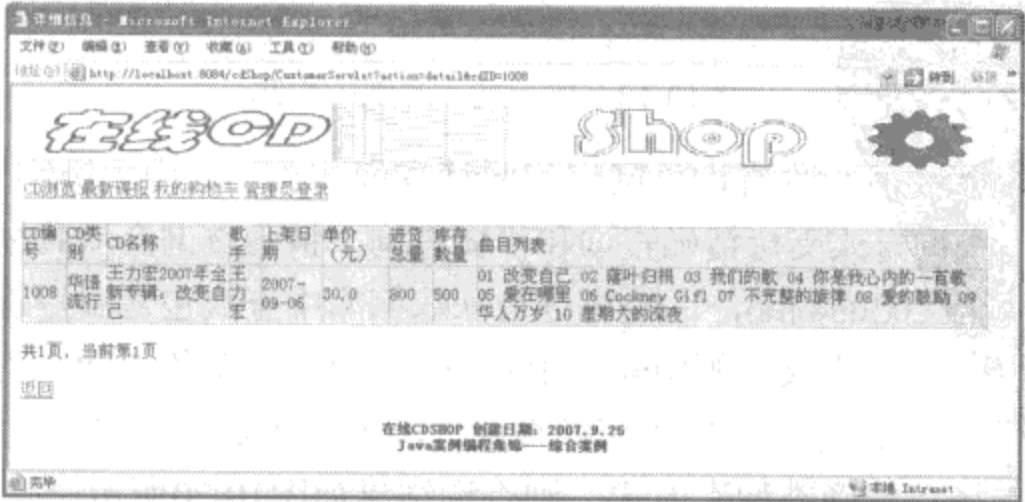


图10-21 商品详细信息

在如图10-17所示窗口中选择编号为1011和2002商品前面的复选框，单击“添加到购物车”按钮，会进入“我的购物车”页面，如图10-22所示。

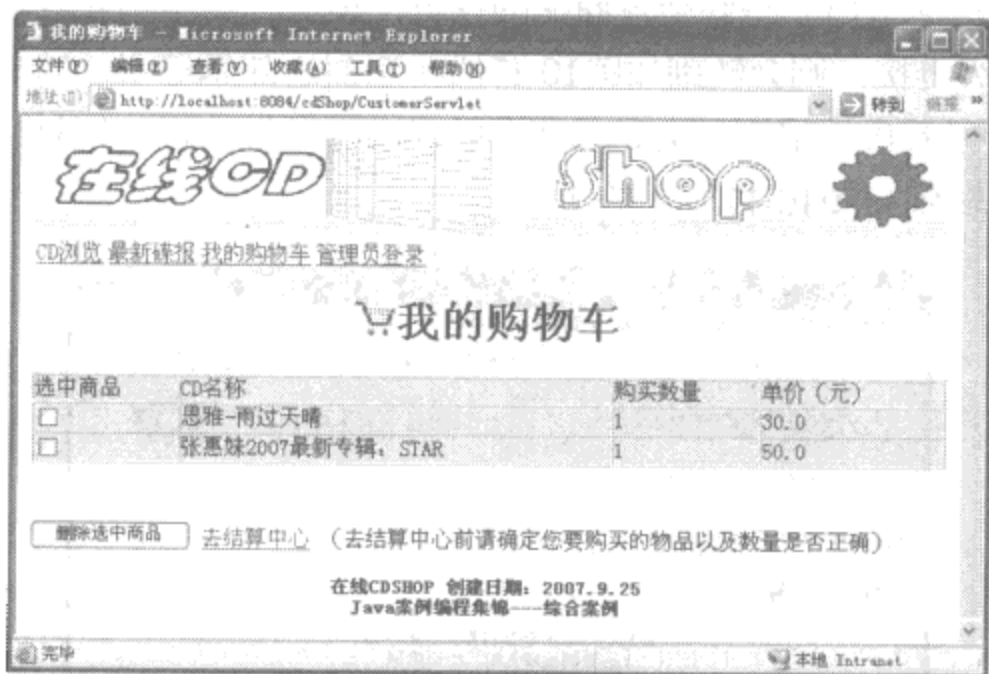


图10-22 “我的购物车”页面

在购物车中选择“恩雅-雨过天晴”前面的复选框，单击“删除选中商品”按钮，此时页面内容如图10-23所示。可以看出，单击“删除选中商品”按钮后，被选中的商品将会从购物车中删除。

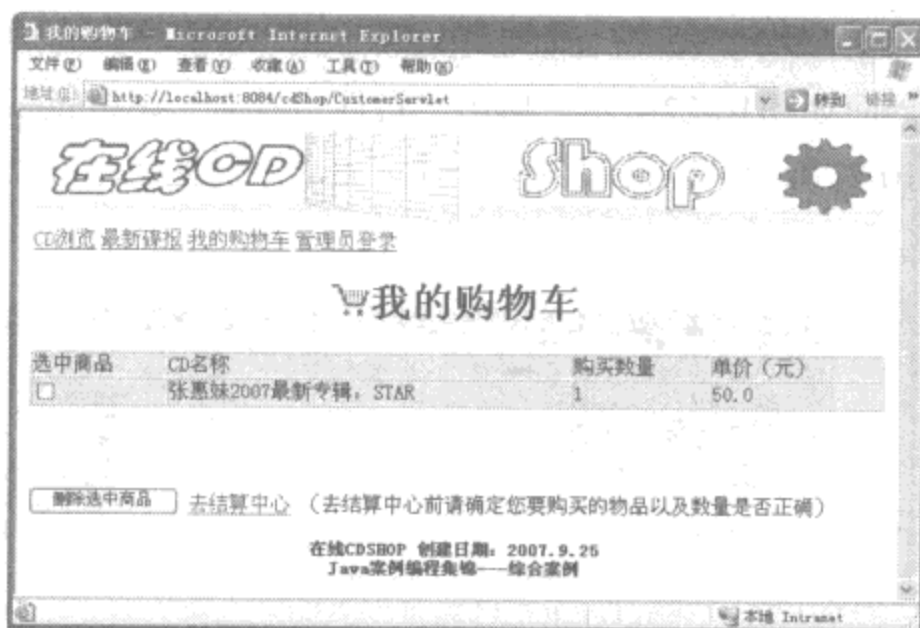


图10-23 删除购物车中的商品

在购物车页面单击“去结算中心”超级链接，会进入“结算中心”页面，如图10-24所示。

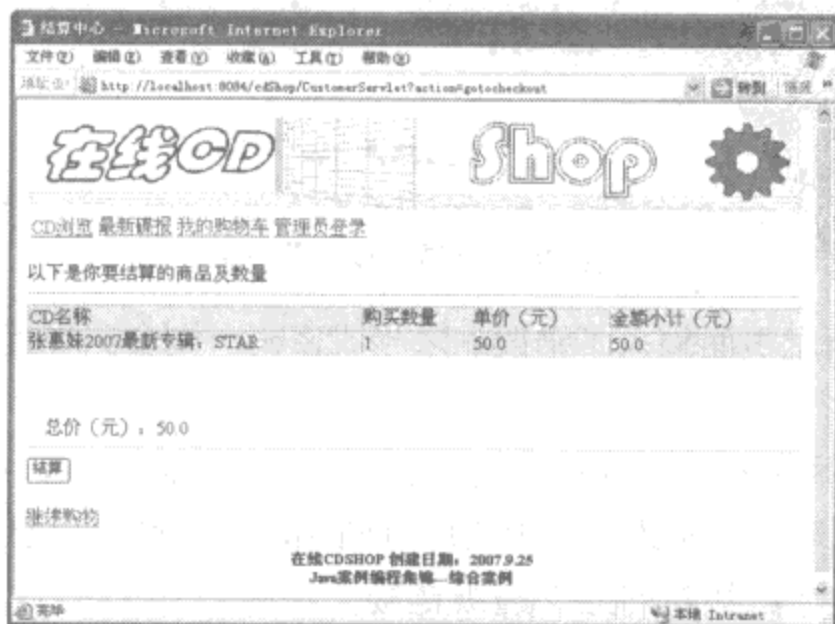


图10-24 “结算中心”页面

“结算中心”页面给出了将要结算商品的数量、单价、金额小计以及总价。此时如果单击“继续购物”超级链接，则会转到如图10-17所示页面。如果单击“结算”按钮，页面内容如图10-25所示。可以看出，结算后会显示“已经成功结算”的提示信息。此时单击“继续购物”超级链接，会转到如图10-17所示的页面。



图10-25 “成功结算”页面

页面上部还有四个导航链接，其中单击“CD浏览”超级链接会显示当前的所有商品，界面如图10-17所示。单击“最新碟报”超级链接会显示一个指定日期（本案例为2007年9月1日）以后上市的所有商品的列表，界面如图10-26所示。单击“我的购物车”超级链接会打开一个购物车页面，界面如图10-22所示。单击“管理员登录”超级链接则会打开一个管理员登录页面，界面如图10-27所示。以管理员身份登录后，可以完成系统的后台管理，比如删除商品、添加新商品等。



图10-26 “最新碟报”页面

如果在如图10-27所示页面输入了错误的账号或密码，则会出现提示信息要求用户重新输入，如图10-28所示。

输入了正确的账号和密码后，系统自动跳转到查看和删除页面，如图10-29所示。可以看出，它以分页的方式显示了当前数据库中的所有商品，每个商品信息右侧有一个“删除”链接，页面左下部有一个“添加新商品”超级链接。

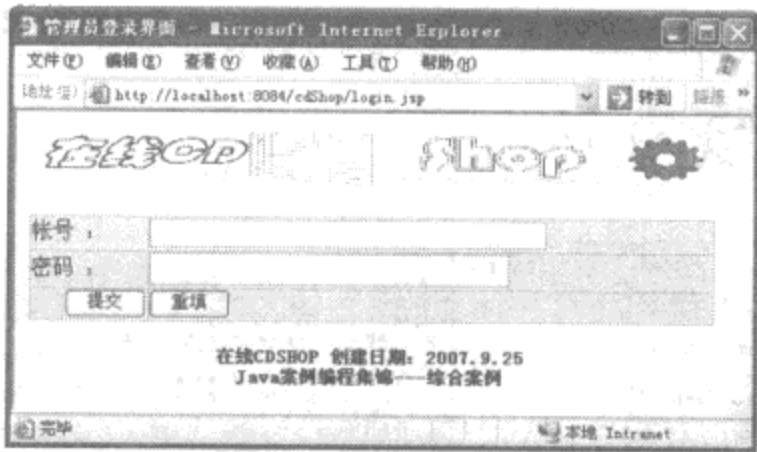


图10-27 管理员登录页面

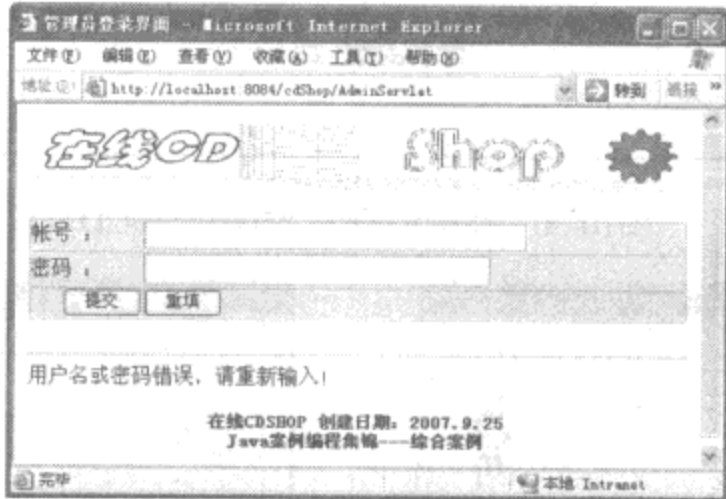


图10-28 管理员登录错误提示页面



图10-29 查看和删除页面

单击“删除”链接可以删除相应的CD，比如单击编号为“5005”的CD所对应的“删除”链接后，页面显示如图10-30所示。

可以看出，单击“删除”链接后，相应的CD确实被删除。单击“添加新商品”链接后，进入添加商品页面，如图10-31所示，可以输入相应的各项信息。

单击“添加”按钮，会显示“成功添加商品”的提示信息，如图10-32所示。

此时单击“查看已有商品”链接，页面内容如图10-33所示。可以看出，编号为“5005”的商品确实被添加到数据库中。

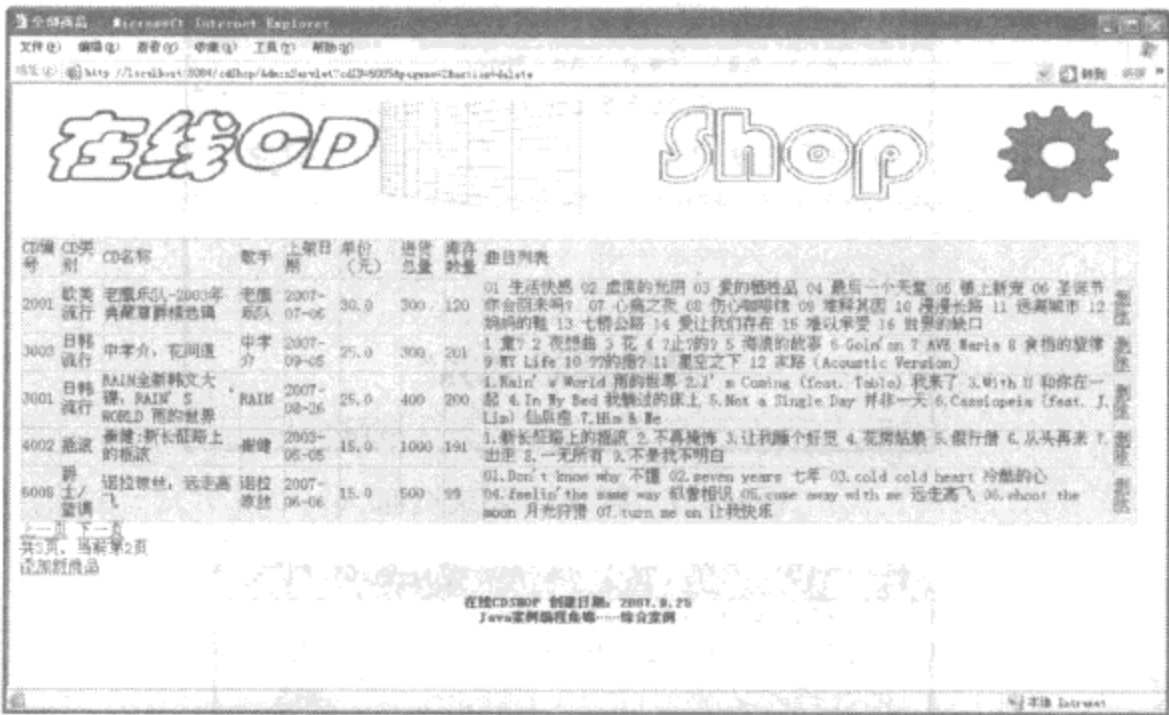


图10-30 删除编号为“5005”的CD



图10-31 添加新商品页面

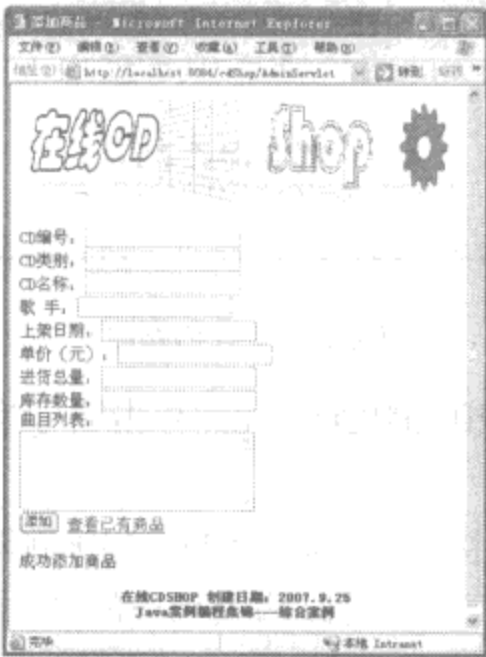


图10-32 添加新商品成功提示页面



图10-33 成功添加编号为“5005”的CD

制作要点

1. MVC模式的应用。

本案例是一个Web应用程序，根据MVC模式，Web层MVC的控制器通常是Java Servlet，Web层MVC的视图通常由HTML和JSP页面组成，而Web层MVC的模型部分可以有很多不同的形式。本案例使用JSP进行视图展现，用Servlet作为控制器进行集中控制，使用JavaBean组件作为模型封装所有的程序逻辑。这样，以往通常要放在每个JSP中的程序代码，可以集中放在控制器中，由其处理所有的请求。

2. 程序设计。

Web应用程序需要Web服务器进行服务，本案例使用了NetBeans捆绑的Tomcat 5.5.17作为Web服务器。数据库使用了Access数据库。

本案例设计成两个模块，即后台管理模块和查询购买模块，根据MVC模式，各子模块的名称以及功能说明如表10-1和表10-2所示。

表10-1 后台管理模块中各子模块名称及说明

文件名称	说明
AdminDataBaseBean.java	封装了与数据库有关操作的JavaBean组件
AdminServlet.java	控制器，根据客户端的请求进行相应的操作
login.jsp	管理员登录页面，如果登录不成功还会返回该页面
showadddelete.jsp	显示和删除商品页面
addproduct.jsp	添加商品页面

表10-2 查询购买模块中各子模块名称及说明

文件名称	说明
DataBaseBean.java	封装了与数据库有关操作的JavaBean组件
CustomerServlet.java	控制器，根据客户端的请求进行相应的操作
CDProductBean.java	表示商品信息的JavaBean组件
CartBean.java	封装购物车信息的JavaBean组件
CartProduct.java	表示购物车中商品信息的JavaBean组件
head.jsp	页面顶端显示通用组件，供其他页面调用
left.jsp	页面左端显示通用组件，供其他页面调用
tail.jsp	页面底端显示通用组件，供其他页面调用
welcome.jsp	程序的欢迎页面
searchview.jsp	商品查询显示页面
detail.jsp	商品详细信息页面
cartview.jsp	显示购物车中的商品页面
checkout.jsp	结算页面

步骤详解

1. 准备工作：创建数据库和表，创建数据源等。在这里选用Access作为项目的后台数据库，用Access创建一个名称为cdshop的数据库，并向该数据库中添加一个名为cdTable的表和一个名为admin的表，按照表10-3和表10-4设定表中各个字段名称及其属性。

表10-3 cdTable表各个字段的名称及属性

字段名称	数据类型	是否主键	字段大小	必填字段	是否可空
cdID	数字	是	长整型	是	否
name	文本	否	100	是	否
singer	文本	否	20	否	是
RegTime	日期/时间	否		否	
price	数字	否	双精度型	是	
amount	数字	否	整型	是	
leftNumber	数字	否	整型	是	
musicList	备注	否		否	是
category	文本	否	20	否	是

表10-4 admin表各个字段的名称及属性

字段名称	数据类型	是否主键	字段大小	必填字段	是否可空
admin_id	文本	是	50	是	否
password	文本	否	50	是	否

将上述数据库文件cdshop.mdb复制到C:\javabook\chapter10目录，再配置一个名称为cdshopDB的数据源，并将cdshop.mdb指定为该数据源所指向的数据库。

2. 建立基于Web的Web应用程序项目cdshop，服务器设置为捆绑的Tomcat 5.5.17。

3. 新建Java包cdshop。

4. 创建Java类文件：CDProductBean、CartBean、CartProduct、DataBaseBean、AdminDataBaseBean。

5. 创建Servlet文件：CustomerServlet和后台管理的AdminServlet。

6. 开发CDProductBean、CartProduct和CartBean：

(1) 为CDProductBean和CartProduct添加字段（成员变量），各字段的选项设置如表10-5和表10-6所示。并为上述字段设置相应的get与set方法。

表10-5 CDProductBean中各个字段的名称及属性

变量名称	变量类型	访问限制修饰符	初始值
cdID	int	private	null
cdName	String	private	null
singer	String	private	null
regTime	Date	private	null
cdPrice	double	private	null
amount	int	private	null
leftNumber	int	private	null
musicList	String	private	null
category	String	private	null

表10-6 CartProduct中各个字段的名称及属性

变量名称	变量类型	访问限制修饰符	初始值
cdID	int	private	null
cdName	String	private	null
cdPrice	double	private	null
selectedCount	int	private	null

(2) 向CartBean中添加一个名称为hash、类型为Hashtable、访问限制修饰符为private、初始值为new Hashtable()的字段。

(3) 向CartBean中添加一个名称为private void productNotExist(Vector vec,String id)的方法，并添加代码如下：

```
//如果当前购物车中不存在该商品，则将该商品添加到购物车中
private void productNotExist(Vector vec,String id){
    //获取当前查询出来的所有商品，并根据其ID号重新组织CartProduct的内容
    Enumeration enume=vec.elements();
    int tempid=Integer.parseInt(id);
    while(enume.hasMoreElements()){
```



```

        CDProductBean pb=(CDProductBean)enum.nextElement();
        if(tempid==(pb.getcdID())){
            CartProduct cp=new CartProduct();
            cp.setcdID(pb.getcdID());
            //从Vector中获取具有相同ID号码的商品名称
            //作为CartProduct的cdName属性的值
            cp.setcdName(pb.getcdName());
            cp.setSelectedCount(1);
            cp.setcdPrice(pb.getcdPrice());
            hash.put(id,cp);
            return ;
        }
    }
}

```

(4) 向CartBean中添加一个名称为public Hashtable getCartContent()的方法, 该方法用来获取购物车中的内容, 返回值为Hashtable对象。添加代码如下:

```

public Hashtable getCartContent(){
    return this.hash;
}

```

(5) 向CartBean中添加方法public void deleteFromCart(String id), 用来删除购物车中指定ID的商品。添加public void clearCart()方法, 用来清空购物车中的内容。添加public boolean isEmpty()方法, 用来判断当前购物车是否为空。

7. 开发DataBaseBean。

(1) 向DataBaseBean中添加一个名称为con、类型为Connection、访问限制修饰符为private、初始值为null的字段。

(2) 添加DataBaseBean的方法private void connectTODB(), 用来获取数据库连接, 并将该连接赋值给con。添加代码如下:

```

private void connectTODB(){
    String CLASSFORNAME="sun.jdbc.odbc.JdbcOdbcDriver";
    String CONNECTSTR="jdbc:odbc:cdshopDB";
    try{
        Class.forName(CLASSFORNAME);
        this.con=DriverManager.getConnection(CONNECTSTR);
    } catch(Exception e) {
        e.printStackTrace();
    }
}

```

(3) 为DataBaseBean添加public Vector selectProduct(String sql)方法, 用来从数据库中查询具有指定ID的商品, 将查询得到的结果封装成CDProductBean对象后添加到Vector对象中返回。添加public void checkOut(CartBean cartbean)方法, 用来在用户确定购买后从数据库中将该指定ID的商品的库存数量减去给定的值。

8. 开发CustomerServlet。

(1) 向CustomerServlet中添加一个名称为private void forward(HttpServletRequest request, HttpServletResponse response, String url) throws ServletException, IOException的方法, 该方法用来将请求转发到由URL指定的页面。添加代码如下:


```
private void forward(HttpServletRequest request, HttpServletResponse response, String url)
    throws ServletException, IOException {
    RequestDispatcher dispatcher = request.getRequestDispatcher(url);
    dispatcher.forward(request, response);
}
```

(2) 处理各种请求的方法 `processRequest`, 部分代码如下:

```
if(action.equals("select")){//处理用户按下“查询”按钮的请求
    String keywords=request.getParameter("keywords");
    String keywordsType=request.getParameter("keywordsType");
    String searchType=request.getParameter("searchType");
    if(keywords==null|| keywords.equals("")){//输入的查询关键字为空
        session.setAttribute("namenull","namenull");
        vec=dbBean.selectProduct("select * from cdTable");
    }else if(searchType.equals("jingque")){//查询方式为“精确查询”
        if(keywordsType.equals("1")){//查询方式为“CD名称”
            vec=dbBean.selectProduct("select * from cdTable
                                     where name like '"+keywords+"'");
        }
    }
}
```

9. 开发 `AdminDataBaseBean`。

(1) 向 `AdminDataBaseBean` 中添加一个名称为 `con`、类型为 `Connection`、访问限制修饰符为 `private`、初始值为 `null` 的字段。

(2) 向 `AdminDataBaseBean` 中添加一个名称为 `private void connectTODB()` 的方法, 该方法用来获取数据库连接, 并将该连接赋值给 `con`。添加代码如下:

```
private void connectTODB(){
    String CLASSFORNAME="sun.jdbc.odbc.JdbcOdbcDriver";
    String CONNECTSTR="jdbc:odbc:cdshopDB";
    try{
        Class.forName(CLASSFORNAME);
        this.con=DriverManager.getConnection(CONNECTSTR);
    } catch(Exception e) {
        e.printStackTrace();
    }
}
```

(3) 为 `AdminDataBaseBean` 添加 `private void close(Connection con, Statement stmt, ResultSet rest)` 方法, 用来关闭与数据库有关的连接。添加 `public boolean adminLogin(String username, String password)` 方法, 用来验证管理员登录, 从 `admin` 表中查询指定记录, 若查询成功则允许登录。添加 `public Vector getAllProduct()` 方法, 该方法用来获取 `cdTable` 表中的所有记录, 将查询得到的结果封装成 `CDProductBean` 对象后添加到 `Vector` 对象中返回。添加 `public void deleteProduct(String id)` 方法, 用来从数据库中删除具有指定 ID 的记录。添加 `public int addProduct(String id, String name, String singer, String regTime, String price, String amount, String leftNumber, String musicList, String category)` 方法, 用来向数据库中插入记录, 并返回插入的记录数。

10. 开发 `AdminServlet`。

(1) 向 `AdminServlet` 中添加一个名称为 `protected void forward(HttpServletRequest request, HttpServletResponse response, String url) throws ServletException, IOException` 的方法,

该方法用来将请求转发到由URL指定的页面。添加代码如下：

```
protected void forward(HttpServletRequest request, HttpServletResponse response, String url)
    throws ServletException, IOException {
    RequestDispatcher dispatcher = request.getRequestDispatcher(url);
    dispatcher.forward(request, response);
}
```

(2) 处理请求的processRequest方法代码如下（部分）：

```
if(action.equals("login")){    //用户单击了“提交”按钮
    String username=request.getParameter("username").trim();
    String password=request.getParameter("password").trim();
    boolean login=database.adminLogin(username,password);    //查看是否通过验证
    if(login){
        vec=database.getAllProduct();
        session.setAttribute("showresult",vec);
        session.setAttribute("adminname",username);
        this.forward(request,response,"showadddelete.jsp");
    }else{
        session.setAttribute("notlogin","loginfail");
        this.forward(request,response,"login.jsp");
    }
} else if(action.equals("delete")){    //用户按下了“删除”链接
...
}
```

11. 开发JSP页面。

(1) 创建JSP文件head.jsp, 以及left.jsp、tail.jsp、searchview.jsp、detail.jsp、cartview.jsp、welcome.jsp、checkout.jsp、login.jsp、showadddelete.jsp、addproduct.jsp等程序中使用的JSP文件。

(2) 添加head.jsp的代码如下：

```
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>
<script language="javascript">
    function openwincart(){
        window.open("cartview.jsp","_blank");
    }
    function openwinlogin(){
        window.open("login.jsp","_blank");
    }
</script>
<body>
    
    <table>
        <tr>
            <td align="justify"><a href="CustomerServlet?action=getall">CD浏览</a></td>
            <td align="justify"><a href="CustomerServlet?action=newest">最新碟报</a></td>
            <td align="justify"><a href="#" onclick="javascript:openwincart()">我的购物车</a></td>
            <td align="justify"><a href="#" onclick="javascript:openwinlogin()">管理员登录</a></td>
        </tr>
    </table>
</body>
</html>
```



```
</table>
</body>
</html>
```

12. 修改web.xml文件，将欢迎文件设置为“welcome.jsp”。

程序源代码与解释

可在华信教育资源网上下载此案例源代码。



综合案例4：航空查询订票系统

案例运行效果与操作

本案例实现了简单的航空查询订票系统，功能有系统管理、订票、查询、退票等，程序运行后，界面如图10-34所示。



图10-34 欢迎界面

菜单项有“文件”、“操作”和“帮助”，其功能如图10-35所示。



图10-35 菜单功能

单击“管理员登录”后，出现如图10-36所示的界面，输入正确的账号和密码后，显示登录成功，如图10-37所示。



图10-36 管理员登录

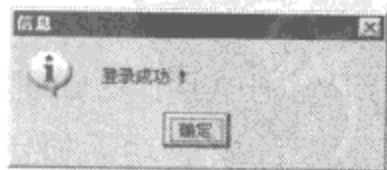


图10-37 登录成功

单击“操作”菜单中的“查询”后，出现的查询界面如图10-38所示。普通查询可按航班号、航空公司或出发城市和目的地查询，单击下拉式按钮可出现已有的数据信息，免去了用户的输入。选择后，单击“查询”按钮，查询结果如图10-39所示。

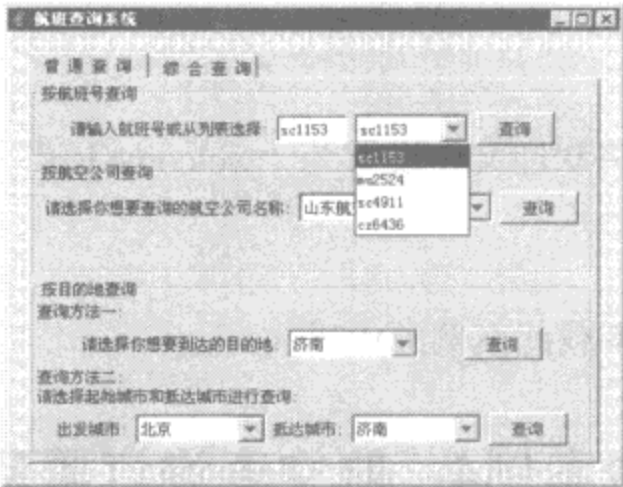


图10-38 普通查询界面

查询结果									
航空公司查询查询的航空公司: 山东航空公司									
航班号	航空公司	起飞地点	抵达地点	起飞时间	抵达时间	儿童票价	成人票价	折扣	班期
sc1153	山东航空公司	济南	北京	09:20	10:10	100.0	150.0	0.98	1357
sc4911	山东航空公司	厦门	上海	11:10	13:20	123.0	321.0	0.81	142

图10-39 查询结果

单击“综合查询”选项卡，如图10-40所示，可以按出发城市、到达城市、出发日期和航空公司进行综合查询，输入后，单击“查询”按钮，结果如图10-41所示。

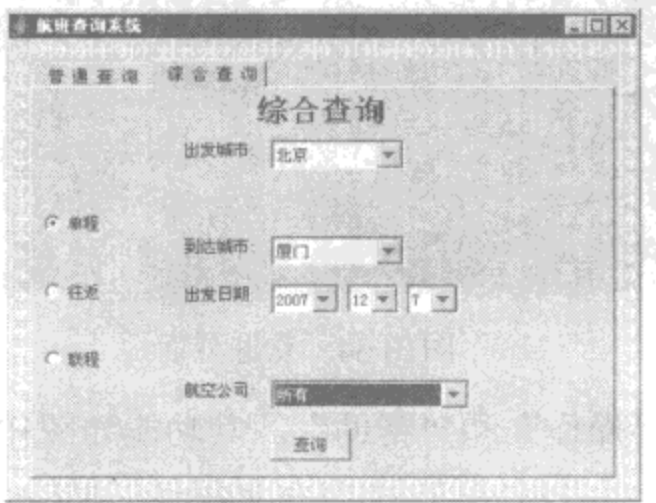


图10-40 综合查询界面

查询结果									
综合查询									
航程: 2007年12月7日(星期五) 北京——厦门									
航班号	航空公司	起飞地点	抵达地点	起飞时间	抵达时间	儿童票价	成人票价	折扣	班期
对不起,找不到你想要的航班信息!									

图10-41 综合查询结果

在订票系统中选择出发日期和航班号之后，需要填写具体的个人信息和票数，如图10-42和图10-43所示，订票成功后显示的界面如图10-44所示。

在退票系统中输入订单号、身份证号和退票数量，如图10-45所示，也可以输入订单号和身份证号码进行查询，退票要收取一定的退票费用。退票成功界面如图10-46所示。

本系统的管理功能有：插入、删除、更新和查看数据库，包括各项数据的更新、航班增减、价格调整、航线变动等，如图10-47和图10-48所示。

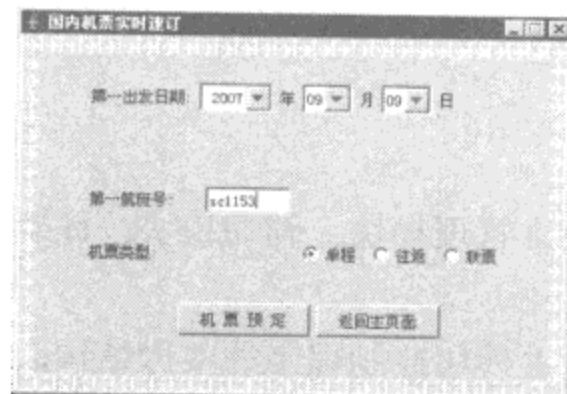


图10-42 订票系统

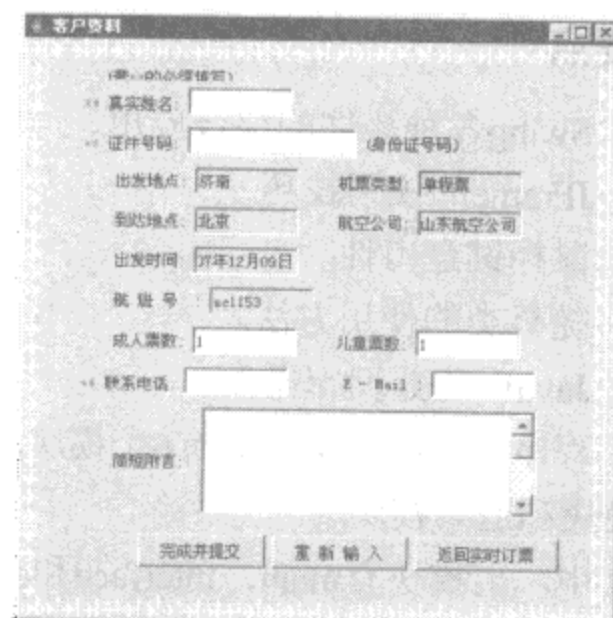


图10-43 个人资料录入

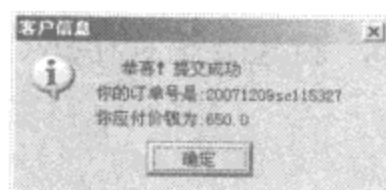


图10-44 订票成功

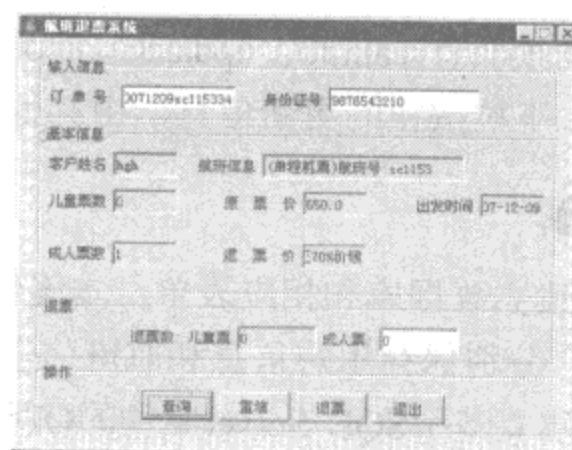


图10-45 退票界面

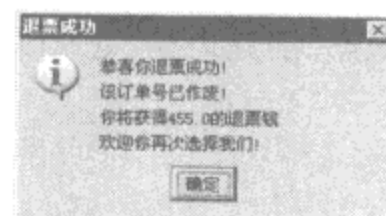


图10-46 退票成功界面



图10-47 更新界面

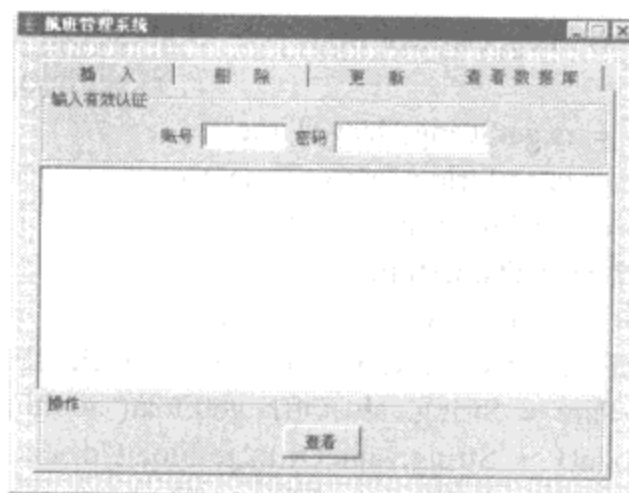


图10-48 查看数据库界面

制作要点

1. Swing各种控件的综合应用。
2. JFrame的使用技巧。
3. 鼠标键盘事件。
4. 随机类的使用方法。
5. Java中的数据库操作。
6. 程序设计：本案例共有6个模块，分别为主模块、管理模块、辅助模块、订票模块、查询模块和退票模块。

其中，主模块有main, interface和checkid 3个类，主模块控制程序的主体结构，负责程序的起、承、转作用，以及对管理员的登录进行验证。

管理模块实现程序的管理功能，包括数据的插入、删除、更新、查看等，有5个类。

辅助模块有seatinfo、updateCombobox和sqlBean3个类，主要负责数据库的连接、航班座位信息和下拉按钮的更新信息，由其他模块调用。

步骤详解

1. 建立数据库和相应表单。本案例使用SQL Server 2000，新建数据库flight、表admin和flight，分别保存用户信息和航班信息。
2. 建立基于SQL Server驱动的ODBC数据源。
3. 新建Java应用项目flight，为该项目建立5个Java包。
4. 为查询模块AirFirmQuery类的showResult方法添加代码，这个方法返回航空公司查询的结果：

```
public void showResult(ResultSet rs) {
    String result = "                " + "航空公司查询" + "\n";
    result += "查询的航空公司:" + airfirm + "\n";
    result += "航班号    航空公司                起飞地点  抵达地点  起飞时间  抵达时"
    间 " + "儿童票价  成人票价  折扣  班期 " + "\n";
    int originLength = result.length();
    String time1,time2;
    String childFare,adultFare,discount1,discount2,seat;
    try{
        while(rs.next()){
            result += rs.getString("flight") + rs.getString("airfirm") + rs.getString("start") +
                    rs.getString("destination");

            time1 = rs.getString("leaveTime");
            time2 = rs.getString("arriveTime");
            time1 = getTime(time1);
            time2 = getTime(time2);
            result += time1 + "    " + time2 + "    ";//更改时间类型
            childFare = String.valueOf(rs.getFloat("childFare"));
            adultFare = String.valueOf(rs.getFloat("adultFare"));
            discount1 = String.valueOf(rs.getFloat("discount1"));
            discount2 = String.valueOf(rs.getFloat("discount2"));
            seat = String.valueOf(rs.getInt("seat"));
            while(childFare.length() != 11)
```



```

        childFare += " ";
        while(adultFare.length() != 11)
            adultFare += " ";
        while(discount1.length() != 8)
            discount1 += " ";
        result += childFare + adultFare + discount1 + rs.getString("week");
        result += "\n";
    }
    catch(SQLException e) {
        e.printStackTrace();
    }
    if (result.length() == originLength) {
result += "
                                " +
                                "对不起,找不到你想要的航班信息!" + "\n";
    }
    JOptionPane.showMessageDialog(null,result,"查询结果",JOptionPane.PLAIN_MESSAGE);
}

```

5. `sqlBean`类包含了数据库的主要操作,有查询、插入、删除、更新和连接的关闭等,为其添加代码如下:

```

public class SqlBean
{
    private Connection conn=null;
    private ResultSet rs=null;
    private String DatabaseDriver="sun.jdbc.odbc.JdbcOdbcDriver";    //数据驱动
    private String DatabaseConnStr="jdbc:odbc:dsStudent";    //数据源
    private String LogId="sa";    //登录用户名
    private String LogPass="";    //登录密码
    public SqlBean(){//Constructor method
        try{
            Class.forName(DatabaseDriver);
        }
        catch(ClassNotFoundException e) {}
    }
    public ResultSet executeQuery(String sql) { //查询操作
        rs=null ;
        try{
            conn = DriverManager.getConnection(DatabaseConnStr,LogId,LogPass);
            Statement stmt=conn.createStatement();
            rs=stmt.executeQuery(sql);
        }
        catch(SQLException ex) {}
        return rs;
    }
    public int executeInsert(String sql) { //插入操作
        int num=0;
        try{
            conn = DriverManager.getConnection(DatabaseConnStr,LogId,LogPass);
            Statement stmt=conn.createStatement();
            num=stmt.executeUpdate(sql);
        }
        catch(SQLException ex) {}
        return num;
    }
}

```



```

    }
    public int executeDelete(String sql) {          //删除操作
        int num=0;
        try{
            conn = DriverManager.getConnection(DatabaseConnStr,LogId,LogPass);
            Statement stmt=conn.createStatement();
            num=stmt.executeUpdate(sql);
        }
        catch(SQLException e) {}
        return num;
    }
    public int executeUpdate(String sql) {          //更新操作
        int num=0;
        try{
            conn = DriverManager.getConnection(DatabaseConnStr,LogId,LogPass);
            Statement stmt=conn.createStatement();
            num=stmt.executeUpdate(sql);
        }
        catch(SQLException e){ }
        return num;
    }
    public void CloseDataBase(){//关闭
        try{
            conn.close();
        }
        catch(Exception e) {}
    }
}

```

6. 订单号随机生成:

```
String dingdan=string[3]+string[4]+String.valueOf((int)(100*Math.random()));
```

7. 订票成功的事件处理:

```

try
{
    Connection con = DriverManager.getConnection("jdbc:odbc:dsStudent","sa","");
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("select adultFare,childFare from flight where flight='"+
string[4]+"");

    while(rs.next()){
        adultfare=rs.getFloat(1);
        childfare=rs.getFloat(2);
    }
    catch(Exception ex) {}
    piaojia=adultnumber*adultfare+childnumber*childfare;
    try{
        Connection con = DriverManager.getConnection("jdbc:odbc:dsStudent","sa","yuanran");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("select adultFare,childFare from flight where flight='"+
string[11]+"");

        while(rs.next()){
            adultfare=rs.getFloat(1);

```



```

        childfare=rs.getFloat(2);
    }}
    catch(Exception ex) {}
    piaojia=piaojia+adultnumber*adultfare+childnumber*childfare;
    String dingdan=string[3]+string[4]+String.valueOf((int)(100*Math.random()));
    string[16]=dingdan;
    string[21]=""+piaojia;
    JOptionPane.showMessageDialog(this,"    恭喜！提交成功\n你的订单号是:"+dingdan+"\n"+"
你应付价钱为:"+piaojia, "客户信息",JOptionPane.INFORMATION_MESSAGE);
    Hangkong.clientFrame.setVisible(false);
    Hangkong.clientFrame.dispose();
    Hangkong.frame.setVisible(true);
    writeToFile writetofile=new writeToFile(string);
}
...
    .else{
        if(i!=-1) {
            JOptionPane.showMessageDialog(this,"非常抱歉！只有"+i+"张第一航班票剩余\n请您重新
选择票数", "客户信息",JOptionPane.INFORMATION_MESSAGE);
            jbtadulthoodticketnumber.setText(" ");
            jbtchildticketnumber.setText(" ");
        }
        else{
            if(j!=-1) {
                JOptionPane.showMessageDialog(this,"非常抱歉！只有"+j+"张返回航班票剩余\n请
您重新选择票数", "客户信息",JOptionPane.INFORMATION_MESSAGE);
                jbtadulthoodticketnumber.setText(" ");
                jbtchildticketnumber.setText(" ");
            }
        }
    }
}

```

8. 其他代码添加与解释就不再多讲，详见程序源代码。

程序源代码与解释

可在华信教育资源网上下载此案例源代码。

本章小结

本章列举了4个Java在图形界面设计（Swing）、移动开发（Java ME）、Web应用和数据库方面的综合案例，体现了Java知识在编程中的综合应用。作为一种程序设计语言，Java有简单、面向对象、不依赖于机器的结构特点，具有可移植性、安全性等，并且提供了并发的机制，具有很高的性能。这些都不是靠几个案例就能说清楚的，需要读者进一步学习和平时大量经验的积累。

反侵权盗版声明

电子工业出版社依法对本作品享有专有出版权。任何未经权利人书面许可，复制、销售或通过信息网络传播本作品的行为；歪曲、篡改、剽窃本作品的行为，均违反《中华人民共和国著作权法》，其行为人应承担相应的民事责任和行政责任，构成犯罪的，将被依法追究刑事责任。

为了维护市场秩序，保护权利人的合法权益，我社将依法查处和打击侵权盗版的单位和个人。欢迎社会各界人士积极举报侵权盗版行为，本社将奖励举报有功人员，并保证举报人的信息不被泄露。

举报电话：（010）88254396；（010）88258888

传 真：（010）88254397

E-mail: dbqq@phei.com.cn

通信地址：北京万寿路173信箱

电子工业出版社总编办公室

邮 编：100036

欢迎与我们联系

为了方便与我们联系，我们已开通了网站（www.medias.com.cn）。您可以在本网站上了解我们的新书介绍，并可通过读者留言簿直接与我们沟通，欢迎您向我们提出您的想法和建议。也可以通过电话与我们联系：

电话号码：（010）68252397

邮件地址：webmaster@medias.com.cn