

原创经典，程序员典藏

涵盖Java语言6大技术要点，详解Java语法的最新特性
精选25个典型模块和4个项目案例，实战Java应用开发

Java

典型模块与 项目实战大全

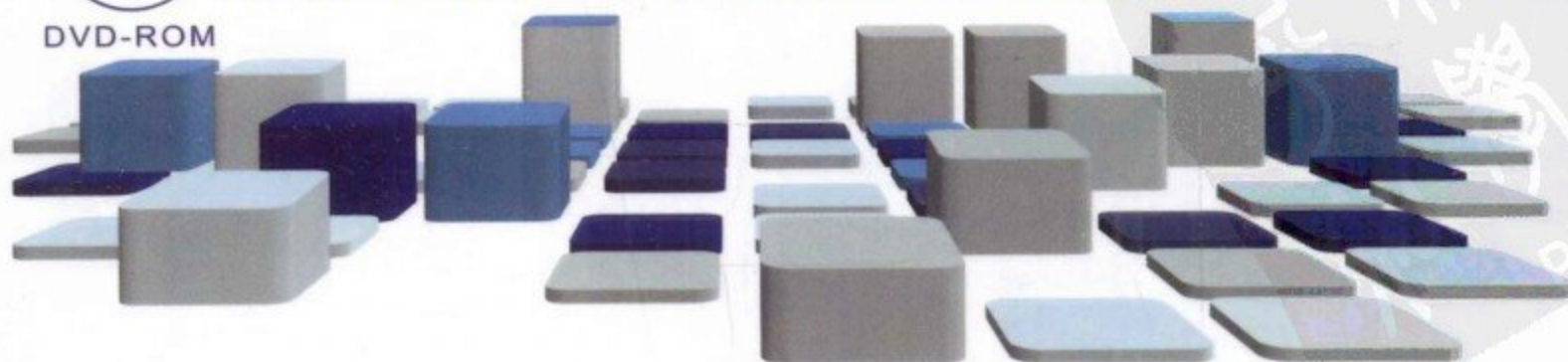
(32小时多媒体教学视频)

周华清 李为民 张昌龙 等编著



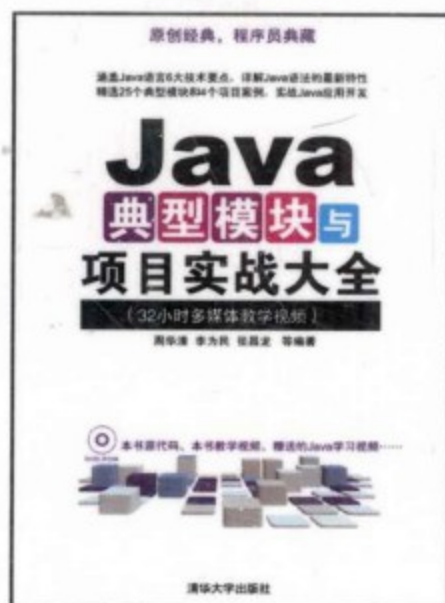
DVD-ROM

本书源代码、本书教学视频、赠送的Java学习视频……

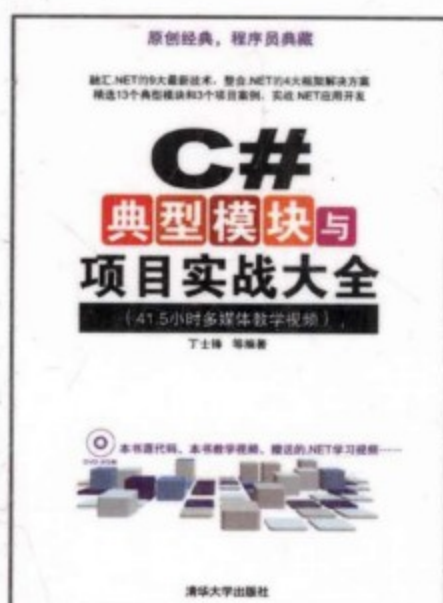


清华大学出版社

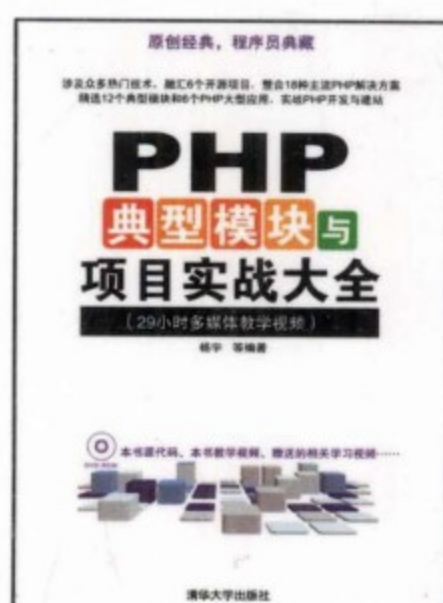
一线开发人员全力打造，分享技术盛宴！



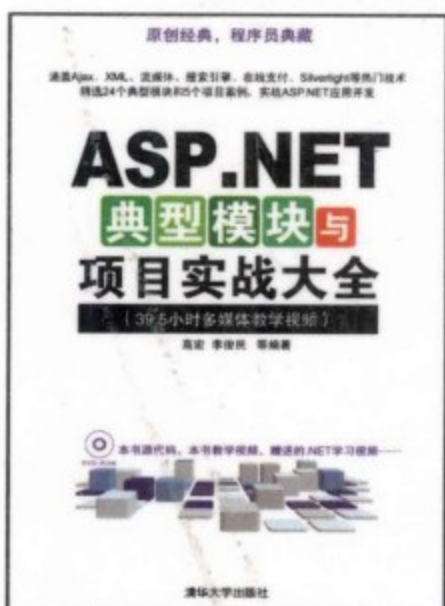
ISBN 978-7-302-26152-0
9 787302 261520 >
定价：89.00元（附1张DVD）



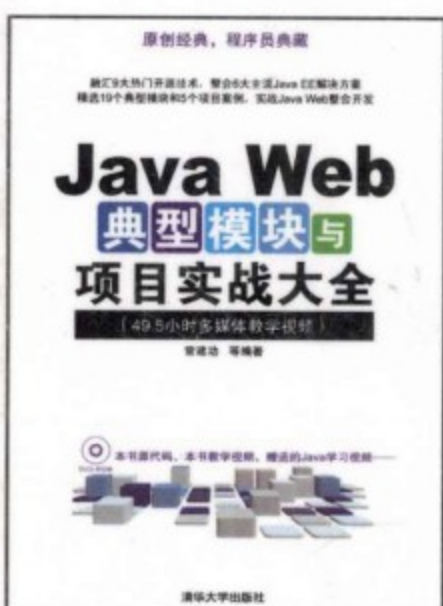
ISBN 978-7-302-26154-4
9 787302 261544 >
定价：89.00元（附1张DVD）



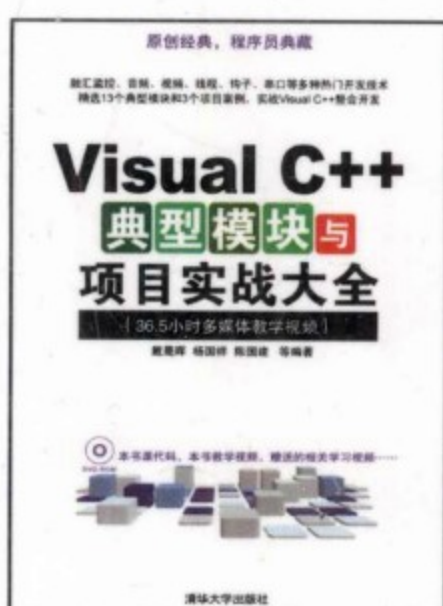
ISBN 978-7-302-25821-6
9 787302 258216 >
定价：79.00元（附1张DVD）



ISBN 978-7-302-25876-6
9 787302 258766 >
定价：89.00元（附1张DVD）



ISBN 978-7-302-22589-8
9 787302 225898 >
定价：99.50元（附1张DVD）



ISBN 978-7-302-25586-4
9 787302 255864 >
定价：79.00元（附1张DVD）

- ◎ 提供极具价值的可扩展程序模块，提高开发效率
- ◎ 实战为王，展示实际项目案例的开发精髓
- ◎ 追踪最新的前沿技术，真正提高程序员的开发水平
- ◎ 应用当前流行的技术或架构，深入剖析并阐释原理
- ◎ 提供完整的源代码、配套视频和超值赠品

数字资源
PDG

原创经典，程序员典藏

Java

典型模块与 项目实战大全

(32小时多媒体教学视频)

周华清 李为民 张昌龙 等编著



清华大学出版社

北 京

内 容 简 介

本书以实战开发为原则,以 Java 热门开发技术与项目案例开发为主线,通过 Java 开发中最常见的 25 个典型模块和 4 个完整的项目案例,详细介绍 Java 语言的特性、线程开发、图形用户开发(GUI)、文件 I/O 操作、Applet 程序、网络编程等知识。

本书附带 1 张 DVD 光盘,内容为与本书配套的多媒体教学视频与源代码,以及免费赠送的 Java 开发教学视频等资料。

本书共 32 章,分为 7 篇。涵盖的主要内容有搭建 Java 开发环境、Java 面向对象编程、Java 新特性、学生并发接水、模拟做饭场景、火车站售票系统、生产者与消费者问题、关机工具、典型的图形用户界面、计算器、秒表、捉迷藏游戏、鼠标绘直线、指针时钟项目、控制动画项目、记事本、拼图游戏、文件属性查看器、文件内容查看器、日记簿、查找和替换项目、图像轮显动画项目、Applet 事件监听项目、动画播放项目、网络聊天室、FTP 服务器客户端、Web 服务器、QQ 聊天工具、人员信息管理项目、中国象棋游戏、俄罗斯方块游戏网络版、图书管理系统项目等。

本书注重编程思想与实际开发相结合,书中的每个技术点都配备了具有典型性和实用价值的应用开发实例,适合想要学习 Java 语言的人员阅读,尤其适合有一定 Java 语言基础和想提高开发 Java 语言经验的程序员阅读。

本书封面贴有清华大学出版社防伪标签,无标签者不得销售。

版权所有,侵权必究。侵权举报电话:010-62782989 13701121933

图书在版编目(CIP)数据

Java 典型模块与项目实战大全 / 周华清, 李为民, 张昌龙等编著. —北京: 清华大学出版社, 2012. 1

ISBN 978-7-302-26152-0

I. ①J… II. ①周… ②李… ③张… III. ①JAVA 语言—程序设计 IV. ①TP312

中国版本图书馆 CIP 数据核字(2011)第 136010 号

责任编辑:夏兆彦

责任校对:徐俊伟

责任印制:杨 艳

出版发行:清华大学出版社

地 址:北京清华大学学研大厦 A 座

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

邮 编:100084

社 总 机:010-62770175

邮 购:010-62786544

投稿与读者服务:010-62795954, jsjic@tup.tsinghua.edu.cn

质 量 反 馈:010-62772015, zhiliang@tup.tsinghua.edu.cn

印 刷 者:清华大学印刷厂

装 订 者:三河市新茂装订有限公司

经 销:全国新华书店

开 本:185×260 印 张:44.75 字 数:1117 千字

(附光盘 1 张)

版 次:2012 年 1 月第 1 版 印 次:2012 年 1 月第 1 次印刷

印 数:1~4000

定 价:89.00 元

产品编号:040087-01

前言

为什么要写这本书？

Java 是 Sun 公司推出的一种程序设计语言，拥有面向对象、便利、跨平台、分布性、高性能、可移植等优点和特性，同时也是目前被程序员使用最广泛的编程语言之一。随着智能手机的发展及 Android 系统的普及，对 Java 人才的需求也更多。所以掌握 Java 语言成了程序员最基本的能力要求之一。

目前市场上关于 Java 语言编程的书不少，入门讲解基础知识的居多。但是真正从实例出发，通过实践方式讲解知识点的书却很少。本书以线程→图形用户界面→文件的访问和操作→Applet 程序→网络编程为主线，辅以开发软件遇到的典型模块代码，让读者在学习关于 Java 语言基础知识的同时，更能快速地适应软件职场。

在学习一门语言时，希望读者能谨记：动手才是硬道理。切合这一主题，本书提供了非常实用的案例，供读者学习和研究。

本书有何特色？

1. 附带多媒体语音教学视频，提高学习效率

为了便于读者理解本书内容，提高学习效率，作者专门为本书的每一章内容都录制了大量的多媒体语音教学视频。这些视频和本书涉及的源代码一起收录于配书光盘中。

2. 涵盖Java开发的各种热门技术及应用

本书以现实职场中经典模块和完整项目系统为背景，以具体应用为目标，把 Java 语言的各种知识点分为线程、图形用户界面、文件访问和操作、Applet 程序和网络编程 5 个方面，从这些方面入手，逐步深入地讲解所涉及的知识。

3. 模块驱动，实用性强

本书采用模块驱动的模式讲解，全书给出了 25 个 Java 开发的典型模块。本书的每章都是一个完整的模块，这些模块涵盖了 Java 语言的主要热门技术和应用。通过学习这些模块的开发，可以让读者深刻体会 Java 应用开发的特点。这些模块稍加修改，即可用于实际的开发中。

4. 项目案例典型，实战性强，有较高的应用价值

本书最后一篇提供了 4 个项目实战案例，这些案例分别使用不同的技术组合实现，便

于读者融会贯通地理解本书中所介绍的技术。这些案例稍加修改，便可用于实际的项目开发中。

5. 提供完善的技术支持和售后服务

本书提供了专门的技术支持邮箱：bookservice2008@163.com。读者在阅读本书的过程中有任何疑问都可以通过该邮箱获得帮助。

本书内容及知识体系

本书共分为 7 篇，共 32 章，结合目前最新编程环境 MyEclipse，全方位介绍了关于 Java 语言基础知识的各种典型模块。本书从 Java 语言的开发环境搭建和新特性讲起，然后进一步详细介绍 Java 语言开发的典型模块，最后结合笔者的实际项目经验详细讲解了 4 个完整项目。这些内容涵盖了 Java 开发中的 5 大经典应用。

第1篇 Java开发必备基础（第1～3章）

本篇主要介绍了关于 Java 语言的一些概念和语法。首先介绍了 Java 语言的一些基础知识，然后详细讲解了关于 Java 语言开发环境的搭建及如何创建、编译、运行 Java 程序；接着介绍了面向对象思想；最后演示了 Java 语言中的高级特性，如静态导入功能、可变参数函数、增强版 for 循环、基本数据的拆装箱操作、枚举语法、反射语法、标注语法、泛型语法、类加载器和动态代理。

第2篇 线程开发（第4～8章）

本篇主要介绍了 Java 语言中最常见的 5 个线程机制的典型模块。主要包括：通过线程类 Thread 模拟“学生并发接水”场景；通过线程类 Thread 的 join() 方法模拟“做饭”场景；通过线程的安全知识实现“火车站售票系统”；通过线程的通信知识实现“生产者与消费者问题”；通过线程类 Timer 模拟 Windows 操作系统的关机工具。

第3篇 图形用户界面开发（第9～17章）

本篇主要介绍 Java 语言中最常见的 9 个图形用户界面的典型模块，包括通过图形用户界面中的各种 AWT 组件，实现各种典型图形用户界面；通过图形用户界面中的布局管理器对象，实现计算器功能；通过图形用户界面中的事件机制和 Java 语言中的线程机制，实现秒表功能；通过事件机制中的自定义事件，实现捉迷藏游戏；通过图形用户界面中的绘图机制和 Java 语言中的线程机制，实现鼠标绘直线的功能；通过图形用户界面中的事件机制，实现指针时钟项目；通过 Swing 的滑动杆 JSlider 组件，实现控制动画项目；通过 Swing 的各种文件对话框组件，模拟 Windows 系统中的记事本功能，以及综合使用图形用户界面中的各种组件和机制，实现拼图游戏项目。

第4篇 文件操作和访问（第18～21章）

本篇主要介绍了 Java 语言中最常见的 4 个文件操作及访问的典型模块。包括通过文件

操作知识和图形用户界面中的各种组件，实现文件属性查看器项目；通过文件访问知识和图形用户界面中的各种组件，实现文件内容查看器项目；通过文件操作访问知识及图形用户界面中的各种组件，实现日记簿项目；通过字符串处理知识和图形用户界面中的各种组件，模拟 Windows 系统中的“查找和替换”功能。

第5篇 Applet程序开发（第22～24章）

本篇主要介绍了 Java 语言中最常见的 3 个 Applet 程序的典型模块。包括通过 Applet 程序中的显示图像和线程机制，实现图像轮显动画项目；通过 Applet 程序中的事件处理机制，实现鼠标和键盘中的按键监听项目；通过 Applet 程序中的音频操作及多线程机制，实现动画播放项目。

第6篇 网络编程（第25～28章）

本篇主要介绍 Java 语言中最常见的 4 个网络程序的典型模块，包括通过网络程序中的 UDP 协议和多线程机制，实现网络聊天室项目；通过网络程序中的 FTPClient 类和文件访问，实现 FTP 服务器客户端；通过网络程序中的 HTTP 协议，实现 Web 服务器；通过综合网络程序知识，实现 QQ 项目。

第7篇 项目案例实战（第29～32章）

本篇主要介绍 4 个项目案例的开发过程，主要包括人员信息管理项目、中国象棋游戏、俄罗斯方块网络游戏和图书管理系统项目。通过本篇内容的学习，可以大大提高读者的 Java 项目开发水平，解决实际项目开发中的种种难题。

配书光盘内容介绍

为了方便读者学习，本书附带 1 张 DVD 光盘。内容如下：

- ☐ 本书所有实例的源代码；
- ☐ 本书每章内容的多媒体语音教学视频；
- ☐ 免费赠送的 Java 语言开发教学视频及相关电子书。

适合阅读本书的读者

- ☐ 有 Java 基础，需要提高实际开发水平的人员；
- ☐ Java 项目经验不足的行业新手；
- ☐ 希望掌握 Java 典型模块开发的人员；
- ☐ Java 软件工程师；
- ☐ Java 项目开发人员；
- ☐ 专业培训机构的学员；
- ☐ 软件开发项目经理；
- ☐ 需要一本案头必备查询手册的人员。

阅读本书的建议

- 没有 Java 语言基础的读者，建议从第 1 章顺次阅读并演练每一个实例。
- 有一定 Java 语言基础的读者，可以根据实际情况有重点地选择阅读各个模块和项目案例。
- 对于每一个模块和项目案例，先自己思考一下实现的思路，然后再阅读，这样学习效果更好。
- 可以先对书中的模块和项目案例阅读一遍，然后结合光盘中提供的多媒体教学视频再理解一遍，这样理解起来就更加容易，也会更加深刻。

本书作者及编委会成员

本书第 1~15 章主要由东华理工大学的周华清编写，第 16~28 章主要由空军工程大学的李为民编写，第 29~32 章主要由空军工程大学的张昌龙编写。其他参与编写和资料整理的人员有王征、王石、姜海英、邵毅、张路平、李臻、武勇、徐宁、刘玉珊、麻雪、齐晓宁、范永龙、赵盟、傅靖、李佳、刘丹、肖冰、王行恒、冯浩楠、纪超、段桂东、黄宝生、张珍珍、石淑珍、陈超、牛晓辉、刘聪、任潇、张双、于志华、李秀劲、李胜美、蔡文仙、杜阳阳、吴兴亮、陈水望、黄任桢、梅婷婷、皇波、白雪蛟。在此一并表示感谢！


本书编委会成员有欧振旭、陈杰、陈冠军、项宇峰、张帆、陈刚、程彩红、毛红娟、聂庆亮、王志娟、武文娟、颜盟盟、姚志娟、尹继平、张昆、张薛。

编著者

目 录



第 1 篇 Java 开发必备基础



第 1 章 搭建 Java 开发环境.....	2
📺 教学视频：18 分钟	
1.1 Java 的过去、现在和未来.....	2
1.1.1 Java 的历史.....	2
1.1.2 Java 的语言特点.....	3
1.1.3 Java API 简介.....	3
1.1.4 Java 未来发展.....	4
1.2 Java 程序设计环境.....	5
1.2.1 命令行工具——JDK 6.0.....	5
1.2.2 安装工具包 JDK.....	6
1.2.3 设置 JDK 环境.....	8
1.2.4 集成开发环境安装——MyEclipse 8.5.....	10
1.2.5 MyEclipse 的一些常用操作.....	11
1.3 创建和运行 Java 程序.....	14
1.3.1 手工创建、编译和运行 Java 程序.....	14
1.3.2 在 MyEclipse 8.5 中创建、运行、调试和管理 Java 项目.....	15
1.4 小结.....	16
第 2 章 Java 面向对象编程.....	17
📺 教学视频：13 分钟	
2.1 面向对象的一些概念.....	17
2.1.1 面向对象涉及的概念.....	17
2.1.2 类和对象.....	19
2.2 面向对象的一些特性.....	20
2.2.1 继承特性.....	20
2.2.2 多态特性.....	21
2.2.3 封装特性.....	22
2.3 Java 中实现的面向对象特性.....	22
2.3.1 定义类.....	22
2.3.2 创建对象.....	23
2.3.3 实现继承.....	23

2.3.4 实现多态	24
2.3.5 实现封装	26
2.4 小结	27
第 3 章 Java 新特性	28
 教学视频：29 分钟	
3.1 Java 的一些简单新特性	28
3.1.1 静态导入	28
3.1.2 可变参数函数	30
3.1.3 增强版 for 循环	31
3.1.4 基本数据的拆、装箱操作（autoboxing 和 unboxing）	32
3.2 枚举	34
3.2.1 枚举的实现原理	34
3.2.2 枚举的简单应用	36
3.2.3 枚举的高级特性	37
3.3 反射	39
3.3.1 反射的基石——Class 类	39
3.3.2 反射的基本应用	41
3.3.3 反射的高级应用	46
3.4 标注	49
3.4.1 标注的简单使用	49
3.4.2 JDK 的内置标注	51
3.5 泛型	53
3.5.1 为什么要使用泛型	54
3.5.2 泛型的一些特性	57
3.5.3 泛型的通配符	59
3.6 类加载器	62
3.6.1 什么是类加载器	63
3.6.2 什么是类加载器的委派模型	64
3.6.3 编写一个自己的加载器	67
3.7 动态代理	73
3.7.1 什么是代理	73
3.7.2 动态代理基础类	74
3.7.3 InvocationHandler 接口	78
3.7.4 动态代理类的设计模式	81
3.8 小结	83

第 2 篇 线程开发



第 4 章 学生并发接水（线程 Thread）	86
 教学视频：9 分钟	
4.1 学生并发接水原理	86




4.1.1	项目结构框架分析	86
4.1.2	项目功能业务分析	86
4.2	不排队形式学生并发接水	87
4.2.1	水龙头类	88
4.2.2	学生类	88
4.2.3	测试类	89
4.3	学生并发接水的其他形式	90
4.3.1	“排队接水”水龙头类	90
4.3.2	“接完水后一起回教室”水龙头类	90
4.4	知识点扩展——线程的基础知识	91
4.4.1	为什么要使用线程	92
4.4.2	多线程程序的编写方式	93
4.5	小结	98
第 5 章	模拟做饭场景（线程的 join() 方法）	99
	教学视频：7 分钟	
5.1	做饭场景原理	99
5.1.1	项目结构框架分析	99
5.1.2	项目功能业务分析	99
5.2	纷乱的做饭场景	100
5.2.1	儿子的类	100
5.2.2	妈妈的类	101
5.2.3	做饭场景的类	102
5.2.4	修改后的妈妈类	102
5.3	知识点扩展——线程的状态	103
5.3.1	线程的创建状态	103
5.3.2	线程的暂停状态	104
5.3.3	线程的结束状态	106
5.4	小结	109
第 6 章	火车站售票系统（线程安全知识）	110
	教学视频：12 分钟	
6.1	火车站售票系统原理	110
6.1.1	项目结构框架分析	110
6.1.2	项目功能业务分析	111
6.2	没有实现线程安全的火车票售票系统	112
6.2.1	火车票的类	112
6.2.2	售票台的类	113
6.2.3	实现线程安全的火车票售票系统	113
6.3	知识点扩展——线程的同步知识	115
6.3.1	为什么要使用同步机制	115
6.3.2	Synchronized 的同步块	117
6.3.3	Synchronized 的同步方法	119



6.3.4 死锁的问题	122
6.4 小结	124
第 7 章 生产者与消费者问题（线程通信知识）	125
 教学视频：10 分钟	
7.1 生产者与消费者原理	125
7.1.1 项目结构框架分析	125
7.1.2 项目功能业务分析	125
7.2 无线程通信的生产者与消费者项目	126
7.2.1 生产者类	127
7.2.2 消费者类	128
7.2.3 储存库类	128
7.2.4 测试类	129
7.3 实现线程通信的生产者与消费者项目	130
7.3.1 生产者和消费者的类	130
7.3.2 储存库的类	131
7.4 知识点扩展——线程的通信知识	132
7.4.1 线程通信的基本知识	132
7.4.2 线程通信的具体实例	134
7.5 小结	136
第 8 章 关机工具（Timer 类+系统命令）	137
 教学视频：5 分钟	
8.1 关机工具原理	137
8.1.1 项目结构框架分析	137
8.1.2 项目功能业务分析	137
8.2 关机工具的实现过程	139
8.2.1 关机工具的工具类	139
8.2.2 关机工具的工具类	143
8.3 知识点扩展——关机工具项目涉及的知识	144
8.3.1 Timer 和 TimerTask 类	144
8.3.2 shutdown 命令	147
8.3.3 通过 shutdown 命令实现网络远程关机	150
8.4 小结	152


第 3 篇 GUI（图形用户界面）开发

第 9 章 典型的图形用户界面（各种组件）	154
 教学视频：15 分钟	
9.1 Label 和 Button 的用户界面	154
9.1.1 分析按钮和面板的用户界面	154
9.1.2 按钮和面板的用户界面	155



9.1.3	组件 Button 和 Label 的基本知识	156
9.2	复选框的用户界面	157
9.2.1	分析复选框的用户界面	157
9.2.2	按钮和面板的用户界面	158
9.2.3	组件 Checkbox 和 CheckboxGroup 的基本知识	159
9.3	下拉菜单和列表的用户界面	160
9.3.1	分析下拉菜单和列表的用户界面	161
9.3.2	下拉菜单和列表的用户界面	162
9.3.3	Choice 和 List 组件的基本知识	164
9.4	输入的用户界面	165
9.4.1	分析输入的用户界面	165
9.4.2	输入的用户界面	167
9.4.3	TextField 和 TextArea 组件的基本知识	169
9.5	滚动条的用户界面	171
9.5.1	分析滚动条的用户界面	171
9.5.2	滚动条的用户界面	172
9.5.3	滚动组件的基本知识	174
9.6	菜单的用户界面	175
9.6.1	分析菜单组件的用户界面	175
9.6.2	菜单的用户界面	177
9.6.3	菜单组件的基本知识	179
9.7	对话框的用户界面	182
9.7.1	分析对话框和文件对话框的用户界面	182
9.7.2	对话框的用户界面	184
9.7.3	Dialog 和 FileDialog 组件的基本知识	187
9.8	小结	188
第 10 章	计算器（布局管理器）	189
	 教学视频：5 分钟	
10.1	计算器原理	189
10.1.1	项目结构框架分析	189
10.1.2	项目功能业务分析	189
10.2	计算器的实现过程	191
10.3	知识点扩展——事件机制的高级知识	193
10.3.1	为什么需要版面的配置	194
10.3.2	Java 语言中的各种布局管理器	195
10.4	小结	201
第 11 章	秒表（事件+线程）	202
	 教学视频：7 分钟	
11.1	秒表原理	202
11.1.1	项目结构框架分析	202
11.1.2	项目功能业务分析	202



11.2	秒表的实现过程	203
11.2.1	秒表类	203
11.2.2	测试秒表的类	206
11.3	知识点扩展——事件机制的基础知识	206
11.3.1	事件处理机制	206
11.3.2	Window 事件	208
11.3.3	Mouse 事件	210
11.3.4	Key 事件	213
11.3.5	其他底层事件	216
11.3.6	事件的高级编写方法	217
11.4	小结	219
第 12 章	捉迷藏游戏（事件）	220
	教学视频：5 分钟	
12.1	捉迷藏游戏原理	220
12.1.1	项目结构框架分析	220
12.1.2	项目功能业务分析	220
12.2	捉迷藏游戏的实现过程	221
12.2.1	捉迷藏游戏项目的原理	221
12.2.2	自定义按钮类	222
12.2.3	测试的类	223
12.3	知识点扩展——事件机制的高级知识	223
12.3.1	事件多重应用	223
12.3.2	事件处理的详细过程	228
12.4	小结	230
第 13 章	鼠标绘直线（绘图+事件）	231
	教学视频：8 分钟	
13.1	鼠标绘直线原理	231
13.1.1	项目结构框架分析	231
13.1.2	项目功能业务分析	231
13.2	鼠标绘直线的实现过程	233
13.2.1	直线的类	233
13.2.2	实现窗口类——通过 paint() 方法	234
13.2.3	实现窗口类——通过双缓冲技术	236
13.3	知识点扩展——画图的基础知识	237
13.3.1	画图的基础知识	237
13.3.2	各种类型对象的绘制	239
13.4	小结	245
第 14 章	指针时钟项目（Swing 组件+时间算法）	246
	教学视频：7 分钟	
14.1	指针时钟原理	246
14.1.1	项目结构框架分析	246

14.1.2	项目功能业务分析	247
14.2	指针时钟的实现过程	247
14.2.1	指针时钟的界面	247
14.2.2	绘制指针时钟的类	249
14.3	知识点扩展——从 AWT 到 Swing 的过渡	253
14.3.1	窗口类 JFrame	253
14.3.2	按钮类 JButton 和面板类 JLabel	254
14.3.3	单选按钮和复选框组件	258
14.3.4	选择框组件	261
14.3.5	输入框组件	263
14.4	小结	265
第 15 章	控制动画项目 (JSlider 和 Timer 组件)	266
	教学视频: 7 分钟	
15.1	控制动画原理	266
15.1.1	项目结构框架分析	266
15.1.2	项目功能业务分析	267
15.2	控制动画的实现过程	267
15.2.1	控制动画的主界面	267
15.2.2	控制动画的逻辑	269
15.3	知识点扩展——JSlider 和 Timer 组件的基础知识	272
15.3.1	使用 JSlider 组件创建无刻度的滑杆	272
15.3.2	使用 JSlider 组件创建带数字刻度的滑杆	275
15.3.3	使用 JSlider 组件创建带字符刻度的滑杆	277
15.3.4	JSlider 组件的高级应用	279
15.3.5	Swing 中的多线程	283
15.3.6	Timer 组件的基础知识	288
15.3.7	Timer 组件的应用	289
15.4	小结	293
第 16 章	记事本 (对话框组件)	294
	教学视频: 54 分钟	
16.1	记事本原理	294
16.1.1	项目结构框架分析	294
16.1.2	项目功能业务分析	294
16.2	记事本的实现过程	300
16.2.1	实现记事本的界面	300
16.2.2	实现菜单功能	303
16.2.3	文件类型的过滤	306
16.3	记事本的实现过程——字体设置对话框	307
16.3.1	字体设置对话框——主界面	307
16.3.2	字体设置对话框——JPanel1 组件界面	310

16.3.3 字体设置对话框——其他组件	312
16.4 小结	313
第 17 章 拼图游戏 (GUI 综合应用)	314
 教学视频: 20 分钟	
17.1 拼图游戏原理	314
17.1.1 项目结构框架分析	314
17.1.2 项目功能业务分析	314
17.1.3 拼图游戏项目的原理	316
17.2 拼图游戏的实现过程	317
17.2.1 实现移动功能的按钮类	317
17.2.2 主面板的类	318
17.2.3 主窗口的类	320
17.3 小结	322

第 4 篇 文件操作和访问

第 18 章 文件属性查看器 (GUI+文件操作)	324
 教学视频: 6 分钟	
18.1 文件属性查看器原理	324
18.1.1 项目结构框架分析	324
18.1.2 项目功能业务分析	324
18.2 文件属性查看器项目	326
18.2.1 实现显示文件信息的自定义窗口	326
18.2.2 自定义窗口的显示	329
18.3 知识点扩展——文件的操作和访问	329
18.3.1 通过 FileOp 类实现文件创建和删除功能	329
18.3.2 通过 FileDir 类实现列举文件和目录的功能	331
18.3.3 File 类提供的属性和方法	332
18.3.4 文件访问的基本概念	334
18.3.5 文件的基本访问方式——字节方式	336
18.3.6 文件的基本访问方式——字符方式	338
18.3.7 文件的高级访问方式	339
18.4 小结	342
第 19 章 文件内容查看器 (GUI+文件访问)	343
 教学视频: 6 分钟	
19.1 文件内容查看器原理	343
19.1.1 项目结构框架分析	343
19.1.2 项目功能业务分析	343
19.2 文件内容查看器项目	345

19.2.1	设计项目的界面——文件内容查看器输入界面	345
19.2.2	“打开”菜单项的处理方法	348
19.2.3	单击列表选项的处理方法	348
19.3	知识点扩展——管道的访问	349
19.3.1	管道的访问——字节方式	350
19.3.2	管道的访问——字符方式	351
19.4	知识点扩展——内存的访问	353
19.4.1	内存的访问——字节方式	353
19.4.2	内存的访问——字符和字符串方式	354
19.5	小结	356
第 20 章	日记簿 (GUI+文件访问和操作)	357
	教学视频: 30 分钟	
20.1	日记簿原理	357
20.1.1	项目结构框架分析	357
20.1.2	项目功能业务分析	357
20.2	日记簿项目	359
20.2.1	设计项目的界面——日记簿输入界面	360
20.2.2	“保存”按钮的事件处理	363
20.2.3	“查看日记”按钮的事件处理	364
20.2.4	设计项目的界面——日记列表界面	365
20.2.5	“查看”按钮的事件处理	367
20.2.6	“删除”按钮的事件处理	368
20.3	知识点扩展——过滤流的基础知识	368
20.3.1	过滤流的缓存 (Buffering) 类	368
20.3.2	过滤流实现字节和字符相互转换类	372
20.3.3	过滤流特定数据类型类	374
20.3.4	过滤流对象序列化类	378
20.3.5	过滤流打印类	381
20.4	小结	383
第 21 章	查找和替换项目 (GUI+字符串处理)	384
	教学视频: 10 分钟	
21.1	查找和替换原理	384
21.1.1	项目结构框架分析	384
21.1.2	项目功能业务分析	384
21.2	查找和替换项目——利用 AWT 组件	386
21.2.1	设计项目的界面——查找和替换输入界面	386
21.2.2	各种按钮的事件处理	389
21.2.3	字符串处理的类	391
21.3	查找和替换项目——利用 Swing 组件	392
21.3.1	设计项目的界面——查找和替换输入界面	392

21.3.2 各种按钮的事件处理	394
21.4 小结	395

第 5 篇 Applet 程序开发

第 22 章 图像轮显动画项目（显示图像+多线程）	398
---------------------------------	-----



教学视频：5 分钟

22.1 图像轮显动画原理	398
22.1.1 项目结构框架分析	398
22.1.2 项目功能业务分析	398
22.2 图像轮显动画项目	400
22.3 知识点扩展——Applet 程序的基础知识	402
22.3.1 Applet 程序的执行过程	402
22.3.2 Applet 程序的执行环境	402
22.3.3 Applet 程序的输出	403
22.3.4 Applet 程序的标记	403
22.3.5 参数的传递	404
22.3.6 Applet 程序的相关方法	406
22.4 小结	409

第 23 章 Applet 事件监听项目（事件处理机制）	410
------------------------------------	-----



教学视频：5 分钟

23.1 Applet 事件监听原理	410
23.1.1 项目结构框架分析	410
23.1.2 项目功能业务分析	410
23.2 Applet 事件监听项目	412
23.2.1 事件监听的类	412
23.2.2 承载事件监听的页面	414
23.3 知识点扩展——MyEclipse 开发环境对 Applet 程序的支持	414
23.3.1 MyEclipse 开发环境对 Applet 项目的支持	415
23.3.2 MyEclipse 开发环境对 JAR 的支持	419
23.4 小结	422

第 24 章 动画播放项目（音频操作+多线程）	423
-------------------------------	-----




教学视频：5 分钟

24.1 动画播放原理	423
24.1.1 项目结构框架分析	423
24.1.2 项目功能业务分析	423
24.2 动画播放项目	425
24.2.1 动画的类	425

24.2.2 控制动画的类	427
24.3 知识点扩展——Applet 程序的高级知识	428
24.3.1 音频播放	428
24.3.2 Applet 的上下文对象	430
24.4 小结	431


第 6 篇 网络编程

第 25 章 网络聊天室 (UDP 协议+多线程)	434
---------------------------------	-----

 教学视频: 6 分钟


25.1 网络聊天室原理	434
25.1.1 项目结构框架分析	434
25.1.2 项目功能业务分析	434
25.2 网络聊天室的实现过程	436
25.3 知识点扩展——网络编程和 UDP 协议	438
25.3.1 网络编程涉及的基本概念	439
25.3.2 套接字 (Socket) 机制	440
25.3.3 UDP 协议类	441
25.3.4 TCP 协议类	445
25.3.5 TCP 协议客户端类	451
25.4 小结	452

第 26 章 FTP 服务器客户端 (FtpClient+I/O 处理)	453
--	-----


 教学视频: 8 分钟

26.1 FTP 服务器客户端原理	453
26.1.1 项目结构框架分析	453
26.1.2 项目功能业务分析	454
26.2 FTP 服务器客户端的实现过程	455
26.2.1 FTP 服务器操作的工具类	456
26.2.2 实现文件上传的类	459
26.2.3 实现文件下载的类	459
26.3 知识点扩展——FtpClient 类的相关知识	460
26.3.1 实现 FTP 服务器相关操作类	460
26.3.2 相关 JAR 包导入问题	462
26.4 小结	464


第 27 章 Web 服务器 (HTTP 协议)	465
--------------------------------	-----



 教学视频: 8 分钟


27.1 Web 服务器原理	465
27.1.1 项目结构框架分析	465

27.1.2	项目功能业务分析	465
27.2	Web 服务器的实现过程	466
27.2.1	实现与浏览器通信的类	466
27.2.2	实现 Web 服务器的类	469
27.2.3	浏览器所请求的页面	470
27.3	知识点扩展——HTTP 协议知识	470
27.3.1	HTTP 协议原理	470
27.3.2	实现 HTTP 协议服务器的原理	471
27.4	小结	471
第 28 章	QQ 聊天工具 (Swing+多线程+网络编程)	472
	教学视频: 39 分钟	
28.1	QQ 聊天工具原理	472
28.1.1	项目结构框架分析	472
28.1.2	项目功能业务分析	473
28.2	QQ 项目——对象模型的类	476
28.2.1	信息的类	476
28.2.2	“用户”的类	478
28.3	QQ 项目——登录功能	478
28.3.1	QQ 服务器界面的设计	479
28.3.2	QQ 服务器后台程序	480
28.3.3	QQ 客户端登录界面的设计	481
28.3.4	QQ 客户端后台程序	484
28.3.5	成员列表窗口	485
28.4	QQ 项目——聊天功能	487
28.4.1	服务器端的信息转发	488
28.4.2	客户端信息的发送和接收	490
28.4.3	客户端信息转发类	492
28.5	小结	494

第 7 篇 项目案例实战

第 29 章	人员信息管理项目 (接口设计模式+MySQL 数据库)	496
	教学视频: 30 分钟	
29.1	人员信息管理原理	496
29.1.1	项目结构框架分析	496
29.1.2	项目功能业务分析	497
29.2	人员信息管理项目前期准备	502
29.2.1	设计数据库	502
29.2.2	数据库操作相关类	504

29.3	人员信息管理项目——DAO 层	505
29.3.1	实现数据连接操作 (DAO) 的接口	506
29.3.2	实现数据连接操作 (DAO) 的实现类	507
29.3.3	实现数据连接操作 (DAO) 的代理类	510
29.3.4	实现数据连接操作 (DAO) 的工厂类	512
29.4	人员信息管理项目——服务层和表示层	512
29.4.1	人员信息管理项目的服务层	512
29.4.2	人员信息管理项目的表示层	515
29.4.3	工具类	516
29.5	人员信息管理项目——代理类测试	517
29.5.1	测试实现业务功能的各种方法	518
29.5.2	人员信息管理入口类	520
29.6	知识点扩展——设计模式的基础知识	521
29.6.1	工厂设计模式	521
29.6.2	代理设计模式	525
29.7	小结	527
第 30 章	中国象棋游戏 (GUI+游戏规则算法)	528
	教学视频: 37 分钟	
30.1	象棋游戏原理	528
30.1.1	象棋游戏的基本规则	528
30.1.2	项目结构框架分析	529
30.1.3	项目功能业务分析	529
30.2	象棋游戏项目——象棋游戏的主类	534
30.2.1	实现象棋游戏的主界面	534
30.2.2	实现象棋游戏中添加棋子的功能	537
30.2.3	实现象棋游戏中棋子闪烁的功能	539
30.2.4	处理单击棋子事件	539
30.2.5	处理单击按钮事件	543
30.3	象棋游戏项目——规则的内部类	546
30.3.1	实现卒移动和吃的方法	546
30.3.2	实现炮、车移动和吃的方法	550
30.3.3	实现马移动和吃的方法	554
30.3.4	实现象移动和吃的方法	561
30.3.5	实现士移动和吃的方法	567
30.3.6	实现将移动和吃的方法	572
30.4	小结	578
第 31 章	俄罗斯方块游戏网络版 (Swing+多线程+网络编程)	579
	教学视频: 60 分钟	
31.1	俄罗斯方块游戏项目原理	579

31.1.1	基本原理	579
31.1.2	项目结构框架分析	580
31.1.3	项目功能业务分析	580
31.2	俄罗斯方块游戏项目——初步设计涉及的对象	589
31.2.1	正方形类	589
31.2.2	俄罗斯方块类	590
31.2.3	俄罗斯方块游戏项目的 TOP10 分数对象	599
31.3	俄罗斯方块游戏项目——服务器端和客户端	602
31.3.1	表示出俄罗斯方块游戏项目的服务器端	602
31.3.2	表示出俄罗斯方块游戏项目的客户端	605
31.4	俄罗斯方块游戏项目——游戏主界面	607
31.4.1	俄罗斯方块游戏的主界面	608
31.4.2	俄罗斯方块游戏的事件处理类	612
31.4.3	俄罗斯方块游戏的状态工具栏	614
31.5	俄罗斯方块游戏项目——其他界面的设计	616
31.5.1	“关于”面板	616
31.5.2	连接对方面板	618
31.5.3	分数报告面板	620
31.5.4	设置级别面板	624
31.5.5	警告面板和对话框	625
31.5.6	游戏结束面板和对话框	627
31.6	小结	630
第 32 章	图书管理系统项目 (GUI+Oracle 数据库)	631
	教学视频: 59 分钟	
32.1	图书管理系统原理	631
32.1.1	项目结构框架分析	631
32.1.2	项目功能业务分析	631
32.2	图书管理系统项目——图书的操作	640
32.2.1	实现添加图书功能的类	640
32.2.2	实现修改图书功能的类	644
32.2.3	实现浏览图书信息的类	647
32.2.4	实现删除图书信息的类	650
32.3	图书管理系统项目——用户的操作	652
32.3.1	实现添加用户功能的类	653
32.3.2	实现删除用户功能的类	656
32.3.3	实现修改用户功能的类	658
32.3.4	实现用户登录功能的类	661
32.3.5	实现用户列表功能的类	664
32.4	图书管理系统项目——出借图书的操作	665
32.4.1	出借图书操作的类	665

32.4.2	借书列表方法	669
32.4.3	修改出借图书信息方法	672
32.5	图书管理系统项目——归还图书的操作	676
32.5.1	归还图书类	676
32.5.2	修改归还图书信息类	679
32.6	图书管理系统项目——该项目的其他类	683
32.6.1	主窗口类	683
32.6.2	数据库操作	689
32.7	小结	691

第 1 篇 *Java* 开发必备基础

- ▶▶ 第 1 章 搭建 Java 开发环境
- ▶▶ 第 2 章 Java 面向对象编程
- ▶▶ 第 3 章 Java 新特性

第 1 章 搭建 Java 开发环境

Java 这个名词得名于印度尼西亚一个盛产咖啡的岛屿，中文名为爪哇，其寓意是为世人端上一杯热咖啡。Java 语言从刚开始的默默无闻到现在的大红大紫，其中的奥妙到底是什么？Java 为什么会有这么大的魅力？本章将带领读者进入 Java 语言的世界。

本章的学习目标如下：

- 理解 Java 的历史、基本概念和发展趋势；
- 掌握 Java 环境的搭建；
- 编写简单的 Java 程序。

1.1 Java 的过去、现在和未来

在网络出现之后，全世界的目光都被 Java 语言所吸引。之所以这样，是因为 Java 语言改变了 Internet 上的信息都是一些死板的 HTML 文档的现象，使得用户在 Web 应用中能够看到一些交互的内容。为了让读者对 Java 语言有一个整体的了解，本节将详细介绍 Java 语言的历史背景、现实情况和发展趋势。

1.1.1 Java 的历史

接触过 Java 的人，一定会对 Java 的两个 logo 非常熟悉，一个是 Java Cup，另一个是名叫 Duke 的吉祥物。对于扮演了类似于 Office 2000 助手的 Duke 吉祥物，是在 1992 年由 Joe Palrang 创造出来。而对于 Java Cup，如果想了解它的具体寓意，则必须追溯到 1990 年。

在 1990 年 12 月，为了能够开发出一种基于未来智能设备的新编程语言，Sun 公司决定由 Patrick Naughton、Mike Sheridan 和 James Gosling 组建一个名为 Green Team 的小组。该小组的 James Gosling 成员由于对 C++ 执行过程的表现非常不满意，所以把自己封闭在办公室里编写了一种新的语言，并将其命名为 Oak（即 Java 语言的前身）。之所以起名为 Oak，是因为 James Gosling 办公室的窗外，正好有一棵 Oak（橡树）。

那么 Oak 为什么会改名为 Java 呢？这是因为当时去注册 Oak 商标时，发现其已经被其他公司注册，于是就不得不更换一个新名字。在当时那个年代，工程师们经常边喝咖啡边讨论着，看着手上的咖啡再联想到印度尼西亚有一个重要的盛产咖啡的岛屿（爪哇），突然灵机一动，就改名为 Java。

作为一种非常优秀的语言产品，Java 在当时的消费市场上却不被接受，直到全世界第一个万维网浏览器（Mosaic）出现后，其才以优异的功能开辟了另一片天地。在 1995 年 5 月 23 日，JDK（Java Development Kit）1.0a2 版本也正式对外发布，这一天也就成为了 Java

的生日。

1.1.2 Java 的语言特点

了解了 Java 的历史背景后,本节将具体讲解 Java 语言的特点。每个程序员都知道,Java 语言具有简单、一次编写处处运行、健壮、分布性、多线程机制和垃圾回收机制等特性。它们的具体含义如下。

1. 简单

与其他面向对象语言相比,Java 语言更具有纯面向对象的特性,例如,Java 用接口取代了多重继承并取消了指针等。该特性不仅使得开发各式各样的应用程序易如反掌,而且还使得调试和修改程序、增加新功能等方面更加容易。

2. 一次编写处处运行

由于 Java 语言具有与体系无关的特性,因此其可以更方便地移植到网络上的不同机器中。该特性不仅是 Java 程序员的精神指南,而且还是 Java 语言能够受到众多程序员喜欢的原因。

3. 健壮

为了让 Java 程序更安全、稳定,Java 语言引入了异常处理机制。所谓异常处理机制,就是在程序中可能发生异常情况的地方,加上相对应的处理,让程序不至于因为突发的错误,造成运行中断或死机的情况。

4. 分布性

Java 语言从诞生就和网络联系在一起,通过该语言,不仅可以编写出互联网的程序,如 Socket 和 E-mail,而且还可以实现服务器端程序 Servlet、JSP,甚至还支持分布式网络程序。

5. 多线程机制和垃圾回收机制

所谓多线程机制,就是能够使应用程序并行执行多项目任务。使用该机制,不仅可以用不同的线程完成特定的行为,而且还使得程序具有更好的交互能力和实时运行能力。所谓垃圾回收机制,就是把内存的动态管理(程序需要多少内存、哪些对象的内存需要归还系统等)交由 JVM 管理。使用该机制,可以使程序员专心地写程序,而不需要担心内存问题。

1.1.3 Java API 简介

自从 Sun 公司推出 Java 以后,就力图使之无所不能,因此为该语言量身订做了各种 API。通过丰富的 Java API,使得 Java 应用程序不仅能够简单、快速地完成,而且还能够在各种不同的平台上运行。本节将对这些 Java API 进行详细介绍。


在具体介绍 Java API 之前,需要先了解一些概念,它们分别为 API (Application Programming Interface, 应用程序接口)、JDK 和 SDK (Software Development Toolkit)。API 是一组由其他程序员写好的程序。如果想在程序中使用这些接口,必需遵守它们的规则。JDK 是 Java Development Kit 的简写,当 Java 1.0 和 Java 1.1 发布后,它们的 API 称为 JDK。可是在 Java 1.2 版本后,则改名为 Java 2 SDK。SDK 除了包含语言提供的 API 之外,还包含用到如编写、编译、运行、测试等工具。

目前,Java 基本技术架构包含了以下 3 方面。

(1) J2SE (Java 2 Platform Stand Edition), 是开发任何 Java 程序都需要的套件。在该套件中除了包含基本类库之外,还包含了一些编译的程序、额外的辅助工具等。主要用于桌面开放和低端商务应用的解决方案。

(2) J2EE (Java 2 Platform Enterprise Edition), 是开发企业级应用的套件。在该套件中除了包含 J2SE 中的基本类库之外,还包含了编写服务器端、分布式应用程序、事务处理等其他企业级应用程序所会用到的类库。

(3) J2ME (Java 2 Platform Micro Edition), 是开发消费性电子产品和嵌入式系统的套件。在该套件中包含的类库是 Java 平台套件中最少的。

 **注意:** 由于 J2EE 和 J2ME 中只包含类库和运行 Java 程序时的 Java 虚拟机,所以在开发这两个平台方面的程序时,还需要加入 J2SE 平台。因为只有 J2SE 平台中包含编译 Java 程序所需的工具程序。

对于 J2SE 这些基本类库,只能开发一些简单的应用程序。如果想开发一些其他应用程序,例如 3D 动画、多媒体等应用程序,则需要一些额外的辅助套件,如针对 3D 绘画的 Java 3D、针对多媒体的 JMF 和针对串口和并口的 Java Communication API 和安全方面的 JCE。

1.1.4 Java 未来发展

Java 是当今最流行的技术之一,其涉及的领域十分广阔。J2EE 和 J2ME 是 Java 最重要的两个套件。由于技术的规范化和大量 J2EE 架构师的产生,让 J2EE 的开发人群变得空前地繁荣。由于主流手机的硬件种类繁多,系统平台也不统一,比较混乱,但是都支持 Java,使得 J2ME 大展身手。

由于 Java 在消费性电子平台上具有显著的跨平台的特性,因此使得各大厂商纷纷发表了内置 Java 虚拟机系统。例如新的 Mac 操作系统内置了 Java 虚拟机系, Sun 公司为 Linux 推出 Linux 版的 JDK, IBM 和 Sun 公司为 Java 推出了 Windows 平台的解决方案,微软也推出了所谓 UMP 技术,可以把 Java 程序转换成.NET 平台的 C#程序。通过上述各大企业的支持,Java 必定会越发展越好。

最近在国外大红大紫的 Google Android 平台,更是给了 Java 一剂强心针。Android 应用程序包括 E-mail 客户端、SMS 短消息程序、日历、地图、浏览器、联系人管理程序等。所有的应用程序都是使用 Java 语言编写的,而且伴随 Google 增值业务的点缀,逐步壮大起来。Google 带着这个自己主推的移动理念,成立了包括手机制造商、电信运营商和手机配件厂商在内的 34 家企业联合组成的“开放手机联盟 (Open Handset Alliance)”,此联

盟将面向全球研发和推广这一移动设备系统平台。

说明：Android 软件平台开发效率高，又以比较主流的 Java 语言为基础，未来的前景相当广阔。

总之，可以说 Java 的未来一片光明。

1.2 Java 程序设计环境

学习一门语言，除了相应的语法外，还需要熟练掌握 Java 程序的开发工具和运行环境的配置。虽然 Java 程序是纯文本文件，用任何一个文本编辑软件都能够开发与编写，但是一个好的编辑器可以帮助程序员减少很多麻烦。本节将具体介绍如何搭建 Java 程序的设计环境。

1.2.1 命令行工具——JDK 6.0

JDK 是 Java 标准版开发工具包，是 Java 开发和运行的基本平台。Java 语言程序代码的运行离不开该 JDK，使用其可以编译 Java 源代码为类文件。目前最稳定的版本为 JDK 6.0，注意下载时不要选择 JRE（Java Runtime Environment，Java 运行时环境），因为该种版本不包含 Java 编译器和 JDK 类源码。具体下载步骤如下。

(1) 访问下载 JDK 的官方网站（<http://java.sun.com/javase/downloads/index.jsp>），如图 1.1 所示。



图 1.1 JDK 下载首页

(2) 打开下载页后，单击相应版本后的 Download 按钮，如图 1.2 所示。这时就会弹出浏览器安全链接警告，在该对话框中单击“确定”按钮就会进入下载页面，如图 1.3 所示。

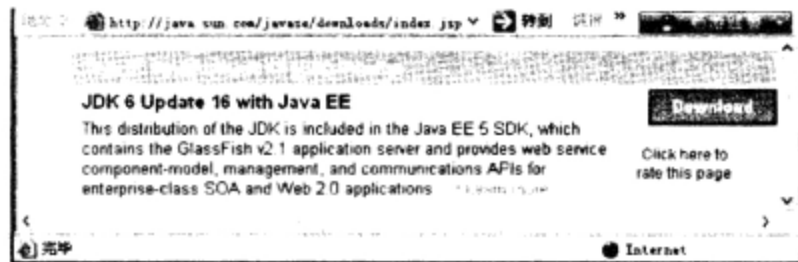


图 1.2 选择 JDK 版本



图 1.3 JDK 下载页面

(3) 在下载页面里选择要安装的平台, JDK 能支持多个主流操作系统, 包括 Windows、Linux、Solaris 操作系统。对于各种操作系统又根据 CPU 分成两种, 即 X86 系列针对家用电脑的 32 位 CPU; X64 系列针对 64 位 CPU。一般只关注 Windows X86 版本即可。所以, 在 Platform 下拉列表框中选择 Windows 选项。在 Language 下拉列表框中选择 Multi-Language 选项, 表示该软件包支持多国语言。完成上面步骤后, 还必须选择下面的复选框表示接受下载协议。单击 Continue 按钮后, 就会转到 JDK 安装文件的页面, 如图 1.4 所示。单击 jdk-6u16-windows-i586-p.exe 链接就会自动下载该软件。

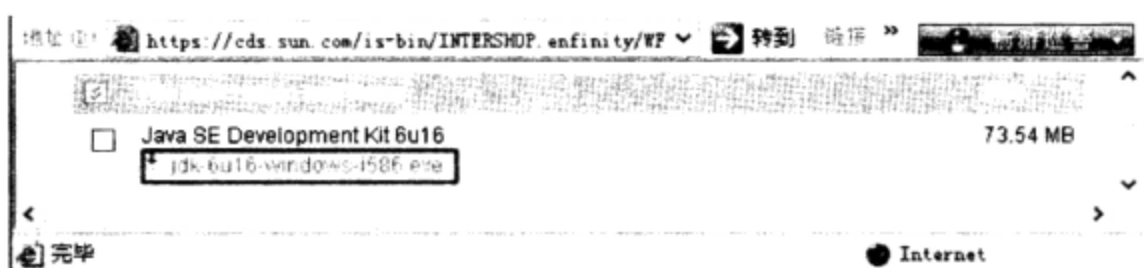


图 1.4 下载 JDK 安装文件

1.2.2 安装工具包 JDK

1.2.1 节介绍了如何下载 JDK 安装程序, 下载完 JDK 安装程序后就可以开始安装 JDK 了。具体安装步骤如下。

(1) 双击 JDK 安装程序(jdk-6u10-windows-i586-p.exe), 接着就会通过 Windows Installer 开始安装过程, 如图 1.5 所示。

(2) 先仔细阅读许可证协议, 然后单击“接受”按钮。在弹出的自定义安装对话框中, 如图 1.6 所示, 就可以进行安装内容和安装路径的选择。

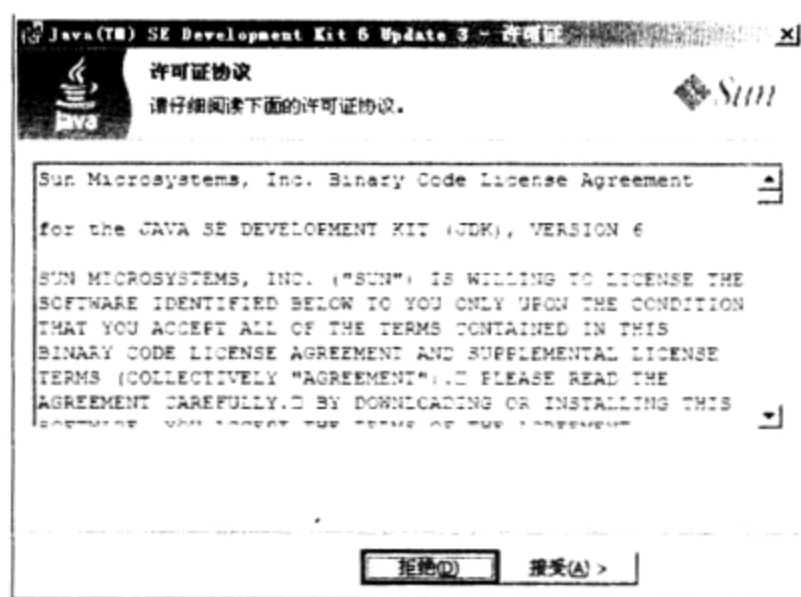


图 1.5 许可协议对话框

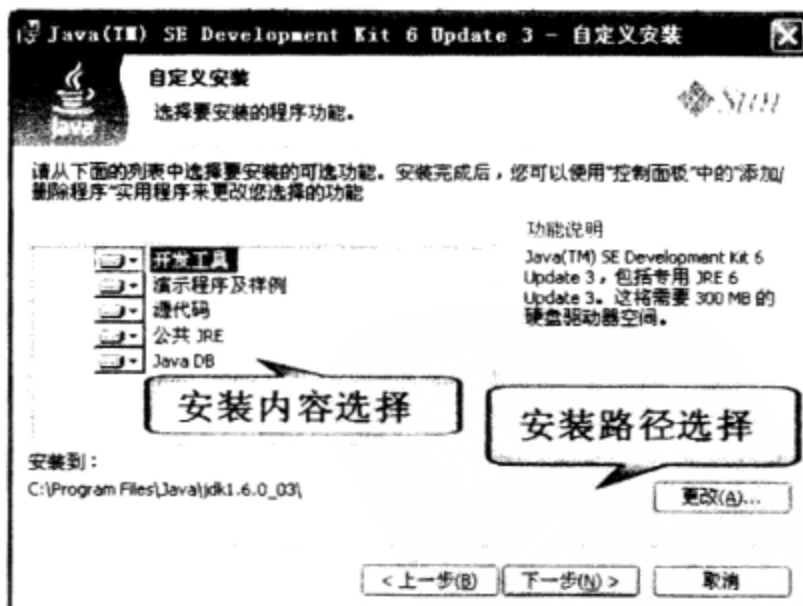


图 1.6 自定义安装对话框

默认的安装内容如下所示。

- ☐ 开发工具: 所谓的 JDK, 是必须要安装的部分。
- ☐ 演示程序及样例: 包含了代码的小程序、应用程序的演示和样例, 建议初学者尽

量安装。

- ☐ 源代码：构成 Java 公共 API 类的源代码。
- ☐ 公共 JRE：独立的 JRE。
- ☐ Java DB：支持开源的 Java 技术数据库。

如果想不安装最后 3 项内容，可以单击选项前的下三角按钮，在出现的下拉列表框中选择“现在不安装此功能”选项，如图 1.7 所示。

一般推荐路径是“C:\jdk1.6.0_10\”，所以需要更改默认安装路径。单击“更改”按钮，如图 1.8 所示，在弹出的“更改当前目标文件夹”对话框中选择相对应的路径，如图 1.9 所示。

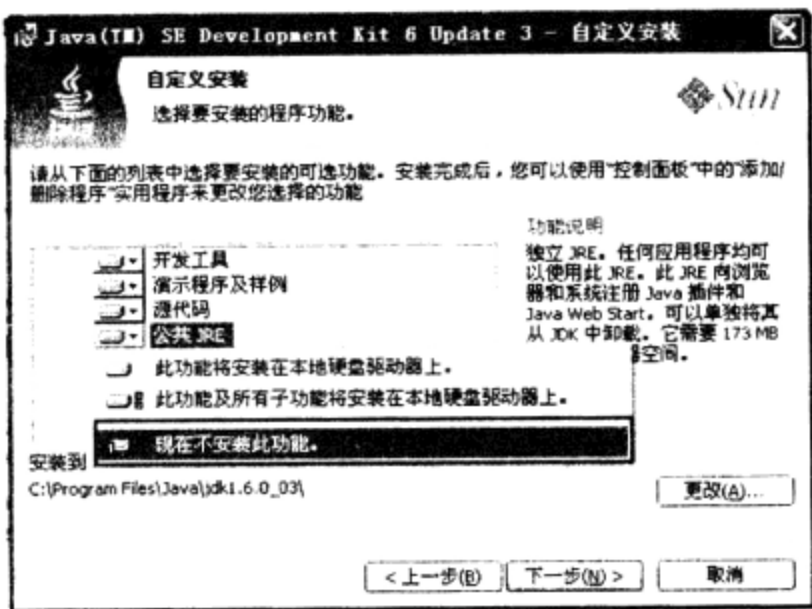


图 1.7 选择安装程序内容

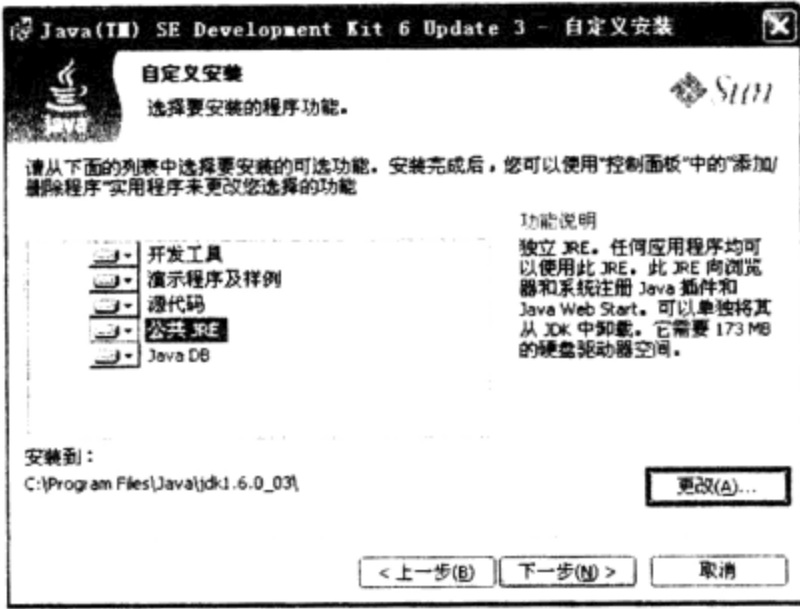


图 1.8 更改安装路径

注意：输入的路径中不推荐有空格和中文，之所以这样做是因为路径有这些内容会出现不必要的问题，导致某些 Java 程序运行失败。

(3) 确认无误后，在“自定义安装”对话框中单击“下一步”按钮，开始执行安装程序。如果安装成功，会出现如图 1.10 所示的对话框，然后单击“完成”按钮结束该 JDK 的安装。

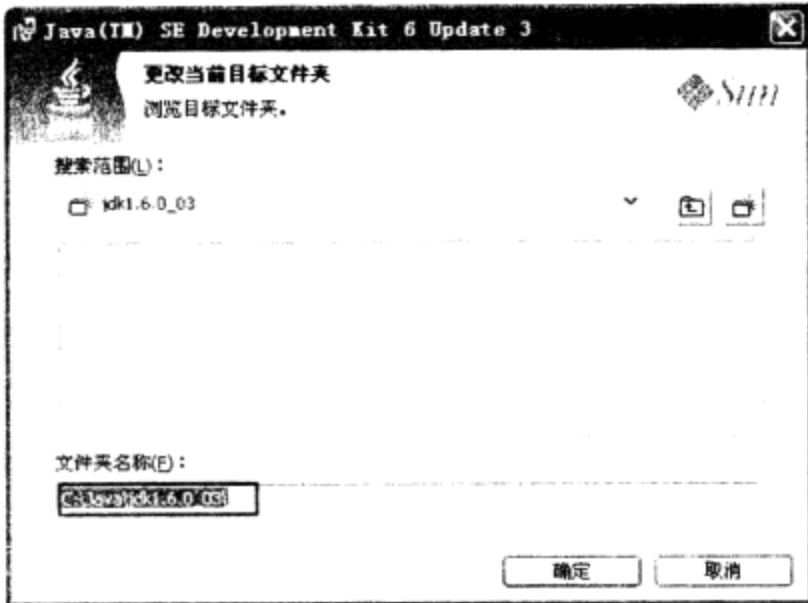


图 1.9 选择安装路径

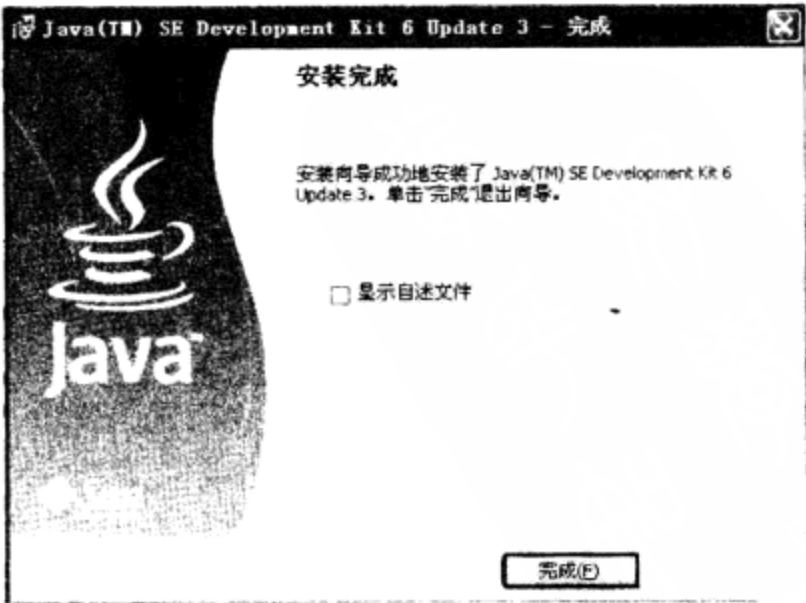


图 1.10 完成 JDK 安装

1.2.3 设置 JDK 环境


1.2.2 节介绍了如何安装 JDK 安装程序，安装完 JDK 安装程序后还必须经过一些必要的设置，才能实现对 Java 程序的编译和运行。在具体配置 JDK 的环境变量之前，首先需要了解 3 个环境变量，它们分别如下。

1. JAVA_HOME 环境变量

JAVA_HOME 环境变量的设置主要是为以后修改方便，即当以后重新安装 J2DK 到其他目录，或是安装其他版本时，只需要修改该变量的值就可以，其他变量的值不需要再变动。

2. PATH 环境变量

PATH 环境变量的设置主要是让系统找得到 J2SE 所提供的工具程序，而不用在每次使用这些工具时，都需要指定它们的完整路径的名称。

 注意：“.”和“..”在 DOS 系统下分别表示当前目录和上一级目录。

3. CLASSPATH 环境变量

CLASSPATH 环境变量的设置主要是让系统找得到所要运行的类，有了这个设置后，在运行 Java 程序时，JVM、J2SE 中的工具程序及 Java 的应用程序都会依照该环境变量的值，找到所需要的相关类。

了解了相关的环境变量后，下面接着来讲解如何配置这些环境变量。在具体配置之前，需要先打开“环境变量”对话框。右击“我的电脑”，在弹出的快捷菜单中选择“属性”选项，就会弹出如图 1.11 所示的“系统属性”对话框。



图 1.11 “系统属性”对话框

(1) 选择“高级”标签下的“环境变量”按钮，如图 1.12 所示，弹出如图 1.13 所示的“环境变量”对话框，该对话框就是用来设置环境变量的。

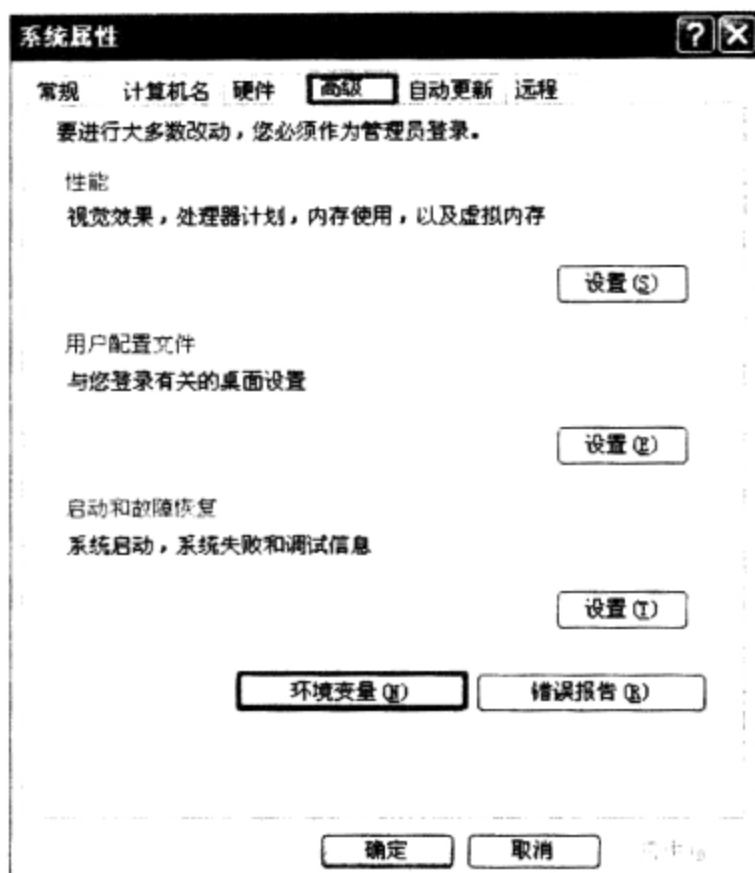


图 1.12 单击“环境变量”按钮

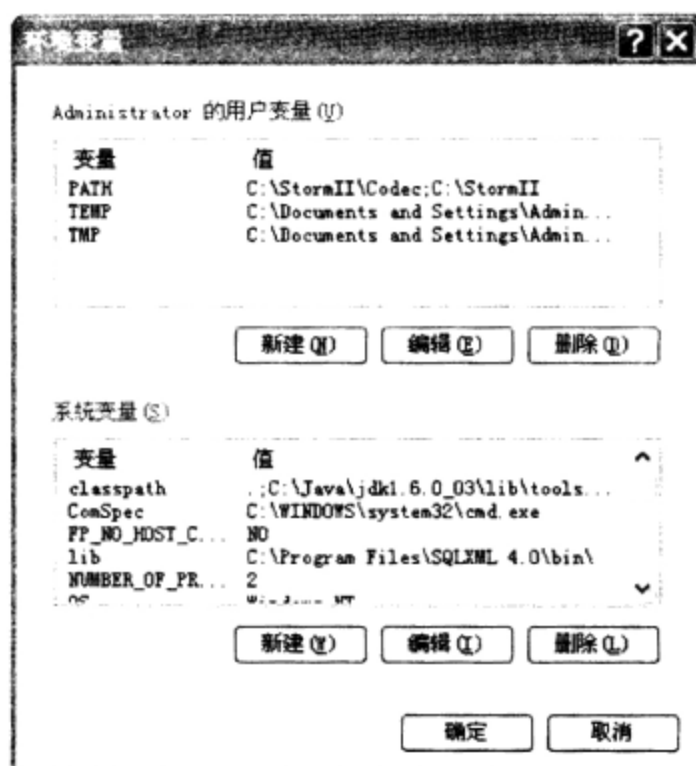


图 1.13 “环境变量”对话框

(2) 然后单击“新建”按钮，弹出“新建系统变量”对话框。对该对话框进行如图 1.14 所示的配置，就可以实现对 JAVA_HOME 环境变量的设置。对该对话框进行如图 1.15 所示的配置，就可以实现对 PATH 环境变量的设置。

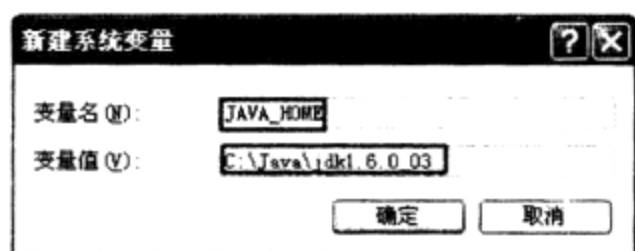


图 1.14 JAVA_HOME 设置

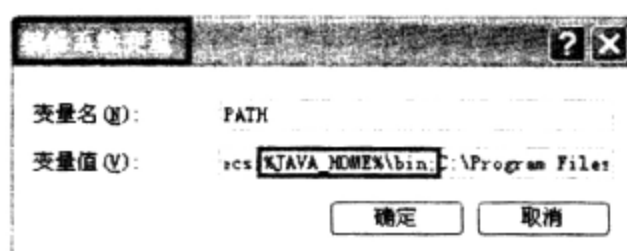



图 1.15 PATH 设置

对于 PATH 环境变量的设置，可以在原有值的基础上添加新的路径，因为想在任意路径下运行 Java 方面的命令程序，所以应该在 PATH 原来值的末尾加上分号(;)，然后再加上 Java 编译器所在的路径(%JAVA_HOME%\bin)。对于“%JAVA_HOME%”所起的作用，就是将环境变量 JAVA_HOME 的当前值取出，即上述的设置值，相当于“C:\Java\jdk1.6.0_03\bin”。

 **注意：**一个环境变量可以存放多个路径，路径和路径之间可以用分号(;)隔开。

(3) 在“新建系统变量”对话框中，参照图 1.16 进行配置，就可以实现对 CLASSPATH 环境变量的设置。

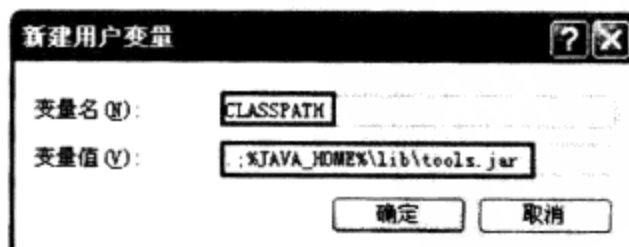



图 1.16 CLASSPATH 设置

如果想彻底了解 CLASSPATH 环境变量，必须要了解 Java 的 3 种 Class 分类，即 Bootstrap classes、Extension classes 和 User classes。Bootstrap classes 类指是 Java 2 Platform 内置的类库，一般存放在 jre/lib 目录下的 rt.jar 和 il8n.jar 文件中；Extension classes 类是指 Java 2 Platform 内置的类库，一般存放在 jre/lib/ext 目录下；User classes 类是指用户自行设计的类。对于前两种类，不需要设置 CLASSPATH 环境变量值，当应用程序使用时，会自动去找到它们。而对于最后一种类，就需要设置 CLASSPATH 环境变量值，因为系统并不知道程序员设计了哪些类、存放在什么目录下。对于“.”的设置值，指的就是当前目录。

 **注意：**通过上述方式进行设置的环境变量，虽然对以后在当前操作系统上运行的任何程序都有效，但是对先前已经运行起来的程序却不会影响。

1.2.4 集成开发环境安装——MyEclipse 8.5

MyEclipse 是由 Genuitec 公司开发的一款商业软件，从本质上讲它是基于 Eclipse 的 Java EE 方面的插件。该软件除了支持代码编写、编译和测试等，还增加了 UML 双向建模工具、JSP/Srutsdesigner、可视化的 Hibernate/ORM 工具、Spring 和 Web services 等各个方面的功能。目前最新的版本为 MyEclipse 8.5，不同类型的 MyEclipse 安装过程不一样。ALL In One 类型的 MyEclipse 安装过程很简单，只要双击安装程序就可以了，不需要进行复杂的设置。具体安装步骤如下。

(1) 双击 MyEclipse 8.5 安装程序(myecclipse-8.5M1-win32.exe)，接着就会通过 Windows Installer 开始安装过程，然后弹出 Welcome to the MyEclipse 8.5 MI Installer（欢迎对话框），如图 1.17 所示。

(2) 单击 Next 按钮后，进入如图 1.18 所示的 Accept License（接受协议）对话框。在该对话框中先仔细阅读许可证协议，然后选择接受后再单击 Next 按钮，进入 Customize MyEclipse install location（配置安装路径）对话框，如图 1.19 所示。



图 1.17 MyEclipse 欢迎对话框

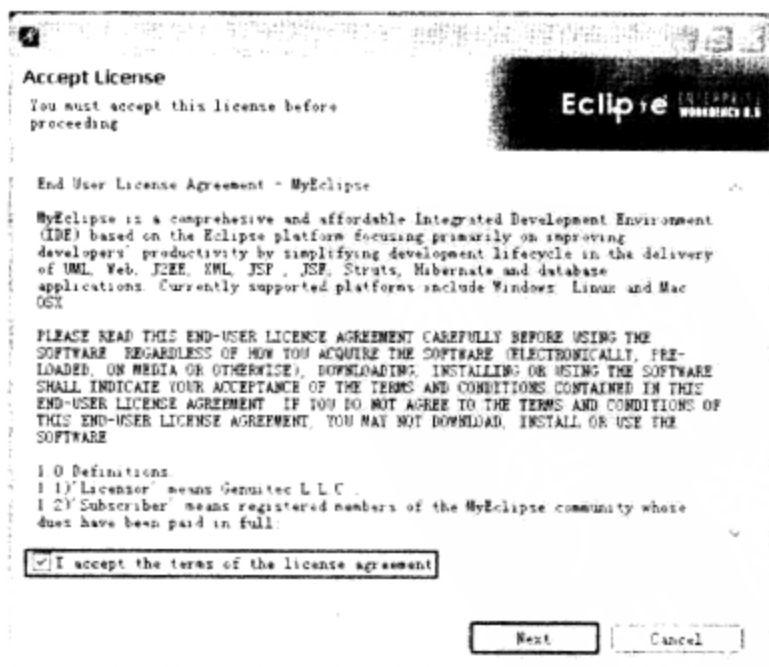


图 1.18 接受协议对话框

(3) 在该对话框中，如果想修改安装路径，单击 Chang 按钮会进入如图 1.20 所示的 Configure runtime and plugin installation details(配置运行时软件和插件的安装路径)对话框。



图 1.19 配置安装路径对话框



图 1.20 选择路径对话框

如果想配置运行时软件或插件的安装路径，单击相应的 Browse 按钮，在弹出的对话框中，如图 1.21 所示，选择相应路径即可。最后配置运行时软件和插件的安装路径对话框，如图 1.22 所示。单击 Next 按钮就会返回到如图 1.23 所示的配置安装路径对话框。

(4) 在配置好路径的对话框中，单击 Install 按钮，弹出如图 1.24 所示的安装对话框。当安装过程结束后，在弹出的对话框中，单击 Finish 按钮即完成 MyEclipse 8.5 软件的安装。

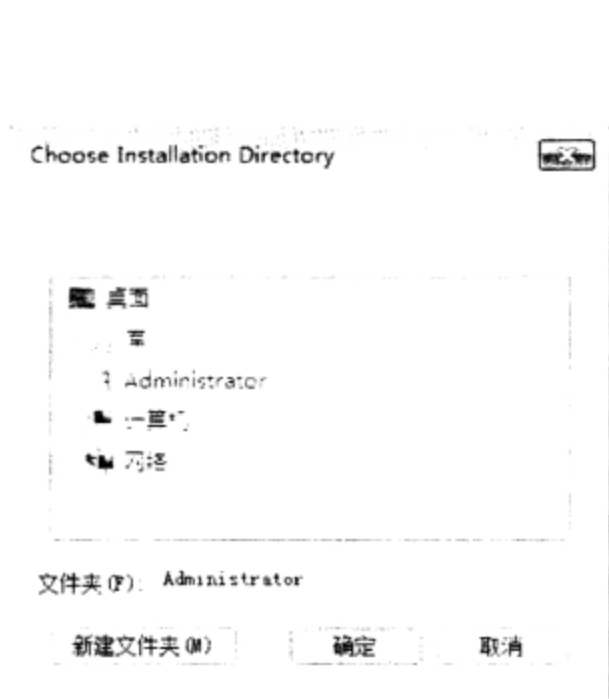


图 1.21 选择文件对话框



图 1.22 选择的相应路径

1.2.5 MyEclipse 的一些常用操作

安装完 MyEclipse 集成开发工具后，不能马上进行 Java 开发，还必须要进行一些必要的配置。具体步骤如下。

1. 工作空间设置操作

当 MyEclipse 第一次启动时，需要设置工作空间，在本书中将工作空间设置为 c:\Workspaces 路径，如图 1.25 所示。

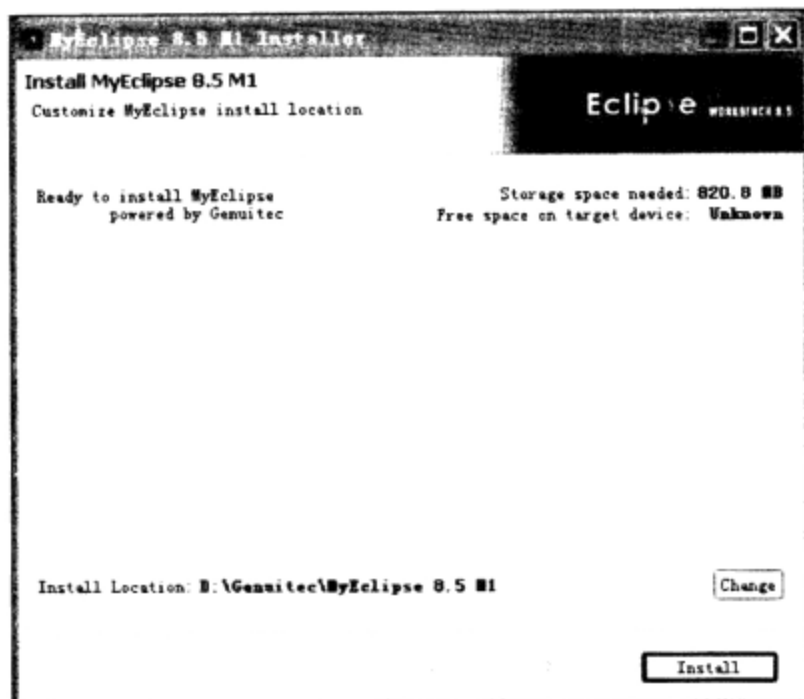


图 1.23 配置好安装路径

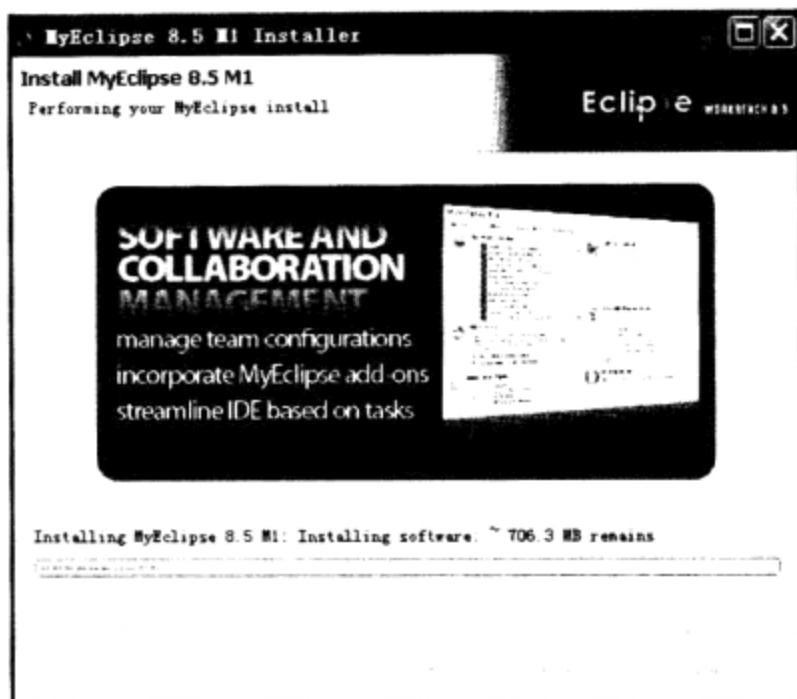


图 1.24 安装过程

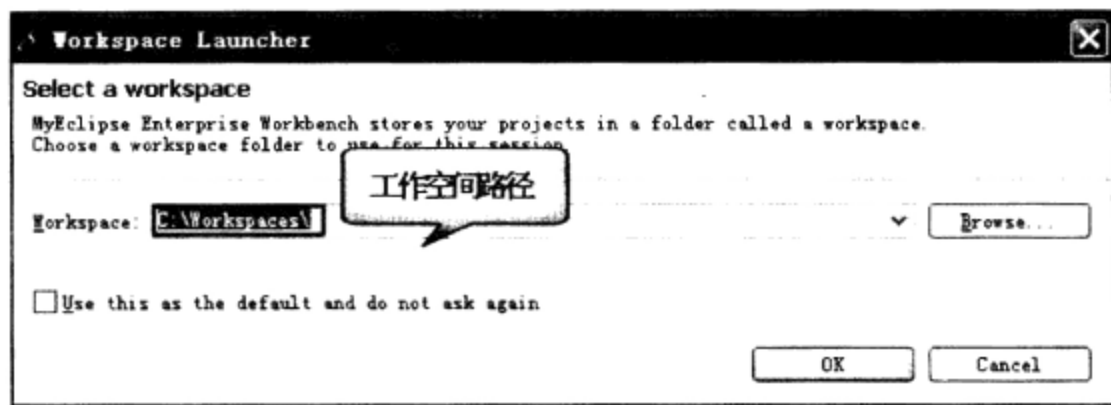


图 1.25 Workspace Launcher 对话框

注意： 如果不想每次启动 MyEclipse 都出现 Workspace Launcher 对话框，可以通过取消 Use this as the default and do not ask again 选项将该对话框屏蔽。

当进入 MyEclipse 后，想进入另一个工作空间，可以通过选择 File | Switch Workspace | Other 命令，打开 Workspace Launcher 对话框，这时选择相应的工作空间即可。MyEclipse 集成环境的设置是应用于其工作空间，而不是其本身，即在一个工作空间的设置改变时并不影响另一个空间。

2. 编译器的操作

在使用 MyEclipse 开发 Java 程序时，应先确定该集成环境所使用的编译器版本。如果想查看编译器的版本，可以在通过菜单 Window | Preference 命令打开的 Preferences 对话框（如图 1.26 所示）中实现，具体步骤如下。

（1）查看所安装的 JRE，即选择 Java>Installed JREs 结点，弹出如图 1.27 所示的 Installed JREs 对话框，在该对话框中显示的是 MyEclipse 集成环境自带的 JRE。

如果想安装自己的 JRE，可以通过单击 Installed JREs 对话框上的 Add 按钮，打开 Add JRE 对话框。单击 Browse 按钮，在弹出的“浏览文件”对话框中选择自己所安装 JRE 的根目录，单击“确定”按钮即可配置好 Add JRE 对话框，如图 1.28 所示。最后单击 Finish 按钮返回 Installed JREs 对话框，在其中选择新出现的 JRE 即可实现修改，如图 1.29 所示。

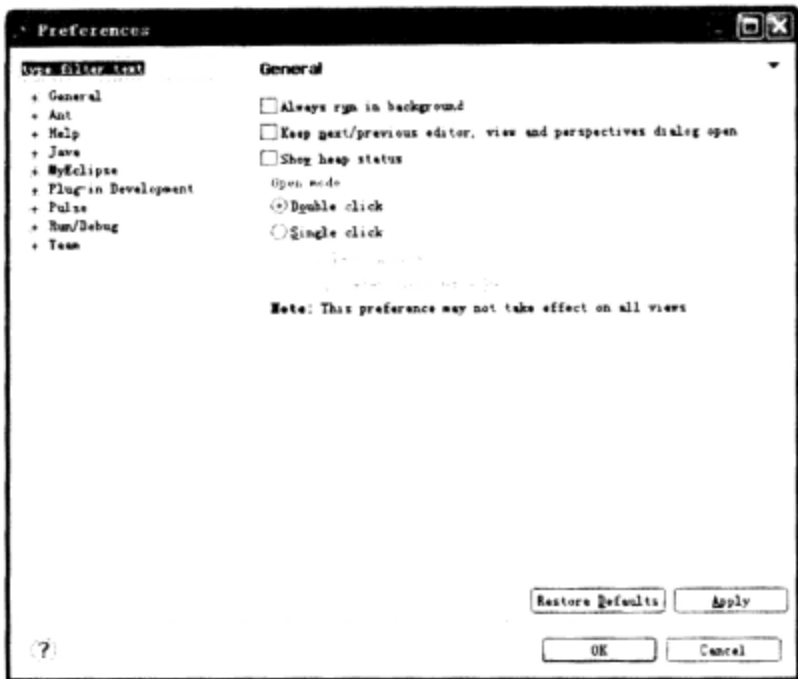


图 1.26 Preferences 对话框



图 1.27 Installed JREs 对话框

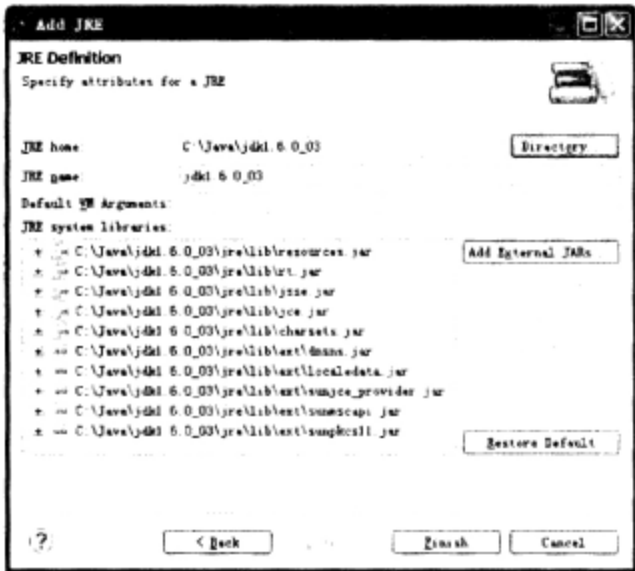


图 1.28 Add JRE 对话框

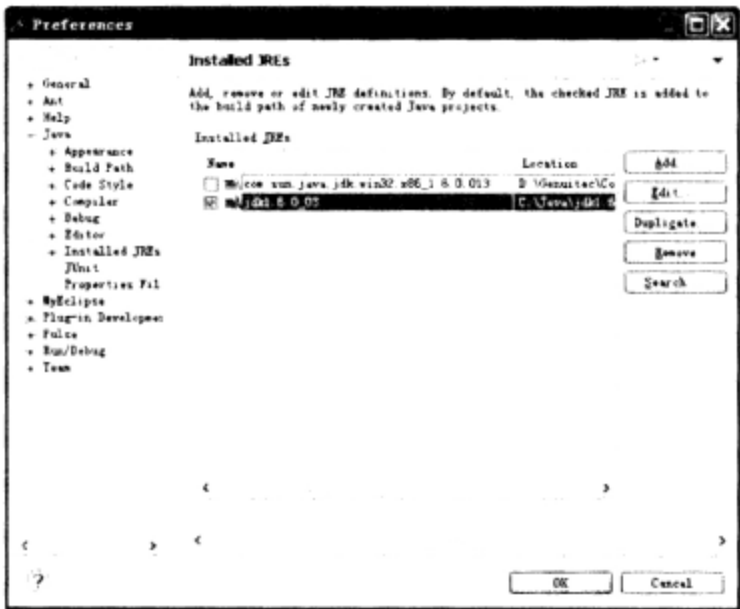


图 1.29 Installed JREs 对话框

(2) 查看编译器的版本，即选择 Java>Compiler 结点，弹出如图 1.30 所示的 Compiler 对话框。在 Compiler compliance level 下拉列表框中显示的是默认的编译版本。如果想修改编译器为 5.0 的版本，可以在 Compiler compliance level 下拉列表框中选择“1.5”，然后单击 OK 按钮即可。

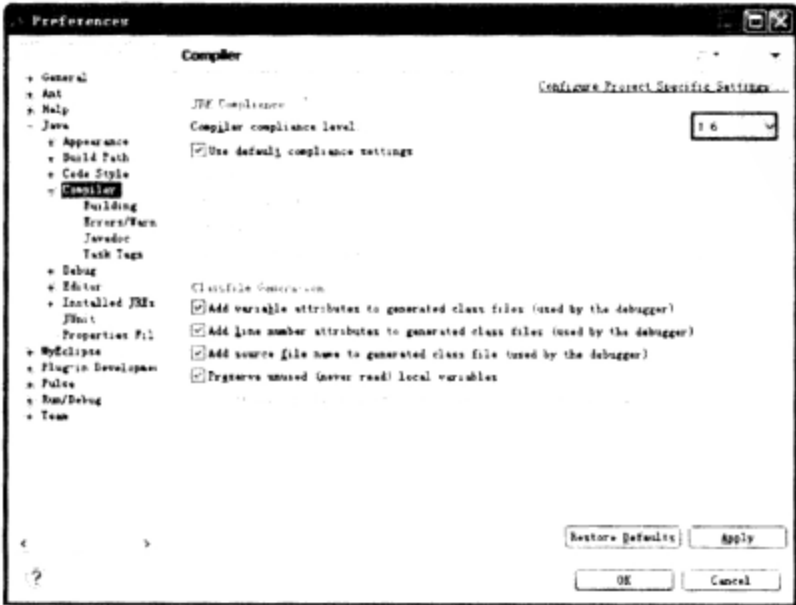


图 1.30 Compiler 对话框

1.3 创建和运行 Java 程序

本节介绍如何编写一个“Hello World!”简单小程序。为了能够让读者清楚 Java 的编译和运行过程，下面分别用手工和集成环境的方式来编写、编译及运行 Java 程序。

1.3.1 手工创建、编译和运行 Java 程序


Java 程序代码其实是一般的纯文本文件，所以用一般的文本编辑器就可以编写 Java 程序。本节将通过 Windows 自带的记事本程序来手工编写、编译和运行 Java 程序。

1. 编写Java程序

打开记事本后，输入如下代码，如代码 1.1 所示。

代码 1.1 第一 Java 项目：First.java

```
public class First {
    public static void main(String[] args) {
        System.out.println("这是第一个 Java 项目");
    }
}
```

 **注意：**在关闭记事本时，文本的后缀名必须为 java，而保存类型必须为“所有文件”，具体设置信息如图 1.31 所示。

2. 编译Java程序

在命令窗口中，首先用 cd 命令进入 First 文件夹，然后运行 javac First.java 命令编译 First.java 文件，最后通过 Dir 命令查看 First 文件夹里是否多了一个名为 First.class 的文件。具体命令过程如图 1.32 所示。

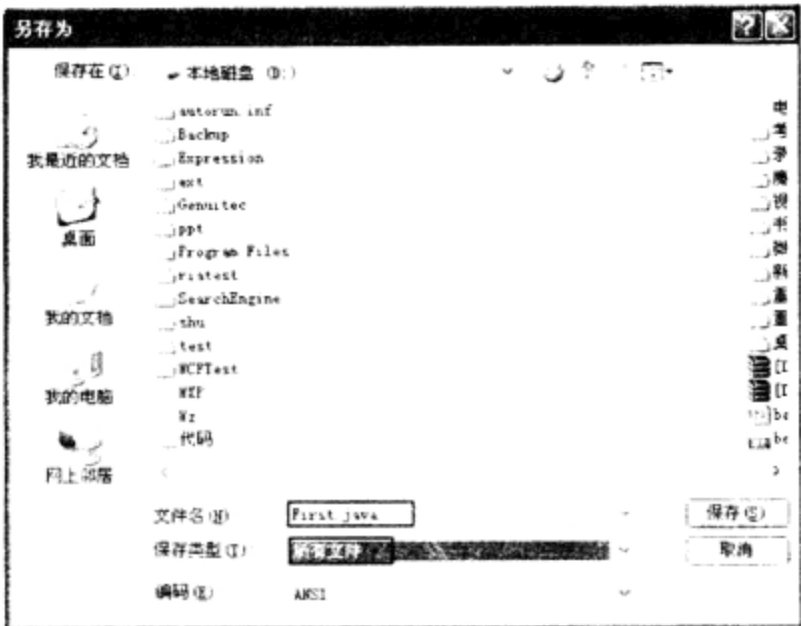


图 1.31 “另存为”对话框



图 1.32 编译过程

3. 运行Java程序

在命令窗口中，用 cd 命令进入 First 文件夹，运行 java First.class 命令。执行完毕后，在该命令窗口的屏幕中出现了“这是第一个 Java 项目”的文字，如图 1.33 所示。这样，编译运行第一个 Java 程序的过程就算完成了。



图 1.33 运行 Java 程序

注意：运行时的命令为 java First，而不是 java First.class，即运行时不要带上“.class”扩展名。

1.3.2 在 MyEclipse 8.5 中创建、运行、调试和管理 Java 项目

MyEclipse 编写 Java 程序的流程是：新建 Java 项目、创建 Java 类、编写 Java 代码和运行 Java 程序。下面将分别介绍。

1. 新建Java项目

在 MyEclipse 集成开发环境中，通过选择菜单 File | New | Java Project 命令打开 New Java Project 对话框。在对话框中进行如图 1.34 所示的设置，单击 Finish 按钮完成 Java 项目的创建。

2. 创建Java类

在 MyEclipse 集成开发环境的包资源管理器中，右击所创建 Java 类的项目，在弹出的快捷菜单中选择 New | Class 命令，就可以打开 New Java Class 对话框。对该对话框进行如图 1.35 所示的设置，单击 Finish 按钮完成 Java 类的创建。

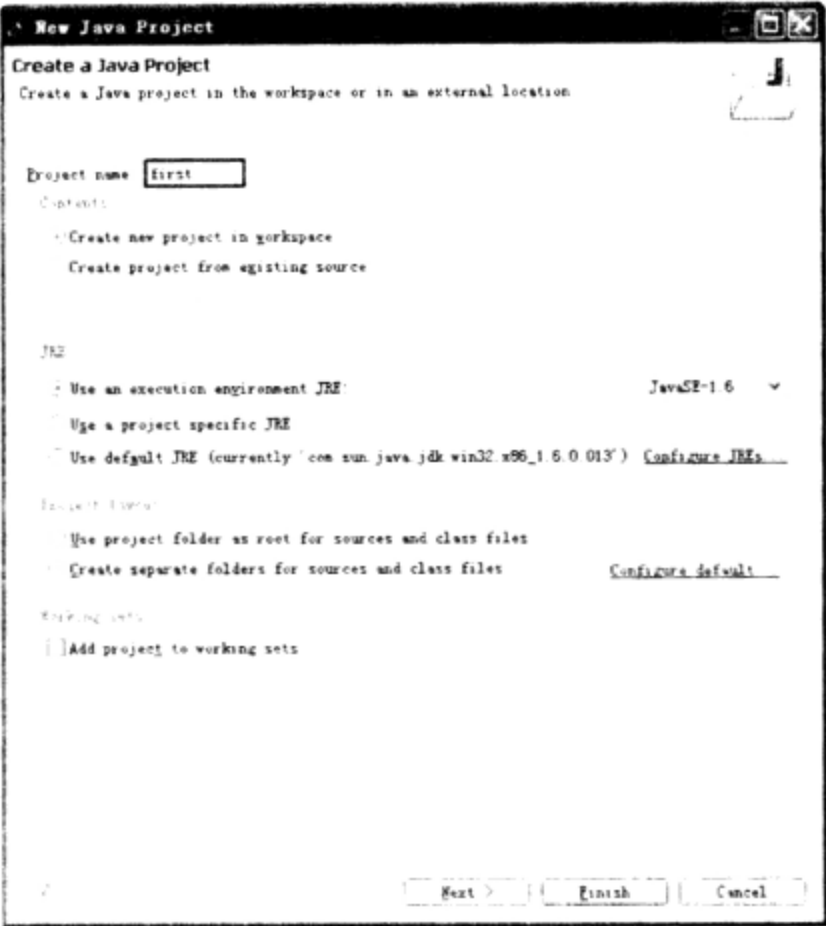


图 1.34 New Java Project 对话框



图 1.35 New Java Class 对话框

3. 编写Java代码

在 MyEclipse 集成开发环境的包资源管理器中，双击创建的 Java 类，会打开该类的源代码编辑器。在该编辑器中编写如下内容，完成 Java 代码的编写，如代码 1.2 所示。

代码 1.2 第一个 Java 项目：First.java

```
public class First {  
    public static void main(String[] args) {  
        System.out.println("集成环境中的第一个 Java 项目");  
    }  
}
```

【代码解析】

上述代码中，只有“System.out.println("集成环境中的第一个 Java 项目")”这句代码是笔者自己编写的，其他的代码都由集成环境 MyEclipse 自动生成。

4. 运行Java代码

在 MyEclipse 集成开发环境的包资源管理器中，右击要运行的 Java 类，在弹出的快捷菜单中选择 Run As | Java Application 菜单项，就可以运行该 Java 类。运行结束后，在控制台视图中将显示程序的运行结果，如图 1.36 所示。

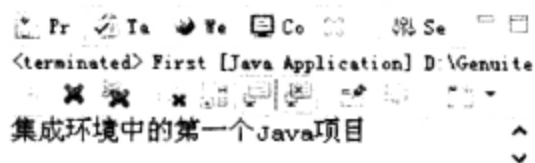


图 1.36 运行结果

1.4 小 结

本章首先讲解 Java 语言的一些基础知识，然后详细介绍开发 Java 语言所需要的基础性工具，即 JDK 6.0 工具包和 MyEclipse 8.5 开发工具。通过对这些基础软件的下载和安装方面的详细讲解，使读者能够很容易地实现 Java 项目的开发。最后通过一个简单的实例详细介绍了如何创建、编译和运行 Java 程序。

第 2 章 Java 面向对象编程

对于大部分程序员来说，由于面向对象是个很抽象的东西，所以理解起来是一个比较难的名词。面向对象其实是一种思想，起源于 20 世纪 60 年代中期的仿真程序设计语言 Simula 1。掌握面向对象，不是知道面向对象的概念，而是应该掌握面向对象的精髓。本章将通过通俗易懂的方式来讲解面向对象。

本章的学习目标如下：

- ☐ 面向对象的基本概念；
- ☐ 面向对象的相关特性；
- ☐ 如何在程序中实现面向对象。

2.1 面向对象的一些概念

在我们的日常生活中，随处可见的都是对象，小到一粒沙，大到中国的万里长城、埃及的金字塔，所有的东西都是对象。于是出现了一种模拟我们日常生活中的对象的软件开发模式，即面向对象。在该开发模式中，把软件系统抽象成各种对象的集合，以对象为最小系统单位，这就更接近于人类的自然思维，给程序开发人员更灵活的思维空间。

2.1.1 面向对象涉及的概念

随着计算机的应用越来越广泛，社会对软件开发提出了更高的要求。然而软件技术的进步却远远落后于硬件技术的进步，人们常常无法控制软件开发的周期和成本，软件质量总是无法让人满意，即所谓的软件危机。

为了摆脱软件危机，必须按照工程化的原则和方法来组织软件开发工作。在涉及大量计算的问题上，面向过程的设计方法暴露了越来越多的不足。例如：

- ☐ 功能与数据分离，不符合对现实世界的认识。
- ☐ 基于模块的设计方式，导致软件修改困难。
- ☐ 自顶向下的设计方法，限制了软件的可重用性，降低了开发效率，也导致最后开发出来的系统难以维护。

为了解决面向过程设计的这些问题，面向对象的技术应用而生。它是一种非常强有力的软件开发方法，符合人们的思维习惯，同时有助于控制软件的复杂性，提高软件的开发效率，从而得到了广泛的应用，已成为目前最流行的一种软件开发方法。

1. 面向对象基本概念

可以这样说：“面向对象=对象+类+继承+通信”。如果一个软件系统是由这 4 个概念

来设计和实现的，就认为是面向对象的。

- 对象：对象是面向对象开发方法的基本成分，可以是现实社会中的一个物理对象，还可以是某一概念实体的实例。
- 类：类是一组具有相同数据结构和相同操作对象的集合，是对一系列具有相同性质的对象的抽象，是对对象共同特征的描述。例如对于不同名字的人，因具有一系列相同性，所以可以抽象出一个数据类型，即“人类”的类。
- 继承：继承是使用已经存在的定义作为基础建立新定义的技术。新类的定义可以是已经存在的类的所有成员和新类所增加的成员组合。已经存在的类可以作为基类来引用，而新类可以作为派生类来引用。这种技术大大降低了软件的开发费用。

2. 对象模型

对象模型技术是美国通用电气公司提出的一套系统开发技术。它以面向对象的思想为基础通过对问题进行抽象，造出一组相关的模型，从而能够全面地捕捉问题空间的信息。对象模型技术把分析时收集到的信息构造在3类模型中，即对象模型、功能模型和动态模型。3个模型从不同的角度对系统进行描述，分别着重于系统的一个方面，组合起来构成对系统的完整描述。

- 对象模型：定义“对谁做”，描述系统的静态结构，包括类和对象，它们的属性和操作，以及它们之间的关系。
- 动态模型：定义“何时做”，着重于系统的控制逻辑，考察在任何时候对象及关系的改变，描述这些涉及时序和改变的状态。
- 功能模型：定义“做什么”，着重于系统内部数据的传送和处理，即通过计算，只要知道输入数据就能得到输出数据，而不需要考虑参加计算的数据按什么时序执行。

3. 面向对象的分析

面向对象的分析（object-oriented analysis, OOA）是软件开发过程中的问题定义阶段，这一阶段最后得到对问题清晰、精确的定义。传统的系统分析产生一组面向过程的文档，定义目标系统的功能。面向对象分析则产生一种描述系统功能和问题空间的基本特征的综合文档。面向对象的分析过程可分为两个阶段，即论域分析阶段和应用分析阶段。

- 论域分析：是软件开发的基本组成部分，目的是使开发人员了解问题空间的组成，建立大致的系统实现环境。
- 应用分析：是依据在论域分析时建立起来的问题论域模型，并把问题论域模型用于当前特定的应用之中。

📌 注意：模型识别的要求可以针对一个应用，也可以针对多个应用。通常我们着重考虑两个方面，即应用视图和类视图。在类视图中，必须详细表示每个类的规格说明和操作，并表示出类之间的相互作用。

4. 面向对象的设计

从面向对象的分析到面向对象的设计是一个逐步扩充模型的过程。面向对象的分析是

以实际问题为中心，可以不包括任何与特定计算机有关的问题，主要考虑“做什么”的问题；面向对象的设计则是面向计算机的实地开发活动，考虑“怎么做”的问题。面向对象的设计分为两个阶段，即高层设计和低层设计。

- 高层设计阶段：开发系统的结构，构造待开发软件的总体模型。
- 低层设计阶段：集中于类的详细设计阶段。

注意：在高层设计过程中，应当使子系统高层部件之间的通信量达到最少，把子系统中相互之间存在高度交互的类进行逻辑分组。而在低层设计过程中，类的设计需要采用信息隐蔽、高内聚低耦合等设计原则。

2.1.2 类和对象

在面向对象语言中有两个最基本的概念：类和对象。类是用来创建对象的模板，它包含了被创建对象的属性和方法的定义。

1. 类和对象的定义

在现实世界中，对象随处可见，但是平常不直接以“对象”称呼他们，那么如何称呼这些对象呢？通常会以这个对象分类的名称来称呼，例如路边生长的树、天上飞的鸟、水里游的鱼、路上跑的车等，这里的树、鸟、鱼、车都只是一种对象的分类而已，而这些分类我们习惯称为“类”。所以世界上存在许许多多的类，比如刚刚提到的树类、鸟类等。

那么什么是对象呢？所谓对象就是符合某种类定义所产生出来的实例（instance）。虽然在上一段中用类名称呼对象，但实际上看到的还是类的实例，而不是一个类。例如看见路旁的一棵树，这里的“树”虽然是一个类名，但实际上看见的是树类的一个实例对象，而不是树类。又例如去电器商店买一台计算机，这里的“计算机”只是类的名称，最后买回家来的是一台计算机的实例对象，而不是一个类。

类跟对象的关系如图 2.1 所示，在计算机世界里，类只是个抽象的称呼，而对象则是与现实生活中的事物相对应的实体，即是个看得到、摸得到、听得到的实体。

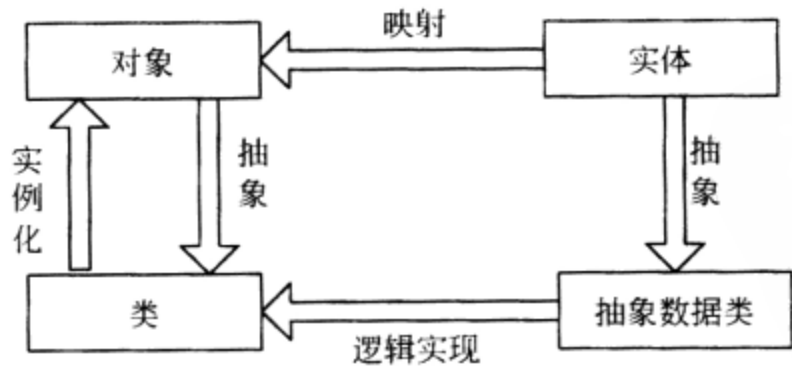


图 2.1 类与对象的关系

2. 类的成员

如果只存在类和对象，还不足以很好地描述现实生活中的任何一个实体。例如小王家养了一条鱼，这里的鱼是类的名称，这时还不能想象到这条鱼是什么样子。但是如果说小

王家养了一条鱼，它的名字是小黑，这里的小黑就是对象的名称，这时就可以想象到这条鱼的样子，即通过名字是小黑，而不是小白、小黄，就可以肯定该条鱼有异于其他鱼对象的地方，比如鱼身体的颜色等。而“鱼的颜色”等，这些用来描述这条鱼的东西，就称之为“属性（attribut）”。所谓属性就是用来形容一个对象用的，正是因为有了这些属性，世界上每个对象都不相同。

对于小黑这条鱼来说，它能在水里游泳等，把这种行为称之为“方法（method）”，所谓方法就是对象的行为或使用对象的方法，正是因为有了方法，才能操作一个对象。

📌注意：对于对象来说，属性属于对象静态的一面，用来形容对象的一些特性；而方法属于对象动态的一面，用来操作对象。

在 Java 语言中，属性和方法称为对象的“成员”，是构成对象的主要部分，如果没有这两个东西，对象就没有存在的意义。

总之，现实生活中实体的抽象化就是类，而类的具体实例化就是对象。在具体定义类时，会在其内部定义许多产生该类对象时所需要具备的一些属性和方法。通过属性和方法可以很容易地辨别出那一个对象属于什么类，同时也会知道该怎么使用这个对象和该对象具有的一些特征。

2.2 面向对象的一些特性

经过长期的实践可以发现，面向对象开发模式更符合人的思维模式，而利用该模式开发的程序更加健壮和强大，在具体的开发过程中更有利于应用程序的划分、组织和管理。面向对象之所以这么强大，主要是因为面向对象具有继承性（inheritance）、多态性（polymorphism）和封装性（encapsulation）。

2.2.1 继承特性

为什么要出现继承呢？

要想清楚上面的问题，可以通过现实生活中的手机实体来理解。例如，如果要定义一个手机类，可是手机的种类很多，比如普通手机、智能手机、时尚手机等，如果只用一个手机类来定义所有类手机的属性和方法，那么这个类实例化出来的对象一定会多出不属于自己的属性和方法，例如智能手机有登录 QQ 的方法，但普通手机并不需要登录 QQ 的方法。

为了解决上述问题，可以先定义一个手机类，然后再去定义其他各种类型的手机类。各种类型的手机类会继承手机类中所有开发出来的可以继承的属性和方法，然后再加上一些自己所有的属性和方法，或是修改原本不适用于这个类的方法。在面向对象程序设计中，“手机类”习惯上叫父类，而“各种类型的手机类”习惯上叫子类。

例如，首先需要定义一个手机类，在该类中可以定义两个方法，即接听电话的方法（receive()）和拨打电话的方法（send()）。然后就可以定义一个智能手机类，它继承了手机类，所以在智能手机类中不需要再重新定义原本在手机类中就有的方法（receive()和 send()），另外还必须再加上一个登录 QQ 的方法（landQQ()）。最后还可以定义一个时尚

手机类，同样它也继承了手机类，所以具有了 `receive()`和 `send()`方法。不过在智能手机类中需要改写 `receive()`方法，因为它接电话的方法与其他手机不同，另外还需要加上拍照方法（`photo()`）。上述 3 个类的关系如图 2.2 所示。

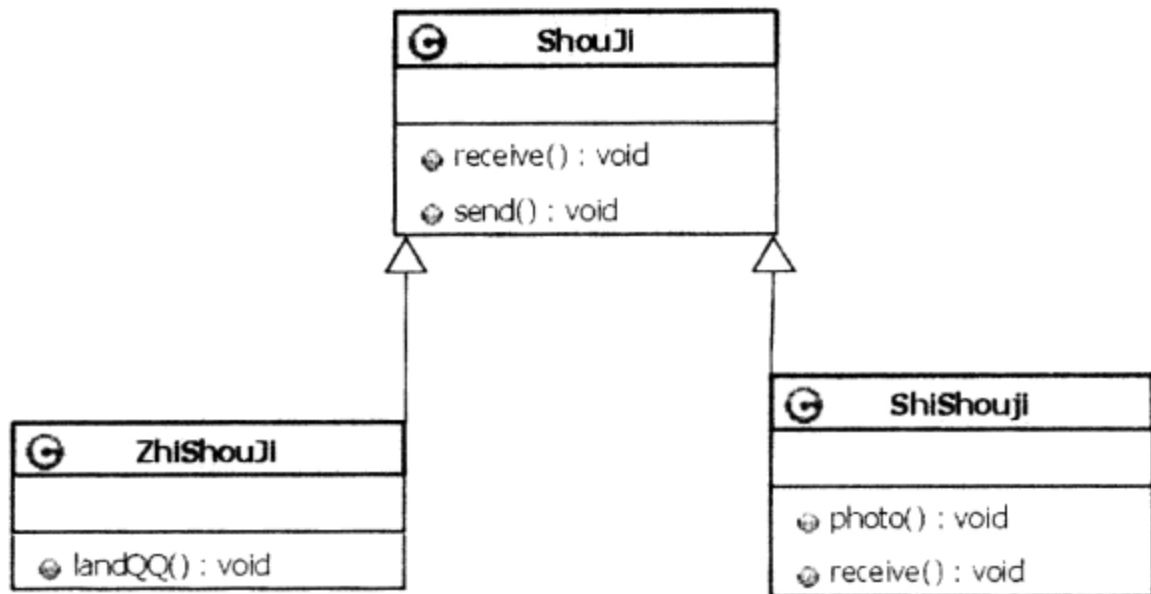


图 2.2 类间的关系

通过手机的类可以发现，继承的最主要目的就是为了“扩展”父类的功能，加强或改进父类所没有定义的属性和方法。

2.2.2 多态特性

什么是多态呢？在面向对象中，多态是指在父类中定义的属性和方法被子类继承之后，可以具有不同的数据类型或表现出不同的行为，即同一个属性或方法在父类和其子类中具有不同的意义。

例如在动物园里有一只名叫小虎的老虎，没见过老虎的小明第一次来到关小虎的笼子旁边，就会问他妈妈“这是什么动物？”。他妈妈就会回答说：“这是老虎呀！”。但是如果问饲养员就会回答“这是小虎呀！”。这里提出一个问题，小明不认识老虎，为什么他知道它是动物？这是因为老虎具备了动物该有的特性，所以就算不知道它真正的类，但是可以用它的父类来看待它的一些属性和行为。

在具体使用多态时，有 3 点必须注意：

- ❑ 小虎就是小虎，不会因为用不同的表示法来形容就会改变它的实例。例如虽然小虎也叫动物，但是实际上它还是一只老虎类实例化的对象，而不是一个动物类实例化的对象。
- ❑ 当把小虎当做动物时，只能访问和使用动物类所具有的属性和方法。也就是说，如果小虎有一个新的方法，例如跳火圈，如果把它当动物看待的时候，根本不知道它是小虎，又怎么会知道它会跳火圈呢？所以当作动物时，不能使用子类（小虎类）才有的方法（跳火圈方法）。
- ❑ 当在小虎类中修改老虎类中的跳跃方法后（老虎类中的跳跃方法里设置为 5 米，小虎类则为 10 米），当以老虎的观点来调用跳跃方法时，实际上它会跳 10 米而不是 5 米。这是因为小虎就是小虎，不管把它当成什么，它都会运行自己修改过的方法。

2.2.3 封装特性

- 封装也称为信息的隐藏，使用其的目的主要有两个：
- ❑ 第一是保护类中的数据，不让这些数据被错误地使用或破坏。
 - ❑ 第二是隐藏一些不需让其他人知道的细节。

例如，动物类有一个描述脚个数的属性（Legs），如果该属性能被继承该类的子类直接使用或修改，就会存在严重的问题，有些程序员就会把该属性设置成-1 等，可是世界上根本没有一种动物的脚的数目为负数。为了解决该问题，可以把该属性保护起来，防止其他人乱用。保护的方式就是把属性隐藏起来，只能通过特定的方法才能使用、修改这些属性。又例如对于一个手机类，当使用手机打电话时，只需要知道调用哪个方法就可以，并不需知道该方法中的具体步骤。

2.3 Java 中实现的面向对象特性

通过前面几节的学习可以完全掌握面向对象的核心，本节将接着前面的内容，讲解在 Java 语言中如何实现面向对象中的基本概念：类、属性、方法、对象、继承、多态和封装。

2.3.1 定义类

动物园中有许多动物，这里的“动物”就是一个类，如何设计一个动物类呢？UML 的表示如图 2.3 所示。根据动物类的 UML 图可以发现，Animal 类具有一个 legs 属性和两个方法，即表示移动 move() 方法和 eat()方法。

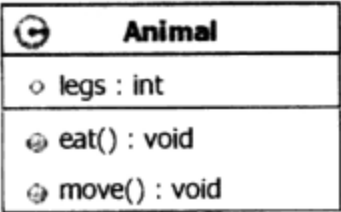


图 2.3 动物类

在 Java 语言中，定义动物类的具体代码如代码 2.1 所示。

代码 2.1 动物类：Animal.java

```
01 public class Animal { //类
02     public int legs; //属性成员
03     public void eat(){ //方法成员
04         System.out.println("吃");
05     }
06     public void move(){ //方法成员
07         System.out.println("移动");
08     }
09 }
```

【代码解析】

- ❑ 定义一个类基本上只需要两行代码（01 和 09 行）就算一个完整的类定义。其中 public 关键字是类的修饰符，用来指定类的访问权限；class 关键字主要用来表示声明的是一个类；Animal 则是类名，用来指定类的名字。
- ❑ 定义一个属性其实就是定义一个变量，所以可以完全按照定义变量的方式定义属

性，legs 就是定义的属性。

□ 最后 eat()和 move()方法就是定义的方法，对于方法则必须有方法返回值的类型。

2.3.2 创建对象

在 Java 语言中定义任何变量都需要指定变量类型，因此在创建对象之前，必须要声明对象。声明对象的基本格式如下：

类名 对象名；

由于对象是类的实例，所以“类名”必须为已经定义的类，而对象名则必须符合 Java 语言的语法。

声明完对象以后，可以实例化对象，基本格式如下：

对象名= new 类名();

在上述定义中，类名用于指定构造函数名。在 Java 语言中，创建动物对象的具体内容如代码 2.2 所示。

代码 2.2 动物对象：AnimalObject.java

```
public class AnimalObject {
    public static void main(String[] args) {
        Animal animal1;
        animal1 = new Animal();
        Animal animal2 = new Animal();
    }
}
```

//声明对象
//实例化对象
//创建对象

2.3.3 实现继承

如果每种动物对象都用类 Animal 来创建，那么该类将会拥有所有动物的属性和方法，显然不符合现实。为了解决该问题，就需要面向对象中的继承特性。在 Java 语言中如果想继承一个类，只需要关键字 extends 就可以实现，基本格式如下：

子类名 extends 父类名{
}

当设计鱼的类时，因为它也是动物，所以可以通过继承动物类（Animal）来实现。该类的具体内容如代码 2.3 所示，与 Animal 类的关系如图 2.4 所示。

代码 2.3 动物鱼的类：Fish.java

```
public class Fish extends Animal {
    public static void main(String[] args) {
        Fish fish = new Fish();
    }
}
```

//创建对象

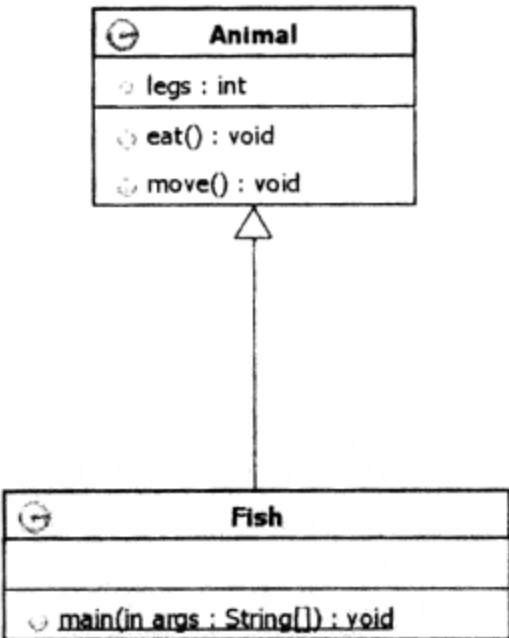


图 2.4 类间的关系

```
        fish.eat();           //调用 eat() 方法
        fish.move();         //调用 move
    }
}
```

运行 Fish.java 类，控制台窗口如图 2.5 所示。



图 2.5 运行结果

【代码解析】

在上述代码中，由于 Fish 类继承了 Animal 类，所以即使在该类中没有定义 eat()和 move()方法，仍然可以在 Fish 类的对象 fish 中调用这些方法。

2.3.4 实现多态

面向对象的多态性表现在父类中定义的属性和方法被子类继承后，可以具有不同的数据类型或表现出不同的行为。即同一个属性或方法在父类及其子类中具有不同的含义。

在程序中如何利用多态的概念呢？设计 3 个类 Animal、Bird 和 Fish，它们的关系如图 2.6 所示。

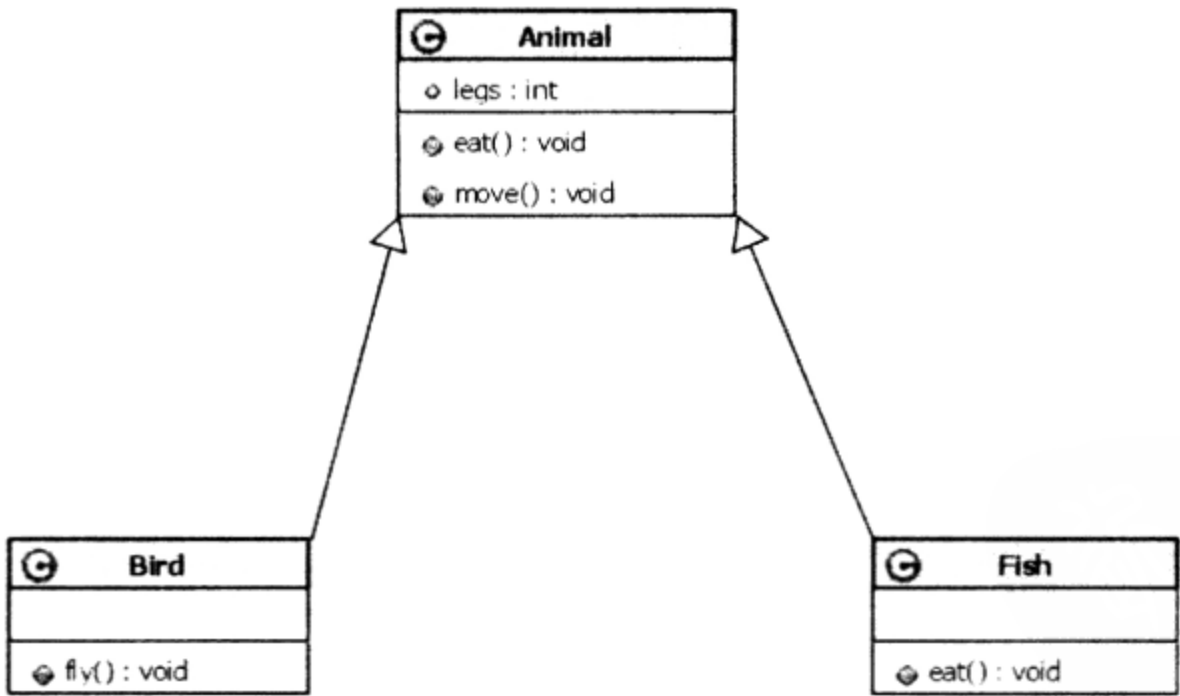


图 2.6 类的关系

(1) 在具体设计 Bird 类时，该类除了继承 Animal 类外，而且还新增加了一个名为 fly()的方法。Bird（鸟）类的具体内容如代码 2.4 所示。

代码 2.4 动物鸟的类：Bird.java

```
public class Bird extends Animal {
    public void fly(){           //创建 fly() 方法
    }
```



```

        System.out.println("飞");
    }
}

```

(2) 在具体设计 Fish 类时, 该类除了继承 Animal 类外, 而且还修改了一个名为 eat() 的方法。Fish (鱼) 类的具体内容如代码 2.5 所示。

代码 2.5 动物鱼的类: Fish.java

```

public class Fish extends Animal {
    public void eat(){ //修改 eat() 方法
        System.out.println("鱼吃");
    }
}

```

(3) 创建一个测试多态的类 polytest, 该类的具体内容如代码 2.6 所示。

代码 2.6 测试多态: polytest.java

```

public class polytest {
    public static void main(String[] args) {
        System.out.println("类 Bird" + "-----");
        Animal birdani1 = new Bird(); //以动物的观点创建一个鸟对象
        //birdani1.fly(); //调用 fly() 方法
        System.out.println("-----");
        Bird birdani2 = new Bird(); //创建一个鸟对象
        birdani2.fly(); //调用 fly() 方法
        System.out.println("类 Fish" + "-----");
        Animal fishan1 = new Fish(); //以动物的观点创建一个鱼对象
        fishan1.eat(); //调用 eat() 方法
        System.out.println("-----");
        Animal ani = new Animal(); //创建一个动物对象
        ani.eat(); //调用 eat() 方法
    }
}

```

运行 polytest.java 类, 控制台窗口如图 2.7 所示。

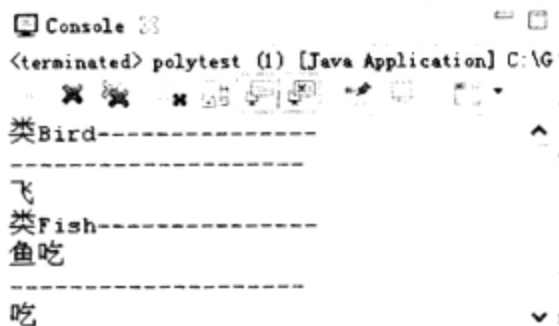


图 2.7 运行结果

【代码解析】

- ❑ 在上述代码中, 既然把对象 birdan1 当作是 Animal 类的对象, 所以当调用 Bird 类中特有的 fly()方法时, 就会出现错误。
- ❑ 在上述代码中, 虽然把对象 fishan1 当作是 Animal 类的对象, 但是在具体调用 eat()方法时输出的却是 Fish.eat()方法中的内容。

2.3.5 实现封装

面向对象的核心之一就是将对对象的属性和方法封装起来，从而实现对它们的保护。下面将通过修改 `Animal.java` 类文件来讲解如何保护 `legs` 属性，具体内容如代码 2.7 所示。

代码 2.7 封装后的动物类：EncapAnimal.java

```
public class EncapAnimal {
    private int legs;
    public EncapAnimal() {                //无参构造函数
        setLegs(2);
    }
    public EncapAnimal(int l) {           //有参构造函数
        setLegs(l);
    }
    public int getLegs() {                 //属性 legs 的 getLegs() 方法
        return legs;
    }
    public void setLegs(int legs) {        //属性 legs 的 setLegs() 方法
        if(legs!=0&&legs!=2&&legs!=4) {    //判断属性的值
            System.out.println("动物的脚个数出错");
            return;
        }
        this.legs = legs;
    }
    public void eat() {                   //吃的方法
        System.out.println("吃");
    }
    public void move() {                  //移动的方法
        System.out.println("移动");
    }
}
```

【代码解析】

在上述代码中添加的 `setLegs()` 方法，实现了在创建动物对象时，如果动物的脚不是 0、2 和 4 只情况，就会输出“物的脚个数出错”的提示。

为了便于测试修改后的 `EncapAnimal` 类是否具有保护作用，创建了一个名为 `EncapAnimalTest` 的测试类，具体内容如代码 2.8 所示。

代码 2.8 测试类：EncapAnimalTest.java

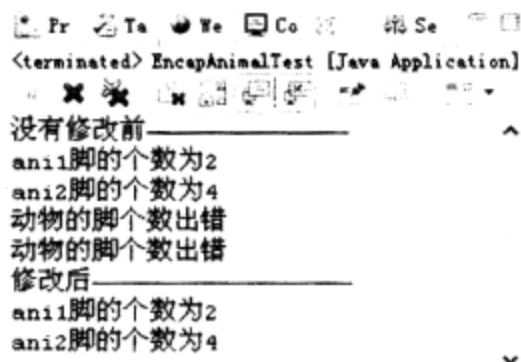
```
public class EncapAnimalTest {
    public static void main(String arg[]) {
        //创建两个对象
        EncapAnimal ani1 = new EncapAnimal();
        EncapAnimal ani2 = new EncapAnimal(4);
        System.out.println("没有修改前————");
        //输出相应对象的脚的个数
        System.out.println("ani1 脚的个数为" + ani1.getLegs());
        System.out.println("ani2 脚的个数为" + ani2.getLegs());
        //修改相应对象的脚的个数
        ani1.setLegs(-1);
    }
}
```

```

        ani2.setLegs(10);
        System.out.println("修改后—————");
        //输出相应对象的脚的个数
        System.out.println("ani1 脚的个数为" + ani1.getLegs());
        System.out.println("ani2 脚的个数为" + ani2.getLegs());
    }
}

```

运行 EncapAnimalTest.java 类，控制台窗口如图 2.8 所示。



```

<terminated> EncapAnimalTest [Java Application]
没有修改前
ani1脚的个数为2
ani2脚的个数为4
动物的脚个数出错
动物的脚个数出错
修改后
ani1脚的个数为2
ani2脚的个数为4

```

图 2.8 运行结果

【代码解析】

在上述代码中，当通过 setLegs() 方法修改 ani1 和 ani2 对象的脚个数为 -1 和 10 时，则输出了“动物的脚个数出错”的信息。这表示 EncapAnimal 类通过封装拥有了保护属性 legs 的功能。

2.4 小 结

本章详细讲解了面向对象思想和 Java 中面向对象的实现。在具体讲解面向对象思想时，首先介绍了面向对象思想的一些基本概念，然后分析了面向对象思想中的组成对象——类和对象，最后分析了面向对象继承性、多态性和封装性的一些特性。由于需要通过 Java 语言来编写程序，所以在 2.3 节还详细介绍了如何通过 Java 语言实现面向对象思想的内容。

第3章 Java 新特性

到了 Java SE 6 版本后，虽然 Java 语言的多数语法没有改变，但是却包含了许多新特性。这些新特性的出现，不仅丰富了 Java 语言的语法，而且更便于程序员开发应用程序，因此 Java 的新特性非常值得探索。

在本章将详细讲解 Java 的新特性。本章的学习目标如下：

- ☐ Java 的一些简单新特性；
- ☐ 枚举；
- ☐ 泛型；
- ☐ 注解；
- ☐ 反射；
- ☐ 动态代理；
- ☐ I/O 的一些新特性；
- ☐ 线程的新特性；
- ☐ 内置的关系数据库 Java DB。

3.1 Java 的一些简单新特性

在具体讲解 Java 的新特性时，先从一些简单新特性讲起。对于这些新特性，如果使用得恰当，可以很好地简化 Java 程序的编写，使程序看起来更加简单、合理。本节涉及的新特性有：

- ☐ 静态导入；
- ☐ 可变参数函数；
- ☐ 增强版 for 循环；
- ☐ 基本数据的拆装箱操作（autoboxing 和 unboxing）。

3.1.1 静态导入

在以前版本的 Java 语法中，关键字 `import` 只能导入一个类或包中的所有类，而最新特性中还可以导入静态方法和静态域。所谓导入，它并不占用系统内存的任何资源，而只是在编写代码时不需要写前缀中的包名。

静态导入语句的语法与 `import` 语句类似，基本格式如下：

```
import static 包名.类名.*
```

例如 `import static java.lang.System.*` 的含义，可以直接使用 `System` 类的静态方法和静态域，即 `out.println("test");` 相当于 `System.out.println("test");`。

还有两种其他形式，分别如下：

```
import static 包名.类名.类变量的名字
import static 包名.类名.类方法的名字
```

下面通过一个具体实例来演示静态导入，具体步骤如下。

(1) 创建一个名为 `StaticClass.java` 的静态类，具体内容如代码 3.1 所示。

代码 3.1 静态类：StaticClass.java

```
package com.cjg.StaticImport;
public class StaticClass {
    public static int MAX12 = 100;           //静态变量
    public static void daying(int x)        //静态方法
    {
        System.out.println(x);
    }
}
```

(2) 创建一个名为 `StaticImport.java` 的类，演示如何使用静态导入，具体内容如代码 3.2 所示。

代码 3.2 测试静态导入：StaticImport.java

```
//静态导入 StaticClass 类下的静态方法和静态变量
import static com.cjg.StaticImport.StaticClass.*;
import static java.lang.Math.abs; //静态导入 Math 类下的静态方法 abs()
public class StaticImport {
    public static void main(String[] args) {
        System.out.println(MAX12); //调用 StaticClass 类下的静态变量 MAX12
        daying(5);                  //调用 StaticClass 类下的静态方法 daying()
        System.out.println(abs(-4)); //调用 Math 类下的静态方法 abs()
    }
}
```

运行 `StaticImport.java` 类，控制台窗口如图 3.1 所示。




图 3.1 运行结果

【代码解析】

- 上述代码中，`import static com.cjg.StaticImport.StaticClass.*` 导入了 `StaticClass` 类中的类方法 `daying()` 和类变量 `MAX12`。所以在具体调用时，只需要直接写类方法名（`daying(5)`）和类变量名（`MAX12`），而不需要写成 `staticclas.daying(5)` 和 `StaticClass.MAX12`。

- 上述代码中, `import static java.lang.Math.abs` 只导入了类 `Math` 中的类方法 `abs()`。所以在具体调用时, 如果直接写 `acos(4)` 类方法时就会报错, 但是直接写 `abs(-4)` 类方法就不会报错。

 **注意:** 当修改上述项目的编译版本为 1.4 时, 上述项目中的 `StaticImport.java` 代码就会报错, 这是因为 JRE1.4 不支持静态导入语法。

最后关于静态导入, 还有以下需要注意的地方:

- 针对一个给定的包, 不可能用一行语句静态地导入包中所有类的所有类方法和类变量。也就是说, 不能这样编写代码:

```
import static java.lang.*;
```

- 如果一个本地方法和一个静态导入方法有着相同的名字, 则本地方法被调用。如果静态导入两个类中同名的类变量或类方法, 则必须通过对象或类名使用类变量、类方法。

3.1.2 可变参数函数

方法重载是 Java 和其他面向对象语言最具特色的特性之一, 该语法特性可以让相同的方法接受各种参数, 但是该语法特性并不能解决函数参数个数不确定的问题。为了解决上述问题, 出现了可变参数函数语法特性, 即函数接受参数的个数不确定。

可变参数函数的语法与普通函数类似, 基本格式如下:

```
函数修饰符 返回类型 函数名 (参数类型...参数名) {  
}
```

在上述基本格式中, 可变参数只能位于参数列表的最后, “...” 只能位于参数类型和参数名之间。例如:

```
public int add(int x,int y,int...z){  
}
```

下面将通过一个具体实例来演示可变参数函数, 具体步骤如下。

(1) 创建一个名为 `VariableArgument.java` 带有可变参数函数的类, 具体内容如代码 3.3 所示。

代码 3.3 可变参数函数类: `VariableArgument.java`

```
public class VariableArgument {  
    public static int add(int...x) {                //可变参数函数  
        int sum = 0;                                //定义一个变量  
        for (int i = 0; i < x.length; i++) {        //循环  
            sum = sum + x[i];  
        }  
        return sum;                                //返回变量 sum  
    }  
}
```

(2) 创建一个名为 `VariableArgumentTest.java` 的类演示如何使用可变参数函数, 具体

内容如代码 3.4 所示。

代码 3.4 调用可变参数函数: VariableArgumentTest.java

```
public class VariableArgumentTest {
    public static void main(String[] args) {
        int sum = 0; //创建变量 sum
        sum = VariableArgument.add(1); //调用带有一个参数的可变参数函数
        System.out.println(sum); //输出相应信息
        sum = VariableArgument.add(1, 2, 3, 4, 5); //调用带有 5 个参数的可变参数函数
        System.out.println(sum); //输出相应信息
    }
}
```

运行 VariableArgumentTest.java 类, 控制台窗口如图 3.2 所示。



图 3.2 运行结果

【代码解析】

通过 VariableArgument.add()方法的调用, 可以看出这个可变参数函数既可以是 1 个参数也可以是 5 个参数, 即函数中参数的个数是不确定的。编译器在具体处理可变参数函数时, 会为该参数创建一个数组, 以数组的形式访问可变参数。

3.1.3 增强版 for 循环

在以前版本的 Java 语法中, 循环流程关键字 for 的语法比较复杂。为了改变该现象, 于是出现了 for 的另一种语法 (增强版 for), 该方式可以简化 for 的书写。

增强版 for 的基本格式如下:

```
for (type 变量名: 集合变量名) {
    // ...
}
```

在上述基本格式中, 迭代变量必须在方法体中定义, 集合变量除了可以是数组, 还可以是实现了 Iterable 接口的集合类。

下面通过修改 VariableArgument.java 类来演示增强版的 for 语法, 具体内容如代码 3.5 所示。

代码 3.5 增强版 for 类: VariableArgument.java

```
public class VariableArgument {
    public static int add(int... xs) { //定义一个实现相加的方法
        int sum = 0; //定义变量 sum
        for (int x : xs) { //加强版循序
            sum += x;
        }
        return sum;
    }
}
```

```

        sum = sum + x;
    }
    return sum;                //返回变量 sum
}
}

```

【代码解析】

上述代码中通过 `for(int x : xs)` 语句代替了 `for (int x = 0; x < xs.length; x++)` 语句，实现了相应的循序迭代功能，即运行 `VariableArgumentTest.java` 类，会显示出正确的信息。

3.1.4 基本数据的拆、装箱操作（autoboxing 和 unboxing）

在面向对象的语法中，处理的对象一般都是对象。但是基本数据类型却不是对象，即用 `int`、`double` 和 `boolean` 等定义的变量都不是对象。在以前的版本中，为了解决基本数据类型转换为对象的问题，出现了打包类型。

为了方便基本数据类型与对象间的转换，在最新的版本中出现了基本数据的自动拆、装箱操作，下面将通过 `AutoUnbox.java` 类来演示基本数据的自动拆、装箱操作功能，具体内容如代码 3.6 所示。

代码 3.6 自动拆、装箱操作：AutoUnbox.java

```

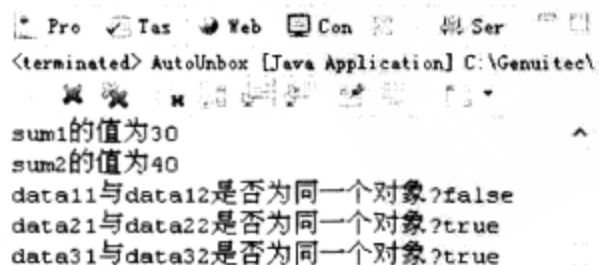
public class AutoUnbox {
    public static void main(String[] args) {
        Integer data11 = new Integer(10);    //实现 int 类型转换成对象方式
        Integer data12 = new Integer(10);
        Integer data21 = Integer.valueOf(10); //实现 int 类型转换成对象方式
        Integer data22 = Integer.valueOf(10);
        //通过自动装箱方式实现 int 类型转换成对象方式
        Integer data31 = 20;
        Integer data32 = 20;
        //通过自动拆箱方式实现对象转换成 int 类型方式
        int sum1 = data11 + 20;
        int sum2 = data31 + 20;
        //输出变量 sum1 和 sum2 的值
        System.out.println("sum1 的值为" + sum1);
        System.out.println("sum2 的值为" + sum2);
        //查看对象是否引用同一个对象
        System.out.println("data11 与 data12 是否为同一个对象?" + (data11 == data12));
        System.out.println("data21 与 data22 是否为同一个对象?" + (data21 == data22));
        System.out.println("data31 与 data32 是否为同一个对象?" + (data31 == data32));
    }
}

```

运行 `AutoUnbox.java` 类，控制台窗口如图 3.3 所示。

【代码解析】

- 通过上述代码可以发现，如果想把 `int` 类型的 10 转换成 `Integer` 类对象，可以通过 3 种方式实现，新建 `Integer` 类方式（`new Integer()`）、`Integer`



```

Pro Tas Web Con Ser
<terminated> AutoUnbox [Java Application] C:\Genuitec\
sum1的值为30
sum2的值为40
data11与data12是否为同一个对象?false
data21与data22是否为同一个对象?true
data31与data32是否为同一个对象?true

```

图 3.3 运行结果

类的 `valueOf()` 方法 (`Integer.valueOf()`) 和自动装箱操作 (`Integer data31 = 20`)。最后一种方式最简单。

- 在上述代码中的 “`int sum1 = data11 + 20`” 语句实现了对象 `data11` 与基本 `int` 类型 20 相加。为了使该句代码不出错，首先需要把对象 `data11` 自动转换成基本 `int` 类型 10，然后才能实现与基本 `int` 类型 20 的相加功能，即所谓的自动拆箱功能。
- 在上述最后 3 句代码中，通过 “`==`” 符号来判断两个对象是否引用同一个对象。对象 `data11` 与对象 `data12` 之所以不是引用同一个对象，是因为这两个对象都是通过关键字 `new` 来创建。但是剩余两对对象在数据小于 127 的情况下（分别为 10 和 20）却引用同一个对象，这是因为对象 `data21` 与 `data22` 是通过 `Integer.valueOf()` 方法返回同一个数值的引用，而对象 `data31` 与 `data32` 是通过自动装箱操作返回同一个数值的引用。

当修改 `AutoUnbox.java` 类中的数值大小时，会出现不同的结果，修改后的具体内容如代码 3.7 所示。

代码 3.7 数值大小: `NumberTest.java`

```
public class NumberTest {
    public static void main(String[] args) {
        //实现 int 类型转换成对象方式
        Integer data21 = Integer.valueOf(127);
        Integer data22 = Integer.valueOf(127);
        Integer data211 = Integer.valueOf(128);
        Integer data221 = Integer.valueOf(128);
        //创建 Integer 类型对象
        Integer data31 = -128;
        Integer data32 = -128;
        Integer data311 = -129;
        Integer data321 = -129;
        //输出相应信息
        System.out.println("data21 与 data22 是否为同一个对象?" + (data21 == data22));
        System.out.println("data211 与 data221 是否为同一个对象?" + (data211 == data221));
        System.out.println("data31 与 data32 是否为同一个对象?" + (data31 == data32));
        System.out.println("data311 与 data321 是否为同一个对象?" + (data311 == data321));
    }
}
```

运行 `NumberTest.java` 类，控制台窗口如图 3.4 所示。



```
<terminated> numbertest [Java Application] C:\Genuitec
data21与data22是否为同一个对象?true
data211与data221是否为同一个对象?false
data31与data32是否为同一个对象?true
data311与data321是否为同一个对象?false
```

图 3.4 运行结果

【代码解析】

通过上述代码的运行结果可以发现，当通过自动装箱方式返回同一数值的对象时，如

果该数值在-128~127 之间（包含它们自己），返回的对象会引用同一对象；否则相反。

3.2 枚 举

在 Java 语言刚发布时，Sun 公司宣称去掉了 C 语言中臃肿、无用的语法，例如指针、枚举等。随着时间的发展，Java 的应用越来越广，Sun 公司不得不使 Java 语言重新支持以前认为臃肿、无用的语法，其中就包含枚举。

3.2.1 枚举的实现原理

在具体编写的项目中，如何定义星期几或性别的变量呢？对于星期一到星期日，有些程序员用 1~7 表示，也有些程序员用 0~6 表示等。当一个程序员开发项目时，星期变量值为任何形式的值都可以。但是当多个程序员共同开发项目时，如果不把星期变量的值统一起来则会报错。

为了解决上述问题，可以使用枚举。枚举的最大作用就是让某种类型变量的取值只能为若干个固定值中的一个，否则编译器将报错。为了能够深刻理解枚举的作用，下面通过普通类来实现枚举的功能，具体步骤如下。

（1）创建一个模拟星期的 WeekDay.java 类，具体内容如代码 3.8 所示。

代码 3.8 星期类：WeekDay.java

```
public class WeekDay {  
    private WeekDay() {                                //私有构造函数  
    }  
    //定义星期的静态变量  
    public final static WeekDay SUN = new WeekDay();    //星期日常量  
    public final static WeekDay MON = new WeekDay();    //星期一常量  
    public final static WeekDay TUE = new WeekDay();    //星期二常量  
    public final static WeekDay WED = new WeekDay();    //星期三常量  
    public final static WeekDay THU = new WeekDay();    //星期四常量  
    public final static WeekDay FN = new WeekDay();     //星期五常量  
    public final static WeekDay SAT = new WeekDay();    //星期六常量  
    //获取下一天的方法  
    public WeekDay nextDay() {                          //通过分支语句实现方法  
        if (this == SUN) {  
            return MON;  
        } else if (this == MON) {  
            return TUE;  
        } else if (this == TUE) {  
            return WED;  
        } else if (this == WED) {  
            return THU;  
        } else if (this == THU) {  
            return FN;  
        } else if (this == FN) {  
            return SAT;  
        } else {  
            return SUN;  
        }  
    }  
}
```



```

    }
}
//重载 toString() 方法
public String toString() {
    if (this == SUN) {
        return "星期日";
    } else if (this == MON) {
        return "星期一";
    } else if (this == TUE) {
        return "星期二";
    } else if (this == WED) {
        return "星期三";
    } else if (this == THU) {
        return "星期四";
    } else if (this == FN) {
        return "星期五";
    } else {
        return "星期六";
    }
}
}

```

//通过分支语句实现方法

【代码解析】

- 上述代码的无参构造函数之所以是私有修饰符 (private)，是因为不允许程序员自己创建该类的对象。当程序员想使用 WeekDay 类的对象时，则必须使用该类里自己定义的对象。对于这些对象，可以用公有的静态成员变量表示。
- 为了便于程序员操作 WeekDay 类的对象，还定义了两个方法，即用来实现获取下一天的 nextDay() 方法和实现输出功能的 toString() 方法。

(2) 创建一个名为 VirtualEnumTest.java 的类来测试 WeekDay 类，具体内容如代码 3.9 所示。

代码 3.9 测试类: VirtualEnumTest.java

```

public class VirtualEnumTest {
    public static void main(String[] args) {
        WeekDay today = WeekDay.SAT;           //today 变量
        System.out.println(today+"的下一天是"+today.nextDay());
                                                //输出今天的下一天
    }
}

```

运行 VirtualEnumTest.java 类，控制台窗口如图 3.5 所示。



图 3.5 运行结果

【代码解析】

在上述代码中，当创建星期 (WeekDay) 的对象时，只能是该类中已经定义好的类对

象，即只能是 SUN、MON、TUE、WED、THU、FN 和 SAT 对象中的一个，如果为其他对象则会报错。这同样也是枚举所要实现的功能。

3.2.2 枚举的简单应用

Enum（枚举）出现之前，在 Java 的接口或类中经常出现 `public static final` 修饰的常量。为了让程序员能够抛弃这种常量，于是就出现了 Enum 语法。即任何使用常量的地方，例如 3.2.1 节中用 if 代码切换常量的地方等，都可以用 Enum 常量来代替。

查看 API 帮助文档，可以发现 `java.lang.Enum` 类。该类为所有枚举类型的公共基本类，即所有枚举类型类都继承于该类。该类的构造函数如下所示。

```
Protected Enum(String name,int ordinal)
```

上述构造函数中，参数 `name` 表示枚举常量的名称，参数 `ordinal` 表示枚举常量的序数。下面通过一个星期的实例来演示 Enum 的用法，具体内容如代码 3.10 所示。

代码 3.10 星期枚举：EnumTest.java

```
public class EnumTest {
    public static void main(String[] args) {
        WeekDay today = WeekDay.SAT;
        System.out.println("今天是"+today);
        System.out.println("今天是"+today.name());
        System.out.println("今天是"+today.ordinal());
        System.out.println("-----");
        System.out.println("今天是"+WeekDay.valueOf("SAT"));
        System.out.println("-----");
        WeekDay[] days=WeekDay.values();
        System.out.println("星期中包含"+days.length+"天");
        for(WeekDay day:days){
            System.out.println("星期里包含"+day);
        }
    }
    public enum WeekDay{                                //定义了星期的枚举
        SUN,MON,TUE,WED,THU,FN , SAT
    }
}
```

运行 EnumTest.java 类，控制台窗口如图 3.6 所示。



```
Console 1:
<terminated> enumtest [Java Application] D:\My2
今天是SAT
今天是SAT
今天是6
-----
今天是SAT
-----
星期中包含7天
星期里包含SUN
星期里包含MON
星期里包含TUE
星期里包含WED
星期里包含THU
星期里包含FN
星期里包含SAT
```

图 3.6 运行结果

【代码解析】

- ❑ 在定义枚举的代码中，WeekDay 表示星期的类，而“SUN,MON,TUE,WED,THU,FN,SAT”这7个对象表示星期（WeekDay）类的对象。因此在具体创建星期六对象（today）时，可以通过 WeekDay.SAT 来赋值。
- ❑ 创建出 today 对象后，首先测试 Enum 类的一些成员方法。today.name()输出 today 对象在 WeekDay 类中该对象的枚举常量；today.ordinal()输出 today 对象的枚举常量在 WeekDay 类中的序数。
- ❑ 对于 Enum 类，除了一些常用的成员方法外，还有一些常用的类方法。例如 WeekDay.values()方法返回该类中的所有对象；WeekDay.valueOf()方法返回字符串所对应的对象。

综上所述可以发现枚举的本质是类，即可以当成类来使用。在没有枚举之前，虽然可以按照 Java 最基本的编程手段来解决需要用到枚举的地方，但是使用枚举却可以屏蔽枚举值的类型信息，而不需要像 public static final 定义变量那样还必须指定类型。

3.2.3 枚举的高级特性

既然 Enum（枚举）的实质是类，那么 Java 语法中类的一些成员在 Enum 中也可以出现。在以前的 Java 语法中，经常会遇见构造函数和抽象类，那么在 Enum 中也可以出现它们吗？答案是肯定的。

1. 带有构造函数的Enum

虽然 Enum 与类很相似，但是 Enum 存在许多限制。为了能够让读者清楚 Enum 的构造函数，下面通过一个带有构造函数的 Enum 实例来具体讲解，详细内容如代码 3.11 所示。

代码 3.11 带有构造函数的枚举：EnumConsTest.java

```
public class EnumConsTest {
    public static void main(String[] args) {           //主函数
        WeekDay today = WeekDay.SAT;
    }
    public enum WeekDay {                             //星期枚举
        SUN, MON, TUE, WED, THU, FN("星期五"), SAT(); //星期枚举常量
        private WeekDay() {                           //无参构造函数
            System.out.println("没有参数构造函数");
        }
        private WeekDay(String s) {                   //有参构造函数
            System.out.println(s + "有参数构造函数");
        }
    }
}
```

运行 EnumConsTest.java 类，控制台窗口如图 3.7 所示。

【代码解析】

- ❑ 对于枚举的构造函数，必须放在枚举常量的后面，同时构造函数的修饰符必须是 private。枚举类型的自定义构造函数不能覆盖默认执行的构造函数，只会在其后运行。

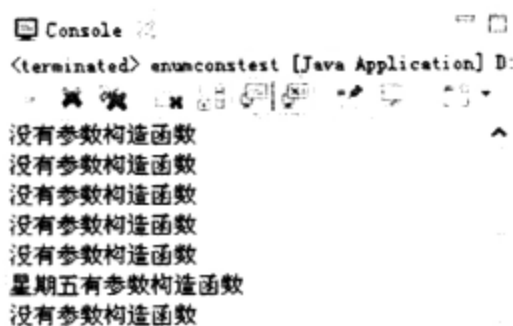


图 3.7 运行结果

- 当初始化对象 FN 时，会调用带参构造函数；当初始化其他对象时，则会调用无参构造函数。枚举类 WeekDay 在初始化相应对象时，根据什么调用相应的构造函数呢？答案是根据枚举常量括号里的参数来决定。当无括号或空括号时，则调用无参构造参数。

2. 带有抽象方法的Enum

虽然 Enum 与类很相似，但是 Enum 存在许多限制。为了能够让读者清楚 Enum 的构造函数，下面通过一个带有构造函数 Enum 的实例来具体讲解，详细内容如代码 3.12 所示。

代码 3.12 带有抽象方法的枚举：EnumConsAbstract.java

```
public class EnumConsAbstract {
    public static void main(String[] args) {                //主方法
        WeekDay today = WeekDay.SAT;
        System.out.println("SAT 的下一天为"+today.nextDay());
    }
    public enum WeekDay {                                    //枚举类
        SUN {                                                //SUN 对象
            public WeekDay nextDay() {
                return MON;
            }
        },
        MON {                                                //MON 对象
            public WeekDay nextDay() {
                return TUE;
            }
        },
        ...                                                  //省略部分代码
    },
    SAT {                                                    //SAT 对象
        public WeekDay nextDay() {
            return SUN;
        }
    };
    public abstract WeekDay nextDay();                      //抽象方法
}
```

运行 EnumConsAbstract.java 类，控制台窗口如图 3.8 所示。

【代码解析】

- 在上述代码中首先创建一个抽象方法 nextDay()，该方法返回的对象为类本身

WeekDay。由于枚举 WeekDay 中包含了抽象方法，所以枚举 WeekDay 也是抽象类。

- ❑ 在具体实现抽象方法 nextDay()时，必须在枚举 WeekDay 的实例中来实现，这是因为抽象类必须在它的子类中实现。
- ❑ 运行完 EnumConsTest.java 类后，打开该项目的 bin 目录可以发现如图 3.9 所示的 class 文件，其中前 7 个类文件为枚举 WeekDay 的类对象。

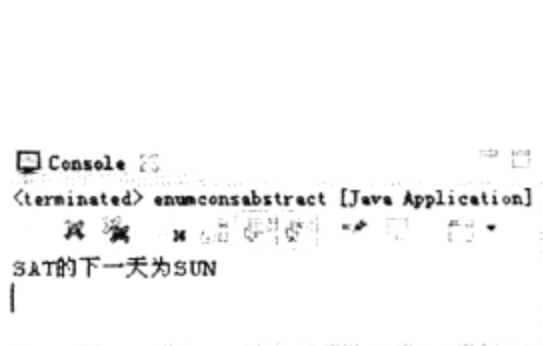


图 3.8 运行结果

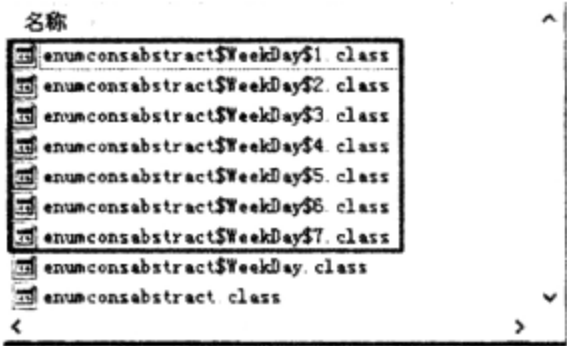


图 3.9 生成的类

3.3 反 射

所谓反射（Reflection），其实就是程序自己能够检查自身信息，就像程序会通过镜子反光来查看自己本身一样。反射使得 Java 语言具有了“动态性”，即程序首先会检查某个类中的方法、属性等信息，然后再动态地调用或动态创建该类或该类的对象。本节将详细介绍反射的相关知识。

3.3.1 反射的基石——Class 类

为什么要出现 Class 类了？注意这是大写的 Class，而不是关键字小写 class。根据面向对象的思想可以知道，Java 语言中的类用于描述一类事物的共性，即该类拥有什么属性，没有什么属性。至于这些属性的值，则是由实现该类的对象来确定的，不同的对象拥有不同的值。既然任何事物都可以用类来表示，那么 Java 中的类可以用一个什么类来表示？Java 中类的对象又是什么呢？

其实从 JDK 1.2 就出现了 Class 类，该类用来描述 Java 中的一切类事物，该类描述了类名字、类的访问属性、类所属于的包名、字段名称的列表、方法名称的列表等。例如 Class 类的 getName()方法可以获取所描述类的类名。

Class 实例代表内存中的一份字节码，所谓字节码就是当 Java 虚拟机加载某个类的对象时，首先需要把硬盘上该类的二进制源码编译成 class 文件的二进制代码（字节码），然后把 class 文件的字节码加载到内存中，之后再创建该类的对象。

那么如何获取 Class 类的对象，即相应类的字节码呢？根据 API 的帮助文档，可以发现以下 3 种方式。

- (1) 最常见的方法为调用相应类对象的 getClass()方法，例如：


```
Data data;
Class dataclass = data.getClass();
```


(2) 还可以通过 Class 类中的静态方法 `forName()`，来获取与字符串对应的 Class 对象，例如：

```
Class dataclass = Class.forName("Data");
```

(3) 还可以通过类名.class 形式来实现，例如：

```
Class dataclass = Data.class
```

 **注意：**第一种方式为对象的方法，而后两种方式为类的方法。

下面通过一个具体的实例 `ClassTest.java` 来演示类 Class 的用法，具体内容如代码 3.13 所示。

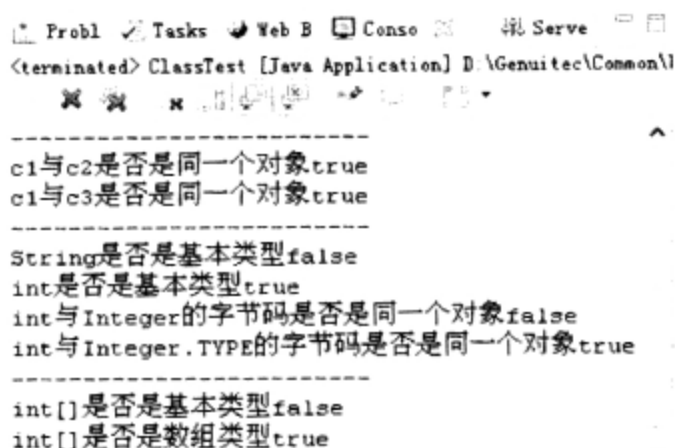
代码 3.13 Class 类的使用：ClassTest.java

```
public class ClassTest {
    public static void main(String[] args) throws ClassNotFoundException,
        InstantiationException, IllegalAccessException {
        String s1= "1234";                //创建一个字符串变量 s1
        //获取 s1 和 String 类的字节码
        Class c1 = s1.getClass();
        Class c2 = String.class;
        Class c3 = Class.forName("java.lang.String");
        //比较字节码是否相同
        System.out.println("-----");
        System.out.println("c1 与 c2 是否是同一个对象" + (c1==c2));
        System.out.println("c1 与 c3 是否是同一个对象" + (c1==c3));
        System.out.println("-----");
        //检测是否为基本类型
        System.out.println("String 是否是基本类型" + String.class.isPrimitive());
        System.out.println("int 是否是基本类型" + int.class.isPrimitive());
        //检测 int 和 Integer 是否指向同一字节码
        System.out.println("int 与 Integer 的字节码是否是同一个对象" + (int.class
            ==Integer.class));
        System.out.println("int 与 Integer.TYPE 的字节码是否是同一个对象" + (int.
            class==Integer.TYPE));
        //数组方面的字节码
        System.out.println("-----");
        System.out.println("int[] 是否是基本类型" + int[].class.isPrimitive());
        System.out.println("int[] 是否是数组类型" + int[].class.isArray());
    }
}
```

运行 `ClassTest.java` 类，控制台窗口如图 3.10 所示。

【代码解析】

- ❑ 在上述代码中首先创建一个字符串变量 `s1`，然后通过获取字节码的 3 种方式获取字符串的字节码，最后通过符号 `==` 比较 3 份字节码是否指向同一个对象。
- ❑ 在 Class 类中有一个名为 `isPrimitive()` 的方法，用来判断字节码的类是否是基本类型。查看 API 帮助文档，可以发现只有 9 种基本类型，分别是 `boolean`、`byte`、`char`、`short`、`int`、`long`、`float`、`double` 和 `void`。所以运行结果里 `String` 不是基本类型，`int[]` 不是基本类型而 `int` 是基本类型。



```

<terminated> ClassTest [Java Application] D:\Genuitec\Common\l
-----
c1与c2是否是同一个对象true
c1与c3是否是同一个对象true
-----
String是否是基本类型false
int是否是基本类型true
int与Integer的字节码是否是同一个对象false
int与Integer.TYPE的字节码是否是同一个对象true
-----
int[]是否是基本类型false
int[]是否是数组类型true

```


图 3.10 运行结果

- int 类型与 Integer 类的字节码是否为同一个对象呢?通过运行结果可以发现它们不为同一个对象,但是 Integer 类中的静态常量 TYPE 却返回 int 基本类型。
- 如果判断是否为数组的字节码,可以通过 Class 类的 isArray()方法来实现。

3.3.2 反射的基本应用

所谓反射就是把 Java 类中的各种成分映射成相应的 Java 类。通过反射,在具体编写程序时,不仅可以动态地生成某个类中所需要的成员,而且还能动态调用相应的成员。查看 API 帮助文档可以发现,不仅一个 Java 类可以用 Class 类的对象表示,而且 Java 类的各种成员,如成员变量、方法、构造方法、包等,也可以用相应的 Java 类表示。

反射一般会涉及如下类:Class(表示一个类的类)、Field(表示属性的类)、Method(表示方法的类)和 Constructor(表示类的构造方法的类)。那么如何获取这些类(除了 Class 类)的对象呢?通过查看 API 帮助文档可以发现,Class 类里存在一系列的方法,来获取相关类中的变量、方法、构造方法、包等信息。

 **注意:** Class 类位于 java.lang 包中,而后面 3 个类都位于 java.lang.reflect 包中。

1. 构造方法的反射

下面通过一个具体的实例 SimpleReflect.java,来演示 Constructor 类的用法,具体内容如代码 3.14 所示。

代码 3.14 构造函数的反射: SimpleReflect.java

```

//导入相应的包
import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
public class ConstructorRef {
    public static void main(String[] args) throws SecurityException,
        NoSuchMethodException, IllegalArgumentException,
        InstantiationException, IllegalAccessException,
        InvocationTargetException {
        //创建字符串的常用方法
        String s1 = new String(new StringBuffer("cjgong"));
        //获取 String 类的构造函数对象

```

```

        Constructor cs1 = String.class.getConstructor (StringBuffer.
        class);
        //通过 Constructor 类对象的方法创建字符串
        String s11 = (String) cs1.newInstance(new StringBuffer("cjgong"));
        // String s12=(String)cs1.newInstance("cjgong");
        System.out.println("-----");
        //输出字符串的一些信息
        System.out.println("s1 对象的第 5 个元素为" + s1.charAt(4));
        System.out.println("s11 对象的第 5 个元素为" + s11.charAt(4));
        System.out.println("-----");
    }
}

```

运行 SimpleReflect.java 类，控制台窗口如图 3.11 所示。

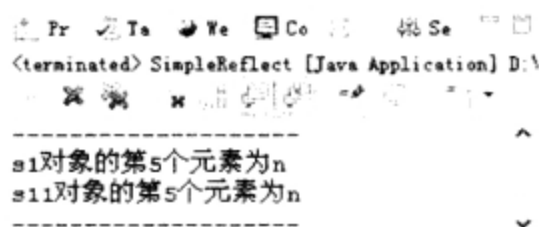


图 3.11 运行结果

【代码解析】

- ❑ 在普通创建字符串的方式中，只要通过 `new String()` 方法就可以实现。
- ❑ 在通过反射方式实现创建字符串的方式中，首先需要获取 `String` 类的构造函数。查看 API 帮助文档，在 `Class` 类中存在 `getConstructor()` 方法，所以可以先通过 `String.class` 获取 `String` 类的字节码，然后再通过 `String.class.getConstructor()` 方法获取 `String` 类的构造函数。
- ❑ 在 `String` 类中存在许多构造函数，那么 `getConstructor()` 方法如何决定生成某个构造函数对象呢？查看 `getConstructor()` 方法的定义，如下：

```
getConstructor(Class <?>... parameterTypes)
```

在上述定义中，参数的类型为类的类型，因此该方法根据传入参数的类型来决定返回的构造函数。

- ❑ 获取到 `String` 类的构造函数后，如何创建 `String` 类的实例对象呢？在 `Constructor` 类中存在一个 `newInstance()` 方法，用来利用构造函数创建一个实例对象。该方法的参数类型必须与获取构造函数的参数类型相同。

2. 成员字段的反射

下面通过一个具体的实例来演示类 `Field` 的用法，具体步骤如下。

(1) 创建一个表示坐标的类 `Point.java`，具体内容如代码 3.15 所示。

代码 3.15 坐标的类：Point.java

```

public class Point {
    //创建 4 个字段
    private int x;
    public int y;
}

```

```

public String s1="abababab";
public String s2="aaaabbbb";
public Point(int x, int y) { //构造函数
    super();
    this.x = x;
    this.y = y;
}
public Point(int x, int y, String s1, String s2) { //构造函数
    super();
    this.x = x;
    this.y = y;
    this.s1 = s1;
    this.s2 = s2;
}
public String toString(){ //重写 toString() 函数
    return "s1 的值为"+s1+" : "+s2 的值为"+s2;
}
}

```

(2) 创建一个名为 FieldRef.java 的类，该类实现对成员字段的反射，具体内容如代码 3.16 所示。

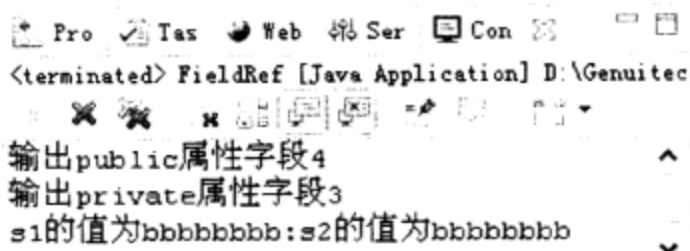
代码 3.16 成员字段反射的类: FieldRef.java

```

public class FieldRef {
    public static void main(String[] args) throws SecurityException,
        NoSuchFieldException, IllegalArgumentException,
        IllegalAccessException {
        Point point = new Point(3, 4); //定义一个坐标
        //获取字段 y 的值
        Field fieldY = point.getClass().getField("y"); //获取类中的类字段
        System.out.println("输出 public 属性字段" + fieldY.get(point)); //获取对象中的值
        //获取字段 y 的值
        Field fieldX = point.getClass().getDeclaredField("x");
        //获取类中的类字段
        fieldX.setAccessible(true); //改变类字段的属性
        System.out.println("输出 private 属性字段" + fieldX.get(point));
        //调用 chang() 方法
        chang(point);
        System.out.println(point); //输出对象 point
    }
    //通过反射改变字段中的字母方法
    public static void chang(Object obj) throws IllegalArgumentException,
        IllegalAccessException {
        Field[] fields = obj.getClass().getFields(); //获取所有成员字段
        for (Field field : fields) { //遍历字段数组
            if (String.class == field.getType()) { //当类型为 Sting 类型时
                String oldValue = (String) field.get(obj); //获取成员字段的值
                String newValue = oldValue.replace('a', 'b'); //实现替换
                field.set(obj, newValue); //设置成员字段的值
            }
        }
    }
}

```

运行 FieldRef.java 类，控制台窗口如图 3.12 所示。



```
<terminated> FieldRef [Java Application] D:\Genuitec
输出public属性字段4
输出private属性字段3
s1的值为bbbbbbbb:s2的值为bbbbbbbb
```

图 3.12 运行结果

【代码解析】

- ❑ 在上述代码中，可以通过“`point.getClass().getField("y")`”，获取 `point` 对象字节码中字段 `y` 的 `Field` 对象 `fieldY`。`fieldY` 对象是 `Point` 类上面的成员字段，而不是 `point` 对象上的成员字段，这是因为类只有一个，而类的实例对象却有多，如果对应到对象的成员字段上，则没办法确定关联到哪个对象上。
- ❑ 由于 `fieldY` 对象是 `Point` 类上面的成员字段，所以如果要得到 `point` 对象字段 `y` 的值，必须通过以 `point` 对象为参数的 `get()` 方法来实现。
- ❑ 通过反射方法 `getField()` 得到的 `Field` 对象，只能是 `Public` 修饰的字段。如果想获取其他属性字段的 `Field` 对象，则必须通过 `getDeclaredField()` 方法。获取到 `Field` 对象后，如果想获取相应对象上该字段的值，还必须通过 `setAccessible()` 进行设置。
- ❑ 在上述代码中还存在一个方法 `chang()`，该方法主要用来实现把一个对象中所有 `String` 类型的成员字段，所对应的字符串中的 `a` 改成 `b`。

3. 成员方法的反射

下面通过一个具体的实例 `MethodRef.java` 来演示 `Method` 类的用法，具体内容如代码 3.17 所示。

代码 3.17 Method 类的用法：MethodRef.java

```
public class MethodRef {
    public static void main(String[] args) throws SecurityException,
        NoSuchMethodException, IllegalArgumentException,
        IllegalAccessException, InvocationTargetException {
        String s1 = "abcdef"; //定义一个字符串
        //获取 String 类中参数为 int 的 charAt() 方法
        Method methodCharAt = String.class.getMethod("charAt", int.class);
        //对象 s1 调用 charAt() 方法
        System.out.println("对象 s1 中第二个字母为" + methodCharAt.invoke(s1,
            1));
        //对象 s1 调用 charAt() 方法
        System.out.println("对象 s1 中第二个字母为" + methodCharAt.invoke(s1,
            new Object[]{ 1 }));
    }
}
```

运行 `MethodRef.java` 类，控制台窗口如图 3.13 所示。

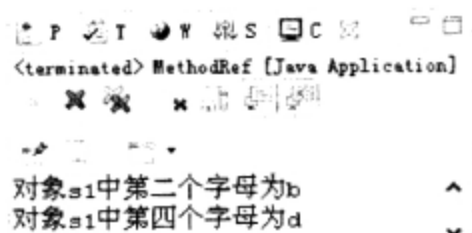


图 3.13 运行结果

【代码解析】

- 在上述代码中，首先创建了一个字符串对象 `s1`，接着通过 `s1` 字节码对象的 `getMethod()` 方法获取 `String` 类中带有 `int` 类型参数的 `charAt()` 方法。
- 通过反射方式获取 `Method` 对象后，如果想调用该方法，可以通过 `invoke()` 方法来实现。查看 API 帮助文档，`invoke()` 方法的定义如下：

```
public Object invoke(Object obj,
                    Object... args)
```

在上述定义中，第一个参数为实体对象，第二参数为方法调用所需要的参数。

通过上述代码可以知道，通过调用 `invoke()` 方法可以实现调用实体对象的 `method()` 方法，如果传递给 `invoke()` 方法的第一个参数为 `null` 对象，还有意义吗？答案是肯定的，这说明该 `Method` 对象对应的是一个静态方法。

下面通过一个具体的实例来演示如何通过反射方式调用类的 `main()` 方法，具体步骤如下。

- (1) 创建一个带有 `main()` 方法的类 `StaticMain`，具体内容如代码 3.18 所示。

代码 3.18 带有静态方法类：StaticMain.java

```
public class StaticMain {
    public static void main(String[] args) {
        System.out.println("-----");           //输出相应信息
        for (String arg : args) {                  //遍历参数
            System.out.println(arg);
        }
    }
}
```

- (2) 创建 `StaticMainRef` 类，该类通过反射调用 `StaticMain` 类中的 `main()` 方法，具体内容如代码 3.19 所示。

代码 3.19 带有静态方法类：StaticMainRef.java

```
public class StaticMainRef {
    public static void main(String[] args) throws SecurityException,
        IllegalArgumentException, NoSuchMethodException,
        ClassNotFoundException, IllegalAccessException,
        InvocationTargetException {
        //调用类的静态方式
        StaticMain.main(new String[] { "111", "222", "333", "444" });
        //通过反射调用类的静态方法
        startClass("com.cjg.method.StaticMain");
    }
}
```

```

//通过反射调用类的静态方法的方法
public static void startClass(String className) throws Security-
Exception,
    NoSuchMethodException, ClassNotFoundException,
    IllegalArgumentException, IllegalAccessException,
    InvocationTargetException {
    //获取相应类的 main() 方法
    Method mainMethod = Class.forName(className).getMethod("main",
        String[].class);
    //执行 main() 方法
    mainMethod.invoke(null, new Object[] { new String[] { "111", "222",
        "333", "444" } });
    //执行 main() 方法
    mainMethod.invoke(null, (Object) new String[] { "111", "222", "333",
        "444" });
}
}

```

运行 StaticMainRef.java 类，控制台窗口如图 3.14 所示。



```

-----
111
222
333
444
-----
111
222
333
444
-----
111
222
333
444

```

图 3.14 运行结果

【代码解析】

- ❑ 在上述代码中首先通过普通方式（StaticMain.main(new String[] { "111", "222", "333", "444" });），调用 StaticMain 类中的 main() 方法。
- ❑ 在上述代码中存在一个 startClass() 方法，该方法主要用来通过反射调用静态方法。该方法需要传入一个表示类的参数，即 “tartClass("com.cjg.method.StaticMain")” 中，参数 com.cjg.method.StaticMain 为一个具体的类。

3.3.3 反射的高级应用

既然 Java 类中的各种成分可以映射成相应的 Java 类，那么数组也可以反射成 Array 类吗？查看 API 帮助文档可以发现，Array 类为数组的字节码。同时还存在一个名为 Arrays 的类，该类主要用来实现数组的各种操作。本节主要讲解数组等集合方法的反射，具体步骤如下。

(1) 通过一个具体的实例 ArrayRef.java 来演示数组反射的用法, 具体内容如代码 3.20 所示。

代码 3.20 数组的反射: ArrayRef.java

```
public class ArrayRef {
    public static void main(String[] args) {
        //创建了 4 种类型的数组
        int[] a1 = new int[] { 1, 2, 3 }; //一维 int 型数组
        int[] a2 = new int[4]; //一维 int 型数组
        int[][] a3 = new int[2][3]; //二维 int 型数组
        String[] a4 = new String[] { "1", "2", "3" }; //一维 String 型数组
        System.out.println("-----");
        //判断数组 a1 和数组 a2 的字节码是否相同
        System.out.println("a1 与 a2 数组的字节码是否相同"+a1.getClass() ==
            a2.getClass());
        //Array 类的一些方法
        System.out.println("-----");
        System.out.println("a3 数组的名字"+a3.getClass().getName());
        System.out.println("a1 数组超类的名字"+a1.getClass().getSuperclass().
            getName());
        //数组与 Object[] 的对应关系
        Object obj1 = a1;
        Object boj3 = a4;
        Object[] boj4 = a4;
        Object boj5 = a3;
        Object[] boj6 = a3;
        System.out.println("-----");
        System.out.println("无工具类 Arrays 的输出"+a1);
        System.out.println("无工具类 Arrays 的输出"+a4);
        System.out.println("-----");
        //调用工具类 Arrays 的 asList() 方法
        System.out.println("有工具类 Arrays 的输出"+Arrays.asList(a1));
        System.out.println("有工具类 Arrays 的输出"+Arrays.asList(a4));
        System.out.println("-----");
        //调用 printObject() 方法
        printObject(a1);
        printObject(1);
    }
    //打印对象中的成员方法
    private static void printObject(Object obj) {
        Class cla = obj.getClass(); //获取字节码
        if (cla.isArray()) { //判断是否为数组
            System.out.println("调用自定义方法的数组的输出");
            int len = Array.getLength(obj); //获取数组的长度
            for (int i = 0; i < len; i++) { //输出数组中的各个成员
                System.out.println(Array.get(obj, i));
            }
            System.out.println("-----");
        } else { //输出相应信息
            System.out.println("调用自定义方法的普通对象的输出");
            System.out.println(obj);
            System.out.println("-----");
        }
    }
}
```

运行 ArrayRef.java 类，控制台窗口如图 3.15 所示。

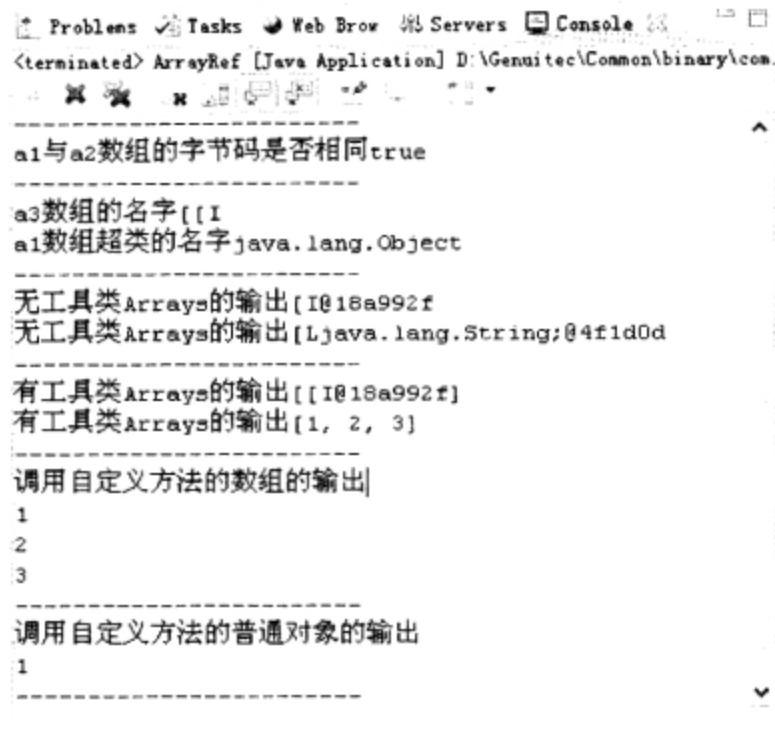


图 3.15 运行结果

【代码解析】

- ❑ 通过“a1 与 a2 数组的字节码相同”的运行结果可以说明，具有相同维数和元素类型的数组的字节码相同，即具有相同的 Class 实例对象。
- ❑ 通过调用 Class 实例对象的 getName()方法可以返回字节码名字，例如“[[I”表示是二维数组，类型为 int。通过调用 Class 实例对象的 getSuperclass()方法可以返回父类，任何数组字节码的父类都是 Object 类对应的字节码，例如 a1 数组父类的名字 java.lang.Object。
- ❑ 当直接输出数组时，得到的输出结果非常不理想。例如数组 a1 的输出结果为“[I@18a992f”，其中“[I”表示为 int 类型数组，“18a992f”表示该对象的 hascode 值；数组 a4 的输出结果为“[Ljava.lang.String;@4f1d0d”，其中“[Ljava.lang.String”表示为 String 类型数组，“4f1d0d”表示该对象的 hascode 值。
- ❑ 通过工具类 Arrays 的 asList()方法可以输出数组中参数的值，例如数组 a4 的输出结果为“[1,2,3]”，但是为什么 a1 的输出结果是“[[I@18a992f”，而不是[1,2,3]呢？这是因为数组 a1 为 int 类型的一维数组，只能转换成 Object 对象，而不能转换成 Object[]对象。即基本类型的一维数组可以被当作 Object 类型使用，而不能当作 Object[]类型使用（“Object[] obj2=a1”是错误）；非基本类型的一维数组，即可以当作 Object 类型使用，又可以当作 Object[]类型使用。
- ❑ 为了避免工具类 Arrays 的 asList()方法的弊端，所以编写了一个名为 printObject 的方法，该方法可以打印出对象的成员，不论该对象是数组还是一个对象。

(2) 上述代码讲解了数组 (Array) 反射的相关方法，那么集合也可以反射吗？答案是肯定的。下面通过一个具体的实例 CollectRef.java 来演示集合反射的用法，具体内容如代码 3.21 所示。

代码 3.21 集合的反射：CollectRef.java

```
public class CollectRef {
    public static void main(String[] args) throws IOException,
```

```

        InstantiationException, IllegalAccessException,
        ClassNotFoundException {
//读取属性文件
InputStream is = new FileInputStream("Config.properties");
Properties props = new Properties();    //创建 Properties 类型对象
props.load(is);
is.close();                            //关闭输入流
String className = props.getProperty("className"); //获取相应的值
//创建相应的集合对象
Collection collections = (Collection) Class.forName(className)
        .newInstance();
//为集合 collections 添加数据
collections.add("1");
collections.add("2");
collections.add("3");
collections.add("4");
System.out.println("collections 集合中的成员" + collections);
    }
}

```

【代码解析】

- ❑ 在上述代码中首先通过 I/O 类读取属性文件，然后获取属性文件中 className 的值。
- ❑ 获取 className 的值后，首先通过 Class.forName() 方法获取该值的字节码，然后通过 newInstance() 方法创建一个实例对象，最后为实例对象添加几个成员对象并输出。

(3) 上述的代码如果想运行成功，还必须在 reflect/src 目录下创建一个名为 Config.properties 的文件，该文件的具体内容如代码 3.22 所示。

代码 3.22 属性文件：Config.properties

```
className=java.util.ArrayList
```

运行 CollectRef.java 类，控制台窗口如图 3.16 所示。

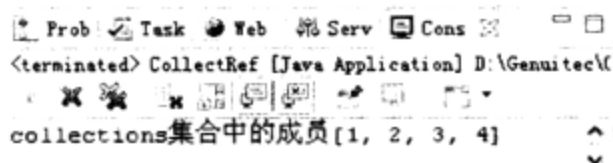


图 3.16 运行结果

3.4 标 注

标注（Annotation）是 Java 语言新出现的一个特性，在实际应用中其可以部分或者全部取代传统的 XML 等部署描述文件。之所以要出现标注特性，是因为部署描述文件很复杂，在具体编写时很容易出错。为了能够彻底地理解标注，本节将详细介绍标注。

3.4.1 标注的简单使用

标注很早以前就应用在 Java 程序的开发中，但是没引起关注。直到作为规范在 JDK 5.0

中发布以后，才逐渐被程序员了解，并有越来越多的框架、技术加入了标注应用。例如 EJB 3 规范（Java EE 5 规范的子集）为 Bean 类型、接口类型、资源引用、事务属性、安全性等定义了标注。JAX-WS 2.0 规范为 Web 服务提供了一组类似的标注。

当使用 MyEclipse 开发设计 Java 程序时，每当进行自动重写一个方法的时候总会在重写的方法上面自动加入一行代码@Override，其实这个@Override 就是一个标注。本节主要讲解标注的作用，具体步骤如下。

(1) 创建了@SuppressWarnings 标注的类 SimpleAnnotation.java，具体内容如代码 3.23 所示。

代码 3.23 学生类：SimpleAnnotation.java

```
public class SimpleAnnotation {
    public static void main(String[] args) { //主方法
        System.runFinalizersOnExit(true); //调用 runFinalizersOnExit() 方法
    }
}
```

【代码解析】

当在命令窗口中通过 javac 命令来编译上述 Java 文件时，会出现如图 3.17 所示的内容，即编译成功，但是却出现了注意的代码。如果想查看源 Java 文件中需要注意的地方，则可以输入-Xlint:deprecation 命令参数，会出现如图 3.18 所示的内容，即源文件中的 runFinalizersOnExit()方法已经过时。



图 3.17 编译源文件



图 3.18 查看注意的代码

(2) 如果有些程序员就是想用已经过时的方法，但是很讨厌编译时出现的注意代码，那么如何实现呢？这时就可以在 SimpleAnnotation.java 文件中加入@SuppressWarnings 标注，具体内容如代码 3.24 所示。

代码 3.24 不限制内容类型的集合：SimpleAnnotation.java

```
public class SimpleAnnotation {
    @SuppressWarnings("deprecation") // @SuppressWarnings() 标注
    public static void main(String[] args) {
        System.runFinalizersOnExit(true);
    }
}
```

【代码解析】

当在命令窗口中通过 javac 命令来编译上述 Java 文件时，会出现如图 3.19 所示的内容，

即不仅编译成功，同时也不再出现注意代码。



图 3.19 编译源文件

通过上述的代码可以发现，其实@SuppressWarnings 标注就是告诉 Java 编译器不需要再提示“注意”，起到压缩警告的作用。

注意：如果上述代码在 MyEclipse 开发工具中编写，则只会在已过时的方法上画一个横线，如图 3.20 所示。

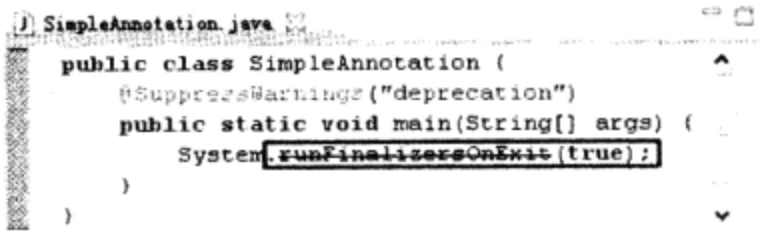


图 3.20 标注

总之，标注其实就相当于一种标志，加上标注就等于打上了某种标记，没加就等于没有某种标记。当 Java 语言的编译器、开发工具等其他程序在具体编译程序时，就会通过“反射”知道类或其他各种元素上是否有标记，然后就会根据标记去实现相应的功能。

3.4.2 JDK 的内置标注

在最新的 JDK 中，Sun 公司已经提供了几个内建的标注，它们分别为@Override、@Depressed 和@SuppressWarnings，本节将详细介绍这些标记的作用。

1. @Override

java.lang.Override 被用作标注方法，主要用于子类在覆盖父类中的方法名时，检测方法名称。如果方法名称正确，则不会有任何提示；否则就会提示错误。下面演示@Override 标注的作用，具体步骤如下。

(1) 创建一个父类 People.java，具体内容如代码 3.25 所示。

代码 3.25 人的类：People.java

```
public class People {  
    public String toString(){ //重写 toString() 方法  
        return "人的名字";  
    }  
}
```

(2) 创建一个继承 People.java 类的学生类 Student.java，具体内容如代码 3.26 所示。

代码 3.26 学生的类：Student.java

```
public class Student extends People {  
    @Override // @Override 标注
```

```

    public String toString1() {
        return "学生的名字";
    }
}

```

【代码解析】


当 `toString1()` 方法上面没有 `@Override` 标注时，该段代码不会报错。但是如果有 `@Override` 标注时，该段代码就会报错。之所以会出现错误，是因为当存在 `@Override` 标注时，编译器就认为该方法为继承类，会从该类的父类中查找是否有与该方法相同的方法。

`@Override` 标注其实就相当于修饰符，与 `public`、`void` 等修饰符一样。其不仅可以放在方法的上边，而且还可以放在方法的前边。因此还可以写成如下的形式：

```

@Override public String toString1() {
    return "学生的名字";
}

```

 注意：`@Override` 是方法标注，只能作用于方法，在覆盖父类方法却又写错了方法名时发挥作用。

2. @Deprecated

对于程序设计员来说，经常会遇到如下的例子：设计了一个包含 `sayHello()` 方法的类 `People.java`，但是经过一段时间发现 `sayHello1()` 方法可以更好、更快地实现相同的功能。这个时候如果去掉 `sayHello()` 方法，那么调用该方法的类则会出现错误。为了兼容以前的类，而又不建议新设计的类使用 `sayHello()` 方法，就需要把 `People.java` 类中的方法 `sayHello()` 作 `@Deprecated` 标注。

下面详细讲解 `@Deprecated` 标注的作用，具体步骤如下。

(1) 设计一个名为 `People.java` 的类，该类的具体内容如代码 3.27 所示。

代码 3.27 人的类：People.java

```

public class People {
    @Deprecated                                //@Deprecated 标注
    public void sayHello() {
        System.out.println("已经过时的方法");
    }
    public void sayHello1() {
        System.out.println("现在的方法");
    }
}

```

【代码解析】

❑ `sayHello1()` 和 `sayHello()` 方法可以实现相同的功能，但是前者为建议使用的方法，而后者为不建议使用的方法，所以后者加上了 `@Deprecated` 标注。

❑ 如果上述代码在 MyEclipse 开发工具中编写，则只会在 `sayHello()` 方法上画一个横线，如图 3.21 所示。

(2) 创建继承 `People.java` 类的学生类 `Student.java`，具体内容如代码 3.28 所示。

代码 3.28 学生的类: Student.java

```

public class Student {
    public static void main() {
        People pl = new People();           //创建 People 类对象
        pl.sayHello();                       //调用 sayHello() 方法
        pl.sayHello1();                      //调用 sayHello1() 方法
    }
}

```

如果上述代码在 MyEclipse 开发工具中编写,则只会在调用 sayHello()方法上画一个横线,如图 3.22 所示,表示该方法不建议使用。

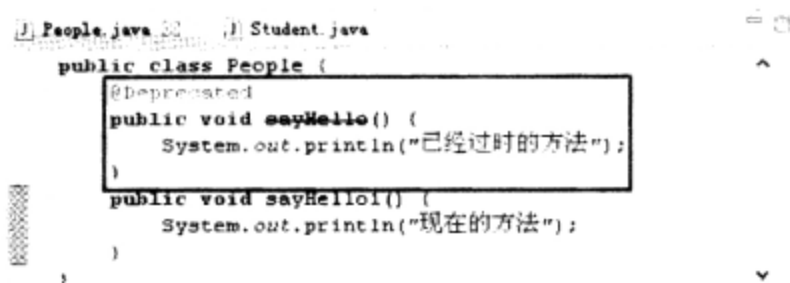


图 3.21 画一个横线的结果

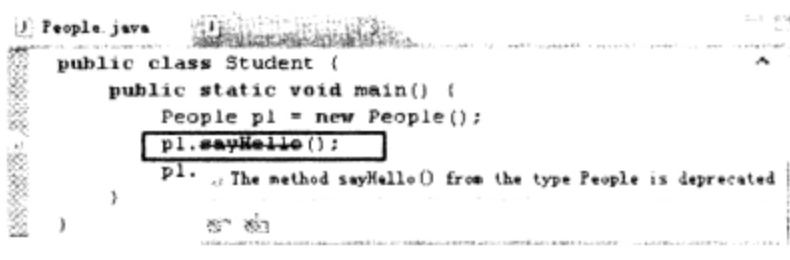


图 3.22 sayHello()方法被画了横线

当一个类或者类成员使用@Deprecated 修饰时,编译器将不鼓励使用这个被标注的程序元素。而且这种修饰具有一定的“延续性”:即在代码中通过继承或者覆盖的方式使用这个过时的类型及成员,虽然继承或者覆盖后的类型及成员并不是被声明为@Deprecated 的,但编译器仍然要报警。

注意: @Deprecated 标注不仅可以用在方法的前边,而且还能用在参数或类的前边。

3. @SuppressWarnings

java.lang.SuppressWarnings 被用作标注类、属性、方法等成员,主要用于屏蔽警告。该标注与前面两个标注的最大不同点在于其带有参数,并且参数不仅可以是一个还可以是多个。参数的值为警告的类型,例如“已经过时警告”的类型为 deprecation、“没有使用警告”的类型为 unused、“类型不安全警告”的类型为 unchecked 等。

注意: 当 @SuppressWarnings 接收的参数为多个值时,必须以数组的方式为参数赋值。例如 @SuppressWarnings({"deprecation","unused","unchecked"}).

3.5 泛型

在 Java 语言的早期版本中,经常会通过对类型 Object 的引用来实现参数的“任意化”,这种“任意化”必然会附着显式强制类型转换,而这种转换是要求程序员在对实际参数类型预先知道的情况下进行的。为了解决上述问题,于是出现了泛型语法。本节将详细介绍反射的相关知识。

3.5.1 为什么要使用泛型

查看 Java 语言 API 的帮助文档，经常会遇到类的后面跟着<E>标识，例如 Vector<E>、ArrayList<E>等，其实<E>标识就是代表泛型。所谓泛型，其本质就是实现参数化类型，也就是说所操作的数据类型被指定为一个参数。

为了能够彻底地理解泛型，下面将通过对集合的操作来讲解泛型出现的原因。在 Java 语言发布初期，程序员经常通过 add() 方法实现把元素添加到集合 Vector 中，具体步骤如下。

(1) 创建学生的类，该类的具体内容如代码 3.29 所示。

代码 3.29 学生类：Student.java

```
public class Student {
    private int stuNum;           //学生编号的属性
    public Student(int number) {   //构造函数
        this.stuNum = number;     //初始化成员变量
    }
    public String toString() {     //重写 toString() 方法
        return " " + this.stuNum;
    }
}
```

(2) 创建 StudentVectory.java 类，该类实现把学生对象添加到集合 Vectory 中，具体内容如代码 3.30 所示。

代码 3.30 不限制内容类型的集合：StudentVectory.java

```
public class StudentVectory {
    public static void main(String[] args) {
        Vector v = new Vector();           //创建集合 Vector 对象
        //创建 4 个学生对象
        Student s1 = new Student(6);
        Student s2 = new Student(7);
        Student s3 = new Student(8);
        Student s4 = new Student(9);
        Integer t = new Integer(10);       //创建一个 Integer 类型对象
        //实现把 5 个对象添加到集合对象 v 中
        v.add(s1);
        v.add(s2);
        v.add(s3);
        v.add(s4);
        //把 Integer 类型的对象 t 添加进集合对象 v 中
        v.add(t);
        //遍历集合 v
        for (int i = 0; i < v.size(); i++) {
            Student s = (Student) v.get(i); //获取学生的编号
            System.out.println(s);
        }
    }
}
```

运行 StudentVectory.java 类，控制台窗口如图 3.23 所示。

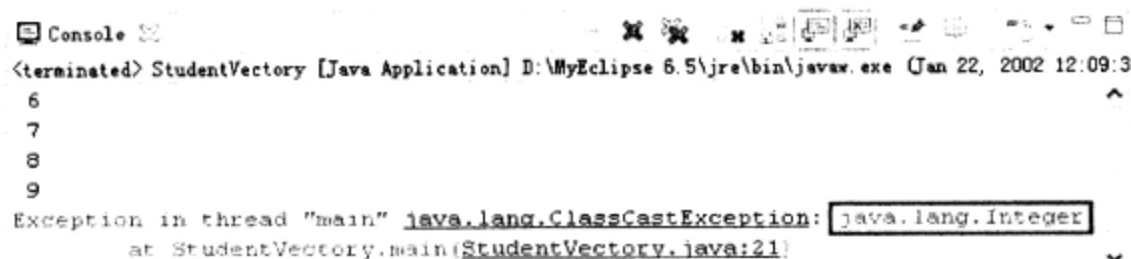


图 3.23 运行结果

【代码解析】

上述代码虽然编译通过，但是在运行时却出现了异常——ClassCastException。这主要是遍历集合对象 v 的最后一个对象时，由于该对象 t 是 Integer 类型对象而不是 Student 类型对象，所以在运行该句 Student s = (Student) v.get(i) 代码时会出错。

(3) 对于上述的代码存在一种安全隐患，即强制类型转换错误时，编译器是不会提示错误的，但是在运行时却会出现异常。为了解决该问题，程序员会限制 Vector 创建的对象 v 中只能添加 Student 类型的对象，于是程序变成如代码 3.31 所示。

代码 3.31 限制内容类型的集合：LimitStudentVectory.java

```

public class LimitStudentVectory {
    private Vector v1 = new Vector();           //创建 Vector 对象 v1
    public void add(Student s) {                 //向 v1 中添加学生功能
        v1.add(s);
    }
    public Student get(int t) {                  //获取学生 ID 号
        return (Student) v1.get(t);
    }
    public int size() {                          //获取集合对象的大小
        return v1.size();
    }
    public static void main(String[] args) {
        //创建 LimitStudentVectory 对象 v
        LimitStudentVectory v = new LimitStudentVectory();
        //创建 4 个学生对象
        Student s1 = new Student(6);
        Student s2 = new Student(7);
        Student s3 = new Student(8);
        Student s4 = new Student(9);
        //创建 1 个 Integer 对象
        Integer t = new Integer(10);
        //添加各个对象到 v 对象
        v.add(s1);
        v.add(s2);
        v.add(s3);
        v.add(s4);
        v.add(t);
        //遍历集合 v
        for (int k = 0; k < v.size(); k++) {
            System.out.println(v.get(k));
        }
    }
}

```

运行 LimitStudentVectory.java 类，控制台窗口如图 3.24 所示。

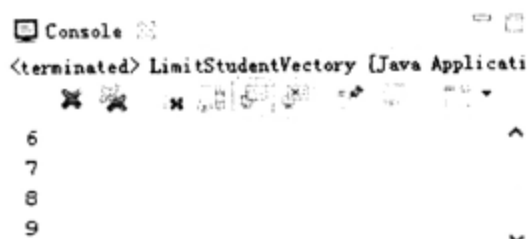


图 3.24 运行结果

【代码解析】

- 在上述代码中，通过定义 `add()` 方法来限制添加的对象必须为 `Student` 类型的对象，通过 `get()` 方法来限制获取的值必须为 `Student` 类型的对象。
- 在上述代码中，如果不注释掉 “`v.add(t)`” 代码，就会出现编译错误。这是因为 `t` 对象不是 `Student` 类型，而是 `Integer` 类型。

(4) 上述的代码虽然解决了 `StudentVectory` 类出现的问题，但是却要写许多的代码。为了解决该问题，可以使用泛型语法。修改后的具体内容如代码 3.32 所示。

代码 3.32 泛型的集合: `GenericStudentVectory.java`

```
public class GenericStudentVectory {
    public static void main(String[] args) {
        Vector<Student> v = new Vector<Student>();
                                //创建存储 Student 类的 Vector 集合对象

        //创建对象 s1、s2、s3 和 s4
        Student s1 = new Student(6);
        Student s2 = new Student(7);
        Student s3 = new Student(8);
        Student s4 = new Student(9);
        Integer t = new Integer(10);           //创建对象 t
        //添加各种对象到集合 v 中
        v.add(s1);
        v.add(s2);
        v.add(s3);
        v.add(s4);
        //v.add(t);
        for (int k = 0; k < v.size(); k++) {    //循环遍历集合 v
            System.out.println(v.get(k));
        }
    }
}
```

运行 `GenericStudentVectory.java` 类，控制台窗口如图 3.25 所示。



图 3.25 运行结果

【代码解析】

在上述代码中，通过 “`Vector<Student> v = new Vector<Student>()`” 限制了集合 `v` 中的

对象只能是 Student 类型。如果一定要添加非 Student 类型对象 (t)，编译器会报错。

最后，通过 LimitStudentVectory 和 GenericStudentVectory 类的比较，可以发现虽然两个类实现了相同的功能，但是后者却比前者的代码更简洁。

3.5.2 泛型的一些特性

在最新版本的 Java 语法中，希望在定义集合类的时候，明确表示要向集合中添加哪种类型的数据。于是 Java 语言 API 的帮助文档中，集合类的后面都跟着 <E> 标识。例如 Vector 类的定义：

```
Vector<E>
```

在上述定义中，此处的 E 指定 Vector 对象中只能存放 E 这种类型的对象。其中 Vector<E> 称为 Vector 泛型类型，E 称为类型变量或类型参数，Vector 称为原始类型。

```
Vector<Integer> v = new Vector< Integer >();
```

在代码中，Vector<Integer> 称为参数化类型，Integer 称为类型参数的实例或实际类型参数，<> 称为 typeof。

到现在为止，已经知道了如何定义泛型和为什么要使用泛型，那么泛型具有什么特性呢？下面详细介绍泛型的特性。

1. 参数化类型与原始类型的兼容

当参数化类型引用一个原始类型的对象时，编译器只是警告而不报错。同样当原始类型引用一个参数化类型对象时，编译器也只是警告而不报错。例如：

```
Vector<String> v = new Vector();  
Vector v = new Vector(String);
```

2. 参数化类型无继承性

为了能够理解该特性，可以先看下面的代码片段：

```
Vector<String> v = new Vector(String);  
Vector<Object> v1 = v;
```

有些程序员认为由于 String 类型是 Object 类型的子类，所以 String 类型的 Vector 对象 (v) 可以赋值给 Object 类型的 Vector 对象 (v1)。

接着看下面的代码片段：

```
v1.add(new Object());  
String s = v.get(0);
```

在上述代码中，首先给 v1 对象中添加一个 Object 类型对象成员，接着由于 v1 与 v 都指向同一个对象，所以可以通过 v 对象获取添加到 v1 对象中的成员。这时就会出现错误，因为 v 对象中的成员不再是 String 类型。所以代码 “Vector<Object> v1 = v” 是错误的。

这就好比教育部提供一个教师信息表给人口普查局，人口普查局可能会向教师信息表里加入学生等信息，这就破坏了教师信息记录。

3. 泛型的“去类型”特性

所谓“去类型”，就是指泛型中的类型只是提供给编译器使用的，当程序编译成功后就会去掉“类型”信息。为了能够彻底理解该特性，下面通过 `AdvancedGeneric.java` 类具体演示，该类的具体内容如代码 3.33 所示。

代码 3.33 实现“去类型”的类：`AdvancedGeneric.java`

```
public class AdvancedGeneric {
    public static void main(String[] args) {
        //创建存储 String 类型的 ArrayList 类对象 arry1
        ArrayList<String> arry1= new ArrayList<String>();
        arry1.add("cjg");           //添加字符串 cjg 到对象 arry1 中
        //创建存储 Integer 类型的 ArrayList 类对象 arry2
        ArrayList<Integer> arry2 = new ArrayList<Integer>();
        arry2.add(27);             //添加对象 27 到对象 arry2 中
        //输出相应信息
        System.out.println("arry1 对象与 arry2 对象是否指向同一份字节码? "+(arry1.
            getClass()==arry2.getClass()));
    }
}
```

运行 `AdvancedGeneric.java` 类，控制台窗口如图 3.26 所示。



```
Console
<terminated> AdvancedGeneric [Java Application] D:\MyEclip
arry1对象与arry2对象是否指向同一份字节码? true
```

图 3.26 运行结果

【代码解析】

- ❑ 泛型的作用只是限制集合中的输入类型，让编译器挡住源程序中的非法输入。即编译器能够辨认出 `arry1` 集合中只能添加字符串对象，而 `arry2` 对象中只能添加整数型对象。当向两个对象中添加其他类型的对象时，编译器就会报错。
- ❑ 代码“`arry1.getClass()==arry2.getClass()`”的结果为 `true`，则说明编译器生成的对象 `arry1` 集合和 `arry2` 集合的字节码为同一个对象。

通过 `AdvancedGeneric.java` 类的运行结果可以发现，编译器编译带参数说明的集合时会去掉“类型”信息。这样有一个好处，程序具体运行时将不会受到泛型的影响。

4. 利用反射绕过泛型的类型限制

由于编译器生成的字节码会去掉泛型的类型信息，所以只要能跳过编译器，还是可以给通过泛型限制类型的集合中加入其他类型数据的。下面通过 `ReflectionGeneric.java` 类来演示如何实现不同类型对象的添加，具体内容如代码 3.34 所示。

代码 3.34 实现不同类型添加的类：`ReflectionGeneric.java`

```
public class ReflectionGeneric {
    public static void main(String[] args) throws IllegalArgumentException,
        SecurityException, IllegalAccessException, InvocationTargetException,
```

```

NoSuchMethodException {
    ArrayList<Integer> arry1 = new ArrayList<Integer>(); //创建集合 arry1
    //添加两个整数型数字
    arry1.add(27);
    arry1.add(29);
    //通过反射向集合中添加字符型 abc
    arry1.getClass().getMethod("add", Object.class).invoke(arry1, "abc");
    //输出集合中的各个元素
    System.out.println("第一个元素为: "+arry1.get(0));
    System.out.println("第二个元素为: "+arry1.get(1));
    System.out.println("第三个元素为: "+arry1.get(2));
}
}

```

运行 ReflectionGeneric.java 类，控制台窗口如图 3.27 所示。

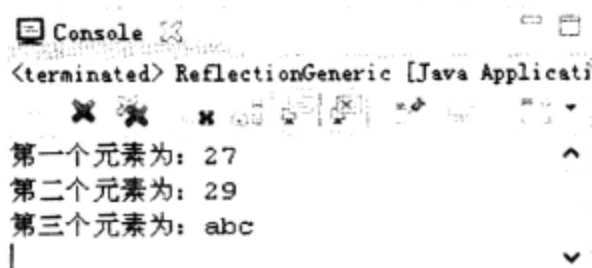


图 3.27 运行结果

【代码解析】

在代码 “arry1.getClass().getMethod("add", Object.class).invoke(arry1, "abc");” 中，首先通过 getClass().getMethod("add", Object.class) 方法获得字节码中的 add()，该方法的参数为一个 Object 类型。接着通过 invoke() 方法实现调用对象 arry1 的 add("abc") 方法，即向对象 arry1 中添加一个字符串对象。代码 “arry1.get(2)” 实现获取对象 arry1 中的第三个元素。

通过 ReflectionGeneric.java 类的运行结果可以发现，虽然泛型限制了对象 arry1 的类型只能是 Integer 类型，但是却可以通过反射绕过编译器向对象 arry1 中添加一个字符串对象 abc。

3.5.3 泛型的通配符

查看 Java 语言 API 的帮助文档，经常会遇到类的后面跟着 “<?>”、“<? extends U>” 和 “<? super U>” 等标识，例如 <U> Class <? Extends U> 等。如果想彻底了解这些通配符，则需要从为什么要出现通配符开始。

从上面的几节内容可以发现，泛型可以实现类型的参数化，但是在具体定义时却使得泛型类型固定化。那么如何实现参数的类型可以任意化？例如，如何定义一个能接受任意参数化类型的集合方法？具体步骤如下。

(1) 有的程序员通过设置方法接受的类型参数的实例为 Object 来实现，具体内容如代码 3.35 所示。

代码 3.35 类型为 Object 的类：RandomGenerObject.java

```

public class RandomGenerObject {
    public static void main(String[] args) {

```

```

Vector<Integer> v = new Vector<Integer>();
//创建 Integer 类型泛型对象
Vector<Object> v1 = new Vector<Object>();
//创建 Object 类型泛型对象

//调用静态方法 radomMeth()
//radomMeth(v);
radomMeth(v1);
}
public static void radomMeth(Vector<Object> vector) {
//接受任意类型泛型方法

//添加各种对象到集合对象 vector 中
vector.add("cjg");
vector.add(156);
for (Object obj : vector) {
//实现遍历功能
    System.out.println(obj);
}
}
}

```

运行 RandomGenerObject.java 类，控制台窗口如图 3.28 所示。



图 3.28 运行结果

【代码解析】

- ❑ 在 radomMeth()方法中，对于 Vector<Object>泛型，实现了可以向该参数中添加任意类型的对象，例如“vector.add("cjg");vector.add(156);”代码。这是因为任何对象的基类都是 Object 类型，所以任何类型的对象都可以自动转换成 Object 类型。
- ❑ 在具体调用 radomMeth()方法时，只要传入的参数为 Object 类型泛型对象，才会正确运行。如果是其他类型泛型对象，则会编译出错。这是因为参数化类型无继承性特性。

(2) 从 RandomGenerObject 类的最后实现功能可以发现，参数类型为 Object 类型泛型的方法 radomMeth()并能接受任何类型的参数。为了实现该功能，于是在泛型中出现了“?”标识。那么 RandomGenerObject 类的内容就可以修改成 RandomGener 类，该类的具体内容如代码 3.36 所示。

代码 3.36 具有通配符的类：RandomGener.java

```

public class RandomGener {
    public static void main(String[] args) {
        Vector<Integer> v = new Vector<Integer>(); //创建 Integer 类型泛型对象
        //添加成员
        v.add(1);
        v.add(2);
        Vector<Object> v1 = new Vector<Object>(); //创建 Object 类型泛型对象
    }
}

```



```

        //添加成员
        v1.add("aa");
        v1.add(2.2);
        radomMeth(v);           //调用 radomMeth() 方法
        radomMeth(v1);          //调用 radomMeth() 方法
    }
    public static void radomMeth(Vector<?> vector) {
        //接受任何类型参数的方法
        System.out.println("输出" + vector + "各个成员-----");
        //输出相应信息
        for (Object obj : vector) {
            //循环
            System.out.println(obj);
        }
        System.out.println("对象的大小" + vector.size()); //输出相应信息
    }
}

```

运行 RandomGenerObject.java 类，控制台窗口如图 3.29 所示。

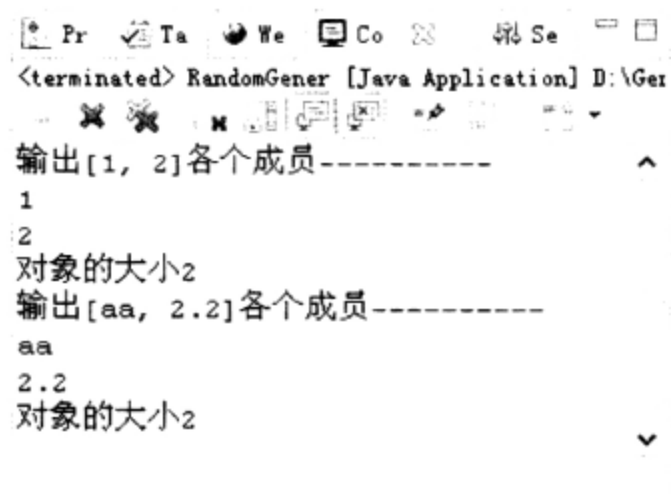


图 3.29 运行结果

【代码解析】

- ❑ 在 radomMeth()方法中，通过“?”标识实现接受任何类型参数的方法，即在具体调用该方法时，传入的对象可以是任意类型，例如 Integer 类型的对象 v 和 Object 类型的对象 v1。
- ❑ 在 radomMeth()方法中，虽然可以接受任意类型的参数，但是具体接受什么类型只有在具体调用该方法时才能确定。因此在该方法中添加确定类型的成员（vector.add("1")）时，则会编译报错。可是如果调用与参数类型无关的方法（vector.size()），则不会编译报错。

（3）类后面的<?>标识可以表示任意类型泛型，那么<? extends U>和<? super U>标识表示什么呢？下面通过 IntegrationGener.java 类来具体演示，该类的具体内容如代码 3.37 所示。

代码 3.37 具有通配符的类：IntegrationGener.java

```

public class IntegrationGener {
    public static void main(String[] args) {
        Number num1 = new Integer(1);           //创建对象 num1
        Number num2 = new Double(1.23);          //创建对象 num2
    }
}

```


```

//创建对象 listNums, 其存储对象是 Number
List<Number> listNums = new ArrayList<Number>();
//创建 ArrayList 类型对象
listNums.add(1); //添加 1 到对象 listNums
listNums.add(1.23); //添加 1.23 到对象 listNums
//创建 ArrayList 类型对象, 其存储对象是 Integer 类型
List<Integer> listInteger = new ArrayList<Integer>();
//创建 ArrayList 类型对象, 其存储对象是 Number 类型
List<Number> listNums1 = new ArrayList<Number>();
List<? extends Number> listNums2 = listInteger; //创建对象 listNums2
List<? super Integer> listNums3 = listInteger; //创建对象 listNums3
listNums3.add(7); //添加对象 7 到对象 listNums3
listNums3.add(null); //添加对象 null 到对象 listNums3
//输出相应信息
System.out.print("listNums2 中的元素" + listNums2.get(0));
List<? super Integer> listNums4 = listNums1; //创建对象 listNums4
listNums4.add(6); //添加对象 6 到对象 listNums4
listNums3.add(null); //添加对象 null 到对象 listNums3
}
}

```

【代码解析】

- ❑ 对象 num1 和 num2 的定义代码, 子类 (Integer、Double) 可以自动转换成其父类 (Number)。同理, ArrayList<Number>泛型也可以自动转换成 List<Number>泛型, 因为 List 类为 ArrayList 类的父类。
- ❑ List<Integer>类型的对象 listInteger, 之所以不能赋值给 List<Number>类型的对象 listNums1, 是因为泛型的参数类型无继承性。因此虽然 Integer 是 Number 的子类, 但是充当泛型的参数时, 则不能实现转换。
- ❑ 为了解决泛型的参数类型无继承性, 出现了 extends 和 super 标识符。List<? extends Number> listNums2 表示 listNums 对象可以被 Number 类型的任何子类对象 (List<Integer> listInteger) 赋值。List<? super Integer> listNums3 表示 listNums3 对象可以被 Integer 类型的任何父类对象 (listNums1) 或 Integer 类型对象 (listInteger) 赋值。

 **注意:** 一条比较通用的规则是, 如果要向列表中添加元素则用 <? super T>, 如果要从列表中获取元素则用 <? extends T>, 如果既要获取又要添加则不使用通配符。

3.6 类加载器

由于 Java 语言是一种解释型编程语言, 所以当程序运行时 Java 虚拟机就将编译生成的.class 文件按照需求和一定的规则加载内存, 并组织成为一个完整的 Java 应用程序。该过程就是由类加载器自动完成, 类加载是 Java 语言提供的最强大的机制之一。尽管类加载并不是 Java 语言的重点知识点, 但是理解其工作机制可以让程序员节省编码时间。

3.6.1 什么是类加载器

所谓类加载器就是加载类的工具，Java 虚拟机（JVM）运行类的第一件事情就是将该类的字节码加载进来，即类加载器根据类的名称定位和生成类的字节码数据，然后返回给 JVM。

Java 源代码运行的具体过程如图 3.30 所示，首先 Java 编译器会把.java 后缀名的 Java 源文件编译成中间层的字节码文件（ByteCode），然后才会由 JVM 解释执行字节码文件。从该运行体系结构图中可以发现，类加载器只要能提供给 JVM 调用的类字节码就可以，因此类加载器也可以描述为字节码的制造器。

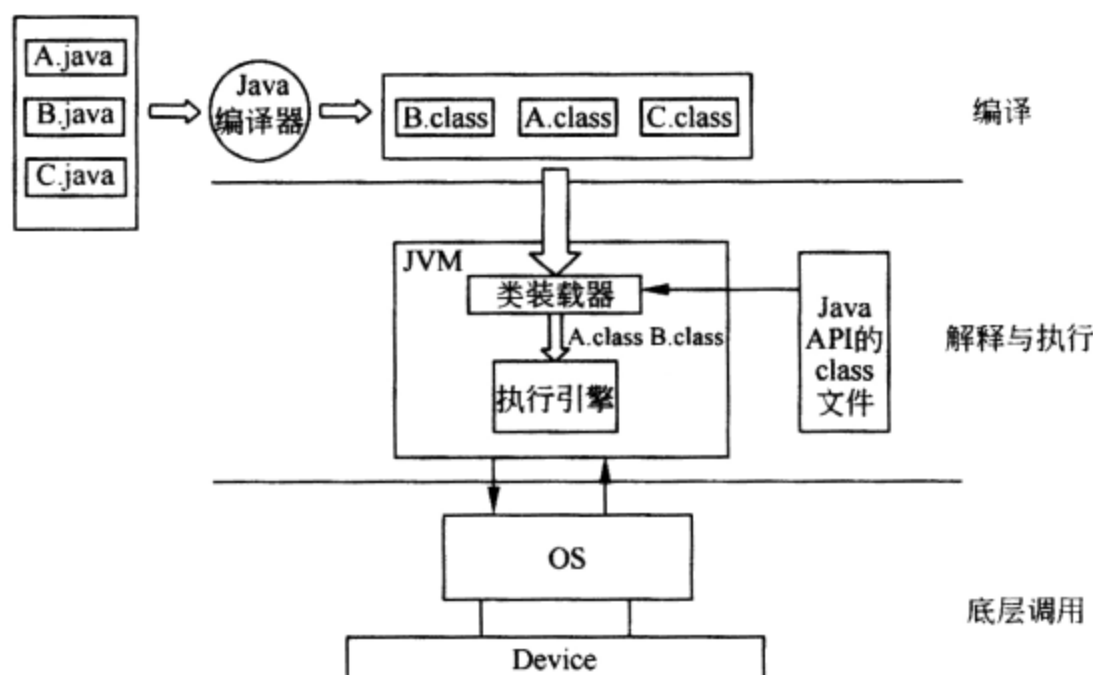


图 3.30 运行体系结构

当一个类被加载后，JVM 将其编译为可以执行的代码（字节码）存储到内存中，同时会将索引信息存储到一个 HashTable 中，注意索引的关键字就是被加载类的完整名字。如果 JVM 想运行某个类时，其首先会使用类名作为关键字在 HashTable 中查找相应的信息，如果该可执行代码已经存在，JVM 就直接会从内存里调用该可执行代码，否则就调用类加载器进行加载和编译。

类加载器其实也是 Java 类，所以任何 Java 类的类加载器本身也要被类加载器加载，那么第一个类加载器是什么呢？根据第 1 章的内容可知，如果想手工运行一个.class 文件，则必须运行 java.exe 命令。该命令首先会根据 %JAVA_HOME%\jre\lib\i386\jvm.cfg 配置来选择激活 JVM，启动及初始化工作完成之后便会产生 Bootstrap Loader 加载器。

注意：由于 BootstrapLoader 加载器不需要加载，所以其不是 Java 类而是利用 C++ 语言编写。

在 JVM 中有两个内置类加载器 ExtClassLoader 和 AppClassLoader，它们定义在 sun.misc.Launcher.class 中为内部类，是由 Bootstrap Loader 加载进入 JVM 的。

最后，由于 Class 类用于描述 Java 程序语言中一个类的有关信息，即会封装具体类的字节码数据，所以可以这样理解类加载器装载某个类字节码的过程实际上就是创建 Class

类的一个实例对象。

3.6.2 什么是类加载器的委派模型

通过 3.6.1 节的知识可以知道，在 JVM 中存在多个类加载器，那么当 JVM 加载一个具体的类时，通过什么方式来选择类加载器呢？这些类加载器之间有什么关系呢？

(1) 第一个类加载器 (BootstrapLoader)——引导类加载器，之所以会出现该类加载器，主要是因为其他类加载器也需要类加载器来加载，那就出现了类似于人类第一位母亲是如何产生出来的问题，于是在 JVM 中嵌入了一个利用 C++ 语言编写的类加载器。引导类加载器由操作系统的本地代码来实现，不需要专门的类加载器去进行加载，主要负责加载 Java 核心包中的类 (%JAVA_HOME%\jre\lib 目录下的 jar 文件)，例如 jce.jar 和 rt.jar 等。

(2) 在 JVM 中还有另外两个类加载器：ExtClassLoader (扩展类加载器) 和 AppClassLoader (应用程序类加载器)，这两个类加载器都是利用 Java 语言编写的 Java 类。顾名思义，扩展类加载器主要负责扩展路径下的代码，即位于 %JAVA_HOME%\jre\lib\ext 目录下的 jar 文件或通过 java.ext.dirs 这个系统属性指定路径下的代码；应用程序类加载器主要负责加载应用程序，即 CLASSPATH 这个系统属性指定路径下的代码。

(3) 由于 JVM 中所有的类加载器利用树形结构进行组织，如图 3.31 所示，所以在实例化一个类加载器对象时，需要为其指定一个父类加载器对象。虽然每个类加载器只能加载特定位置和目录中的类，但是却可以通过委托模式委托它的父类加载器去加载类。如果应用程序类加载器需要加载一个类，它首先委托扩展类加载器，扩展类加载器再委托引导类加载器。如果父类加载器不能加载类，子类加载器就会在自己的库中查找这个类。基于这个特性，类加载器只负责它的祖先无法加载的类。

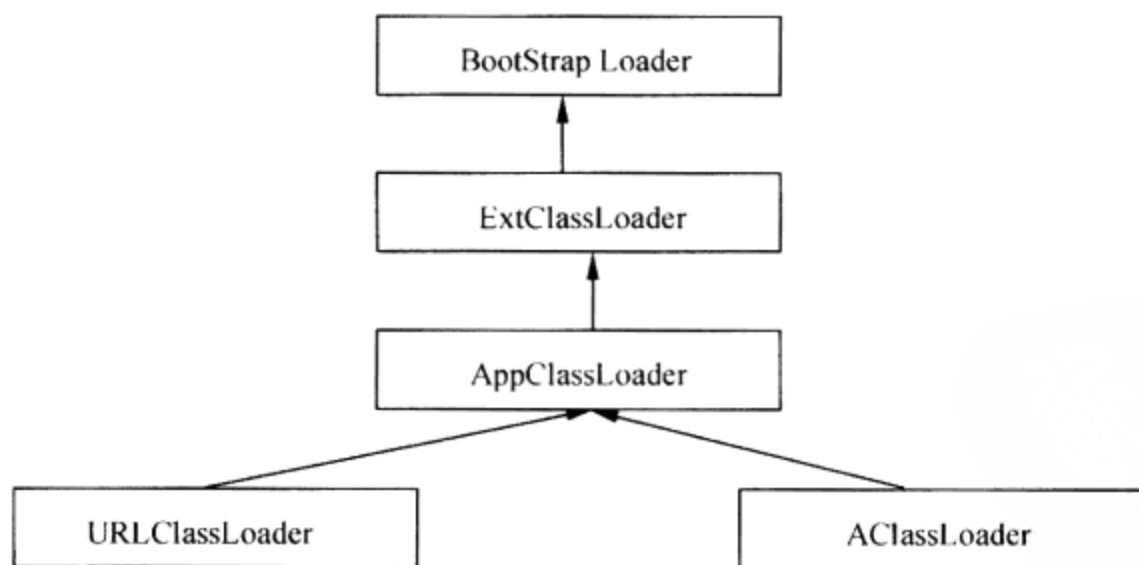


图 3.31 内置类加载器关系

为了能让读者彻底理解类加载器的委托模式，下面通过 InternalLoad.java 类来具体演示，该类的具体内容如代码 3.38 所示。

代码 3.38 类加载器：InternalLoad.java

```
public class InternalLoad {
    public static void main(String[] args) {
        System.out.println("-----");
    }
}
```

```

//InternalLoad 类加载器的名字
System.out.println("InternalLoad 类加载器的名字:"
    + InternalLoad.class.getClassLoader().getClass().getName());
System.out.println("-----");
//System 类加载器的名字
System.out.println("System 类加载器的名字:" + System.class.getClass-
Loader());
//获取 InternalLoad 类加载器
ClassLoader load = InternalLoad.class.getClassLoader();
System.out.println("-----");
//遍历 InternalLoad 类加载器
while (load != null) {
    System.out.println(load.getClass().getName());
    load = load.getParent();
}
System.out.println(load);           //输出相应信息
}
}

```

运行 InternalLoad.java 类，控制台窗口如图 3.32 所示。



```

-----
InternalLoad类加载器的名字:sun.misc.Launcher$AppClassLoader
-----
System类加载器的名字:null
-----
sun.misc.Launcher$AppClassLoader
sun.misc.Launcher$ExtClassLoader
null

```

图 3.32 运行结果

【代码解析】

- ❑ 如果想获取某个类的字节码，可以通过“类名.class”或“对象.getClass()”形式来实现。对于类的字节码对象，如果想获取加载其的类加载器对象，可以通过方法 getClassLoader() 来实现。即代码 InternalLoad.class.getClassLoader().getClass().getName() 能够实现获取加载 InternalLoad 字节码的名字。
- ❑ 对于代码 System.class.getClassLoader()，如果改写成 InternalLoad.class.getClassLoader().getClass().getName()，则会出现 NullPointerException 错误。这是因为类 System 的加载器为 Null 不是 Java 类对象，当某个类的加载器为 Null 时，不代表该类没有加载器而是为默认类加载器——BootstrapLoader。
- ❑ 如果想获取某个类加载器的父类加载器，可以通过 getParent() 方法来实现。通过遍历 InternalLoad 类的加载器可以发现，AppClassLoader 父类加载器为 ExtClassLoader，而 ExtClassLoader 加载器的父类加载器为 BootstrapLoader。对于运行结果中的 sun.misc.Launcher 则为类加载器类的包。
- ❑ 由于类 System 在 %JAVA_HOME%\jre\lib\rt.jar 文件中，所以其的类加载器为 BootstrapLoader，而 InternalLoad 类存储在 CLASSPATH 系统属性指定的目录里，所以其类加载器为 AppClassLoader 来加载。

如果想测试 ExtClassLoader 类加载器的加载目录，首先需要通过开发工具把 InternalLoad.java 类导入到 %JAVA_HOME%\jre\lib\ext 目录下，具体步骤如下。

(1) 确定运行 InternalLoad.java 类的运行环境，在 MyEclipse 开发环境中右击 InternalLoad.java 类，在弹出的快捷菜单中选择 Properties 菜单项，就可以打开 Properties 对话框，如图 3.33 所示。在该对话框的 Run/Debug Settings 列表框中选择 InternalLoad 选项，然后单击 Edit 按钮就可以打开 Edit Configuration 对话框，如图 3.34 所示。在其中选择 JRE 标签，然后在 Runtime JRE 选项区域中选择运行环境，例如 JavaSE-1.6 (jdk.6.0_03)，即 jdk.6.0_03 运行环境中的 JavaSE-1.6 版本。

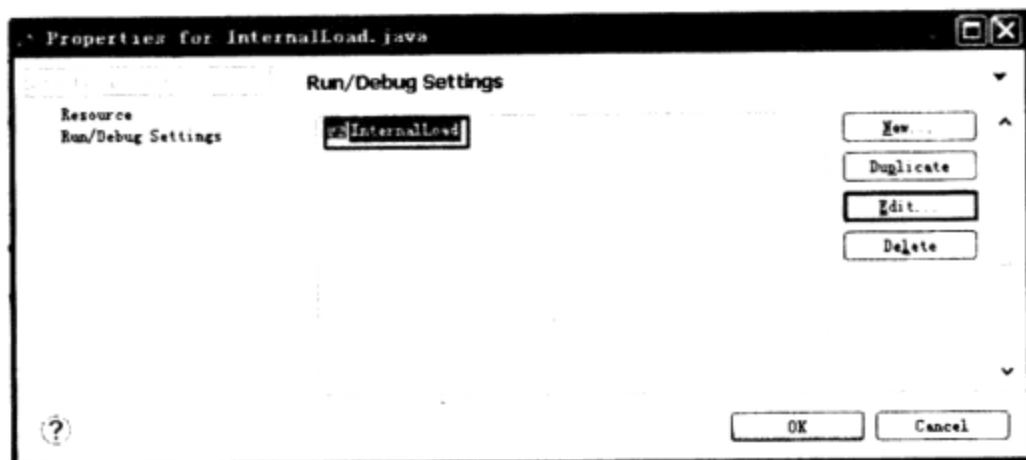


图 3.33 属性对话框

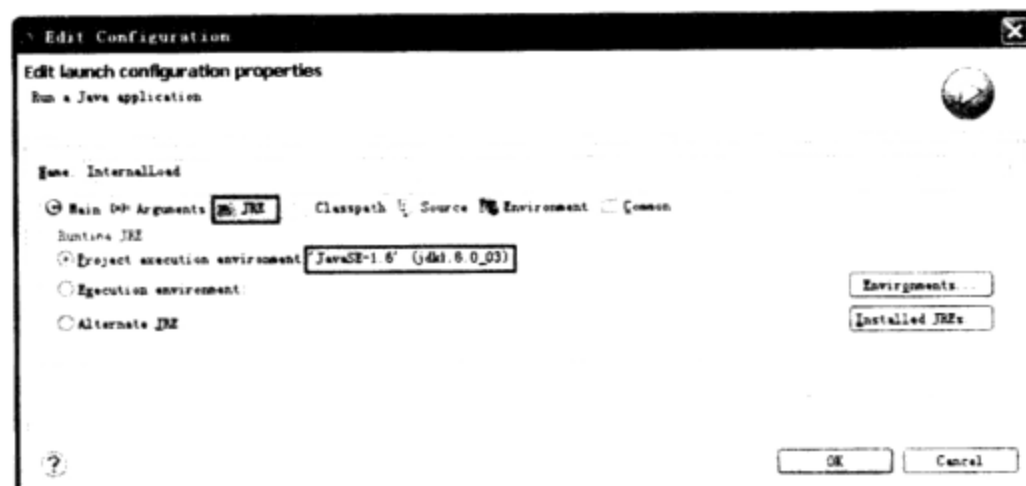


图 3.34 运行环境

(2) 确定运行环境的目录，在 MyEclipse 开发环境中通过选择菜单 Window | Preferences 命令打开 Preferences 对话框，如图 3.35 所示。在该对话框中选择 Java>Installed JREs 就可以显示出 Installed JREs 的信息，由于运行环境是 jdk.6.0_03，所以其路径为 D:\Java\jdk1.6.0_03。

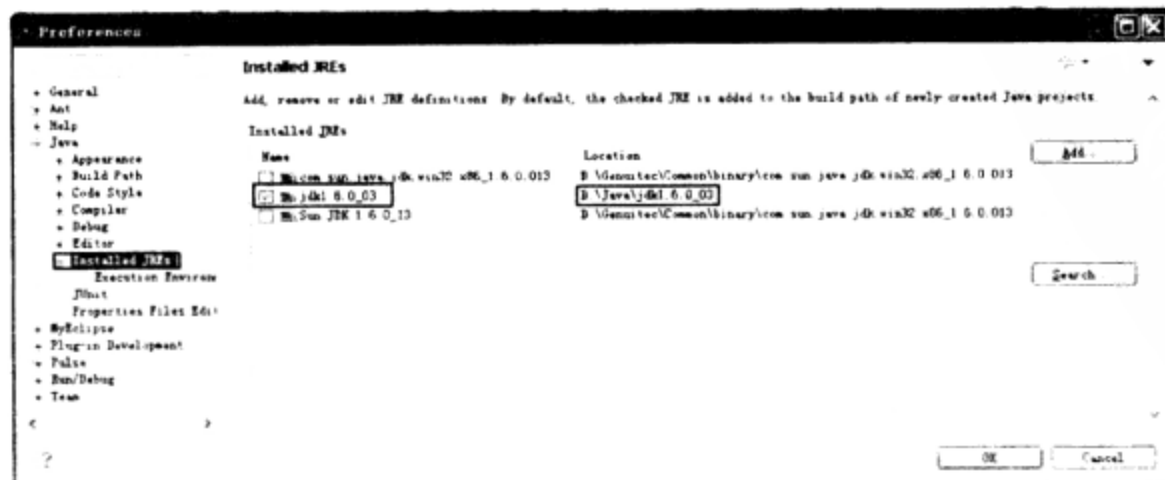


图 3.35 Installed JREs 信息

(3) 实现导入, 在 MyEclipse 开发环境中右击 InternalLoad.java 类, 在弹出的快捷菜单中选择 Export 选项, 就可以打开 Export 对话框, 如图 3.36 所示。在该对话框中选择 Java>JAR file 选项, 单击 Next 按钮就可以打开 JAR Export 对话框。在该对话框中通过单击 Browse 按钮来选择相应的导出目录, 从上述的两个步骤可以知道具体目录为 D:\Java\jdk1.6.0_03\jre\lib\ext, 最后单击 Finish 按钮就可以实现相关 JAR 文件的导出, 如图 3.37 所示。

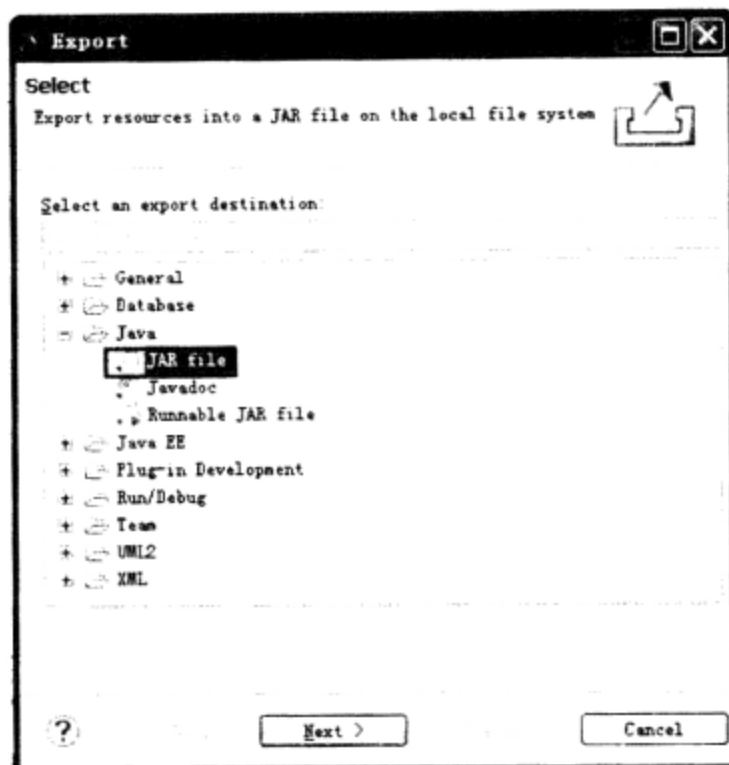


图 3.36 Export 对话框

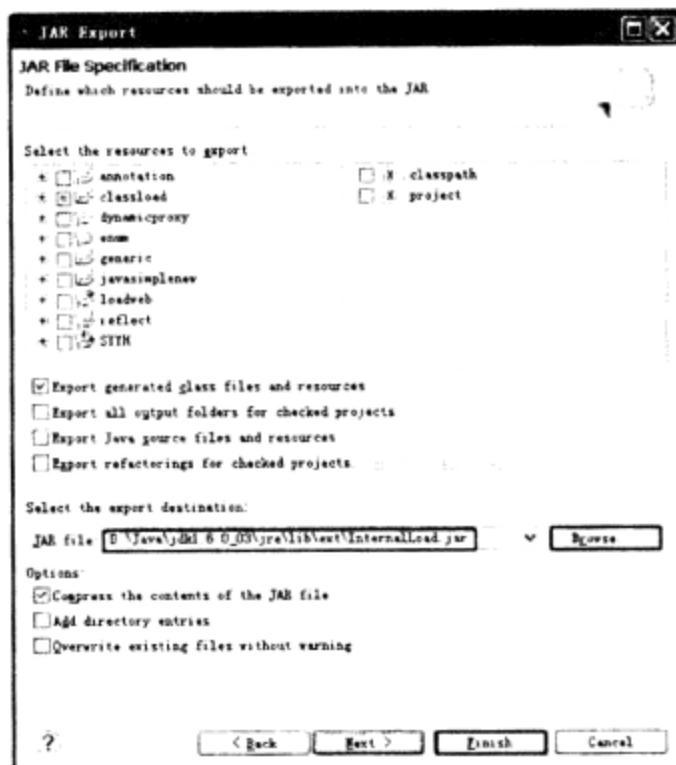


图 3.37 实现导出

(4) 当再次运行 InternalLoad.java 类时, 控制台窗口如图 3.38 所示。从图 3.38 中可以发现, InternalLoad.java 类加载器的名字为 ExtClassLoader 而不是 AppClassLoader。



图 3.38 运行结果

此时的运行环境里 CLASSPATH 系统环境下有 InternalLoad.jar, ext 目录下也有 InternalLoad.jar, 可是运行结果却为 ExtClassLoader。这主要是因为每个类加载器加载类时, 又会委托给其上级类加载器, 当所有的祖类加载器没有加载到类时, 就会回到发起者的类加载器, 这时如果还加载不了, 则会抛出 ClassNotFoundException, 而不是再去委托给发起者类加载器的子类。

3.6.3 编写一个自己的加载器

JVM 需要一个类的时候, 就把一个类名传给类加载器, 然后类加载器试图返回一个对


应的类实例。可以通过在不同的阶段覆盖相应的方法来创建自定义的类加载器。接下来将介绍类加载器的一些主要方法。

1. loadClass()方法

loadClass()方法加载指定名称（包括包名）的二进制类型，定义如下：

```
public Class<?> loadClass(String name) throws ClassNotFoundException{//...}
protected synchronized Class<?> loadClass(String name, boolean resolve)
throws ClassNotFoundException{//...}
```

在上述定义中，参数 name 指定 Java 虚拟机需要的类的全名（含包名），参数 resolve 指定该类是否需要解析。

 注意：所谓类的解析，就是完全为运行做好准备，一般没有必有。

2. findClass()方法

findClass()方法加载指定名称类，定义如下：

```
protected Class<?> findClass(String name) throws ClassNotFoundException
{//...}
```

该方法一般被 loadClass()方法调用去加载指定名称类，同时还实现加载委派规则。

3. defineClass()方法

defineClass()方法生成类实例对象，定义如下：

```
protected final Class<?> defineClass(String name, byte[] b, int off, int
len) throws ClassFormatError{//...}
```

该方法通过一个字节数组来构建类实例，即主要通过解释字节码把它转化为运行时数据结构，这个包含数据的原始字节数组可能来自文件系统，也可能来自网络。

如果想设计一个自定义类加载器，必须要清楚自定义类加载器的工作流程，具体步骤如下。

(1) 需要先检查请求的类是否已经被这个类装载机装载到命名空间中了，如果已经装载，直接返回；否则转入下一步骤。

(2) 委派类加载请求给父类加载器（更准确地说应该是双亲类加载器，每个虚拟机中各种类加载器最终会呈现树状结构），如果父类加载器能够完成，则返回父类加载器加载的 Class 实例；否则转入下一步骤。

(3) 调用本类加载器的 findClass(...)方法，试图获取对应的字节码，如果可以获取到，则调用 defineClass(...)导入类型到方法区；如果获取不到对应的字节码或者因其他原因而失败，返回异常给 loadClass(...)，loadClass(...)会抛出异常，终止加载过程。

下面将通过具体实例来讲解如何编写类加载器，为了便于讲解，本例专门设计了一个实现加密 class 文件的类，然后通过实现解密 class 文件的自定义类加载器来加载加密后的 class 文件，最后测试自定义类加载器是否成功。具体步骤如下。

(1) 创建一个名为 CypherFile.java 的类，该类用来实现对文件进行加密，具体内容如

代码 3.39 所示。

代码 3.39 对文件进行加密: CypherFile.java

```
public class CypherFile {
    public static void main(String[] args) throws Exception {
        //定义源文件和目的文件的目录
        String srcPath=args[0];
        String destDir=args[1];
        FileInputStream fis= new FileInputStream(srcPath);
                                                //读取源文件的文件流

        //创建目的文件的目录
        String destFileName=srcPath.substring(srcPath.lastIndexOf(
            '\\')+1);
        String destPath=destDir+"\\ "+destFileName;
        FileOutputStream fos=new FileOutputStream (destPath);
                                                //写入相应目的文件
        cypher(fis, fos);
                                                //实现加密功能
        //关闭输入流和读取流
        fis.close();
        fos.close();
    }
    private static void cypher(InputStream ips, OutputStream ops)
                                                //实现加密功能

        throws Exception {
        int b = -1;
                                                //定义一个变量
        //遍历字节实现加密和解密
        while ((b = ips.read()) != -1) {
            ops.write(b ^ 0xff);
        }
    }
}
```

【代码解析】

- ❑ 在上述代码中实现了一个简单的加密 cypher()方法,让相关文件的二进制字节数据与 0xff 进行与或运算。之所以使用该算法,就是因为不需要再设计一个解密方法,即方法 cypher()不仅可以实现加密功能,也能实现解密功能。
 - ❑ 在上述代码的主方法 main()中,实现对相应的类进行加密和解密的功能。
- (2) 接着创建需要加密的类 CypherClass.java,具体内容如代码 3.40 所示。

代码 3.40 需要加密的类: CypherClass.java

```
public class CypherClass extends Date {
    public String toString() {
        return "abc";
    }
}
```

【代码解析】

在上述代码中,之所以要继承 Date 类,是为了便于后面测试类的编写。

(3)为了测试 CypherFile.java 类,需要创建一个名为 classpath1 的文件夹。右击 classload

项目，在弹出的快捷菜单中选择 New | Folder 命令，如图 3.39 所示，弹出 New Folder 对话框，该对话框的设置如图 3.40 所示，这时就会在该项目中出现名为 classpath1 的文件夹。

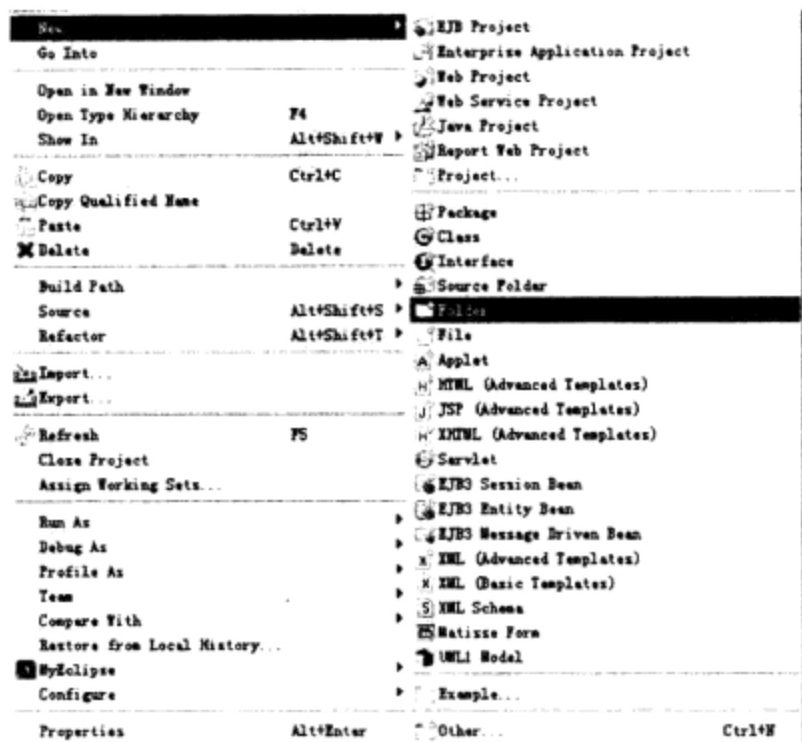


图 3.39 菜单选项

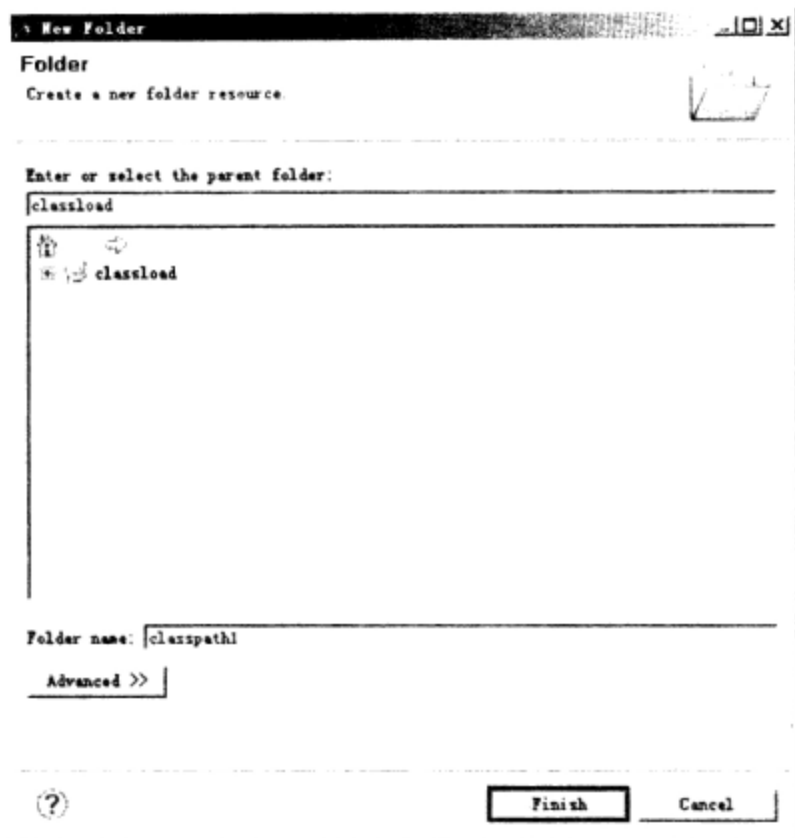


图 3.40 New Folder 对话框

(4) 当测试 CypherFile.java 类时，需要右击该类文件，在弹出的快捷菜单中选择 Run As | Run Configurations 命令，如图 3.41 所示，打开 Run Configurations 对话框。对该对话框的 Main 选项卡中进行如图 3.42 所示的设置；在 Arguments 选项卡中进行如图 3.43 所示的设置。单击 Run 按钮就可以在目录 classpath1 中出现加密后的文件，如图 3.44 所示。

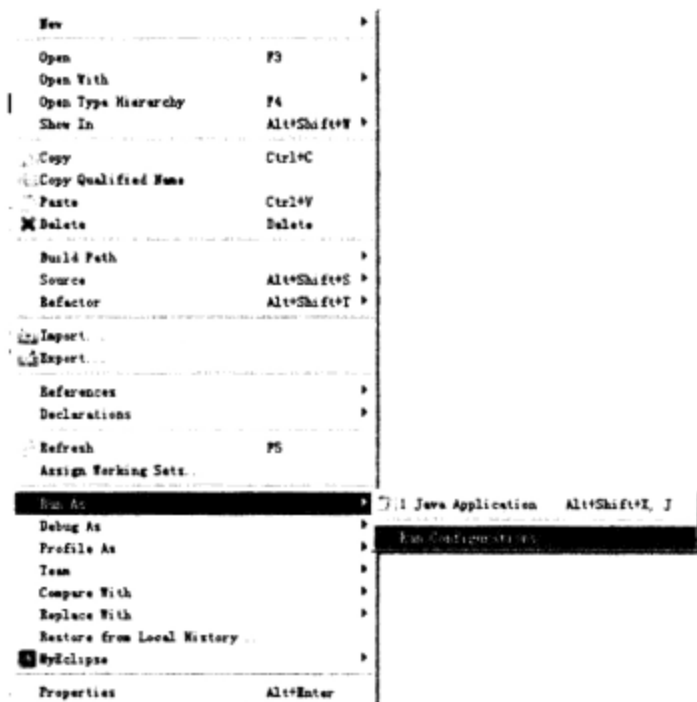


图 3.41 菜单选项

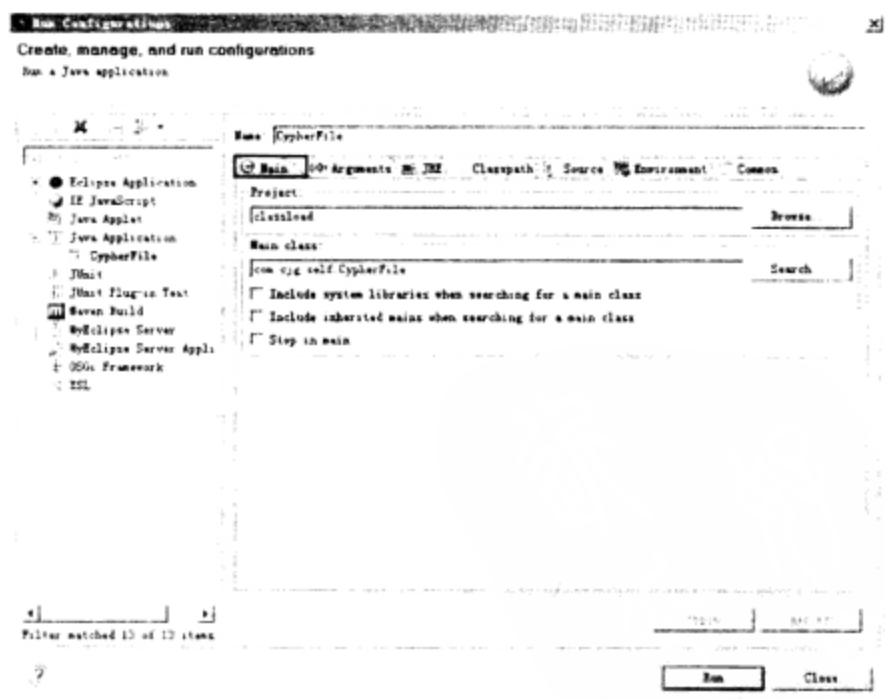


图 3.42 Main 选项卡

(5) 当通过 classpath1 文件夹中的 CypherClass.class 文件创建对象时会出错，这是因为该类文件被加密了。为了解决该问题，可以创建一个名为 SelfClassLoader.java 的类，该类用来实现类加载的功能，具体内容如代码 3.41 所示。

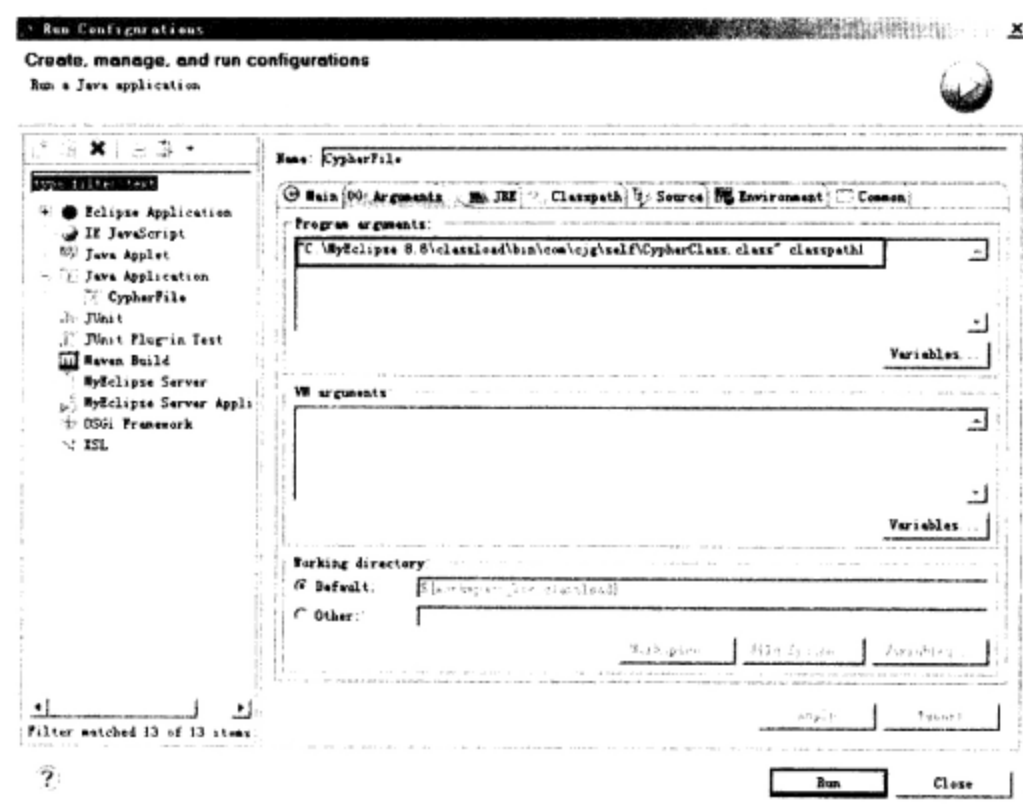


图 3.43 Arguments 选项卡



图 3.44 运行结果

代码 3.41 自定义加载器: SelfClassLoader.java

```

public class SelfClassLoader extends ClassLoader {
    private String classDir;                //定义字节码目录的变量
    public SelfClassLoader() {              //无参构造函数
    }
    public SelfClassLoader(String classDir) { //有参构造函数
        this.classDir = classDir;
    }
    //覆盖 findClass() 方法
    protected Class<?> findClass(String name) throws ClassNotFoundException
    Exception {
        //定义了字节码的名字
        String classFileName = classDir + "\\ " + name + ".class";
        try {
            //读取类文件的文件流
            FileInputStream fis = new FileInputStream("classFileName");
            //字节数组输出流
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            cypher(fis, bos);                //进行解密
            fis.close();                     //关闭输入流
        }
    }
}

```

```

        byte[] bytes = bos.toByteArray(); //获取字节数组
        return defineClass(bytes, 0, bytes.length); //返回加载后的类文件
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
private static void cypher(InputStream ips, OutputStream ops)
    //解密方法
    throws Exception {
    int b = -1; //定义一个变量 b
    while ((b = ips.read()) != -1) { //通过遍历实现加密
        ops.write(b ^ 0xff);
    }
}
}

```

【代码解析】

- ❑ 如果要自定义类加载器，首先需要继承 `ClassLoader` 类，然后覆盖该类的 `findClass()` 方法。
 - ❑ 在 `findClass()` 方法中首先读取已经加密后的 class 文件，然后通过解密 `cypher()` 方法对读取到的 class 文件进行解密，最后通过 `defineClass()` 方法将解密后的字节数组生成 class 文件。
- (6) 最后创建一个测试类加载器的类，具体内容如代码 3.42 所示。

代码 3.42 测试类加载器的类：TestSelfClassLoad.java

```

public static void main(String[] args) throws ClassNotFoundException,
    InstantiationException, IllegalAccessException {
    //获取 CypherClass 类的字节码
    Class d1 = new SelfClassLoad("classpath1").loadClass ("Cypher-
    Class");
    Date d2 = (Date) d1.newInstance(); //创建对象 d2
    System.out.println(d2.toString()); //输出对象 d2 信息
}
}

```

运行 TestSelfClassLoad.java 类，控制台窗口如图 3.45 所示。

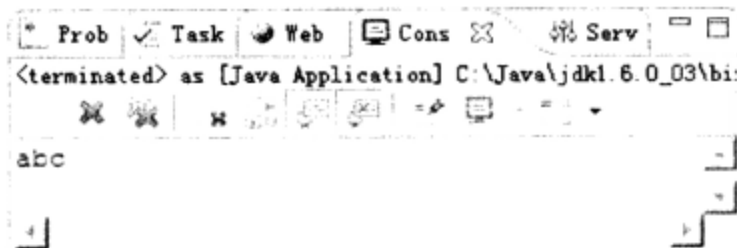


图 3.45 运行结果

【代码解析】

在上述代码中，首先通过自定义加载类 `SelfClassLoad`，加载 `classpath1` 目录下名为 `CypherClass` 的 class 文件，然后通过 `Class` 类的 `newInstance()` 方法根据返回的 class 文件字节码 (`d1`) 生成该 class 文件的对象，最后再输出该对象 `d2` 的相应信息。

3.7 动态代理

由于 Java 语言是一种解释型编程语言，所以当程序运行时 Java 虚拟机就将编译生成的.class 文件按照需求和一定的规则加载内存，并组织成为一个完整的 Java 应用程序，该过程就是由类加载器自动完成。虽然类加载是 Java 语言提供的最强大的机制之一，尽管类加载并不是 Java 语言的重点知识点，但是理解其工作机制可以让程序员节省编码时间。

3.7.1 什么是代理

代理这个术语对于任何人来说都不陌生，因为在现实生活中经常会与其打交道。本节就以买书为例讲解什么是代理。

假设你需要买一本书，一种方法是亲自去各大书店查找是否有自己需要的书，然后才能买。如果你非常忙，没有时间处理这些事情，那么可以去找中介（当当网），让中介帮你处理这些事情，中介实际上就是你的代理。本来是你要做的事情，现在中介帮你一一处理。对于你来说跟书店直接交易与跟同中介直接交易没有任何差异，你甚至可能觉察不到书店的存在，这实际上就是代理的一个最大的好处。

为什么不直接到书店找书而需要中介？其实一个问题恰恰解答了什么时候该用代理模式的问题。

原因一：可能在上班，没时间到各大书店。

对应到应用程序：客户端无法直接操作实际对象。那么为什么无法直接操作呢？一种情况是需要调用的对象在另外一台计算机上，需要跨越网络才能访问。如果想直接去调用对象，需要处理网络连接、处理打包、解包等非常复杂的步骤，所以为了简化客户端的处理出现了代理。即在客户端建立一个远程对象的代理，客户端就会像调用本地对象一样调用该代理，再由代理去跟实际对象联系，对于客户端来说可能根本没有感觉到调用的东西在网络另外一端，这实际上就是 Web Service 的工作原理。另一种情况是虽然需要调用的对象就在本地，但是由于调用非常耗时，怕影响正常的操作，所以特意找个代理来处理这种耗时情况。

原因二：可能不知道到书店的路线，或者说除了会干的事情外，还需要做其他的事情才能达成目的。

对应到应用程序：除了当前类能够提供的功能外，还需要补充一些其他功能。最容易想到的情况就是权限过滤，即有一个类做某项业务，但是由于安全原因只有某些用户才可以调用这个类，此时就可以做一个该类的代理类，要求所有请求必须通过该代理类，由该代理类做权限判断，如果安全则调用实际类的业务开始处理。可能有人说为什么要多加代理类？只需要在原来类的方法里加上权限过滤不就可以了么？这主要是因为程序设计中类具有单一性原则，即每个类的功能尽可能单一。如果把权限判断放在当前类里，当前这个类就既要负责自己本身业务逻辑，又要负责权限判断。如果权限规则或业务逻辑有一个需要变化，这个类就必须改，显然这不是一个好的设计。

综上所述可以发现，代理涉及的各种对象关系如图 3.46 所示。

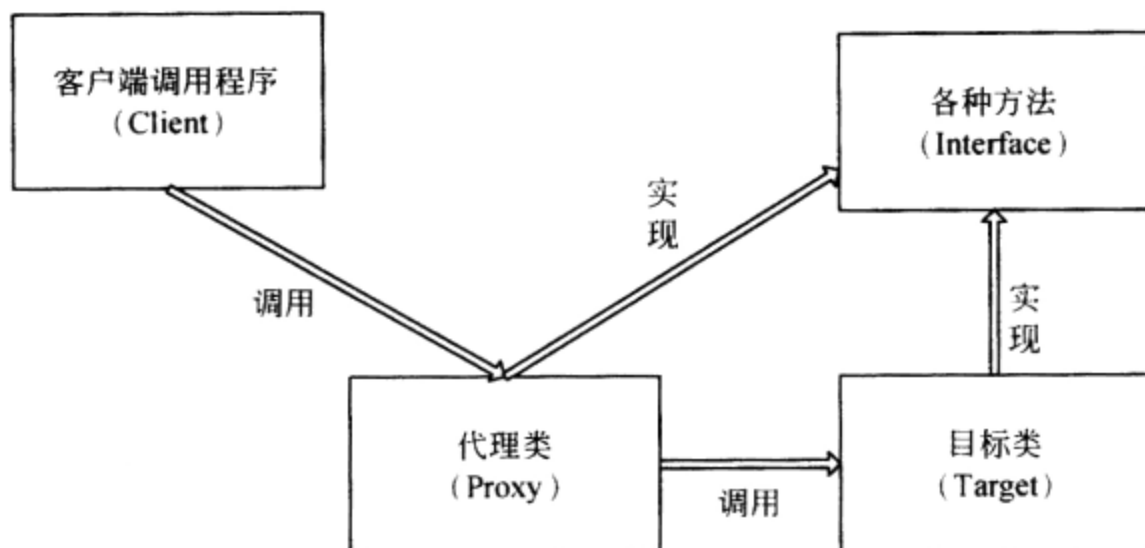


图 3.46 代理中对象的关系

3.7.2 动态代理基础类

在 3.7.1 节中解了代理的基本知识后，例如什么是代理、代理的基本概念等，本节将接着 3.7.1 节的内容，讲解如何创建动态代理。Sun 公司为了方便程序员实现动态代理，设计了许多动态代理的类。

查看 API 帮助文档，可以发现 `java.lang.reflect` 包中的 `Proxy` 类和 `InvocationHandler` 接口提供了生成动态代理类的能力。`Proxy` 类提供了创建动态代理类及其实例的静态方法，介绍如下。

1. `getProxyClass()`方法

`getProxyClass()`方法用于创建动态代理类的字节码。它的完整定义如下：

```
public static Class<?> getProxyClass(ClassLoader loader, Class<?>[]  
interface) throws IllegalArgumentException
```

参数 `loader` 指定了动态代理类的类加载器，参数 `interface` 指定动态代理类所要实现的接口。

2. `newProxyInstance()`方法

`newProxyInstance()`方法用于创建动态代理类的实例。它的完整定义如下：

```
public static newProxyInstance(ClassLoader loader, Class<?>[]  
interface, InvocationHandler handler) throws IllegalArgumentException
```

参数 `loader` 指定动态代理类的类加载器，参数 `interface` 指定动态代理类所要实现的接口，参数 `handler` 指定与动态代理类关联的 `InvocationHandler` 对象。

而 `InvocationHandler` 接口为 `Proxy` 类的拦截器接口，用来指示 `Proxy` 类拦截到方法调用时作何处理，该接口拥有一个重要的方法 `invoke()`，具体定义如下：

```
Object invoke(Object proxy, Method method, Object[] args) throws Throwable
```

参数 `proxy` 为代理实例，参数 `method` 为被拦截的方法，参数 `args` 为参数列表。

下面通过一个简单类 `ProxyFunction.java`，来讲解如何获取动态代理类的构造函数和其

他函数，具体内容如代码 3.43 所示。

代码 3.43 动态代理类的构造函数: ProxyFunction.java

```
public class ProxyFunction {
    public static void main(String[] args) throws SecurityException,
        NoSuchMethodException, IllegalArgumentException,
        InstantiationException, IllegalAccessException,
        InvocationTargetException {
        //获取代理类的字节码
        Class proxy1 = Proxy.getProxyClass(Collection.class.getClass-
            oader(),
            Collection.class);
        System.out.println(proxy1.getName());           //输出字节码的名字
        System.out.println("构造函数的列表-----");
        //获取构造函数集
        Constructor[] constructors = proxy1.getConstructors();
        for (Constructor constructor : constructors) { //遍历构造函数
            String name = constructor.getName();       //获取构造函数的名字
            //创建 name 变量的字符串
            StringBuilder sBuilder = new StringBuilder(name);
            sBuilder.append('(');                      //字符串后添加 "("
            //获取构造函数参数的类型
            Class[] params = constructor.getParameterTypes();
            for (Class param : params) {               //遍历参数的类型
                sBuilder.append(param.getName()).append(',');
            }
            if (params != null && params.length != 0) {
                sBuilder.deleteCharAt(sBuilder.length() - 1);
            }
            sBuilder.append(')');
            System.out.println(sBuilder.toString());   //输出字符串
        }
        System.out.println("方法的列表-----");
        Method[] methods = proxy1.getMethods();       //获取方法集
        for (Method method : methods) {               //遍历方法
            String name = method.getName();
            StringBuilder sBuilder = new StringBuilder(name);
            sBuilder.append('(');
            Class[] params = method.getParameterTypes();
            for (Class param : params) {               //遍历方法的类型
                sBuilder.append(param.getName()).append(',');
            }
            if (params != null && params.length != 0) {
                sBuilder.deleteCharAt(sBuilder.length() - 1);
            }
            sBuilder.append(')');
            System.out.println(sBuilder.toString());   //输出方法
        }
    }
}
```

运行 ProxyFunction.java 类，控制台窗口如图 3.47 所示。

【代码解析】

- ❑ 上述代码首先通过代理类 Proxy 的 getProxyClass()方法，获取接口 Collection 的动态类字节码。
- ❑ 获取 Collection 动态类字节码后，首先通过 getConstructors()方法获取构造函数集，

通过 `getParameterTypes()` 方法获取构造函数的参数类型，然后通过遍历输出所有的构造函数。同时，通过 `getMethods()` 方法获取函数集，通过 `getParameterTypes()` 方法获取函数的参数类型，然后通过遍历输出所有的函数。

```

Console
<terminated> ProxyTest [Java Application] C:\Genustec\Common\binary\com.sun.java.jdk.win32.x86_1.6.0.013\bin\java.exe 2010-3-2 下午
$Proxy0
构造函数的列表-----
$Proxy0(java.lang.reflect.InvocationHandler)
方法的列表-----
add(java.lang.Object)
hashCode()
clear()
equals(java.lang.Object)
toString()
contains(java.lang.Object)
isEmpty()
addAll(java.util.Collection)
iterator()
size()
toArray([Ljava.lang.Object;)
toArray()
remove(java.lang.Object)
containsAll(java.util.Collection)
removeAll(java.util.Collection)
retainAll(java.util.Collection)
isProxyClass(java.lang.Class)
getProxyClass(java.lang.ClassLoader,[Ljava.lang.Class;)
getInvocationHandler(java.lang.Object)
newProxyInstance(java.lang.ClassLoader,[Ljava.lang.Class;,[Ljava.lang.reflect.InvocationHandler)
wait()
wait(long,int)
wait(long)
getClass()
notify()
notifyAll()

```

图 3.47 运行结果

注意：构造函数的参数类型为 `java.lang.reflect.InvocationHandler`。

通过反射方法可以获取动态代理类的各种函数后，就可以通过函数中的构造函数来创建动态类的实例对象。下面通过一个简单类 `ProxyInstan.java`，来讲解如何实例化动态类，具体内容如代码 3.44 所示。

代码 3.44 动态代理类实例化：ProxyInstan.java

```

public class ProxyInstan {
    public static void main(String[] args) throws IllegalArgumentException,
        InstantiationException, IllegalAccessException,
        InvocationTargetException, SecurityException, NoSuchMethod-
        Exception {
        Class proxy1 = Proxy.getProxyClass(Collection.class.getClass-
        Loader(),
            Collection.class); //获取动态类字节码
        Constructor constructor = proxy1
            .getConstructor(InvocationHandler.class); //获取相关参数构造函数
        //InvocationHandler 类型的类
        class MyInvocationHandler implements InvocationHandler {
            @Override
            public Object invoke(Object proxy, Method method, Object[] args)
                throws Throwable {
                return null;
            }
        }
        Collection collection = (Collection) constructor

```

```

        .newInstance(new MyInvocationHandler1());
        //利用构造函数实例化动态类
//利用匿名类的方式实例化动态类
Collection collection1 = (Collection) constructor
        .newInstance(new InvocationHandler() {
            @Override
            public Object invoke(Object proxy, Method method,
                Object[] args) throws Throwable {
                return null;
            }
        });
//输出动态类实例对象
System.out.println(collection);
System.out.println(collection1);
    }
}

```

运行 ProxyInstan.java 类，控制台窗口如图 3.48 所示。

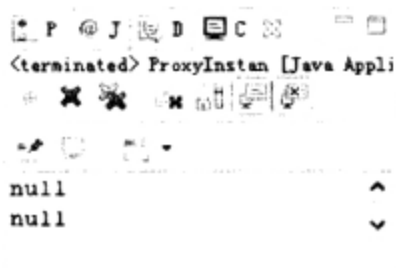



图 3.48 运行结果

【代码解析】

- ❑ 在上述代码中，首先通过代理类 Proxy 的 getProxyClass()方法，获取接口 Collection 的动态类字节码。
- ❑ 由于动态类的构造函数为 constructor()，参数类型为 InvocationHandler，所以在具体实例化动态类时，需要获取到 InvocationHandler 接口的对象。上述代码通过两种方式来调用 constructor()方法，即内部类方式和匿名类方式。

 **注意：**虽然输出结果为 null，但是该值并不表示 collection 和 collection1 对象为 null。

通过上述代码可以发现，如果想创建动态类实例需要经历两个步骤：创建动态类和实例化动态类。那么能不能把两个步骤合并成一个步骤，直接创建动态类实例化对象呢？下面将通过一个简单类 ProxyDirectInstan.java，来讲解如何直接创建动态类实例化对象，具体内容如代码 3.45 所示。

代码 3.45 直接实例化动态代理类：ProxyDirectInstan.java

```

public class ProxyDirectInstan {
    public static void main(String[] args) {
        //通过调用 newProxyInstance() 方法，实例化动态代理类
        Collection proxy1 = (Collection) Proxy.newProxyInstance(
            Collection.class.getClassLoader(),
            new Class[] { Collection.class }, new InvocationHandler() {
                @Override
                public Object invoke(Object proxy, Method method,
                    Object[] args) throws Throwable {
                    return null;
                }
            });
    }
}

```



```

    }
    });
    System.out.println(proxy1);           //输出 proxy1 对象
}
}

```

运行 ProxyDirectInstan.java 类，控制台窗口如图 3.49 所示。

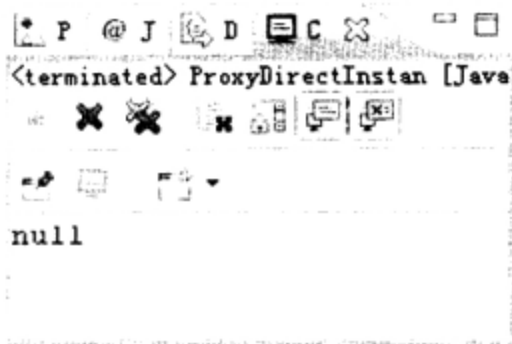


图 3.49 运行结果

【代码解析】

在上述代码中通过 Proxy 类的 newProxyInstance() 方法直接实例化动态代理类，在该方法中需要传入 3 个参数：动态代理类所要实现的接口、动态代理类的类加载器和实现 InvocationHandler 接口对象。

3.7.3 InvocationHandler 接口

通过 3.7.2 节内容可以发现，如果想创建动态代理类对象，需要提供给 JVM 一些必要的信息。首先目标类实现那些接口，即代理类可以拥有接口中的所有方法和一个接受 InvocationHandler 参数的构造函数；接着动态代理类字节码还必须有一个关联的类加载器对象；最后还需要编写动态代理类中的方法的具体代码。

如何编写动态代理类中的方法呢？查看 API 帮助文档可以发现 InvocationHandler 接口，动态代理类的方法必须在该接口对象的方法 invoke() 里编写。对于 InvocationHandler 接口对象则是在创建动态代理类的实例对象的构造方法时传递进去。

当动态代理类调用接口中的相应方法时，程序是如何运行的呢？下面通过一个简单类 InterfaceMethod.java 来讲解动态代理类中接口方法的运行过程，具体内容如代码 3.46 所示。

代码 3.46 接口方法：InterfaceMethod.java

```

public class InterfaceMethod {
    public static void main(String[] args) throws SecurityException,
        NoSuchMethodException, IllegalArgumentException,
        InstantiationException, IllegalAccessException,
        InvocationTargetException {
        Class proxy1 = Proxy.getProxyClass(Collection.class.getClass-
        Loader(),
            Collection.class);           //获取动态类字节码
        Constructor constructor = proxy1
            .getConstructor(InvocationHandler.class);
                                           //获取相关参数构造函数
        // InvocationHandler 类型的类
        class MyInvocationHandler1 implements InvocationHandler {

```



```

        ArrayList target = new ArrayList();    //创建 ArrayList 类型对象
        @Override
        //编写 invoke() 方法
        public Object invoke(Object proxy, Method method, Object[] args)
            throws Throwable {
            //获取系统的当前时间
            long beginTime = System.currentTimeMillis();
            System.out.println("开始时间" + beginTime);
            //调用 target 对象的相应方法
            Object ret = method.invoke(target, args);
            //获取系统的当前时间
            long endTime = System.currentTimeMillis();
            System.out.println("结束时间" + endTime);
            return ret;
        }
    }
    Collection collection = (Collection) constructor
        .newInstance(new MyInvocationHandler1()); //实例化动态代理类
    //调用相应方法
    collection.add("124");
    collection.add("123");
    System.out.println("集合中的元素数"+collection.size());
}

```

运行 InterfaceMethod.java 类，控制台窗口如图 3.50 所示。



图 3.50 运行结果

【代码解析】

- 上述代码中是通过反射的方式来获取动态代理类的构造函数，然后调用该构造函数来实例化动态代理类。根据构造函数的基础知识，可以知道构造函数的运行原理如下所示。

```

$Proxy0 implements Collection{
    InvocationHandler handler;
    public $Proxy0(){
        this.handler=handler;
    }
    ...
}

```

- 如果想彻底理解动态代理类调用接口中的 add() 方法的运行过程，需要理解动态代理类是如何生成 Collection 接口中的代理方法的，具体内容如下：

```

$Proxy0 implements Collection{
    InvocationHandler handler;
    public $Proxy0(){

```

```

        this.handler=handler;
    }
    ...
    int size(){
        return handler.invoke(this,this.getClass().getMethod("size"),null);
    }
    void add(){
        return handler.invoke(this,this.getClass().getMethod("add"), null);
    }
    ...
}

```

因此当运行相关代码时，涉及的 3 要素就会与动态代理类中 add()方法实现如图 3.51 所示的对应。

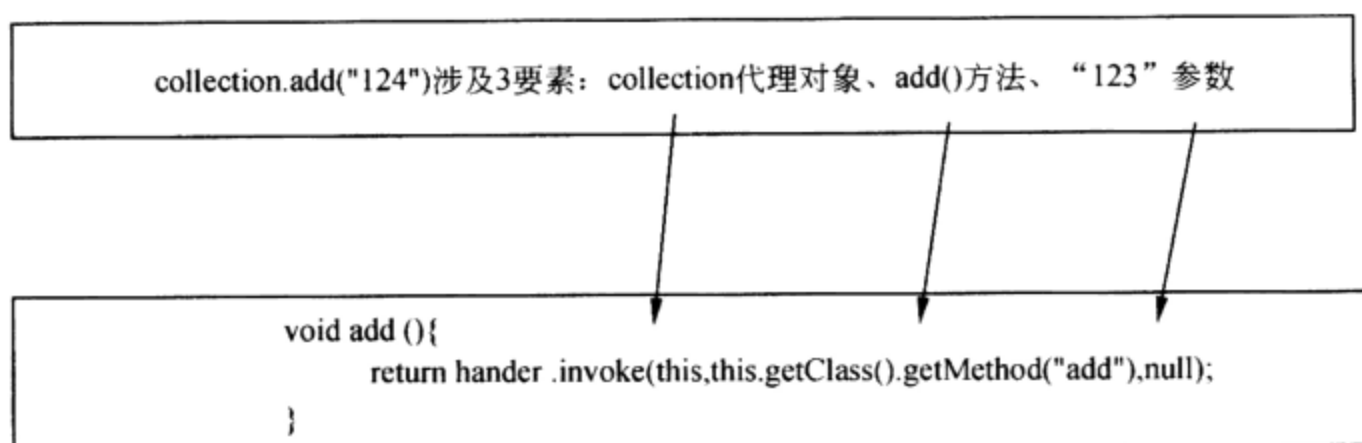


图 3.51 对应关系

动态代理的工作原理如图 3.52 所示。当客户端调用代理类的各个方法时，会把请求传递给由代理类传递进来的 InvocationHandler 对象。InvocationHandler 对象通过 invoke()方法把请求分发给目标对象 (Target) 的各个方法。因此，当调用代理类的 test1()方法时，就会找 Target 对象的 test1()方法；当调用代理类的 test2()方法时，就会找 Target 对象的 test2()方法。

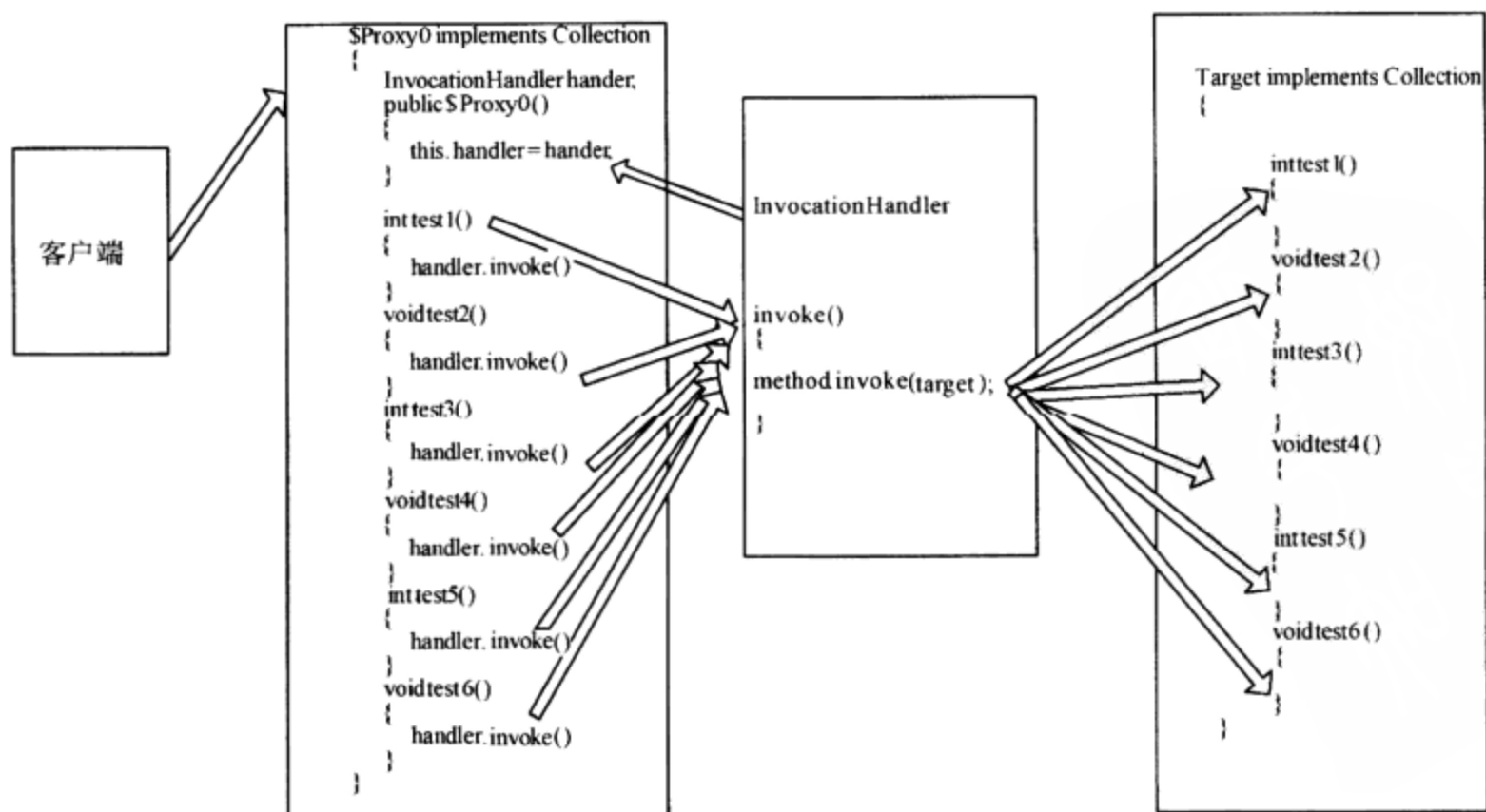


图 3.52 工作原理

3.7.4 动态代理类的设计模式

在 3.7.3 节中介绍了动态代理的工作原理，但是在实际编程中经常会在 `invoke()` 方法中添加一些动态代理类的拦截方法，这时如果还使用前面几节的动态代理设计模式，把拦截方法和目标方法都直接写到 `invoke()` 方法中，则没有任何实际意义。

在实际编程中，应该把拦截方法和目标类以参数的方式传递到 `invoke()` 方法中，以实现程序的最大灵活性。下面通过一个具体的实例来讲解如何把目标对象和拦截方法传递给动态代理类，具体步骤如下。

(1) 创建目标对象类，即需要创建一个名为 `IHello.java` 的接口和实现该接口的名为 `HelloImp.java` 的类，具体内容分别如代码 3.47 所示和代码 3.48 所示。

代码 3.47 接口: `IHello.java`

```
public interface IHello {
    public void toHello(String name);    //toHello() 方法
}
```

代码 3.48 实现类: `HelloImp.java`

```
public class HelloImp implements IHello {
    public void toHello(String name) {    //实现 toHello() 方法
        System.out.println("hello:" + name);
    }
}
```

(2) 创建实现拦截方法类，即需要创建一个名为 `IAdvice.java` 的接口和实现该接口的名为 `AdviceImp.java` 的类，具体内容分别如代码 3.49 和代码 3.50 所示。

代码 3.49 接口: `IAdvice.java`

```
public interface IAdvice {
    public void beforMethod();    //beforMethod() 方法
    public void afterMethod();    //afterMethod() 方法
}
```

代码 3.50 实现类: `AdviceImp.java`

```
public class AdviceImp implements IAdvice {
    @Override
    public void afterMethod() {    //实现 beforMethod() 方法
        System.out.println("before....");
    }
    @Override
    public void beforMethod() {    //实现 afterMethod() 方法
        System.out.println("after....");
    }
}
```

(3) 创建代理类，即需要创建一个名为 ProxyHand.java 的类，具体内容如代码 3.51 所示。

代码 3.51 代理类: ProxyHand.java

```

public class ProxyHand implements InvocationHandler {
    //创建两个成员字段
    private Object target;           //目标对象
    private IAdvice advice;          //拦截方法对象
    public ProxyHand(IAdvice advice) { //带参构造函数
        super();
        this.advice = advice;
    }
    public Object bind(Object target) { //获取动态代理类的实例对象
        this.target = target;          //对目标对象进行赋值
        //通过 Proxy 类的静态方法 newProxyInstance() 获取实例对象
        return Proxy.newProxyInstance(target.getClass().getClassLoader(),
            target.getClass().getInterfaces(), this);
    }
    @Override
    //实现 invoke() 方法
    public Object invoke(Object proxy, Method method, Object[] obj)
        throws Throwable {
        advice.beforeMethod();          //调用相应的拦截方法
        Object result = method.invoke(target, obj); //调用目标对象的相应方法
        advice.afterMethod();           //调用相应的拦截方法
        return result;
    }
}

```

【代码解析】

在上述代码的代理类中，不仅调用了目标类的方法，而且该方法的前面和后面分别调用了拦截方法 doBefore() 和 doAfter() 方法。

(4) 创建客户端，即创建一个名为 ProxyDemo.java 的类来调用代理类，具体内容如代码 3.52 所示。

代码 3.52 测试类: ProxyDemo.java

```

public class ProxyDemo {
    public static void main(String[] args) throws SecurityException,
        NoSuchMethodException {
        //创建动态代理类对象
        ProxyHand ProxyHandler = new ProxyHand(new AdviceImp());
        //实例化动态代理类对象
        IHello hello = (IHello) ProxyHandler.bind(new HelloImp());
        hello.toHello("callan"); //调用目标对象的相应方法
    }
}

```

运行 ProxyDemo.java 类，控制台窗口如图 3.53 所示。

总之，如果采用工厂模式或配置文件的方式进行管理目标类和代理类，则不需要修改

客户端程序就可以实现程序的修改。



图 3.53 运行结果

3.8 小 结

本章详细讲解 Java 语言的高级特性，例如扩展了导入功能的静态导入功能、扩展了函数功能的可变参数函数、增强版 for 循环、基本数据的拆、装箱操作等。除了这些增强功能语法外，还详细介绍了一些新增加的语法，例如枚举语法、发射语法、标注语法、泛型语法、类加载器和动态代理。掌握这些就是掌握 Java 语言发展的基石，是学习好 Java 语言最重要的部分。

第2篇 线程开发

- ▶▶ 第4章 学生并发接水（线程 Thread）
- ▶▶ 第5章 模拟做饭场景（线程的 join()方法）
- ▶▶ 第6章 火车站售票系统（线程安全知识）
- ▶▶ 第7章 生产者与消费者问题（线程通信知识）
- ▶▶ 第8章 关机工具（Timer 类+系统命令）

第 4 章 学生并发接水（线程 Thread）

在现实生活中，经常会同时发生两件事情，例如“两个人同时过一个独木桥”或“两个人同时过一个独木门”。如果要用程序来模拟，则需要用到 Java 语言中的线程。线程是 Java 语言中一个最重要的机制，它能够使程序可以有很多“分身”在计算机中运行，实现复杂的功能。本章除了详细讲解如何通过线程实现许多同学并发接水的实例外，还将详细地介绍线程的基本知识。

本章的学习目标如下：

- ❑ 掌握学生并发接水实例；
- ❑ 理解为什么要使用线程；
- ❑ 掌握创建新线程的两种方式。

4.1 学生并发接水原理

所谓“学生并发接水”，是指学校里学生接水的 3 个过程：下课后许多同学从教室到水房的过程，接水过程和从水房到教室的过程。

4.1.1 项目结构框架分析

对于学生并发接水项目，根据面向对象的思想，需要创建两个对象，即学生和水龙头。该项目中的 3 个包分别描述了现实中存在的不排队接水、排队接水和接完水后一起回教室的 3 种情况。

学生并发接水项目目录如图 4.1 所示，各个目录功能如下。

- ❑ 包 com.cjg.noqueue：模拟学生并发接水时，没有排队。
- ❑ 包 com.cjg.queue：模拟学生并发接水时，有组织地排队。
- ❑ 包 com.cjg.together：模拟学生通过排队形式接完水后，一起回教室。

4.1.2 项目功能业务分析

本节将以直观的方式向读者介绍整个项目要实现的功能。这些功能包括不排队接水、排队接水和接完水后统一回教室。

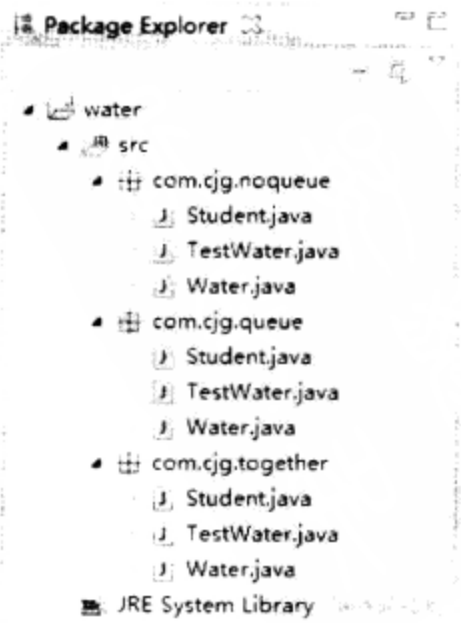


图 4.1 项目目录

1. 不排队接水

假如有 4 个学生小常、小尚、小王和小刘下课后就往水房跑，在具体接水的过程中，由于水房中只有一个水龙头，所以 4 个学生通过抢夺形式接水，因此谁先接完水并不确定。具体过程如图 4.2 所示。

在图 4.2 中，虽然学生小常第一个跑到水房，但是由于在具体接水过程中没有排队，所以学生小王第一个接完水，第一个跑回了教室，而不是小常。

2. 排队接水

4 个学生小常、小尚、小王和小刘下课后就往水房跑，在具体接水的过程中，虽然水房中只有一个水龙头，但是 4 个学生通过排队形式接水。所以谁先开始接水谁就会先结束接水，具体过程如图 4.3 所示。

在图 4.3 中，既然学生小常第一个进行接水，所以会第一个接完水，第一个跑回教室。学生小刘最后一个接水，所以会最后一个接完水，最后一个跑回教室。这是因为这些学生是通过排队的形式进行接水。

3. 接完水后一起回教室

4 个学生小常、小尚、小王和小刘下课后就往水房跑，在具体接水的过程中，虽然通过排队的形式接水，但是在接完水后却没有跑回教室，而是等其他学生全部接完水才向教室跑，具体过程如图 4.4 所示。在图 4.4 中，虽然学生小常第一个接水、第一个接完水，但是其接完水后却没马上跑回教室，而是等其他学生接完水后一起跑回教室。

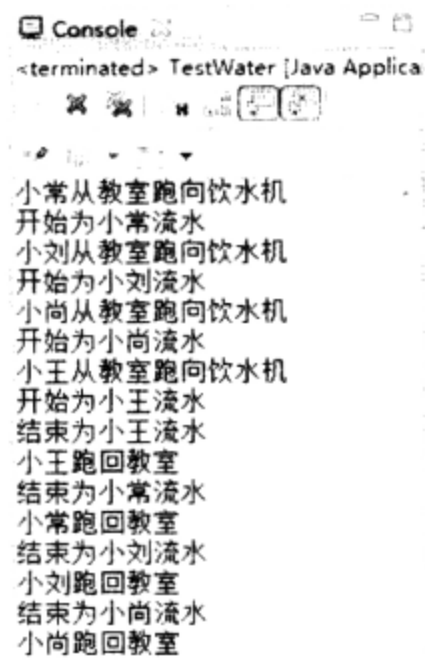


图 4.2 不排队运行结果

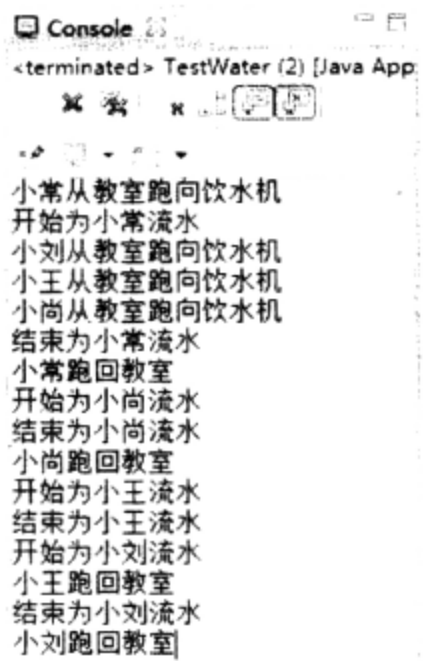


图 4.3 排队运行结果

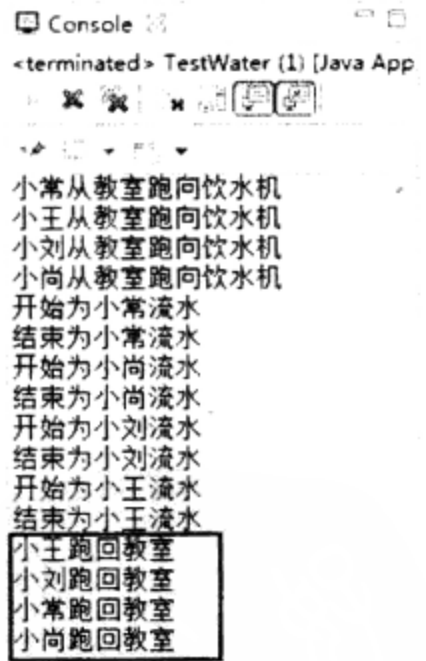


图 4.4 一起回教室运行结果

4.2 不排队形式学生并发接水

本章通过线程技术来模拟学生不排队接水的过程，具体程序架构如图 4.5 所示，它包

含两个类对象 Water.java、Student.java 和一个进行测试的类 TestWater.java。

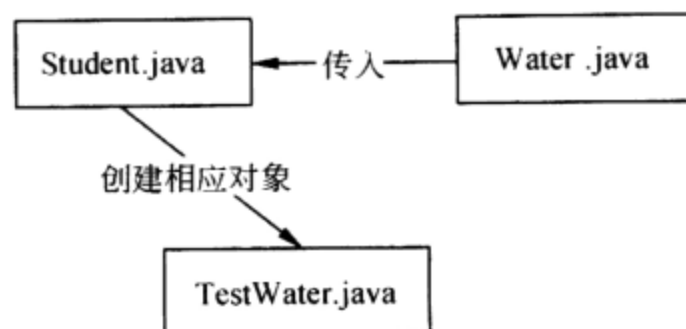


图 4.5 程序关系图

4.2.1 水龙头类

Water.java 类用来模拟现实生活中的水龙头，由于水龙头主要具有流水的功能，所以该类中拥有一个流水的方法，具体内容如代码 4.1 所示，该类的 UML 如图 4.6 所示。

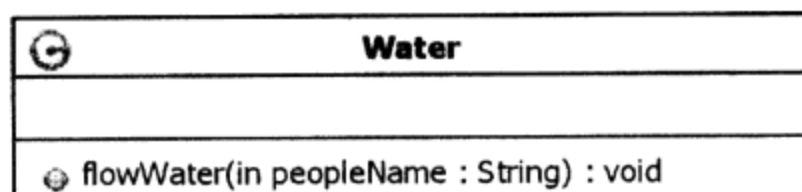


图 4.6 Water 类图

代码 4.1 水龙头类：Water.java

```
public class Water {  
    public void flowWater(String studentName) { //流水的方法  
        System.out.println("开始为" + studentName + "流水");//为某学生流水  
        try {  
            Thread.sleep(3000); //线程休眠  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
        System.out.println("结束为" + studentName + "流水");  
        //结束为某同学流水  
    }  
}
```

【代码解析】

上述代码由于主要是讲解线程的机制，所以如何开始流水给学生及如何结束流水给学生的过程就简化了，只是输出了一句代码。

4.2.2 学生类

Student.java 类用来模拟现实生活中的学生，由于学生要实现并发接水的过程，所以接水的代码全部放在了 run()方法里，具体内容如代码 4.2 所示，该类的 UML 如图 4.7 所示。

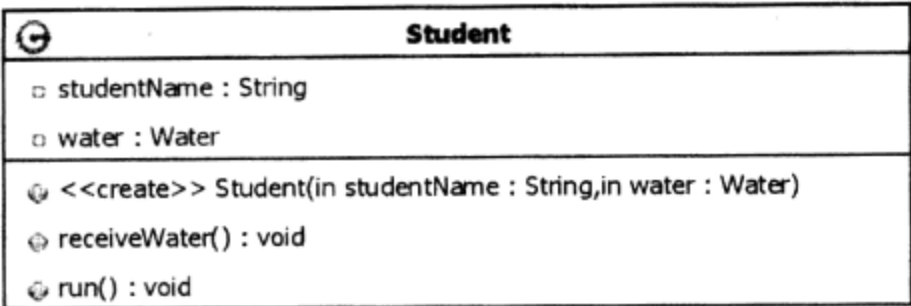


图 4.7 学生类图

代码 4.2 学生类：Student.java

```
public class Student extends Thread { //继承了 Thread 线程类
    //创建了两个成员字段
    private String studentName; //学生名字的字
    private Water water; //水龙头的字
    public Student(String studentName, Water water) { //带参构造函数
        super();
        this.studentName = studentName;
        this.water = water;
    }
    public void receiveWater() { //学生接水的全过程
        //学生跑向水龙头
        System.out.println(studentName + "从教室跑向饮水机");
        water.flowWater(studentName); //水龙头流水
        System.out.println(studentName + "跑回教室"); //学生跑回教室过程
    }
    public void run() { //实现 run() 方法
        receiveWater();
    }
}
```

【代码解析】

- ❑ 上述代码主要是讲解线程的机制，所以学生如何跑向水龙头及学生如何跑向教室的过程就简化了，只是输出了一句代码。
- ❑ Student 类首先继承了线程类 Thread，所以在具体编写该类时，必须把需要并发的业务功能（receiveWater()）编写在 run()方法中，以达到实现 run()方法的目的。

4.2.3 测试类

TestWater.java 类用来测试学生和水龙头的类，是否实现了不排队接水的过程，具体内容如代码 4.3 所示。

代码 4.3 测试类：TestWater.java

```
public class TestWater {
    public static void main(String[] args) {
        Water water = new Water(); //创建一个水龙头对象
        //创建了 4 个学生对象
        Student xiaochang = new Student("小常", water); //学生小常
        Student xiaoshang = new Student("小尚", water); //学生小尚
        Student xiaowang = new Student("小王", water); //学生小王
    }
}
```

```

        Student xiaoliu = new Student("小刘", water); //学生小刘
        //启动 4 个线程
        xiaochang.start(); //启动 xiaochang 线程
        xiaoshang.start(); //启动 xiaoshang 线程
        xiaowang.start(); //启动 xiaowang 线程
        xiaoliu.start(); //启动 xiaoliu 线程
    }
}

```

【代码解析】

在上述代码中，首先创建了一个水龙头对象 `water` 和 4 个学生对象 `xiaochang`、`xiaoshang`、`xiaowang` 和 `xiaoliu`，然后通过 `Thread` 类的 `start()` 方法创建 4 个新线程模拟学生并发接水的功能。

4.3 学生并发接水的其他形式

本节通过线程技术来模拟排队接水和接完水后一起回教室的过程，由于这两个项目与“不排队接水”项目中的代码只有水龙头 `Water` 类的代码不一样，而其他代码完全相同，所以本节只讲解“排队接水”和“接完水后一起回教室”项目中 `Water` 类的内容。

4.3.1 “排队接水”水龙头类

`Water` 类用来模拟现实生活中的水龙头，由于学生需要排队接水，所以在该类中的流水方法中定义为同步方法，具体内容如代码 4.4 所示。

代码 4.4 水龙头类：Water.java

```

public class Water {
    public synchronized void flowWater(String studentName) {
        //定义为同步方法
        System.out.println("开始为" + studentName + "流水");//输出相应信息
        try {
            Thread.sleep(3000); //线程休眠 3 秒
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("结束为" + studentName + "流水");//输出相应信息
    }
}

```

【代码解析】

在上述代码中，通过关键字 `synchronized` 修饰 `flowWater()` 方法为同步方法。这样该方法就具有原子性，再也不会出现不排队接水的情况。

4.3.2 “接完水后一起回教室”水龙头类

`Water` 类用来模拟现实生活中的水龙头，由于学生需要接水，所以在该类中的流水方

法中除了定义同步块外还需要进行一些必要的设置,具体内容如代码4.5所示,该类的UML如图4.8所示。

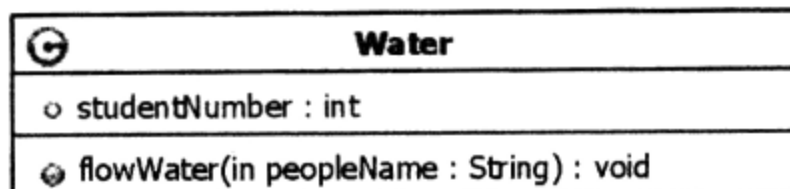


图 4.8 水龙头类图

代码 4.5 水龙头类: Water.java

```

public class Water {
    int studentNumber;                //定义一个学生人数的变量
    public synchronized void flowWater(String studentName) {
        //定义同步块
        synchronized (this) {
            ++studentNumber;           //学生人数自增
            //输出相应信息
            System.out.println("开始为" + studentName + "流水");
            try {
                Thread.sleep(3000);    //线程休眠 3 秒
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            //输出相应信息
            System.out.println("结束为" + studentName + "流水");
            if (studentNumber < 4) {
                try {
                    wait();             //使线程进入等待状态
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            } else {
                notifyAll();           //唤醒所有等待线程
            }
        }
    }
}
  
```

【代码解析】

在上述代码中,首先定义了一个学生人数的成员变量 studentNumber,学生每接水一次(线程每执行一次同步块), studentNumber 变量的值就会加 1。在学生接完水后,如果 studentNumber 值小于 4,系统就通过 wait()方法使该学生(线程)处于等待状态。如果 studentNumber 值为 4,系统就通过 notifyAll()方法唤醒所有处于等待状态的学生(线程),这样就可以实现“接完水后一起回教室”的效果。

4.4 知识点扩展——线程的基础知识

所谓线程就是一个程序内部的一条执行线索,所以就出现了单线程程序和多线程程

序，即一个程序包含一个线程或多个线程。本节将详细介绍为什么要在程序中使用线程机制，和创建线程的各种方式。

4.4.1 为什么要使用线程

如果想了解线程，需要先了解一些基本概念。多任务操作系统是指能够同时执行多个应用程序的操作系统。进程是指正在执行的程序。多任务操作系统能够以非常小的时间间隔交替执行多个程序。那么程序中为什么要使用线程呢？为了解决这该问题，通过下面的两段程序来讲解。

(1) 编写一段程序，在该程序中有两个类，GenerClass 类为主类，CallClass 类为被调用的类，具体内容如代码 4.6 所示。

代码 4.6 普通类：GenerClass.java

```
public class GenerClass {
    public static void main(String[] args) {
        new CallClass().start();           //调用 CallClass 类的 start() 方法
        while (true) {                     //循序语句
            System.out.println("GenerClass 类运行了");
        }
    }
}
class CallClass {                          //定义 CallClass 类
    public void start() {                  //编写 start() 方法
        run();
    }
    public void run() {                    //编写 run() 方法
        while (true) {
            System.out.println("CallClass 类被调用类运行了");
        }
    }
}
```

运行 GenerClass.java 类，控制台窗口如图 4.9 所示。

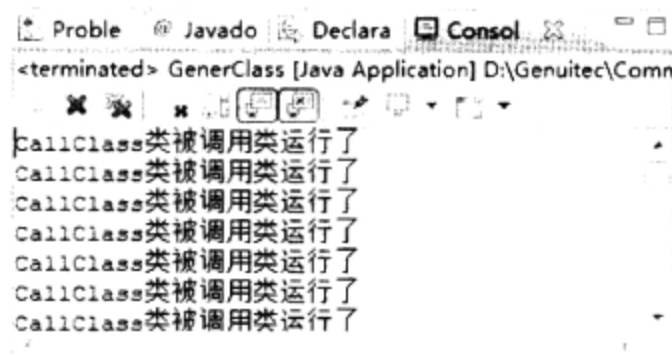


图 4.9 运行结果

【代码解析】

对于运行结果，命令窗口中会不停地输出“CallClass 类被调用类运行了”语句，而不会输出“GenerClass 类运行了”语句，这说明 new CallClass().start() 由于处于其他代码的前面，所以具有较高的优先权。而在 start() 中调用的 run() 方法实现了无限循环，因此该代码后面的语句就没机会执行。

⚠注意：上述代码的文件名之所以为 GenerClass.java 而不是 CallClass.java，是因为 CallClass 类定义时其前面的修饰符无 public 关键字，而 GenerClass 类却有。

(2) 为了实现不仅输出“CallClass 类被调用类运行了”语句，而且还输出“GenerClass 类运行了”语句，可以修改类 GenerClass 的内容如代码 4.7 所示。

代码 4.7 修改后的类：GenerClass.java

```
public class GenerClass {
    public static void main(String[] args) {           //主方法
        new CallClass().start();                       //启动线程
        while (true) {
            System.out.println("GenerClass 类运行了"); //输出相应信息
        }
    }
}
class CallClass extends Thread {                     //继承了 Thread 类
    public void run() {                                //实现方法 run()
        while (true) {                                //实现遍历
            System.out.println("CallClass 类被调用类运行了"); //输出相应信息
        }
    }
}
```

运行 GenerClass.java 类，控制台窗口如图 4.10 所示。



图 4.10 运行结果

【代码解析】

- ❑ 经过修改后，两个类中的循序代码可以交替运行。这主要是因为 CallClass 类继承了 Thread 类，并实现了该类中的 run() 方法。
- ❑ 在上述代码中，代码 new CallClass() 为创建一个新线程，启动该线程则通过 start() 方法来实现。

总之，通过对上述代码的比较可以发现，修改前的 GenerClass 类是单线程程序，所以在该类的 main() 方法中，CallClass().start() 方法返回后才能继续往下执行。而修改后的 GenerClass 类是多线程程序，所以在该类的 main() 方法中，不必等 CallClass().start() 方法返回后就继续往下执行，即 CallClass().start() 方法与下面的代码在不同的代码上运行。

4.4.2 多线程程序的编写方式

一个代码段被执行，一定是在某个线程上运行，同一段代码可以与多个线程相关联，

在多个线程上执行的也可以是相同的一段代码。那么如何编写多线程程序呢？

根据 API 的帮助文档，可以发现代表线程的类 `Thread` 和接口 `Runnable`，因此可以通过两种方式来实现多线程程序。

1. 继承 `Thread` 类

一个 `Thread` 类的对象就代表一个线程，下面通过打印机的类来讲解如何应用 `Thread` 类，具体步骤如下。

(1) 根据面向对象的思想编写一个打印机的类 `PrintThread`，具体内容如代码 4.8 所示，该类的 UML 如图 4.11 所示。

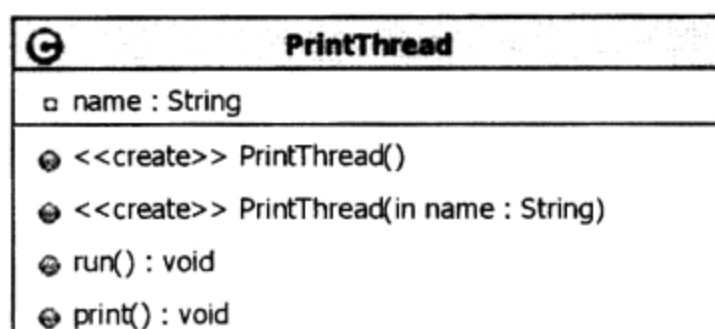


图 4.11 打印机的类图

代码 4.8 打印机类: `PrintThread.java`

```

public class PrintThread extends Thread {
    private String name;                //打印机的名字
    public PrintThread() {              //无参构造函数
    }
    public PrintThread(String name) {    //有参构造函数
        super();
        this.name = name;
    }
    public void run() {                 //实现 run() 方法
        print();
    }
    public void print() {               //打印机业务的方法
        System.out.println(name + "开始打印");
        try {
            Thread.sleep(44);           //线程休眠
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println(name + "结束打印"); //输出相应信息
    }
}
  
```

【代码解析】

打印机类继承了线程类 `Thread`，所以具有了线程类 `Thread` 的全部特征。对于继承了类 `Thread` 的类，只要实现 `run()` 方法就可以。要将一段代码在一个新的线程上运行，这段代码应该在一个类的 `run()` 方法中，同时 `run()` 方法所在的类是 `Thread` 类的子类。

注意：如果要实现多线程，可以编写继承了类 `Thread` 的子类，子类要覆盖 `Thread` 类中的 `run()` 方法，在 `run()` 方法中调用想在新线程上运行的程序代码。

(2) 接着编写一个多线程类 TestThread1, 在该类中调用了打印机的类, 具体内容如代码 4.9 所示。

代码 4.9 多线程类: TestThread1.java

```
public class TestThread1 {  
    public static void main(String[] args) {  
        //创建了 4 个打印机对象  
        PrintThread pNT1 = new PrintThread("打印机 1");  
        PrintThread pNT2 = new PrintThread("打印机 2");  
        PrintThread pNT3 = new PrintThread("打印机 3");  
        PrintThread pNT4 = new PrintThread("打印机 4");  
        //启动线程  
        pNT1.start();  
        pNT2.start();  
        pNT3.start();  
        pNT4.start();  
    }  
}
```

运行 TestThread1.java 类, 控制台窗口如图 4.12 所示。



图 4.12 运行结果

【代码解析】

由于一个 Thread 类对象就是一个线程, 所以在上述代码中创建了 4 个线程, 分别为 pNT1、pNT2、pNT3 和 pNT4。启动一个线程, 不是直接调用 Thread 类子类对象的 run() 方法, 而是调用 Thread 类子类对象的 start()方法。

(3) 根据面向对象思想的继承特性, 可以修改 TestThread1 类的内容如代码 4.10 所示。

代码 4.10 多线程类: TestThread2.java

```
public class TestThread2 {  
    public static void main(String[] args) {  
        //创建了 4 个打印机对象  
        Thread pNT1 = new PrintThread("打印机 1");  
        Thread pNT2 = new PrintThread("打印机 2");  
        Thread pNT3 = new PrintThread("打印机 3");  
        Thread pNT4 = new PrintThread("打印机 4");  
        //启动线程  
        pNT1.start();  
        pNT2.start();  
        pNT3.start();  
        pNT4.start();  
    }  
}
```

```
    }
}
```

运行 TestThread2.java 类，会出现 TestThread1 类的运行结果。

【代码解析】

通过 Thread 子类对象的 start()方法启动一个新线程时，根据面向对象的多态性，在该线程上实际运行的是 Thread 子类对象中（Thread pNT1 等）的 run()方法。因此当把 Thread pNT1 等对象当作父类 Thread 来对待时，调用的同样是子类覆盖后的方法。

（4）查看 API 帮助文档，可以发现 Thread 类除了 Thread()构造函数外，还存在 Thread(Runnable target)构造函数。因此可以修改 TestThread1 类的内容如代码 4.11 所示。

代码 4.11 多线程类：TestThread3.java

```
public class TestThread3 {
    public static void main(String[] args) {
        //创建了 4 个打印机对象
        Thread pNT1 = new Thread(new PrintThread("打印机 1"));
        Thread pNT2 = new Thread(new PrintThread("打印机 2"));
        Thread pNT3 = new Thread(new PrintThread("打印机 3"));
        Thread pNT4 = new Thread(new PrintThread("打印机 4"));
        //启动线程
        pNT1.start();
        pNT2.start();
        pNT3.start();
        pNT4.start();
    }
}
```

运行 TestThread3.java 类，会出现 TestThread1 类的运行结果。

【代码解析】

根据一个 Thread 类对象就是一个线程可以知道，TestThread3 类中又创建了 6 个线程。那么上述代码中的 6 个线程是如何运行呢？如果想了解该过程，可以查看 Thread 类的源代码，具体内容如下所示。

```
class Thread implements Runnable {
    private Runnable target;
    public Thread() {
        init(null, null, "Thread-" + nextThreadNum(), 0);
    }
    public Thread(Runnable target) {
        init(null, target, "Thread-" + nextThreadNum(), 0);
    }
    public void run() {
        if (target != null) {
            target.run();
        }
    }
}
```

从上述代码的 run()方法中可以发现，如果存在 target 对象，则会调用该对象的 run()方法，即 new PrintThread("打印机 1")对象的 run()方法。

总之，通过 Thread()构造函数创建的线程对象，将调用线程对象中的 run()方法作为运行代码。通过 Thread(Runnable target)构造函数创建线程对象，需要传递一个线程类对象，

这样创建的线程将调用那个传递进来的线程对象的 `run()` 方法。

2. 实现Runnable接口

查看 `Runnable` 接口类的帮助文档，可以发现该接口中只有一个 `run()` 方法。那么如何通过实现接口 `Runnable` 来创建多线程应用？下面通过修改打印机的类 `PrintThread` 来讲解，该类的 UML 如图 4.13 所示，具体内容如代码 4.12 所示。

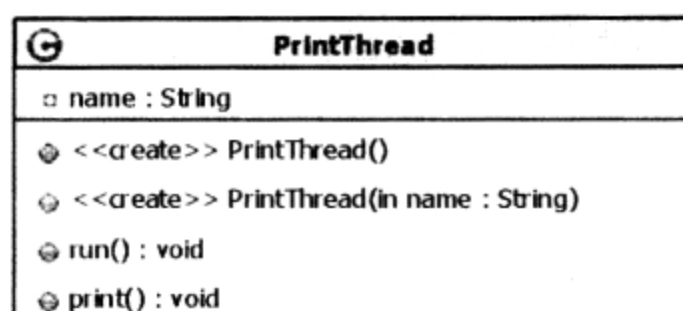


图 4.13 打印类的类图

代码 4.12 打印机类: `PrintThread.java`

```

public class PrintThread implements Runnable {
    private String name;                                //打印机的名字
    public PrintThread() {                               //无参构造函数
    }
    public PrintThread(String name) {                   //带参构造函数
        super();
        this.name = name;
    }
    public void run() {                                  //实现 run() 方法
        print();
    }
    public void print() {                                //编写打印机的业务方法
        System.out.println(name + "开始打印");
        try {
            Thread.sleep(44);                            //线程休眠一段时间
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println(name + "结束打印");          //输出相应信息
    }
}
  
```

那么如何创建 `PrintThread` 类的对象呢？创建一个测试 `TestThread` 类，具体内容如代码 4.13 所示。

代码 4.13 测试类: `TestThread.java`

```

public class TestThread {
    public static void main(String[] args) {
        //创建 4 个线程
        PrintThread pNT1 = new PrintThread("打印机 1");
        //Thread pNT2=new PrintThread("打印机 2");
        Thread pNT3 = new Thread(new PrintThread("打印机 3"));
        Thread pNT4 = new Thread(new PrintThread("打印机 4"));
    }
}
  
```

```
//启动线程
// pNT1.start();
// pNT2.start();
pNT3.start();
pNT4.start();
}
```

运行 TestThread.java 类，控制台窗口如图 4.14 所示。



图 4.14 运行结果

【代码解析】

- ❑ 在上述代码中，pNT1 对象能够成功创建，但是不能成功调用 start() 方法。这是因为 Runnable 接口中只有 run() 方法，而没有 start() 方法。
- ❑ pNT2 对象也不能成功创建，这是因为 Thread 类实现了 Runnable 接口，PrintThread 类也实现了 Runnable 接口，即这两个类没有父子关系，所以不能创建成功。
- ❑ pNT3 和 pNT4 对象能够创建成功，也能够调用 start() 方法启动线程，所以实现 Runnable 接口的对象只能通过 Thread(Runnable target) 构造函数来创建线程。

4.5 小 结

本章主要通过线程模拟现实生活中下课后学生到水房的接水场景。这些场景分别为：不排队接水场景、排队接水场景和接完水后一起回教室场景。笔者通过类和实际的代码介绍了真正线程的应用。本章的最后还详细介绍了线程的一些基础知识，通过这些知识的学习，可以创建出简单的多线程程序。

第 5 章 模拟做饭场景(线程的 join()方法)

在现实生活中，经常会发生这样的场景，即有两件事情 A 和 B，事情 A 的结束需要等待事情 B 完成之后，这时如果 A 在 B 还没完成之前就开始，就必须要做等待工作。在 Java 语言中如果能让某个线程等待另一个线程运行完后才继续运行，就需要用到类 Thread 的 join()方法。本章除了详细讲解如何通过线程实现做饭场景的实例外，还将详细介绍线程的 join()方法。

- 本章的学习目标如下：
- ❑ 掌握做饭场景实例；
 - ❑ 理解 Thread 类的 join()方法。

5.1 做饭场景原理

所谓做饭场景，是指妈妈在下午 5 点开始到厨房做晚餐，可是在准备做饭的材料时发现酱油用完了，因此叫儿子去买。妈妈在儿子把酱油买回来之前没什么能干的只好等待。直到儿子把酱油买回来，妈妈才开始煮饭，最后终于把饭煮好。

5.1.1 项目结构框架分析

对于做饭场景项目，根据面向对象的思想，需要创建两个对象，即妈妈和儿子。做饭场景项目目录如图 5.1 所示，各个文件的功能如下。

- ❑ com.cjg.mistake 包：没有通过线程的 join()方法模拟的“做饭场景”。
- ❑ com.cjg.normal 包：通过线程的 join()方法模拟的“做饭场景”。

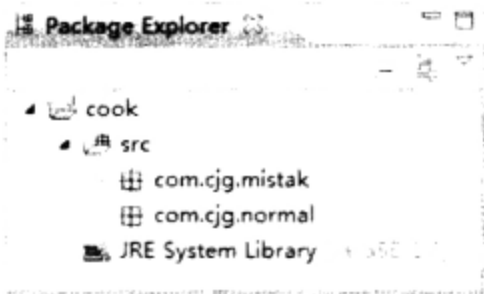


图 5.1 项目目录

5.1.2 项目功能业务分析

本节将以直观的方式向读者介绍整个项目要实现的功能，包括没有使用 join()方法的做

饭场景和使用了 join()方法的做饭场景。

1. 没有使用join()方法的做饭场景

当妈妈叫儿子去买酱油后，儿子就跑去买酱油。在儿子买酱油的 5 分钟里，妈妈没有等待儿子买酱油回到家就做好饭，即妈妈做好饭后，儿子才买回酱油。具体过程如图 5.2 所示。这里的运行结果很乱，完全不符合正常的做饭场景。

2. 使用join()方法的做饭场景

当妈妈叫儿子去买酱油后，儿子就跑去买酱油。在儿子买酱油的 5 分钟里，妈妈一直处于等待状态。当儿子买酱油回到家后，妈妈才又开始她的做饭工作。具体过程如图 5.3 所示。这里的运行结果，完全符合正常的“做饭场景”。

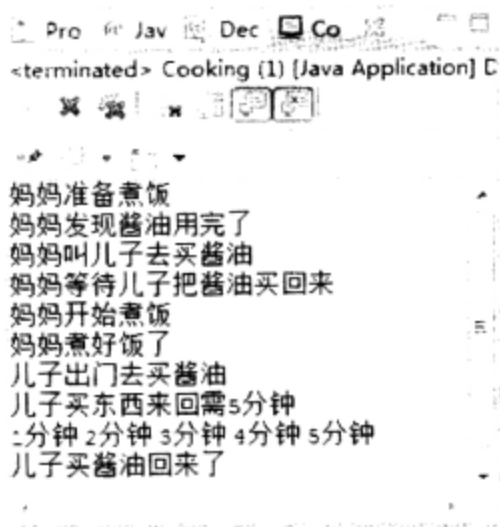


图 5.2 不正确的运行结果

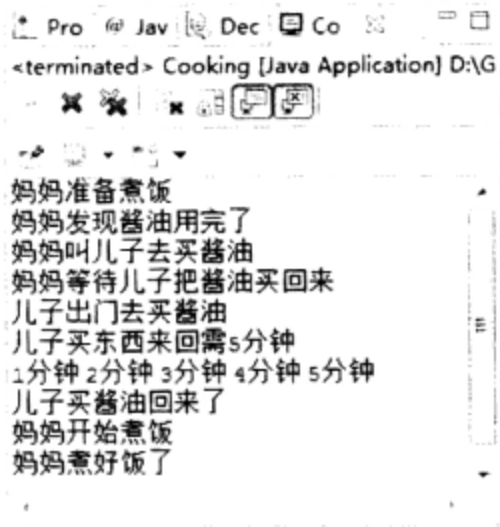


图 5.3 正确的运行结果

5.2 纷乱的做饭场景

本章通过线程技术来模拟纷乱的做饭场景过程，具体程序架构如图 5.4 所示，它包含两个类对象，Son.java、Mother.java 和一个做饭的类 Cooking.java。

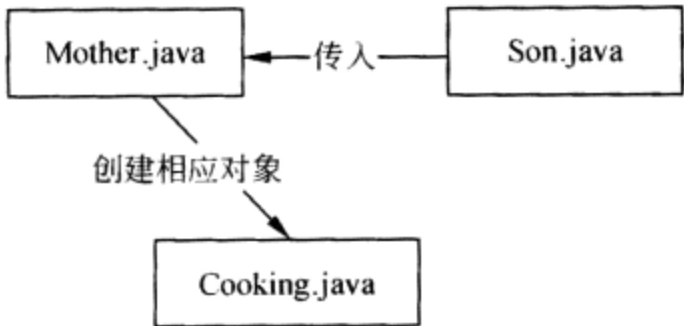


图 5.4 程序关系图

5.2.1 儿子的类

Son.java 类用来模拟现实生活中的儿子，儿子需要在 5 分钟内买酱油回家，具体内容

如代码 5.1 所示，该类的 UML 如图 5.5 所示。

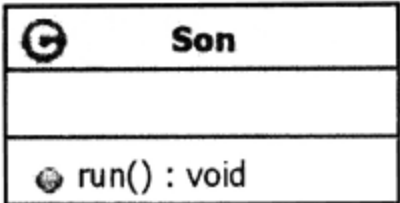


图 5.5 Son 类图

代码 5.1 儿子类：Son.java

```
public class Son implements Runnable {
    public void run() {
        //输出相应的信息，即儿子开始出去买酱油
        System.out.println("儿子出门去买酱油");
        System.out.println("儿子买东西来回需 5 分钟");
        //买酱油的过程
        try {
            for (int i = 1; i <= 5; i++) {
                Thread.sleep(1000);
                System.out.print(i + "分钟 ");
            }
        } catch (InterruptedException ie) {
            System.err.println("儿子发生意外");
        }
        //输出相应信息，即儿子买酱油回来
        System.out.println("\n 儿子买酱油回来了");
    }
}
```

【代码解析】

上述代码主要是讲解线程的机制，利用 sleep()方法模拟儿子买酱油的过程。

5.2.2 妈妈的类

Mother.java 类用来模拟现实生活中的妈妈，由于妈妈要实现做饭的过程，所以做饭的代码全部放在了 run()方法里，具体内容如代码 5.2 所示，该类的 UML 如图 5.6 所示。

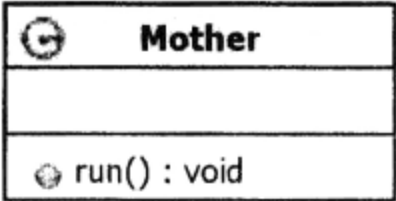


图 5.6 妈妈类图

代码 5.2 妈妈类：Mother.java

```
public class Mother implements Runnable {           //实现 Runnable 接口
    public void run() {                             //实现 run() 方法
        //输出相应的信息，即妈妈发现没酱油，叫儿子去买酱油回来
        System.out.println("妈妈准备煮饭");
        System.out.println("妈妈发现酱油用完了");
        System.out.println("妈妈叫儿子去买酱油");
        Thread son = new Thread(new Son());        //创建一个儿子的线程对象
        son.start();                                //启动儿子线程
        System.out.println("妈妈等待儿子把酱油买回来");
        //儿子买酱油回家后，妈妈开始做饭
    }
}
```

```

        System.out.println("妈妈开始煮饭");
        System.out.println("妈妈煮好饭了");
    }
}

```

【代码解析】

- 上述代码主要是讲解线程的机制，所以妈妈如何让儿子去买酱油和如何煮好饭就简化了，只是输出了一句代码。
- 当妈妈叫儿子去买酱油，在程序中就是产生了一个 Son 对象，然后通过 Thread.start() 方法启动该线程。

5.2.3 做饭场景的类

Cooking.java 类用来测试妈妈和儿子的类，是否实现了“做饭场景”的过程，具体内容如代码 5.3 所示。

代码 5.3 测试类: Cooking.java

```

public class Cooking {
    public static void main(String argv[]) {           //主方法
        Thread mother = new Thread(new Mother());      //创建对象 mother
        mother.start();                                 //启动线程
    }
}

```

【代码解析】

上述代码中，首先创建了一个妈妈对象 mother，然后通过 Thread 类的 start() 方法启动该线程，这样就可以模拟做饭的场景。

5.2.4 修改后的妈妈类

通过 5.2.3 节 Cooking 类的运行结果可以发现不能模拟正常的做饭场景，因为妈妈还没等儿子把酱油买回来就把饭做好了。为了实现正确的做饭场景，可以修改 Mother 类的具体内容，如代码 5.4 所示。

代码 5.4 妈妈类: Mother.java

```

public class Mother implements Runnable {             //实现 Runnable 接口
    public void run() {                                //实现 run() 方法
        //输出相应的信息，即妈妈发现没酱油，叫儿子去买酱油回来
        System.out.println("妈妈准备煮饭");
        System.out.println("妈妈发现酱油用完了");
        System.out.println("妈妈叫儿子去买酱油");
        Thread son = new Thread(new Son());           //创建一个儿子的线程对象
        son.start();                                   //启动儿子线程
        System.out.println("妈妈等待儿子把酱油买回来");
        try {
            //当两个线程合并成功后，妈妈线程的执行必须要等儿子线程执行完毕
            son.join();                                 //合并儿子和妈妈线程
        }
    }
}

```



```

    } catch (InterruptedException ie) {
        //输出相应的信息
        System.err.println("发生异常!");
        System.err.println("妈妈中断煮饭");
        System.exit(1); //退出系统
    }
    //儿子买酱油回家后，妈妈开始做饭
    System.out.println("妈妈开始煮饭");
    System.out.println("妈妈煮好饭了");
}
}

```

【代码解析】

当妈妈叫儿子去买酱油时，在程序中就产生了一个 Son 对象，然后通过 Thread.start() 方法去启动该线程。由于妈妈必须等待儿子把酱油买回来才能开始做饭，所以通过 Thread 类的 join() 方法，把儿子线程与妈妈线程合并在一起，这样就实现了等儿子把事情完成后，妈妈才继续煮饭。由于妈妈在等待儿子买酱油回来的过程中，可能会出现其他事情打断煮饭这件事，所以需要处理相应的异常。

5.3 知识点扩展——线程的状态

线程同样也有生命周期，在自己的生命周期中会经历 5 种状态：new（创建）、runnable（等待运行）、running（运行）、blocked（暂停）和 dead（结束），它们的关系如图 5.7 所示。

5.3.1 线程的创建状态

通过前面的学习可以知道，当产生 Thread 类对象时，该对象就会处于 new 状态，调用 start() 方法后就会进入 runnable 状态。如果 CPU 调用该线程时，其就会从 runnable 状态进入 running 状态，运行线程中 run() 方法中的代码。具体过程如图 5.8 所示。

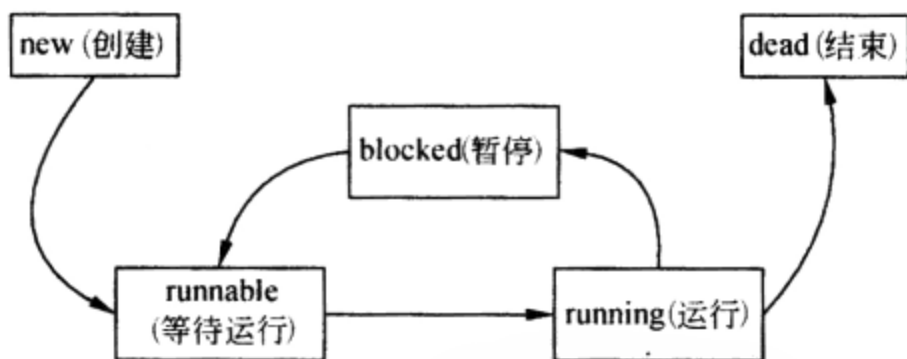


图 5.7 线程的状态转换

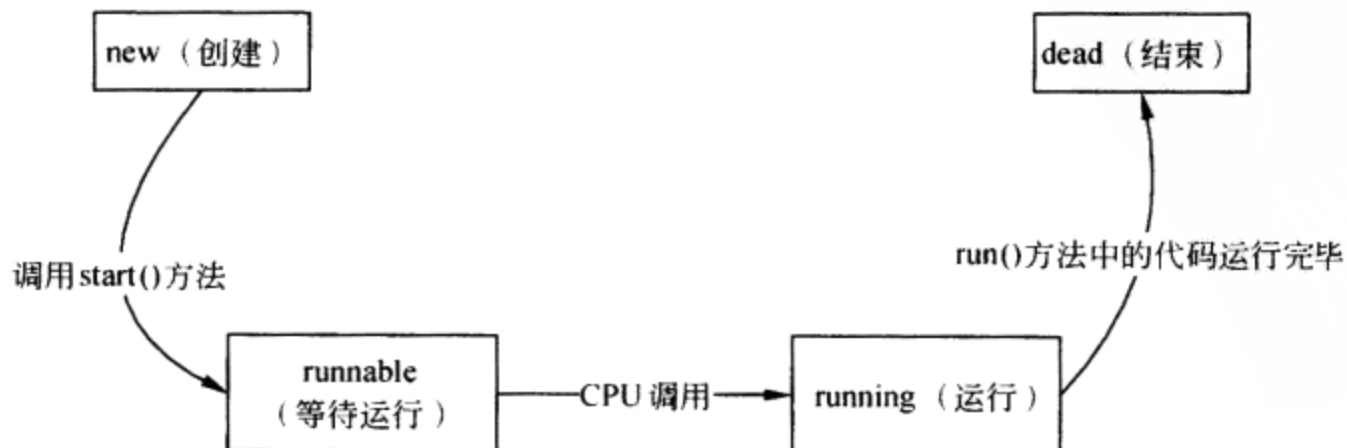


图 5.8 线程状态转换

对于一个 Thread 类对象，能够多次调用 start()方法来启动多个线程吗？下面通过一个名为 NewState 的类来测试一下，具体内容如代码 5.5 所示。

代码 5.5 测试启动方法：NewState.java

```
public class NewState {
    public static void main(String[] args) {
        Thread tt = new test();           //创建一个新线程
        //启动线程
        tt.start();
        tt.start();
        tt.start();
    }
}
class test extends Thread {              //继承线程的类
    public void run() {                   //实现 run() 方法
        System.out.println("测试 start 方法");
    }
}
```

运行 NewState.java 类，控制台窗口如图 5.9 所示。



图 5.9 NewState.java 类运行结果

【代码解析】

在上述代码中，线程对象 tt 调用了多次 start()方法，这是不被允许的。因为线程从 new 状态到 runnable 状态是单行道，如果对已经处于 runnable 状态（调用 start()方法后）的线程再次调用 start()方法，就会产生“java.lang.IllegalThreadStateException”异常。

5.3.2 线程的暂停状态

对于线程对象，如果想实现让其暂时停止，但是恢复运行后又不产生一个新的线程对象时，就需要学习如何使线程处于暂停状态。在 Java 语言中可以通过如图 5.10 所示的 4 种方式来实现该功能。

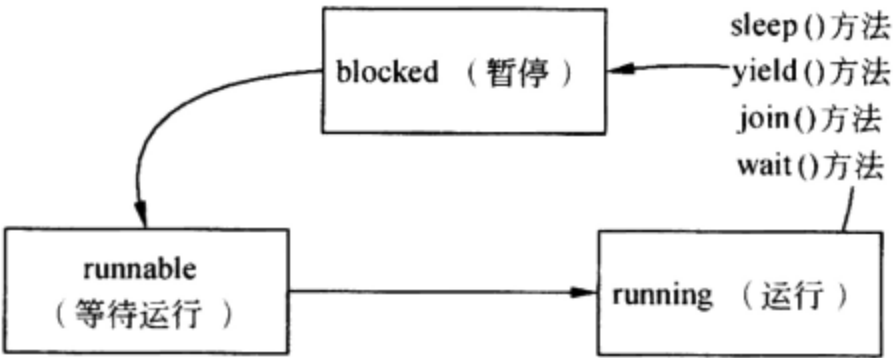


图 5.10 线程状态转换


下面详细介绍进入 blocked 状态的 4 种方式。

1. sleep()方法

sleep 这个单词英文意思为睡觉，其主要用来实现让 Thread 对象“睡觉”，不过在睡觉时设置了个闹钟，时间一到就会恢复运行。该方法的基本格式如下：

```
Thread.sleep(long millis)
```

参数 millis 用来表示睡眠的时间，这个时间的单位为千分之一秒，即如果想睡 1 秒的话，则需要传入 1000 秒。

注意：当一个 Thread 类对象睡醒后，不是立刻进入 running 状态运行，而是进入 runnable 状态。

2. yield()方法

当 CPU 正调用一个线程运行时，如果该线程睡眠一段时间后才被执行，就需要调用 sleep()方法；如果该线程不想这时候运行，可以调用 yield()方法。对于 sleep()方法，线程在指定的时间里肯定不会由 running 状态转换到 runnable 状态；而对于 yield()方法，线程可能马上由 running 状态转换到 runnable 状态。该方法的基本格式如下：

```
Thread.yield()
```

对于一个 Thread 类对象，如果想让其立即由 running 状态转换到 runnable 状态，即立刻暂时停止运行，可以调用 yeild()方法。下面通过一个名为 TestBlockedState 的类来测试一下，具体内容如代码 5.6 所示。

代码 5.6 测试暂停方法：TestBlockedState.java

```
class BlockedState extends Thread {           //创建类 BlockedState
    String name;                               //定义一个变量 name
    public BlockedState(String n) {            //构造函数
        name = n;                             //初始化字段变量
    }
    public void run() {
        for (int i = 1; i <= 10; i++) {
            System.out.println(name + " Hello " + i);
            Thread.yield();                     //调用 yield()方法
        }
    }
}
public class TestBlockedState {
    public static void main(String argv[]) {    //主方法
        //创建类对象 h1 和 h2
        BlockedState h1 = new BlockedState("Thread1");
        BlockedState h2 = new BlockedState("Thread2");
        //启动线程 h1 和 h2
        h1.start();
        h2.start();
    }
}
```

运行 `TestBlockedState.java` 类，控制台窗口如图 5.11 所示。如果注释掉 `Thread.yield()` 这句代码，控制窗口如图 5.12 所示。

图 5.11 `TestBlockedState.java` 类运行结果图 5.12 注释掉 `Thread.yield()` 代码的运行结果

【代码解析】

- ❑ 在上述代码中，线程 `Thread1` 输出 1 后，接着切换到线程 `Thread2` 输出 1，然后继续重复上面的步骤输出 2~5。之所以会出现这样的结果，是因为当 `Thread1` 输出 1 后，调用 `yield()` 方法切换到 `Thread2` 线程，该线程输出 1 后又调用 `yield()` 方法切换到 `Thread1` 线程。这样轮换着输出了 1~5。
- ❑ 对于 CPU 来说，打印 1~5 是一个很短的过程，可能只花 0.01 秒的时间。在这个时间段内，对于 `Thread1` 对象，由于没有调用 `yield()` 方法，CPU 还来不及切换到 `Thread1` 对象，其输出过程就运行完成，所以会出现如图 5.12 所示的运行结果。

3. `join()` 方法

当处于 `runnable` 状态的线程为好几个线程时，如果线程 1 需要等待线程 2 完成某件事情后，其才能继续执行下去，这时就需要用到 `join()` 方法。该方法的基本格式如下：

```
Thread.join()
```

4. `wait()` 方法和 `notify()` 方法

`wait()` 方法与 `join()` 方法非常类似，对于 `join()` 方法，线程 1 需要等待线程 2 完成某件事情后才能继续执行下去，线程 2 完成就表示该线程运行完毕自然死亡；而对于 `wait()` 方法，只有线程 2 调用 `notify()` 方法来唤醒线程 1，其才能执行。这两个方法的具体内容在后面的介绍。


5.3.3 线程的结束状态

在 Java 语言中，如果想结束线程有两种方式：自然消亡和强制消亡。所谓自然消亡，是指一个线程从它的 `run()` 方法的结尾处返回，执行完后自然消亡；所谓强制消亡，是指调用 `Thread` 类的 `stop()` 方法强制线程结束，但是该方法已经过时不建议使用。

在具体编写多线程程序时，并发的功能需要放在 `run()` 方法中。因此当创建和启动一个

线程后，当该线程 run() 方法中的代码执行完毕，该线程就结束。如果程序员想控制线程的结束，循序条件不失是一个好的方法。

如果在继承 Thread 类的子类中没有覆盖 run() 方法，虽然创建和启动该线程时没有任何编译和运行错误，但是新线程一启动就会结束，这对多线程程序没有任何实际意义。

 **注意：**在单线程程序中，许多类中一般都会存在一个名为 main() 的方法，当运行该方法时就会创建和启动一个线程。

在 Java 程序中，线程还有前台和后台之分，前面章节创建的线程都为前台线程。对于运用程序来说，如果还有一个前台线程在运行，则这个应用程序就不会结束；如果是一个后台线程在运行，则该应用程序就会结束。

下面通过一个名为 EndThreadTest.java 的类来具体演示和讲解线程结束的现象，具体内容如代码 5.7 所示。

代码 5.7 线程的结束：ThreadEndTest.java


```
public class ThreadEndTest {
    public static void main(String[] args) {
        //输出当前线程的名字
        System.out.println("现在运行的线程为：" + Thread.currentThread().
            getName());
        new NewThread().start();           //启动一个新线程
    }
}

class NewThread extends Thread {         //继承 Thread 的类
    public void run() {
        int num = 5;                     //定义一个变量
        //循环
        while (num > 0) {
            System.out.println("现在运行的线程为：" + Thread.currentThread().
                getName());
            --num;
        }
    }
}
```

运行 ThreadEndTest.java 类，控制台窗口如图 5.13 所示。

【代码解析】

- ❑ 通过运行结果可以发现，上述代码创建和启动了两个线程：main 线程和 Thread 线程。对于 main() 线程，它实现了输出当前线程的名字及启动创建 Thread-0 线程后就结束；而对于 Thread-0 线程，则在运行完其 run() 后的循环才结束。由于 Thread-0 线程中实现了一个无限循环，所以 Terminate 按钮 (■) 一直处于运行状态。
- ❑ 在上述代码中如果想获取当前线程，可以通过 Thread.currentThread() 方法来实现，如果想获取线程的名称，可以通过 getName() 方法来实现。

 **注意：**对于 Thread 类的 getName() 方法，一般会获取所设置的线程的名字，如果该线程没有被 setName() 方法设置过，那么获取到的名称将是 Thread-X 的格式，X 的值从 0 开始，但是无法预测编号的大小。

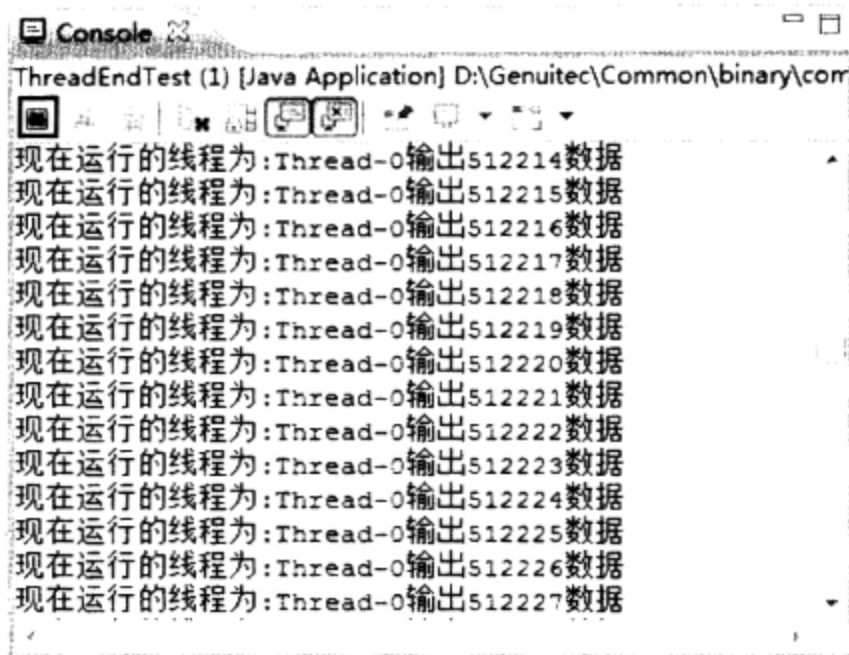


图 5.13 ThreadEndTest.java 类运行结果

对于 EndThreadTes 类, 如果想实现当 main 线程运行结束后, 程序就会运行结束的效果, 则需要把 Thread-0 线程设置为后台线程。修改 EndThreadTes 类的内容如代码 5.8 所示。

代码 5.8 前后台程序: ThreadEndTest.java

```
public class ThreadEndTest {
    public static void main(String[] args) {
        //输出当前线程的名字
        System.out.println("现在运行的线程为:" + Thread.currentThread().
            getName());
        Thread thread = new NewThread();           //创建一个新线程
        thread.setDaemon(true);                     //设置线程为后台线程
        thread.start();                             //启动线程
    }
}
class NewThread extends Thread {
    public void run() {                             //实现 run() 方法
        int num=0;                                 //定义一个变量
        //无限循环
        while (true) {
            ++num;
            System.out.println("现在运行的线程为:第"+num+"线程" );
        }
    }
}
```

运行 ThreadEndTest.java 类, 控制台窗口如图 5.14 所示。

【代码解析】

在上述代码中, main()方法输出当前线程的名字并创建、启动新线程后, main()线程就会结束。在这种情况下, 虽然还有一个无限循环新线程 Thread-0 在运行, 但是该线程被设置为后台线程, 仍然会随着 main()线程的结束而终止运行。

注意: 如果想设置某个线程为后台线程, 则必须在该线程启动之前 (调用 start()方法前) 调用 setDaemon(true)方法。

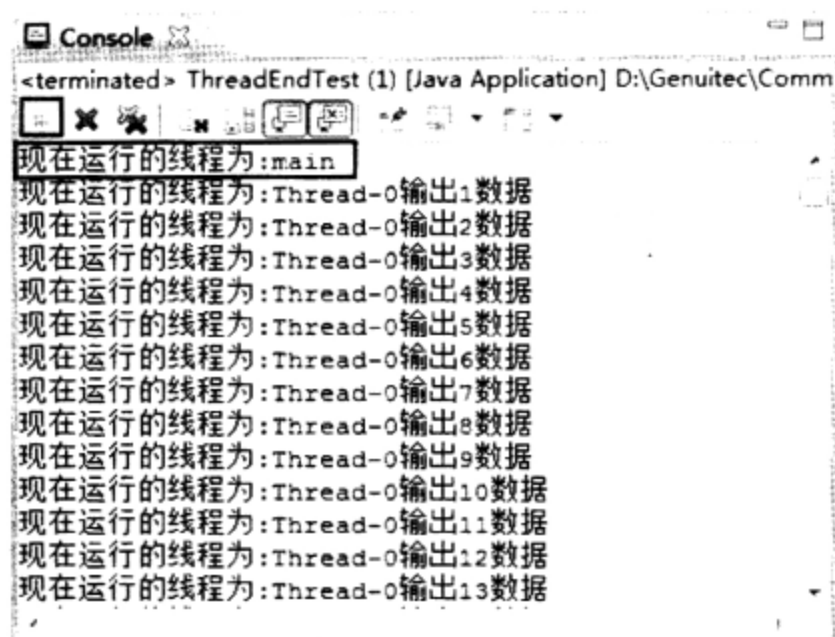


图 5.14 ThreadEndTest.java 类运行结果

5.4 小 结

本章主要通过线程，模拟现实生活中妈妈做饭时发现酱油没了，而让儿子买酱油的场景。这些场景分别为没有通过线程的 join() 方法模拟的“做饭场景”，以及通过线程的 join() 方法模拟的“做饭场景”。本章的最后还详细介绍了线程状态的一些基础知识，通过这些知识的学习，可以更好地实现多线程程序。

第 6 章 火车站售票系统（线程安全知识）

在 Java 语言中，通常要考虑程序线程的安全性，特别是多线程的应用程序。对于程序员来说，程序代码的共用没有什么太大的问题，但是数据的共用则会出现很多问题。这是因为程序代码写好后，在运行过程中是无法改变的，但是数据则相反。本章除了通过火车站售票系统来详细介绍如何实现多线程应用程序的线程安全之外，还将详细介绍线程安全的一些基本知识。

本章的学习目标如下：

- ❑ 掌握火车站售票系统；
- ❑ 理解为什么要使线程具有同步性；
- ❑ 掌握实现线程同步的两种方式。

6.1 火车站售票系统原理

火车站售票系统用来模拟火车票的销售过程，即多个人并发买票的过程。由于在该系统中存在共享资源——火车票，所以需要保证程序线程的安全性。

6.1.1 项目结构框架分析

对于火车站售票系统项目，根据面向对象的思想，需要创建一个对象火车票。该项目中的两个包分别描述两种情况，即没实现线程安全和实现线程安全。

火车站售票系统项目目录如图 6.1 所示，各个目录的功能如下。

- ❑ com.cjg.nosafe 包：模拟没有实现线程安全的火车站售票系统。
- ❑ com.cjg.safe 包：模拟实现线程安全的火车站售票系统。

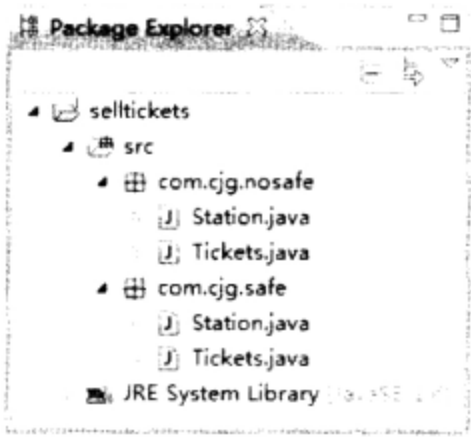


图 6.1 项目目录

6.1.2 项目功能业务分析

本节将以直观的方式向读者介绍整个项目要实现的功能。这些功能包括线程不安全火车站售票系统和线程安全火车站售票系统。

1. 线程不安全火车站售票系统

许多顾客并发到各个售票台买票，各个售票台不仅所卖的票属于共享资源，而且每卖一张票需要 5 分钟的过程。由于售票台在具体卖票过程中没有实现线程的安全，所以出现了卖 0 和负数（0、-1 和-2）的票，具体过程如图 6.2 所示。在图中，虽然 4 个顾客到 4 个售票台（Thread-0、Thread-1、Thread-3 和 Thread-4）买票，但是售票台 Thread-0 卖给顾客第 0 张票，售票台 Thread-2 卖给顾客第-1 张票，售票台 Thread-1 卖给顾客第-2 张票，显然该火车站售票系统不能正常运行。

2. 线程安全火车站售票系统

许多顾客并发到各个售票台买票，各个售票台不仅所卖的票属于共享资源，而且每卖一张票需要 5 分钟的过程。由于售票台在具体的卖票过程中实现了线程安全，所以没有出现不正常的运行情况，具体过程如图 6.3 所示。在图 6.3 中，尽管 4 个顾客并发到 4 个售票台（Thread-0、Thread-3、Thread-3 和 Thread-4）买票，但是每个顾客都拿到了属于共享资源中的一张票。虽然每个售票台都经过 5 分钟卖了属于共享资源中的一张票，但是由于实现了线程的安全，所以没有出现卖重复票和负数票的情况，即售票台 Thread-0 卖出第 4 张票，售票台 Thread-1 卖出第 1 张票，售票台 Thread-2 卖出第 3 张票，售票台 Thread-3 卖出第 2 张票。

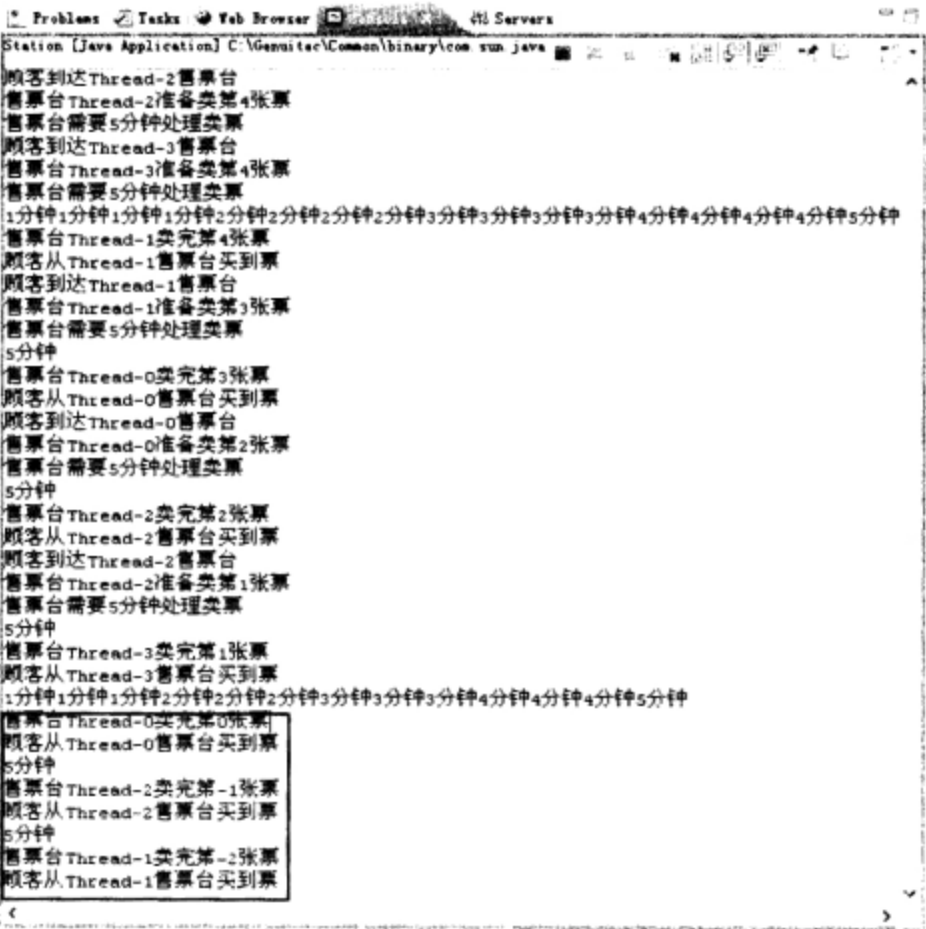


图 6.2 线程不安全运行结果



图 6.3 线程安全运行结果

6.2 没有实现线程安全的火车票售票系统

本章通过线程技术来模拟没有实现线程安全的火车票售票系统,具体程序架构如图 6.4 所示,它包含一个类对象 Tickets.java 和一个进行测试的类 Station.java。

6.2.1 火车票的类

Tickets.java 类用来模拟火车票,由于 Tickets 主要被售票台用来卖票给顾客,所以该类的 run()方法中实现了卖票给顾客的过程,具体内容如代码 6.1 所示,该类的 UML 如图 6.5 所示。

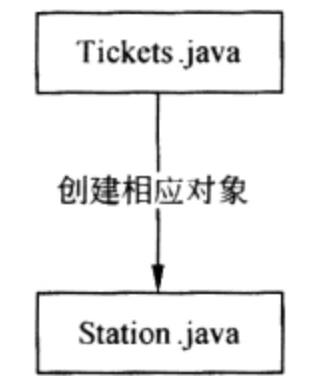


图 6.4 程序关系图

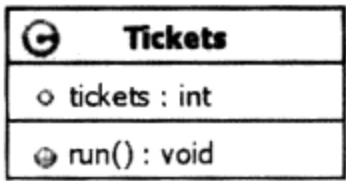


图 6.5 火车票类图

代码 6.1 火车票类: Tickets.java

```
public class Tickets implements Runnable { //实现接口 Runnable
    int tickets = 4; //定义火车票数目的变量
    @Override
    public void run() { //实现 run() 方法
        while (true) { //无限循环
            if (tickets > 0) {
                //输出相应的信息, 为卖票做准备
                System.out.println("顾客到达" + Thread.currentThread().
                    getName()+ "售票台");
                System.out.println("售票台" + Thread.currentThread().
                    getName()+ "准备卖第" + tickets + "张票");
                try { //具体卖票的过程
                    System.out.println("售票台需要 5 分钟处理卖票");
                    for (int i = 1; i <= 5; i++) {
                        Thread.sleep(5);
                        System.out.print(i + "分钟");
                    }
                }
                //输出相应的信息, 卖票成功
                System.out.println("\n 售票台"
                    + Thread.currentThread().getName() + "卖完第"
                    + tickets + "张票");
                System.out.println("顾客从" + Thread.currentThread().
                    getName()+ "售票台买到票");
            }
        }
    }
}
```

```
--tickets;
    } catch (InterruptedException e) {
        System.err.println("卖票不成功");
        System.exit(1);
    }
}
}
```

【代码解析】

上述代码主要是讲解线程的安全机制，所以卖票的具体过程只是通过输出 5 个数据来提示一个。

6.2.2 售票台的类

Station.java 类用来模拟现实生活中的售票台，主要作用是接待顾客买票。具体内容如代码 6.2 所示。

代码 6.2 售票台类: Station.java

```
public class Station {
    public static void main(String[] args) {
        Tickets tickets = new Tickets(); //创建火车票对象
        //创建 4 个线程对象，即顾客对象
        Thread static1 = new Thread(tickets);
        Thread static2 = new Thread(tickets);
        Thread static3 = new Thread(tickets);
        Thread static4 = new Thread(tickets);
        static1.start(); //启动 4 个线程
        static2.start();
        static3.start();
        static4.start();
    }
}
```

【代码解析】

上述代码中，首先创建了火车票对象和 4 个线程对象（顾客），然后通过 Thread 类的 start()方法启动线程。

6.2.3 实现线程安全的火车票售票系统

由于实现线程安全火车票售票系统，与没有实现线程安全的火车票售票系统只有类 `Tickets.java` 中的代码不同，所以本节只讲解火车票的类。

为了让火车票销售系统得到预期的运行结果，必须在该系统中实现线程的安全。因此修改 Tickets.java 中的内容如代码 6.3 所示。

代码 6.3 火车票类: Tickets.java

```
public class Tickets implements Runnable {
```

```

int tickets = 4;
public void run() {
    while (true) {
        synchronized (this) {           //创建同步块
            if (tickets > 0) {
                //输出相应信息
                System.out.println("顾客到达"
                    + Thread.currentThread().getName() + "售票台");
                //输出相应信息
                System.out.println("售票台" + Thread.currentThread().
                    getName()+ "准备卖第" + tickets + "张票");
                try {
                    //输出相应的信息
                    System.out.println("售票台需要 5 分钟处理卖票");
                    //通过循环遍历
                    for (int i = 1; i <= 5; i++) {
                        Thread.sleep(5);
                        System.out.print(i + "分钟");
                    }
                    //输出相应的信息
                    System.out.println("\n 售票台"
                        + Thread.currentThread().getName() + "卖完第"
                        + tickets + "张票");
                    //输出相应的信息
                    System.out.println("顾客从"
                        + Thread.currentThread().getName() + "售票
                        台买到票");
                    --tickets;
                } catch (InterruptedException e) {
                    System.err.println("卖票不成功");
                    System.exit(1);           //退出系统
                }
            }
        }
    }
}

```

【代码解析】

上述代码主要是对共享资源进行了相应设置，即把共享资源的具体内容放在了同步块的 `synchronized()` 语句中。

除了可以使用同步块的方式来实现线程的安全之外，还可以通过同步方法来实现。利用同步方法方式实现火车票类的具体内容如代码 6.4 所示。

代码 6.4 火车票类: Water.java

```

public class Tickets implements Runnable {
    int tickets = 4;           //创建成员变量
    public void run() {
        sellticket();          //调用同步方法
    }
    public synchronized void sellticket() {           //定义了同步方法
        int tickets = this.tickets;                   //定义方法变量
        while (true) {
            if (tickets > 0) {

```



```

//输出相应的信息
System.out.println("顾客到达" + Thread.currentThread().
getName() + "售票台");
//输出相应的信息
System.out.println("售票台" + Thread.currentThread().
getName() + "准备卖第" + tickets + "张票");
try {
    //输出相应的信息
    System.out.println("售票台需要 5 分钟处理卖票");
    for (int i = 1; i <= 5; i++) {
        Thread.sleep(5);
        System.out.print(i + "分钟");
    }
    //输出相应的信息
    System.out.println("\n 售票台"
        + Thread.currentThread().getName() + "卖完第"
        + tickets + "张票");
    //输出相应的信息
    System.out.println("顾客从" + Thread.currentThread().
getName() + "售票台买到票");
    --tickets;
} catch (InterruptedException e) {
    System.err.println("买票不成功");
    System.exit(1); //退出系统
}
}
}
}
}
}

```

【代码解析】

上述代码首先定义一个实现卖票功能的同步方法 `sellticket()`，然后在 `run()` 方法中调用该方法。

6.3 知识点扩展——线程的同步知识

虽然多线程程序可以实现一些特殊的功能，但是多线程程序执行时，不像单线程程序那样简单。这是因为单线程程序中只有一个线程执行，不需要考虑此线程被其他线程打扰；而多线程在执行时，如果多个线程共同访问一个资源，则需要控制，否则容易出现问題。

6.3.1 为什么要使用同步机制

在 Java 语言中存在许多共享资源，例如数据、文件、输入/输出端口或打印机。所谓同步机制就是指应用程序在操作共享资源时，应该保持共享资源的统一性和整体性。为了实现同步机制，出现了加锁和解锁的概念，即当一个线程想要使用共享资源时，需要给该共享资源加锁；当该线程使用完共享资源后，必须对其进行解锁。当一个线程对某个资源加锁后，其他线程是不能访问该资源的，只能等该资源被上锁线程解锁后其他线程才能

访问。

为什么要使共享资源具有整体性和统一性呢？为了讲清楚该问题，下面通过一个具体的实例来讲解，具体步骤如下。

(1) 创建一个共享资源的类 ShareData.java，具体内容如代码 6.5 所示。

代码 6.5 共享资源：ShareData.java

```
public class ShareData implements Runnable {
    int i;                                //定义一个共享属性数据
    public void run() {                   //实现 run() 方法
        while (i < 10) {                 //遍历数据
            i++;
            try {
                Thread.sleep(6000);      //线程休眠 6 秒
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("线程" + Thread.currentThread().getName() +
                "打印数据" + i);
        }
    }
}
```

【代码解析】

在上述代码中，共享的资源为数据 i 属性。ShareData 类实现了接口 Runnable 中的 run() 方法，在该方法中通过循环遍历输出 i 的值。

(2) 创建一个多线程类 TestSyn.java，具体内容如代码 6.6 所示。

代码 6.6 测试共享数据：TestSyn.java

```
public class TestSyn {
    public static void main(String argv[]) {
        ShareData s = new ShareData();    //创建一个共享数据
        Thread t1 = new Thread(s);        //创建线程对象 t1
        Thread t2 = new Thread(s);        //创建线程对象 t2
        //启动线程对象 t1 和 t2
        t1.start();
        t2.start();
    }
}
```

运行 TestSyn.java 类，控制台窗口如图 6.6 所示。

【代码解析】

- ❑ 在上述代码中，首先创建了一个共享资源对象 s 和两个线程对象 t1、t2，接着启动两个线程。
- ❑ 当两个线程启动后，会访问一个属性 i。预期的效果应该是两个线程对象轮流将属性 i 的值加 1，然后显示在控制台窗口里。可是运行结果却如图 6.6 所示。
- ❑ 当查看控制窗口时，可以发现没有打印数据 1。之所以会这样，因为 CPU 首先会调用线程 Thread-0，可是该线程把数据加 1 后却没有打印相应信息。这时 CPU 就切换到



图 6.6 非同步运行结果

线程 Thread-1，该线程也把数据加 1 后，如果没有输出 1 就直接输出数据 2。

- ❑ 查看控制窗口，可以发现打印了两个数据 10。之所以会这样，因为线程 Thread-0 把数据加 1 后，CPU 就切换到运行了“已经把数据加 1 后”的线程 Thread-1，这时线程 Thread-1 一气呵成地输出数据 10。由于线程 Thread-1 运行完了 run() 方法，CPU 再次切换到线程 Thread-0 上，所以该线程也会输出数据 10。

上述代码的运行结果之所以没有出现预想的结果，是因为共享数据 i 加 1 的代码和打印数据 i 的代码没有封装成块。即数据 i 加 1 之后，应该要打印出来，之后才能被切换其他线程或再次重复把数据 i 加 1 之后打印出来的过程。

6.3.2 Synchronized 的同步块

为了保持共享资源的整体性和统一性，即保持共享资源的同步或原子性，Java 语言提供了两种形式的方法：同步块和同步方法。所谓同步块，是指设置程序中的某个代码块为同步区域；所谓同步方法，是指设置程序中的某个方法为同步方法。

本节将详细讲解同步块的原理，同步块的语法格式如下：

```
synchronized(object){
    代码块
}
```

为了使 6.3.1 节的程序输出预想的效果，必须使类 ShareData 中 run() 方法的代码具有原子性形成同步块。修改 ShareData 类的内容如代码 6.7 所示。

代码 6.7 同步块：ShareData.java

```
public class ShareData implements Runnable {
    int i;
    String str = new String("");           //创建一个字符串对象
    public void run() {
        synchronized (str) {               //创建同步块
            while (i < 10) {
                i++;
                try {
                    Thread.sleep(6000);      //线程休眠 6 秒
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                //输出相应信息
                System.out.println("线程" + Thread.currentThread().
                    getName()
                    + "打印数据" + i);
            }
        }
    }
}
```

【代码解析】

- ❑ 在上述代码中，将具有原子性的代码放在了 synchronized 语句内，形成了同步代码块。在这种情况下，同一时刻只能有一个线程可以进入同步块内运行，只有当该线程离开同步块后，其他线程才能进入同步块代码里运行。

□ 在上述代码中，通过“`String str= new String("");`”语句随机产生了一个对象，用在了后面的同步块语句中作为“监视器”。

当修改 `ShareData` 类后，再次运行代码 `TestSyn.java`，其运行结果如图 6.7 所示，与图 6.6 所示的非同步运行结果相比，线程 `Thread-0` 和 `Thread-1` 会交替地打印出数据 1~10。在打印的过程中，不会出现某个数据的消失、重复打印和超出范围数据的现象。



图 6.7 同步运行结果

那么在计算机中是如何执行 `synchronized` 关键字的呢？

在内存中存储的任何类型的对象都拥有一个标记位，该标记位的数值只能是 0 或 1，开始状态为 1。当一个对象作为 `synchronized(object)` 中的 `object` 对象时，一个线程执行该语句就会使 `object` 对象的标记位由 1 变成 0 数值，直到执行完整个同步块中的代码，才会使 `object` 对象的标记位由 0 变成 1 数值。

当一个线程在执行到 `synchronized(object)` 语句时，首先会检查 `object` 对象的标记位。如果为 0 数值，表明该同步块正被其他线程在执行，该线程就会被暂阻塞（blocked）。然后放到阻塞池（blocked pool）中，直到另外的线程执行完同步块代码，将 `object` 对象的标记位从 0 数值恢复到 1 数值，才会取消阻塞池（blocked pool）中一个线程的阻塞。具体过程如图 6.8 所示。

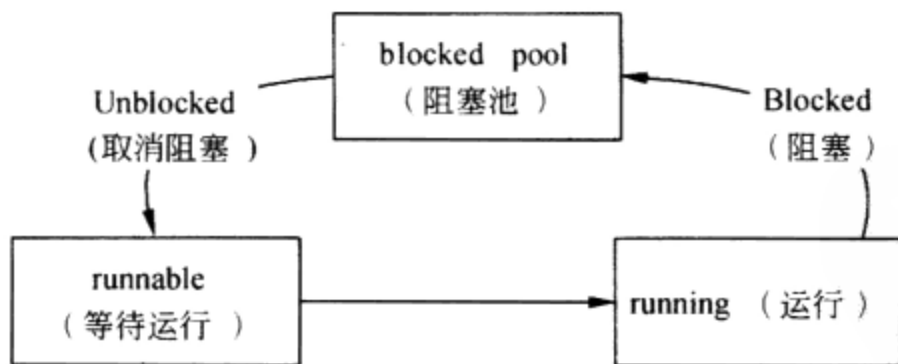


图 6.8 状态的转换

当一个线程在执行到 `synchronized(object)` 语句时，如果检查到 `object` 的标记位为 1 数值，线程就会在执行同步块中代码的同时设置 `object` 的标记位为 0，以防止其他线程再进入有关的同步代码块中。

⚠注意：当一个处于阻塞状态的线程被取消阻塞后，是处于 `Runnable` 状态，而不是 `Running` 状态。

当多个线程因等待同一个对象的标记位而处于 blocked pool 中时，如果对象的标记位恢复到 1 数值时，只会有一个线程能够处于 runnable 状态，其他线程仍然处于 blocked pool 中。

不仅同一个代码块在多个线程间可以实现同步，若干个不同的代码块也可以实现相互之间的同步，只要各 synchronized(object) 语句中的 object 为同一个对象就可以。

6.3.3 Synchronized 的同步方法

在 Java 语言中除了可以使用同步块外，还可以使用同步方法来保持共享资源的整体性和统一性。既然同步块是对代码块进行同步的，那么同步函数就是对函数进行同步。本节将详细地讲解同步方法。

如果想定义同步函数，只需要在同步函数定义前加上关键字 synchronized 就可以，其语法格式如下：

```
synchronized void 方法名() {
    方法体
}
```

下面通过一些具体的实例来讲解同步方法，具体步骤如下。

(1) 如果想使 ShareData 类得到预想的结果，除了可以使用同步块外，还可以通过同步方法来实现，具体内容如代码 6.8 所示。

代码 6.8 同步方法：ShareData.java

```
public class ShareData implements Runnable {
    int i;
    public void run() {                //实现 run() 方法
        test();                        //调用同步方法
    }
    synchronized void test() {        //同步方法
        while (i < 10) {
            i++;
            try {
                Thread.sleep(5);        //实现线程休眠
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("线程" + Thread.currentThread().getName() +
                "打印数据"+ i);
        }
    }
}
```

【代码解析】

在上述代码中，首先把原来 ShareData.run() 方法中的共享资源抽离出来，作为方法 test() 的方法体，接着通过 synchronized 关键字使该方法称为同步方法，最后也实现了线程间的同步。当修改类 ShareData 后，再次运行代码 TestSyn.java，同样也会出现图 6.6 所示的运行结果。

(2) 既然若干个不同的代码块可以实现相互之间的同步，那么若干个不同的方法也可以实现相互同步，只要各个方法通过 synchronized 关键字修饰成为同步方法。如果再往深

一点思考，同步块和同步方法也可以实现相互同步吗？答案是肯定的，这是因为只要同步块的监视器和同步方法的监视器为同一个对象就可以。

下面编写一个具体的实例 `ShareData.java`，来演示同步块和同步方法之间如何实现相互同步，具体内容如代码 6.9 所示。

代码 6.9 同步块和方法相互同步：ShareData.java

```
public class ShareData implements Runnable {
    //定义两个变量
    int i;
    String str = new String("");
    public void run() {
        if (str.equals("cjgong")) {    //根据字符串决定调用同步块还是同步方法
            test();
        } else {
            synchronized (str) {      //定义了一个同步块
                while (i < 10) {
                    i++;
                    try {
                        Thread.sleep(6000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    System.out.println("线程" + Thread.currentThread().
                        getName() + "打印数据" + i);
                }
            }
        }
    }
    synchronized void test() {        //定义同步方法
        while (i < 10) {
            i++;
            try {
                Thread.sleep(5);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("线程" + Thread.currentThread().getName() +
                "打印数据"+ i);
        }
    }
}
```

【代码解析】

在上述代码中首先把共同资源的代码抽离出来，作为共同方法 `test()` 的方法体，然后通过字符串 `str` 变量的值来决定调用的是同步方法还是同步块。

(3) 接着编写一个测试类 `TestSyn.java`，具体内容如代码 6.10 所示。

代码 6.10 测试类：TestSyn.java

```
public class TestSyn {
    public static void main(String argv[]) {
        ShareData s = new ShareData();    //创建共享数据资源
        //创建两个线程对象
        Thread t1 = new Thread(s);
```



```

Thread t2 = new Thread(s);
t1.start();                                     //启动线程对象 t1
try {
    Thread.sleep(5);                             //线程休眠 5
} catch (InterruptedException e) {
    e.printStackTrace();
}
s.str = "cjgong";                               //设置对象 s 的值
t2.start();                                     //启动线程对象 t2
}
}

```

运行 TestSyn.java 类，控制台窗口如图 6.9 所示。



图 6.9 非同步运行结果

【代码解析】

在上述代码中首先产生两个线程 t1 和 t2，接着启动线程 t1。t1 线程启动的 run() 方法会检查到 str 变量的取值不等于 cjgong，就调用程序中的同步代码块来运行。最后将 str 变量的值修改成 cjgong，再启动第二个线程 t2，这样 t2 线程的 run() 方法就会调用同步方法。

(4) 通过图 6.9 所示的运行结果可以发现，同步块和同步方法没有实现相互同步，这主要是因为同步块以 str 作为监视器，而同步方法肯定没有以 str 作为监视器。

那么同步方法到底以什么对象作为监视器呢？通过面向对象的语法可以知道，类中的静态方法始终能访问到的一个对象就是自己本身，所以同步方法只能以包含自己的类本身作为监视器，即 this。修改 ShareData.java 代码的具体内容如代码 6.11 所示。

代码 6.11 同步方法：ShareData.java

```

public class ShareData implements Runnable {
    //定义两个变量
    int i;
    String str = new String("");
    public void run() {
        if (str.equals("cjgong")) {           //根据字符串决定调用同步块还是同步方法
            test();
        } else {
            synchronized (this) {           //定义一个同步块
                while (i < 10) {
                    i++;
                    try {

```

```

        Thread.sleep(6000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("线程" + Thread.currentThread().
        getName()+ "打印数据" + i);
    }
}
}
}
synchronized void test() {           //定义同步方法
    while (i < 10) {
        i++;
        try {
            Thread.sleep(5);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("线程" + Thread.currentThread().getName() +
            "打印数据"+ i);
    }
}
}
}

```

【代码解析】

在上述代码中，与原来的代码相比较，只修改同步块的监视器为 `this` 对象。这时如果再次运行 `TestSyn.java`，控制台窗口如图 6.10 所示。该运行结果不仅验证了同步块可以与同步方法实现相互同步，而且还验证了同步方法的监视器为 `this` 对象。


 **注意：**对于共享的数据，应该是类的 `private` 数据成员，这样可以禁止来自类外的随意访问来破坏数据的一致性。



图 6.10 同步运行结果

6.3.4 死锁的问题

在开发多线程程序时，很容易就会遇到死锁的问题。死锁对于程序员来说是一种很难调试的错误。所谓死锁，是指两个线程对两个同步对象具有循环依赖，即线程 A 在等待线程 B，而线程 B 等待线程 A。

在现实生活中，经常会遇到死锁的情况。例如，一个美国人和一个中国人同时在吃饭，

美国人拿到了一根筷子和一把刀子，而中国人却拿到了一把叉子和一根筷子。这时就会发生如下的对话。

美国人：“你给我叉子，我才给你筷子”。

中国人：“你给我筷子，我才给你叉子”。

结果可想而知，美国人和中国人谁也没吃到饭。下面将通过程序来模拟上述场景，具体步骤如下。

(1) 创建一个中国人的类 China.java，具体内容如代码 6.12 所示。

代码 6.12 中国人的类：China.java

```
public class China {
    synchronized void cTake(American ameri) {                //同步方法
        String name = Thread.currentThread().getName(); //获取当前线程的名字
        //输出相应的信息
        System.out.println(name + "有筷子和叉子");
        System.out.println(name + "先拿到筷子，才放弃叉子");
        try {
            Thread.sleep(5);                                //线程休眠
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        //输出相应的信息
        System.out.println(name + "开始拿筷子");
        ameri.atoc();                                       //调用 ameri 的 atoc() 方法
        System.out.println(name + "拿到筷子");
    }
    synchronized void cToA() {                               //编写把叉子给美国人的方法
        System.out.println("中国人放弃叉子给了美国人");
    }
}
```

(2) 创建一个美国人的类 American.java，具体内容如代码 6.13 所示。

代码 6.13 美国人的类：American.java

```
public class American {
    synchronized void aTake(China china) {                //同步方法
        String name = Thread.currentThread().getName(); //输出相应信息
        //输出相应信息
        System.out.println(name + "有筷子和刀子");
        System.out.println(name + "先拿到叉子，才放弃筷子");
        try {
            Thread.sleep(5);                                //线程休眠
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        //输出相应信息
        System.out.println(name + "开始拿叉子");
        china.cToA();                                       //调用 china 的 ctoA() 方法
        System.out.println(name + "拿到叉子");
    }
    synchronized void atoc() {                               //编写把筷子给中国人的方法
```

```

        System.out.println("美国人放弃筷子给了中国人");
    }
}

```

(3) 创建一个模拟交谈的类 TakeTest, 具体内容如代码 6.14 所示。

代码 6.14 交谈类: TakeTest.java

```

public class TakeTest implements Runnable {
    //创建相应的线程
    China a = new China();           //中国人对象
    American b = new American();      //美国人对象
    TakeTest() {                      //构造函数
        Thread.currentThread().setName("中国人"); //设置当前线程的名字
        new Thread(this).start();      //调用 run() 方法
        a.cTake(b);                   //调用 cTake() 方法
        System.out.println("返回主线程"); //输出相应的信息
    }
    public static void main(String[] args) {
        new TakeTest();
    }
    @Override
    public void run() {
        Thread.currentThread().setName("美国人"); //设置线程的名字
        b.aTake(a);                               //调用 aTake() 方法
        System.out.println("返回其他线程");       //输出相应信息
    }
}

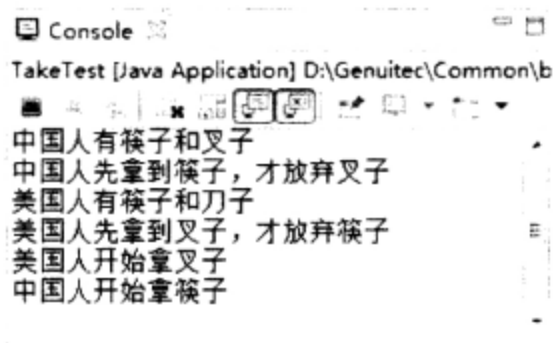
```

运行 TakeTest.java 类, 控制台窗口如图 6.11 所示。

【代码解析】

从运行结果可以看出, “中国人”线程进入了 a 的监视器, 然后又在等待 b 的监视器。同时“美国人”线程进入了 b 的监视器, 然后又在等待 a 的监视器。

死锁的处理是操作系统设计时一个非常重要的主题, 一些操作系统的书籍中会很详细地介绍, 所以这里不再详细讲解。



```

Console
TakeTest [Java Application] D:\Genuitec\Common\b
中国人有筷子和叉子
中国人先拿到筷子, 才放弃叉子
美国人有筷子和刀子
美国人先拿到叉子, 才放弃筷子
美国人开始拿叉子
中国人开始拿筷子

```

图 6.11 运行结果

6.4 小 结

本章主要通过线程实现火车站售票系统, 为了能够让读者知道线程安全的重要性, 分别讲解了没有实现线程安全的火车站售票系统和实现了线程安全的火车站售票系统。本章的最后还详细介绍了实现线程安全的各种方式和原理, 通过这些知识的学习, 可以让读者创建的多线程程序更安全。

第7章 生产者与消费者问题(线程通信知识)

在 Java 语言中，经常会遇到这样的情况：线程 A 正在等待另一个线程 B 把数据送过来，数据还没有送到之前，线程 A 先进入等待状态，等到数据送到时，线程 B 再通知线程 A 可以处理数据。这就是操作系统中比较有名的生产者与消费者问题（producer and consumer）。本章除了通过模拟生产者与消费者问题来详细介绍如何实现线程的通信外，还将详细介绍线程通信的一些基本知识。

本章的学习目标如下：

- ❑ 掌握生产者与消费者问题；
- ❑ 掌握线程通信的各种方法。

7.1 生产者与消费者原理

在操作系统中有一个比较著名的“生产者和消费者”问题，即生产者生产产品时，不可能无限制地生产下去，因为库存量有限制，所以当还未到一定的库存量时，生产者会继续生产，如果库存够了，就等待消费者来用掉库存。对于消费者只要有库存，才会进行消费，如果库存用完了，他们就会等待，直到生产者又生产了新的产品。

7.1.1 项目结构框架分析

对于生产者与消费者项目，根据面向对象的思想，需要创建 3 个对象，分别为生产者、消费者和储存库。对于该项目中的两个包，分别描述没有实现线程通信和实现线程通信的两种情况。

生产者与消费者项目目录如图 7.1 所示，各个目录的功能如下。

- ❑ `com.cjg.nocommunicate` 包：模拟没有实现线程通信的生产者与消费者项目。
- ❑ `com.cjg.communicate` 包：模拟实现线程通信的生产者与消费者项目。

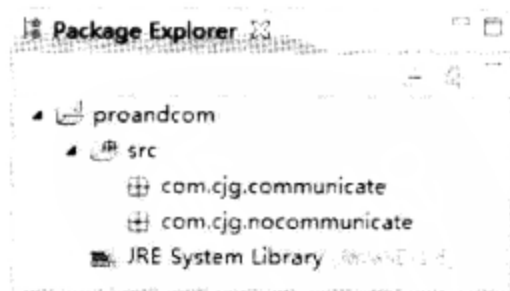


图 7.1 项目目录

7.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。即在生产者与消费者项目中有一个数据存储空间，该空间被分成两部分，其中一部分用来存储学生的名字，另一部分用来存储学生

的性别。数据存储空间会被两个线程来访问，一个线程（生产者）用来向存储空间中添加学生的相关信息，另一个线程（消费者）从存储空间中取出学生的相关信息。在具体编写时包含了无线程通信的生产者与消费者项目和实现线程通信的生产者与消费者项目。

1. 无线程通信的生产者与消费者

生产者通过循环不断地向储存库里添加学生信息，单数循环添加“小明和男”双数循环添加“小红和女”。而消费者也通过循环不断地输出学生信息。具体过程如图 7.2 所示。在图 7.2 中，虽然避免了这种情况：“生产者”在向存储空间添加一个学生的信息时，刚添加完学生的姓名还没来得及添加性别时，CPU 就切换到了“消费者”线程，“消费者”线程就会将这个学生的姓名和上一个学生的性别输出。但是却没有避免这种情况：“消费者”刚输出学生（小红和女）的信息后，还没等到“生产者”输入学生（小明和男）的信息，又重复输出已经输出的学生信息。

2. 实现线程通信的生产者与消费者项目

生产者通过循环不断地向储存库里添加用户信息，单数循环添加“小明和男”双数循环添加“小红和女”。而消费者也通过循环不断地把用户名和密码输出，具体过程如图 7.3 所示。在图 7.3 中，真正实现了当生产者每输入一个学生的信息，消费者就取出并打印出该学生的信息。

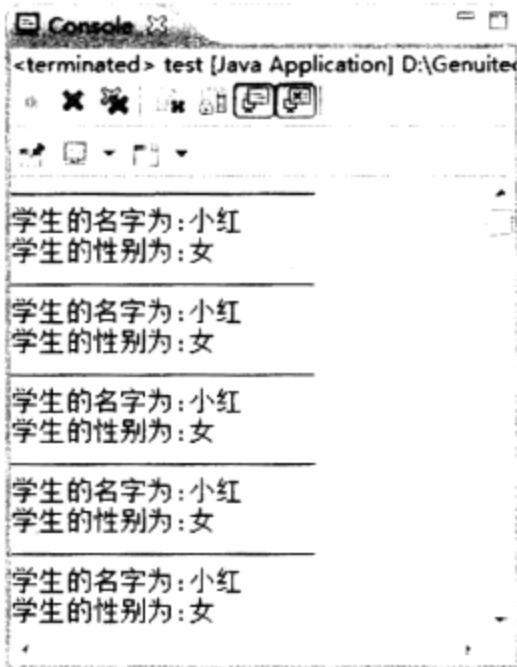


图 7.2 无线程通信运行结果

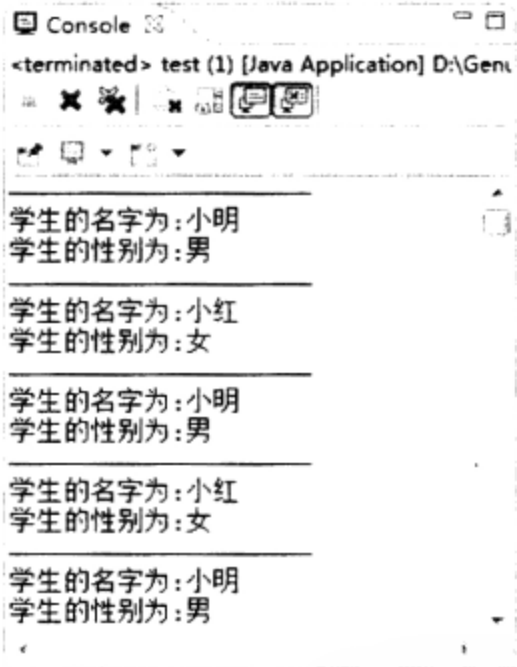


图 7.3 实现线程通信运行结果

7.2 无线程通信的生产者与消费者项目

本章通过线程技术来模拟无线程通信的“生产者与消费者”项目，具体程序架构如图 7.4 所示，它包含 Comsumer.java、Producer.java 和 StoreHouse.java 3 个类对象及一个进行测试的类 Test.java。

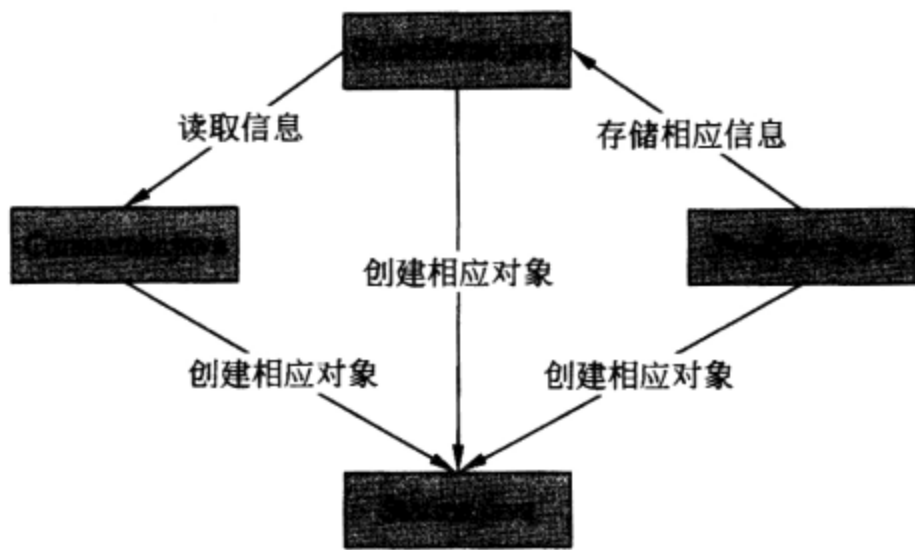


图 7.4 程序关系图

7.2.1 生产者类

Producer.java 类用来模拟生产者，该类主要用来实现向类 StoreHouse 中添加相应的用户信息。Producer.java 类的具体内容如代码 7.1 所示，该类的 UML 如图 7.5 所示。

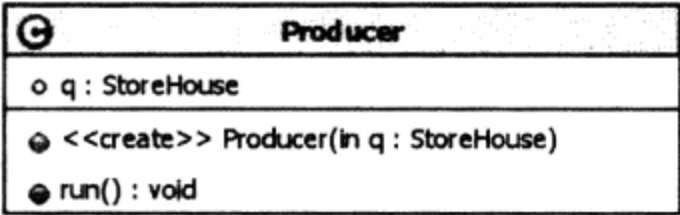


图 7.5 生产者类图

代码 7.1 生产者类：Producer.java

```
public class Producer implements Runnable {
    StoreHouse q; //存储库对象
    public Producer(StoreHouse q) { //构造函数
        this.q = q;
    }
    public void run() { //实现 run() 方法
        int i = 0; //设置变量 i
        while (true) {
            synchronized (q) { //同步块
                if (i == 0) { //当为偶数时
                    q.name = "小明";
                    try {
                        Thread.sleep(2);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    q.sex = "男";
                } else { //当为奇数
                    q.name = "小红";
                    q.sex = "女";
                }
                i = (i + 1) % 2;
            }
        }
    }
}
```

```
    }  
    }  
}
```

【代码解析】

上述代码在 `run()`方法中通过无限循环方法，不断地把学生的信息存储到存储库 `q` 中。

7.2.2 消费者类

`Comsumer.java` 类用来模拟生产者，该类主要用来实现从类 `StoreHouse` 中取出相应的用户信息并且输出。`Comsumer.java` 类的具体内容如代码 7.2 所示，该类的 UML 如图 7.6 所示。

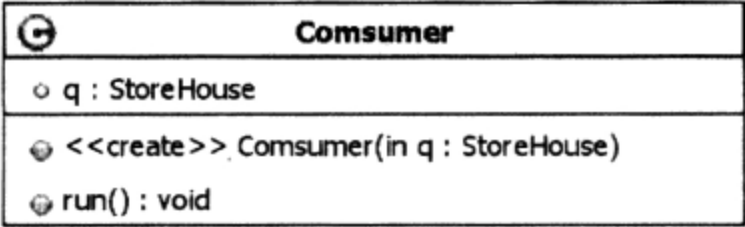


图 7.6 消费者类图

代码 7.2 消费者类: `Comsumer.java`

```
public class Comsumer implements Runnable {  
    StoreHouse q;                                //存储库对象  
    public Comsumer(StoreHouse q) {               //构造函数  
        this.q = q;  
    }  
    public void run() {                            //实现 run() 方法  
        while (true) {  
            synchronized (q) {                    //同步块  
                System.out.println("—————");  
                //输出相应的信息  
                System.out.println("学生的名字为:" + q.name);  
                System.out.println("学生的性别为:" + q.sex);  
            }  
        }  
    }  
}
```

【代码解析】

上述代码在 `run()`方法中通过无限循环方法，不断地从存储库 `q` 中获取相应的信息并输出。

7.2.3 储存库类

`StoreHouse.java` 类用来模拟储存库，该类主要用来实现从 `Producer.java` 类输入的相应信息。`StoreHouse.java` 类的具体内容如代码 7.3 所示，该类的 UML 如图 7.7 所示。

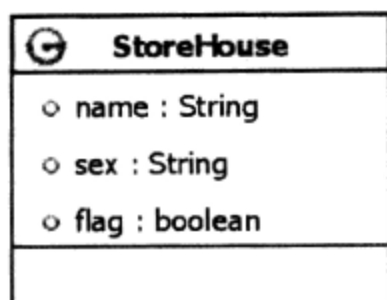


图 7.7 储存库类图

代码 7.3 储存库类：StoreHouse.java

```
public class StoreHouse {
    String name = "no";           //姓名的成员变量
    String sex = "no";           //性别的成员变量
}
```

【代码解析】

在上述代码中编写一个数据存储库对象，该对象通过两个成员变量 `name` 和 `sex` 模拟数据存储库被分成两部分，其中一部分用来存储学生的名字（`name` 变量），另一部分用来存储学生的性别（`sex` 变量）。

7.2.4 测试类

为了查看生产者与消费者项目是否能够出现预期的运行效果，编写一个名为 `Test.java` 测试该项目的类，具体内容如代码 7.4 所示。

代码 7.4 测试类：Test.java

```
public class test {
    public static void main(String[] args) {
        StoreHouse q = new StoreHouse();           //创建了储存库对象
        new Thread(new Producer(q)).start();       //创建并启动生产者线程
        try {
            Thread.sleep(5);                       //线程休眠
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        new Thread(new Consumer(q)).start();       //创建并启动消费者线程
    }
}
```

【代码解析】

在上述代码中，首先创建了一个储存库的对象 `q`，接着又创建了生产者的对象和消费者的对象，最后通过 `Thread` 类的 `start()` 方法启动两个线程对象模拟生产者和消费者问题。

最后，虽然上述项目通过同步块可以避免出现如下情况：“生产者”在向存储空间添加一个学生信息时，刚添加完学生的姓名还没来得及添加性别时，CPU 就切换到了“消费者”线程，“消费者”线程就会将这个学生的姓名和上一个学生的性别输出，但是却没有实现预期的效果。

7.3 实现线程通信的生产者与消费者项目

为了得到生产者与消费者项目的预想效果，必须要实现如此通信功能：当生产者向储存库中添加一次学生信息后，消费者就取一次。同时，生产者必须等到消费者取完后，才能再次向储存库中添加一次学生信息。

7.3.1 生产者和消费者的类

Producer.java 类用来模拟生产者，该类除了需要对共享资源实现同步外，还必须通过 Object 类的通信方法实现通信功能，修改该类的具体内容如代码 7.5 所示。

代码 7.5 生产者类：Producer.java

```
public class Producer implements Runnable {
    StoreHouse q;                                //创建存储库的对象
    public Producer(StoreHouse q) {              //构造函数
        this.q = q;
    }
    public void run() {                          //实现 run() 方法
        int i = 0;                               //定义一个判断奇偶的变量
        while (true) {
            synchronized (q) {                  //同步块
                if (q.flag)                     //如果储存库满时
                    try {
                        q.wait();               //处于等待状态
                    } catch (InterruptedException e1) {
                        e1.printStackTrace();
                    }
                if (i == 0) {                   //如果为偶数
                    q.name = "小明";
                    try {
                        Thread.sleep(2);        //线程阻塞一段时间
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    q.sex = "男";
                } else {                       //如果为奇数
                    q.name = "小红";
                    q.sex = "女";
                }
                q.flag = true;                 //设置标记变量 flag
                q.notify();                    //唤醒线程
                i = (i + 1) % 2;
            }
        }
    }
}
```

【代码解析】

上述代码在 run()方法中通过无限循环方法，不断地把学生的信息存储到存储库 q 中。

Comsumer.java 类用来模拟生产者，该类除了需要对共享资源实现同步外，还必须通过 Object 类的通信方法实现通信功能，修改该类的具体内容如代码 7.6 所示。

代码 7.6 消费者类：Comsumer.java

```
public class Comsumer implements Runnable {
    StoreHouse q;                                //创建存储库的对象
    public Comsumer(StoreHouse q) {              //构造函数
        this.q = q;
    }
    public void run() {                          //实现 run() 方法
        while (true) {                          //无限循环
            synchronized (q) {                  //同步块
                if (!q.flag) {                  //当储存库中为空时
                    try {
                        q.wait();                //处于等待状态
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                } else {                        //当储存库中不为空时
                    System.out.println("—————");
                    //取出并输出相应的数据
                    System.out.println("学生的名字为:" + q.name);
                    System.out.println("学生的性别为:" + q.sex);
                    q.flag = false;              //设置标记变量 flag
                    q.notify();                  //唤醒线程
                }
            }
        }
    }
}
```

【代码解析】

上述代码在 run()方法中通过无限循环方法，不断地从存储库 q 中获取相应的信息并输出。

当消费者（Comsumer）线程运行时，首先会检查储存库是否为满（StoreHouse.flag），如果为空（!StoreHouse.flag），则会调用 wait()方法进入对象 q 的 wait pool 并同时失去监视器对象 q。如果为满，则会获取并输出储存库中的信息，然后不仅设置储存库标记位 StoreHouse.flag 的值，而且还通过 notify()方法唤醒正在等待的生产者对象。

当生产者（Producer）线程运行时，首先会检查储存库是否为满（StoreHouse.flag），如果为满，则会调用 wait()方法进入对象 q 的 wait pool（等待池）并同时失去监视器对象 q。如果为空，则会通过判断是奇数还是偶数向储存库中存储相应的信息，然后不仅设置储存库标记位 StoreHouse.flag 的值，而且还通过 notify()方法唤醒正在等待的消费者对象。

7.3.2 储存库的类

StoreHouse.java 类用来模拟储存库，该类主要用来实现从 Producer.java 类输入相应信

息。StoreHouse.java 类的具体内容如代码 7.7 所示，该类的 UML 如图 7.8 所示。

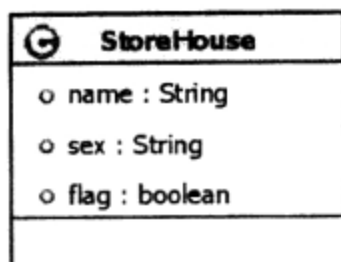


图 7.8 储存库的类

代码 7.7 储存库类：StoreHouse.java

```

public class StoreHouse {
    String name = "no";
    String sex = "no";
    boolean flag = false;           //储存库是否为满的标记
}
  
```

【代码解析】

在上述代码中编写一个数据存储库对象，该对象通过两个成员变量 `name` 和 `sex` 模拟数据存储库被分成两部分，其中一部分用来存储学生的名字（`name` 变量），另一部分用来存储学生的性别（`sex` 变量）。最后还设置了一个 `boolean` 变量 `flag`，用来表示存储库是否为满。

7.4 知识点扩展——线程的通信知识

操作系统中的生产者和消费者问题只说明了一个简单的状况，由多个不相同的线程来完成不同的任务，但是这些任务间有一定的联系，所以操作这些线程需要进行相互交互。例如，有两个线程 A 和 B，线程 A 正在等待另一个线程 B 把数据送过来，数据还没送到之前，该线程进入等待状态，等到数据送到时，线程 B 会通知线程 A 可以处理数据了。这里的“通知”就是所谓的线程交互。

7.4.1 线程通信的基本知识

通过前面章节的学习可以知道，当产生 `Thread` 类对象时，该对象就会处于 `new` 状态，调用 `start()` 方法后就会进入 `runnable` 状态。如果 CPU 调用该线程时，其就会从 `runnable` 状态进入 `running` 状态，运行线程中 `run()` 方法中的代码。那么当调用相应的通信方法 `wait()` 和 `notify()` 后，线程会处于什么状态呢？

当某个线程调用 `wait()` 方法后，其就会进入 `wait pool`（等待池），即处于等待状态，会等待其他线程来通知它。当其他线程调用 `notify()` 方法后，系统就会在 `wait pool` 中“任意”挑选一个正在等待的线程出来，进入 `lock pool`（锁定池），具体过程如图 7.9 所示。

之所以会出现 `lock pool`，是因为通信方法必须在被锁定的程序语句（即同步块或同步方法）中，一个线程等待和唤醒的具体过程如图 7.10 所示。首先线程 `t` 得到对象 `o` 的锁旗

标，当该线程调用 wait()方法后，线程 t 不仅被放置在对象 o 的等待池中，而且同时会自动释放 o 的锁旗标。当其他线程执行了对象 o 的 notify()方法后，线程 t 就有可能从 o 的 wait pool 中被唤醒，并且移动到 lock pool 中，当线程 t 再次获得锁旗标时就会执行下去。

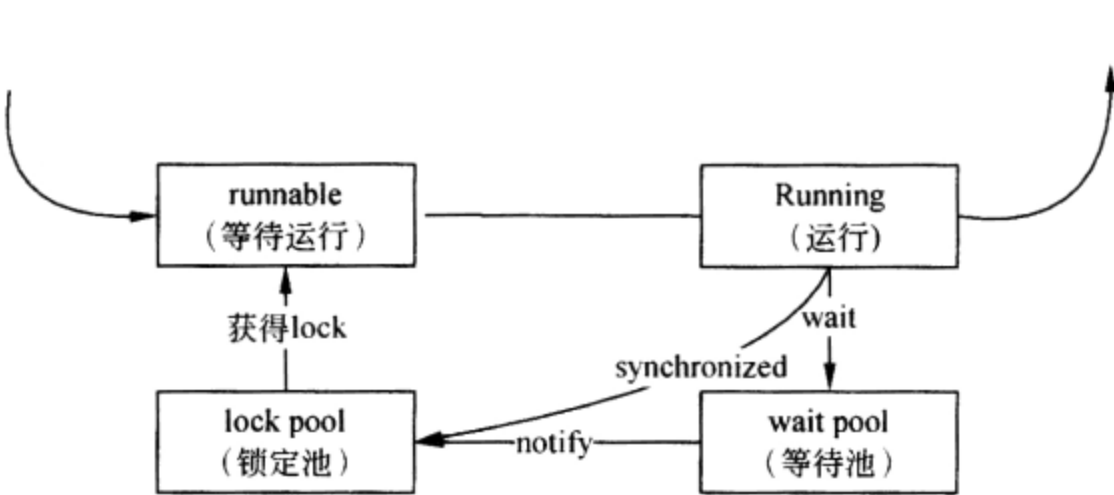


图 7.9 状态转移

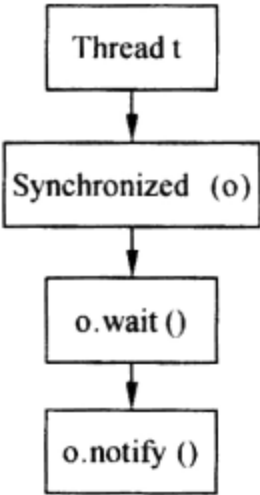


图 7.10 线程的执行过程

到现在为止，对于线程一共存在 3 个 pool 对象，分别为 blocked pool、wait pool 和 lock pool。这 3 个对象是有区别的，对于 blocked pool 来说，其属于系统的，即一个应用系统中只会存在一个该对象。而对于其他两个对象来说，其属于某个对象的。所以在具体应用通信方法时，无论线程调用一个对象的方法 wait()还是 notify()方法时，该线程必须先获取该对象的锁旗标，这样 notify()方法只能唤醒同一个监视器中的 wait()方法。如果应用程序中存在多个监视器，属于某个监视器的方法 wait()和 notify 就是一组，同组里的方法 wait()只能被同组的方法 notify()唤醒。

下面详细讲解线程通信的各种方法。

1. wait()方法

wait 英文意思为等待，其主要用来实现让线程进入等待状态。该方法有两种使用方式。

1.thread.wait(long millis)

参数 millis 用来表示等待的时间，该种方式基本与 sleep()方法的语法相同。

2.thread.wait()

该种方式一般会与 notify()方法配合使用，这种方式让线程无限等下去，直到线程接收到 notify()或 notifyAll()方法为止。

⚠注意：thread 为线程对象，而不是线程类。


2. notify()和notifyAll()方法

notify()方法主要用来唤醒处于 wait pool 中的线程，该方法的基本格式如下：

thread.notify()

如果想用一个方法把所有 wait pool 中的线程全部唤醒，可以使用 notifyAll()方法，该方法的基本格式如下：

```
thread.notifyAll()
```

 注意：与方法 `sleep()`、`yield()` 和 `join()` 不同，方法 `wait()`、`notify()` 和 `notifyAll()` 都是 `java.lang.Object` 类的方法，所以任何对象都可以调用这 3 个方法。

7.4.2 线程通信的具体实例

为了让读者能够掌握线程通信的各种方法，本节将通过一个实现了的线程通信的具体实例来详细讲解，具体步骤如下。

(1) 创建一个库存的类 `Storage.java`，具体内容如代码 7.8 所示。

代码 7.8 库存类：Storage.java

```
public class Storage {
    //定义了两个成员变量
    private int count;                //库存量的大小
    private int size;                //库存量的最大上限
    public Storage(int s) {           //构造函数
        size = s;
    }
    public synchronized void addData(String n) { //添加数据的同步方法
        while (count == size) {         //遍历
            try {
                this.wait();
            } catch (InterruptedException e) {
            }
        }
        count++;
        System.out.println(n + " make data count: " + count);
        this.notify();
    }
    public synchronized void delData(String n) { //取出数据的同步方法
        while (count == 0) {             //遍历
            try {
                this.wait();
            } catch (InterruptedException e) {
            }
        }
        System.out.println(n + " use data count: " + count);
        count--;
        this.notify();
    }
}
```

【代码解析】

- ❑ 上述代码中，定义了两个变量 `count` 和 `size`，由于这两个变量属于共享资源，所以都用 `private` 关键字修饰。
- ❑ 在上述代码中，创建了两个实现业务的方法 `addData()` 和 `delData()`。添加者会调用 `addData()` 方法来累积库存量，而删除者则会调用 `delData()` 方法来递减库存量。
- ❑ 当添加者调用 `addData()` 方法时，会检查库存量 `count` 是否已满，如果是，则会调用 `wait()` 方法进入等待状态；如果不是，则会累积库存量 `count`，然后通过 `notify()`

方法通知正在等待数据的删除者。

- 当删除者调用 `delData()` 方法时，会检查库存量 `count` 是否为空，如果是，则会调用 `wait()` 方法进入等待状态；如果不是，则会递减库存量 `count`，然后通过 `notify()` 方法通知正在等待数据的删除者。

(2) 创建一个添加者和删除者的类 `AddClass.java` 和 `DelClass.java`，具体内容分别如代码 7.9 和代码 7.10 所示。

代码 7.9 添加者类: `AddClass.java`

```
public class AddClass extends Thread {
    //定义了两个成员变量
    private String name;
    private Storage s;
    public AddClass(String n, Storage s) {           //构造函数
        name = n;
        this.s = s;
    }
    public void run() {                             //实现 run() 方法
        while (true) {
            s.addData(name);                         //调用添加数据方法
            try {
                sleep((int) Math.random() * 3000);
            } catch (InterruptedException e) {
            }
        }
    }
}
```

【代码解析】

在上述代码中，由于继承了线程类 `Thread`，所以重写了 `run()` 方法。该类的业务内容主要在 `run()` 方法体中。

代码 7.10 删除者类: `DelClass.java`

```
public class DelClass extends Thread {
    //定义了两个成员变量
    private String name;
    private Storage s;
    public DelClass(String n, Storage s) {           //构造函数
        name = n;
        this.s = s;
    }
    public void run() {                             //实现 run() 方法
        while (true) {
            s.delData(name);                         //调用删除数据方法
            try {
                sleep((int) Math.random() * 3000);
            } catch (InterruptedException e) {
            }
        }
    }
}
```

【代码解析】

在上述代码中，由于继承了线程类 Thread，所以重写了 run()方法。该类的业务内容主要在 run()方法体中。

(3) 创建一个测试类 Test.java，具体内容如代码 7.11 所示。

代码 7.11 测试类：Test.java

```
public class test {
    public static void main(String argv[]) {
        Storage s = new Storage(5);           //创建库存对象
        //创建了三个线程对象
        AddClass p1 = new AddClass("Producer1", s); //添加者 p1
        AddClass p2 = new AddClass("Producer2", s); //添加者 p2
        DelClass c1 = new DelClass("Consumer1", s); //删除者 c1
        //启动 3 个线程
        p1.start();
        p2.start();
        c1.start();
    }
}
```

运行 Test.java 类，其控制台窗口如图 7.11 所示。

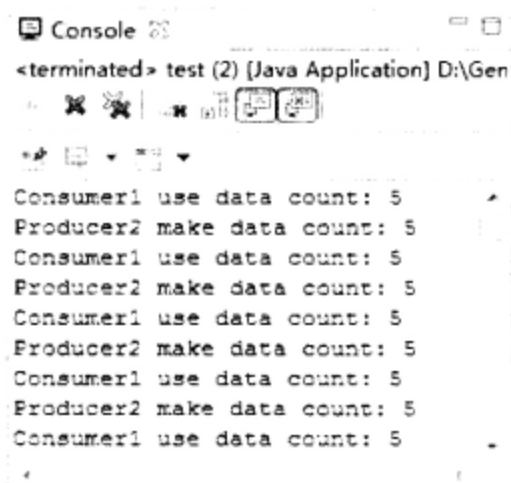


图 7.11 运行结果

从上面的运行结果可以发现，添加者添加数据时不可能无限制地添加下去，因为有库存量的限制，所以在还没有到达一定的库存量时，添加者会继续添加下去，如果库存够了，就等待删除者来删除库存。

7.5 小 结

本章主要讲解了线程高级知识点——生产者和消费者问题，为了能够让读者知道线程通信的重要性，分别讲解了无线程通信的生产者和消费者及实现线程通信的生产者和消费者。本章的最后还详细介绍了实现线程通信的基础知识和原理，通过这些知识的学习，可以使创建的多线程程序更安全、更实用。

第 8 章 关机工具（Timer 类+系统命令）

在 Java 语言中，线程的类除了类 Thread 外，还有类 Timer 和类 TimerTask。因此对于程序员来说，如果想掌握好事件机制，除了掌握类 Thread，还必须要学习类 Timer 和类 TimerTask。本章将通过模拟关机工具的功能，介绍如何调用 Windows 系统命令，还将详细介绍线程的类 Timer。

本章的学习目标如下：

- ❑ 掌握关机工具项目；
- ❑ 掌握如何调用 Windows 系统命令；
- ❑ 理解 Timer 和 TimerTask 类。

8.1 关机工具原理

关机工具项目用来模拟计算机的关机功能，即 Windows 系统的关机功能，除此之外还将实现定时关闭计算机的功能等。

8.1.1 项目结构框架分析

对于关机工具项目，除了该项目的界面外，剩下的就是两个工具类。关机工具项目项目目录如图 8.1 所示，该项目中的 3 个类分别为关机工具类 CutDownTool.java、定时关机工具类 CountTimeTool.java 和关机工具项目界面类 CloseComputer.java。

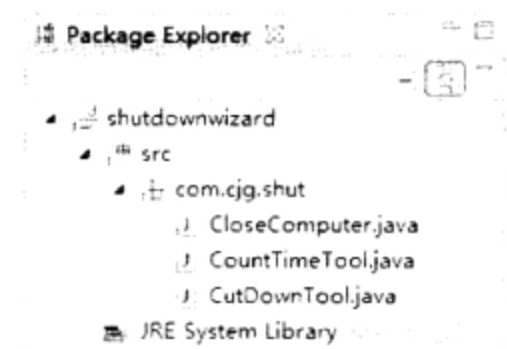


图 8.1 项目目录

8.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括关机工具的初始化、按住鼠

标功能和放开鼠标后的功能。

1. 关机工具初始化

当运行关机工具项目中的 CloseComputer 类后，就会出现如图 8.2 所示的初始界面。在该界面中将显示出实现关机的各种方式。

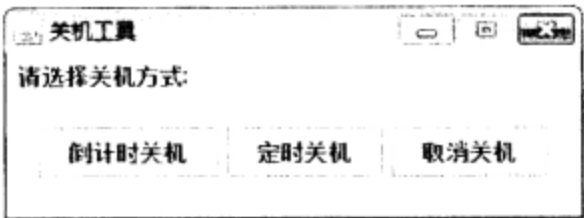


图 8.2 关机工具初始界面

2. 倒计时关机功能

关机工具项目要实现倒计时关机功能，可以在初始化对话框中单击“倒计时关机”按钮，弹出时间输入对话框。在该对话框中输入相应的时间后，单击“确定”按钮就会实现以输入框中的时间倒计时关机的功能。如果想取消该功能，可以在初始化对话框中单击“取消关机”按钮，弹出确认对话框。在该对话框中单击“确定”按钮就可以实现取消关机的功能，具体过程如图 8.3 所示。

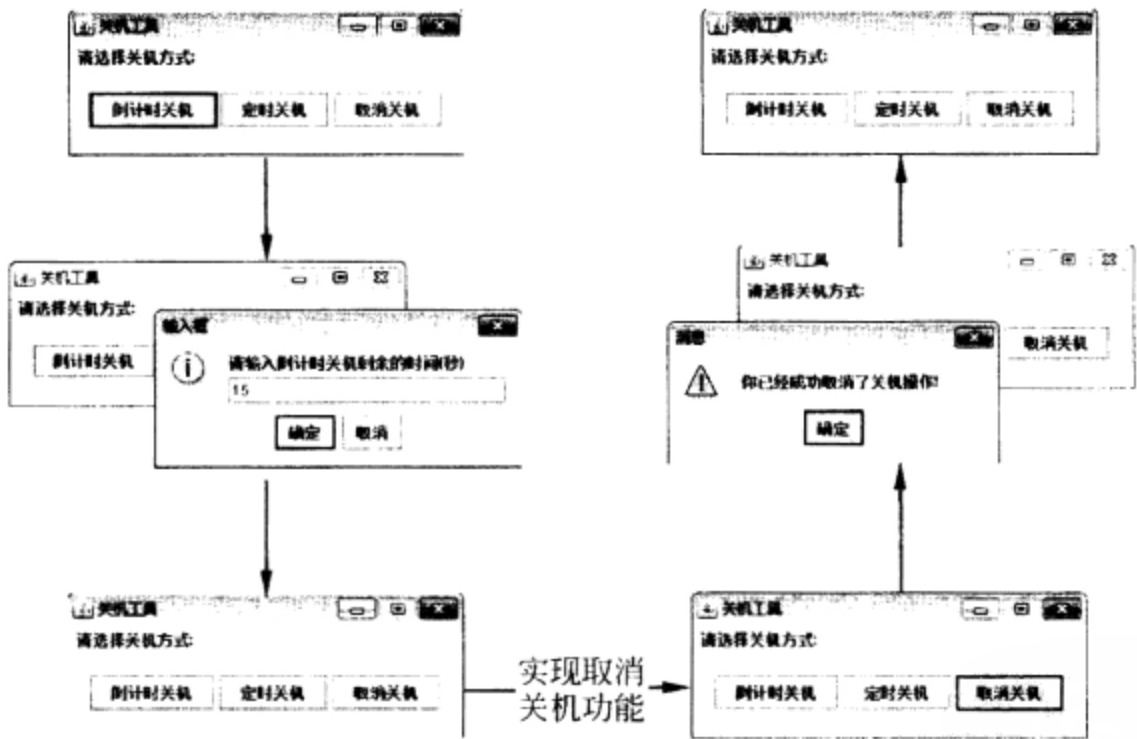


图 8.3 倒计时关机的过程

3. 定时关机功能

关机工具项目要实现定时关机功能，可以在初始化对话框中单击“定时关机”按钮，弹出时间输入对话框。在该对话框中输入相应的时间后，单击“确定”按钮就会实现以输入框中的时间倒计时关机功能。如果想取消该功能，可以在初始化对话框中单击“取消关机”按钮，弹出确认对话框。在该对话框中单击“确定”按钮就可以实现取消关机的功能，具体过程如图 8.4 所示。

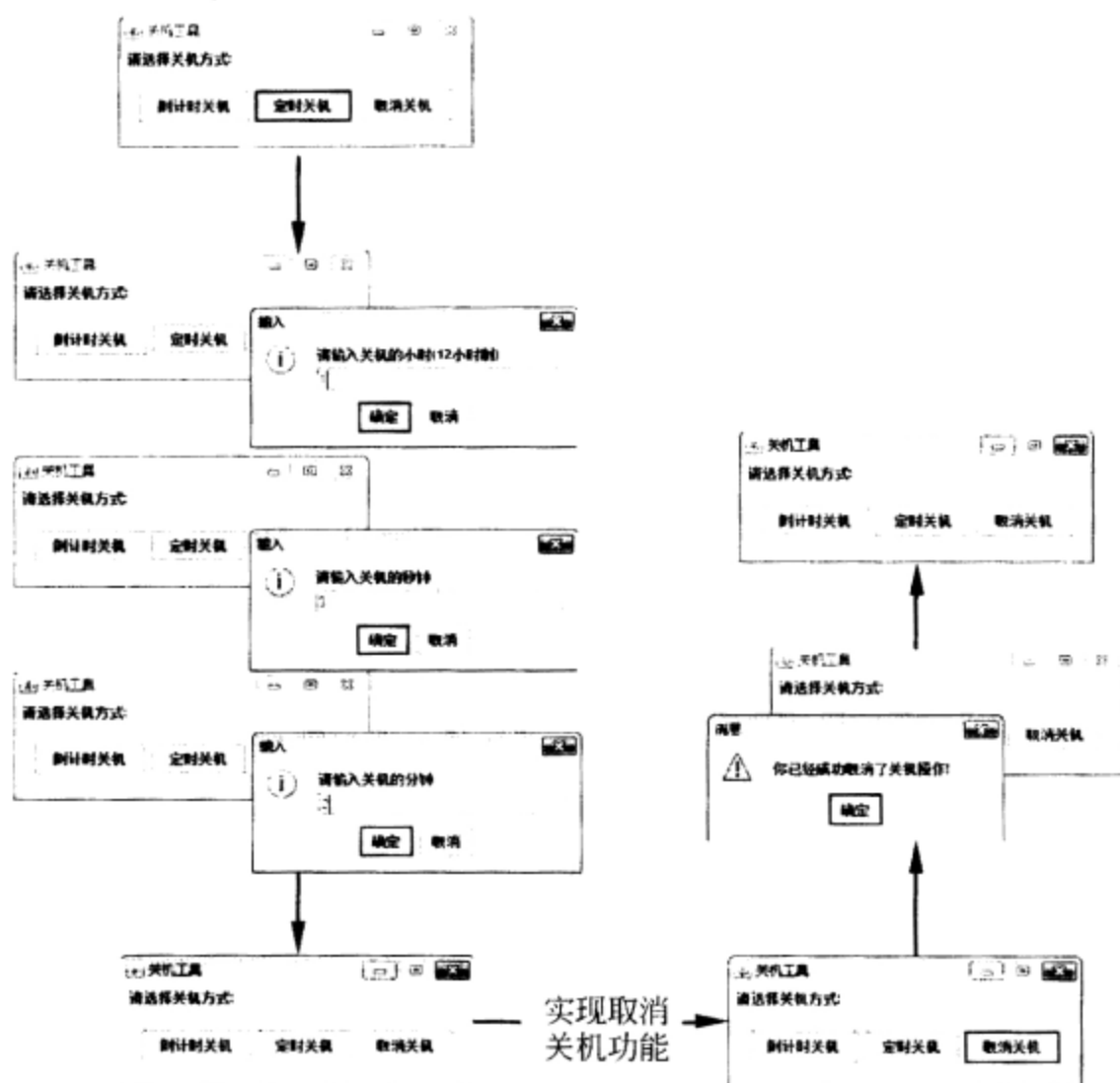


图 8.4 定时关机的过程

8.2 关机工具的实现过程

本章通过调用系统命令和多线程的相关知识来实现关机工具项目，具体程序架构如图8.5所示，它包含两个工具类和一个界面类。

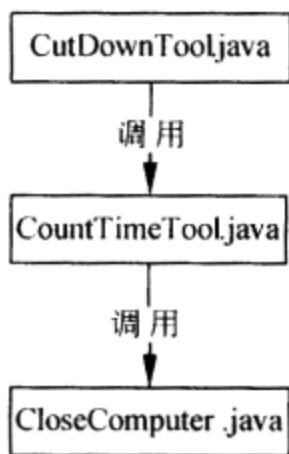


图 8.5 程序关系图

8.2.1 关机工具的类

CloseComputer.java 类用来实现“关机工具”项目的界面，所以该类继承了类 JFrame，

又由于该类需要实现事件机制，所以实现了接口 `ActionListener`。该类的具体内容如代码 8.1 所示，UML 如图 8.6 所示。

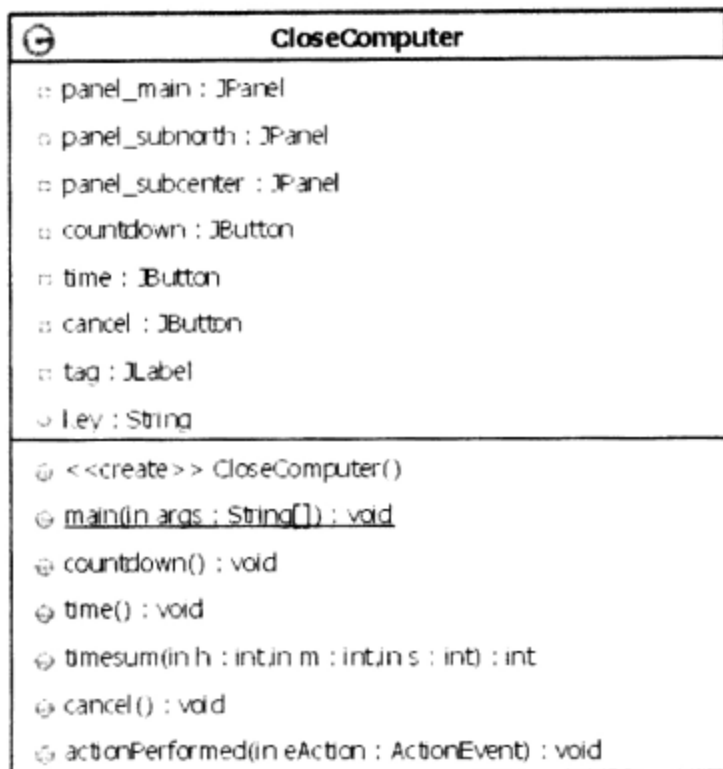


图 8.6 关机工具类图

代码 8.1 关机工具类: `CloseComputer.java`

```

public class CloseComputer extends JFrame implements ActionListener {
    //创建成员变量
    //创建实现 BorderLayout 布局的面板对象 panel_main
    private JPanel panel_main = new JPanel(new BorderLayout(5, 10));
    //创建实现 FlowLayout 布局的面板对象 panel_subnorth
    private JPanel panel_subnorth = new JPanel(new FlowLayout(3));
    //创建实现 FlowLayout 布局的面板对象 panel_subcenter
    private JPanel panel_subcenter = new JPanel(new FlowLayout(1, 5, 5));
    //创建了 3 个按钮对象 countdown、time 和 cancel
    private JButton countdown = new JButton("倒计时关机");
    private JButton time = new JButton("定时关机");
    private JButton cancel = new JButton("取消关机");
    private JLabel tag; //创建标签对象 tag
    String key; //创建字符串对象 key
    public CloseComputer() { //构造函数
        this.getContentPane().add(panel_main); //添加对象 panel_main 到主窗口里
        //添加对象 panel_subnorth 到对象 panel_main 窗口里
        panel_main.add(panel_subnorth, BorderLayout.NORTH);
        //添加对象 panel_subcenter 到对象 panel_main 窗口里
        panel_main.add(panel_subcenter, BorderLayout.CENTER);
        //添加标签对象 tag 到对象 panel_subnorth 里
        panel_subnorth.add(tag = new JLabel("请选择关机方式:"));
        //添加 3 个按钮到对象 panel_subcenter 里
        panel_subcenter.add(countdown);
        panel_subcenter.add(time);
        panel_subcenter.add(cancel);
    }
}
  
```

```

        //为 3 个按钮注册事件监听器
        countdown.addActionListener(this);
        time.addActionListener(this);
        cancel.addActionListener(this);
    }
    public static void main(String[] args) throws Exception { //主方法
        CloseComputer frame = new CloseComputer(); //创建 CloseComputer 对象
        //设置窗口关闭功能
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setSize(320, 120); //设置窗口的大小
        frame.setTitle("关机工具"); //设置标题
        frame.setLocation(350, 350);
        //设置显示
        frame.setVisible(true);
        frame.setResizable(false);
    }
    public void countdown() { //“倒计时关机”调用的方法
        //获取输入的信息
        key = JOptionPane.showInputDialog(this, "请输入倒计时关机剩余的时间(秒)", "输入框", 1);
        CountTimeTool.delaytime(Long.parseLong(key));
        //调用类 CountTimeTool 的 delaytime
    }
    public void time() { //“定时关机”调用的方法
        Calendar calendar = Calendar.getInstance(); //获取当前系统的时间
        //获取当前的时、分和秒
        int h = calendar.get(Calendar.HOUR);
        int m = calendar.get(Calendar.MINUTE);
        int s = calendar.get(Calendar.SECOND);
        String a = String.valueOf(h); //转换变量 h 为字符串
        int hour, minute, second; //定义输入的时、分和秒 int 类型变量
        String hourtmp, minutetmp, secondtmp;
        //定义输入的时、分和秒 string 类型变量
        //为变量 hourtmp, minutetmp, secondtmp 赋值
        hourtmp = JOptionPane.showInputDialog(this, "请输入关机的小时(12 小时制)", "输入", 1);
        minutetmp = JOptionPane.showInputDialog(this, "请输入关机的分钟", "输入", 1);
        secondtmp = JOptionPane.showInputDialog(this, "请输入关机的秒钟", "输入", 1);
        //把 Sting 类型变量转换成 int 类型变量
        hour = Integer.parseInt(hourtmp);
        minute = Integer.parseInt(minutetmp);
        second = Integer.parseInt(secondtmp);
        //通过调用 timesum() 方法, 计算出当前系统的时间 currently_time 和输入的时间 set_time
        long set_time = timesum(hour, minute, second);
        long currently_time = timesum(h, m, s);
        //获取设置时间与系统时间之间的差
        long discrepancy_time = set_time - currently_time;
        if (discrepancy_time < 0) { //当设置时间比系统时间早时
            try {

```

```

        //执行关闭功能
        Runtime.getRuntime().exec("shutdown -s");
    } catch (IOException e) {
        e.printStackTrace();
    }
} else {
    //调用类 CountTimeTool 的 delaytime() 方法
    CountTimeTool.delaytime(discrepancy_time);
    JOptionPane.showMessageDialog(this, "恭喜你, 设置成功!", "确认",
    2);
}
}
//计算出时间总和, 并返回
public int timesum(int h, int m, int s) { //把时和分变量分别转换成秒
    int sum = h * 3600 + m * 60 + s;
    return sum;
}
public void cancel() { //“取消关机”调用的方法
    try {
        //显示信息框
        JOptionPane.showMessageDialog(this, "你已经成功取消了关机操作!",
        "消息", 2);
        //执行取消关闭计算机命令
        Runtime.getRuntime().exec("shutdown -a");
    } catch (IOException e) {
    }
}
public void actionPerformed(ActionEvent eAction) { //事件监听事件
    String ActionCommand = eAction.getActionCommand();
    if (eAction.getSource() instanceof JButton) {
        //发生事件的组件是 JButton 类型
        if ("倒计时关机".equals(ActionCommand)) {
            //如果为“倒计时关机”按钮
            countdown(); //调用 countdown() 方法
        }
        if ("定时关机".equals(ActionCommand)) { //如果为“定时关机”按钮
            time(); //调用 time() 方法
        }
        if ("取消关机".equals(ActionCommand)) { //如果为“取消关机”按钮
            cancel(); //调用 cancel() 方法
        }
    }
}
}
}

```

【代码解析】

- ❑ 上述代码实现了关机工具项目界面, 该用户界面涉及的具体容器、对象和布局如图 8.7 所示。
- ❑ 上述代码中, 语句“Runtime.getRuntime().exec("shutdown -s")”的意思就是启动一个新的进程执行 shutdown -s 命令。该命令的含义为关闭计算机, 而命令 shutdown -a 的含义为取消关闭计算机。

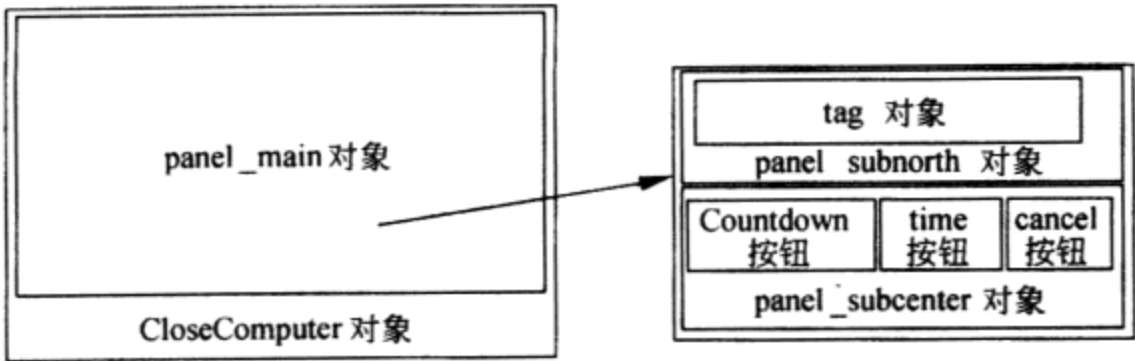


图 8.7 布局

8.2.2 关机工具的工具类

CutDownTool.java 类主要用来实现关闭计算机的功能，由于该类需要被定时调用，所以继承了 TimerTask 类，该类的具体内容如代码 8.2 所示。

代码 8.2 关机工具类：CutDownTool.java

```
public class CutDownTool extends TimerTask {           //继承了 TimerTask 类
    public void run() {                                //重写 run() 方法
        try {
            //执行关闭计算机命令
            Runtime.getRuntime().exec("shutdown -s");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

【代码解析】

在上述代码中，首先继承了 TimerTask 类，然后重写该类的 run()方法，在该方法中通过执行命令 shutdown-s 实现了关闭计算机的功能。

CutDownTool.java 类主要用来实现定时关闭计算机的功能，即在一定时间后执行 CutDownTool 类，该类的具体内容如代码 8.3 所示。

代码 8.3 定时关机工具类：CountTimeTool.java

```
public class CountTimeTool {
    public static void delaytime(long dt) {
        long delay = 1000;                               //间隔变量 delay
        Timer timer = new Timer();                         //创建 Timer 变量
        CutDownTool w1 = new CutDownTool();               //创建 CutDownTool 对象
        timer.schedule(w1, delay * dt);                   //在 delay*dt 时间段后执行 w1
    }
}
```

【代码解析】

上述代码中为了实现定时的效果，调用了类 Timer 的 schedule()方法，表示 w1 被安排在 delay*dt 指定的时间后执行。

8.3 知识点扩展——关机工具项目涉及的知识

在关机工具项目中，除了涉及用户界面的知识外，还通过 `Timer` 和 `TimerTask` 类实现了多线程，同时还通过调用系统命令实现了计算机关闭功能。本节除了详细介绍线程的类 `Timer` 和 `TimerTask` 外，还将详细讲解系统命令——`Shutdown`。

8.3.1 `Timer` 和 `TimerTask` 类

在 `Java` 语言中如果想实现每隔一段时间去执行某个任务，除了可以使用线程类 `Thread` 外，还可以使用包 `java.util` 中的类 `Timer` 和 `TimerTask`。对于前者首先需要创建后台进程，然后让其每隔一段时间执行，具体实现时比较麻烦。对于后者，只需要在 `TimerTask` 类的 `run()` 方法中实现任务，然后 `Timer` 类就会安排其每隔一段时间执行，具体实现时比较简单。

所谓 `Timer`，就是一种定时器工具，用来在一个后台线程计划执行指定任务。它可以计划执行该任务一次或反复多次。所谓 `TimerTask`，其实质是一个抽象类，其子类代表一个可以被 `Timer` 计划的任務。

下面通过一个具体的类 `TimerTest` 来讲解 `Timer` 和 `TimerTask` 类的使用，该类的具体内容如代码 8.4 所示。

代码 8.4 测试类: `TimerTest.java`

```
public class TimerTest extends TimerTask {
    //创建成员变量
    String index;                                     //创建一个字符串变量
    Timer myTimer = new Timer();                     //创建一个 Timer 对象
    public TimerTest(String index) {                 //构造函数
        this.index = index;
    }
    public void run() {                               //重写 run() 方法
        System.out.println(index);                  //反复执行的任务
    }
    public void start(int delay, int internal) {      //实现反复执行功能
        //在 delay * 1000 时间后开始执行 TimerTest 对象，执行后每隔 internal * 1000
        反复执行
        myTimer.schedule(this, delay * 1000, internal * 1000);
    }
    public void end() {
        myTimer.cancel();                            //调用 cancel() 方法结束
    }
    public static void main(String args[]) {          //主方法
        //创建和设置 myTask1 对象
        TimerTest myTask1 = new TimerTest("线程 1 执行");
        myTask1.start(0, 3);                          //调用 start() 方法
        //创建和设置 myTask2 对象
        TimerTest myTask2 = new TimerTest("线程 2 执行");
        myTask2.start(0, 1);                          //调用 start() 方法
        try {

```



```

        Thread.sleep(6000);           //使线程休眠 6 秒
    } catch (InterruptedException e) {
    }
    //结束
    myTask1.end();
    myTask2.end();
}
}

```

运行 TimerTest.java 类，其控制台窗口如图 8.8 所示。

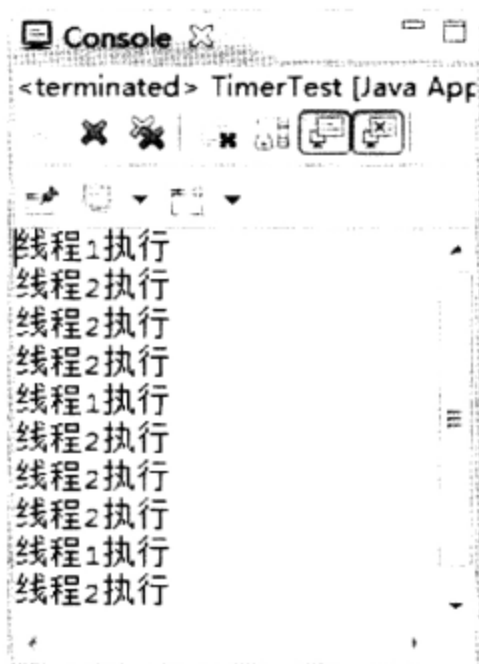


图 8.8 运行结果

【代码解析】

- ❑ 在上述代码的众多方法中，由于继承了 TimerTask 类，所以需要重写 run()方法实现业务功能；在启动线程的 start()方法中，通过调用 Timer 类的 schedule()方法执行 TimerTest 对象；在结束线程的 end()方法中，通过调用 Timer 类的 cancel()方法结束线程。
- ❑ 在上述代码的主方法 main()中，首先创建了两个线程对象 myTask1 和 myTask2，然后通过 start()方法启动线程，通过 end()方法结束线程，在具体调用这两个方法之间，通过 Thread.sleep()方法实现当前线程休眠一段时间。

最后，通过查看 API 帮助文档可以发现，如果想实现执行计划、定时任务，首先需要继承 TimerTask 类并重写该类的 run()方法来实现任务功能，然后通过调用 Timer 类的 schedule()方法执行 TimerTask 类对象。详细介绍如下。

1. TimerTask类

对于 TimerTask 类，其是扩展了 Object 并实现了 Runnable 接口的抽象类，因此在具体编程时，可以在 public void run()方法中编写具体的执行任务。该类拥有几个重要方法，如下所示。

(1) public boolean cancel()

该方法用来终止计时器任务的执行运行。

⚠注意: Timer 类要求循环执行任务 (TimerTask 任务) 时, 如果该对象正在执行, 则会在执行完之后不会再循环执行; 如果还未执行或处于停歇期, 则不会执行。

(2) `public abstract void run()`

该方法为计时器任务要执行的操作。

(3) `public long scheduledExecutionTime()`

该方法用来返回计时器任务被安排最后执行的时间, 一般用来确定计时器任务的运行是否足够及时, 执行是否正当。

2. Timer类

对于 Timer 类, 主要用来控制 TimerTask 类的任务执行一次或者定期重复执行, 该类拥有 4 个构造函数, 分别如下。

(1) `Timer()`

该构造函数用来创建一个默认的计时器。

(2) `Timer(boolean isDaemon)`

参数 `isDaemon` 指定线程的守护, 创建一个指定守护线程的计时器。

⚠注意: 当参数 `isDaemon` 为 `true` 时, 该计时器对象相关线程为后台进程线程, 根据线程知识可以知道, 如果应用程序中还有程序继续运行, 后台线程进程就不会停止执行。

(3) `Timer(String name)`

参数 `name` 为线程的名字, 用来创建一个指定名称的相关线程的计时器。

(4) `Timer(String name, boolean isDaemon)`

该函数用来创建一个新计时器, 不仅指定相关线程的名称, 而且还指定其为后台线程。

在具体创建计时器对象时, 不推荐使用 `Timer timer = new Timer(true)` 语句。因为这种方式创建的计时器对象, 在应用程序结束后会自动结束, 非常不利于使用。

除了构造函数外, 该 Timer 类还拥有其他几个重要方法, 其中最重要的方法就是各种重载的 `schedule()` 方法和 `scheduleAtFixedRate()` 方法, 分别如下:

(1) `public void schedule(TimerTask task, Date time)`

参数 `task` 为所计时器任务对象, 参数 `time` 为 `Date` 类型对象, 该方法用来实现在指定的时间后执行计时器任务。

⚠注意: 如果参数 `time` 为过去时, 则任务 `task` 对象会立刻执行。

(2) `public void schedule(TimerTask task, long delay)`

参数 `delay` 为 `long` 类型, 该方法用来实现在指定的时间 (毫秒) 后执行计时器任务。


(3) `public void schedule(TimerTask task, Date firstTime, long period)`

参数 `firstTime` 为 `Date` 类型, 表示第一次执行的时间; 参数 `period` 为 `long` 类型, 表示

间隔的时间（毫秒）；该方法用来在 firstTime 时间后执行任务 task，同时在第一次执行后每隔 period（毫秒）反复执行。每一次重复的间隔时间会因为前一次 task 的执行时间受到影响。

(4) `public void schedule(TimerTask task,long delay, long period)`

参数 delay 为 long 类型，该方法用来在 firstTime 时间（毫秒）后执行任务 task，同时在第一次执行后每隔 period（毫秒）反复执行。每一次重复的间隔时间会因为前一次 task 的执行时间受到影响。

 **注意：**当表示时间的参数为 long 类型时，表示毫秒。由于 long 类型的时间并不能保证与时钟准确同步，所以拥有 long 类型的时间参数适合短期的保持频率。

(5) `public void scheduleAtFixedRate(TimerTask task,Date firstTime, long period)`

参数 firstTime 为 Date 类型，该方法用来在 firstTime 时间后执行任务 task，同时在第一次执行后每隔 period（毫秒）反复执行。每一次重复的间隔时间不会因为前一次 task 的执行时间受到影响。

(6) `public void scheduleAtFixedRate(TimerTask task,long delay,long period)`


参数 firstTime 为 long 类型，该方法用来在 delay 时间后执行任务 task，同时在第一次执行后每隔 period（毫秒）反复执行。每一次重复的间隔时间不会因为前一次 task 的执行时间受到影响。

当需要反复执行的任务时，如果注重任务执行的平滑度，那么需要使用 `schedule()` 方法，如果注重任务执行的频度，那么需要使用 `scheduleAtFixedRate()` 方法。例如，当使用 `schedule()` 方法时，意味着所有计时器任务之间的时间间隔至少为 1 秒，即如果有一个计时器任务因为某种原因迟到了（未按计划执行），那么余下的所有计时器任务都要延时执行。当使用 `scheduleAtFixedRate()` 方法时，如果有一个计时器任务因为某种原因迟到了（未按计划执行），那么后面余下的所有计时器任务还会以固定的间隔时间执行。即 `schedule()` 方法会实现固定延迟执行功能，而 `scheduleAtFixedRate()` 方法会实现固定速率执行功能。

最后还存在一个关闭 Timer 类的对象方法，如下所示。

```
public void cancel()
```

该方法用来终止 Timer 的功能执行。

 **注意：**当执行该方法后，虽然不会对正在执行的任务有影响，但是却不会再分配其他任务。

8.3.2 shutdown 命令

在 Windows 系统中，shutdown 命令能够实现关闭或重新启动一台本地或远程计算机的功能。本节不仅讲解了 shutdown 命令，而且还讲解了实现该命令的程序 shutdown.exe。对于 shutdown 命令，其具体语法如下：

```
shutdown [/i | /l | /s | /r | /a | /p | /h | /e] [/f] [/m //ComputerName]
[/t XXX] [/d [p:]XX:YY/c"Comment"]
```

上述语法中各个参数的含义如表 8.1 所示。

表 8.1 参数含义

命令参数	含 义
/i	显示“远程关机对话框”
/l	立即注销当前用户
/s	关闭计算机
/r	关机后重新启动计算机
/a	取消关机操作
/p	仅关闭本地计算机（而不是远程计算机），没有超时期或警告
/h	使本地计算机处于休眠状态（如果已启用休眠）
/e	强制关闭正在运行的应用程序而不提前警告用户
/f	强制关闭正在运行的应用程序而不提前警告用户
/m //ComputerName	指定目标计算机
/t XXX	将重新启动或关机前超时期限或延迟设置为 XXX 秒
/d [p:]XX:YY	列出系统重新启动、关机或关闭电源的原因
/c"Comment"	对关机原因做出详细注释

注意：在具体使用 shutdown 命令时，用户必须是 Administrators 组的成员才能对本地或远程管理计算机的意外关机进行批注。

在 Windows XP 系统中存在一个名为 shutdown.exe 的程序，该程序的具体目录如图 8.9 所示，当在运行 shutdown 命令时，其实就是执行该程序。

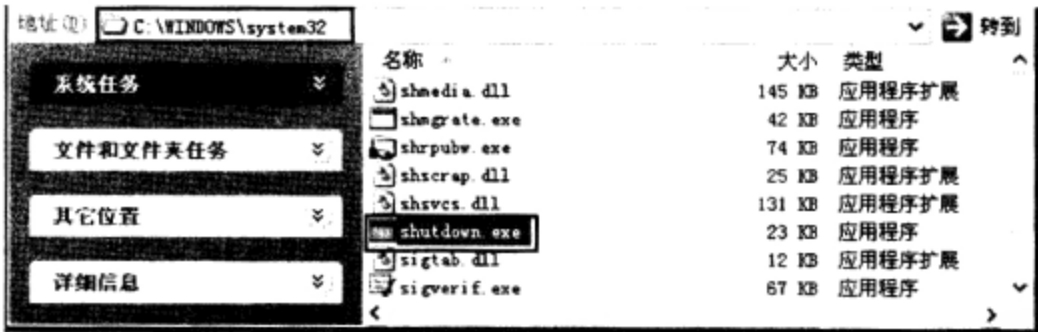


图 8.9 文件目录

下面介绍 shutdown 命令的几个具体实例。

(1) 如果想让计算机在 23:00 关机，可以在运行对话框中输入如下命令：

```
at 23:00 shutdown -s
```

当执行上述命令后，只要到了 23 点，计算机就会出现“系统关机”对话框，默认有 30 秒钟的倒计时来提示保存工作。

(2) 如果想在 1 小时后自动关闭计算机，即实现倒计时关闭功能，可以在运行对话框中输入如下命令：

```
shutdown -s -t 3600
```

之所以为 3600，是因为一分钟为 60 秒，而 1 小时共 3600 秒。

(3) 如果在倒计时自动关机过程中想取消倒计时自动关机，可以在运行对话框中输入如下命令：

shutdown - a

(4) 如果想打开“远程关机对话框”，如图 8.10 所示，对自动关机进行设置，可以在运行对话框中输入如下命令：

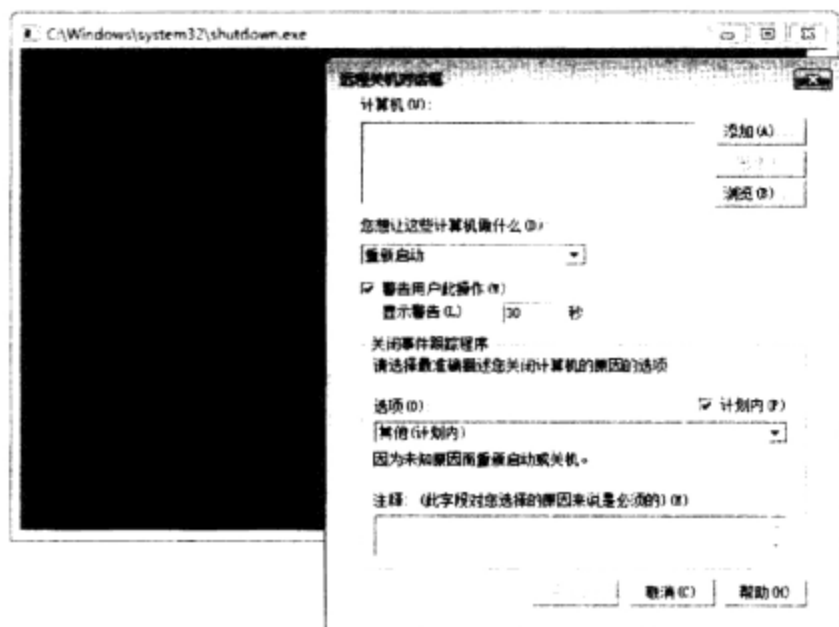


图 8.10 远程关机对话框

```
shutdown -i
```

(5) 如果想显示 shutdown 命令的帮助信息, 如图 8.11 和图 8.12 所示, 可以在运行对话框中输入如下命令:

shutdown

或

shutdown /?

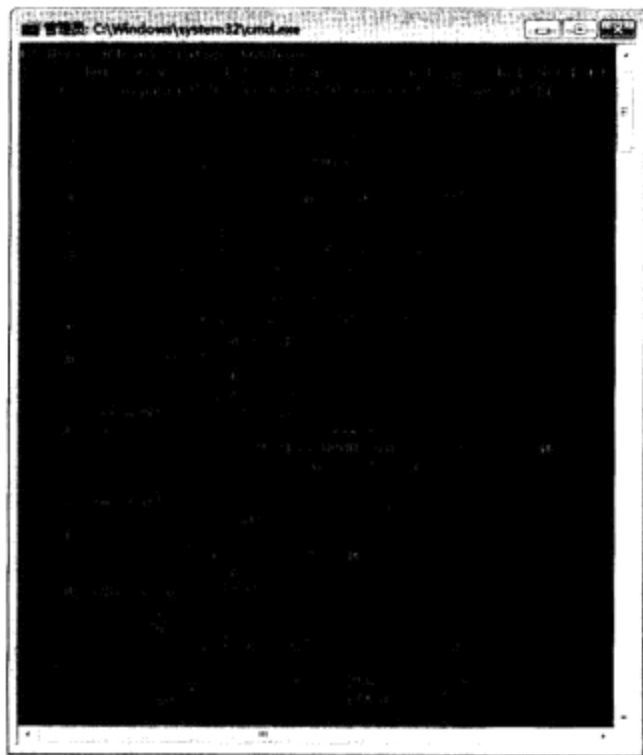


图 8.11 显示帮助信息

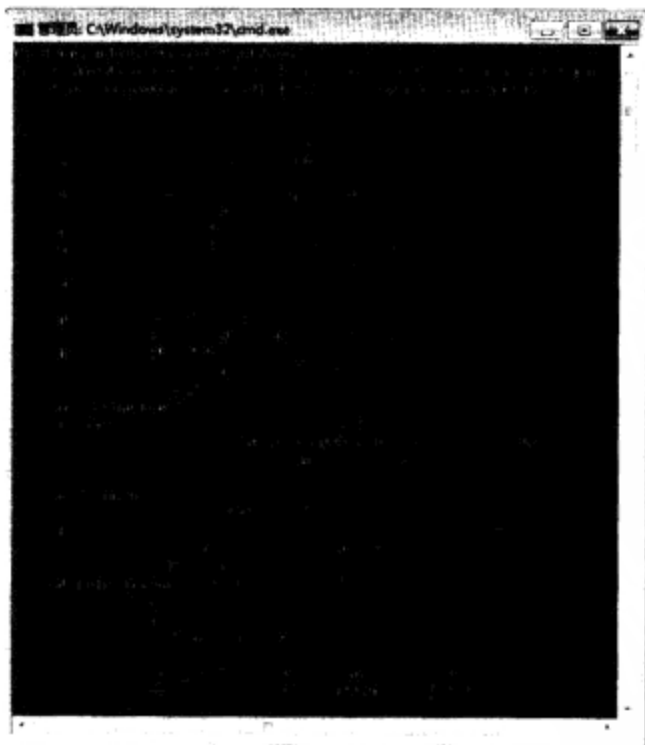


图 8.12 显示帮助信息

8.3.3 通过 shutdown 命令实现网络远程关机

平时经常会遇到这种情况，公司下班后常有很多员工不关电脑，领导发现后要求解决这个问题。假如你是网管该如何办呢？有些网管会到每个办公室去查看，把每台电脑都手工关闭。其实不需要这么麻烦，只要为每台计算机安装 Windows XP 系统，通过 shutdown 命令就可以实现网络远程关机，具体步骤如下。

(1) 需要为每个网络计算机设置一个用于进行远程关机的用户，选择“开始” | “运行”命令，弹出“运行”对话框如图 8.13 所示。在其中输入 Gpedit.msc 命令，单击“确定”按钮，打开组策略编辑器，如图 8.14 所示。

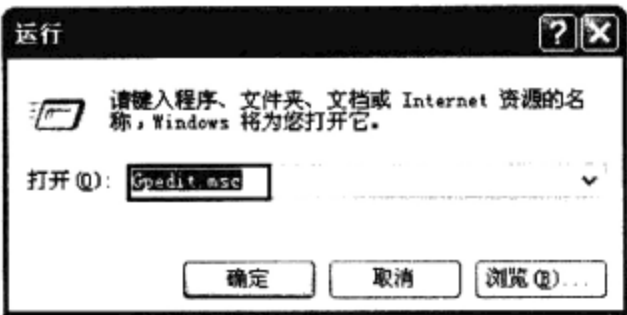


图 8.13 “运行”对话框

(2) 在组策略编辑器对话框的左边选择“计算机配置” | “Windows 设置” | “安全设置” | “本地策略” | “用户权利指派”命令，在右边的窗口选择“从远端系统强制关机”命令，如图 8.14 所示。之后会弹出如图 8.15 所示对话框，在该对话框中默认显示只有 Administrators 组的成员才有权远程关机。单击对话框下方的“添加用户或组(U)”按钮，然后在弹出的对话框中输入 cjgong（管理员账号）命令，再单击“确定”按钮，如图 8.16 所示。这时在“从远端系统强制关机”的属性中便添加了一个 cjgong 用户，如图 8.17 所示。最后单击“确定”按钮关闭“组策略”编辑器对话框。

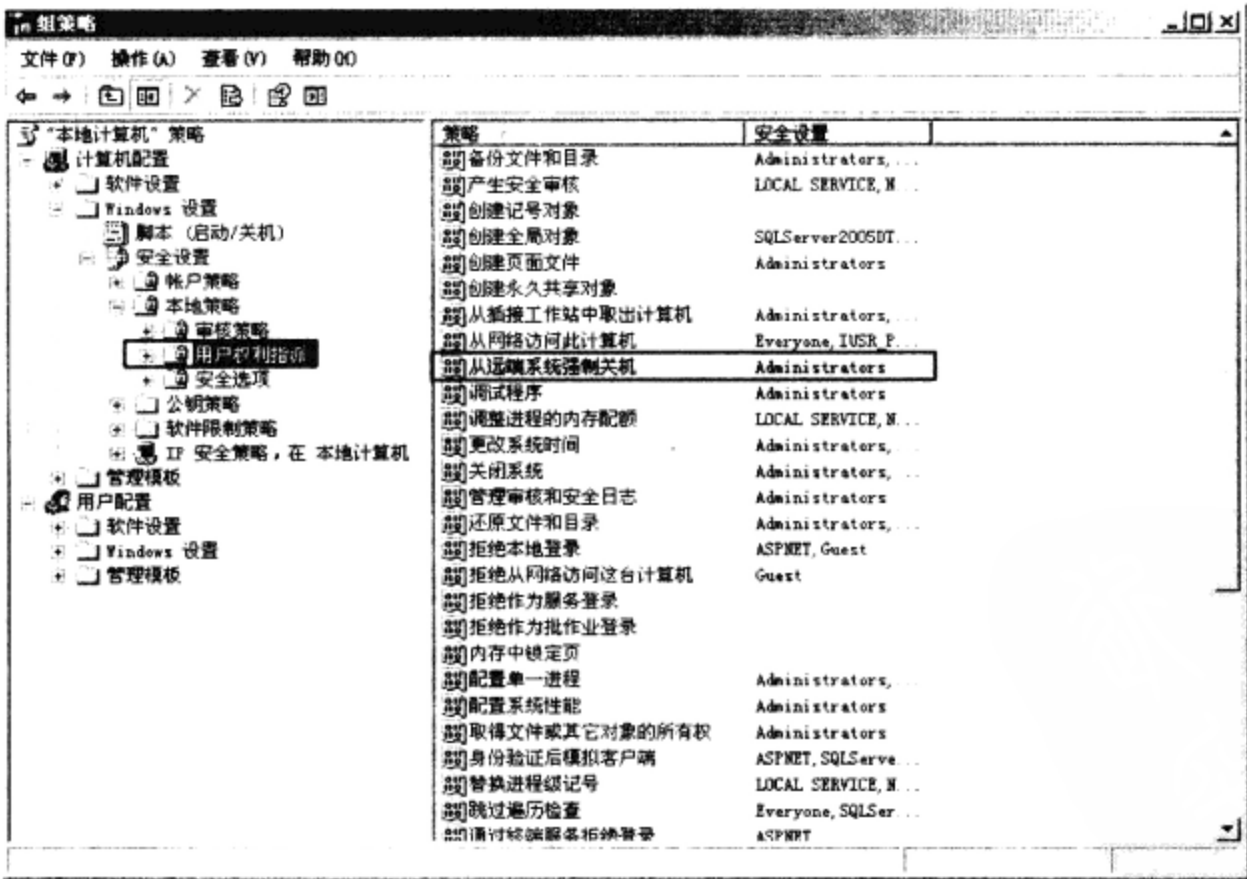


图 8.14 组策略编辑器

(3) 如果对各个网络计算机进行上述操作后，就会实现给每台计算机的 cjgong 用户授予了远程关机的权限。这时如果登录了网络计算机中的任何一台，就可以通过网络关闭其他的任何一台计算机，即选择“开始” | “运行”命令，在弹出的对话框中输入 shutdown

-I 命令，屏幕上将显示“远程关机”对话框，如图 8.18 所示。

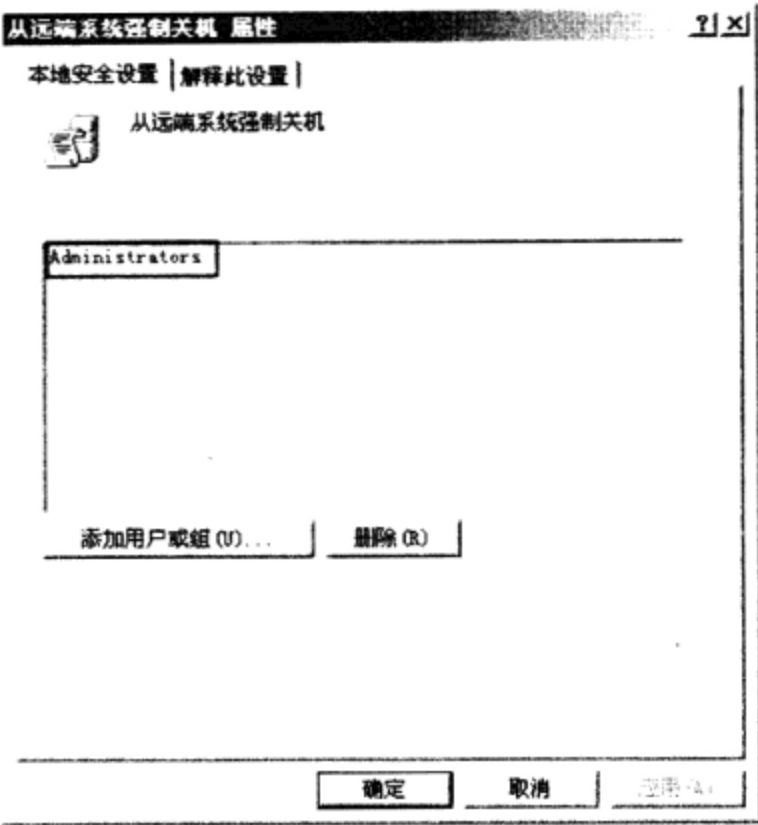


图 8.15 从远端系统强制关机



图 8.16 选择用户或组

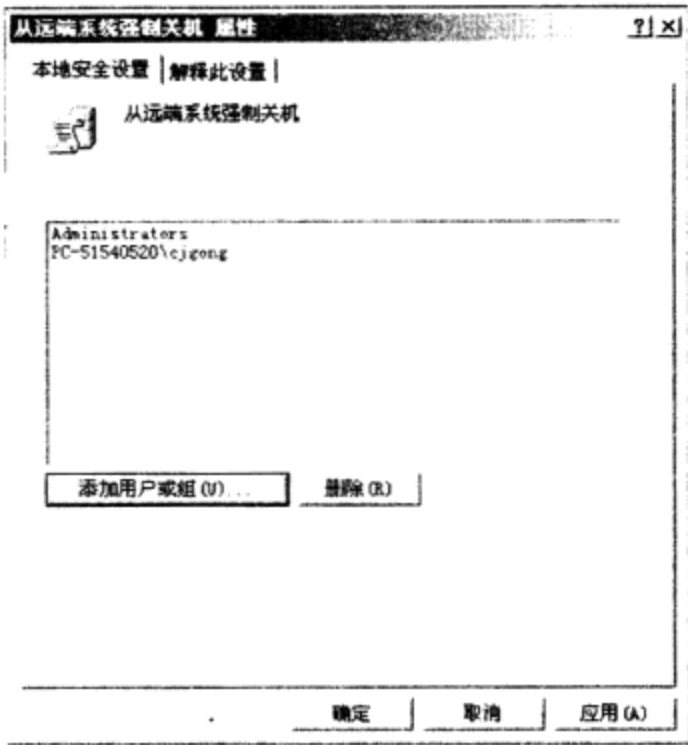


图 8.17 从远端系统强制关机

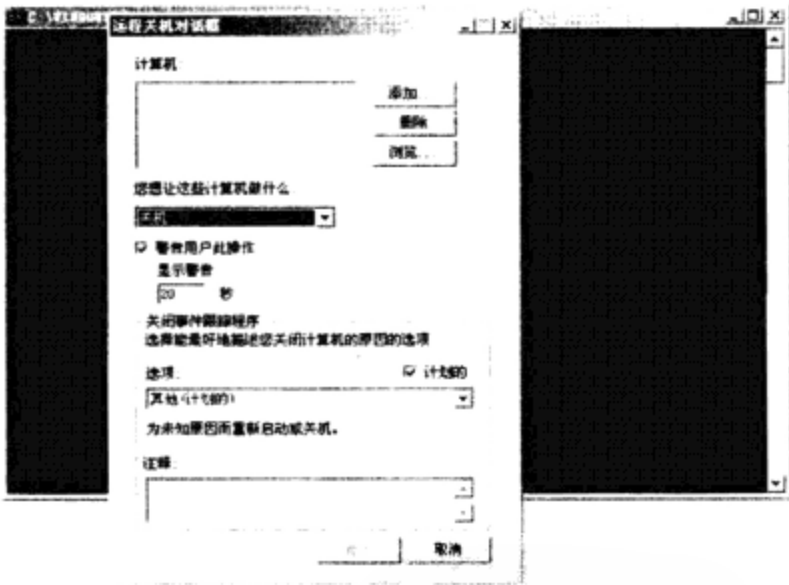


图 8.18 远程关机对话框

(4) 在“远程关机”对话框中，单击“浏览”按钮可以为“计算机”文本框中选择需要关机的网络计算机，如图 8.19 所示。然后在“您想让这些计算机做什么”下拉列表框中选择“关机”选项，最后在“选项”下拉列表框中选择一个合适的关闭理由，然后点击“确定”按钮就可以实现关闭网络计算机的功能。

注意：虽然 shutdown.exe 是 Windows XP 下的程序，但是在 Windows 2000 中也可以调用该命令实现关闭计算机的功能，但是必须把该命令程序进行复制。具体方法是在 Window XP 安装目录下的 system32 文件夹中找到 shutdown.exe 项，将它复制到 Window 2000 安装目录下的 system32 文件夹中即可。



图 8.19 远程关机对话框

8.4 小 结

本章主要通过 TimerTask 类和 shutdown 命令实现关机项目，虽然该项目比较小，只包含实现关机的 CloseComputer 类、关机类调用的工具类 CutDownTool 和 CountTimeTool 3 个类，但是该项目涉及的知识点却不少，例如线程的知识、调用系统命令的知识和 shutdown 命令的知识。本章最后还详细介绍了关机命令——shutdown 的基础知识和具体使用方式。

第3篇 GUI（图形用户界面）开发

- ▶▶ 第9章 典型的图形用户界面（各种组件）
- ▶▶ 第10章 计算器（布局管理器）
- ▶▶ 第11章 秒表（事件+线程）
- ▶▶ 第12章 捉迷藏游戏（事件）
- ▶▶ 第13章 鼠标绘直线（绘图+事件）
- ▶▶ 第14章 指针时钟项目（Swing 组件+时间算法）
- ▶▶ 第15章 控制动画项目（JSlider 和 Timer 组件）
- ▶▶ 第16章 记事本（对话框组件）
- ▶▶ 第17章 拼图游戏（GUI 综合应用）

第 9 章 典型的图形用户界面（各种组件）

对于一个应用程序来说，一般会分成前台和后台两部分。后台主要用来实现数据的操作和用户需求的逻辑功能，而前台（应用程序界面）则用来实现与用户之间的友好交互。其实前台跟后台一样重要，因为拥有一个友好的界面，会使软件更容易推广。本章不仅介绍图形用户界面的基础知识而且还将讲解项目中常用的图形用户模块。

本章的学习目标如下：

- ❑ 掌握图形用户界面的基础知识；
- ❑ 绘制出项目中常用的图形用户模块。

9.1 Label 和 Button 的用户界面

本节将通过一个按钮和面板的用户界面，详细讲解使用按钮和面板组件的相关技巧。在该用户界面中，不仅实现了组件的显示，而且还将实现两种组件间的互动功能。

9.1.1 分析按钮和面板的用户界面

本节将以直观的方式向读者介绍用户界面所要实现的功能。这些功能包括初始化界面、按下左边按钮和按下右边按钮。

1. 初始化界面

当开始运行程序时，就会出现如图 9.1 所示的运行界面。在该界面中，面板上的文本为“检查哪个按钮被单击”，左右两边的按钮都处于可使用状态。

2. 按下左边按钮

当单击一下左边按钮时，就会出现如图 9.2 所示的运行界面。在该界面中，面板上的文本变成了“左边按钮被单击”，右边按钮处于不可用状态。这时如果单击右边按钮，界面将不会改变。



图 9.1 初始化界面



图 9.2 按下左边按钮

如果想使处于不可用状态的右边按钮改变状态，可以再次单击左边按钮，具体过程如图 9.3 所示。



图 9.3 运行过程

3. 按下右边按钮

当单击一下右边按钮时，就会出现如图 9.4 所示的运行界面。在该界面中，面板上的文本变成了“右边按钮被单击”，左边按钮处于不可用状态。这时如果单击左边按钮，界面将不会改变。



图 9.4 按下右边按钮

如果想使处于不可用状态的左边按钮改变状态，可以再次单击右边按钮，具体过程如图 9.5 所示。

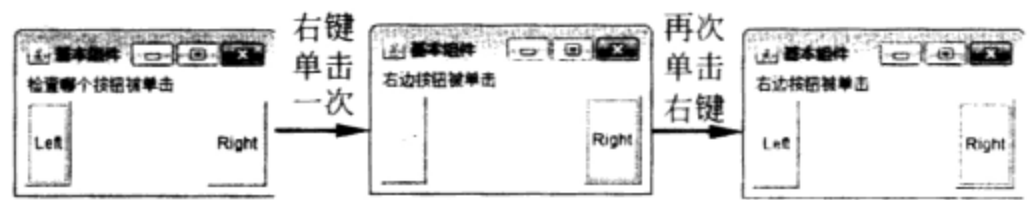


图 9.5 运行过程

9.1.2 按钮和面板的用户界面

BaseComponent.java 类用来实现包含有按钮和面板的用户界面，该用户界面中包含了 3 种组件：窗口组件、按钮组件和面板组件。该类的具体内容如代码 9.1 所示。

代码 9.1 按钮和面板类：BaseComponent.java

```
public class BaseComponent implements ActionListener {
    //创建成员变量
    Frame f;                                     //窗口对象
    Label result;                                //面板对象
    Button b1, b2;                               //按钮对象
    public static void main(String argv[]) {     //主方法
        new BaseComponent();
    }
}
```

```

public BaseComponent() { //构造函数
    //为窗口对象和面板对象赋值
    f = new Frame("基本组件");
    result = new Label("检查哪个按钮被单击");
    //为按钮对象 b1 赋值, 设置属性
    b1 = new Button("Left");
    b1.setSize(50, 100); //设置按钮的大小
    b1.setActionCommand("b1"); //设置按钮的 ActionCommand
    b1.addActionListener(this); //注册事件监听器
    b2 = new Button("Right");
    b2.setActionCommand("b2");
    b2.setSize(50, 100);
    b2.addActionListener(this);
    //添加两个按钮和面板到窗口上
    f.add(result, BorderLayout.NORTH);
    f.add(b1, BorderLayout.WEST);
    f.add(b2, BorderLayout.EAST);
    f.pack();
    f.setVisible(true);
}
@Override
public void actionPerformed(ActionEvent e) { //事件监听器
    String cmd = e.getActionCommand(); //获取按钮的 ActionCommand
    if (cmd.equals("b1")) { //当按钮 b1 被按下时
        b2.setEnabled(!b2.isEnabled());
        result.setText("左边按钮被单击");
    } else { //当按钮 b1 被按下时
        b1.setEnabled(!b1.isEnabled());
        result.setText("右边按钮被单击");
    }
}
}
}

```

【代码解析】

在上述代码中, 存在 4 个组件对象: 窗口、面板和两个按钮。由于为按钮注册了事件监听器, 所以当单击按钮 b1 时, 不仅修改另一个按钮 b2 的状态, 而且还在面板上输出相应文本。

9.1.3 组件 Button 和 Label 的基本知识

查看 API 帮助文档, 可以发现按钮的类 Button 和面板的类 Label 都继承于 Component 类。按钮 (Button) 的作用再熟悉不过, 而面板 (Label) 主要用来将文字显示用户界面。下面将分别介绍 Button 类和 Label 类。

1. Button类

Button 类拥有两个构造函数, 具体定义如下:

(1) Button()

创建一个无标签的按钮。

(2) Button(String label)

参数 label 为按钮的标签，即创建一个指定标签的按钮。

2. Label类

Label 类拥有 3 个构造函数，具体定义如下：

(1) Label()

创建一个具有默认值的面板。

(2) Label(String text)

参数 text 为面板的默认文本，即创建一个带有默认文本的面板。

(3) Label(String text, int alignment)

参数 alignment 为面板的对齐方式，即创建一个带有默认文本和指定对齐方式的面板。

Button 类的常用方法如表 9.1 所示。

表 9.1 Button类的常用方法

方法名称	功能
addActionListener()	注册事件监听器
removeActionListener()	移除事件监听器
getLabel()	获取按钮的标签
setLabel()	设置按钮的标签
getActionCommand()	获取按钮的 ActionCommand 字符串
setActionCommand()	设置按钮的 ActionCommand 字符串
isEnabled()	检查按钮是否为可用状态
setEnabled()	设置按钮的使用状态

9.2 复选框的用户界面

本节将通过一个选择的用户界面，详细讲解使用选择组件 Checkbox 的相关技巧。在该用户界面中，不仅实现了复选和单选组件的显示，而且还实现了与其他组件间的互动功能。

9.2.1 分析复选框的用户界面

本节将以直观的方式向读者介绍用户界面所要实现的功能。这些功能包括初始化界面、选择复选框和选择单选按钮。

1. 初始化界面

当开始运行程序时，就会出现如图 9.6 所示的运行界面。在该界面中，对于复选框只有“多选框 2”复选框被选择，对于单选按钮只有“单选按钮 2”被选择。

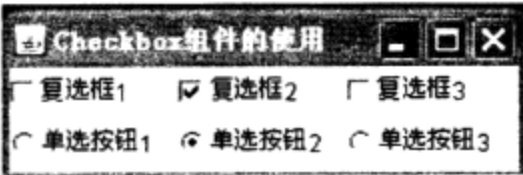


图 9.6 初始界面

2. 复选框

由于是复选框，所以可以进行多选，即如图 9.7 所示可以同时选择 3 个复选框。如果想取消对某复选框的选择，可以再次单击该复选框，如图 9.8 所示。

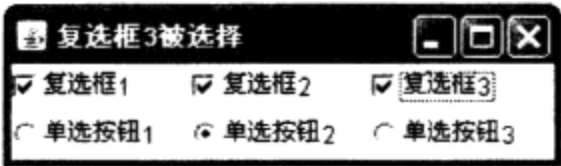


图 9.7 复选框的多选图

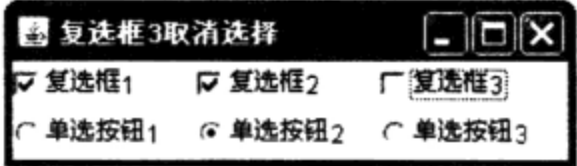


图 9.8 取消复选框的选择

3. 对于单选按钮

由于是单选按钮，所以不能多选，即如图 9.9 所示每次只能选择一个单选按钮。

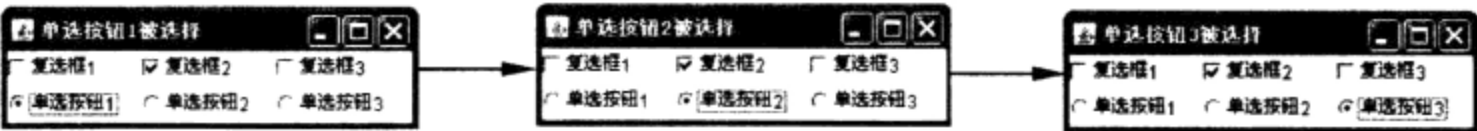


图 9.9 单选按钮具体情况

9.2.2 按钮和面板的用户界面

TestCheckbox.java 类用来实现包含有复选框和单选按钮的用户界面，该用户界面中包含窗口组件和复选框组件。该类的具体内容如代码 9.2 所示。

代码 9.2 复选框组件类：TestCheckbox.java

```
public class TestCheckbox implements ItemListener {
    //创建成员变量
    Frame f;
    //创建两组复选框对象
    Checkbox cb[] = new Checkbox[3];
    Checkbox cbg[] = new Checkbox[3];
    CheckboxGroup cbg1 = new CheckboxGroup();
    public static void main(String argv[]) {
        new TestCheckbox();
    }
    public TestCheckbox() {
        f = new Frame("Checkbox 组件的使用");
        f.setLayout(new GridLayout(2, 3));
        //为 cb 数组赋值
        cb[0] = new Checkbox("复选框 1");
        //创建窗口对象
        //创建一个复选框组
        //创建 TestCheckbox 对象
        //构造函数
        //为窗口对象赋值
        //设置窗口的布局
    }
}
```

```

        cb[1] = new Checkbox("复选框 2", true);
        cb[2] = new Checkbox();
        cb[2].setLabel("复选框 3");
        //为 cbg 数组赋值
        cbg[0] = new Checkbox("单选按钮 1");
        cbg[0].setCheckboxGroup(cbg1);
        cbg[1] = new Checkbox("单选按钮 2", true);
        cbg[1].setCheckboxGroup(cbg1);
        cbg[2] = new Checkbox("单选按钮 3", cbg1, false);
        //通过循环为 cb 数组对象注册事件
        for (int i = 0; i < 3; i++) {
            cb[i].addItemListener(this);
            f.add(cb[i]);
        }
        //通过循环为 cbg 数组对象注册事件
        for (int i = 0; i < 3; i++) {
            cbg[i].addItemListener(this);
            f.add(cbg[i]);
        }
        f.pack();
        f.setVisible(true);                                //显示窗口
    }
    public void itemStateChanged(ItemEvent e) {            //实现事件监听器
        Checkbox ch = (Checkbox) e.getSource();           //获取发生事件的组件对象
        String label = ch.getLabel();                       //获取组件的标签
        if (e.getStateChange() == ItemEvent.SELECTED)      //判断组件的状态
            f.setTitle(label + "被选择");
        else
            f.setTitle(label + "取消选择");
    }
}




```




【代码解析】

在上述代码中，存在 7 个组件对象：窗口、3 个复选框和 3 个单选按钮。由于为 3 个复选框和单选按钮注册了事件监听器，所以这些组件的状态发生改变时，就会调用事件监听器中的方法。

9.2.3 组件 Checkbox 和 CheckboxGroup 的基本知识

查看 API 帮助文档，可以发现复选框的类 `Checkbox` 和 `CheckboxGroup`。前者主要用来实现复选框，而后者主要用来实现单选按钮。`CheckboxGroup` 对象中可以包含好几个 `Checkbox` 对象，但是对属于同一个 `CheckboxGroup` 中的 `Checkbox` 对象，同一时间内只能有一个 `Checkbox` 对象被选择。

所谓复选框，其外形是一个小正方形（），单击该小正方形后就会打勾（），表示其被选择（checked）。再次单击小正方形时打勾就会消失（），表示取消了选择（unchecked）。对于复选框，在同一时间内，可以同时选择好几个 `Checkbox` 对象。

所谓单选按钮，其外形是圆形（），单击该圆形时会出现黑点（），表示其被选择（checked）。再次单击圆形时黑点消失（），表示取消了选择（unchecked）。对于单选按钮，在同一个时间内，只能有一个 `Checkbox` 对象被选择。

1. Checkbox类

Checkbox 类拥有 4 个构造函数，具体定义如下：

- (1) Checkbox()
创建一个空白的复选框。
- (2) Checkbox(String label)
参数 label 为复选框的标签，即创建一个指定标签的复选框。
- (3) Checkbox(String label, boolean state)
参数 state 为复选框的状态，即创建一个指定标签和状态的复选框。
- (4) Checkbox(String label, boolean state, CheckboxGroup group)
参数 group 为复选框的复选框组，即创建一个指定标签、状态和复选框组的复选框。

2. CheckboxGroup类

CheckboxGroup 类拥有一个构造函数，具体定义如下：

CheckboxGroup()

创建一个具有默认值的复选框组实例。

Checkbox 类的常用方法如表 9.2 所示，CheckboxGroup 类的常用方法如表 9.3 所示。

表 9.2 Checkbox类的常用方法

方法名称	功 能
addItemListener()	注册事件监听器
getLabel()	获取复选框的标签
setLabel()	设置复选框的标签
getState()	获取复选框的状态
setState()	设置复选框的状态

表 9.3 CheckboxGroup类的常用方法

方法名称	功 能
getSelectedCheckbox()	取得该组中被选的 Checkbox 对象
setSelectedCheckbox()	设置该组中被选的 Checkbox 对象

9.3 下拉菜单和列表的用户界面

本节将通过一个选择的用户界面，详细讲解使用下拉菜单组件 Choice 和列表组件 List 的相关技巧。在该用户界面中，不仅实现了下拉菜单和列表组件的显示，而且还实现了与其他组件间的互动功能。

9.3.1 分析下拉菜单和列表的用户界面

本节将以直观的方式向读者介绍用户界面所要实现的功能。这些功能包括初始化界面、选择下拉菜单和选择列表。

1. 初始化界面

当开始运行程序时，需要输入如图 9.10 所示的参数，这时就会出现如图 9.11 所示的初始界面，输出窗口如图 9.12 所示。

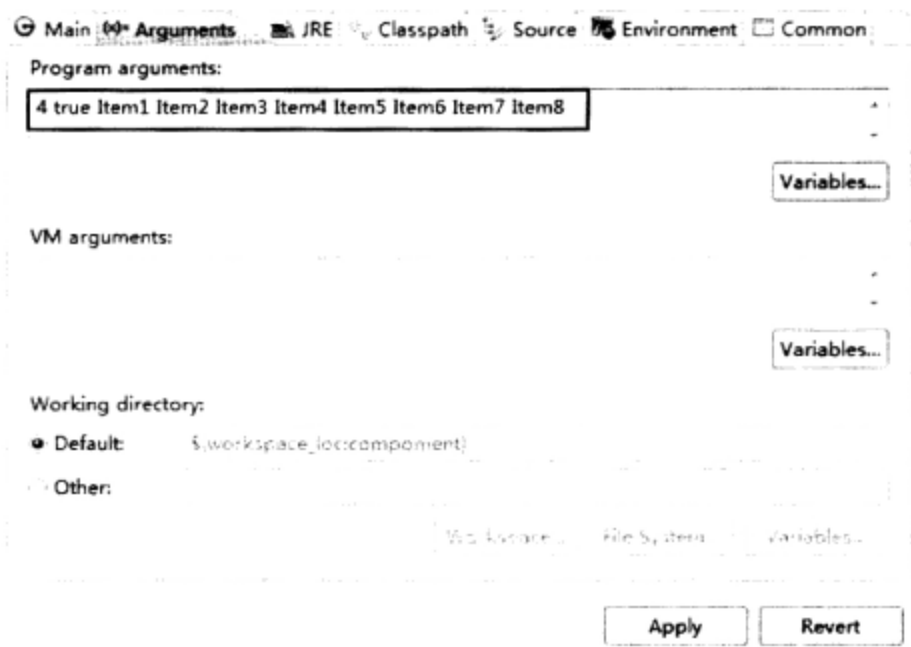


图 9.10 相关参数

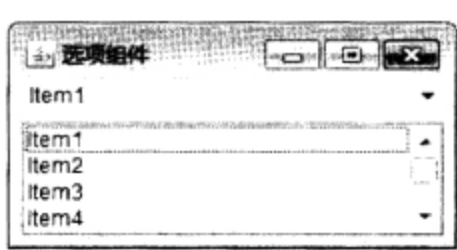


图 9.11 初始界面



图 9.12 输出窗口

在输入的参数中，第 1 个参数为列表同时要显示的选项数，第 2 个参数为列表是否允许多选，剩下的参数为列表和下列菜单的选项。

2. 选择下拉菜单

在初始界面中，如果想查看下拉菜单的所有选项，可以通过单击 ▼ 按钮就会显示出所有的选项，如图 9.13 所示。这时如果任意选择一个选项（例如 Item4，如图 9.14 所示），输出窗口就会输出如图 9.15 所示的信息。

3. 选择列表

在初始界面中，如果想查看列表的选项，可以通过移动滚动条来查看所有的选项，如图 9.16 所示。由于传入的参数为 true，所以对于列表中的选项，可以单选也可以多项。如

果只选择 Item4 选项，输出窗口也会输出相应信息，如图 9.17 所示。如果同时选择 Item5 和 Item6 选项，输出窗口也会输出相应信息，如图 9.18 所示。

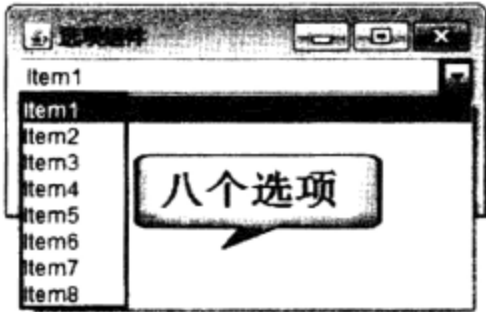


图 9.13 下拉菜单选项

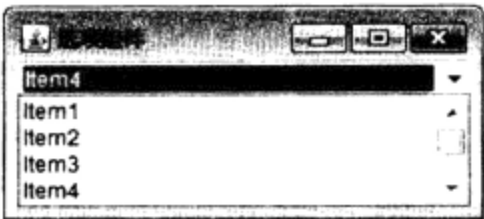


图 9.14 选择相应选项

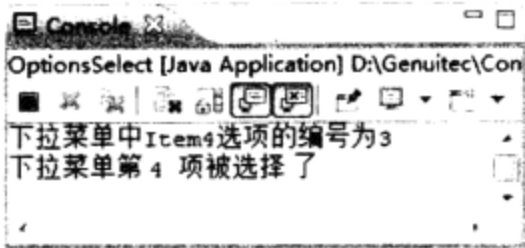


图 9.15 输出窗口信息

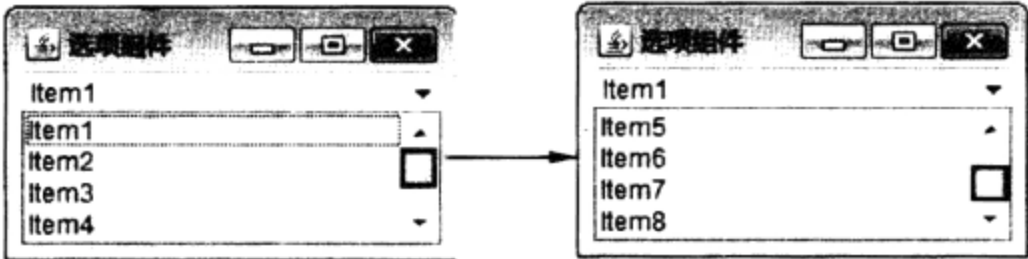


图 9.16 查看所有的选项

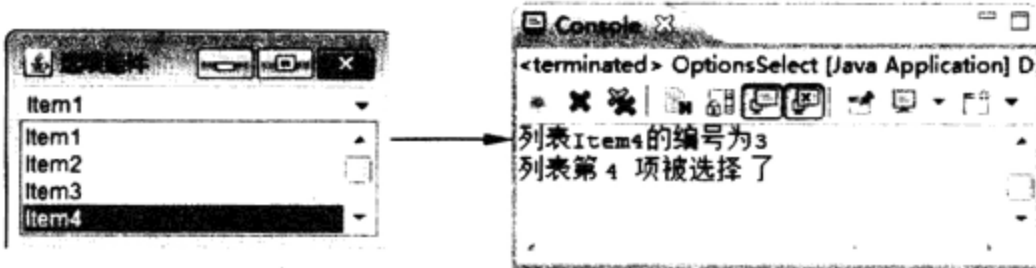


图 9.17 单选情况

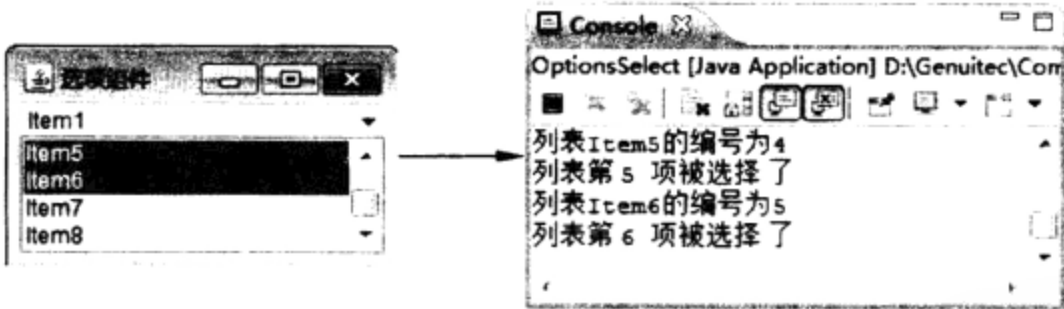


图 9.18 多选情况

9.3.2 下拉菜单和列表的用户界面

OptionsSelect.java 类用来实现包含有下拉菜单和列表的用户界面，该用户界面中包含下拉菜单组件（Choice）和列表（List）组件。该类的具体内容如代码 9.3 所示。

代码 9.3 下拉菜单和列表组件类：OptionsSelect.java

```
public class OptionsSelect implements ItemListener {
    //创建 3 个成员变量
```



```

Frame f;                                //窗口对象
List ls;                                //列表对象
Choice ch;                              //下拉菜单对象
public static void main(String argv[]) {
    new OptionsSelect(argv);             //创建 OptionsSelect 对象
}
public OptionsSelect(String argv[]) {    //构造函数
    f = new Frame("选项组件");           //为窗口对象赋值
    //列表组件 ls 的赋值和设置
    ls = new List(Integer.parseInt(argv[0]), Boolean.valueOf(argv[1])
        .booleanValue());                //为列表对象赋值
    ls.addItemListener(this);            //为列表对象注册监听器
    for (int i = 2; i < argv.length; i++) //为列表添加选项
        ls.add(argv[i]);
    //下拉菜单组件 ch 的赋值和设置
    ch = new Choice();                   //为下拉菜单对象赋值
    //t 通过匿名类方式为 ch 对象注册事件监听器
    ch.addItemListener(new ItemListener() {
        public void itemStateChanged(ItemEvent e) {
            Choice c = (Choice) e.getSource();
            System.out.println("下拉菜单中" + c.getSelectedItem() + "
                选项的编号为"
                + c.getSelectedIndex());
            System.out.println("下拉菜单 " + (c.getSelectedIndex() + 1)
                + " 项被选择了");
        }
    });
    for (int i = 2; i < argv.length; i++) //为对象 ch 添加选项
        ch.add(argv[i]);
    f.add(ls, BorderLayout.SOUTH);        //向窗口对象添加组件
    f.add(ch, BorderLayout.NORTH);
    f.pack();
    f.setVisible(true);                  //显示窗口
    //输出相应信息
    System.out.println("下拉菜单一共有 " + ch.getItemCount() + " 项");
    System.out.println("列表一共有 " + ls.getItemCount() + " 项");
}
public void itemStateChanged(ItemEvent e) { //实现 itemStateChanged()
    List l = (List) e.getSource();
    int index[] = l.getSelectedIndexes();
    String str[] = l.getSelectedItems();
    for (int i = 0; i < index.length; i++) {
        System.out.println("列表" + str[i] + "的编号为" + index[i]);
        System.out.println("列表第" + (index[i] + 1) + " 项被选择了");
    }
}
}
}

```

【代码解析】

在上述代码中，存在 3 个组件对象：窗口、下拉菜单和列表。由于下拉菜单和列表注册了事件监听器，所以这些组件的状态发生改变时，就会调用事件监听器中的方法。

9.3.3 Choice 和 List 组件的基本知识

虽然复选框和单选框也可以实现选择的功能，但是当所选择的选项太多时，就可能出现窗口中放不下这么多的选项，或者客户在拥有一堆选项的窗口上不容易进行操作的情况。为了节省窗口空间又能放入许多选项且方便用户选择，Sun 公司提供了一个下拉菜单组件（Choice）。

虽然 Choice 组件可以实现许多项的选择，但是同一时间里只能有一个选项被选择。为了解决该问题，Sun 公司又提供了一个名叫列表（List）的组件，该组件不仅提供了类似于 Choice 组件的功能，而且在同一时间里还能进行多选。

查看 API 帮助文档，可以发现下拉菜单的类 Choice 和列表 List。前者主要用来实现下拉菜单，而后者主要用来实现列表。这两个类的相关语法具体如下。

1. Choice类

Choice 类拥有一个构造函数，具体定义如下：

```
Choice()
```

创建一个下拉式菜单实例。

2. List类

List 类拥有 3 个构造函数，具体定义如下：

```
List()
```

意思为创建一个具有默认值的复选框组实例。

```
List(int rows)
```

参数 rows 为行数，创建一个指定可视行数初始化的列表。

```
List(int rows, boolean multipleMode)
```

参数 multiple 为是否允许多选，创建一个指定可视行数初始化的列表。

Choice 类的常用方法如表 9.4 所示。List 类的常用方法如表 9.5 所示。

表 9.4 Choice类的常用方法

方 法 名 称	功 能
Add()	添加选项
addItemListener()	注册相关的事件监听器
getItem()	取得所指定 index 选项的内容
getItemCount()	取得选项总数
getSelectedIndex()	取得当前被选择选项的 index

续表

方法名称	功 能
getSelectedItem()	取得当前被选择选项的内容
insert()	在指定位置上插入新的选项
remove()	删除指定内容或指定位置的选项
removeAll()	删除所有选项
select()	选择指定内容的选项

表 9.5 List类的常用方法

方法名称	功 能
add()	添加选项
addActionListener()	ActionEvent 事件监听器
addItemListener()	ItemEvent 事件监听器
deselect()	删除指定位置的选项
getItem()	获取指定 index 的选项内容
getItemCount()	获取选项的总数
getRow()	获取被选择项的数目
getSelectedIndex()	获取被选择项的 index
getSelectedIndexes()	获取所有被选择项的 index
getSelectedItem()	获取被选择项的内容
getSelectedItems()	获取所有被选择项的内容
isIndexSelected()	检查指定位置的选项是否被选择
isMultipleMode()	检查是否为多选模式
replaceItem()	替换指定位置的选项
remove()	删除指定内容或位置的选项
removeAll()	删除所有选项
select()	选择指定位置的选项
selMultipleMode()	设置单选或多选模式

9.4 输入的用户界面

本节将通过一个输入的用户界面，详细讲解使用文本框组件 TextField 和文本域组件 TextArea 的相关技巧。在该用户界面中，不仅将实现文本框和文本域组件的显示，而且还将实现与其他组件间的互动功能。

9.4.1 分析输入的用户界面

本节将以直观的方式向读者介绍用户界面所要实现的功能。这些功能包括初始化界面、选择复选框和选择单选框。

1. 初始化界面

当开始运行程序时，就会出现如图 9.19 所示的运行界面。在该界面中，存在 2 个显示面板、1 个文本框和 4 个文本域。

2. 文本框的操作

在初始界面中，当在文本框中输入相应信息时，第一块显示面板就会显示出相同的信息，具体过程如图 9.20 所示。

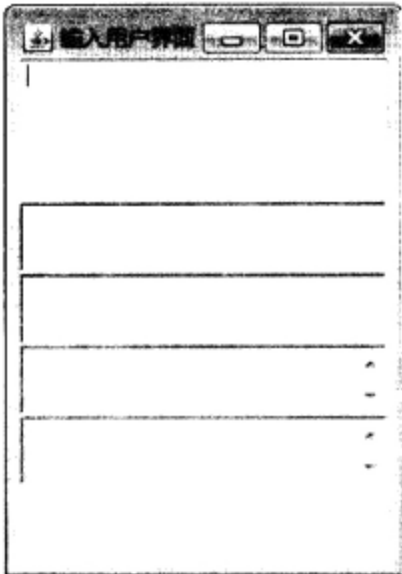


图 9.19 初始界面

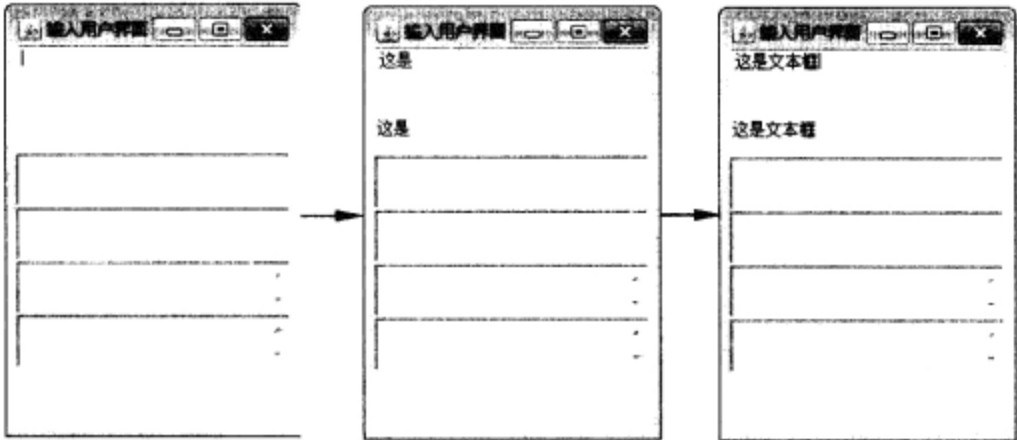


图 9.20 文本框具体过程

3. 文本域的操作

在初始界面中，当在文本域中输入相应信息时，第 2 块显示面板就会显示出相同的信息。第 1 个文本域的操作过程如图 9.21 所示，即当文本域里的文字超过文本域范围时也不出现滚动条。第 2 个文本域的操作具体过程如图 9.22 所示，即如果想查看超过文本域范围外的文字时，可以移动水平滚动条。第 3 个文本域的操作具体过程如图 9.23 所示，即如果想查看超过文本域范围外的文字时，可以移动垂直滚动条。第 4 个文本域的操作具体过程如图 9.24 所示，即如果想查看超过文本域范围外的文字时，可以移动垂直或水平滚动条。

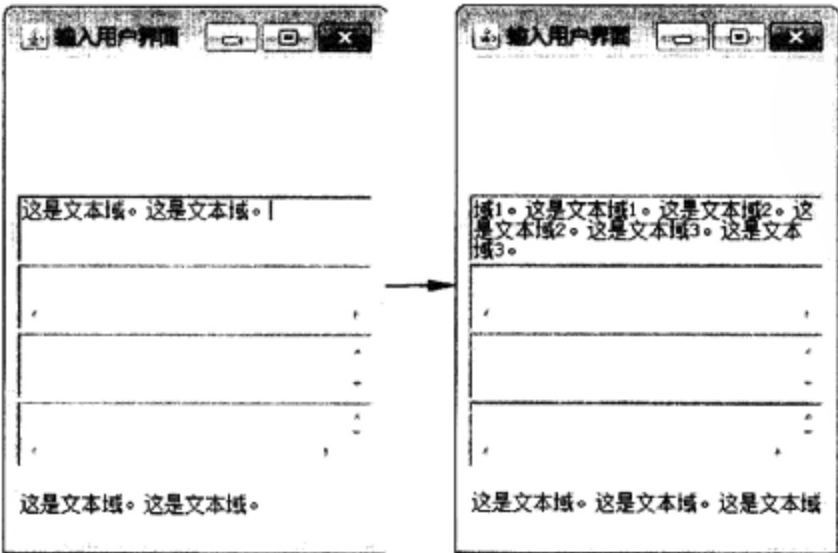


图 9.21 无滚动条文本域

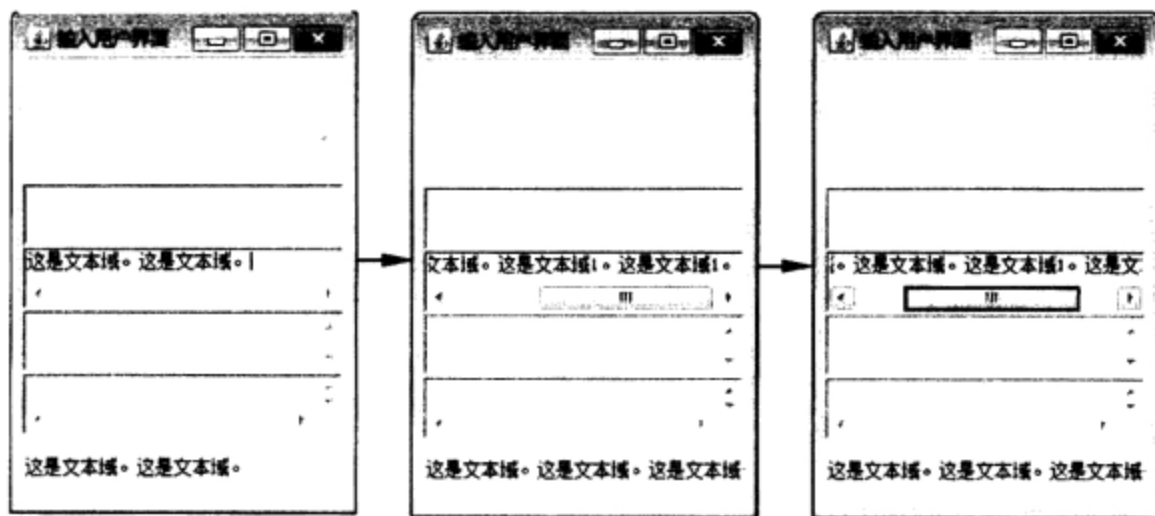


图 9.22 拥有水平滚动条文本域

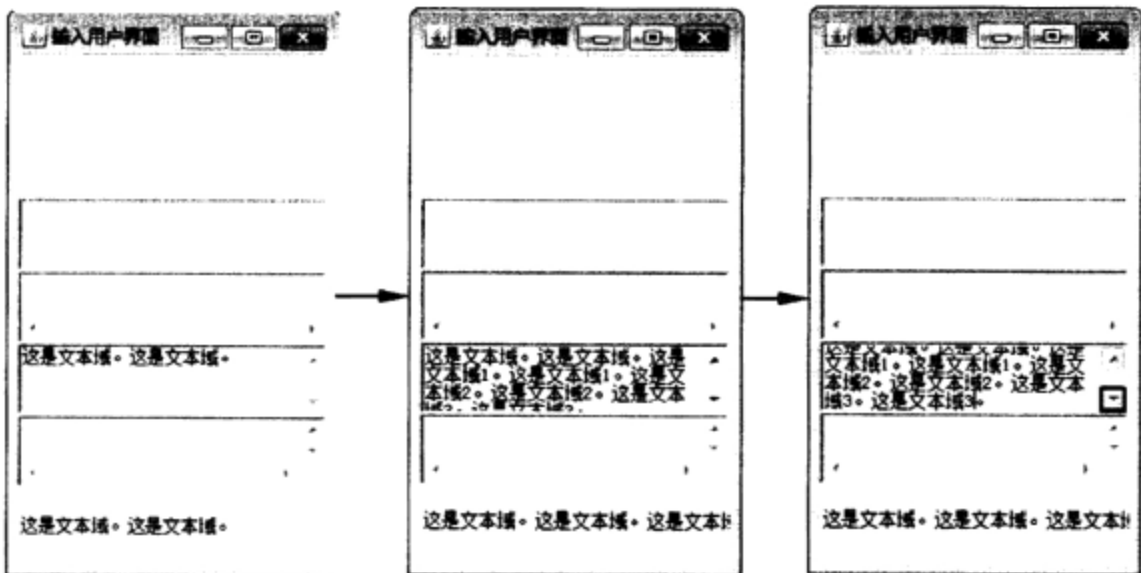


图 9.23 拥有垂直滚动条文本域

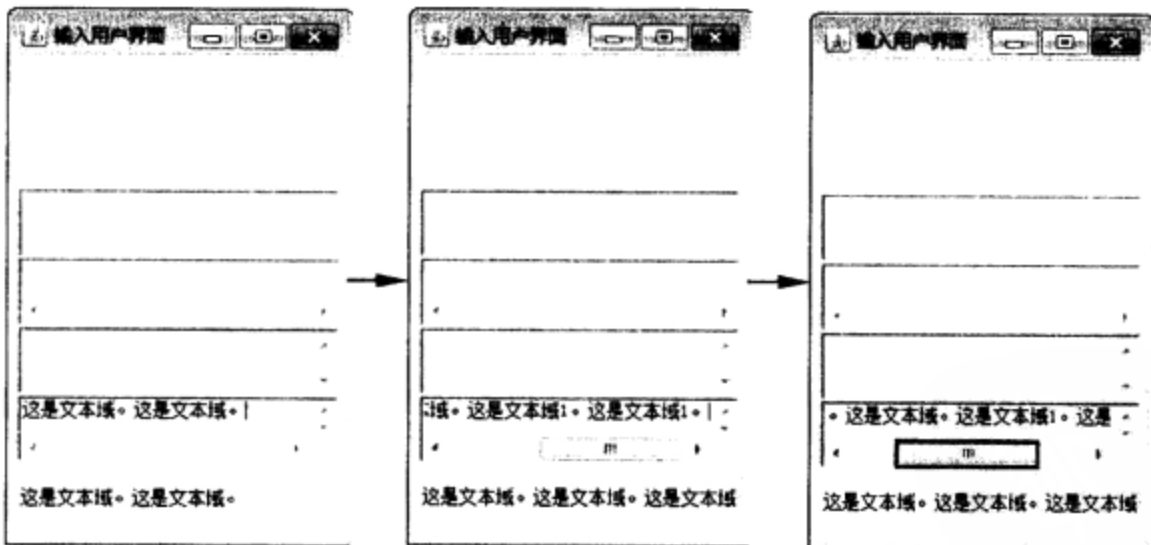


图 9.24 拥有水平和垂直滚动条文本域

9.4.2 输入的用户界面

Input.java 类用来实现包含有复选框和单选框的用户界面，该用户界面中包含窗口组件和复选框组件。该类的具体内容如代码 9.4 所示。

代码 9.4 复选框组件类: Input.java

```

public class Input implements TextListener, ActionListener {
    //创建成员变量
    Frame f; //定义窗口变量
    Label lb, lbl; //定义面板变量
    TextField tf; //定义输入框变量
    TextArea ta1, ta2, ta3, ta4; //定义 TextArea 变量
    public static void main(String argv[]) { //主方法
        new Input (); //创建 Input 对象
    }
    public Input () { //构造函数
        f = new Frame("输入用户界面"); //为窗口对象赋值
        f.setLayout(new GridLayout(7, 1)); //设置窗口布局管理器
        //创建和设置 TextField 对象
        tf = new TextField("", 20); //为 TextField 对象赋值
        tf.addTextListener(this); //添加事件监听器
        tf.addActionListener(this);
        lb = new Label(); //为面板对象 lb 赋值
        f.add(tf); //为窗口添加对象
        f.add(lb);
        //创建和设置 TextArea 对象
        ta1 = new TextArea("", 2, 10, TextArea.SCROLLBARS_NONE);
        ta2 = new TextArea("", 2, 10, TextArea.SCROLLBARS_HORIZONTAL_ONLY);
        ta3 = new TextArea("", 2, 10, TextArea.SCROLLBARS_VERTICAL_ONLY);
        ta4 = new TextArea("", 2, 10, TextArea.SCROLLBARS_BOTH);
        //为每个 TextArea 对象添加事件监听器
        ta1.addTextListener(new TextListener() { //为对象 ta1 添加事件监听器
            public void textValueChanged(TextEvent e) {
                lbl.setText(ta1.getText());
            }
        });
        ta2.addTextListener(new TextListener() { //为对象 ta2 添加事件监听器
            public void textValueChanged(TextEvent e) {
                lbl.setText(ta2.getText());
            }
        });
        ta3.addTextListener(new TextListener() { //为对象 ta3 添加事件监听器
            public void textValueChanged(TextEvent e) {
                // TODO Auto-generated method stub
                lbl.setText(ta3.getText());
            }
        });
        ta4.addTextListener(new TextListener() { //为对象 ta4 添加事件监听器
            public void textValueChanged(TextEvent e) {
                // TODO Auto-generated method stub
                lbl.setText(ta4.getText());
            }
        });
    }
}

```



```

    );
    lbl = new Label(); //为面板对象 lbl 赋值
    //为窗口对象 f 添加对象
    f.add(ta1);
    f.add(ta2);
    f.add(ta3);
    f.add(ta4);
    f.add(lbl);
    f.pack(); //显示窗口
    f.setVisible(true);
}
public void textValueChanged(TextEvent e) {
    //实现 textValueChanged() 方法
    lb.setText(tf.getText());
}
public void actionPerformed(ActionEvent e) {
    //实现 actionPerformed () 方法
    tf.setText("");
}
}

```

【代码解析】

在上述代码中，存在 8 个组件对象，分别是 1 个窗口、1 个 TextField、4 个 TextArea 和 2 个面板。由于为 3 个复选框和单选按钮注册了事件监听器，所以这些组件的状态发生改变时，就会调用事件监听器中的方法。

9.4.3 TextField 和 TextArea 组件的基本知识

Label 组件的出现使得应用程序的输出可以显示在窗口上，脱离了只能输出到命令窗口的尴尬。而 TextField 和 TextArea 组件的出现，使应用程序的输入脱离了在命令窗口输入参数等形式。对于图形用户界面来说，为了使用户与应用程序的交换更容易，必须输入组件。这两个组件间的关系如图 9.25 所示。

查看 API 帮助文档，可以发现输入框的类 TextField 和 TextArea。前者主要用来实现输入框，而后者主要用来实现带有滚动条的输入框。这两个类的相关语法具体如下。

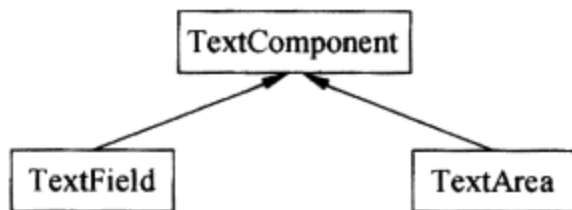


图 9.25 类的关系

1. TextField 类

TextField 类拥有 4 个构造函数，具体定义如下。

(1) TextField()

创建一个文本框实例。

(2) TextField(int columns)

参数 columns 为文本的列数，创建一个指定列数的文本实例。

(3) TextField(String text)

参数 `text` 为初始化文本，创建一个具有初始文本的文本实例。

(4) `TextField(String text, int columns)`

创建一个具有初始文本和指定列数的文本实例。

2. `TextArea`类

`TextArea` 类拥有 5 个构造函数，具体定义如下。

(1) `TextArea()`

创建一个文本域实例。

(2) `TextArea(int rows, int columns)`

参数 `rows` 表示允许文字显示的行数，参数 `columns` 表示允许文字显示的列数，该函数为创建一个指定行数和列数的文本域实例。

(3) `TextArea(String text)`

参数 `text` 为初始化文本，该函数为创建一个拥有初始值的文本域实例。

(4) `TextArea(String text, int rows, int columns)`

创建一个指定行数、列数和初始化文本的文本域实例。

(5) `TextArea(String text, int rows, int columns, int scrollbars)`

参数 `scrollbars` 为文本框的滚动条，该函数为创建一个指定行数、列数、初始化文本和滚动条的文本域实例。

`TextComponent` 类的常用方法如表 9.6 所示，`TextField` 类的常用方法如表 9.7 所示，`TextArea` 类的常用方法如表 9.8 所示，

表 9.6 `TextComponent`类的常用方法

方法名称	功 能
<code>addTextListener()</code>	<code>TextEvent</code> 监听器
<code>getCaretPosition()</code>	取得文字插入点的位置
<code>getSelectedText()</code>	取得被选择的文字
<code>getSelectionStart()</code>	取得所选文字的起点位置
<code>getSelectionEnd()</code>	取得所选文字的终点位置
<code>getText()</code>	取得组件中所显示的文字
<code>isEditable()</code>	判断组件是否可以输入文字
<code>select()</code>	选择指定起点和终点位置的文字
<code>selectAll()</code>	选择所有文字
<code>setCaretPosition()</code>	设置文字插入点的位置
<code>setEditable()</code>	设置组件是否可以输入文字
<code>setSelectionStart()</code>	设置选择文字的起点位置
<code>setSelectionEnd()</code>	设置选择文字的终点位置
<code>setText()</code>	设置组件显示的文字

表 9.7 `TextField`类的常用方法

方法名称	功 能
<code>echoCharIsSet()</code>	检查是否有设置响应字符

续表

方 法 名 称	功 能
getColumns()	取得显示字符的行数
getEchoChar()	取得所设置的响应字符
setColumns()	设置显示字符的行数
setEchoChar()	设置响应字符
setText()	设置组件中显示的文字

表 9.8 TextArea类的常用方法

方 法 名 称	功 能
append()	检查是否有设置响应字符
getColumns()	取得显示字符的行数
getRows()	取得显示字符的列数
setColumns()	设置显示字符的行数
setRows()	设置显示字符的列数
getScrollbarVisibility()	取得目前所使用的滚动条模式
insert()	将指定文字插入指定的位置
replaceRange()	将指定范围中的文字替换成指定的文字
setColumns()	设置显示字符的行数
setRows()	设置显示字符的列数

9.5 滚动条的用户界面

本节将通过一个滚动条的用户界面，详细讲解使用滚动条组件 Scrollbar 和滚动面板 ScrollPane 的相关技巧。在该用户界面中，不仅实现了滚动条组件的显示，而且还实现了与其他组件间的互动功能。

9.5.1 分析滚动条的用户界面

本节将以直观的方式向读者介绍用户界面所要实现的功能。这些功能包括初始化界面、选择下拉菜单和选择列表。

1. 滚动条初始化界面

当开始运行程序时，会出现如图 9.26 所示的运行界面。在该界面中，存在两个滚动条，分别是垂直滚动条和水平滚动条，同时还有一个按钮。

2. 滚动条的操作

在初始界面中，如果想在水平方向移动按钮，可以用鼠标移动水平滚动条，具体过程如图 9.27 所示。如果想在垂直方向移动按钮，可以用鼠标移动垂直滚动条，具体过程如图

9.28 所示。



图 9.26 初始界面

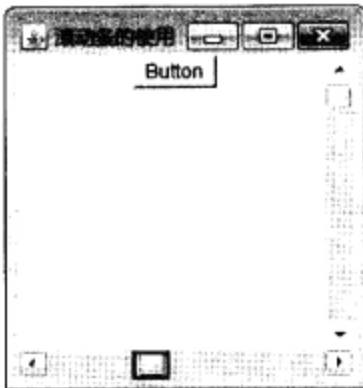


图 9.27 水平移动



图 9.28 垂直移动

3. 滚动面板的初始界面

当开始运行程序时，如果传入的参数为 3，则会出现如图 9.29 所示的运行界面。如果传入的参数为 6，则会出现如图 9.30 所示的运行界面。

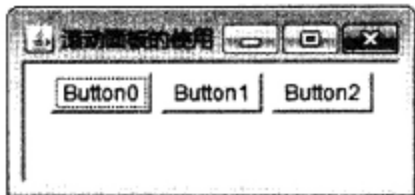


图 9.29 无滚动条初始界面

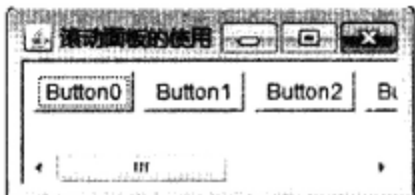


图 9.30 带有滚动条的初始界面

根据运行的界面可以发现，用户界面会根据参数的大小来决定是否产生滚动条。

4. 滚动面板的操作

在初始界面中，如果想查看水平方向超出界面的部分，可以用鼠标移动水平滚动条，具体过程如图 9.31 所示。

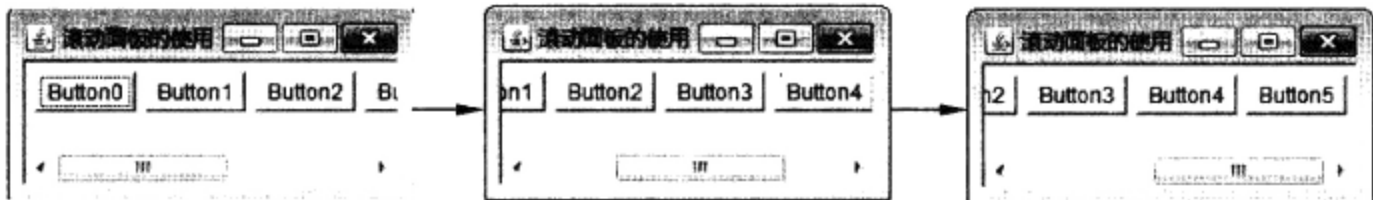


图 9.31 运行过程

9.5.2 滚动条的用户界面

ScrollbarTest.java 类用来实现包含有滚动条的用户界面，该用户界面中包含了两种滚动条组件，分别是垂直滚动条和水平滚动条。该类的具体内容如代码 9.5 所示。

代码 9.5 Scrollbar 组件类：ScrollbarTest.java

```
public class ScrollbarTest implements AdjustmentListener {
    //创建成员变量
    Frame f;
    //创建 Frame 对象
```

```

Button btn; //创建按钮对象
Panel p; //创建 Panel 对象
Scrollbar HSB, VSB; //创建滚动条对象
int x = 0, y = 0; //创建坐标变量
public static void main(String argv[]) { //创建 ScrollbarTest 对象
    new ScrollbarTest();
}
public ScrollbarTest() { //构造函数
    f = new Frame("滚动条的使用"); //为对象 f 赋值
    p = new Panel(null); //为对象 p 赋值
    //为对象 btn 赋值和设置
    btn = new Button("Button"); //为对象 btn 赋值
    btn.setSize(50, 20); //设置大小
    btn.setLocation(x, y); //设置放置的位置
    //为滚动条赋值和设置
    HSB = new Scrollbar(Scrollbar.HORIZONTAL, 0, 10, 0, 200);
    VSB = new Scrollbar(Scrollbar.VERTICAL, 0, 10, 0, 200);
    HSB.addAdjustmentListener(this); //为滚动条注册事件
    VSB.addAdjustmentListener(this);
    p.add(btn); //添加按钮到 p 组件上
    f.add(p, BorderLayout.CENTER); //添加组件 p 到窗口上
    //添加滚动条到窗口上
    f.add(HSB, BorderLayout.SOUTH);
    f.add(VSB, BorderLayout.EAST);
    f.setSize(250, 250); //设置窗口的大小
    f.setVisible(true); //设置窗口的显示
}

public void adjustmentValueChanged(AdjustmentEvent e) {
    Scrollbar sb = (Scrollbar) e.getSource(); //获取事件源对象
    //判断滚动条的类型
    if (sb.getOrientation() == Scrollbar.HORIZONTAL) //当为水平滚动条时
        x = sb.getValue();
    else
        y = sb.getValue();
    btn.setLocation(x, y); //设置按钮的位置
}
}

```

【代码解析】

在上述代码中，窗口中有一个 Panel 对象和两个 Scrollbar 对象，而 Button 对象却处于 Panel 对象上。由于需要通过 Scrollbar 对象控制 Button 对象的放置位置，所以 Panel 对象没有设置布局管理器，而 Button 对象需要设置大小和通过 setLocation()方法进行放置位置的设置。

ScrollbarTest.java 类用来实现包含有滚动面板的用户界面，该类的具体内容如代码 9.6 所示。

代码 9.6 ScrollPane 组件类：ScrollPaneTest.java

```

public class ScrollPaneTest {
    public static void main(String argv[]) {
        Frame f = new Frame("滚动面板的使用"); //创建窗口对象
        int n = Integer.parseInt(argv[0]); //获取输入参数
        Panel p = new Panel(new FlowLayout()); //创建 Panel 对象
    }
}

```

```

        Button btn[] = new Button[n];                //创建按钮对象
        //设置按钮
        for (int i = 0; i < n; i++) {
            btn[i] = new Button("Button" + i);
            p.add(btn[i]);
        }
        //创建滚动面板
        ScrollPane sp = new ScrollPane(ScrollPane.SCROLLBARS_AS_NEEDED);
        sp.add(p);                                    //添加面板到 sp 里
        f.add(sp);                                    //添加 sp 对象到窗口上
        f.pack();
        f.setVisible(true);                          //显示窗口
    }
}

```

【代码解析】

在上述代码中，窗口中有一个 Panel 对象、一个 ScrollPane 对象及许多 Button 对象，它们之间的关系是对象，而 Button 对象处于 Panel 对象上，Panel 对象处于 ScrollPane 对象上，最后 ScrollPane 对象放置在窗口对象上。由于 Panel 对象使用了流式管理器，所以 Panel 对象数目决定了 Panel 对象的大小，而 Panel 对象的大小决定 ScrollPane 对象是否出现滚动条。

9.5.3 滚动组件的基本知识

当选项的内容太多时，可以使用 Choice 和 List 组件来节省界面空间。当程序中要显示的内容远远大于窗口或屏幕的大小时，那么如何查看超出窗口或屏幕的内容呢？Sun 公司提供了名为 Scrollbar 组件来显示想要看的内容，虽然该组件可以改变其他组件的位置，但是如果想让容器组件自己决定是否该产生滚动条，其就显得有点无能为力了。为了解决这个问题，Sun 公司又提供了一个名为 ScrollPane 的组件，该组件单独使用时，与组件 Panel 的功能一样，但是如果内容超过该组件的大小时，就会自动产生滚动条。

查看 API 帮助文档，可以发现滚动条的类 Scrollbar 和 ScrollPane。前者主要用来实现滚动条，而后者主要用来实现滚动面板。这两个类的相关语法说明如下。

1. Scrollbar 类

Scrollbar 类拥有 3 个构造函数，具体定义如下。

(1) Scrollbar()

该函数为创建一个垂直滚动条。

(2) Scrollbar(int orientation)

参数 orientation 为滚动条的方向，该函数为创建一个具有指定方向的菜单项实例。

(3) Scrollbar(int orientation, int value, int visible, int minimum, int maximum)

参数 value 为滚动条的滚动初始值；参数 visible 为单击滚动条时其一次要滚动的数值大小；参数 minimum 和 maximum 为滚动条滚动的最小值和最大值。

2. ScrollPane类

ScrollPane 类拥有两个构造函数，具体定义如下：

(1) ScrollPane()

该函数为创建一个滚动面板实例。

(2) ScrollPane(int scrollbarDisplayPolicy)

参数 scrollbarDisplayPolicy 为滚动条的方向，该函数为创建一个具有指定方向的滚动面板实例。

类 Scrollbar 的常用方法如表 9.9 所示。

表 9.9 Scrollbar类常用的方法

方 法 名 称	功 能
addAdjustmentListener()	添加选项
getMaximum()	注册相关的事件监听器
setMaximum()	取得所指定 index 选项的内容
getMinimum()	取得选项总数
setMinimum()	取得当前被选择选项的 index
getOrientation()	取得当前被选择选项的内容
setOrientation()	在指定位置上插入新的选项
getValue()	删除指定内容或指定位置的选项
setValue()	删除所有选项

9.6 菜单的用户界面

本节将通过一个菜单系统的用户界面，详细讲解使用菜单组件的相关技巧。在该用户界面中，不仅将实现各种菜单组件的显示，而且还将实现与其他组件间的互动功能。

9.6.1 分析菜单组件的用户界面

本节将以直观的方式向读者介绍用户界面所要实现的功能。这些功能包括初始化界面、选择下拉菜单和选择列表。

1. 初始化界面

当开始运行程序时，会出现如图 9.32 所示的运行界面。在该界面中存在一个菜单栏，在该菜单栏里有两个菜单（Menu1 和 Help），显示面板显示 No menu item selected!。

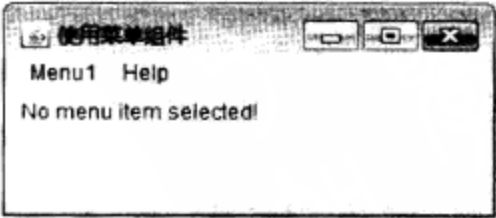


图 9.32 初始界面

2. Menu1菜单

当选择菜单 Menu1 时，会弹出如图 9.33 所示的菜单选项。选择 Item1 菜单项时会出现如图 9.34 所示的菜单项，如果选择了 Item2 菜单项，同时会在输出窗口中显示出现相应的信息。对于 Item2 菜单项，如果想显示如图 9.35 所示信息，除了直接选择该菜单选项，还可以通过快捷键 Ctrl+A 来实现。当选择 Item3 菜单项时，不仅菜单前的选择状态会改变，而且还会显示出相应的信息，如图 9.36 所示。

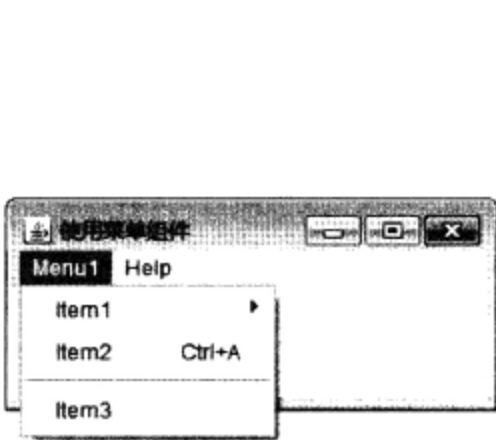


图 9.33 菜单 Menu1 选项

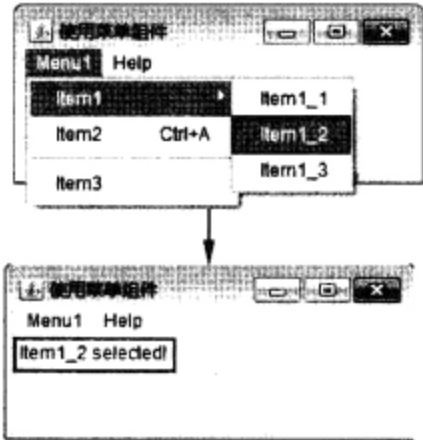


图 9.34 Item1 菜单项

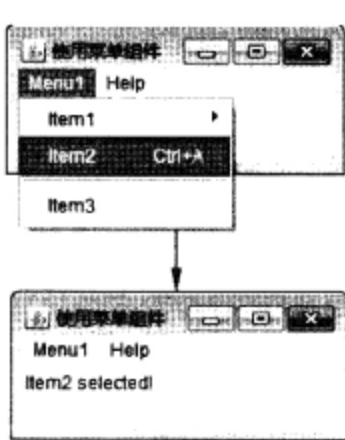


图 9.35 Item2 菜单项

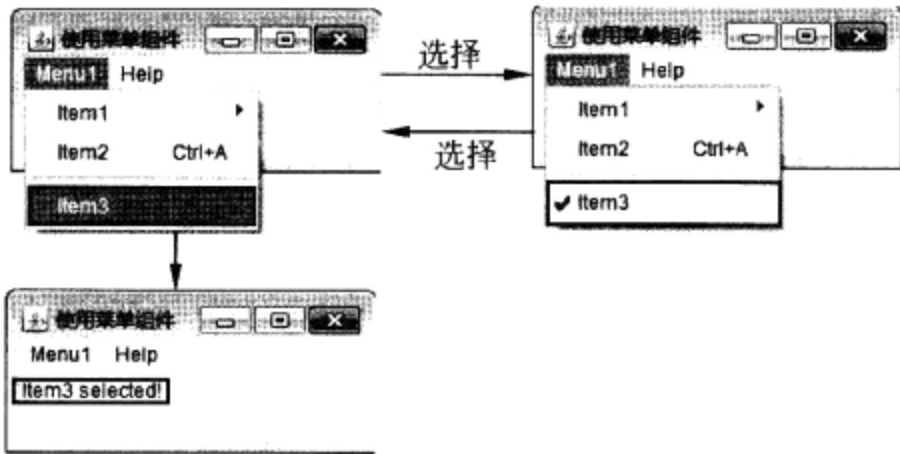


图 9.36 Item3 菜单项

3. Help菜单

当选择菜单 Help 时，就会出现如图 9.37 所示的菜单选项。选择 Index 选项，会弹出如图 9.38 所示的信息；选择 About 选项时，会弹出如图 9.39 所示的信息。

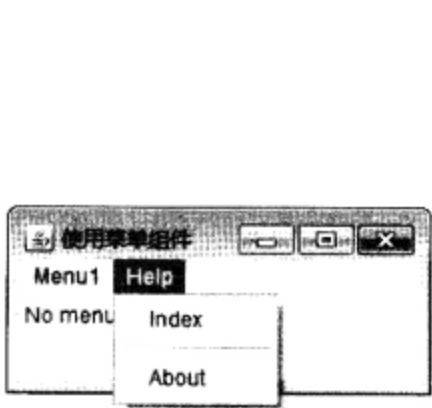


图 9.37 菜单 Help 选项

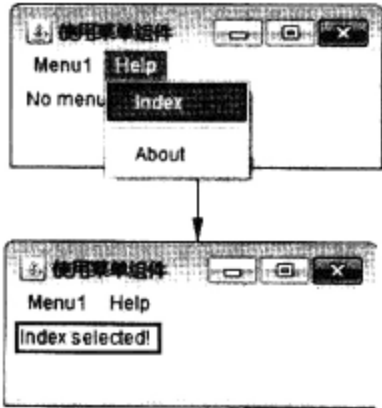


图 9.38 Index 菜单项

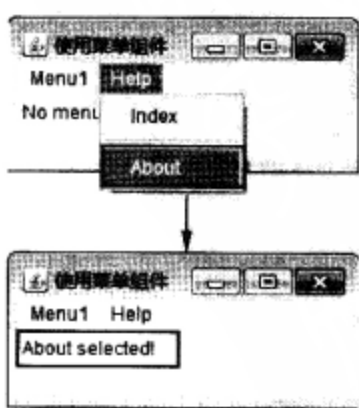


图 9.39 About 菜单项

4. 快捷菜单

在界面上右击，会弹出如图 9.40 所示的快捷菜单。当分别选择 popup1 和 popup2 选项时，显示面板会出现相应的信息，具体过程分别如图 9.41 和图 9.42 所示。



图 9.40 快捷菜单

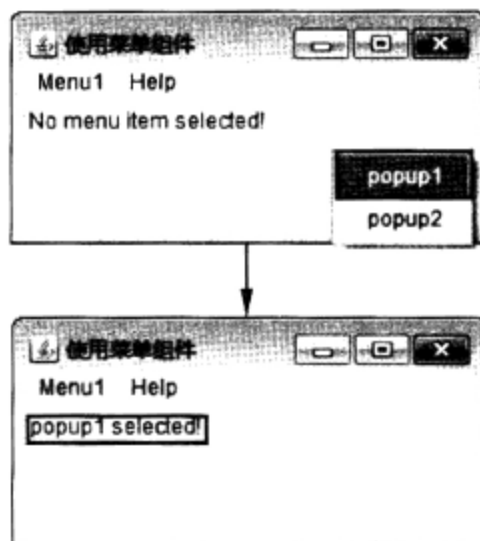


图 9.41 popup1 菜单项

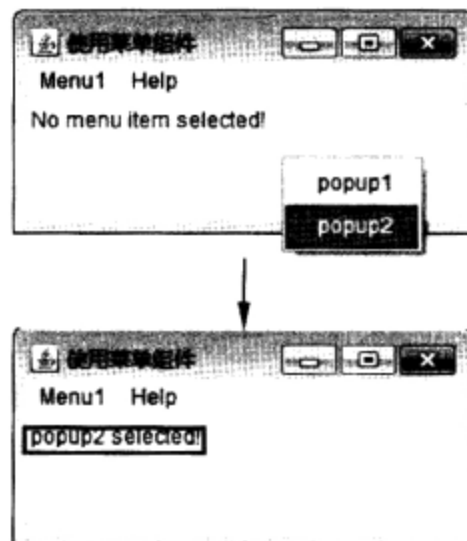


图 9.42 popup2 菜单项

9.6.2 菜单的用户界面

MenuExample.java 类用来实现包含有菜单组件的用户界面，该用户界面中包含了各种菜单组件。该类的具体内容如代码 9.7 所示。

代码 9.7 菜单组件类：MenuExample.java

```
public class MenuExample extends MouseAdapter implements ActionListener,
    ItemListener {
    //创建 3 个成员变量
    Frame f;
    Label l;
    PopupMenu pm;
    public static void main(String argv[]) {
        new MenuExample();
    }
    public MenuExample() {
        f = new Frame("使用菜单组件");
        f.addMouseListener(this);
        //创建并设置 ite9 的菜单
        MenuItem ite9_1 = new MenuItem("Ite9_1");
        MenuItem ite9_2 = new MenuItem("Ite9_2");
        MenuItem ite9_3 = new MenuItem("Ite9_3");
        //为各个菜单选项添加事件监听器
        Item1_1.addActionListener(this);
        Item1_2.addActionListener(this);
        Item1_3.addActionListener(this);
        //创建菜单 Item1
        Menu ite9 = new Menu("Ite9", false);
        ite9.add(item1_1);
        ite9.add(item1_2);
```

//窗口对象
//面板对象
//创建快捷菜单对象
//创建 MenuExample 对象
//构造函数
//为对象 f 赋值
//添加鼠标监听器
//创建菜单选项 Item1_1
//创建菜单选项 Item1_2
//创建菜单选项 Item1_3
//添加相应的菜单项到菜单上

```

ite9.add(item1_3);
//创建菜单项的快捷方式
MenuShortcut ms = new MenuShortcut(KeyEvent.VK_A, false);
MenuItem item2 = new MenuItem("Item2", ms);
//创建带有快捷方式的菜单项 Item2

item2.addActionListener(this); //添加事件监听器
//创建具有复选框的菜单项
CheckboxMenuItem item3 = new CheckboxMenuItem("Item3");
item3.addItemListener(this); //添加事件监听器
//创建和设置菜单 Menu1
Menu menu1 = new Menu("Menu1");
menu1.add(ite9);
menu1.add(item2);
menu1.addSeparator(); //添加分隔线
menu1.add(item3);
//创建和设置菜单 Help
Menu help = new Menu("Help");
MenuItem help1 = new MenuItem("Index");
MenuItem help2 = new MenuItem("About");
help1.addActionListener(this); //添加事件注册器
help2.addActionListener(this);
help.add(help1);
help.addSeparator(); //添加分隔线
help.add(help2);
//创建菜单项
MenuItem popup1 = new MenuItem("popup1");
MenuItem popup2 = new MenuItem("popup2");
popup1.addActionListener(this); //添加事件注册器
popup2.addActionListener(this);
//创建和设置快捷菜单 pm
pm = new PopupMenu();
pm.add(popup1);
pm.add(popup2);
//创建和设置菜单栏对象 mb
MenuBar mb = new MenuBar();
mb.add(menu1);
mb.setHelpMenu(help);
f.setMenuBar(mb); //设置窗口的菜单栏
l = new Label("No menu item selected!"); //为面板对象 l 赋值
f.add(pm); //为窗口对象添加快捷菜单
f.add(l, BorderLayout.NORTH); //设置窗口对象的布局管理器
f.setSize(200, 100); //设置窗口的大小和显示
f.setVisible(true);
}

public void actionPerformed(ActionEvent e) {
//实现 actionPerformed() 方法
MenuItem mi = (MenuItem) e.getSource();
l.setText(mi.getLabel() + " selected!");
}

public void itemStateChanged(ItemEvent e) {
//重写 itemStateChanged() 方法
CheckboxMenuItem cmi = (CheckboxMenuItem) e.getSource();
l.setText(cmi.getLabel() + " selected!");
}

public void mouseClicked(MouseEvent e) { //重写 mouseClicked() 方法
pm.show(f, e.getX(), e.getY());
}
}

```

【代码解析】

上述代码主要实现了使窗口中具有菜单的菜单栏，各个对象间的关系如图 9.43 所示。菜单栏具有两个菜单项，分别为 Menu1 和 Help 菜单。对于 Menu1 菜单，其有 3 个子菜单项，分别为：Item1 菜单项拥有 Item1_1、Item1_2 和 Item1_3 3 个菜单选项；对于菜单项 Item2，其具有 Ctrl+A 的快捷键；对于菜单项 Item3，其具有复选框。对于菜单 Help，其具有两个子菜单项，分别为 Index 和 About 子菜单。对于快捷菜单，其具有两个子菜单项，分别为 popup1 和 popup2 子菜单。

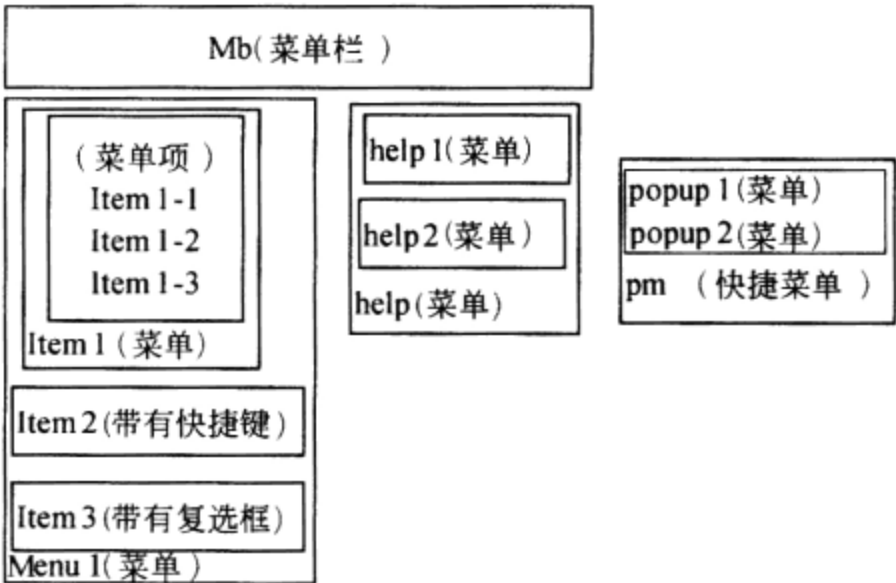


图 9.43 各个对象的关系

9.6.3 菜单组件的基本知识

对于一个完整的窗口界面，一套完整的菜单系统是必不可少的，而一套完整的菜单系统由菜单条、菜单和菜单项组成。查看 API 帮助文档，可以发现许多与菜单相关的类，它们的关系如图 9.44 所示，分别介绍如下。

1. MenuBar类

MenuBar（菜单栏）：一个 Frame 窗口只有一个菜单栏，对于类 Frame，一般用 setMenuBar()方法来制定它的菜单栏。MenuBar 类拥有一个构造函数，具体定义如下：

```
MenuBar () // 创建菜单栏实例
```

该函数为创建一个菜单栏实例。

2. MenuItem类

MenuItem（菜单项）：该类用来表示菜单里的菜单项。其拥有 3 个构造函数，具体定义如下。

(1) MenuItem ()

该函数用于创建一个菜单项实例。

(2) MenuItem (String label)

参数 label 为菜单项的标签，该函数用于创建一个具有指定标签的菜单项实例。

(3) MenuItem (String label, MenuShortcut s)

参数 s 为菜单项快捷键，该函数用于创建一个指定标签和快捷键的菜单项实例。

3. Menu类

Menu:（菜单）：菜单栏上可以拥有许多不同的菜单，而每一个菜单又可以拥有许多菜单项、分隔线甚至另一个菜单。该类拥有 3 个构造函数，具体定义如下：

(1) Menu()

该函数用于创建一个菜单实例。

(2) Menu(String label)

参数 label 为菜单的标签，创建一个具有指定标签的菜单实例。

(3) Menu(String label, boolean tearOff)

参数 tearOff 设置菜单是否可以分开，该函数用于创建一个指定标签和是否分开的菜单实例。

4. CheckboxMenuItem类

CheckboxMenuItem（复选框菜单选项）：该类表示菜单里的复选框选项。该类拥有 3 个构造函数，具体定义如下。

(1) Menu() ()

该函数用于创建一个具有复选框的菜单实例。

(2) CheckboxMenuItem (String label)

参数 label 为菜单的标签，该函数用于创建一个具有指定标签的复选框菜单实例。

(3) CheckboxMenuItem(String label, boolean state)

参数 state 设置菜单的选择状态，该函数用于创建一个指定标签和选择状态的具有复选框的菜单实例。

5. PopupMenu类

PopupMenu（快捷菜单）：该菜单不在菜单栏里，而是当右击时才出现的快捷菜单。**PopupMenu** 类拥有两个构造函数，具体定义如下。

(1) PopupMenu()

该函数用于创建一个快捷菜单实例。

(2) PopupMenu(String label)

参数 label 为菜单的标签，该函数用于创建一个具有指定标签的快捷菜单实例。

6. MenuShortcut类

MenuShortcut（快捷键）：该类用来表示菜单选项的快捷方式。其有两个构造函数，具体定义如下。

(1) MenuShortcut(int key)

参数 key 为键代码值，该函数用于创建一个指定键代码的具有快捷键的菜单实例。

(2) MenuShortcut(int key, boolean useShiftModifier)

参数 useShiftModifier 表示设置快捷键是否用 Shift 作为快捷键，该函数用于创建一个指定键代码的具有快捷键的菜单实例。

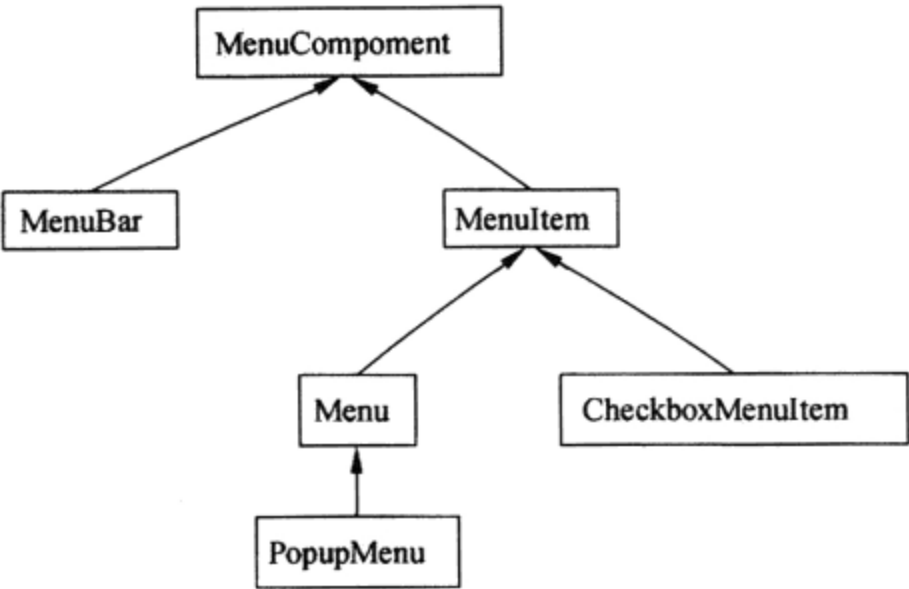


图 9.44 菜单类的继承关系

MenuItem 类的常用方法如表 9.10 所示; Menu 类的常用方法如表 9.11 所示; PopupMenu 类的常用方法如表 9.12 所示; MenuShortcut 类的常用方法如表 9.13 所示; MenuBar 类的常用方法如表 9.14 所示。

表 9.10 MenuItem类的常用方法

方法 名 称	功 能
delecteShortcut()	取消菜单项的快捷方式
getActionCommand()	取消菜单项的 ActionCommand
getLabel()	取得菜单项的标题文字
getShortcut()	取得菜单项所指定的 MenuShortcut 对象
isEnabled()	检查菜单是否可用
setActionCommand()	设置菜单项的 ActionCommand
setEnabled()	设置菜单项的可用状态
setLabel()	获取菜单选项的标题文字
setShortcut()	设置菜单项的快捷键
getSelectedObjects()	取得被选择的 CheckboxMenuItem 对象
getState()	获取菜单项的选择状态
setState()	设置菜单项的选择状态

表 9.11 Menu类的常用方法

方法 名 称	功 能
add()	添加 MenuItem 对象
addSeperator()	添加分隔线
getItem()	获取指定的 MenuItem 对象

续表

方 法 名 称	功 能
getItemCount()	获取所有 MenuItem 对象的个数
insert()	添加字符串到指定的位置
insertSeperator()	添加分隔线到指定的位置
remove()	移除 Menu 对象中的指定对象
removeAll()	移除 Menu 对象中的所有对象

表 9.12 PopupMenu类的常用方法

方 法 名 称	功 能
show()	显示快捷菜单

表 9.13 MenuShortcut类的常用方法

方 法 名 称	功 能
getKey()	获取快捷键的代码
usesShiftModifier()	检查是否使用了 Shift 赋值键

表 9.14 MenuBar类的常用方法

方 法 名 称	功 能
add()	取消菜单项的快捷方式
deleteShortcut()	取消菜单项的 ActionCommand
getHelpMenu()	取得菜单项的标题文字
getMenu()	取得菜单项所指定的 MenuShortcut 对象
getMenuCount()	检查菜单是否可用
setShortcutMenuItem()	设置菜单项的 ActionCommand
remove()	设置菜单项的可用状态
setHelpMenu()	获取菜单选项的标题文字
shortcuts()	设置菜单项的快捷键

9.7 对话框的用户界面

本节将通过对话框的用户界面，详细讲解使用对话框组件 Dialog 和文件对话框组件 FileDialog 的相关技巧。在该用户界面中，不仅将实现对话框组件的显示，还将实现与其他组件间的互动功能。

9.7.1 分析对话框和文件对话框的用户界面

本节将以直观的方式向读者介绍用户界面所要实现的功能。这些功能包括对话框初始

化界面、对话框的操作、文件对话框初始界面和选择列表。

1. 对话框初始化界面

当开始运行程序时，就会出现如图 9.45 所示的运行界面。在该界面中有一个文本框和两个按钮。



图 9.45 初始界面

2. 对话框的操作

在 Java 语言中存在两种模式的对话框，即模态对话框和非模态对话框。当单击“模态显示”按钮时会弹出模态对话框，同时文本框里也出现“模态测试”语句，如图 9.46 所示。当单击“非模态显示”按钮时会弹出非模态对话框，同时文本框里会出现“非模态测试”语句，如图 9.47 所示。

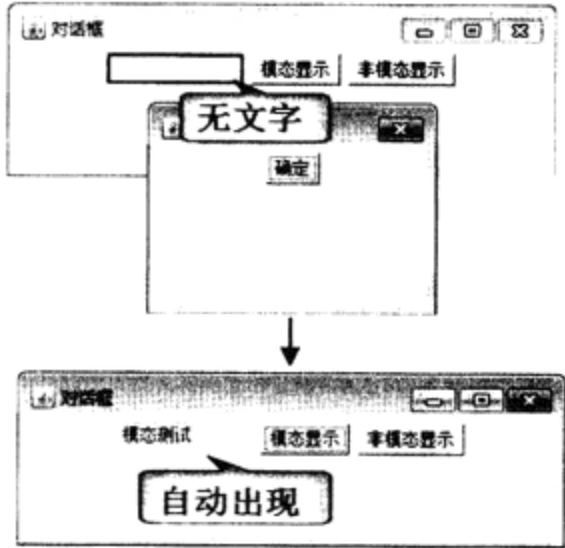


图 9.46 模态对话框

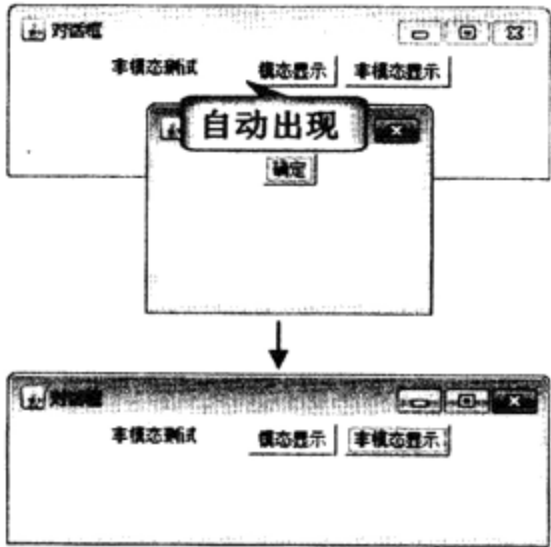


图 9.47 非模态对话框

注意：所谓模态对话框，是指在该对话框被关闭之前，用户不能操作其他窗口。所谓非模态对话框，是指该对话框在显示时，用户能够操作其他窗口。

3. 文件对话框初始界面

当开始运行程序时，会出现如图 9.48 所示的运行界面。在该界面中有两个单选框、两个显示面板和一个按钮。

4. 打开文件

在初始界面中，首先选择 LOAD 单选按钮，然后单击 Show 按钮会弹出打开文件对话框。在该对话框中进行相应的设置

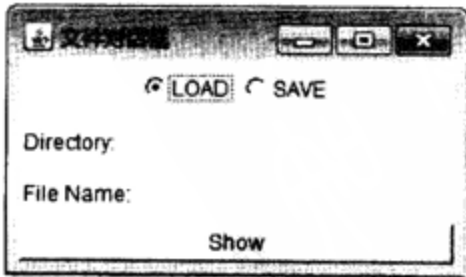


图 9.48 初始界面

后，单击“打开”按钮，文件的相应信息就会在初始界面显示出来，具体过程如图 9.49 所示。

5. 保存文件

在初始界面中，首先选择 SAVE 单选按钮，然后单击 Show 按钮会弹出保存文件对话框。在该对话框中进行相应的设置后，单击“保存”按钮，文件的相应信息就会在初始界面显示出来，具体过程如图 9.50 所示。

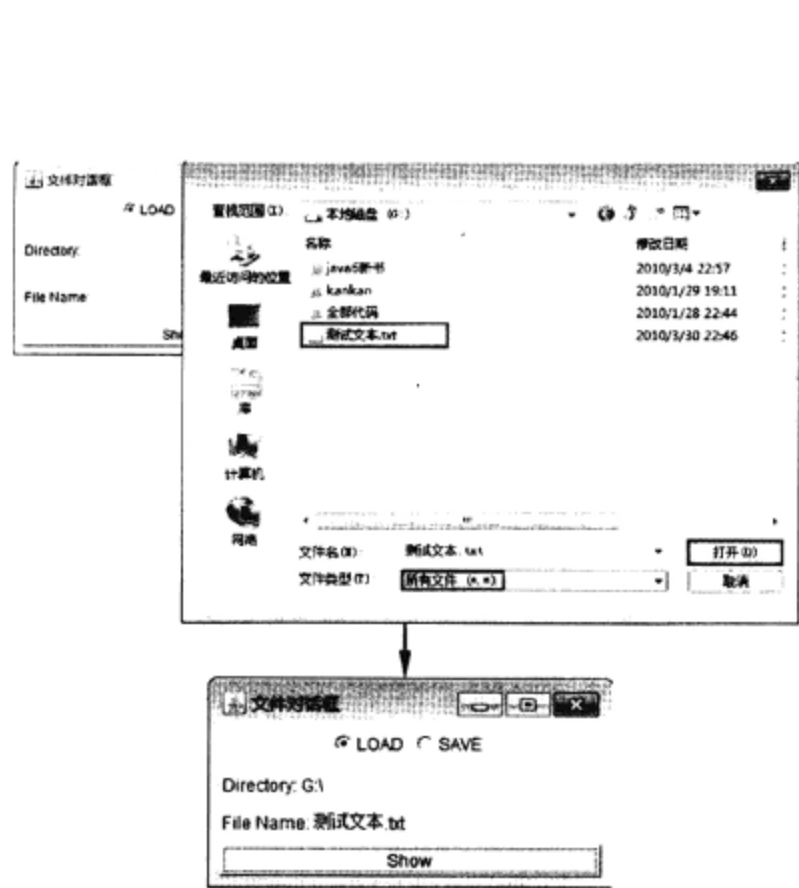


图 9.49 运行过程

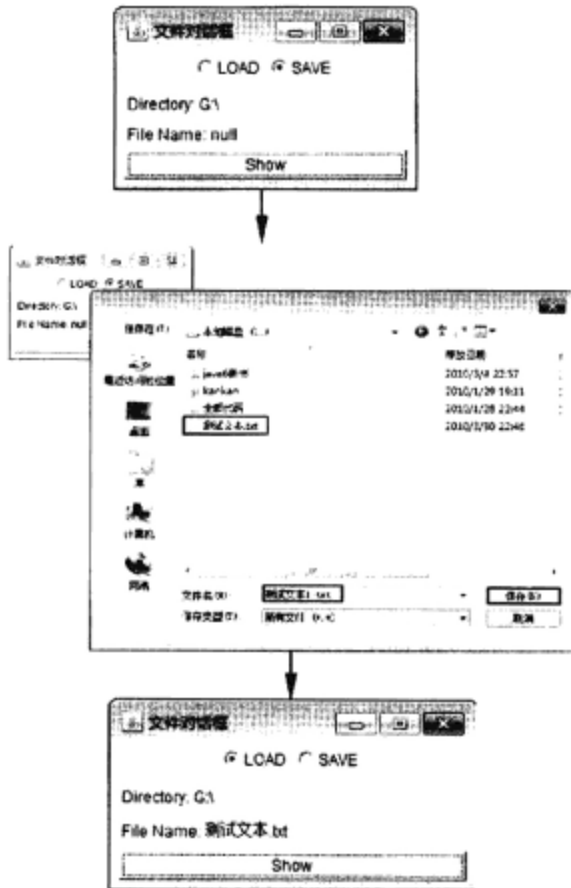


图 9.50 运行过程

9.7.2 对话框的用户界面

TestDialog.java 类用来实现包含对话框的用户界面，该用户界面根据单击的按钮，决定显示模态对话框还是非模态对话框。该类的具体内容如代码 9.8 所示。

代码 9.8 Dialog 组件类：TestDialog.java

```
public class TestDialog {
    //创建各种成员对象
    TextField tf = new TextField(10);
    Button b1 = new Button("模态显示");
    Button b2 = new Button("非模态显示");
    Frame f = new Frame("对话框");
    Button b3 = new Button("确定");
    Dialog dlg = new Dialog(f, "对话框", true);
    FlowLayout fl = new FlowLayout();
    TestDialog() {
        //创建 TextField 对象
        //创建按钮对象 b1
        //创建按钮对象 b2
        //创建窗口 f
        //创建按钮对象 b3
        //创建对话框
        //布局管理器
        //构造函数
    }
}
```

```

f.setLayout(fl); //设置布局管理器
f.add(tf); //添加文本框
//添加按钮
f.add(b1);
f.add(b2);
b1.addActionListener(new ActionListener() { //添加事件监听器
    public void actionPerformed(ActionEvent e) {
        dlg.setModal(true);
        dlg.setVisible(true);
        tf.setText("模态测试");
    }
});
b2.addActionListener(new ActionListener() { //添加事件监听器
    public void actionPerformed(ActionEvent e) {
        dlg.setModal(false);
        dlg.setVisible(true);
        tf.setText("非模态测试");
    }
});
f.setBounds(0, 0, 400, 200); //设置按钮的大小
f.setVisible(true); //显示窗口
f.addWindowListener(new WindowAdapter() { //关闭窗口
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
dlg.setLayout(fl); //设置对话框布局管理器
dlg.add(b3); //向对话框添加按钮
b3.addActionListener(new ActionListener() { //事件监听器
    public void actionPerformed(ActionEvent e) {
        dlg.dispose();
    }
});
dlg.setBounds(0, 0, 200, 150);
}
public static void main(String[] args) {
    new TestDialog(); //创建 TestDialog 对象
}
}

```

【代码解析】

在上述代码中,当执行到用模态方式显示对话框时,程序中的“tf.setText("模态测试")”并没有马上执行,而是当关闭对话框后才被执行。当执行到用非模态方式显示对话框时,程序中的“tf.setText("非模态测试")”就马上执行。

TestFileDialog.java 类用来实现包含文件对话框的用户界面,该用户界面根据选择的单选按钮决定显示打开对话框还是保存对话框。该类的具体内容如代码 9.9 所示。

代码 9.9 TestFileDialog 组件类: TestFileDialog.java

```

public class TestFileDialog implements ActionListener {
    //创建成员变量
    Frame f; //窗口对象
    FileDialog fd; //文件对话框对象
    Checkbox chLoad, chSave; //复选框对象
    CheckboxGroup cg; //复选框组对象
    Button b; //按钮对象 b
    Label lbDir, lbFile; //面板对象

    public static void main(String arv[]) {
        new TestFileDialog();
    }

    public TestFileDialog() { //构造函数
        f = new Frame("文件对话框"); //窗口对象
        fd = new FileDialog(f); //对话框对象
        cg = new CheckboxGroup(); //赋值复选框组
        //赋值单选框
        chLoad = new Checkbox("LOAD", true, cg);
        chSave = new Checkbox("SAVE", false, cg);
        Panel p1 = new Panel(); //创建面板对象
        //为面板添加单选框
        p1.add(chLoad);
        p1.add(chSave);
        //为面板对象赋值
        lbDir = new Label("Directory: ");
        lbFile = new Label("File Name: ");
        //创建并设置面板对象
        Panel p2 = new Panel(new GridLayout(2, 1));
        p2.add(lbDir);
        p2.add(lbFile);
        b = new Button("Show"); //创建按钮对象
        b.addActionListener(this); //添加事件监听器
        //为窗口对象添加相应的对象
        f.add(p1, BorderLayout.NORTH);
        f.add(p2, BorderLayout.CENTER);
        f.add(b, BorderLayout.SOUTH);
        f.pack();
        f.setVisible(true); //显示窗口
    }

    public void actionPerformed(ActionEvent e) {
        //实现 actionPerformed() 方法
        if (chLoad.getState())
            fd.setMode(FileDialog.LOAD);
        else
            fd.setMode(FileDialog.SAVE);
        fd.show();
        lbDir.setText("Directory: " + fd.getDirectory());
        lbFile.setText("File Name: " + fd.getFile());
        f.pack();
    }
}

```


【代码解析】

在上述代码中，对象 `FileDialog` 主要用来选择文件，即通过对话框选择要打开和保存的文件后，就可以通过该对象获取文件的相关信息。

9.7.3 `Dialog` 和 `FileDialog` 组件的基本知识

对于一个完整的窗口界面，经常需要用到一套完整对话框系统。查看 API 帮助文档，可以发现许多与对话框相关的类，它们的关系如图 9.51 所示，分别介绍如下。

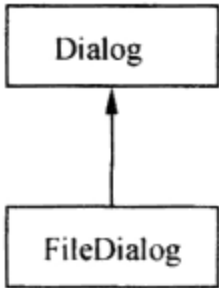


图 9.51 类的关系

1. `Dialog`类

`Dialog()`类常用的有 4 个构造函数，具体定义如下：

(1) `Dialog(Dialog owner)`

参数 `owner` 为该对话框的所有者，创建一个指定所有者的对话框。

(2) `Dialog(Dialog owner, String title)`

参数 `title` 为该对话框的标题，创建一个指定所有者和标题的对话框。

(3) `Dialog(Dialog owner, String title, boolean modal)`

参数 `modal` 为该对话框的显示模式，创建一个指定所有者、标题和显示模式的对话框。

(4) `Dialog(Dialog owner, String title, boolean modal, GraphicsConfiguration gc)`

函数功能为创建一个指定所有者、标题、显示模式和 `GraphicsConfiguration` 的对话框。

⚠注意：上述构造函数中第 1 个参数是必须的，其参数除了是 `Dialog` 类型外，还可以是 `Frame` 类型。

2. `FileDialog`类

`FileDialog` 类拥有 3 个构造函数，具体定义如下：

(1) `FileDialog(Dialog parent)`

参数 `parent` 为该文件对话框的所有者，函数功能为创建一个指定所有者的文件对话框。

(2) `FileDialog(Dialog parent, String title)`

参数 `title` 为该文件对话框的标题，函数功能为创建一个指定所有者和标题的文件对话框。

(3) `FileDialog(Dialog parent, String title, int mode)`

参数 `modal` 为该文件对话框的显示模式，函数功能为创建一个指定所有者、标题和显示模式的文件对话框。

⚠注意：上述构造函数中第 1 个参数是必须的，其参数除了是 `Dialog` 类型外，还可以是 `Frame` 类型。

`Dialog` 类的常用方法如表 9.15 所示。`FileDialog` 类的常用方法如表 9.16 所示。

表 9.15 Dialog类的常用方法

方 法 名 称	功 能
dispose()	释放对话框的资源
getTitle()	获取对话框的标题文字
hide()	隐藏对话框
isModal()	检查对话框的显示模式
isResizable()	检查对话框是否可以改变窗口的大小
setModal()	设置对话框的显示模式
setResizabel()	设置对话框是否可以改变窗口的大小
setTitle()	设置对话框的标题文字
show()	显示对话框

表 9.16 FileDialog类的常用方法

方 法 名 称	功 能
getDirectory()	获取所选文件的完整路径名称
setDirectory()	设置所选文件的完整路径名称
getFile()	获取所选文件的完整名称
setFile()	设置所选文件的完整名称
getFilenameFilter()	获取文件名称过滤器对象
setFilenameFilter()	设置文件名称过滤器对象
getMode()	获取 FileDialog 的显示模式
setMode()	设置 FileDialog 的显示模式

9.8 小 结

本章详细讲解了 Java 语言的典型图形用户界面，在讲解每个典型界面时，首先演示了该界面，然后介绍了如何实现该界面，最后详细介绍了该界面涉及的组件的基础知识。

本章涉及的典型用户界面包含 Label 和 Button 的用户界面、复选框的用户界面、下拉菜单和列表的用户界面、输入的用户界面、滚动条的用户界面、菜单的用户界面和对话框的用户界面。

第 10 章 计算器（布局管理器）

在图形用户界面的开发中,笔者不建议直接设置各个组件的大小和位置,而是通过 Java 语言中的布局管理机制来实现。本章不仅将讲解如何实现计算机,还将详细介绍布局管理机制的具体内容。

本章的学习目标如下:

- ❑ 掌握“计算器”项目;
- ❑ 理解和掌握各种布局管理器。

10.1 计算器原理

计算器项目用来模拟现实中的计算器,在具体运行该项目时,如果想实现各种计算功能,只需要用鼠标单击相应的按钮就可以实现。

10.1.1 项目结构框架分析

对于计算器项目,根据面向对象的思想,需要创建一个对象,即计算器。“计算器”项目目录如图 10.1 所示,该项目中存在一个计算器的类 Calculator.java。

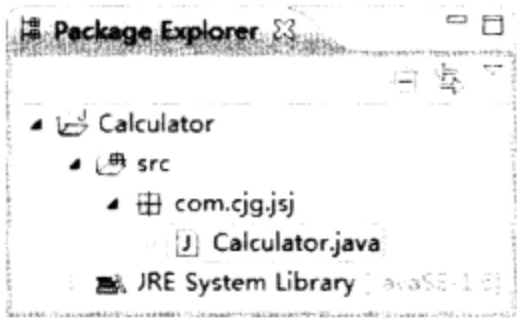


图 10.1 项目目录

10.1.2 项目功能业务分析

本节将以直观的方式向读者介绍整个项目要实现的功能。这些功能包括“计算器”项目的初始化界面和各种操作功能。

1. 计算器项目的初始化界面

当运行 Calculator 类后,会出现如图 10.2 所示的初始界面。该界面的最上方是一个文

本框，中间是显示数字和操作的按钮，最下面是一个关闭按钮。

2. 实现相加功能

在具体运行计算器项目时，如果想实现相加功能，可以先单击带有数字“4”的按钮，接着单击带有字符“+”的按钮，然后再单击带有数字“5”的按钮，最后单击带有字符“=”的按钮，这样在文本框中就可以显示出相应的结果了。具体过程如图 10.3 所示。



图 10.2 计数器的初始界面

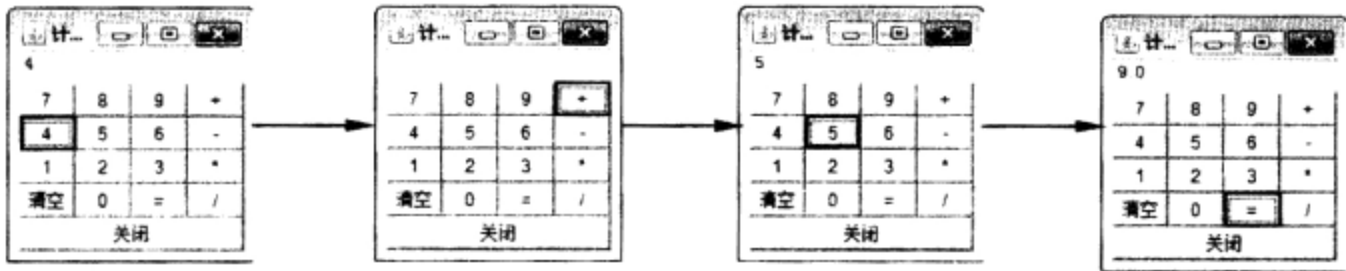


图 10.3 相加过程

3. 实现相减功能

在具体运行计算器项目时，如果想实现相减功能，可以先单击带有数字“8”的按钮，接着再单击带有字符“-”的按钮，然后单击带有数字“5”的按钮，最后单击带有字符“=”的按钮，这样在文本框中就可以显示出相应的结果了。具体过程如图 10.4 所示。

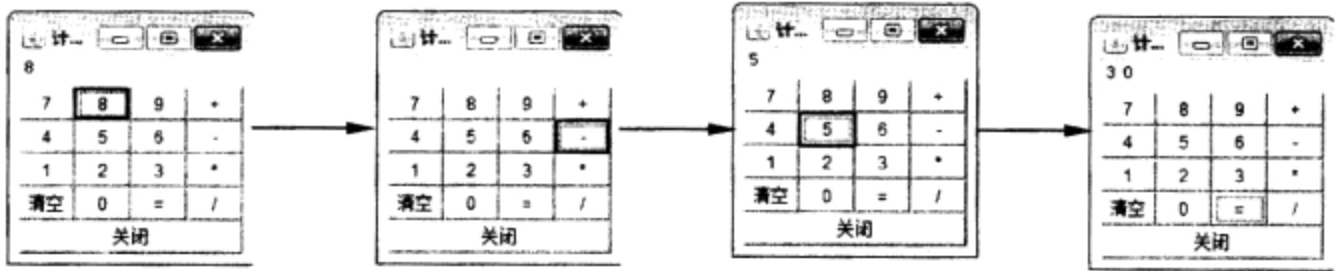


图 10.4 相减过程

4. 实现相乘功能

在具体运行计算器项目时，如果想实现相乘功能，可以先单击带有数字“1”的按钮，接着再单击带有字符“*”的按钮，然后再单击带有数字“2”的按钮，最后单击带有字符“=”的按钮，这样在文本框中就可以显示出相应的结果了。具体过程如图 10.5 所示。

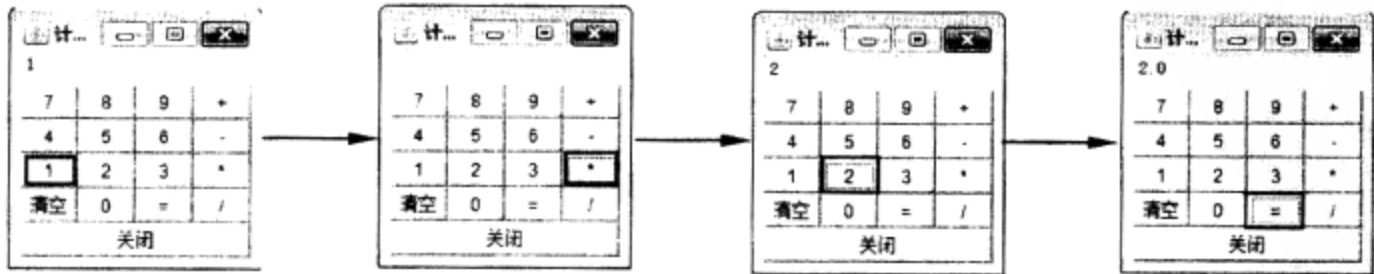


图 10.5 相乘过程

5. 实现相除功能

在具体运行计算器项目时，如果想实现相除功能，可以先单击带有数字“8”的按钮，再单击带有字符“/”的按钮，然后再单击带有数字“2”的按钮，最后单击带有字符“=”的按钮，这样在文本框中就可以显示出相应的结果了。具体过程如图 10.6 所示。

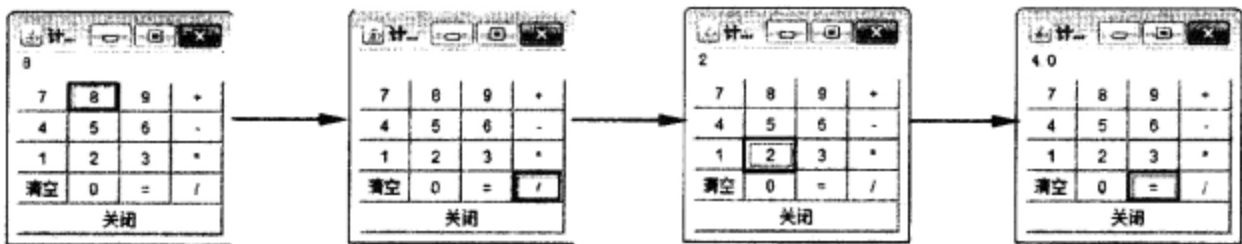


图 10.6 相除过程

6. 实现关闭功能

在具体运行计算器项目时，如果想实现关闭功能，可以通过单击窗体标题栏上的“关闭”按钮，也可以单击黄色按钮，如图 10.7 所示。



图 10.7 关闭功能

10.2 计算器的实现过程

为了更好地模拟计算器的功能，该项目不仅实现各种的计算功能，还会通过布局管理器设计和实现计算器项目的界面。

Calculator.java 类为计算器类，在该类中通过布局管理器统一管理各个组件的大小和位置。该类的具体内容如代码 10.1 所示，其 UML 如图 10.8、图 10.9 和图 10.10 所示。

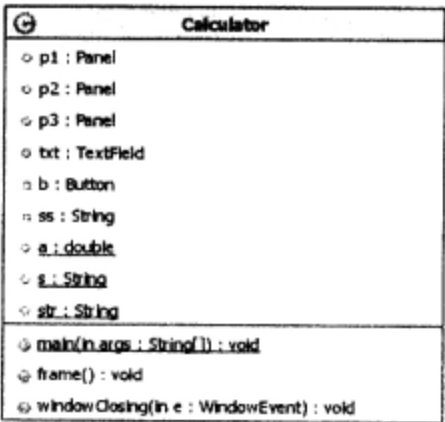


图 10.8 计算器类图

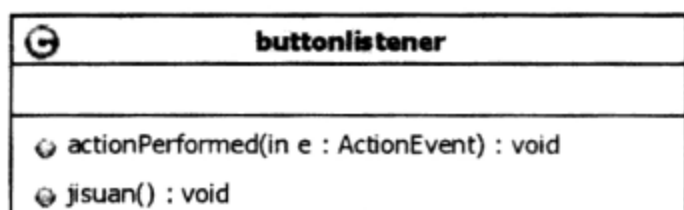


图 10.9 按钮监听器的类

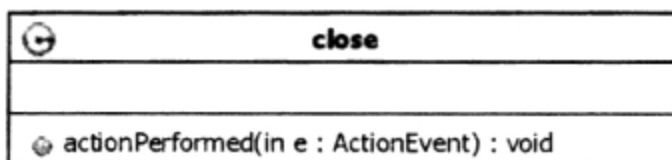


图 10.10 实现关闭功能的类

代码 10.1 计算器类: Calculator.java

```

public class Calculator extends WindowAdapter {
    //定义 3 个面板
    Panel p1 = new Panel(); //创建面板 p1
    Panel p2 = new Panel(); //创建面板 p2
    Panel p3 = new Panel(); //创建面板 p3
    TextField txt; //创建文本框对象
    private Button[] b = new Button[17]; //创建按钮数组
    private String ss[] = { "7", "8", "9", "+", "4", "5", "6", "-", "1", "2",
        //创建字符串数组
        "3", "*", "清空", "0", "=", "/", "关闭" };
    static double a; //创建 double 类型变量
    static String s, str; //创建 String 类型变量
    public static void main(String args[]) {
        (new Calculator()).frame(); //创建 Calculator 对象
    }
    public void frame() { //实现界面
        Frame fm = new Frame("计算器"); //创建窗口对象
        for (int i = 0; i <= 16; i++) {
            b[i] = new Button(ss[i]); //为按钮数组赋值
        }
        for (int i = 0; i <= 15; i++) {
            p2.add(b[i]); //添加按钮到面板 p2
        }
        b[16].setBackground(Color.yellow); //设置按钮的背景色为黄色
        //创建和设置文本框
        txt = new TextField(15);
        txt.setEditable(false);
        for (int i = 0; i <= 16; i++) {
            b[i].addActionListener(new buttonlistener()); //为按钮添加监听器
        }
        b[16].addActionListener(new close()); //为按钮添加关闭监听器
        fm.addWindowListener(this);
        fm.setBackground(Color.red); //设置窗口背景色为红色
        p1.setLayout(new BorderLayout()); //设置面板 p1 布局管理器
        p1.add(txt, "North"); //添加文本框到北面部分
        p2.setLayout(new GridLayout(4, 4)); //设置面板 p2 布局管理器
        p3.setLayout(new BorderLayout()); //设置面板 p3 布局管理器
        p3.add(b[16]); //添加按钮到面板 p3
        //添加各个面板到窗口上
        fm.add(p1, "North");
        fm.add(p2, "Center");
        fm.add(p3, "South");
        fm.pack();
        fm.setVisible(true); //显示窗口
    }
}

```



```

public void windowClosing(WindowEvent e) {
    System.exit(0); //退出系统
}
//编写事件监听器
class buttonlistener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        Button btn = (Button) e.getSource(); //获取发生事件按钮
        if (btn.getLabel() == "=") {
            jisuan();
            str = String.valueOf(a);
            txt.setText(str);
            s = "";
        } else if (btn.getLabel() == "+") {
            jisuan();
            txt.setText("");
            s = "+";
        }...
    }
    public void jisuan() { //编写具体计算方法
        if (s == "+")
            a += Double.parseDouble(txt.getText());
        else if (s == "-")
            a -= Double.parseDouble(txt.getText());
        else if (s == "*")
            a *= Double.parseDouble(txt.getText());
        else if (s == "/")
            a /= Double.parseDouble(txt.getText());
        else
            a = Double.parseDouble(txt.getText());
    }
}
}
class close implements ActionListener { //退出方法
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
}
}

```

【代码解析】

- 在上述代码中编写了一个算法的方法 jisuan(), 在该方法中根据算法(+、-、*和/)实现对变量 a 的赋值。接着在事件监听器的 actionPerformed()方法中根据事件源的文本调用 jisuan()方法为字符串 str 赋值。
- 整个窗口通过 BorderLayout 布局管理器来管理中 p1、p2 和 p3 3 个面板对象。对于容器组件 p1, 由于其只包含一个文本框组件, 所以通过 BorderLayout 布局管理器来管理; 对于组件 p2, 由于其包含了 16 个按钮组件, 所以通过 GridLayout 布局管理器把整个面板分成 4×4 一共 16 个区域; 对于容器组件 p3, 由于其只包含一个按钮组件, 所以通过 BorderLayout 布局管理器来管理。

10.3 知识点扩展——事件机制的高级知识

在应用程序中, 如果想在容器中放置若干个组件, 则必须确定这些组件的位置和大小。

为了简化程序员对容器上组件的布局控制，Sun 公司提供了一个 Layout（布局管理器）机制，即一个容器内所有组件的显示位置可以由一种 Layout 对象来自动管理。

10.3.1 为什么需要版面的配置

查看 API 帮助文档可以发现，只有继承了 Container 类的类才能使用 setLayout() 方法设置自己的布局管理器。java.awt 包中包含了 5 种基本的 Layout 类，分别如下。

- ❑ BorderLayout: 将容器分成东（EAST）、南（SOUTH）、西（WEST）、北（NORTH）和中（CENTER）5 个区域，如图 10.11 所示。
- ❑ FlowLayout: 将容器中的所有组件按照从左到右、由上到下的顺序排列。如果窗口宽度够，则把全部组件放在同一列，如果宽度不够，则放在下一列。容器 Panel 和 applet 默认的 Layout 就是 FlowLayout。
- ❑ CardLayout: 将容器中的所有组件放置在同一区域内，只显示最上面的组件。相当于把扑克牌里 52 张牌叠成一叠，只显示最上面的那一张牌。
- ❑ GridLayout: 将容器划分成若干行、列的网格，接着将需要添加的组件按从左到右、从上到下的顺序在网格中排列。
- ❑ GridBagLayout: 前面 4 种布局管理器都很死板，没有办法让组件任意放置。而对于该布局管理器，只要能想得出来的布局，几乎都可以实现。

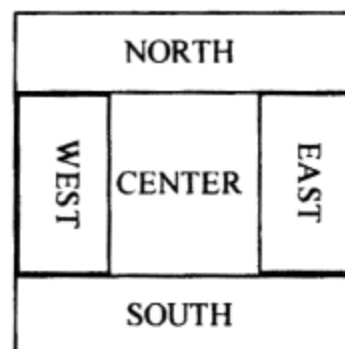


图 10.11 BorderLayout 版面配置

下面通过一个名为 LayoutWhy 的类具体讲解，如果没有使用布局管理器，将无法出现预想的效果，该类的具体内容如代码 10.2 所示。

代码 10.2 没使用 Layout: LayoutWhy.java

```
public class LayoutWhy {
    public static void main(String argv[]) {
        Frame f = new Frame("为什么要使用 Layout");           //创建窗口对象
        //创建 5 个按钮对象
        Button b1 = new Button("One");
        Button b2 = new Button("Two");
        Button b3 = new Button("Three");
        Button b4 = new Button("Four");
        Button b5 = new Button("Five");
        //添加按钮到窗口对象
        f.add(b1);
        f.add(b2);
        f.add(b3);
        f.add(b4);
        f.add(b5);
        f.setSize(10, 100);                                     //设置窗口的大小
        f.setVisible(true);
    }
}
```

运行 LayoutWhy.java 类，出现如图 10.12 所示的用户图形界面。

【代码解析】

在上述代码中，虽然给窗口对象 f 添加了 5 个按钮，但是最后的显示结果却只显示出第 5 个按钮，其他 4 个却没有显示出来。之所以会这样，是因为这 5 个按钮放置在对象 f 的同一个位置上。

从图 10.10 所示的运行结果中可以发现，如果在图形界面的程序中没有使用布局管理器，则不能完全按照程序员的意愿设置组件的大小和位置。

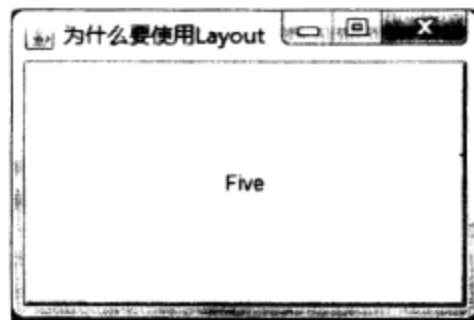


图 10.12 运行结果

10.3.2 Java 语言中的各种布局管理器

在 Java 语言中，布局管理器不仅负责组件在容器中的排列方式，而且还使生成的图形用户界面具有良好的平台无关性。因此在具体的应用程序中不允许直接设置组件的位置和大小，而是通过管理组件的布局管理器来实现。本节将通过具体的实例来详细讲解 Java 语言中的所有布局管理器。

1. BorderLayout

为了实现 BorderLayout 布局，在 Java API 中存在一个名为 BorderLayout 的类，该类的构造函数基本格式如下：

(1) BorderLayout()

该构造函数用于构造组件之间没有间距的边界布局。

(2) BorderLayout(int hgap, int vgap)

该构造函数用于构造组件之间有间距的边界布局，其中参数 hgap 为组件间的水平距离，参数 vgap 为组件间的垂直距离。

注意：使用了无参构造函数实现 BorderLayout 布局后，如果还想设置组件间的距离，可以通过 setHgap() 和 setVgap() 方法来设置组件间的水平和垂直距离。

下面通过一个具体的类 Border，来讲解如何使用 BorderLayout 布局管理器，具体内容如代码 10.3 所示。

代码 10.3 使用 BorderLayout: Border.java

```
public class Border {
    public static void main(String argv[]) {
        Frame f = new Frame("使用 BorderLayout 布局管理器"); //创建窗口对象
        //创建 5 个按钮对象
        Button b1 = new Button("One");
        Button b2 = new Button("Two");
        Button b3 = new Button("Three");
        Button b4 = new Button("Four");
        Button b5 = new Button("Five");
        //添加按钮到窗口对象上
```

```

        f.add(b1, BorderLayout.EAST);
        f.add(b2, BorderLayout.SOUTH);
        f.add(b3, BorderLayout.WEST);
        f.add(b4, BorderLayout.NORTH);
        f.add(b5, BorderLayout.CENTER);
        f.pack();
        f.setVisible(true);           //显示按钮
    }
}

```

运行 Border.java 类，会出现如图 10.13 所示的用户图形界面。

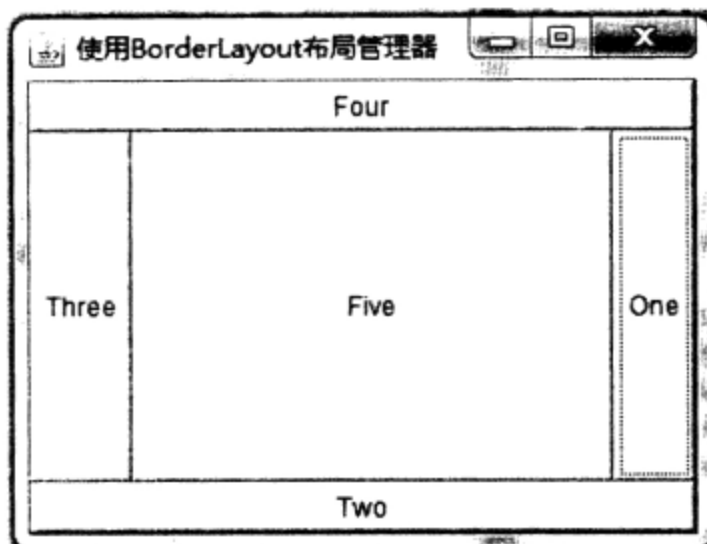


图 10.13 运行结果

【代码解析】

- ❑ 上述代码创建了一个窗口对象 f，然后创建了 5 个按钮对象 b1、b2、b3、b4 和 b5，最后通过 add() 方法把按钮添加到对象 f 里。
- ❑ 在具体使用 add() 方法时，该方法需要传入两个参数，分别是容器对象和指定的位置。对于需要指定的位置在 BorderLayout 类中，已经定义了 5 个常量，分别为 BorderLayout.EAST、BorderLayout.SOUTH、BorderLayout.WEST、BorderLayout.NORTH 和 BorderLayout.CENTER。
- ❑ 在上述代码中，之所以没有利用 f.setSize() 方法设置窗口的大小，而是调用了 f.pack() 方法，这是因为 pack() 方法会让窗口调整到最佳大小，即系统会按照当前窗口上出现的组件所需要的最佳大小来设置整个窗口的大小，其会保证整个窗口的大小刚好能够让全部组件显示出来。

对于 BorderLayout 布局管理器允许最多放置 5 个组件，如果想在窗口上放置更多的组件，可以先将若干组件添加到一个 Panel 上，然后再把该 Panel 作为一个组件放置到窗口上。如果窗口上的组件少于 5 个，则没有放置组件的区域将被相邻的区域占有。

当对图 10.13 左右拉大和缩小时，北、中和南区域的宽度会改变，但是东和西区域的宽度却不会改变，如图 10.14 所示。当对图 10.13 上下拉大和缩小时，东、中和西区域的高度会改变，但是北和南区域的高度却不会改变，如图 10.15 所示。

2. FlowLayout

为了实现 FlowLayout 布局，在 Java API 中存在一个名为 FlowLayout 的类，该类的构造函数基本格式如下：

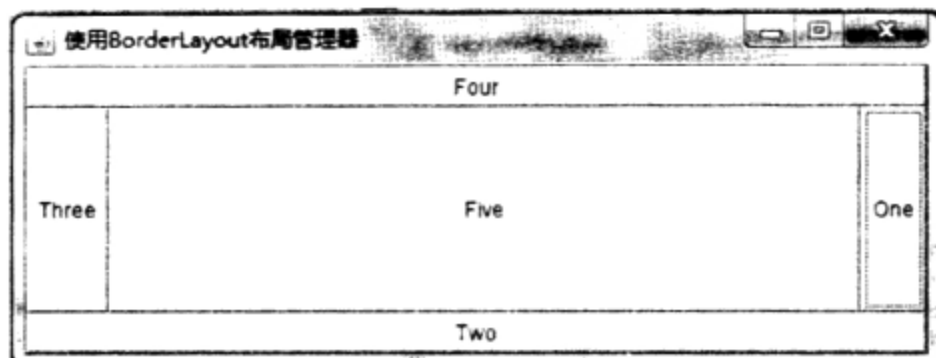


图 10.14 拉长后的结果



图 10.15 拉长后的结果

(1) FlowLayout()

该构造函数用于构造居中对齐方式和组件之间没有间距的布局管理器。

(2) FlowLayout(int align)

该上述构造函数中，参数 align 为组件的对齐方式。

注意：使用了无参构造函数实现 FlowLayout 布局后，如果还想设置组件间的对齐方式，可以通过 setAlignment() 方法设置组件间的对齐方式。

(3) FlowLayout(int align, int hgap, int vgap)

在该构造函数中，参数 align 为组件间的对齐方式，参数 hgap 为组件间的水平距离，参数 vgap 为组件间的垂直距离。

下面通过一个具体的类 Flow，来讲解如何使用 FlowLayout 布局管理器，具体内容如代码 10.4 所示。

代码 10.4 使用 FlowLayout: Flow.java

```

public class Flow {
    public static void main(String argv[]) {
        Frame f = new Frame("使用 FlowLayout 布局管理器");    //创建窗口对象
        //创建 5 个按钮对象
        Button b1 = new Button("One");
        Button b2 = new Button("Two");
        Button b3 = new Button("Three");
        Button b4 = new Button("Four");
        Button b5 = new Button("Five");
        f.setLayout(new FlowLayout());                        //设置布局管理器
        //添加按钮到窗口对象上
        f.add(b1);
        f.add(b2);
        f.add(b3);
        f.add(b4);
        f.add(b5);
        f.pack();
        f.setVisible(true);
    }
}

```

```

    }
}

```

运行 Flow.java 类，出现如图 10.16 所示的用户图形界面。

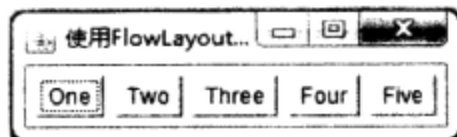


图 10.16 运行结果

【代码解析】

在上述代码中，通过 `f.setLayout(new FlowLayout())` 语句设置窗口对象 `f` 的 `Layout` 为 `FlowLayout` 类型。

对于运行结果，如果窗口变小时，则一行组件将变成两行组件，如图 10.17 所示。如果窗口变大时可以发现，`FlowLayout` 管理器默认使用居中对齐方式，如图 10.18 所示。如果想修改布局管理器的对齐方式，可以通过方法 `setAlignment()` 来实现。`FlowLayout` 类提供了 3 个常量 `FlowLayout.LEFT`、`FlowLayout.RIGHT` 和 `FlowLayout.CENTER`，作为方法 `setAlignment()` 的参数。如果以左对齐方式，则运行结果如图 10.19 所示，如果以右对齐方式，则运行结果如图 10.20 所示。

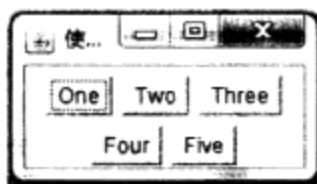


图 10.17 窗口变小

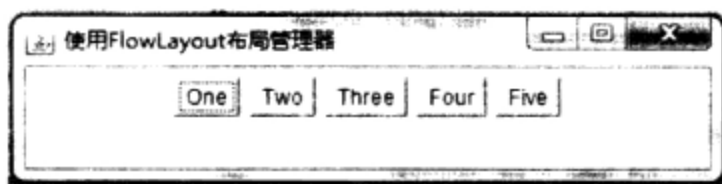


图 10.18 窗口变大

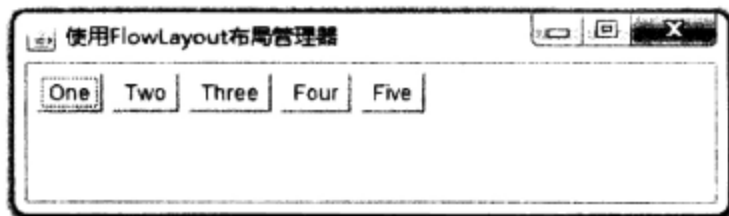


图 10.19 左对齐方式

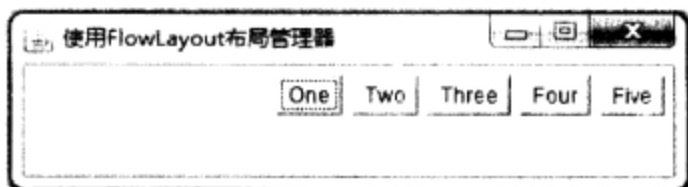


图 10.20 右对齐方式

3. CardLayout

为了实现 `CardLayout` 布局，在 Java API 中存在一个名为 `CardLayout` 的类，该类的构造函数基本格式如下：

(1) `CardLayout()`

该构造函数用于构造组件之间没有间距的边界布局。

(2) `CardLayout(int hgap, int vgap)`

该构造函数中，参数 `hgap` 为组件间的水平距离，参数 `vgap` 为组件间的垂直距离。

下面通过一个具体的类 `Card`，来讲解如何使用 `CardLayout` 布局管理器，具体内容如代码 10.5 所示。

代码 10.5 使用 Card Layout: Card.java

```

public class Card implements ActionListener {
    Frame f;                                //静态常量 f
    CardLayout card;                        //静态常量 card
    public static void main(String argv[]) {
        new Card();
    }
    public Card() {                          //构造函数
        f = new Frame("使用 CardLayout 布局管理器"); //为常量 f 赋值
        //创建 5 个按钮对象
        Button b1 = new Button("One");
        Button b2 = new Button("Two");
        Button b3 = new Button("Three");
        Button b4 = new Button("Four");
        Button b5 = new Button("Five");
        //注册事件监听器
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);
        b4.addActionListener(this);
        b5.addActionListener(this);
        card = new CardLayout();            //创建布局管理器
        f.setLayout(card);                  //设置对象 f 的布局管理器
        //添加按钮到窗口对象里
        f.add(b1, "1");
        f.add(b2, "2");
        f.add(b3, "3");
        f.add(b4, "4");
        f.add(b5, "5");
        f.pack();                           //自适应窗口大小
        f.setVisible(true);                 //显示
    }
    public void actionPerformed(ActionEvent e) { //创建事件监听器
        card.next(f);                       //显示下一个组件
    }
}

```

运行 Card.java 类，出现如图 10.21 所示的运行过程。当程序开始运行时，只会看到 One 按钮在窗口上，当单击按钮后，Two 按钮就会在窗口上显示出来。依次类推，每按一下按钮就会在 5 个按钮间循环，但是每次只有一个按钮显示在窗口里。

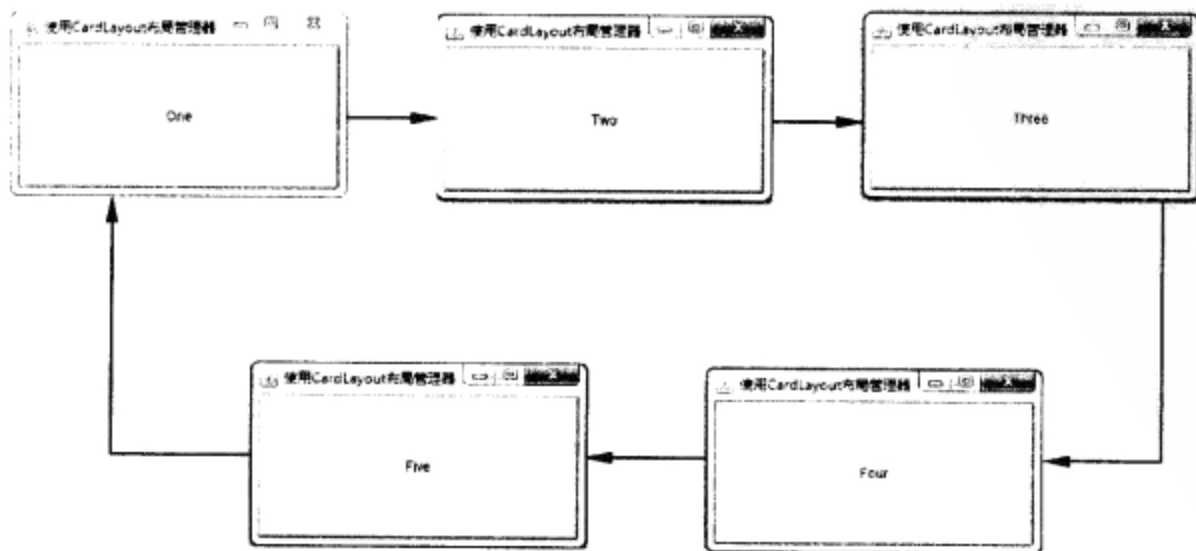


图 10.21 运行过程

【代码解析】

- ❑ 在上述代码中，首先创建了两个窗口和布局管理器的成员变量 f 和 card，然后创建了 ActionEvent 事件的监听器，最后在构造函数中实现相应的功能。
- ❑ 在上述代码的构造函数中，首先为成员变量 f 和 card 赋值，然后为 f 注册事件监听器并添加创建的 5 个按钮到对象 f 上，最后通过相应的方法设置对象 f 的布局管理器并显示出按钮。
- ❑ 在添加 5 个按钮的具体代码 f.add(b1, "1")中，方法 add()中两个参数的含义分别为所要添加按钮和为所要添加按钮设置一个名字。

最后查看 API 帮助文档可以发现，对于 CardLayout 类的显示方法，除了 next()方法外还包含如表 10.1 所示的方法。

表 10.1 显示的方法

方 法 名 称	参数类型说明	使 用 说 明
Next()	Container	显示下一个组件
previous()	Container	显示上一个组件
first()	Container	显示第一个组件
last()	Container	显示最后一个组件
show()	Container,Strng	显示指定名称的组件

⚠注意：show()方法中第二个参数 Strnig 就是代码 f.add(b1, "1")中重新定义的名字，即为 1。

4. GridLayout

为了实现 CridLayout 布局，在 Java API 中存在一个名为 CridLayout 的类，该类构造函数的基本格式如下：

(1) GridLayout()

该构造函数用于创建具有构造默认值的布局。

(2) GridLayout(int rows, int cols)

在该构造函数中，参数 hgap 为组件间的水平距离，参数 vgap 为组件间的垂直距离。

(3) GridLayout(int rows, int cols, int hgap, int vgap)

在该构造函数中，参数 rows 为网格的行数，参数 cols 为网格的列数，参数 hgap 为组件间的水平距离，参数 vgap 为组件间的垂直距离。

下面通过一个具体的类 Grid，来讲解如何使用 CardLayout 布局管理器，具体内容如代码 10.6 所示。

代码 10.6 使用 GridLayout: Grid.java

```
public class Grid {
    public static void main(String argv[]) {
        Frame f = new Frame("使用 GridLayout 布局管理器"); //创建窗口对象 f
        //创建 5 个按钮
```

```

    Button b1 = new Button("One");
    Button b2 = new Button("Two");
    Button b3 = new Button("Three");
    Button b4 = new Button("Four");
    Button b5 = new Button("Five");
    f.setLayout(new GridLayout(2, 3));           //设置窗口的布局管理器
    //添加 5 个按钮到窗口对象 f
    f.add(b1);
    f.add(b2);
    f.add(b3);
    f.add(b4);
    f.add(b5);
    f.pack();
    f.setVisible(true);                          //显示窗口
}
}

```

运行 Grid.java 类，出现如图 10.22 所示用户图形界面。

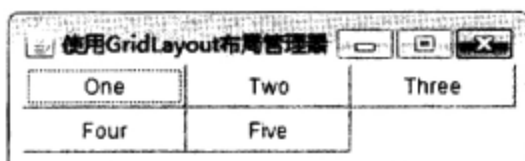


图 10.22 运行结果

【代码解析】

在上述代码中，通过 `f.setLayout(new GirdLayout(2,3))` 语句设置窗口对象 `f` 的 Layout 为 `GirdLayout` 类型。

10.4 小 结

本章主要通过 Java 语言中的布局机制来实现计算器项目，在具体实现该项目的过程中，最主要的过程就是实现计算器的布局。为了让读者能够更好地掌握 Java 语言的布局机制，在本章的最后还详细介绍了 Java 语言的各种布局管理器。

要设计一个好的软件，界面是给用户的第一印象，所以本章的布局管理器对读者以后设计软件会起到非常关键的作用。

第 11 章 秒表（事件+线程）

在 Java 语言中利用事件机制可以实现许多意想不到的功能，所以对于程序员来说，掌握事件机制是必须的。对于一个应用程序，如果想实现与用户良好的互动功能，对事件的处理是必不可少的。本章将通过模拟秒表的功能介绍 Java 语言中的事件机制，还将详细介绍事件机制的基本知识。

本章的学习目标如下：

- 掌握秒表项目；
- 理解为什么要使线程具有同步性；
- 掌握实现线程同步的两种方式。

11.1 秒表原理

“秒表项目”用来模拟现实生活中秒表的功能，在具体使用时，如果想记时开始，则按住鼠标，如果想停止计时，则松开鼠标。由于该系统需要处理鼠标的事件，所以必须了解事件机制。

11.1.1 项目结构框架分析

对于“秒表项目”，根据面向对象的思想，需要创建一个对象——秒表。“秒表项目”目录如图 11.1 所示，该项目中的两个类分别为秒表类 Stopwatch 和测试秒表的类 TestStopWatch。

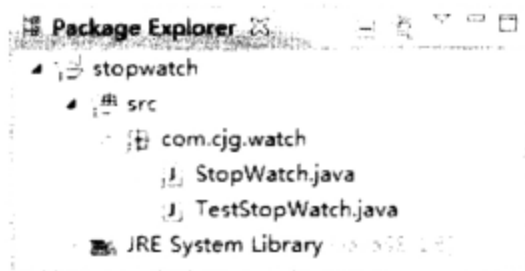


图 11.1 项目目录

11.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括秒表初始化，按住鼠标功能和放开鼠标后的功能。

1. 秒表的初始化

当运行测试秒表的类 TestStopWatch 后，会出现如图 11.2 所示的初始界面。在该界面中显示出秒表的初始化数字，即 00:00:00。

2. 按住鼠标功能

秒表要实现计时功能，则需要用鼠标按住界面，这时界面上的时间值就会随着时间的增长而增加，具体过程如图 11.3 所示。

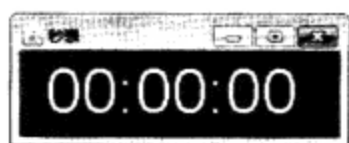


图 11.2 秒表的初始界面

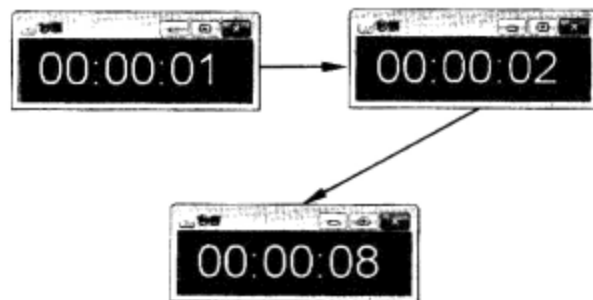


图 11.3 按住鼠标的过程

3. 释放鼠标功能

当放开鼠标后，秒表就停止计时。这时秒表界面就会显示出从鼠标开始按住到释放过程所经历的时间。如果在界面上再次单击鼠标，就会返回到初始界面，具体过程如图 11.4 所示。

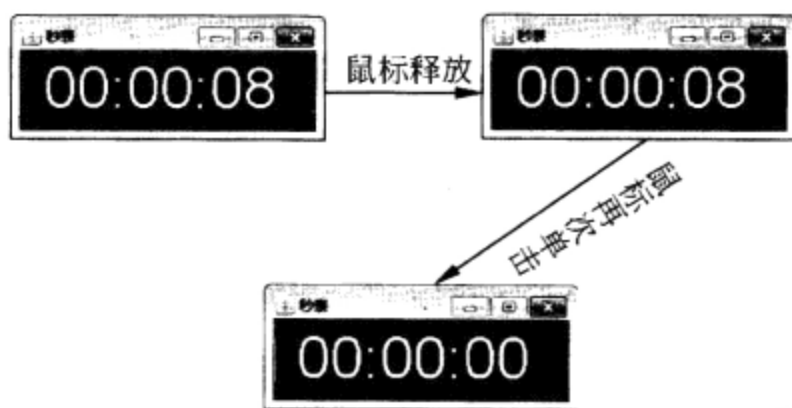


图 11.4 鼠标释放后的过程

11.2 秒表的实现过程

本章将通过事件机制和多线程的相关知识来实现秒表项目，具体程序架构如图 11.5 所示，它包含一个类对象 Stopwatch.java 和一个进行测试的类 TestStopWatch.java。

11.2.1 秒表类

StopWatch.java 类用来模拟秒表，该类的具体内容如代码 11.1 所示，UML 如图 11.6

所示。

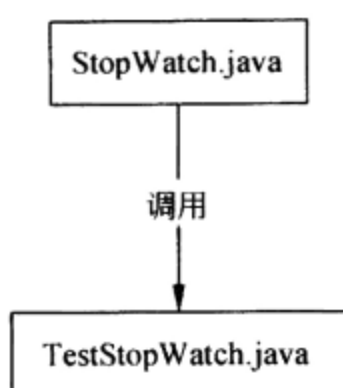


图 11.5 程序关系图

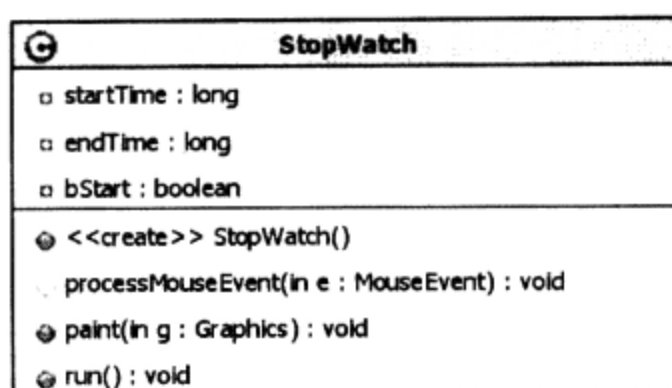


图 11.6 秒表类图

代码 11.1 秒表类: Tickets.java

```

class Stopwatch extends Canvas implements Runnable {
    //创建 3 个成员变量
    private long startTime = 0; //开始时间变量
    private long endTime = 0; //结束时间变量
    private boolean bStart = false; //状态变量
    public Stopwatch() { //构造函数
        enableEvents(AWTEvent.MOUSE_EVENT_MASK); //设置组件事件
        setSize(80, 30);
    }
    protected void processMouseEvent(MouseEvent e) {
        //重写 processMouseEvent() 方法
        //鼠标按下时，启动计时线程，并让起始时间变量和终止时间变量都等于当前时间
        if (e.getID() == MouseEvent.MOUSE_PRESSED) {
            bStart = true; //设置状态值
            startTime = endTime = System.currentTimeMillis();
            //为初始时间和结束时间赋值
            repaint(); //调用 repaint() 方法
            new Thread(this).start();
        } else if (e.getID() == MouseEvent.MOUSE_RELEASED) {
            //鼠标释放时，终止计时线程，并重绘窗口表面上的内容
            bStart = false; //设置状态值
            repaint(); //调用 repaint() 方法
        }
        super.processMouseEvent(e);
    }
    public void paint(Graphics g) { //重写 paint() 方法
        //创建时间的格式
        SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
        Date elapsedTime = null; //创建 Data 变量
        try {
            //获取“00:00:00”对应的时间
            elapsedTime = sdf.parse("00:00:00");
        } catch (Exception e) {
        }
        //为所经历的时间段赋值
        elapsedTime.setTime(endTime - startTime + elapsedTime.getTime());
        String display = sdf.format(elapsedTime); //获取相应格式所经历的时间值
        //画秒表的界面
        g.drawRect(0, 0, 250, 100); //画一个矩形
    }
}
  
```



```

        g.fill3DRect(2, 2, 248, 98, true);           //为矩形填充颜色（背景色）
        g.setColor(Color.WHITE);                     //设置字体的颜色
        Font font = new Font("Default", Font.PLAIN, 50); //创建相应属性的字体
        g.setFont(font);                             //设置字体
        g.drawString(display, 20, 50);               //画字符
    }
    public void run() {                               //重写 run() 方法
        while (bStart) {                             //根据状态值实现循环
            try {
                Thread.sleep(500);
            } catch (Exception e) {
                e.printStackTrace();
            }
            endTime = System.currentTimeMillis();    //为结束时间赋值
            repaint();                               //调用 repaint() 方法
        }
    }
}

```

【代码解析】

- ❑ 在上述代码中，首先调用 `paint()` 方法初始化秒表的界面，这时如果发生鼠标事件，则会调用 `processMouseEvent()` 方法。在具体执行 `processMouseEvent()` 方法时，如果为鼠标按下事件，除了对开始时间、结束时间赋值及修改状态值为 `true` 外，还会启动一个新线程。新线程会根据状态值的值为结束时间赋值。如果为鼠标释放事件，则会修改状态值为 `false`，从而使新线程结束。
- ❑ 秒表类 `StopWatch` 之所以继承组件 `Canvas`，而不是其他组件，是因为组件 `Canvas` 是最基本、最简单的 GUI 功能的组件。秒表类继承 `Canvas` 类，主要就是需要继承 GUI 最基本的功能和事件处理功能。
- ❑ 秒表类 `StopWatch` 的事件处理过程是，首先通过“`enableEvents(AWTEvent.MOUSE_EVENT_MASK)`”语句使秒表类能够产生鼠标事件，接着通过在 `processMouseEvent()` 方法中编写鼠标事件发生后的运行过程。
- ❑ 在显示时间的代码中，首先通过“`SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");`”语句创建时间的 `HH:mm:ss` 格式。接着通过“`elapsedTime.setTime(endTime - startTime + elapsedTime.getTime());`”语句获取近的时间段。最后通过 `sdf.format()` 方法转换时间段为所有显示格式的时间段。

在 `elapsedTime.setTime(endTime - startTime + elapsedTime.getTime())` 方法中，其参数为什么是 `endTime - startTime + elapsedTime.getTime()`，而不是 `endTime - startTime` 呢？

如果修改成 `elapsedTime.setTime(endTime - startTime)` 语句，那么即使 `endTime - startTime` 等于 0，但是 `elapsedTime` 显示的时间却不是“00:00:00”，而是“08:00:00”。查看 API 帮助文档可以知道，时间在计算机内存中也是用一个长整数来表示的，虽然内存中的长整数等于 0 时，但是由于 `Date` 类考虑了本地时区问题，所以表示的时间就不一定为“零点：零分：零秒”。

为了解决上述问题，可以通过 `elapsedTime = sdf.parse("00:00:00")` 语句求出显示时间为“00:00:00”的时间对象在内存中对应的那个长整数，然后在这个基础上加上计时器所记下的时间值，最后就可以显示出预期的结果了。

11.2.2 测试秒表的类

TestStopWatch.java 类用来测试秒表类是否能够实现预期的效果, 该类的具体内容如代码 11.2 所示。

代码 11.2 测试秒表类: TestStopWatch.jav

```
public class TestStopWatch {
    public static void main(String[] args) {
        Frame f = new Frame("秒表");           //创建窗口对象
        f.add(new Stopwatch());                 //添加秒表对象到窗口上
        //设置窗口的大小
        f.setSize(250, 100);
        f.setVisible(true);                     //使窗口显示
        f.addWindowListener(new WindowAdapter() { //为窗口注册关闭功能
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }
}
```

【代码解析】

在上述代码中, 首先创建了一个窗口对象和秒表对象, 然后把秒表对象添加到窗口对象里, 让窗口显示出秒表。

11.3 知识点扩展——事件机制的基础知识

用户界面主要实现两方面的功能, 一方面可以通过用户界面上的组件对应用程序进行操作, 另一方面应用程序可以通过用户界面上的组件收集用户的操作信息。用户界面并不对用户的各种操作结果负责, 而是交由相应的事件处理程序来处理。因此设计一个用户喜欢的界面, 相关事件的处理是少不了。

11.3.1 事件处理机制

当单击窗口标题栏上的关闭按钮后, 就可以关闭窗口并结束应用程序, 这是因为在具体实现窗口时添加了事件监听器。那么什么是事件监听器呢? 如何为窗口添加事件监听器?

如果想解决上面的问题, 首先需要了解 Java 语言的事件处理机制, 该机制涉及事件的 4 个重要概念, 分别介绍如下。

- ❑ 事件 (Event): 用户对组件的操作, 对于组件对象来说, 除了自己本身提供给别人操作它的方法外, 还可以因为一些外来事情的发生做一些相应的处理。所谓外来事情, 就是其他人指示组件的一些动作, 这些动作就是事件。
- ❑ 事件源: 发生事件的组件, 即事件产生的来源。
- ❑ 事件处理器: 负责处理事件的方法。

❑ 事件监听器（Listener）：事件处理器的对象，即对事件进行处理的方法是放在事件监听器对象中的。

事件、事件源和事件处理器的关系如图 11.7 所示，在上图中三者是通过委托处理模式来联系，即事件处理器（事件监听器对象）首先与组件（事件源）建立关联，当组件受到外部作用（事件）时，组件会产生一个相应的事件对象，并把此对象传给与之关联的事件处理器，事件处理器就会被启动并执行相关的代码 来处理该事件。

对于窗口组件（Frame）来说，如果需要处理被关闭这个事件，则必须委托某个类来负责 Frame 对象被关闭的事件处理。介于事件产生的来源和负责处理事件的类之间，就是接口 Listener。某个对象如果需要去委托某个类来处理各种事件时，就需要通过 addXXXListener()方法实现监听器的注册，如果想取消某个类的委托，可以通过 removeXXXListener 方法实现监听器的取消。

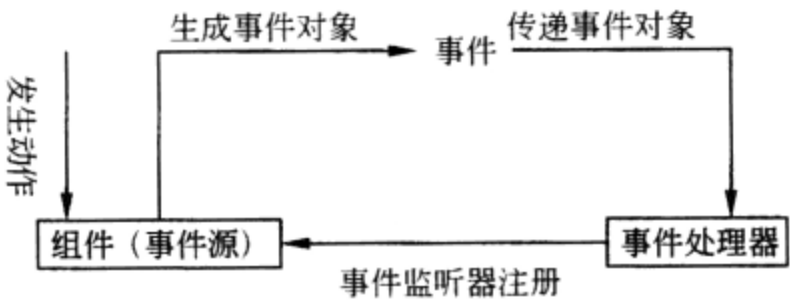


图 11.7 三者之间的关系

理解了 Java 语言中的事件处理机制后，接着就需要了解一下事件处理机制涉及的 API，它们分别如下。

1. 事件

XXXEvent: 事件的类，一般都放在 java.awt.event 包里，被称为 AWT Event。AWT Event 一般被分成两类，底层事件（Low-level Event）和语义事件（Semantic-level Event）。

⚠注意：语义事件关心的是一个具有特殊作用的 GUI 组件对应的动作发生了，而不关心这个动作是如何发生的，例如按钮被单击了等。

最常见的底层事件如图 11.8 所示，具体意义如表 11.1 所示。最常见的语义事件如图 11.9 所示，具体意义如表 11.2 所示。

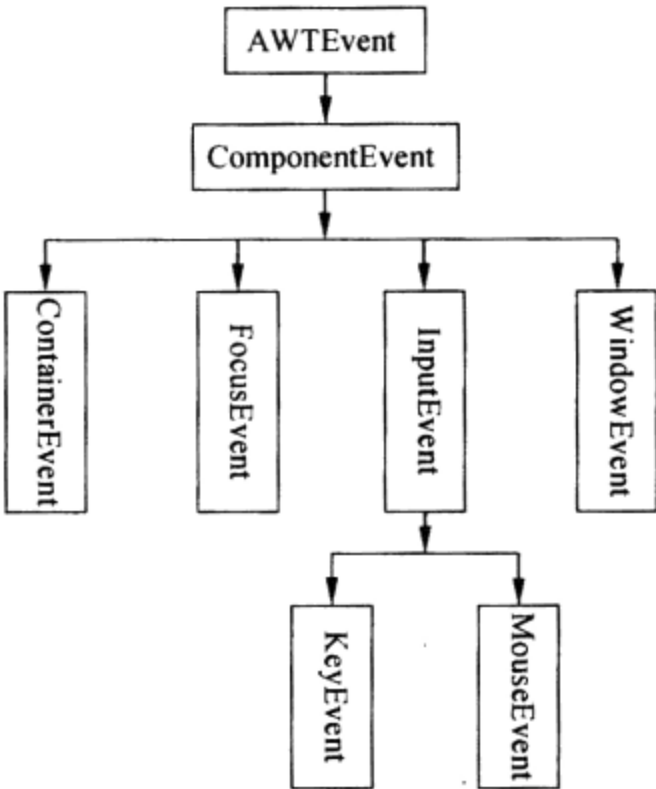


图 11.8 底层类关系

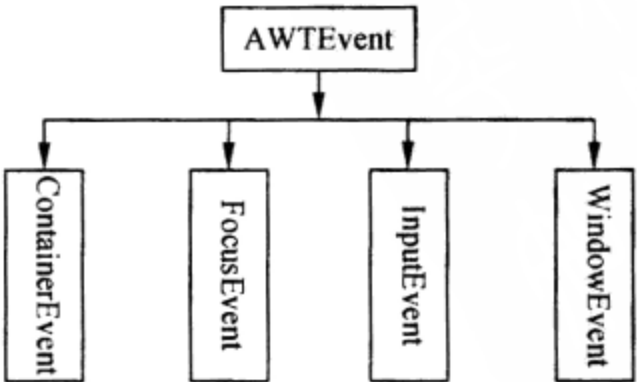


图 11.9 语义事件关系

表 11.1 底层事件

事 件 类	意 义
FocusEvent	焦点的事件
WindowEvent	窗口的事件
KeyEvent	键盘的事件
MouseEvent	鼠标的事件

表 11.2 语义事件

事 件 类	产生事件的组件	产生事件的时机
ActionEvent	Button	单击按钮
	List	选择 List 中的项目
	MenuItem	选择菜单栏项目
	TextField	按下 Enter 键
AdjustmentEvent	Scrollbar	滚动条滚动
ItemEvent	Checkbox	状态改变时
	CheckboxMenuItem	选择菜单中的项目
	Choice	选择下拉菜单中的项目
	List	选择 List 中的项目
TextEvent	TextField	文字内容改变
	TextArea	文字内容改变

2. 事件监听器

对于事件来说，其包含触发这一事件的若干具体动作，针对每种动作，都可以在与其相关的事件监听器中找到合适的处理方法。事件监听器接口的名称和事件的名称是相对的，例如 MouseEvent 事件的监听器接口名为 MouseListener；WindowEvernt 事件的监听器接口名为 WindowListener；ActionEvernt 事件的监听器接口名为 ActionListener。

11.3.2 Window 事件

对于窗体组件（Frame）经常会发生许多具体动作，例如关闭窗口、最大化窗口、最小化窗口等，当它们发生改变时，就会产生 WindowEvent。与 WindowEvent 相对应的事件处理器为 WindowListener。

下面将通过一个常见名为 WindowEventTest.java 的窗口图形用户界面，来讲解 Window 事件，具体内容如代码 11.3 所示。

代码 11.3 Window 事件处理：WindowEventTest.java

```
public class WindowEventTest {
    public static void main(String[] args) {
        Frame frame = new Frame("实现关闭功能"); //创建一个窗口对象时
        //设置窗口的相关属性
        frame.setSize(300, 300);
    }
}
```


```

        frame.setVisible(true);
        frame.addWindowListener(new CloseTest());           //注册 Window 监听器
    }
}
class CloseTest implements WindowListener {
    @Override
    public void windowActivated(WindowEvent e) {           //成为前景窗口时
    }
    @Override
    public void windowClosed(WindowEvent e) {              //关闭窗口时
    }
    @Override
    public void windowClosing(WindowEvent e) {             //关闭窗口时
        e.getWindow().setVisible(false);
        e.getWindow().dispose();
    }
    @Override
    public void windowDeactivated(WindowEvent e) {         //成为背景窗口时
    }
    @Override
    public void windowDeiconified(WindowEvent e) {         //由最小化还原时
    }
    @Override
    public void windowIconified(WindowEvent e) {           //窗口缩小到最小时
    }
    @Override
    public void windowOpened(WindowEvent e) {              //窗口打开
    }
}

```

【代码解析】

- ❑ 上述代码中创建了一个实现接口 WindowListener 的类 CloseTest, 该接口包含 Frame 组件发生的 7 种动作。在处理关闭的方法中, 实现了关闭窗口功能。
- ❑ 在具体实现关闭窗口的功能时, 可以根据 WindowEvent 事件对象的 getWindow() 方法返回发生事件的组件对象, 然后再通过 setVisible() 和 dispose() 方法实现窗口的关闭。
- ❑ Frame 组件对象 frame 如果想注册 WindowListener 处理器, 可以通过 addWindowListener() 方法来实现。

 **注意:** windowClosed() 方法指的是整个窗口被关闭后所产生的事件, 而 windowClosing() 方法指的是 Frame 组件右上角的交叉按钮被单击后所产生的事件。

在 WindowEventTest 程序代码中, 由于只想处理单击窗口标题栏上的关闭按钮这种动作, 对于其他窗口的动作不关心, 所以只对 windowClosing() 方法进行编码。根据面向对象的语法可以知道, 如果要实现接口, 则必须要实现接口中的所有方法。为了简化事件监听器的编写, 在 API 中定义了相应事件监听器接口的实现类, 这些类被称为事件适配器。在事件适配器中实现了相应事件监听器的所有方法, 但不做任何事情。


下面将修改 WindowEventTest 类, 使其通过 WindowAdaper 事件适配器来实现事件处理方法, 具体内容如代码 11.4 所示。

代码 11.4 Window 事件处理: WindowEventTest.java

```

public class WindowEventTest {
    public static void main(String[] args) {
        Frame frame = new Frame("实现关闭功能");    //创建一个窗口对象时
        //设置窗口相关属性
        frame.setSize(300, 300);
        frame.setVisible(true);
        frame.addWindowListener(new CloseTest()); //注册 Window 监听器
    }
}
class CloseTest extends WindowAdapter {
    @Override
    public void windowClosing(WindowEvent e) { //重写 windowClosing() 方法
        e.getWindow().setVisible(false);
        e.getWindow().dispose();
    }
}

```

运行类 WindowEventTest, 会出现如图 11.10 所示的图形界面。当单击图形界面中的  按钮时该界面就会关闭。

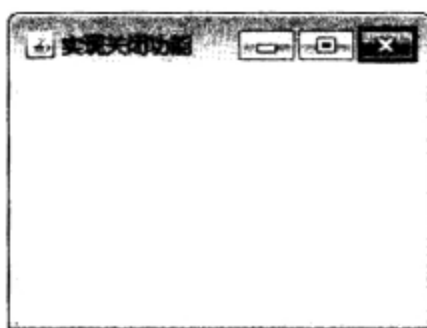


图 11.10 运行界面

【代码解析】

在上述代码中, 由于是通过继承事件适配器来实现事件的处理方法, 所以只需要重写相应的方法就可以, 而不需要关注该事件适配器的其他方法。

至于底层事件中的其他事件, 如容器事件、焦点事件和输入事件, 则将在其他章节详细介绍。

11.3.3 Mouse 事件

在 Java 语言中, 类 InputEvent 表示所有输入事件, 其有一个表示鼠标事件名为 MouseEvent 的子类。对于鼠标产生的事件可以分成两类: 一类为鼠标按键所产生的事件, 由 MouseListener 监听器来处理; 另一类为鼠标移动所产生的事件, 由 MouseMotionListener 监听器来处理。

下面将通过一个名为 Mousekey 的类详细讲解鼠标按键所产生的事件处理方法, 具体内容如代码 11.5 所示。

代码 11.5 鼠标按键所产生的事件: Mousekey.java

```

public class Mousekey extends MouseAdapter implements MouseMotionListener {
    Frame f;    //创建窗口对象
}

```



```

public static void main(String argv[]) {
    new Mousekey(); //创建 MouseKey 对象
}
public Mousekey() { //构造函数
    f = new Frame("MouseEvent test"); //为窗口对象赋值
    //为窗口注册鼠标按键事件和移动事件
    f.addMouseListener(this);
    f.addMouseMotionListener(this);
    f.setSize(300, 200); //设置窗口的大小
    f.setVisible(true); //显示窗口
}
public void mouseClicked(MouseEvent e) { //鼠标单击事件监听器
    String title; //创建一个字符串对象
    int i = e.getButton(); //获取单击的次数
    //判断鼠标的哪个键被单击
    if (i == MouseEvent.BUTTON1) {
        title = "开始单击: 用鼠标左键单击";
    } else if (i == MouseEvent.BUTTON3) {
        title = "开始单击: 用鼠标右键单击";
    } else {
        title = "开始单击: 用鼠标滚轮单击";
    }
    title = title + ("一共单击了" + e.getClickCount()); //为字符串赋值
    f.setTitle(title); //设置窗口对象的标题
}
public void mouseEntered(MouseEvent e) { //鼠标光标进入组件事件监听器
    f.setTitle("鼠标移动到按钮上");
}
public void mouseExited(MouseEvent e) { //鼠标光标离开组件事件监听器
    f.setTitle("鼠标从按钮上移开");
}
public void mousePressed(MouseEvent e) { //鼠标按键被按下事件监听器
    String title;
    int i = e.getButton();
    if (i == MouseEvent.BUTTON1) {
        title = "鼠标按下: 用鼠标左键按下";
    } else if (i == MouseEvent.BUTTON3) {
        title = "鼠标按下: 用鼠标右键按下";
    } else {
        title = "鼠标按下: 使用鼠标滚轮按下";
    }
    f.setTitle(title);
}
@Override
public void mouseReleased(MouseEvent e) { //鼠标按键被释放事件监听器
    String title;
    int i = e.getButton();
    if (i == MouseEvent.BUTTON1) {
        title = "鼠标释放: 用鼠标左键释放";
    } else if (i == MouseEvent.BUTTON3) {
        title = "鼠标释放: 用鼠标右键释放";
    } else {
        title = "鼠标释放: 使用鼠标滚轮释放";
    }
    f.setTitle(title);
}
public void mouseMoved(MouseEvent e) { //鼠标移动事件监听器

```

```
int x = e.getX();
int y = e.getY();
f.setTitle("鼠标在 (" + x + ", " + y + ") " + "地方");
}
public void mouseDragged(MouseEvent e) {           //鼠标拖动事件监听器
}
}
```

运行类 Mousekey，将会出现该类的初始化图形界面，这时该界面的标题就是“鼠标事件”。如果把鼠标移动到该图形界面上，该界面的标题就会显示出鼠标在界面中的坐标；如果鼠标从界面移开，该界面的标题显示为“鼠标从按钮上移开”，具体过程如图 11.11 所示。

在初始化界面中，如果左击一下，该界面的标题就会变成“用鼠标左键单击一共单击了 1”，如果左击两下，该界面的标题就会变成“用鼠标左键单击一共单击了 2”，具体过程如图 11.12 所示。



图 11.11 运行过程



图 11.12 左键单击过程

右击初始化界面，该界面的标题就会变为“用鼠标右键单击一共单击了 1”，如果右击两下，该界面的标题就会变为“用鼠标右键单击一共单击了 2”，具体过程如图 11.13 所示。



图 11.13 右键单击过程

【代码解析】

在上述代码中，通过继承 MouseAdapter 适配器及实现 MouseMotionListener 监听器，使类 Mousekey 发生鼠标的所有事件。对于窗口对象，当发生键盘按键的各种事件时，会在该窗口的标题上显示出相应的信息。

通过 API 帮助文档可以发现，MouseEvent 类中常用的方法如表 11.3 所示，如果需要判断通过 getButton()方法得到的值代表哪个键，可以查看表 11.4。

表 11.3 MouseEvent类的常用方法

方 法	功 能
getSource()	获取产生事件的组件
getButton()	获取触发事件时按下、释放或单击按键的 int 值
getClickCount()	获取单击按键的次数

表 11.4 静态常量

静 态 常 量	值	代 表 的 键
BUTTON1	1	鼠标左键
BUTTON2	2	鼠标滚轮
BUTTON3	3	鼠标右键

11.3.4 Key 事件

在 Java 语言中，类 InputEvent 表示所有输入事件，其有一个表示键盘事件的子类 KeyEvent。对于键盘产生的事件由 KeyListener 监听器来处理。

下面将通过一个名为 KeyEventTest 的类，详细讲解键盘按键所产生的事件的处理方法，具体内容如代码 11.6 所示。

代码 11.6 键盘按键所产生的事件：KeyEventTest.java

```
public class KeyEventTest extends KeyAdapter {
    //创建成员变量 f 和 l
    Frame f;
    Label l;
    public static void main(String argv[]) {
        new KeyEventTest(); //创建 KeyEventTest 对象
    }
    public KeyEventTest() { //构造函数
        f = new Frame("键盘按键"); //为 KeyEventTest 对象赋值
        l = new Label(); //为 l 对象赋值
        f.add(l); //把 l 添加到窗口上
        f.addKeyListener(this); //注册键盘事件监听器
        f.setSize(200, 100); //设置窗口大小
        f.setVisible(true); //设置窗口显示
    }
    public void keyPressed(KeyEvent e) { //键被按下
        String str;
        str = "一些键被按了，它的实际编号 (" + "KeyCode )为" + e.getKeyCode();
```

```

String str1 = KeyEvent.getKeyModifiersText(e.getModifiers());
str = str + ";" + "按钮用 Modifier 表示的文字 (" + "ModifiersText) 为"
+ str1;
String str2 = KeyEvent.getKeyText(e.getKeyCode());
str = str + ";" + "按键实际的表示文本 (" + "KeyText ) 为" + str2;
l.setText(str);
}
public void keyTyped(KeyEvent e) {                //单击键
    String str;
    str = "一些可见字符键被按了, 该按键实际的表示字符 (" + "KeyChar ) 为" +
    e.getKeyChar();
    l.setText(str);
}
public void keyReleased(KeyEvent e) {              //键被释放
    super.keyReleased(e);
    String str = "按键释放";
    l.setText(str);
}
}

```

运行类 `KeyEventTest`, 会出现如图 11.14 所示的图形界面。当某个按键被按下时, 界面中就会出现该键的信息, 例如图 11.15 所示的 A 按键过程、图 11.16 所示的“Shift”按键过程和图 11.17 所示的空格按键过程。



图 11.14 初始化界面

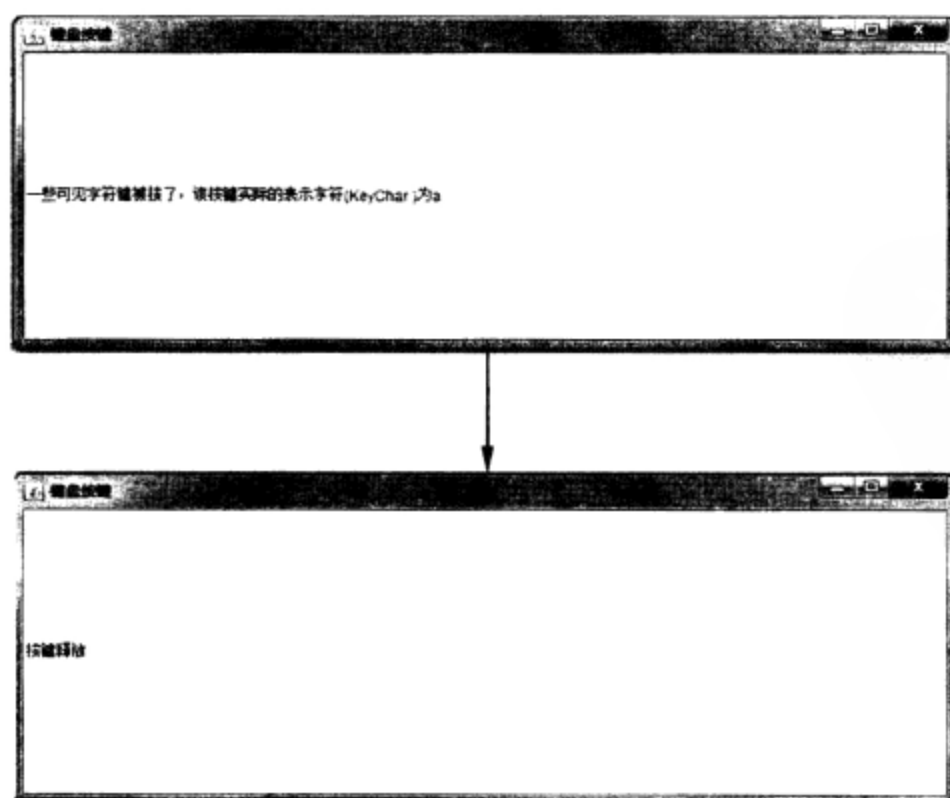


图 11.15 A 键被按下

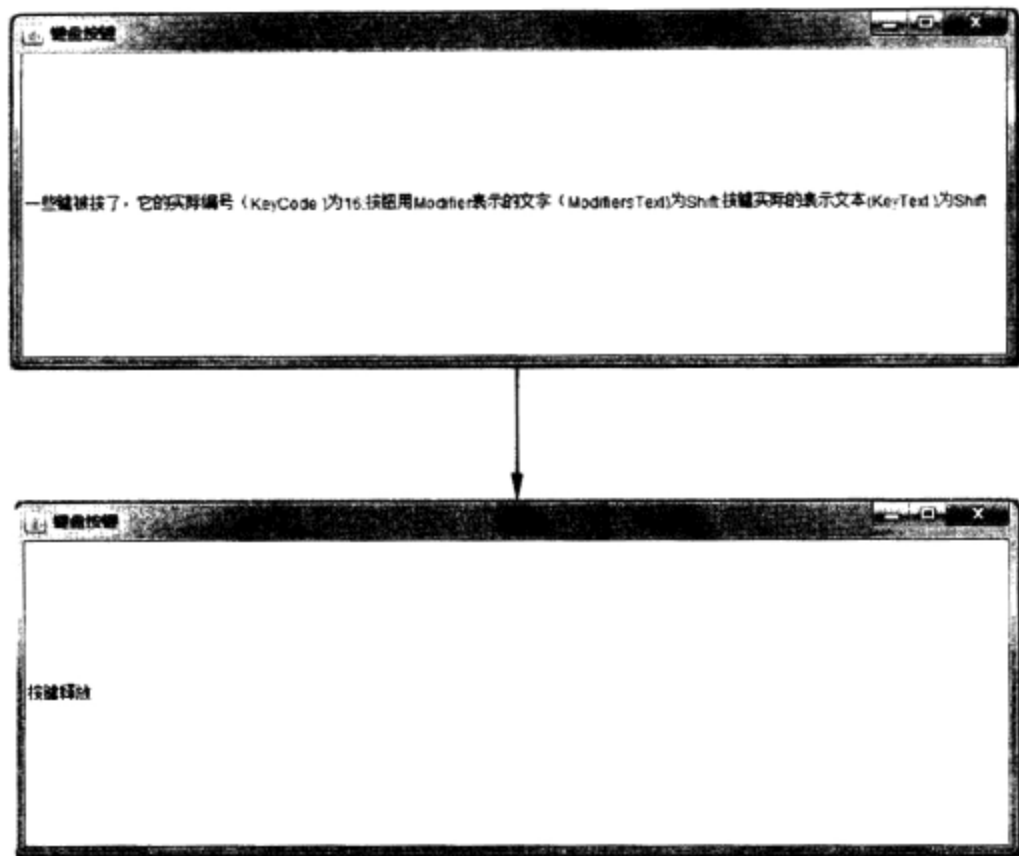


图 11.16 Shift 键被按下

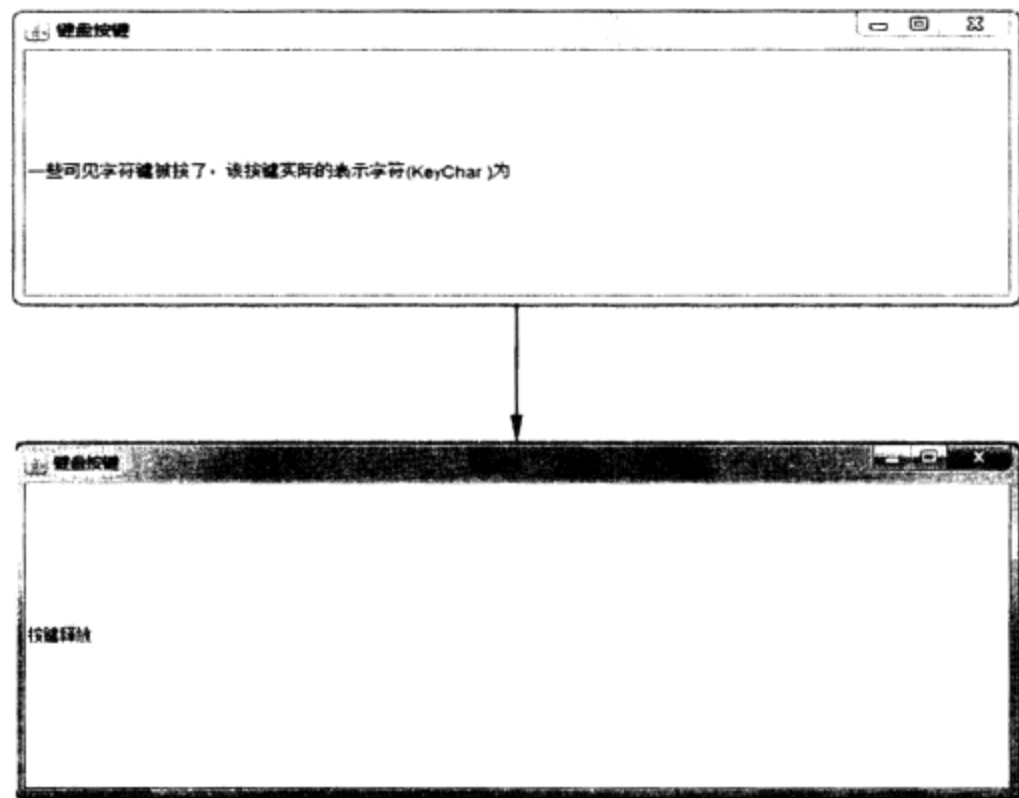


图 11.17 空格键被按下

【代码解析】

在上述代码中，通过继承 KeyAdapter 适配器使类 KeyEventTest 能够发生键盘的所有事件。在该类中为窗口对象添加了一个面板对象，当发生键盘按键的各种事件时，就会在面板上显示出相应的信息。

通过 API 帮助文档可以发现，KeyEvent 类中常用的方法如表 11.5 所示。


表 11.5 KeyEvent类的常用方法

方 法	功 能
getSource()	获取产生事件的组件

续表

方 法	功 能
getKeyChar()	获取触发事件时按下、释放或单击按键的 int 值
getKeyCode()	获取单击按键的次数
getKeyText(int KeyCode)	获得描述 KeyCode 的文本
isControlDown()	查看是否 Ctrl 键被按下
isActionKey()	查看是否按键被按下的动作
isAltDown()	查看是否 Alt 键被按下
isShiftDown()	查看是否 Shift 键被按下

对于 A 按键，如果用 KeyCode 表示，则为 65；如果用 KeyChar 表示，则为 a。对于 Shift+A 按键，如果用 KeyCode 表示仍然为 65，但是用 KeyChar 表示则为 A。对于一些不可见的字符按键，则只能获取其 Modifiers 编码的文本。KeyText 表示所有键的文本。

 注意：可见字符的按键为英文字母、数字等按键，而不可见字符的按钮则为 Alt、Shift 和 F1 等按键。

11.3.5 其他底层事件

在 Java 语言中除了前面介绍的 Windows 事件、Mouse 事件和 key 事件外，还有 Focus 事件和 Container 事件。由于后 2 个事件不如前面 3 个事件复杂繁琐，所以本节只是简单地介绍一下。

对于 ContainerEvent 类，与其对应的监听器为 ContainerListener 接口，该接口中事件的处理方法如表 11.6 所示。

表 11.6 ContainerListener 接口的方法

方 法	功 能
componentAdd()	当有 component 对象添加时执行的方法
componentRemove()	当有 component 对象删除时执行的方法

在 Windows 系统中，同一时间内只能有一个窗口能被进行各种操作，习惯上这个窗口称为“活动窗口”。操作除了可以直接作用于活动窗口外，还可以作用于活动窗口里的各种组件。同理在同一时间里，操作也只能作用于活动窗口上的一个组件，习惯上这个过程叫做组件“获取焦点”。

对于 FocusEvent 类，与其对应的监听器为 FocusListener 接口，该接口中事件的处理方法如表 11.7 所示。

表 11.7 FocusListener 接口的方法

方 法	功 能
focusGained()	组件获取焦点
focusLost()	组件失去焦点

11.3.6 事件的高级编写方法

在 WindowEventTest 程序代码中，虽然事件监听器的类 CloseTest 和产生 GUI 组件的类 WindowEventTest 是两个完全分开的类，但是事件监听器的类中访问的对象正好是事件源。如果事件监听器的类代码中需要访问非事件源的对象，则事件监听器的类和产生 GUI 组件的类必须在同一个类中。例如，要实现通过窗口中的按钮关闭窗口功能，具体步骤如下。

(1) 创建一个名为 ButtonClose.java 的类，该类实现通过窗口中的按钮关闭窗口的功能，具体内容如代码 11.7 所示。

代码 11.7 按钮关闭窗口：ButtonClose.java

```
public class ButtonClose implements ActionListener {
    Frame frame = new Frame("实现关闭功能");    //创建窗口对象
    public static void main(String[] args) {
        ButtonClose f = new ButtonClose();    //创建 ButtonClose 类对象
        //设置窗口对象的大小及显示相应组件
        f.frame.setSize(300, 300);
        f.frame.setVisible(true);
        Button button = new Button("关闭");    //创建按钮对象
        f.frame.add(button);    //为窗口添加按钮对象
        button.addActionListener(f);    //为按钮添加注册器
    }
    @Override
    public void actionPerformed(ActionEvent e) {    //监听(ActionEvent) 事件
        frame.setVisible(false);    //设置窗口不显示
        frame.dispose();    //释放资源
    }
}
```

【代码解析】

- ❑ 在上述代码中，由于在 actionPerformed() 方法中需要访问非事件源对象 frame，所以该对象不能在 main() 方法定义，而是被定义为类变量。这是因为一个方法不能访问另一个方法中的变量。
- ❑ 在 main() 方法中，不能直接调用成员变量 frame。这是因为 main() 方法为静态方法，而 frame 变量不是静态变量，所以需要通过 new ButtonClose().frame 方式来实现。

(2) 上述代码虽然实现了预期的效果，但是 main() 方法中的代码太不规范，修改 ButtonClose 类的内容如代码 11.8 所示。

代码 11.8 按钮关闭窗口：ButtonClose2.java

```
public class ButtonClose2 implements ActionListener {
    Frame frame = new Frame("关闭功能");    //创建窗口对象
    private void init() {    //编写初始化方法
        frame.setSize(300, 300);
        frame.setVisible(true);
    }
}
```

```

        Button button = new Button("关闭");
        frame.add(button);
        button.addActionListener(this);
    }
    public static void main(String[] args) {
        ButtonClose2 test = new ButtonClose2(); //创建 ButtonClose2 类对象
        test.init();
    }
    @Override
    public void actionPerformed(ActionEvent e) { //编写监听器
        frame.setVisible(false);
        frame.disable();
        System.exit(0);
    }
}

```

【代码解析】

在上述代码中,把不符合规范的代码抽离出来作为方法 `init()` 的方法体,该方法主要实现初始化窗口的功能。

(3) 为了使程序看起来更紧凑一点,可以利用匿名类的方式修改 `ButtonClose2` 的代码,具体内容如代码 11.9 所示。

代码 11.9 按钮关闭窗口: `ButtonClose3.java`

```

public class ButtonClose3 {
    Frame frame = new Frame("关闭功能"); //创建窗口对象
    private void init() { //编写初始化方法
        frame.setSize(300, 300);
        frame.setVisible(true);
        Button button = new Button("关闭");
        frame.add(button);
        button.addActionListener(new ActionListener() {
            //利用匿名方法注册监听器
            public void actionPerformed(ActionEvent e) {
                frame.setVisible(false);
                frame.disable();
                System.exit(0);
            }
        });
    }
    public static void main(String[] args) {
        ButtonClose3 test = new ButtonClose3(); //创建 ButtonClose3 类对象
        test.init(); //初始化
    }
}

```

【代码解析】

事件监听器的类只用在—个组件中时,为了让程序更紧凑,可以使用匿名类的语法产生事件监听器对象。在上述代码中, `ActionEvent` 事件的监听器对象就是使用匿名类的语法来实现的。

11.4 小 结

本章主要通过 Java 语言中的事件机制实现秒表多线程项目，虽然该项目比较简单，但是涉及的知识点比较多。在具体实现的过程中，要特别注意该项目是如何注册事件和监听事件的。为了让读者能够更好地掌握 Java 语言的事件机制，在本章的最后还详细介绍了 Java 语言中经常遇到的各种事件。

第 12 章 捉迷藏游戏（事件）

通过第 11 章的学习后，读者对于编写普通的事件的项目还可以，但是如果编写一些实现复杂功能的项目则显的有些不足。为了能够编写出复杂事件的项目，本章将详细介绍事件内部处理的机制，不仅详细讲解通过按钮模拟的“捉迷藏游戏”项目，还将详细介绍事件的详细处理过程。

本章的学习目标如下：

- ❑ 掌握“捉迷藏游戏”项目；
- ❑ 理解事件处理的内部机制。

12.1 捉迷藏游戏原理

“捉迷藏游戏”通过两个按钮来模拟捉迷藏的效果，在具体运行该游戏时，如果单击按钮，则该按钮就会消失，然后出现在界面的其他地方。

12.1.1 项目结构框架分析

对于捉迷藏游戏，根据面向对象的思想，需要创建一个对象，即按钮。捉迷藏游戏项目目录如图 12.1 所示，该项目中的两个类，分别为自定义按钮类 `MyButton` 和测试按钮的类 `TestMyButton`。



图 12.1 项目目录

12.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括捉迷藏游戏的初始化界面和通过鼠标单击按钮模拟的捉迷藏效果。

1. 捉迷藏游戏的初始化界面

当运行测试定义按钮的类 `TestMyButton` 后，会出现如图 12.2 所示的初始界面。在该界面的下方会出现一个名为“你来抓我呀！”的按钮。

2. 捉迷藏游戏的效果

在具体玩捉迷藏游戏时，如果单击界面下方的“你来抓我呀！”的按钮，该按钮就会从界面下方消失，然后在界面的上方出现；这时如果接着单击界面上方的“你来抓我呀！”按钮，同样也会实现该按钮从界面上方消失，然后在界面下方出现，具体过程如图 12.3 所示。



图 12.2 秒表的初始界面

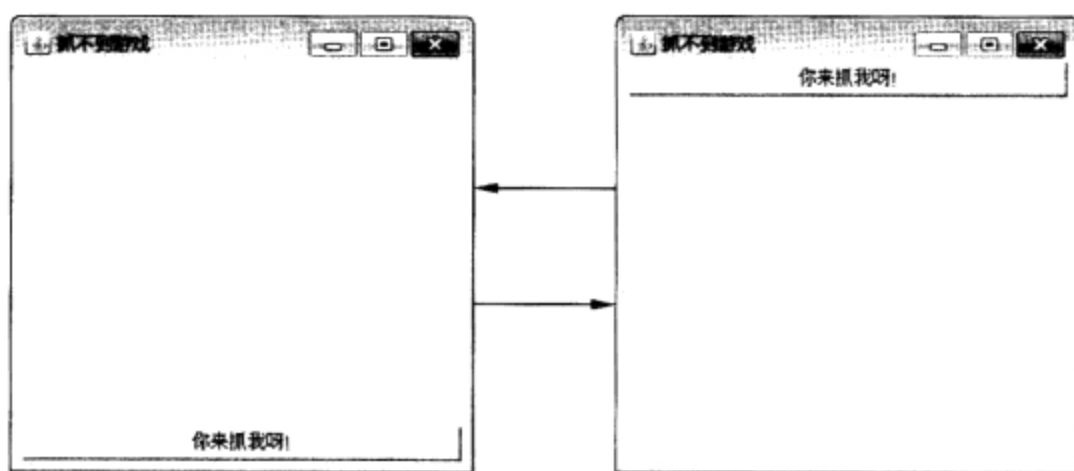


图 12.3 运行过程

12.2 捉迷藏游戏的实现过程

本章通过事件机制的相关知识来实现捉迷藏游戏项目，具体程序架构如图 12.4 所示，它包含一个类对象 `MyButton.java` 和一个进行测试的类 `TestMyButton.java`。

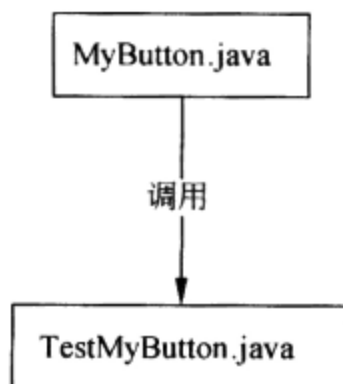


图 12.4 程序关系图

12.2.1 捉迷藏游戏项目的原理

当游戏玩家在玩捉迷藏游戏时感到非常惊讶，因为在一个窗口上显示一个按钮，当鼠标移动到该按钮上时，其就移动到其他位置，鼠标永远无法单击到按钮。如果想了解上述

游戏效果的原理，可以先听听下面的故事。

假设动画片“火影忍者”中的鸣人跟小樱在玩“抓人”游戏，一开始鸣人就把使用“分身术”出现的替身隐身起来，所以抓鸣人的小樱只能看到一个鸣人。当小樱靠近鸣人时，鸣人马上使用隐身术消失，同时让他的替身现身，这样小樱就以为鸣人跑到他替身的位置上。同样小樱靠近替身时，该替身也马上使用隐身术消失，同时通知鸣人现身，这样小樱还以为鸣人又跑到他原来的地方。如此往复，小樱永远抓不到鸣人。

在捉迷藏游戏项目中，虽然游戏玩家只看到一个按钮在界面上，其实界面一共存在两个按钮，其中一个在界面中显示，另一个不显示。当鼠标移到显示出来的按钮时，该按钮就会在界面上隐藏，同时让另一个不显示的按钮在界面中显示出来。

12.2.2 自定义按钮类

MyButton.java 类为自定义的按钮类，在该类的鼠标监听器中实现了隐藏自己和显示另一个按钮的功能。具体内容如代码 12.1 所示，该类的 UML 如图 12.5 所示。

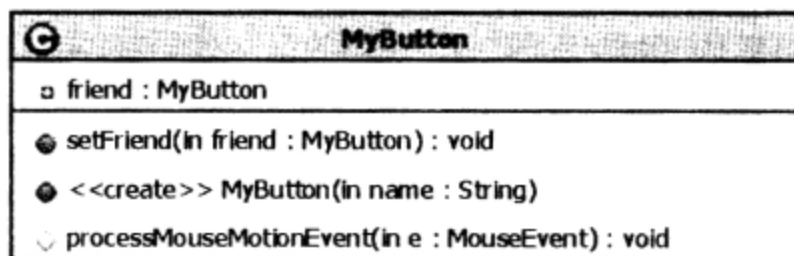


图 12.5 按钮类图

代码 12.1 按钮类：MyButton.java

```
class MyButton extends Button {
    private MyButton friend; //创建一个 MyButton 类型属性
    public void setFriend(MyButton friend) { //编写属性的 set 方法
        this.friend = friend;
    }
    public MyButton(String name) { //构造函数
        super(name);
        //具有发生鼠标事件的能力
        enableEvents(AWTEvent.MOUSE_MOTION_EVENT_MASK);
    }
    protected void processMouseEvent(MouseEvent e) { //事件监听器方法
        setVisible(false); //不显示
        friend.setVisible(true); //显示另一个按钮
    }
}
```

【代码解析】

- 由于类 MyButton 要继承按钮的一些特性，所以该类要继承 Button 类。为了使自定义的按钮类能够发生鼠标事件，所以使用了语句 enableEvents(AWTEvent.MOUSE_MOTION_EVENT_MASK)。最后在鼠标监听器中编写了实现使自己不显示，另一个按钮显示的功能。
- 对于 MyButton 类型的属性 friend，其实就相当于自定义按钮类的替身。

12.2.3 测试的类

TestMyButton.java 类用来测试自定义按钮类是否能够实现预期的效果，该类的具体内容如代码 12.2 所示。

代码 12.2 测试自定义按钮类：TestMyButton.jav

```
public class TestMyButton {
    public static void main(String[] args) {
        //创建两个按钮对象
        MyButton btn1 = new MyButton("你来抓我呀!");
        MyButton btn2 = new MyButton("你来抓我呀!");
        //设置按钮对象的 friend 属性
        btn1.setFriend(btn2);
        btn2.setFriend(btn1);
        btn1.setVisible(false);           //按钮 btn1 不显示
        Frame f = new Frame("捉迷藏游戏"); //创建一个窗口对象
        f.add(btn1, "North");             //将 btn1 增加到 f 的北部
        f.add(btn2, "South");             //将 btn2 增加到 f 的南部
        f.setSize(300, 300);              //设置窗口的大小
        f.setVisible(true);
        btn1.setVisible(false);
    }
}
```

【代码解析】

在上述代码中，首先创建了两个自定义按钮对象，然后分别设置对方为自己的 friend 属性值。接着创建一个包含这两个按钮对象的窗口对象。

12.3 知识点扩展——事件机制的高级知识

对于图形用户界面中的某个组件，当用户对该组件进行一次操作后，可能不仅触发底层事件，也可能触发语义事件。那么应用程序在具体处理时，会选用哪种类型的事件监听器呢？对于事件，JVM 是通过什么流程来处理事件的呢？

12.3.1 事件多重应用

当用户在一个按钮上单击鼠标时，不仅会发生鼠标底层事件，而且还会发生按钮的动作语义事件。根据一般的处理原则，如果对于语义事件的处理能够满足用户的要求，就不需要再处理底层事件。但是如果用户希望在单击按钮时除了使包含按钮的窗口变小外，还希望鼠标按下时能改变按钮的内容，鼠标释放时能恢复按钮的内容，那么就需要同时处理按钮的动作事件和鼠标事件。

那么事件源、事件和事件监听器之间到底存在什么样的关系呢？下面将一一介绍。

在事件处理机制上，由于一个组件上的动作可以产生多种不同的事件，所以同一个事

件源上可以注册多种不同类型的监听器，如图 12.6 所示。

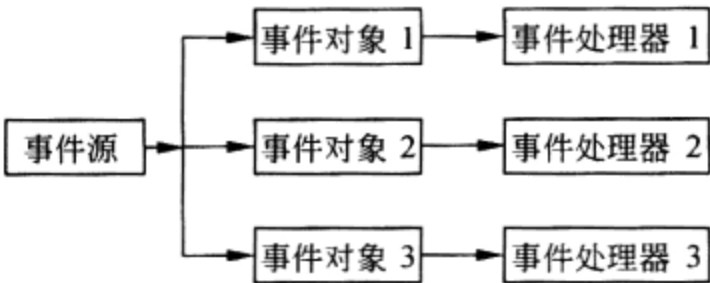


图 12.6 事件与事件源的关系

下面通过一个具体的实例 Relation1，来演示多个事件源的同一事件可以委托一个事件处理器来处理的过程，该类的 UML 如图 12.7 所示，具体内容如代码 12.3 所示。

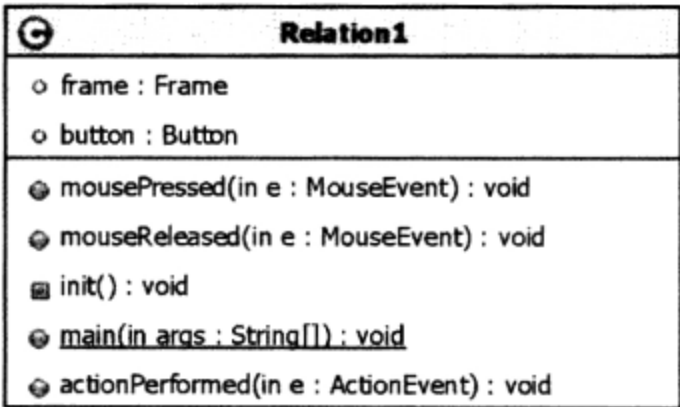


图 12.7 类的 UML

代码 12.3 关系一：Relation1.java

```
public class Relation1 extends MouseAdapter implements ActionListener {
    //创建两个成员变量
    Frame frame = new Frame("事件源事件和事件处理器的关系");
    Button button = new Button("鼠标单击前");
    private void init() { //创建一个初始化方法
        frame.add(button); //为窗体添加按钮对象
        frame.setSize(300, 300); //设置窗体的大小
        frame.setVisible(true); //使窗体显示
        button.addActionListener(this); //为按钮对象添加动作监听器
        button.addMouseListener(this); //为按钮对象添加鼠标监听器
    }
    public static void main(String[] args) { //主方法
        Relation1 test = new Relation1();
        test.init();
    }
    @Override
    public void actionPerformed(ActionEvent e) { //处理动作事件方法
        frame.setSize(600, 600);
    }
    @Override
    public void mousePressed(MouseEvent e) { //处理鼠标按住事件方法
        button.setLabel("鼠标按住");
        super.mousePressed(e);
    }
    @Override
    public void mouseReleased(MouseEvent e) { //处理鼠标释放事件方法
```

```
        button.setLabel("鼠标释放");
        super.mouseReleased(e);
    }
}
```

运行类 Relation1，过程如图 12.8 所示。

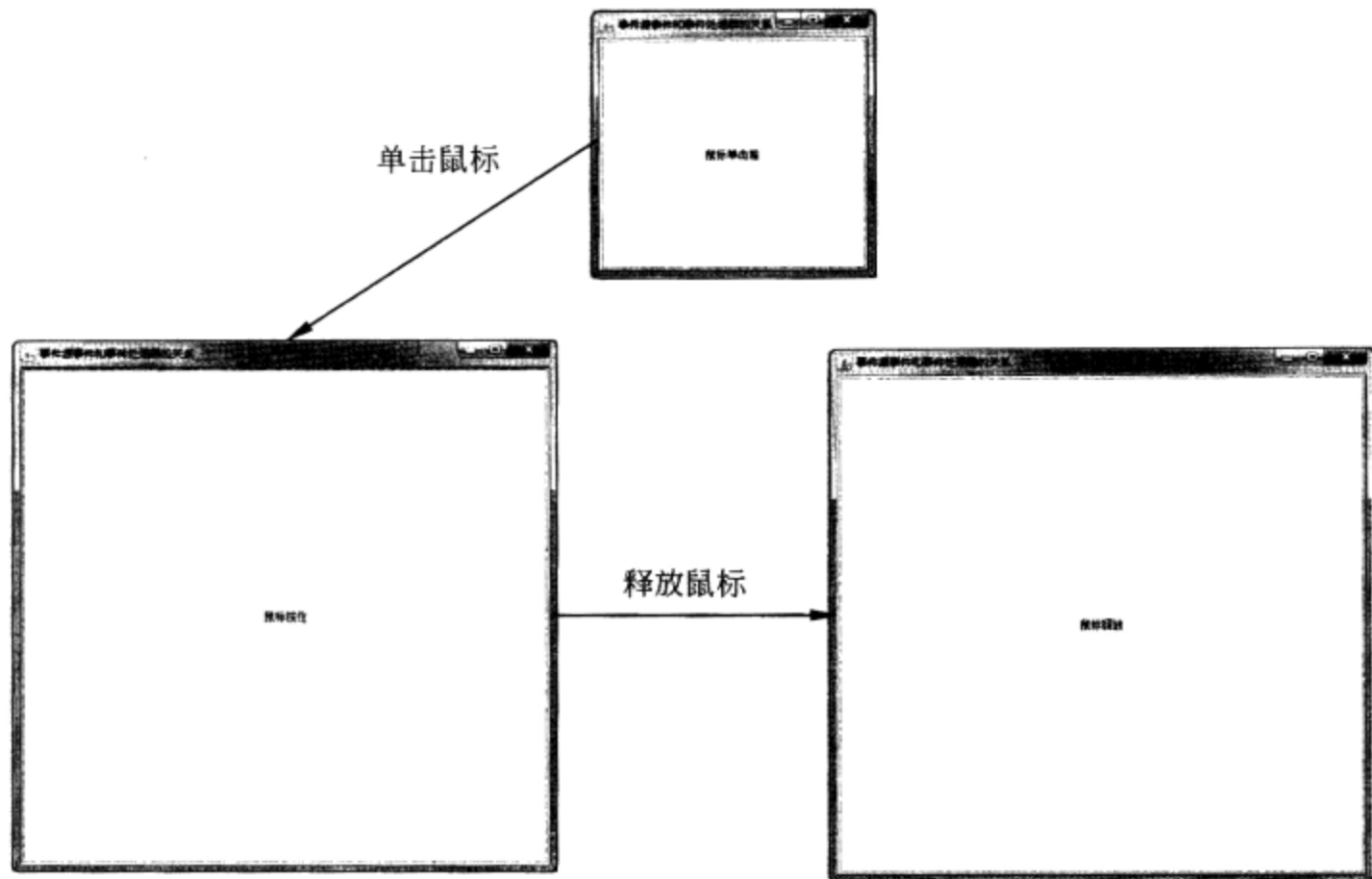


图 12.8 运行过程

【代码解析】

- ❑ 在上述代码主要实现如下功能，当用户在按钮上按下鼠标时，不仅使窗口变大，而且还修改按钮上的内容为“鼠标按住”；当释放鼠标时，修改按钮上的内容为“鼠标释放”。
- ❑ 由于需要实现处理两种事件，所以 Relation1 类不仅继承了鼠标事件的适配器 MouseAdapter，而且还实现了动作监听器 ActionListener。

在事件处理机制上，既然同一个事件源上可以产生多种不同的事件，那么多个事件源可以产生同一个事件，即多个事件源的同一事件可以由一个事件处理器来处理，如图 12.9 所示。

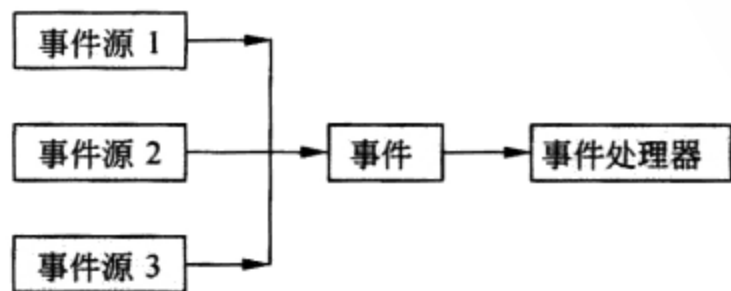


图 12.9 事件与事件源的关系

下面通过一个具体的实例 Relation2，来演示多个事件源的同一事件可以委托一个事件处理器来处理的过程，具体代码包含两个类，它们的 UML 图分别如图 12.10 和图 12.11 所

示，具体内容如代码 12.4 所示。

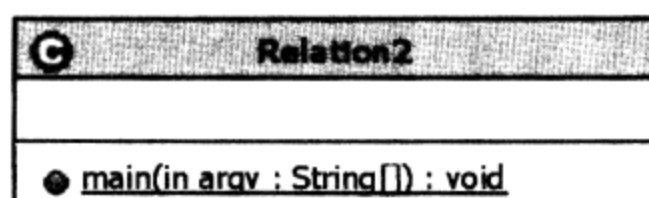


图 12.10 关系二的类图

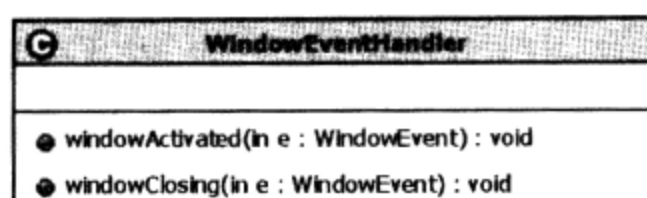


图 12.11 处理事件的监听器类图

代码 12.4 关系二: Relation2.java

```

public class Relation2 {
    public static void main(String argv[]) {
        Frame f1 = new Frame("窗口 1");
        f1.addWindowListener(new WindowEventHandler());
        f1.setSize(200, 200);
        Frame f2 = new Frame("窗口 2");
        f2.addWindowListener(new WindowEventHandler());
        f2.setSize(300, 100);
        f1.setVisible(true);
        f2.setVisible(true);
    }
}

class WindowEventHandler extends WindowAdapter {
    //事件处理器 WindowEventHandler
    public void windowActivated(WindowEvent e) {
        //改写了 windowActivated() 方法
        Frame f = (Frame) e.getSource(); //获取发生事件的源对象
        System.out.println(f.getTitle() + " 在最前面");//输出相应信息
    }
    public void windowClosing(WindowEvent e) { //改写了 windowClosing() 方法
        Frame f = (Frame) e.getSource(); //获取发生事件的源对象
        f.setVisible(false); //隐藏相应的窗口
        f.dispose(); //释放相应的资源
        System.out.println(f.getTitle() + " 隐藏");//输出相应的信息
    }
}
  
```

运行类 Relation2，过程如图 12.12 所示，输出窗口过程如图 12.13 所示。

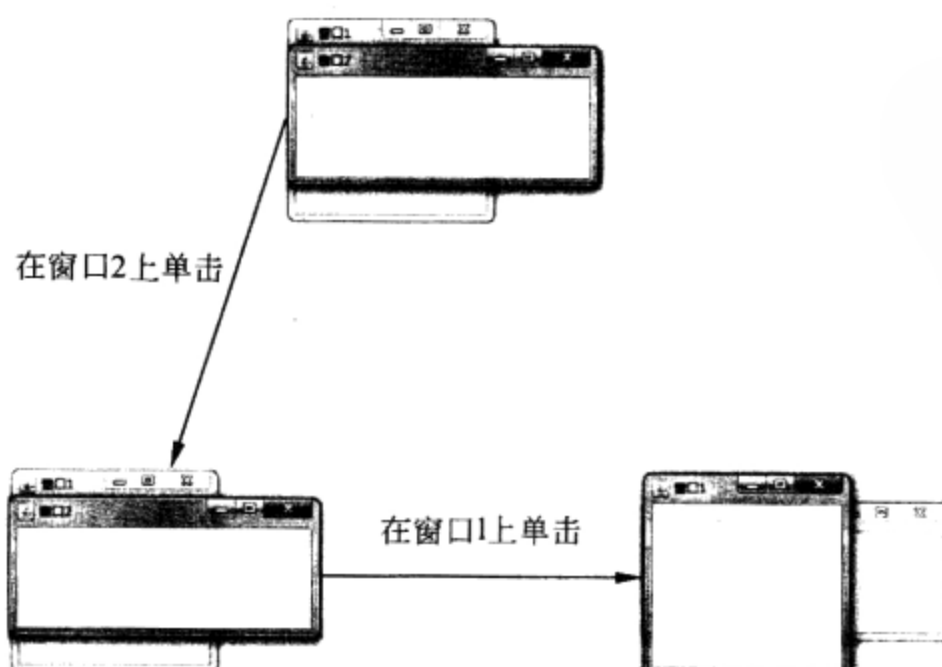


图 12.12 运行过程

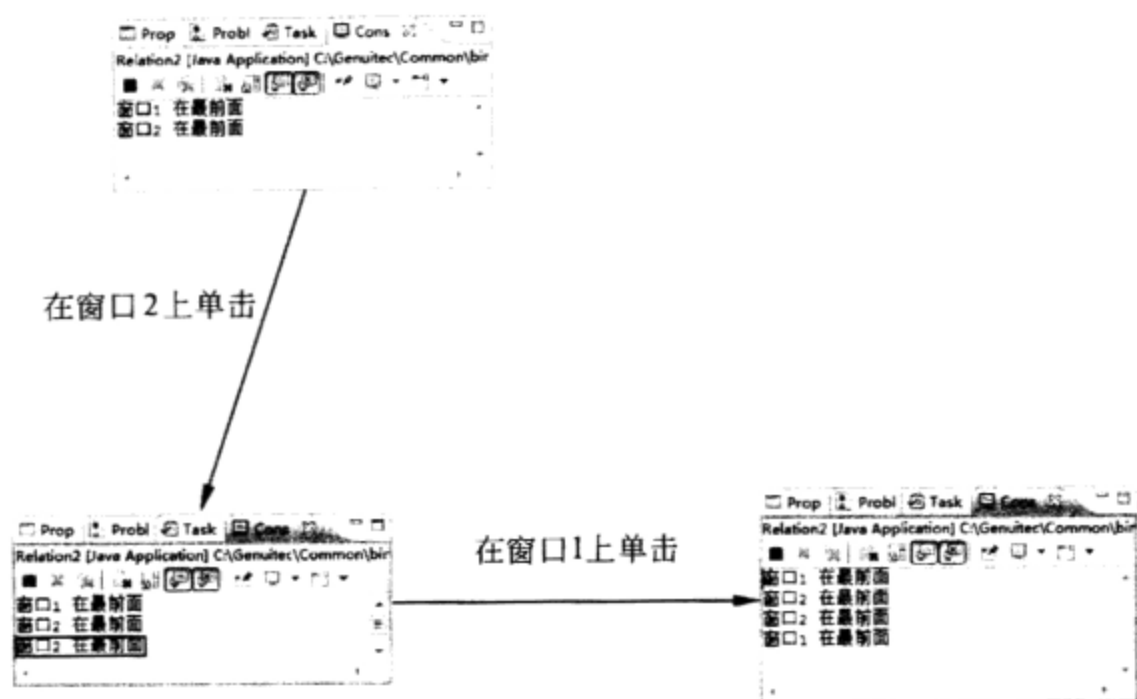


图 12.13 输出窗口过程

【代码解析】

在上述代码中，为两个窗口对象 f1 和 f2 分别注册了两个处理 WindowEvent 事件的事件处理器 WindowEventHandler1 和 WindowEventHandler2。在 WindowEventHandler1 处理器中，改写了 windowClosing()方法，输出了相应的信息。在 WindowEventHandler2 处理器中，改写了 windowClosing()方法，不仅输出了相应的信息，而且还实现退出系统的功能。

在事件处理机制上，一个事件源上也可以注册对同一事件进行处理的多个事件监听器对象，如图 12.14 所示。

下面通过一个具体的实例 Relation3.java 来，讲解同一个事件源的某个事件可以委托多个事件处理器来处理的过程，这些类的 UML 分别如图 12.15、图 12.16 和图 12.17 所示，该类的具体内容如代码 12.5 所示。

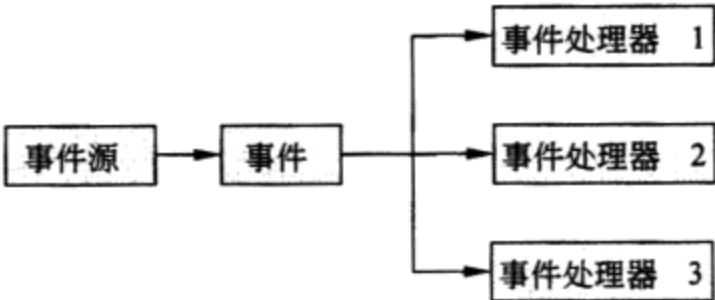


图 12.14 事件与事件处理器的关系

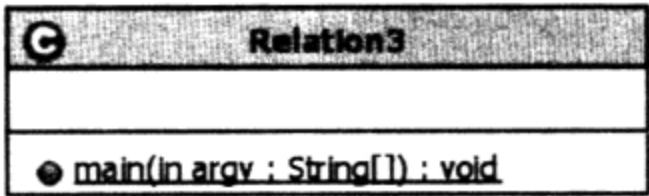


图 12.15 关系三类的类图

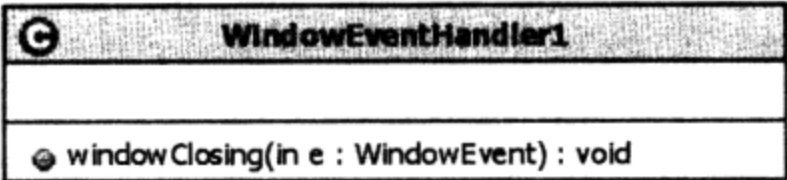


图 12.16 事件 1 监听器类图

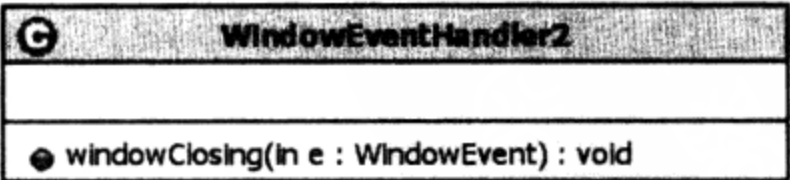


图 12.17 事件 2 监听器类图

代码 12.5 按钮关闭窗口：Relation3.java

```
public class Relation3 {
    public static void main(String argv[]) {
```

```

//创建窗口对象
Frame f = new Frame("事件源事件和事件处理器的关系");
//为 f 对象注册两个事件处理器
f.addWindowListener(new WindowEventHandler1());
f.addWindowListener(new WindowEventHandler2());
f.setSize(200, 200);           //设置窗口的大小
f.setVisible(true);           //窗口显示
}
}
class WindowEventHandler1 extends WindowAdapter {
    //定义事件处理器 WindowEventHandler1
    public void windowClosing(WindowEvent e) {
        System.out.println("Handler 1");    //输出相应的信息
    }
}
class WindowEventHandler2 extends WindowAdapter
    //定义事件处理器 WindowEventHandler2
    public void windowClosing(WindowEvent e) {
        System.out.println("Handler 2");    //输出相应的信息
        System.exit(0);                     //退出系统
    }
}

```

运行类 Relation3，过程如图 12.18 所示。

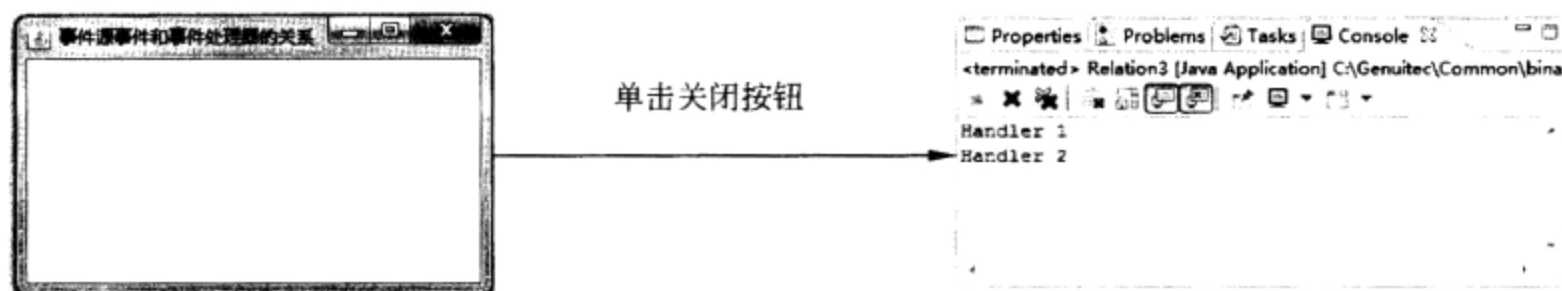


图 12.18 运行过程

【代码解析】

在上述代码中，为窗口对象 `f` 分别注册了两个处理 `WindowEvent` 事件的事件处理器 `WindowEventHandler1` 和 `WindowEventHandler2`。在 `WindowEventHandler1` 处理器中，改写了 `windowClosing()` 方法，输出了相应的信息。在 `WindowEventHandler2` 处理器中，改写了 `windowClosing()` 方法，不仅输出了相应的信息，而且还实现了退出系统的功能。


12.3.2 事件处理的详细过程

12.3.1 节讲解了事件源、事件和事件监听器三者之间的关系，本节将讲解剩下的另一个问题：JVM 是通过什么流程来处理事件的呢？

查看 API 帮助文档可以发现，组件上事件的产生与是否注册该事件的事件监听器有关。当组件上没有注册任何事件的事件监听器时，该组件会屏蔽对其的所有动作，即不管发生什么动作，任何事件都不会发生。只有在组件上注册了某种事件的事件监听器，当相应的动作发生时，该组件才可以做出响应产生相应的事件。

当一个组件产生事件后，系统会调用该组件对象的 `processEvent()` 方法来处理。在默认

情况下，`processEvent()`方法根据事件的类型调用相应的`processXxxEvent()`方法，而`processXxxEvent()`方法则会将Xxx事件传递给已经注册的相应事件监听器去处理。

 **注意：**在`processXxxEvent()`方法中，Xxx代表事件类型。

如果没有在组件上注册Xxx事件监听器，该组件是不会发生Xxx事件的，因此`processXxxEvent()`方法也根本不可能被调用。查看API帮助文档可以发现，即使没有注册事件监听器，只要调用`enableEvents()`函数设置了组件的事件，当相应的动作发生后该组件仍然能够产生对应的事件。

对于`enableEvents()`函数，具体语法如下：

```
protected final void enableEvents(long eventsToEnable)
```

在上述定义中，参数`eventsToEnable`为组件需要响应事件类型所对应的数值。在`AWTEvent`类中，定义了许多事件类型的常量，例如鼠标移动事件对应的常量为`MOUSE_MOTION_EVENT_MASK`，如果想让组件响应鼠标移动事件，具体内容如下：

```
eventsToEnable (AWTEvent.MOUSE_MOTION_EVENT_MASK)
```

下面将通过一个具体类`EventProcess`，来演示事件处理的详细过程，内容如代码12.6所示。

代码12.6 事件处理过程：EventProcess.java

```
public class EventProcess extends Button {
    Frame frame = new Frame("事件处理过程");           //创建对象 frame
    //重写 EventProcess() 方法
    public EventProcess(String name) {
        this.setLabel(name);
        enableEvents(AWTEvent.MOUSE_MOTION_EVENT_MASK);
    }
    //重写 processMouseEvent() 方法
    protected void processMouseEvent(MouseEvent e) {
        frame.setSize(600, 600);
    }
    private void init() {                                //初始化方法
        frame.add(this);
        frame.setSize(300, 300);                         //设置大小
        frame.setVisible(true);                          //设置窗口可见
    }
    public static void main(String[] args) {            //主方法
        EventProcess button = new EventProcess("按钮");  //创建按钮对象 button
        button.init();                                   //调用初始化方法
    }
}
```

运行类`EventProcess`，过程如图12.19所示。

【代码解析】

在上述代码中，通过`enableEvents()`方法使按钮能够产生鼠标事件对象，然后在鼠标事件监听器中实现了让窗口变大的功能。

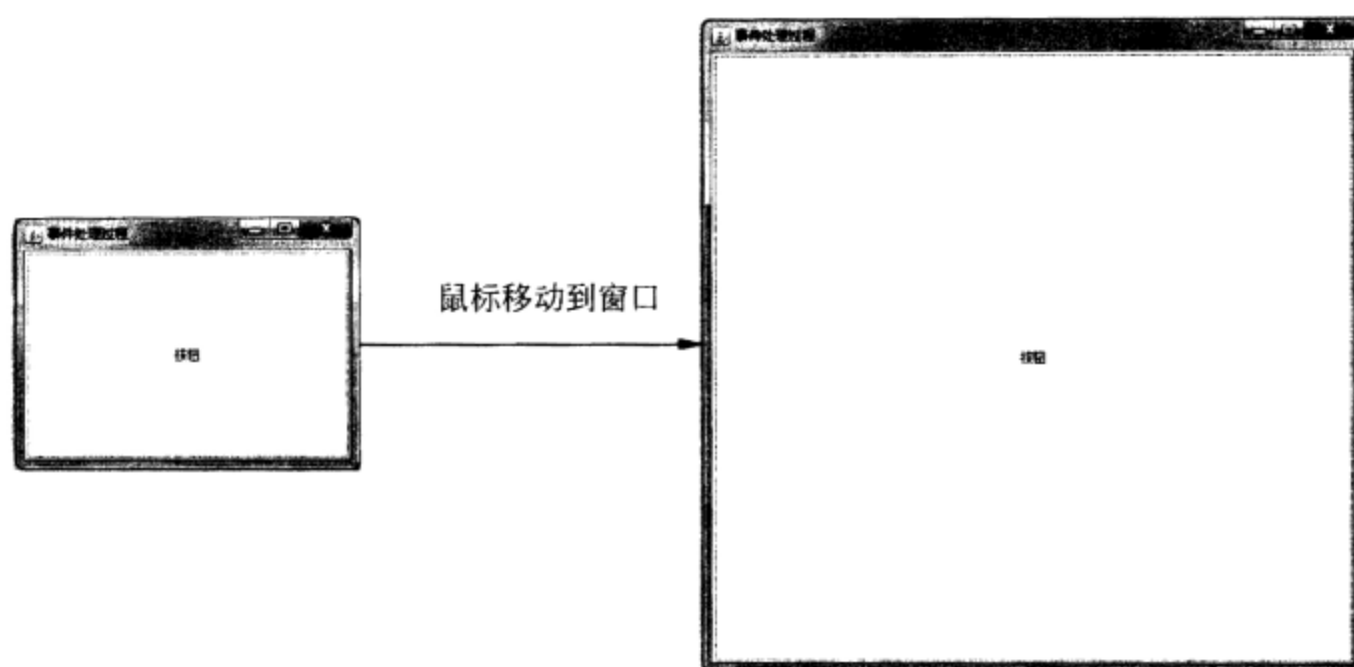


图 12.19 运行结果

12.4 小 结

本章主要通过事件的知识讲解了捉迷藏游戏，在具体实现该游戏的过程中，最主要的就是实现了拥有自定义事件的按钮类。在本章的最后还详细介绍了事件的高级知识点，分别为事件的多重应用和事件处理的详细过程。事件是游戏处理不可或缺的知识点，读者掌握后，对 Java 开发游戏会有大的帮助。

第 13 章 鼠标绘直线（绘图+事件）

在 Java 语言中利用绘图机制可以绘制许多图形，对于图形用户界面来说，一些简单的图形是必须的，所以程序员必须掌握绘图机制。对于一个应用程序，如果想实现与用户良好的互动功能，除了事件处理机制外，对于绘图的处理也是必不可少的。本章不仅通过鼠标绘直线的功能介绍 Java 语言中的绘图机制和事件处理机制，而且还详细介绍绘图的基础知识。

本章的学习目标如下：

- ❑ 掌握鼠标绘直线项目；
- ❑ 理解为什么要使线程具有同步性；
- ❑ 掌握实现线程同步的两种方式。

13.1 鼠标绘直线原理

“鼠标绘直线”项目用来模拟用鼠标绘直线的功能，在具体使用时首先用鼠标左键单击画布，然后拖动鼠标到另一个地方，最后释放鼠标左键。这时不仅在两点间绘制直线，同时还将显示出端点的坐标。

13.1.1 项目结构框架分析

对于鼠标绘直线项目，根据面向对象的思想，需要创建两个对象，即直线和窗口。鼠标绘直线项目目录如图 13.1 所示，各个目录的功能如下。

- ❑ 包 com.cjg.repaine：通过重画技术实现鼠标绘直线功能。
- ❑ 包 com.cjg.buffer：通过双缓冲技术实现鼠标绘直线功能。

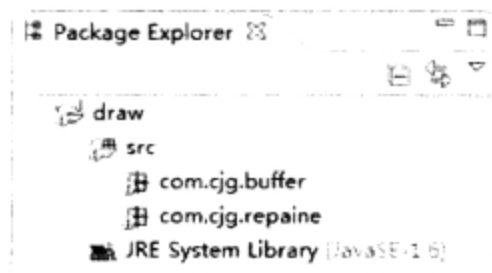


图 13.1 项目目录

13.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括鼠标绘直线的初始化，按住

鼠标功能和放开鼠标后的功能。

1. 初始化界面

当运行鼠标绘直线项目中的类 DrawLine 后，就会出现如图 13.2 所示的初始界面，该界面只是一个窗口。



图 13.2 初始界面

2. 画点和鼠标绘直线功能

如果想绘制点时，只要用鼠标左键单击画布，然后释放鼠标左键，这时就会在画布中显示出一个点，同时显示出该点的坐标。如果想绘制直线，需要在画布中单击鼠标左键，然后在不释放鼠标键的情况下移动鼠标，最后在画布的其他地方释放左键。这时就会在鼠标单击的两个地方之间显示出一条直线，同时会显示出直线端点的坐标，具体过程如图 13.3 所示。

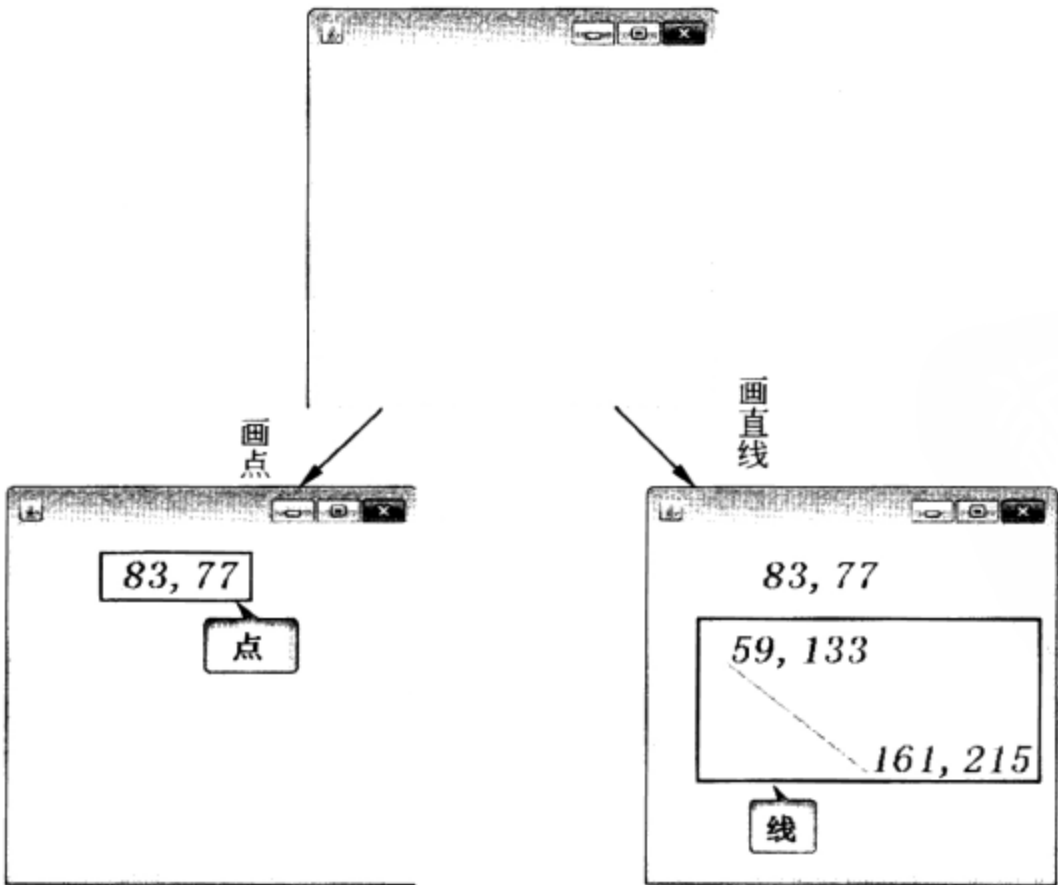


图 13.3 绘制的具体过程

3. 重绘功能

将窗口最小化后再恢复正常化显示的过程，其实就是重绘的过程。如果考虑重绘功能，恢复正常化显示就会出现正常的窗口；否则就会出现空白窗口，具体过程如图 13.4 所示。

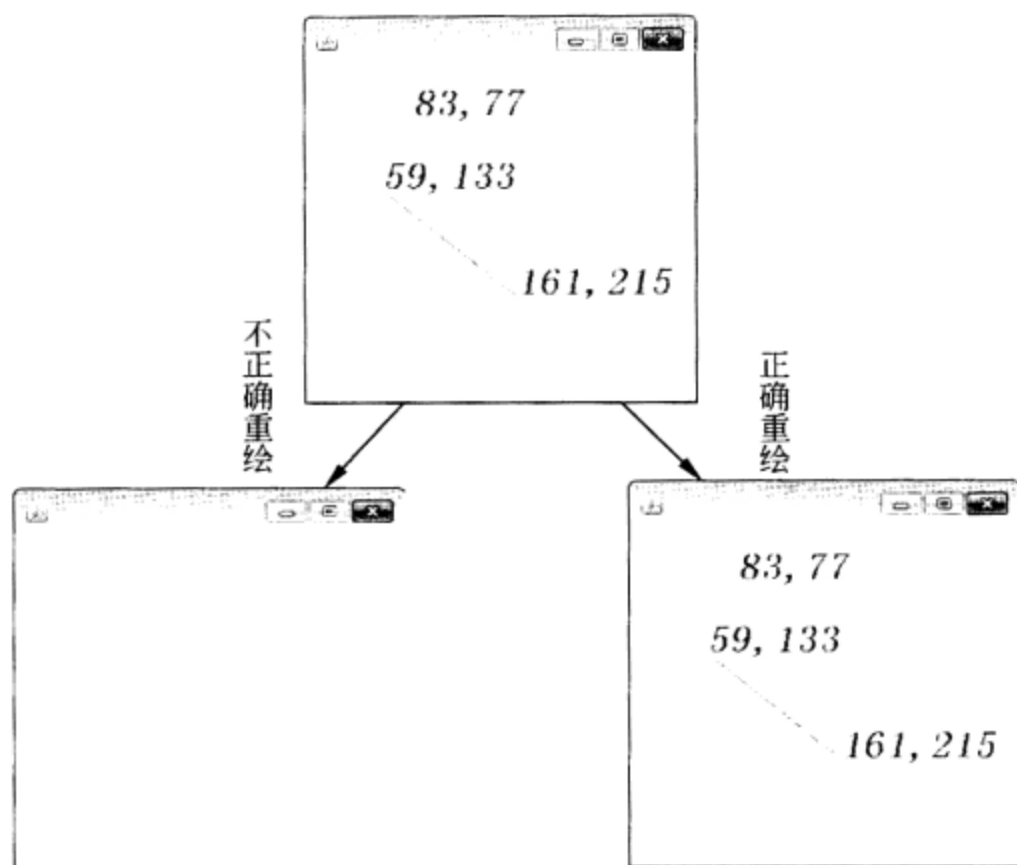


图 13.4 重绘的过程

13.2 鼠标绘直线的实现过程

本节通过绘图机制和事件机制的相关知识来实现鼠标绘直线项目，具体程序架构如图 13.5 所示，它包含一个直线的类 `MyLine.java` 和一个窗口的类 `DrawLine.java`。

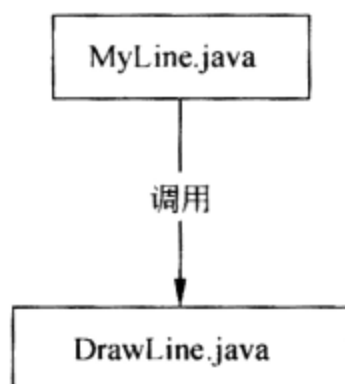


图 13.5 程序关系图

13.2.1 直线的类

`MyLine.java` 类用来模拟直线对象，该类除了拥有直线端点的坐标成员变量外，而且还拥有一个画自己的 `drawMe()` 方法，具体内容如代码 13.1 所示，该类的 UML 如图 13.6 所示。

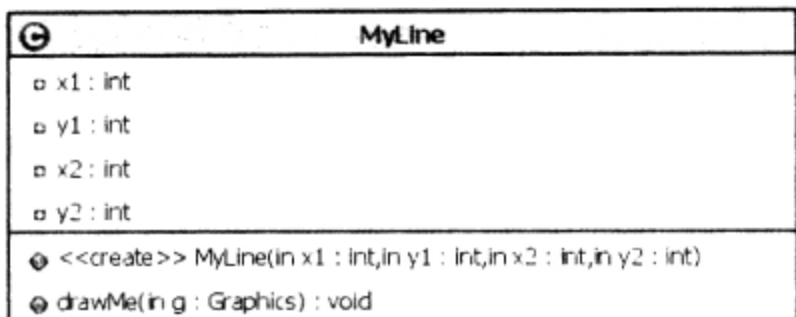


图 13.6 直线类图

代码 13.1 直线类: MyLine.java

```
class MyLine {
    //创建成员变量
    private int x1;                //直线起点的 x 坐标
    private int y1;                //直线起点的 y 坐标
    private int x2;                //直线终点的 x 坐标
    private int y2;                //直线终点的 y 坐标
    public MyLine(int x1, int y1, int x2, int y2) { //构造函数
        this.x1 = x1;
        this.y1 = y1;
        this.x2 = x2;
        this.y2 = y2;
    }
    public void drawMe(Graphics g) {
        g.setColor(Color.red);    //设置绘图颜色为红色
        //设置文本的字体
        g.setFont(new Font("隶书", Font.ITALIC | Font.BOLD, 30));
        g.drawString(new String(x1 + "," + y1), x1, y1);
        //打印鼠标按下时的坐标文本
        g.drawString(new String(x2 + "," + y2), x2, y2);
        //打印鼠标释放时的坐标文本
        g.drawLine(x1, y1, x2, y2); //绘制直线
    }
}
```

【代码解析】

在上述代码中，该类拥有 4 个字段 x1、y1、x2 和 y2，同时在 drawMe()方法中通过 drawString()和 drawLine()方法绘制出相应的端点坐标和直线。

13.2.2 实现窗口类——通过 paint()方法

DrawLine 类主要通过 paint()方法实现具有鼠标绘制直线功能的窗口，该类的具体内容如代码 13.2 所示，该类的 UML 如图 13.7 所示。

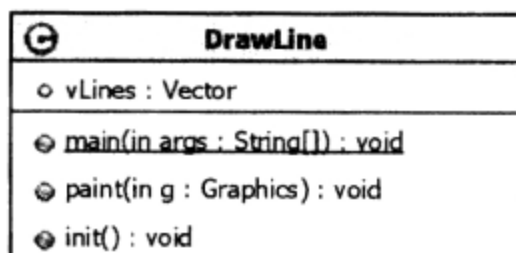


图 13.7 绘制类图

代码 13.2 窗口类: DrawLine.java

```

public class DrawLine extends Frame {
    Vector vLines = new Vector(); //集合变量
    public static void main(String[] args) {
        DrawLine f = new DrawLine(); //创建 DrawLine 对象
        f.init(); //初始化
    }
    public void paint(Graphics g) { //重写 paint() 方法
        g.setColor(Color.red); //设置颜色
        //遍历集合元素
        Enumeration e = vLines.elements();
        while (e.hasMoreElements()) {
            MyLine ln = (MyLine) e.nextElement(); //获取直线元素
            ln.drawMe(g); //鼠标绘直线
        }
    }
    public void init() { //初始化方法
        this.addWindowListener(new WindowAdapter() { //窗口关闭的方法
            public void windowClosing(WindowEvent e) {
                ((Window) e.getSource()).dispose();
                System.exit(0);
            }
        });
        addMouseListener(new MouseAdapter() { //鼠标的监听器
            //直线起点的坐标
            int orgX;
            int orgY;
            public void mousePressed(MouseEvent e) { //鼠标的按下方法
                orgX = e.getX(); //为直线起点赋值
                orgY = e.getY();
            }
            public void mouseReleased(MouseEvent e) { //鼠标释放的方法
                //获取事件源对象
                Graphics g = e.getComponent().getGraphics();
                //设置文本的字体
                g.setFont(new Font("隶书", Font.ITALIC | Font.BOLD, 30));
                g.setColor(Color.red); //设置颜色
                //绘制鼠标按下时的坐标文本
                g.drawString(new String(orgX + "," + orgY), orgX, orgY);
                //绘制鼠标释放时的坐标文本
                g.drawString(new String(e.getX() + "," + e.getY()),
                    e.getX(), e.getY());
                g.drawLine(orgX, orgY, e.getX(), e.getY()); //绘制直线
                //添加到集合中
                vLines.add(new MyLine(orgX, orgY, e.getX(), e.getY()));
            }
        });
        this.setSize(300, 300); //设置窗口大小
        setVisible(true); //显示窗口
    }
}

```

【代码解析】

- 在上述代码中存在一个实现初始化功能的 init()方法, 在该方法中主要实现了两个事件监听器窗口监听器和鼠标监听器。在鼠标监听器中, 鼠标按下的方法中实现

了对直线端点的赋值；鼠标释放的方法中，首先获取了鼠标释放时的坐标，然后不仅绘制出直线而且还输出端点的坐标，最后还将直线添加到集合中。在窗口监听器中，主要实现了窗口的关闭功能。

- 在上述代码中还存在一个实现重新绘制功能的 `paint()` 方法，在该方法中首先遍历集合，然后通过调用 `MyLine.drawMe()` 方法输出集合中的所有直线。

🔔注意：由于上述代码需要重写 `paint()` 方法来实现重绘功能，所以继承了 `Frame` 类。

13.2.3 实现窗口类——通过双缓冲技术

`DrawLine` 类主要通过双缓冲技术来实现具有鼠标绘制直线功能的窗口，该类的具体内容如代码 13.3 所示，该类的 UML 如图 13.8 所示。

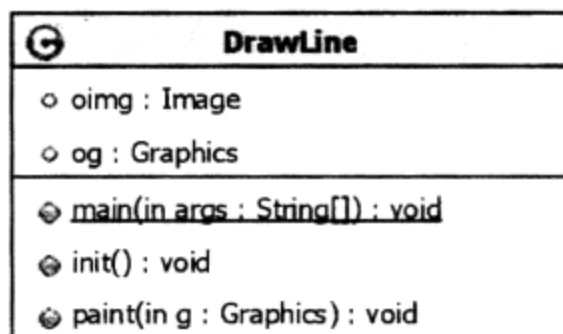


图 13.8 绘制的类图

代码 13.3 窗口类: `DrawLine.java`

```

public class DrawLine extends Frame {
    //成员变量
    Image oimg = null;
    Graphics og = null;
    public static void main(String[] args) {
        new DrawLine().init(); //调用初始化方法
    }
    public void init() { //实现初始化方法
        setSize(300, 300); //设置窗口大小
        setVisible(true); //显示窗口
        Dimension d = getSize();
        oimg = createImage(d.width, d.height);
        og = oimg.getGraphics();
        addMouseListener(new MouseAdapter() {
            //创建直线端点的成员变量
            int orgX;
            int orgY;
            public void mousePressed(MouseEvent e) {
                //为直线端点的成员变量赋值
                orgX = e.getX();
                orgY = e.getY();
            }
            public void mouseReleased(MouseEvent e) {
                //g 对象的设置
                Graphics g = getGraphics(); //获取 Graphics 对象
                g.setColor(Color.red); //设置绘图颜色为红色
            }
        });
    }
}
  
```

```

        //设置文本的字体
        g.setFont(new Font("隶书", Font.ITALIC | Font.BOLD, 30));
        //输出鼠标按下时的坐标文本
        g.drawString(new String(orgX + "," + orgY), orgX, orgY);
        //输出鼠标释放时的坐标文本
        g.drawString(new String(e.getX() + "," + e.getY()),
            e.getX(), e.getY());
        //创建 MyLine 对象
        MyLine line = new MyLine(orgX, orgY, e.getX(), e.getY());
        line.drawMe(g); //调用 drawMe() 方法
        //og 对象的设置
        og.setColor(Color.red); //设置绘图颜色为红色
        //设置文本的字体
        og.setFont(new Font("隶书", Font.ITALIC | Font.BOLD, 30));
        //输出鼠标按下时的坐标文本
        og.drawString(new String(orgX + "," + orgY), orgX, orgY);
        //输出鼠标释放时的坐标文本
        og.drawString(new String(e.getX() + "," + e.getY()),
            e.getX(), e.getY());
        line.drawMe(og); //调用 drawMe() 方法
    }
    });
}

public void paint(Graphics g) { //实现 paint() 方法
    if (oimg != null)
        g.drawImage(oimg, 0, 0, this);
}
}

```

【代码解析】

- ❑ 在上述代码中存在一个实现初始化功能的 `init()` 方法，在该方法中首先通过组件的 `createImage()` 方法在内存中创建一个 `Image` 对象 `oimg`，然后通过该对象的 `getGraphics()` 方法获取 `og` 对象。
- ❑ 在 `init()` 方法中还存在一个鼠标监听器，鼠标按下的方法中，实现了对直线端点的赋值；在鼠标释放的方法中，不仅把直线绘制在组件上，而且还绘制在内存中的 `oimg` 对象上，即 `oimg` 对象上的图形是组件表面内容的复制。
- ❑ 最后在重新绘制 `paint()` 方法，把内存中的 `oimg` 对象上的图形重绘在组件上。

13.3 知识点扩展——画图的基础知识

查看 API 帮助文档，如果想要在 GUI 组件上绘制图形、打印文字、显示图像等操作，需要分成两个步骤：首先组件需要通过 `getGraphics()` 方法获取类 `Graphics` 对象，然后 `Graphics` 对象通过相应的方法实现各种所需的功能。

13.3.1 画图的基础知识

无论在组件上画一条线、一个矩形或者一张图片等，都需要 `Graphics` 类。该类不仅包含了组件的屏幕外观信息，而且还提供了在组件上显示表面绘画图形、打印文字和显示图

像等操作方法。

如果想学好 Java 语言中的绘图功能，需要从基本的类和方法开始，下面详细介绍。

1. Graphics类

由于 Graphics 类是抽象类，所以不能直接产生对象来使用，同时也没有 static()方法直接获取。那么在 Java 应用程序中如何获取 Graphics 对象呢？通常有以下两种方式。

- ❑ 通过系统获取：所有 Component 对象都会有一个名为 getGraphics()的方法，该方法会获取作用于该 Component 对象上的 Graphics 对象。所谓作用于 Component 对象，是指该对象的任何绘画的动作，都只会显示在这个对象上。
- ❑ 通过 Image 类获取：尽管 Image 类同样拥有一个名为 getGraphics()的方法，但是该类并不是 Component 类的子类。

2. paint()方法和repaint()方法

在 Java 应用程序中，一般不自己产生 Graphics 对象来使用，而是去改写 paint()方法，在该方法的方法体中调用传递进来的 Graphics 对象。所谓 paint()方法，就是负责处理 Component 对象画面，即当 Component 对象觉得需要更新自己的画面时，就会调用该方法。查看 API 帮助文档，可以发现 paint()方法为 Component 类的一个方法，其拥有一个传入参数，该参数就是 Graphics 对象。

⚠注意：如果想让所画的东西一直显示在组件上，就需要继承该组件类，同时改写 paint()方法。

根据字面意思可以知道，repaint 为重新 paint 的意思，方法 repaint()与 paint()的关系如图 13.9 所示。即当 Component 对象觉得需要更新自己的画面时，就会运行 paint()方法里的程序代码。在实际的运行过程中，AWT 线程首先会调用 repaint()方法，然后该方法会调用 update()方法，最后 update()方法才会调用 paint()方法，执行该方法体中的代码内容。

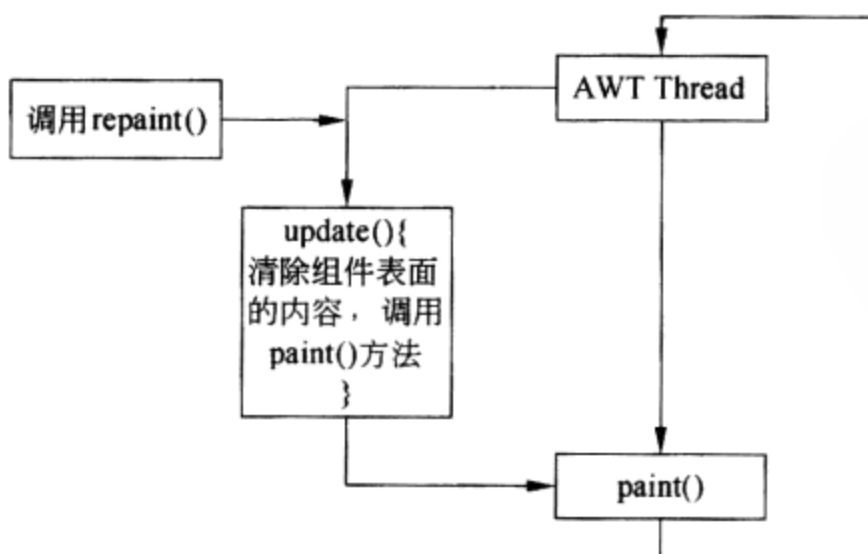


图 13.9 方法的关系

3. Canvas类

在 Java API 中存在一个名为“画布”的类 Canvas，该类的用途非常简单，就是供程序

员在其上作画。虽然 `Frame` 类和 `Panel` 类对象上也可以作画，但是 Java API 中不建议使用。在具体的实现中，首先会编写一个类，该类继承了 `Canvas` 类，接着改写 `paint()` 方法，最后再把该类对象加到 `Frame` 或 `Panel` 对象中。

4. 坐标系

在绘图的过程中，经常会需要坐标参数，所以必须了解计算机的坐标系。与数学上的坐标系类似，只是计算机坐标系的原点 (0,0) 是在 `Component` 对象的左上角。

⚠注意：对于 `Frame` 对象来说，不同的系统对于坐标原点的表示不一样。例如，有的系统以整个 `Frame` 对象的左上角作为原点，有的系统会忽略标题栏左上角作为原点。

13.3.2 各种类型对象的绘制

1. 直线

在 `Graphics` 类中存在一个 `drawLine()` 方法，主要用来实现鼠标绘直线的功能，该方法的基本格式如下：

```
abstract void drawLine(int x1, int y1, int x2, int y2)
```

参数 (x1,y1) 为直线的起始坐标点，参数 (x2,y2) 为直线的终结点坐标。下面通过一个具体的类 `DrawLine`，来讲解如何来鼠标绘直线，具体内容如代码 13.4 所示。

代码 13.4 鼠标绘直线：DrawLine.java

```
public class DrawLine extends Canvas {           //继承 Canvas 组件
    public static void main(String argv[]) {
        DrawLine d = new DrawLine();           //创建一个对象 d
        Frame f = new Frame("鼠标绘直线");      //创建一个窗口对象
        f.add(d, BorderLayout.CENTER);          //添加对象 d 到窗口对象 f 中
        f.pack();
        f.setVisible(true);                     //显示图形
    }
    public DrawLine() {                          //构造函数
        setSize(50, 50);                        //设置 d 对象的大小
    }
    public void paint(Graphics g) {              //改写 paint() 方法
        g.drawLine(10, 15, 30, 40);            //鼠标绘直线
    }
}
```

运行 `DrawLine.java` 类，会出现如图 13.10 所示的用户图形界面。

【代码解析】

- ❑ `DrawLine` 类首先继承了 `Canvas` 类，然后改写了 `paint()` 方法，最后把该对象添加到窗口对象 `f` 中。
- ❑ `DrawLine` 类之所以继承 `Canvas` 类而不直接继承 `Frame` 类，主要是因为在 Windows 系统里，原点坐标会在整个 `Frame` 对象的左上角，而不是忽略标题栏左上角。

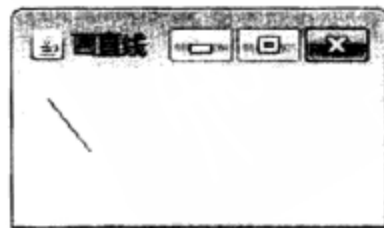


图 13.10 用户图形界面

2. 字符串

在 `Graphics` 类中存在一个 `drawLine()` 方法, 主要用来实现画字符的功能。该方法的基本格式如下:

```
abstract void drawString(String str, int x, int y)
```

参数 `str` 为所要输出的字符串, 而参数 `(x,y)` 为字符串最左边字符的基准线坐标。下面通过一个具体的类 `DrawString.java`, 来讲解如何画字符, 具体内容如代码 13.5 所示。

代码 13.5 画字符: `DrawString.java`

```
public class DrawString extends Canvas {
    public static void main(String argv[]) {           //主函数
        DrawString d = new DrawString();             //创建对象 d
        Frame f = new Frame("画字符");               //创建对象 f
        f.add(d, BorderLayout.CENTER);
        f.pack();                                     //自适应大小
        f.setVisible(true);                           //设置可见
    }
    public DrawString() {                             //构造函数
        setSize(460, 200);                           //设置大小
    }
    public void paint(Graphics g) {                   //重写 paint() 方法
        g.drawString("所要画的字符", 30, 50);        //输出相应文字
        g.drawLine(20, 50, 180, 50);                 //画直线
    }
}
```

运行 `DrawString.java` 类, 出现如图 13.11 所示的用户图形界面。

【代码解析】

- ❑ `DrawString` 类首先继承了 `Canvas` 类, 然后改写了 `paint()` 方法, 最后把该对象添加到窗口对象 `f` 中。
- ❑ 在具体改写 `paint()` 方法的方法体中, 首先调用 `drawString()` 画出相应的字符串, 接着通过 `drawLine()` 方法画出字符串的基准线。

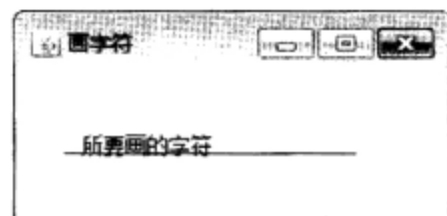


图 13.11 运行结果

3. 矩形

在 `Graphics` 类中存在一些例如 `drawXxxRec()` 的方法, 主要用来实现各种类型矩形的功能, 如果想画实心的矩形, 则需要把 `drawXxxRec()` 改成 `fillXxxRec()` 方法。这些方法的基本格式如下:

(1) `abstract void drawRect(int x, int y, int width, int height)`

参数 `(x,y)` 为矩形左上角的坐标, 参数 `width` 和 `height` 分别为矩形的宽度和高度。

(2) `abstract void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)`

参数 `arcWidth` 为矩形圆角的宽度, 参数 `arcHeight` 为矩形圆角的高度。

(3) void draw3DRect(int x, int y, int width, int height, boolean raised)

参数 raised 用来表示是否为凸起或凹陷的样子。

下面通过一个具体的类 DrawRectangle.java, 来讲解如何画矩形, 具体内容如代码 13.6 所示。

代码 13.6 画矩形: DrawRectangle.java

```
public class DrawRectangle extends Canvas {
    public static void main(String argv[]) {           //主方法
        DrawRectangle d = new DrawRectangle();        //创建对象 d
        Frame f = new Frame("画矩形");               //创建窗体对象 f
        f.add(d, BorderLayout.CENTER);                //添加对象 d 到对象 f 里
        f.pack();                                     //自适应大小
        f.setVisible(true);                           //设置窗体可见
    }
    public DrawRectangle() {                           //构造函数
        setBackground(Color.blue);                   //设置大小
        setSize(460, 170);                           //设置窗口大小
    }
    public void paint(Graphics g) {                   //重绘方法
        //创建矩形对象 drawRect
        g.setColor(Color.cyan);                       //设置颜色
        g.drawString("drawRect", 15, 20);             //输出相应字符串
        g.drawRect(10, 30, 100, 50);                  //绘制矩形
        //创建矩形对象 drawRoundRect
        g.drawString("drawRoundRect", 125, 20);        //输出相应字符串
        g.drawRoundRect(120, 30, 100, 50, 10, 10);    //绘制矩形
        //创建矩形对象 draw3DRect(true)
        g.drawString("draw3DRect(true)", 235, 20);    //输出相应字符串
        g.draw3DRect(230, 30, 100, 50, true);        //绘制矩形
        //创建矩形对象 draw3DRect(false)
        g.drawString("draw3DRect(false)", 355, 20);   //输出相应字符串
        g.draw3DRect(350, 30, 100, 50, false);       //绘制矩形
        //创建矩形对象 fillRect
        g.drawString("fillRect", 15, 100);            //输出相应字符串
        g.fillRect(10, 110, 100, 50);                 //绘制矩形
        //创建矩形对象 fillRoundRect
        g.drawString("fillRoundRect", 125, 100);      //输出相应字符串
        g.fillRoundRect(120, 110, 100, 50, 10, 10);  //绘制矩形
        //创建矩形对象 fillRoundRect
        //创建矩形对象 fill3DRect(true)
        g.drawString("fill3DRect(true)", 235, 100);   //输出相应字符串
        g.fill3DRect(230, 110, 100, 50, true);       //绘制矩形
        //创建矩形对象 fill3DRect(false)
        g.drawString("fill3DRect(false)", 355, 100);  //输出相应字符串
        g.fill3DRect(350, 110, 100, 50, false);      //绘制矩形
    }
}
```

运行 DrawRectangle.java 类, 会出现如图 13.12 所示的用户图形界面。

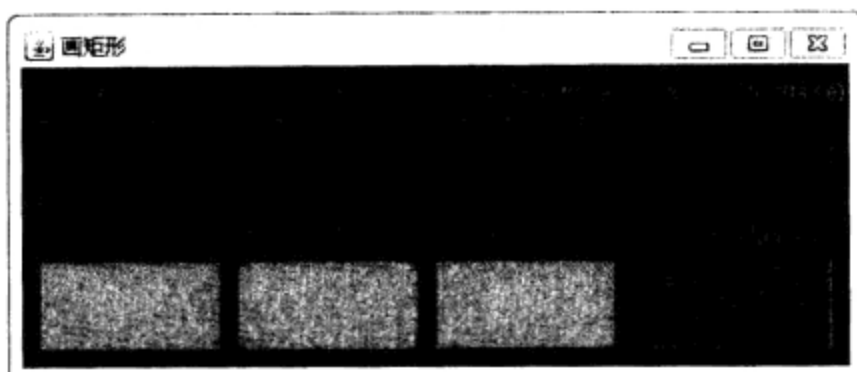


图 13.12 运行结果

【代码解析】

- ❑ DrawString 类首先继承了 Canvas 类，然后改写了 paint() 方法，最后把该对象添加到窗口对象 f 中。
- ❑ 在具体改写 paint() 方法的方法体中，首先调用 drawString() 画出相应的字符串，接着通过 drawLine() 方法画出字符串的基准线。

4. 椭圆

在 Graphics 类中存在一个名为 drawOval() 的方法，该方法主要用来实现画一个空心椭圆，如果想画实心椭圆，则需要调用 fillOval() 方法。如果想画一个圆弧，则需要调用 drawArc() 方法，同样，如果想画实心圆弧，则需要调用 fillArc() 方法。

这些方法的基本格式如下：

(1) abstract void drawOval(int x, int y, int width, int height)


参数 x 和 y 为椭圆边框左上角的坐标，而参数 width 和 height 为椭圆边框的宽度和高度。

(2) abstract void fillOval(int x, int y, int width, int height)

该方法参数的含义与 drawOval() 方法参数的含义一样。

(3) abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)

参数 x 和 y 为圆弧边框左上角的坐标，而参数 width 和 height 为圆弧边框的宽度和高度，参数 startAngle 为圆弧的起点，参数 arcAngle 为圆弧的角度。

 **注意：**圆弧的起点与数学上常用表达方式一样，以 360° 的方式表示。即 0° 在钟表三点钟的位置，逆时针方向旋转，90° 在十二点钟的位置。

(4) fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)

该方法参数的含义与 drawArc() 方法参数的含义一样。

下面通过一个具体的类 DrawOval.java，来讲解如何画椭圆，具体内容如代码 13.7 所示。

代码 13.7 画椭圆：DrawOval.java

```
public class DrawOval extends Canvas {
    public static void main(String argv[]) {           //主方法
        DrawOval d = new DrawOval();                 //创建类 DrawOval 的对象
    }
}
```

```

        Frame f = new Frame("画椭圆");           //创建窗口对象
        f.add(d, BorderLayout.CENTER);           //添加对象 d 到窗口对象上
        f.pack();                                 //显示窗口对象
        f.setVisible(true);
    }
    public DrawOval() {
        setSize(250, 150);                       //设置窗口对象的大小
    }
    public void paint(Graphics g) {               //重写 paint() 方法
        g.drawString("Circle", 20, 20);          //输出相应信息
        g.drawOval(10, 30, 50, 50);              //绘制椭圆
        g.drawString("Oval-shape", 80, 20);       //输出相应信息
        g.drawOval(70, 30, 100, 50);             //绘制椭圆
        g.drawString("drawArc", 200, 20);         //输出相应信息
        g.drawArc(190, 30, 50, 50, 0, 90);       //绘制椭圆
        g.fillOval(10, 90, 50, 50);              //绘制椭圆
        g.fillOval(70, 90, 100, 50);             //绘制椭圆
        g.fillArc(190, 90, 50, 50, 0, 90);       //绘制椭圆
    }
}

```

运行 DrawOval.java 类, 出现如图 13.13 所示的用户图形界面。

【代码解析】

- ❑ DrawString 类首先继承了 Canvas 类, 然后改写了 paint() 方法, 最后把该对象添加到窗口对象 f 中。
- ❑ 在具体改写 paint() 方法的方法体中, 首先调用 drawString() 方法画出相应的字符串, 接着通过 drawLine() 方法画出字符串的基准线。

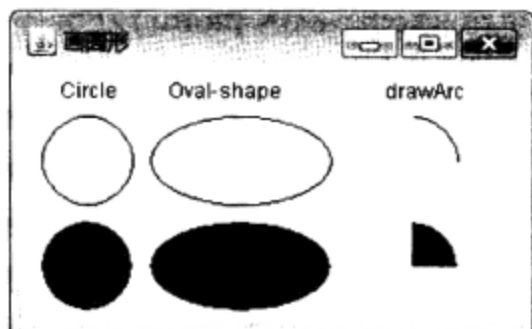


图 13.13 运行结果

5. 多边形

在 Graphics 类中存在两个多边形的类的方法 drawPolygon() 和 drawPolyline(), 其中前者画出的多边形会把起点和终点连接起来, 而后者却不会。因此方法 drawPolygon() 用来画多边形, 而方法 drawPolyline() 用来画曲折线。这些方法的基本格式如下:

(1) abstract void drawPolygon(int[] xPoints, int[] yPoints, int nPoints)

参数 xpoints 为多边形所有顶点的 x 坐标的值集, 参数 yPoints 为多边形所有顶点的 y 坐标的值集, 参数 nPoints 为多边形顶点的个数。

(2) drawPolyline(int[] xPoints, int[] yPoints, int nPoints)

曲折线方法中参数的含义与方法 drawPolyline() 中参数的含义一样。

下面通过一个具体的类 DrawPolygon.java, 来讲解如何画多边形, 具体内容如代码 13.8 所示。

代码 13.8 画多边形: DrawPolygon.java

```
public class DrawPolygon extends Canvas {
```

```

int x[] = { 40, 70, 100, 100, 70, 40, 10, 10 }; //所有顶点的 x 坐标值
int y[] = { 10, 10, 40, 70, 100, 100, 70, 40 }; //所有顶点的 y 坐标值
public static void main(String argv[]) { //主方法
    Polygon d = new Polygon(); //创建 Polygon 类的对象
    Frame f = new Frame("画多边形"); //创建窗口对象
    f.add(d, BorderLayout.CENTER); //添加多边形到窗口对象上
    f.pack(); //自适应窗口大小
    f.setVisible(true); //设置窗体的可见形状
}
public Polygon() { //构造函数
    setSize(110, 110); //设置大小
}
public void paint(Graphics g) { //重写 paint() 方法
    g.drawPolygon(x, y, 8); //绘制多边形
}
}

```

运行 DrawPolygon.java 类, 会出现如图 13.14 所示的用户图形界面。

【代码解析】

在上述代码中, 首先为画多边形准备所有顶点的 x 坐标值和 y 坐标值, 然后在 paint() 方法中通过代码 g.drawPolygon(x, y, 8) 实现画八边形。

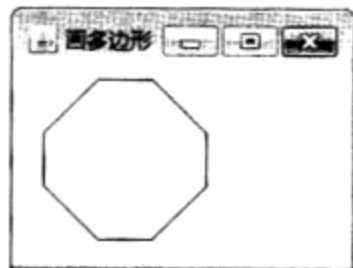


图 13.14 运行结果

下面通过一个具体的类 DrawPolyline.java, 来讲解如何来画曲折线, 具体内容如代码 13.9 所示。

代码 13.9 画曲折线: DrawPolyline.java

```

public class DrawPolygon extends Canvas {
    int x[] = { 40, 70, 100, 100, 70, 40, 10, 10 }; //所有顶点的 x 坐标值
    int y[] = { 10, 10, 40, 70, 100, 100, 70, 40 }; //所有顶点的 y 坐标值
    public static void main(String argv[]) { //主方法
        Polygon d = new Polygon(); //创建 Polygon 类的对象
        Frame f = new Frame("画曲折线"); //创建窗口对象
        f.add(d, BorderLayout.CENTER); //添加多边形到窗口对象上
        f.pack(); //自适应窗口大小
        f.setVisible(true);
    }
    public Polygon() { //构造函数
        setSize(110, 110); //设置大小
    }
    public void paint(Graphics g) { //重写 paint() 方法
        g.drawPolyline(x, y, 8); //绘制多边形
    }
}

```

运行 DrawPolyline.java 类, 会出现如图 13.15 所示用户图形界面。

【代码解析】

上述代码首先为画曲折线准备所有顶点的 x 坐标值和 y 坐标值, 然后在 paint() 方法中通过代码 g.drawPolyline(x, y, 8) 实现画曲折线。

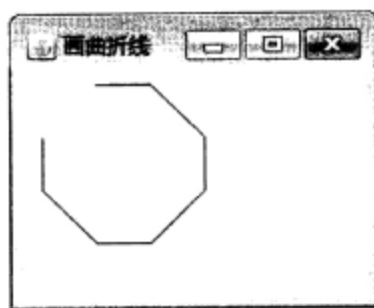


图 13.15 运行结果

13.4 小 结

本章主要通过 Java 语言中的绘图机制实现鼠标绘直线项目，为了更好地让读者理解该项目，分别通过两种方式来实现：利用 `paint()` 方法实现重绘及利用双缓存技术来实现重绘。为了让读者能够更好地掌握 Java 语言的绘制机制，在本章的最后还详细介绍了 Java 语言经常遇到的各种图形的绘制。

第 14 章 指针时钟项目（Swing 组件+时间算法）

在 Java 语言中利用事件机制可以实现许多意想不到的功能，所以对于程序员来说必须掌握事件机制。对于一个应用程序，如果想实现与用户良好的互动功能，对事件的处理也是必不可少的。本章将通过模拟指针时钟的功能，介绍 Java 语言中的事件机制，还将详细介绍事件机制的基本知识。

本章的学习目标如下：

- ❑ 掌握指针时钟项目；
- ❑ 理解为什么要使线程具有同步性；
- ❑ 掌握实现线程同步的两种方式。

14.1 指针时钟原理

“指针时钟”项目用来模拟现实生活中指针时钟的功能，在具体运行该项目时，其界面中不仅会正确显示数字时间，而且还会正确显示指针时间，同时还会与时间变化实现同步。

14.1.1 项目结构框架分析

对于指针时钟项目，根据面向对象的思想，需要创建一个对象即指针时钟。指针时钟项目目录如图 14.1 所示，该项目存在两个类，Clock 类用来实现指针时钟的界面；DrawClock 类用来实现绘制指针时钟。

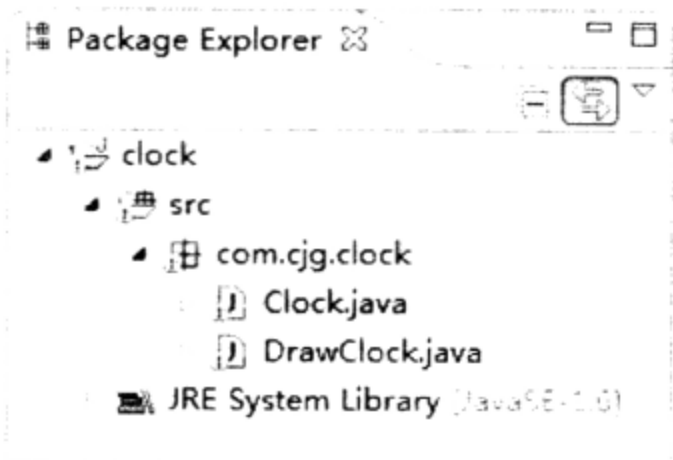


图 14.1 项目目录

14.1.2 项目功能业务分析

本节将以直观的方式向读者介绍整个项目要实现的功能。这些功能包括指针时钟的初始化，按住鼠标功能和时间同步功能。

1. 指针时钟的初始化

当运行测试指针时钟的类 Clock 后，会出现如图 14.2 所示的初始界面。在该界面中不仅显示出指针时间（2010-5-3 6:51:05），而且还会在表盘中指出正确的时间。

2. 时间同步功能

当出现指针时钟的初始界面后，随着时间的推移，不仅表盘指针会随着时间的变化实现正确的移动，而且还会正确地显示出时间。具体过程如图 14.3 所示。



图 14.2 指针时钟的初始界面



图 14.3 时间移动过程



14.2 指针时钟的实现过程

本节通过事件机制和多线程的相关知识来实现指针时钟项目，具体程序架构如图 14.4 所示，它包含一个实现时钟界面的类 Clock.java 和一个进行指针绘制的类 DrawClock.java。

14.2.1 指针时钟的界面

Clock.java 类用来实现指针时钟界面，由于该类利用的线程机制，所以其不仅继承了 JPanel 组件，而且还实现了 Runnable 接口。该类具体内容如代码 14.1 所示，该类的 UML 如图 14.5 所示。

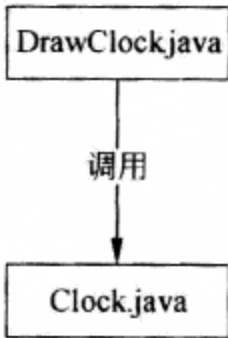


图 14.4 程序关系图

代码 14.1 指针时钟类：Clock.java

```
public class Clock extends JPanel implements Runnable
//创建成员变量
```

```

private JLabel jl; //创建标签对象
private DateFormat df; //创建时间格式对象
public Clock() { //构造函数
    jl = new JLabel(); //为面板对象赋值
    jl.setHorizontalAlignment(JLabel.CENTER); //设置面板对象的对齐方式
    df = DateFormat.getDateInstance(); //为时间格式对象赋值
    new Thread(this).start(); //启动一个新线程
    this.setLayout(new BorderLayout()); //设置布局格式
    this.add(jl, BorderLayout.SOUTH); //添加标签对象到窗口中
}
public void run() { //实现 run() 方法
    while (true) {
        try {
            Thread.sleep(1000);
        } catch (InterruptedException ie) { //休眠 1 秒
            ie.printStackTrace();
        }
        jl.setText(df.format(new Date())); //输出新时间
        repaint(); //调用 repaint() 方法
    }
}
public void paintComponent(Graphics g) { //重写 paintComponent() 方法
    super.paintComponent(g);
    DrawClock ts = new DrawClock(); //创建 DrawClock 对象
    ts.drawSelfClock(this, g); //调用该对象的 drawSelfClock
}
public static void main(String args[]) {
    JFrame jf = new JFrame("指针时钟"); //创建窗口对象
    //添加指针时钟对象到窗口中
    jf.getContentPane().add(new Clock(), BorderLayout.CENTER);
    jf.setBounds(300, 300, 300, 300);
    jf.setVisible(true); //显示窗口
    jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //设置窗口的关闭功能
}
}

```

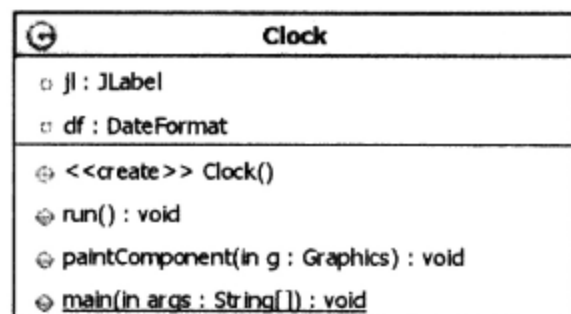


图 14.5 指针时钟类图

【代码解析】

上述代码实现了指针时钟的自定义窗口类，该用户界面涉及的具体容器、对象和布局如图 14.6 所示。该项目的原理与“秒表记时器”的原理基本相同，即在一个 JPanel 组件上通过线程技术每隔一段时间进行一次重绘。在具体执行时，每隔 1 秒不仅会重新设置对象 jl 的显示信息，而且还会通过 repaint() 方法调用实现指针时钟重绘的 paintComponent() 方法。

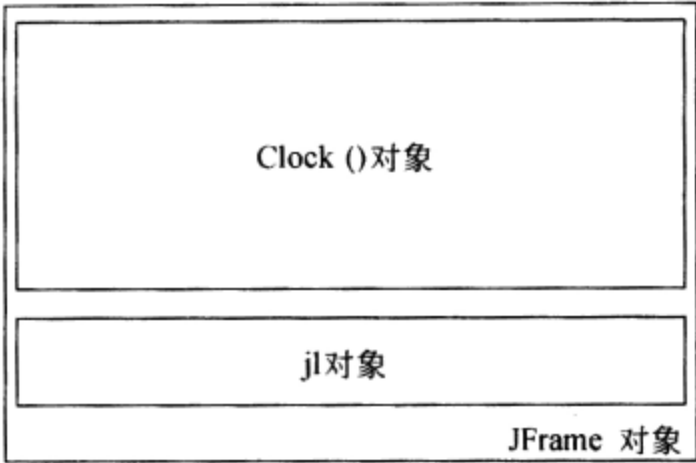


图 14.6 布局

⚠注意: paintComponent()方法的效率比 paint()方法的效率高一些, 因为 paint()方法会把组件上的内容全部重画, 而前者只是重画组件容器里的东西。

14.2.2 绘制指针时钟的类

DrawClock.java 类用来实现在 Clock 类对象上绘制指针时钟图形, 该类的具体内容如代码 14.2 所示, 该类的 UML 如图 14.7 所示。

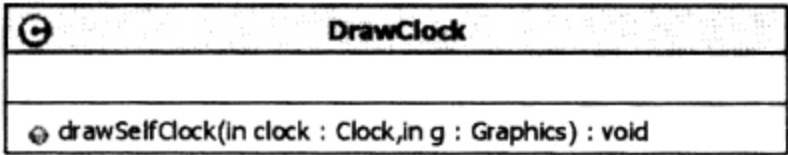


图 14.7 DrawClock 类的类图

代码 14.2 测试指针时钟类: DrawClock.java

```
public class DrawClock {
    public void drawSelfClock(Clock clock, Graphics g) {
        Calendar cal = Calendar.getInstance(); //获取当前日期对象 cal
        //得到当前的时间信息
        int hour = cal.get(Calendar.HOUR);
        int minute = cal.get(Calendar.MINUTE);
        int second = cal.get(Calendar.SECOND);
        //获取表盘半径 radius 的值
        //得到当前窗口的长框信息
        int width = clock.getWidth();
        int height = clock.getHeight();
        //表盘半径取两个之中小的那个
        int small = width < height ? width : height;
        int diameter = (int) (small * 0.8);
        int radius = diameter / 2; //设置半径的值
        Point center = new Point(width / 2, height / 2); //获取圆点的值
        //设置时针、分针、秒针的长度
        int secondLength = (int) (radius * 0.8);
        int minuteLength = (int) (secondLength * 0.8);
        int hourLength = (int) (minuteLength * 0.8);
        //设置时针、分针、秒针的另一端坐标
```

```

//秒针的 x 和 y 坐标
int secondX = center.x
    + (int) (secondLength * Math.sin(second * 2 * Math.PI / 60.0));
int secondY = center.y
    - (int) (secondLength * Math.cos(second * 2 * Math.PI / 60.0));
//分针的 x 和 y 坐标
int minuteX = center.x
    + (int) (minuteLength * Math.sin(minute * 2 * Math.PI / 60.0));
int minuteY = center.y
    - (int) (minuteLength * Math.cos(minute * 2 * Math.PI / 60.0));
//时针的 x 和 y 坐标
int hourX = center.x
    + (int) (hourLength * Math.sin((minute / 60.0 + hour) * Math.PI
    / 6.0));
int hourY = center.y
    - (int) (hourLength * Math.cos((minute / 60.0 + hour) * Math.PI
    / 6.0));
//绘制表盘和刻度
Graphics2D g2d = (Graphics2D) g; //获取绘制对象 g2d
//绘制表盘
g.drawOval(center.x - radius, center.y - radius, diameter, diameter);
//绘制刻度
for (int i = 0; i < 60; i++) {
    //获取刻度线条的终点坐标
    int x2 = center.x
        + (int) (radius * Math.sin(i * 2 * Math.PI / 60.0));
    int y2 = center.y
        - (int) (radius * Math.cos(i * 2 * Math.PI / 60.0));
    if (i % 5 == 0) { //当为整时刻度时
        //获取刻度线条的起点坐标
        int x1 = center.x
            + (int) ((secondLength + 1) * Math.sin(i * 2 * Math.PI
            / 60.0));
        int y1 = center.y
            - (int) ((secondLength + 1) * Math.cos(i * 2 * Math.PI
            / 60.0));
        g2d.setStroke(new BasicStroke(2.5f)); //设置 g2d 对象
        g2d.drawLine(x1, y1, x2, y2); //绘制刻度
        //绘制刻度的值
        int sj = i / 5;
        if (sj == 0) {
            sj = 12;
        }
        g2d.drawString(String.valueOf(sj), x2, y2); //绘制刻度的值
    } else {
        //获取刻度线条的起点坐标
        int x1 = center.x
            + (int) ((secondLength + 10) * Math.sin(i * 2 * Math.PI
            / 60.0));
        int y1 = center.y
            - (int) ((secondLength + 10) * Math.cos(i * 2 * Math.PI
            / 60.0));
        g2d.setStroke(new BasicStroke(0.8f));
        g2d.drawLine(x1, y1, x2, y2); //绘制刻度
    }
}
//绘制时针、分针、秒针

```

```

        g2d.setColor(Color.RED); //绘制时针
        g2d.setStroke(new BasicStroke(3.0f));
        g2d.drawLine(center.x, center.y, hourX, hourY);
        g2d.setColor(Color.BLUE); //绘制分针
        g2d.setStroke(new BasicStroke(1.5f));
        g2d.drawLine(center.x, center.y, minuteX, minuteY);
        g2d.setColor(Color.MAGENTA); //绘制秒针
        g2d.setStroke(new BasicStroke(1.0f));
        g2d.drawLine(center.x, center.y, secondX, secondY);
    }
}

```

对于 DrawClock 类, 为了实现绘制指针时钟, 有以下几个问题需要解决:

- ☐ 如何绘制指针时钟的表盘。
- ☐ 如何绘制表盘的刻度。
- ☐ 如何根据当前时间绘制表盘的时针、分针和秒针。

1. 绘制表盘

表盘绘制主要是通过 drawOval() 方法来实现, 该方法比较常用的重载方式为:

```
abstract void drawOval(int x, int y, int width, int height)
```

参数 x 和 y 为椭圆外切矩形左上角的坐标, 而参数 width 和 height 为椭圆外切矩形的宽度和高度。

对于表盘外切矩形, 是指窗口的边界减去非工作区域, 又由于表盘是特殊的椭圆——圆, 所以外切矩形的宽度和高度相同。在该项目中首先获取窗口的长度和宽度, 然后取这两个长度值中较小值的 0.8 倍作为圆外切矩形的宽度和高度。而对于外切矩形的左上角坐标, 则是通过圆的原点坐标减去外切矩形的宽度的一半 (即圆的半径) 来实现。为了方便编写, 以窗口的中心点作为圆的原点。

2. 绘制表盘刻度

如何绘制表盘的刻度呢? 由于它们实质上都是线, 所以只需确定它们的起始点和终点就可以实现。在本项目中刻度终点可以形成一个以表盘原点为原点, 表盘半径为半径的圆; 而起始点可以形成以表盘原点为原点, 秒针长度为半径的圆。在具体获取坐标值之前, 需要先了解一下三角函数的概念。

所谓三角函数是指, sin 值通过对边/斜边得到, cos 值通过邻边/斜边得到, 要求出一个点的坐标就是求出以这个点为直角顶点三角形的两条边, 一条边的长度为 x, 另一条的长度为 y。在该项目中已经知道的是斜边的长度, 要求出这两条边, 就需要先知道弧度值。

对于表盘的刻度, 由于一共有 60 个, 所以可以计算出来:

$$360/60=6$$

之所以为该表达式, 因为整个圆是 360° , 所以分子为 360; 又因为表盘有 12 个时钟, 在相邻时钟之间又有 5 个刻度, 所以一共是 $12 \times 5 = 60$ 个刻度, 所以分母为 60。在程序里必须要将角度转化成弧度才能进行 sin 运算, 即为 $2 \times \text{PI} / 60$ 。

得到弧度值, 又知道斜边长分别为表盘半径 (radius) 和秒针长度 (secondLength),

就可以得到终点坐标和起点坐标了:

```
//获取刻度线条的终点坐标
int x2 = center.x
    + (int) (radius * Math.sin(i * 2 * Math.PI / 60.0));
int y2 = center.y
    - (int) (radius * Math.cos(i * 2 * Math.PI / 60.0));
//获取刻度线条的起点坐标
int x1 = center.x
    + (int) ((secondLength + 1) * Math.sin(i * 2 * Math.PI
        / 60.0));
int y1 = center.y
    - (int) ((secondLength + 1) * Math.cos(i * 2 * Math.PI
        / 60.0));
```

3. 绘制表盘的秒针、分针和时针

最后, 如何绘制时针、分针和秒针呢? 由于它们实质上都是线, 所以只需确定它们的起始点和终点就可以实现。同时又由于它们的起始点都是表盘的原点, 所以只需确定 3 个指针直线的终点。

对于最长的秒针, 其每一秒都会移动一个角度, 这个可以计算出来:

$$360/60=6$$

公式原理与表盘刻度相同, 这里不再重述。得到弧度值, 就可以得到第 1 秒秒针的终点坐标了:

```
//秒针的 x 和 y 坐标
int secondX = center.x
    + (int) (secondLength * Math.sin(2 * Math.PI / 60.0));
int secondY = center.y
    - (int) (secondLength * Math.cos(2 * Math.PI / 60.0));
```

上述表达式中 center.x 为表盘原点的 x 坐标, center.y 为表盘原点的 y 坐标, secondLength 为秒针的长度。如果想表示第 2 秒秒针的终点坐标, 只要使弧度表达式变成 $\text{Math.sin}(2*2 * \text{Math.PI} / 60.0)$; 同理只要通过每秒弧度 \times 当前的秒, 就可以获取任何一秒的弧度。

对于分钟, 由于只是个很简单的模拟, 所以不需要对细节作过多的处理, 就让它每过一分钟就跳到下一个刻度, 这样求时针的顶点坐标和求秒针的顶点坐标就是一样的算法。

```
//分针的 x 和 y 坐标
int minuteX = center.x
    + (int) (minuteLength * Math.sin(minute * 2 * Math.PI / 60.0));
int minuteY = center.y
    - (int) (minuteLength * Math.cos(minute * 2 * Math.PI / 60.0));
```

上述表达式中 center.x 为表盘原点的 x 坐标, center.y 为表盘原点的 y 坐标, minuteLength 为秒针的长度, minute 为当前的分钟数。

时针的处理方式要不同, 因为时针的取值只有 12 个数字, 而分针和秒针都有 60 个数字, 如果以之前的方式处理时针, 效果就是时针到点的那一瞬间它一下子转动 30° , 为了解决该问题, 在具体计算时加入了分钟值。

例如现在是 06:15:30, 在整 6 点时它的度数是 30×6 , 整 7 点是 30×7 , 为了达到一种

过渡的效果, 我们加入分针值来参与计算, 这时其弧度值为 $\text{minute} / 60.0 + \text{hour}) * \text{Math.PI} / 6.0$, 得到弧度值, 就可以得到时针的终点坐标了:

```
//时针的 x 和 y 坐标
int hourX = center.x
    + (int) (hourLength * Math.sin((minute / 60.0 + hour) * Math.PI
    / 6.0));
int hourY = center.y
    - (int) (hourLength * Math.cos((minute / 60.0 + hour) * Math.PI
    / 6.0));
```

上述表达式中 `center.x` 为表盘原点的 x 坐标, `center.y` 为表盘原点的 y 坐标, `hourLength` 为时针的长度。minute 为当前的分钟数, hour 为当前的时钟数。

14.3 知识点扩展——从 AWT 到 Swing 的过渡

作为 JFC 类库中的一部分, Swing 提供了 40 多个组件, 是 AWT 提供组件的 4 倍。在这众多的组件中, 绝大多数组件是用来替代 AWT 组件的轻量级组件, 而剩下的一小部分是用来作为开发图形用户界面的附加组件。

为了便于程序员的学习, Swing 中提供的 AWT 组件的替代组件, 一般都是在 AWT 组件名称前增加一个 J 字母。例如窗口类 `JFrame`、显示信息的面板类 `JLabel` 等。

14.3.1 窗口类 JFrame

在 AWT 类库中如果想创建一个窗口对象, 需要通过类 `Frame` 来实现, 而在 Swing 类库中却存在一个名为 `JFrame` 的类实现窗口对象的创建。

查看 API 帮助文档可以发现, 虽然类 `JFrame` 和 `Frame` 功能相当, 但是两者在使用上却存在许多区别。首先 `JFrame` 对象只存在一个子组件 (`JRootPane`), 如果想获得该子组件, 可以通过 `getContentPane()` 方法来实现。对于 `JFrame` 的窗口对象, 如果想增加子组件及设置布局管理器, 则不能直接操作该对象, 而是作用在 `JRootPane` 对象上。其次, 当单击 `JFrame` 对象上的“关闭按钮”时, 该对象会自动隐藏但没有真正关闭, 需要在事件监听器的 `windowClosing` 事件中, 通过 `dispose()` 方法释放在内存中的资源。

下面通过一个具体的 `JFrameTest.java` 类, 来讲解如何使用 `JFrame` 组件, 具体内容如代码 14.3 所示。

代码 14.3 窗口组件: `JFrameTest.java`

```
public class JFrameTest extends JFrame {           //继承窗体类 JFrame
    public static void main(String args[]) {
        JFrameTest frame = new JFrameTest();      //创建 JFrame 对象
        frame.setVisible(true);                   //设置窗体可见, 默认为不可见
    }
    public JFrameTest() {                           //构造函数
        super();                                   //继承父类的构造方法
        setTitle("利用 JFrame 类创建的窗体");      //设置窗体的标题
        setBounds(100, 100, 500, 375);            //设置窗体的显示位置及大小
```

```
getContentPane().add(new Button("按钮")); //添加子组件到窗口上
//设置窗体关闭按钮的动作为退出
setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
}
}
```

运行 JFrameTest.java 类，会出现如图 14.8 所示的用户图形界面。



图 14.8 运行结果

【代码解析】

在上述代码中，通过调用类 JFrame 的 setDefaultCloseOperation()方法，设置 JFrame 对这个事件的处理方式为 JFrame.EXIT_ON_CLOSE，即当用户单击 JFrame 对象上的关闭窗口按钮时，将直接关闭窗口并结束程序的运行。

对于设置关闭按钮动作的 JFrame.setDefaultCloseOperation()方法，其默认动作是将窗口隐藏，可以通过该方法的入口参数来修改动作，这些参数的值如表 14.1 所示。

表 14.1 参数含义

静态常量	动作
HIDE_ON_CLOSE	隐藏窗口
DO_NOTHING_ON_CLOSE	不执行任何操作
DISPOSE_ON_CLOSE	移除窗口
EXIT_ON_CLOSE	退出窗口

14.3.2 按钮类 JButton 和面板类 JLabel

在 AWT 类库中如果想创建一个按钮和面板对象，需要通过类 Button 和 JLabel 来实现，而在 Swing 类库中是 JButton 类实现按钮对象的创建，JLabel 类实现面板对象的创建。

查看 API 帮助文档可以发现，虽然 JButton 类和 Button 类功能相当，但是组件 JButton 却增强了按钮的功能，即按钮除了可以设置标签文本之外，还可以设置图片。同样，虽然 JLabel 类和 Label 类功能相当，但是组件 JLabel 却增强了面板的功能，即面板除了可以显

示文本外，还可以显示图片。下面将通过具体的实例来讲解组件 JButton 和组件 JLabel 的使用，具体步骤如下。


1. 类JButton

(1) 设置大小

为了设置按钮的大小，在 Java API 中存在一个名为 `setBounds` 的方法，该类构造函数基本格式如下：

```
setBounds (int x,int y,int width,int height)
```

参数 `x` 和参数 `y` 用来设置按钮的显示位置，参数 `width` 和 `height` 用来设置按钮的宽度和高度。创建一个指定位置和大小按钮。

 注意：setBounds()方法的 4 个参数含义如图 14.9 所示。

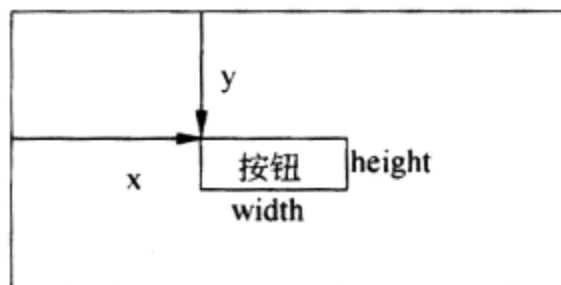


图 14.9 显示位置和大小

(2) 设置显示图片

如果想设置按钮的标签文本，通过 `setText()` 方法就可以实现。如果想设置图片，则通过如下方法来实现。

```
setIcon(Icon defaultIcon)
```

该方法用来设置按钮在默认状态下显示的图片。

```
setRolloverIcon(Icon rolloverIcon)
```

该方法用来设置当鼠标移动到按钮上显示的图片。

```
setPressedIcon(Icon pressedIcon)
```


该方法用来设置当鼠标被按下时按钮上显示的图片。

(3) 设置边框

如果设置按钮显示图片，需要设置按钮的边框和其四周的间隔都为 0。该方法的具体定义如下：

```
setMargin(Insets m)
```

参数 `m` 为 `Insets` 类的实例，用来设置按钮的边框和四周的间隔。


 注意：Insets 类的构造函数为 `Insets(int top,int left,int bottom,int right)`，各个参数的含义依次为按钮上方、左侧、下方和右侧的间隔。

为了设置按钮的大小，在 Java API 中存在一个名为 `setBounds` 的方法，该类构造函数

的基本格式如下：

```
setBounds (int x,int y,int width,int height)
```

参数 *x* 和参数 *y* 用来设置按钮的显示位置，参数 *width* 和 *height* 用来设置按钮的宽度和高度。创建一个指定位置和大小按钮。

 注意：setBounds()方法的4个参数的含义如图14.9所示。

下面通过一个具体的类 JButtonTest.java，来讲解如何使用 JButton 组件，具体内容如代码14.4所示。

代码14.4 按钮组件：JButtonTest.java

```
public class JButtonTest extends JFrame {    //继承窗体类 JFrame
    public static void main(String args[]) {    //主方法
        JButtonTest frame = new JButtonTest(); //创建 JButtonTest 对象
        frame.setVisible(true);                //设置窗体可见，默认为不可见
    }
    public JButtonTest() {                    //构造函数
        super();                             //继承父类的构造方法
        setTitle("按钮组件示例");             //设置窗体的标题
        setBounds(100, 100, 500, 375);         //设置窗体的显示位置及大小
        getContentPane().setLayout(null);      //设置为不采用任何布局管理器
        //设置窗体关闭按钮的动作为退出
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        final JButton button = new JButton();   //创建按钮对象
        button.setMargin(new Insets(0, 0, 0, 0)); //设置按钮边框和标签之间的间隔
        button.setContentAreaFilled(false);     //设置不绘制按钮的内容区域
        button.setBorderPainted(false);         //设置不绘制按钮的边框
        //设置默认情况下按钮显示的图片
        button.setIcon(new ImageIcon("img/png-1697.png"));
        //设置光标经过时显示的图片
        button.setRolloverIcon(new ImageIcon("img/png-1700.png"));
        //设置按钮被按下时显示的图片
        button.setPressedIcon(new ImageIcon("img/png-1701.png"));
        button.setBounds(200, 150, 70, 70);    //设置标签的显示位置及大小
        getContentPane().add(button);          //将按钮添加到窗体中
    }
}
```

运行 JButtonTest.java 类，会出现如图14.10所示的用户图形界面。



图14.10 运行过程

【代码解析】

- ❑ 在上述代码中,为了让按钮能够显示图片,首先通过 `setMargin(new Insets(0, 0, 0, 0))` 方法将按钮边框和标签四周的间隔均设置为 0;然后通过 `setContentAreaFilled(false)` 方法设置按钮的不绘制区域,即设置按钮的背景为透明;最后通过 `setBorderPainted(false)`方法设置按钮的边框。
- ❑ 在具体设置按钮上的图片时, `setIcon()`方法用来设置按钮默认状态下显示的图片, `setRolloverIcon()`方法用来设置鼠标移动到按钮显示的图片, `setPressedIcon()`方法用来设置按钮被按小时显示的图片。

2. 类 JLabel

(1) 显示文本

如果想显示文本可以通过 `setText(String text)` 方法来实现;如果想设置文本的字体和大小,可以通过 `setFont(Font font)`方法来实现;如果想设置文本的对齐方式,可以通过 `setHorizontalAlignment(int alignment)`方法来实现。

(2) 显示图片

如果想显示图片,可以通过 `setIcon(Icon icon)`方法来实现。如果想同时显示图片和文本,可以通过 `setHorizontalTextPosition(int textPosition)`方法来设置文字相对于图片在水平方向的显示位置;通过 `setVerticalTextPosition(int textPosition)`方法来设置文字相对于图片在垂直方向的显示位置。参数 `textPosition` 的具体值如表 14.2 所示。

表 14.2 参数的值

静态常量	含 义
TOP	文字显示在图片的上方
CENTER	文字与图片重叠显示
BOTTOM	文字显示在图片的下方

下面通过一个具体的类 `JLabelTest.java`,来讲解如何使用 `JLabel` 组件,具体内容如代码 14.5 所示。

代码 14.5 标签组件: JLabelTest.java

```
public class JLabelTest extends JFrame { //继承窗体类 JFrame
    public static void main(String args[]) { //主方法
        JLabelTest frame = new JLabelTest(); //创建 JLabelTest 类型对象
        frame.setVisible(true); //设置窗体可见,默认为不可见
    }
    public JLabelTest() { //构造函数
        super(); //继承父类的构造方法
        setTitle("标签组件示例"); //设置窗体的标题
        setBounds(100, 100, 500, 375); //设置窗体的显示位置及大小
        getContentPane().setLayout(null); //设置为不采用任何布局管理器
        //设置窗体关闭按钮的动作为退出
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        final JLabel label = new JLabel(); //创建标签对象
        label.setBounds(0, 0, 492, 341); //设置标签的显示位置及大小
        label.setText("欢迎进入 Swing 世界!"); //设置标签显示文字
```

```

        label.setFont(new Font("", Font.BOLD, 22)); //设置文字的字体及大小
        label.setHorizontalAlignment(JLabel.CENTER); //设置标签内容居中显示
        label.setIcon(new ImageIcon("img/logo.jpg")); //设置标签显示图片
        //设置文字相对图片在水平方向的显示位置
        label.setHorizontalTextPosition(JLabel.CENTER);
        //设置文字相对图片在垂直方向的显示位置
        label.setVerticalTextPosition(JLabel.BOTTOM);
        getContentPane().add(label); //将标签添加到窗体中
    }
}

```

运行 JLabelTest.java 类，会出现如图 14.11 所示的用户图形界面。



图 14.11 运行结果

【代码解析】

- 在上述代码中，为了能够在标签上显示文字，首先通过 `setText()` 方法设置标签所显示的文字，然后通过 `setFont()` 方法设置标签所显示文字的字体和大小，最后通过 `setHorizontalAlignment()` 方法设置标签所显示的文字居中。
- 在上述代码中，通过 `setIcon()` 方法设置标签所显示的图片，然后通过 `setHorizontalTextPosition()` 方法设置文字相对图片在水平方向的显示位置，最后通过 `setVerticalTextPosition()` 方法设置文字相对图片在垂直方向的显示位置。

14.3.3 单选按钮和复选框组件


在 AWT 类库中如果想创建一个选择的对象，需要通过 `Checkbox` 和 `CheckboxGroup` 类来实现。在 Swing 类库中，对于复选框的实现则需要通过 `JCheckBox` 类来实现，对于单选按钮（单选框）的实现则需要通过 `JRadioButton` 和 `ButtonGroup` 类来实现。

单选按钮组件可以单独使用，也可以通过按钮组联合使用。当单独使用时，该单选按钮可以处于选定状态和被取消状态；当联合使用时，用户只能选择按钮组中的一个单选按钮，取消操作将由按钮组对象自动完成。

`JRadioButton` 类，存在如下方法来实现相应功能：如果想设置单选按钮的标签文本，可以通过 `setText(String text)` 方法来实现；如果想设置单选按钮的状态，可以通过

setSelected(boolean b)方法来实现。

ButtonGroup 类主要负责维护组中按钮的“开启”状态，即在按钮组中只能有一个单选按钮处于被选定状态。该类存在如下方法来实现相应功能：如果想添加按钮到按钮组中，可以通过 add()方法来实现；如果想把按钮从按钮组中删除，可以通过 remove()方法来实现；如果想获取按钮组中按钮的数量，可以通过 getButtonCount()方法来实现。

说明：与单选按钮组件相比，复选框除了可以被选定和取消外，还可以同时选定多个。

JCheckBox 类存在如下方法来实现相应功能：如果想设置复选框的标签文本，可以通过 setText(String text)方法来实现；如果想设置复选框的状态，可以通过 setSelected(boolean b)方法来实现。

下面通过一个具体的 JRadioButtonTest.java 类，来讲解如何使用 JRadioButton 和 JCheckBox 组件，具体内容如代码 14.6 所示。

代码 14.6 单选按钮和复选框组件：JRadioButtonTest.java

```
public class JBoxTest extends JFrame {           //继承窗体类 JFrame
    public static void main(String args[]) {     //主方法
        JBoxTest frame = new JBoxTest();        //创建 JBoxTest 类的对象
        frame.setVisible(true);                 //设置窗体可见，默认为不可见
    }
    public JBoxTest() {                          //构造函数
        super();                                //继承父类的构造方法
        setTitle("单选按钮和复选框组件示例");   //设置窗体的标题
        setBounds(100, 100, 500, 375);          //设置窗体的显示位置及大小
        getContentPane().setLayout(null);       //设置为不采用任何布局管理器
        //设置窗体关闭按钮的动作为退出
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //单选按钮的创建和设置
        final JLabel label = new JLabel();       //创建标签对象
        label.setText("性别: ");                //设置标签文本
        label.setBounds(200, 110, 46, 15);      //设置标签的显示位置及大小
        getContentPane().add(label);            //将标签添加到窗体中
        ButtonGroup buttonGroup = new ButtonGroup(); //创建按钮组对象
        //创建单选按钮对象 manRadioButton
        final JRadioButton manRadioButton = new JRadioButton();
        buttonGroup.add(manRadioButton);         //将单选按钮添加到按钮组中
        manRadioButton.setSelected(true);         //设置单选按钮默认为被选中
        manRadioButton.setText("男");            //设置单选按钮的文本
        manRadioButton.setBounds(252, 106, 46, 23); //设置单选按钮的显示位置及大小
        getContentPane().add(manRadioButton);   //将单选按钮添加到窗体中
        //创建单选按钮对象 womanRadioButton
        final JRadioButton womanRadioButton = new JRadioButton();
        buttonGroup.add(womanRadioButton);       //将单选按钮添加到按钮组中
        womanRadioButton.setText("女");          //设置单选按钮的文本
        womanRadioButton.setBounds(304, 106, 46, 23); //设置单选按钮的显示位置及大小
        getContentPane().add(womanRadioButton); //将单选按钮添加到窗体中
        //复选框的创建和设置
        final JLabel label1 = new JLabel();      //创建标签对象
```

```

label1.setText("爱好: "); //设置标签文本
label1.setBounds(200, 150, 46, 15); //设置标签的显示位置及大小
getContentPane().add(label1); //将标签添加到窗体中
//创建和设置复选框对象 readingCheckBox
//创建复选框对象 readingCheckBox
final JCheckBox readingCheckBox = new JCheckBox();
readingCheckBox.setText("读书"); //设置复选框的标签文本
readingCheckBox.setBounds(262, 146, 55, 23);
//设置复选框的显示位置及大小
getContentPane().add(readingCheckBox); //将复选框添加到窗体中
//创建和设置复选框对象 musicCheckBox
//创建复选框对象 musicCheckBox
final JCheckBox musicCheckBox = new JCheckBox();
musicCheckBox.setText("听音乐"); //设置复选框的标签文本
musicCheckBox.setBounds(314, 146, 68, 23);
//设置复选框的显示位置及大小
getContentPane().add(musicCheckBox); //将复选框添加到窗体中
//创建和设置复选框对象 pingpongCheckBox
//创建复选框对象 pingpongCheckBox
final JCheckBox pingpongCheckBox = new JCheckBox();
pingpongCheckBox.setText("乒乓球"); //设置复选框的标签文本
pingpongCheckBox.setBounds(387, 146, 75, 23);
//设置复选框的显示位置及大小
getContentPane().add(pingpongCheckBox); //将复选框添加到窗体中
}
}

```

运行 JRadioButtonTest.java 类，会出现如图 14.12 所示的用户图形界面。

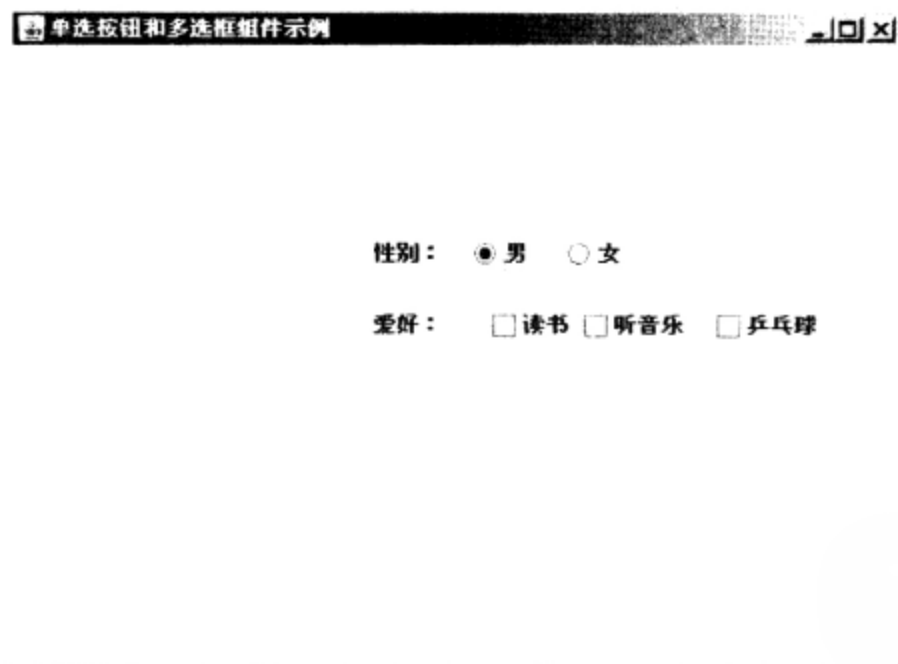


图 14.12 运行结果

【代码解析】

- ❑ 为了能够显示单选按钮，首先创建按钮组和单选按钮对象，同时通过 add() 方法把单选按钮添加到按钮组里；然后通过 setSelected() 和 setText() 方法设置单选按钮的选中状态和文本；最后通过 setBounds() 方法设置单选按钮的显示位置及大小。
- ❑ 为了能够显示复选框，首先创建复选框对象；然后通过 setText() 方法设置复选框对象的文本；最后通过 setBounds() 方法设置复选框对象的显示位置及大小。

14.3.4 选择框组件

在 AWT 类库中如果想创建一个选择框的对象，需要通过 Choice 和 List 类来实现，而在 Swing 类库中只存在 JList 的复选框，同时利用 JComboBox 组件实现和加强了 Choice 组件的功能。查看 API 帮助文档可以发现，JComboBox 组件除了可以实现单选功能外，还可以实现进行编辑功能。

JComboBox 类存在如下方法来实现相应功能：如果想设置选择框的默认选项，可以通过 setSelectedIndex()方法来实现；如果想设置选择框的编辑状态，可以通过 setEditable()方法来实现。

JList 类存在如下方法来实现相应功能：如果想设置列表框的选择模式，可以通过 setSelectionMode(int selectionMode)方法实现。selectionMode 值可以是表 14.3 中的任何一个。

表 14.3 selectionMode值

常 量	含 义
SINGLE_SELECTION	只允许单选
SINGLE_INTERVAL_SELECTION	只允许连续多选
MULTIPLE_INTERVAL_SELECTION	允许连续选择，又允许间隔选择

下面通过一个具体的类 JRadioButtonTest.java，来讲解如何使用 JComboBox 和 JList 组件，具体内容如代码 14.7 所示。

代码 14.7 选择框和列表框组件：JListTest.java

```
public class JListTest extends JFrame { //继承窗体类 JFrame
    public static void main(String args[]) { //主方法
        JListTest frame = new JListTest(); //创建 JListTest 类对象
        frame.setVisible(true); //设置窗体可见，默认为不可见
    }
    public JListTest() { //构造函数
        super(); //继承父类的构造方法
        setTitle("选择框和列表框相关组件示例"); //设置窗体的标题
        setBounds(100, 100, 500, 375); //设置窗体的显示位置及大小
        getContentPane().setLayout(null); //设置为不采用任何布局管理器
        //设置窗体关闭按钮的动作为退出
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //创建和设置选择框对象
        final JLabel label = new JLabel(); //创建标签对象
        label.setText("学历: "); //设置标签文本
        label.setBounds(10, 10, 46, 15); //设置标签的显示位置及大小
        getContentPane().add(label); //将标签添加到窗体中
        String[] schoolAges = { "本科", "硕士", "博士" }; //创建选项数组
        //创建选择框对象 comboBox
        JComboBox comboBox = new JComboBox(schoolAges);
        comboBox.setEditable(true); //设置选择框为可编辑
        //设置选择框弹出时显示选项的最多行数
        comboBox.setMaximumRowCount(3);
```

```

comboBox.insertItemAt("大专", 0);           //在索引为 0 的位置插入一个选项
comboBox.setSelectedItem("本科");           //设置索引为 0 的选项被选中
comboBox.setBounds(62, 7, 104, 21);        //设置选择框的显示位置及大小
getContentPane().add(comboBox);            //将选择框添加到窗体中
//创建和设置列表对象
final JLabel label1 = new JLabel();         //创建标签对象 label1
label1.setText("爱好: ");                  //设置标签文本
label1.setBounds(10, 60, 46, 15);          //设置标签的显示位置及大小
getContentPane().add(label1);              //将标签添加到窗体中
//创建字符串数组对象 likes
String[] likes = { "读书", "听音乐", "跑步", "乒乓球", "篮球", "游泳", "滑雪" };
JList list = new JList(likes);              //创建列表对象
list.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
list.setFixedCellHeight(20);               //设置选项高度
list.setVisibleRowCount(4);                //设置选项可见个数
JScrollPane scrollPane = new JScrollPane(); //创建滚动面板对象
//将列表添加到滚动面板中
scrollPane.setViewportView(list);
scrollPane.setBounds(62, 55, 65, 80);      //设置滚动面板的显示位置及大小
getContentPane().add(scrollPane);          //将滚动面板添加到窗体中
}
}

```

运行 JListTest.java 类, 会出现如图 14.13 所示的用户图形界面。



图 14.13 运行结果

【代码解析】

- ❑ 为了能够显示选择框, 首先创建设置好选项内容的选择框对象, 同时通过 `setEditable()` 方法设置选择框是否为可编辑; 然后通过 `setMaximumRowCount()`、`insertItemAt()` 和 `setSelectedItem()` 方法设置多选框对象弹出时显示选项的最多行数、在相应索引处设置相应选项内容和设置多选框的默认选中项; 最后通过 `setBounds()` 方法设置单选按钮的显示位置及大小。
- ❑ 为了能够在滚动面板上添加列表, 首先创建设置好选项的列表对象; 然后通过 `setFixedCellheight()` 和 `setVisibleRowCount()` 方法设置选项高度和列表对象选项的可见个数; 最后通过 `setSelectionMode()` 方法设置列表框的选择模式。

14.3.5 输入框组件

在 AWT 类库中如果想创建一个输入对象，需要通过 TextField 和 TextArea 类来实现，而在 Swing 类库中除了存在 JTextField 的文本框和 JTextArea 文本域外，还有一个 JPasswordField 的密码框。

对于 JTextField 类，存在如下方法来实现相应功能：如果想设置文本的对齐方式，可以通过 setHorizontalAlignment(int alignment)方法来实现。参数 alignment 的具体值如表 14.4 所示。

表 14.4 参数的值

静 态 常 量	含 义
LEFT	靠左显示
CENTER	居中显示
RIGHT	靠右侧显示

对于 JTextField 类，存在如下方法来实现相应功能：如果想设置文本的换行情况，可以通过 setLineWrap(boolean wrap)方法来实现。

对于 JPasswordField 类，存在如下方法来实现相应功能：如果想获取密码，可以通过 getPassword()方法来实现；如果想修改和获取回显字符，可以通过 setEchoChar(char c)和 getEchoChar(char c)方法。

对于 JTextArea 类，存在如下方法来实现相应功能：如果想设置文本是否自动换行，可以通过 setLineWrap()方法来实现，默认为 false，即不自动换行。

下面通过一个具体的类 JFieldTest，来讲解如何使用 JTextField、JTextArea 和 JPasswordField 组件，具体内容如代码 14.8 所示。

代码 14.8 输入组件：JFieldTest.java

```
public class JFieldTest extends JFrame {           //继承窗体类 JFrame
    public static void main(String args[]) {
        JFieldTest frame = new JFieldTest();      //创建 JFieldTest 类对象
        //设置窗体可见，默认为不可见
        frame.setVisible(true);
    }
    public JFieldTest() {                           //创建构造函数
        super();                                    //继承父类的构造方法
        setTitle("输入组件示例");                  //设置窗体的标题
        setBounds(100, 100, 500, 375);             //设置窗体的显示位置及大小
        //设置为不采用任何布局管理器
        getContentPane().setLayout(null);
        //设置窗体关闭按钮的动作为退出
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //创建和设置 JTextField 对象
        final JLabel label = new JLabel();          //创建标签对象
        label.setText("姓名: ");                   //设置标签文本
        label.setBounds(10, 10, 46, 15);           //设置标签的显示位置及大小
        getContentPane().add(label);               //将标签添加到窗体中
        //创建文本框对象 textField
```



```

JTextField textField = new JTextField();
//设置文本框内容的水平对齐方式
textField.setHorizontalAlignment(JTextField.CENTER);
textField.setFont(new Font("", Font.BOLD, 12));

//设置文本框内容的字体样式
textField.setBounds(62, 7, 150, 21); //设置文本框的显示位置及大小
getContentPane().add(textField); //将文本框添加到窗体中
//创建和设置 passwordField 对象
final JLabel label1 = new JLabel(); //创建标签对象 label1
label1.setText("密码: "); //设置标签文本
label1.setBounds(10, 70, 46, 15); //设置标签的显示位置及大小
getContentPane().add(label1); //将标签添加到窗体中
//创建密码框对象 passwordField
JPasswordField passwordField = new JPasswordField();
passwordField.setEchoChar('¥'); //设置回显字符为 '¥'
passwordField.setBounds(62, 67, 150, 21); //设置密码框的显示位置及大小
getContentPane().add(passwordField); //将密码框添加到窗体中
//创建和设置 JTextArea 对象
final JLabel label2 = new JLabel(); //创建标签对象 label2
label2.setText("备注: "); //设置标签文本
label2.setBounds(10, 140, 46, 15); //设置标签的显示位置及大小
getContentPane().add(label2); //将标签添加到窗体中
//创建文本域对象 textArea
JTextArea textArea = new JTextArea();
textArea.setColumns(15); //设置文本域显示文字的列数
textArea.setRows(3); //设置文本域显示文字的行数
textArea.setLineWrap(true); //设置文本域自动换行
final JScrollPane scrollPane = new JScrollPane(); //创建滚动面板对象
scrollPane.setViewportView(textArea); //将文本域添加到滚动面板中
Dimension dime = textArea.getPreferredSize(); //获得文本域的首选大小
scrollPane.setBounds(62, 125, dime.width, dime.height);
//设置滚动面板的位置及大小
getContentPane().add(scrollPane); //将滚动面板添加到窗体中
}
}

```

运行 JFieldTest.java 类, 会出现如图 14.14 所示的用户图形界面。



图 14.14 运行结果

【代码解析】

在上述代码中，窗口对象中存在 3 种组件，分别为面板对象 label、文本域对象和密码框对象 passwordField。在具体设置 passwordField 对象时，通过 setEchoChar('Y')方法设置回显字符为 Y。

14.4 小 结

本章主要通过线程和事件的知识实现绘制指针时钟的功能，虽然该项目比较小，只包含两个类：Clock 类用来实现“指针时钟”的界面；DrawClock 类用来实现绘制指针时钟，但是却涉及许多知识点。在具体实现 Clock 类时，涉及 Swing 的知识和多线程知识，而在具体实现 DrawClock 类时，涉及绘制指针的算法。

第 15 章 控制动画项目

(JSlider 和 Timer 组件)

在 Java 语言的 Swing 中存在一个名叫滑杆的组件，该组件非常实用。本章不仅讲解滑杆 JSlider 组件的基础知识，还将通过该组件实现一个小项目——控制动画项目。

本章的学习目标如下：

- ❑ 掌握控制动画项目；
- ❑ 掌握 Swing 组件的多线程组件 Timer 和滑杆组件 JSlider 的基础知识；
- ❑ 如何修改组件的 UI。

15.1 控制动画原理

“控制动画项目”主要通过滑杆对象实现控制图片的播放速度，在具体运行时，只要通过拖动滑杆对象上的滑块，就可以改变图片的播放速度。

15.1.1 项目结构框架分析

控制动画项目非常简单，只通过一个类就可以实现该项目功能的类。通过该项目的目录可以发现，除了控制动画的类外，还存在一个 images 文件，该文件里存放运动主角运动过程中的各种图片，如图 15.1 所示。

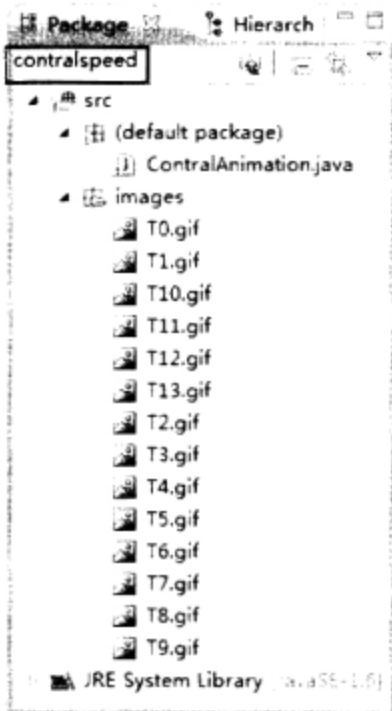


图 15.1 项目目录

15.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括动画的初始化界面、减慢动画的速度和增加动画的速度。

1. 动画的初始化

当运行测试控制动画的类 `ContralAnimation` 后, 会出现如图 15.2 所示的初始界面。在该界面中一只狗将以一定的速度移动, 滑杆对象上滑块的值 15。

2. 减慢动画的速度

在动画运行的过程中, 如果想减慢狗的运动速度, 可以移动滑杆对象上的滑块, 狗的动画速度会随着滑块值的减小而变慢, 这时滑块的值 10, 如图 15.3 所示。

3. 增加动画的速度

在动画运行的过程中, 如果想增加狗的运动速度, 可以移动滑杆对象上的滑块, 狗的动画速度会随着滑块值的增大而变快, 这时滑块的值 30, 如图 15.4 所示。



图 15.2 控制动画的初始界面



图 15.3 减慢动画速度



图 15.4 增加动画速度

15.2 控制动画的实现过程

为了实现控制动画项目, 通过计时器组件 `Timer` 实现该项目的多线程机制, 通过监听滑杆 `JSlider` 组件的值改变事件来控制图片的播放速度。 `ContralSpeed.java` 类为控制动画项目的主类, 该类的 UML 如图 15.5 所示。

15.2.1 控制动画的主界面

首先在控制动画类中实现该项目的界面, 在该界面中不仅涉及 `Timer` 计时器组件而且还涉及 `JSlider` 滑杆组件, 具体内容如代码 15.1 所示。



图 15.5 控制动画类图

代码 15.1 控制动画的界面: ContralSpeed.java

```

public class ContralSpeed extends JPanel implements ActionListener,
    WindowListener, ChangeListener {
...
    //设置动画速度的参数
    static final int FPS_MIN = 0;           //设置最小值
    static final int FPS_MAX = 30;          //设置最大值
    static final int FPS_INIT = 15;         //初始数值
    int frameNumber = 0;                    //图片的帧数
    int NUM_FRAMES = 14;                    //图片的数目
    ImageIcon[] images = new ImageIcon[NUM_FRAMES]; //创建 ImageIcon 类型数组
    int delay;                             //计时器的延迟时间
    Timer timer;                           //创建计时器对象
    boolean frozen = false;                 //boolean 型变量
    //这个标签用来显示这只小狗
    JLabel picture;                         //显示图像的标签对象
    public ContralSpeed() {                 //构造函数
        //设置布局管理器
        setLayout(new BoxLayout(this, BoxLayout.PAGE_AXIS));
        delay = 1000 / FPS_INIT;
        //信息提示标签
        JLabel sliderLabel = new JLabel("设置动画播放速度", JLabel.CENTER);
        //设置对齐方式
        sliderLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        //创建一个滑杆对象, 定义其的最小值和最大值及初始值
        JSlider framesPerSecond = new JSlider(JSlider.HORIZONTAL, FPS_MIN,
            FPS_MAX, FPS_INIT);
        framesPerSecond.addChangeListener(this); //添加事件监听器
        //设置滑杆对象参数
        framesPerSecond.setMajorTickSpacing(10); //设置每隔 10 刻度标注一次
        framesPerSecond.setMinorTickSpacing(1); //设置最小刻度为 1
        framesPerSecond.setPaintTicks(true);    //设置显示刻度
        framesPerSecond.setPaintLabels(true);   //设置显示滑动块
    }
}
  
```

```
//设置滑杆对象的位置
framesPerSecond.setBorder(BorderFactory.createEmptyBorder(0, 0, 10,
0));
//创建和设置显示图片的标签
picture = new JLabel(); //为对象 picture 赋值
//设置对象 picture 的水平和垂直对齐方式
picture.setHorizontalAlignment(JLabel.CENTER);
picture.setAlignmentX(Component.CENTER_ALIGNMENT);
//设置对象 picture 的位置
picture.setBorder(BorderFactory.createCompoundBorder(BorderFactory
.createLoweredBevelBorder(),BorderFactory.createEmptyBorder(
10, 10, 10, 10)));
updatePicture(0); //显示第一张图
//将成员添加到主面板上
add(sliderLabel);
add(framesPerSecond);
add(picture);
//设置主面板的位置
setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
//设置一个计时器对象 timer
timer = new Timer(delay, this); //为对象 timer 赋值
timer.setInitialDelay(delay * 7); //设置每轮循环停顿的时间
timer.setCoalesce(true); //设置计时器进行重复循环
}
...
}
```

【代码解析】

上述代码实现了控制动画速度的项目界面，该用户界面涉及的具体容器、对象和布局如图 15.6 所示。

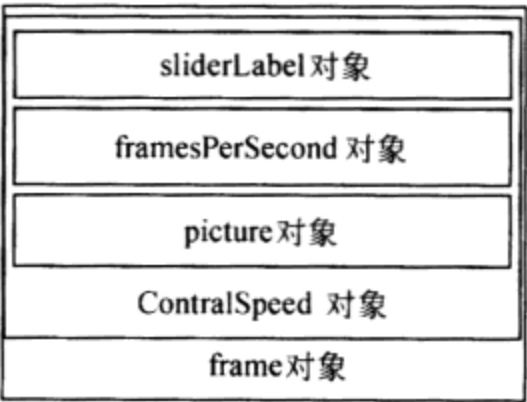


图 15.6 布局

15.2.2 控制动画的逻辑

在类 ContralSpeed 中为了实现控制动画项目逻辑，主要监听了滑杆组件的值改变事件和动作事件，具体内容如代码 15.2 所示。

代码 15.2 控制动画的逻辑: ContralSpeed.java

```
public class ContralSpeed extends JPanel implements ActionListener,
WindowListener, ChangeListener {
```

```

...
//处理各种窗口事件
void addWindowListener(Window w) {
    w.addWindowListener(this);
}
public void windowIconified(WindowEvent e) {
    stopAnimation();
}
public void windowDeiconified(WindowEvent e) {
    startAnimation();
}
public void windowOpened(WindowEvent e) {
}
public void windowClosing(WindowEvent e) {
}
public void windowClosed(WindowEvent e) {
}
public void windowActivated(WindowEvent e) {
}
public void windowDeactivated(WindowEvent e) {
}
public void stateChanged(ChangeEvent e) { //编写事件监听器
    JSlider source = (JSlider) e.getSource(); //获取发生事件的事件源
    if (!source.getValueIsAdjusting()) {
        int fps = (int) source.getValue(); //获得滑动杆的值
        if (fps == 0) {
            if (!frozen)
                stopAnimation();
        } else {
            delay = 1000 / fps;
            timer.setDelay(delay);
            timer.setInitialDelay(delay * 10);
            if (frozen)
                startAnimation();
        }
    }
}
public void startAnimation() { //开始动画方法
    timer.start(); //启动计时器对象
    frozen = false; //改变对象 frozen 的值
}
public void stopAnimation() { //停止动画方法
    timer.stop(); //停止计时器对象
    frozen = true; //改变对象 frozen 的值
}
public void actionPerformed(ActionEvent e) { //编写事件监听方法
    //改变图片帧数
    if (frameNumber == (NUM_FRAMES - 1)) {
        frameNumber = 0;
    } else {
        frameNumber++;
    }
    updatePicture(frameNumber); //显示下一张图片
    if (frameNumber == (NUM_FRAMES - 1)
        || frameNumber == (NUM_FRAMES / 2 - 1)) {
        timer.restart(); //重启计时器对象
    }
}

```



```

    }
    //更新图片方法
    protected void updatePicture(int frameNum) {
        if (images[frameNumber] == null) {
            //创建 images 对象
            images[frameNumber] = createImageIcon("images/T" + frameNumber
                + ".gif");
        }
        //绘制图片
        if (images[frameNumber] != null) { //当存在图片时
            picture.setIcon(images[frameNumber]);
        } else { //当不存在图片时
            picture.setText("image #" + frameNumber + " not found");
        }
    }
    //获取图片方法
    protected static ImageIcon createImageIcon(String path) {
        //创建 URL 类型对象 imgURL
        java.net.URL imgURL = ContralSpeed.class.getResource(path);
        if (imgURL != null) { //判断对象 imgURL
            return new ImageIcon(imgURL); //创建 ImageIcon 对象
        } else {
            //输出出错信息
            System.err.println("Couldn't find file: " + path);
            return null;
        }
    }
}

public static void main(String[] args) { //主方法
    JFrame.setDefaultLookAndFeelDecorated(true);
    //定义窗体
    JFrame frame = new JFrame("控制动画速度");
    //设置关闭功能
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    ContralSpeed animator = new ContralSpeed(); //创建 ContralSpeed 类
    animator.setOpaque(true); //设置透明度
    frame.setContentPane(animator); //添加对象 animator 到窗体
    //显示窗体
    frame.pack(); //窗体自适应大小
    frame.setVisible(true); //显示窗体
    animator.startAnimation();
}
...
}

```

【代码解析】

在上述代码中，主要监听两方面的事件：滑杆改变事件（stateChanged()）和动作事件（actionPerformed()）。对于前者，当获得滑杆的值为 0 时，则通过调用停止动画方法（stopAnimation()）实现停止功能；当获得滑杆的值不为 0 时，则通过修改类 timer 延迟间隔时间参数 delay 的值改变动画的速度。而对于后者，主要通过 updatePicture()方法显示下一张图像及修改帧参数的值。

对于 updatePicture()方法，其主要用来实现在标签组件上显示图片，但到底显示哪一张图片，主要通过方法 createImageIcon()来实现。

15.3 知识点扩展——JSlider 和 Timer 组件的基础知识

在控制动画速度项目中，除了涉及用户界面的基础知识外，还用到了两个 Swing 组件，即 JSlider（滑杆组件）和 Timer（记时器组件）。本节将详细讲解滑杆组件的基本知识。

15.3.1 使用 JSlider 组件创建无刻度的滑杆

对于 Swing 中的 JSlider 组件，其允许用户能够选择某一范围内的一个整数值。由于该组件继承 JComponent 类并拥有丰富的属性和方法，所以可以通过详细地定制其属性，以符合用户的需求。下面通过一个实例来讲解组件 JSlider 的属性和方法，从而可以实现定制的各种滑杆。

下面通过类 SliderTest1 详细演示如何创建无刻度的滑杆，具体内容如代码 15.3 所示，该类的类图如图 15.7 所示。

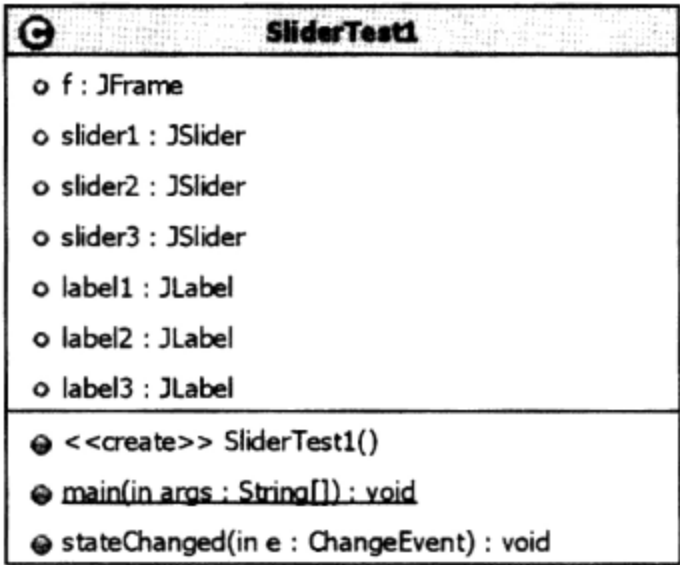


图 15.7 无刻度滑杆类图

代码 15.3 无刻度滑杆：SliderTest1.java

```
public class SliderTest1 implements ChangeListener {
    //创建成员变量
    JFrame f = null; //主窗口
    //创建 3 个滑杆对象
    JSlider slider1;
    ... //省略部分代码
    //创建用来显示信息的标签对象
    JLabel label1;
    ... //省略部分代码
    public SliderTest() { //构造函数
        f = new JFrame("滑杆测试");
        Container contentPane = f.getContentPane(); //获取 Container
        //创建和设置面板对象 panell
        JPanel panell = new JPanel();
        panell.setLayout(new GridLayout(2, 1)); //设置布局管理器
    }
}
```

```

slider1 = new JSlider();           //建立一个默认的 JSlider 组件
//设置标签的显示系信息
label1 = new JLabel("目前刻度: " + slider1.getValue());
//添加对象 label1 和 slider1 到面板 panel1 中
panel1.add(label1);
panel1.add(slider1);
//设置面板对象 panel1 边框
panel1.setBorder(BorderFactory.createTitledBorder(BorderFactory
    .createEtchedBorder(), "Slider 1", TitledBorder.LEFT,
    TitledBorder.TOP));

//创建和设置面板对象 panel2
JPanel panel2 = new JPanel();
panel2.setLayout(new GridLayout(2, 1));
slider2 = new JSlider(JSlider.HORIZONTAL); //创建一个水平滑杆对象
slider2.setMinimum(0);                     //设置取值的最小值
slider2.setMaximum(100);                   //设置取值的最大值
slider2.setValue(30);                      //设置取值的当前值
slider2.setExtent(50);                     //设置取值的延伸区值
label2 = new JLabel("目前刻度: " + slider2.getValue());
panel2.add(label2);
panel2.add(slider2);
panel2.setBorder(BorderFactory.createTitledBorder(BorderFactory
    .createEtchedBorder(), "Slider 2", TitledBorder.LEFT,
    TitledBorder.TOP));

//创建和设置面板对象 panel3
JPanel panel3 = new JPanel();
panel3.setLayout(new GridLayout(2, 1));
//建立一个具有最大、最小值的滑杆对象 slider3
slider3 = new JSlider(20, 80);
slider3.setOrientation(JSlider.VERTICAL); //设置对象 slider3 为垂直方向滑杆
label3 = new JLabel("目前刻度: " + slider3.getValue());
panel3.add(label3);
panel3.add(slider3);
panel3.setBorder(BorderFactory.createTitledBorder(BorderFactory
    .createEtchedBorder(), "Slider 3", TitledBorder.LEFT,
    TitledBorder.TOP));
//为 3 个滑杆对象添加事件监听器
slider1.addChangeListener(this);
... //省略部分代码
//设置 3 个面板对象的大小
panel1.setPreferredSize(new Dimension(300, 100));
... //省略部分代码
GridBagConstraints c;
int gridx, gridy, gridwidth, gridheight, anchor, fill, ipadx, ipady;
double weightx, weighty;
Insets inset;
//创建 GridBagLayout 对象
GridBagLayout gridbag = new GridBagLayout();
contentPane.setLayout(gridbag); //设置 contentPane 对象的布局管理器
gridx = 0;                      //第 0 行
gridy = 0;                      //第 0 列
gridwidth = 2;                  //占两单位宽度
gridheight = 1;                 //占一单位高度
weightx = 0;                    //窗口增大时组件宽度增大比率 0
weighty = 0;                    //窗口增大时组件高度增大比率 0

```

```

anchor = GridBagConstraints.CENTER; //容器大于组件 size 时将组件
//置于容器中央
fill = GridBagConstraints.BOTH; //窗口拉大时会填满水平与垂直空间
inset = new Insets(0, 0, 0, 0); //组件间间距
ipadx = 0; //组件内水平宽度
ipady = 0; //组件内垂直高度
//创建对象 c
c = new GridBagConstraints(gridx, gridy, gridwidth, gridheight,
    weightx, weighty, anchor, fill, inset, ipadx, ipady);
... //省略部分代码
//修改对象 c
c = new GridBagConstraints(gridx, gridy, gridwidth, gridheight,
    weightx, weighty, anchor, fill, inset, ipadx, ipady);
gridbag.setConstraints(panel2, c);
contentPane.add(panel2);
gridx = 2;
gridy = 0;
gridwidth = 1; //占一单位宽度
gridheight = 2; //占两单位高度
... //省略部分代码
//显示窗口对象 f
f.pack();
f.setVisible(true);
f.addWindowListener(new WindowAdapter() { //窗口事件监听器
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
});
}
public static void main(String[] args) { //主方法
    new SliderTest1();
}
//处理 ChangeEvent 事件, 当用户移动滑杆时, label 上的值会随着用户的移动而改变
public void stateChanged(ChangeEvent e) {
    if ((JSlider) e.getSource() == slider1)
        label1.setText("目前刻度: " + slider1.getValue());
    if ((JSlider) e.getSource() == slider2)
        label2.setText("目前刻度: " + slider2.getValue());
    if ((JSlider) e.getSource() == slider3)
        label3.setText("目前刻度: " + slider3.getValue());
}
}

```

运行 SliderTest1.java 类, 会出现如图 15.8 所示的界面。

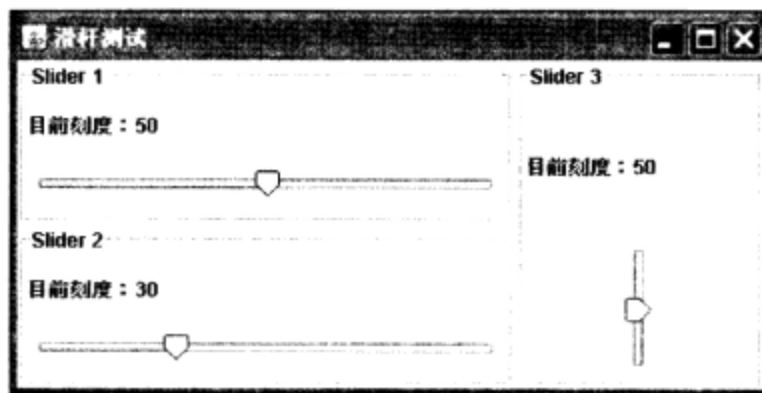


图 15.8 运行结果

【代码解析】

- ❑ 对于图中的滑杆对象 slider1，由于其是通过无参构造函数实现的，所以其取值的最小值为 0，最大值为 100，初始值为 50。对于滑动杆对象，如果初始值没有指定，其大小可以通过表达式“(最大值-最小值)/2”来获取。
- ❑ 对于滑杆对象 slider2，在通过构造函数具体创建时指定了为水平滑杆，创建后又通过 setMinimum()、setMaximum()和 setValue()方法设置其取值的最小值为 0，最大值为 100，默认值为 30，最后还通过 setExtent(50)方法设置了取值的延伸区值为 50。所谓延伸区是指限制 JSlider 组件刻度可变动的范围，其就像是一个障碍区，不允许滑杆通行的区。如果存在延伸区值，那么 JSlider 组件刻度可变动区域大小可以通过表达式“最大值-最小值-延伸区值”来获取。对于对象 slider2，其刻度可变区域大小为 100-0-50=50，即当 slider2 的刻度超过 50 时，其刻度值仍然维持在 50 上。
- ❑ 对于滑杆对象 slider3，在通过构造函数具体创建时，指定了其取值的最小值 (20) 和最大值 (80)，创建后又通过 setOrientation()方法设置为垂直滑杆。

15.3.2 使用 JSlider 组件创建带数字刻度的滑杆

下面通过类 SliderTest2 详细讲解如何创建数字刻度的滑杆，具体内容如代码 15.4 所示，该类的类图如图 15.9 所示。

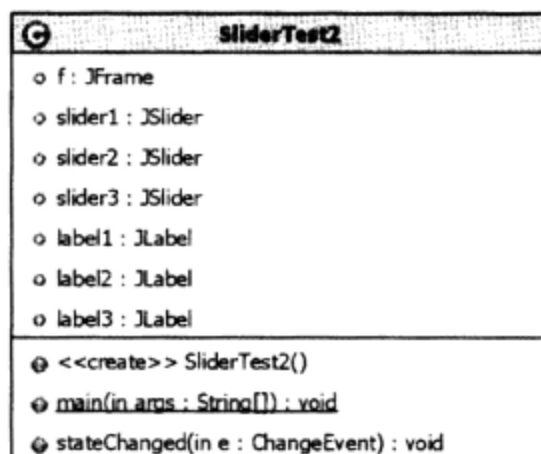


图 15.9 数字刻度滑杆类图

代码 15.4 数字刻度滑杆: SliderTest2.java

```
public class SliderTest2 implements ChangeListener {
    ....
    public SliderTest2() {
        f = new JFrame("滑杆测试");
        Container contentPane = f.getContentPane();
        //创建和设置面板对象 panel1
        JPanel panel1 = new JPanel();
        panel1.setLayout(new GridLayout(2, 1));
        slider1 = new JSlider();
        //滑杆对象 slider1 刻度的设置
        slider1.setPaintTicks(true);           //设置对象 slider1 是显示刻度
        slider1.setMajorTickSpacing(20);       //设置每 20 刻度标注一次
        slider1.setMinorTickSpacing(10);       //设置最小刻度为 10
    }
}
```

```

slider1.setPaintLabels(true);           //设置刻度的显示为数字标记
slider1.setPaintTrack(true);            //设置显示滑块
slider1.setSnapToTicks(true);           //设置滑杆的滑块一次移动一个小刻度
label1 = new JLabel("目前刻度: " + slider1.getValue());
panel1.add(label1);
panel1.add(slider1);
panel1.setBorder(BorderFactory.createTitledBorder(BorderFactory
    .createEtchedBorder(), "Slider 1", TitledBorder.LEFT,
    TitledBorder.TOP));

//创建和设置面板对象 panel2
JPanel panel2 = new JPanel();
panel2.setLayout(new GridLayout(2, 1));
slider2 = new JSlider(JSlider.HORIZONTAL);
slider2.setMinimum(0);
slider2.setMaximum(100);
slider2.setValue(30);
slider2.setExtent(50);
//设置滑杆对象 slider2 的刻度
slider2.setPaintTicks(true);             //设置对象 slider2 显示刻度
slider2.setMajorTickSpacing(10);         //设置每 10 刻度标注一次
slider2.setMinorTickSpacing(2);         //设置最小刻度为 2
slider2.setPaintLabels(true);           //设置刻度的显示为数字标记
//使滑杆滑块的左右颜色不一样
slider2.putClientProperty("JSlider.isFilled", Boolean.TRUE);
label2 = new JLabel("目前刻度: " + slider2.getValue());
panel2.add(label2);
panel2.add(slider2);
panel2.setBorder(BorderFactory.createTitledBorder(BorderFactory
    .createEtchedBorder(), "Slider 2", TitledBorder.LEFT,
    TitledBorder.TOP));

//创建和设置面板对象 panel3
JPanel panel3 = new JPanel();
panel3.setLayout(new GridLayout(2, 1));
slider3 = new JSlider(20, 80);
slider3.setOrientation(JSlider.VERTICAL);
//设置滑杆对象 slider2 的刻度
slider3.setPaintTicks(true);             //设置对象 slider3 的显示刻度
slider3.setMajorTickSpacing(30);         //设置每 30 刻度标注一次
slider3.setMinorTickSpacing(10);        //设置最小刻度为 10
slider3.setPaintLabels(true);           //设置刻度的显示为数字标记
//使滑杆滑块的左右颜色不一样
slider3.putClientProperty("JSlider.isFilled", Boolean.TRUE);
label3 = new JLabel("目前刻度: " + slider3.getValue());
panel3.add(label3);
panel3.add(slider3);
panel3.setBorder(BorderFactory.createTitledBorder(BorderFactory
    .createEtchedBorder(), "Slider 3", TitledBorder.LEFT,
    TitledBorder.TOP));

...                                     //省略部分代码
}
public static void main(String[] args) { //主方法
    new SliderTest2();
}

public void stateChanged(ChangeEvent e) { //事件监听器方法
...

```



```
    }  
}
```

运行 SliderTest2.java 类，会出现如图 15.10 所示的界面。

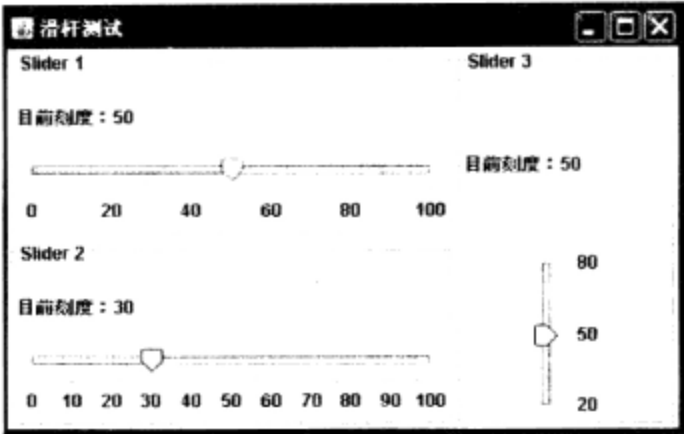


图 15.10 运行结果

【代码解析】

- ❑ 对于滑杆对象 slider1，在具体设置刻度时，首先通过 setPaintTicks(true)方法使该对象能够显示刻度；然后通过 setMajorTickSpacing()方法设置主刻度线间隔为 20，通过 setMinorTickSpacing()方法设置次刻度线间隔为 10；最后通过 setPaintLabels()方法设置主刻度线的标签为数字。为了方便用户移动刻度，通过 setPaintTrack()方法显示滑杆的滑块，通过 setSnapToTicks()方法设置移动的小刻度为次刻度线间隔。
- ❑ 对于滑杆对象 slider2，不仅通过设置刻度的方法，使其主刻度线间隔为 10，次刻度线间隔为 2，主刻度线的标签为数字，而且还通过 putClientProperty("JSlider.isFilled", Boolean.true)方法使滑杆滑块左右的颜色不一样。
- ❑ 对于滑杆对象 slider3，通过设置刻度的方法，使其主刻度线间隔为 30，次刻度线间隔为 10，主刻度线的标签为数字，滑杆滑块左右的颜色不一样。

15.3.3 使用 JSlider 组件创建带字符刻度的滑杆

下面通过类 SliderTest3 详细讲解如何创建字符刻度的滑杆，具体内容如代码 15.5 所示，该类的类图如图 15.11 所示。

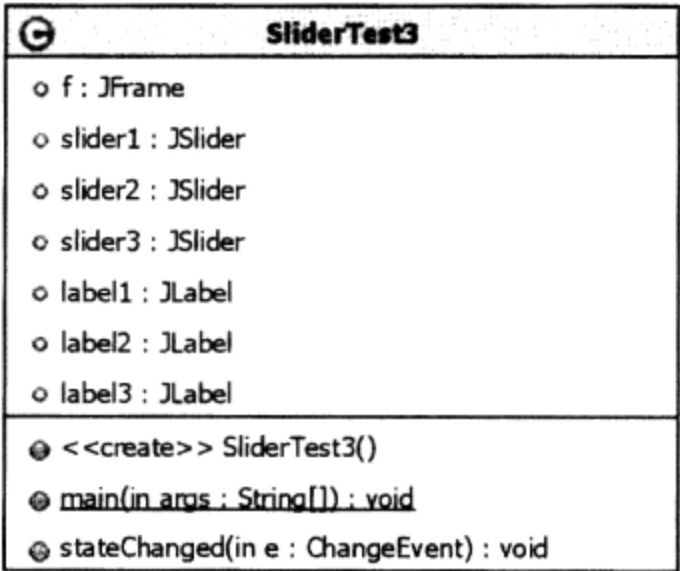


图 15.11 字符刻度滑杆类图

代码 15.5 字符刻度滑杆: SliderTest3.java

```

public class SliderTest3 implements ChangeListener {
    ...
    public SliderTest3() {
        f = new JFrame("滑杆测试"); //创建类 JFrame 对象
        Container contentPane = f.getContentPane(); //获取对象 contentPane
        //创建和设置面板对象 panel1
        JPanel panel1 = new JPanel();
        panel1.setLayout(new GridLayout(2, 1));
        slider1 = new JSlider();
        //设置滑杆对象 slider1 的刻度
        slider1.setPaintTicks(true);
        slider1.setMajorTickSpacing(20);
        slider1.setMinorTickSpacing(10);
        slider1.setPaintLabels(true);
        slider1.setPaintTrack(true);
        slider1.setSnapToTicks(true);
        Hashtable table = new Hashtable(); //创建一个集合对象 table
        //向对象 table 中添加数据
        table.put(new Integer(0), new JLabel("低"));
        table.put(new Integer(50), new JLabel("中"));
        table.put(new Integer(100), new JLabel("高"));
        slider1.setLabelTable(table); //设置刻度的标签
        label1 = new JLabel("目前刻度: " + slider1.getValue());
        panel1.add(label1);
        panel1.add(slider1);
        panel1.setBorder(BorderFactory.createTitledBorder(BorderFactory
            .createEtchedBorder(), "Slider 1", TitledBorder.LEFT,
            TitledBorder.TOP));
        //创建和设置面板对象 panel2
        JPanel panel2 = new JPanel();
        panel2.setLayout(new GridLayout(2, 1));
        slider2 = new JSlider(JSlider.HORIZONTAL);
        slider2.setMinimum(0);
        slider2.setMaximum(100);
        slider2.setValue(30);
        slider2.setExtent(50);
        slider2.setPaintTicks(true);
        slider2.setMajorTickSpacing(10);
        slider2.setMinorTickSpacing(5);
        slider2.setPaintLabels(true);
        slider2.putClientProperty("JSlider.isFilled", Boolean.TRUE);
        table = new Hashtable(); //创建一个集合对象 table
        //向对象 table 中添加数据
        table.put(new Integer(0), new JLabel("弱"));
        table.put(new Integer(25), new JLabel("有点弱"));
        table.put(new Integer(50), new JLabel("中"));
        table.put(new Integer(75), new JLabel("有点强"));
        table.put(new Integer(100), new JLabel("强"));
        slider2.setLabelTable(table); //设置刻度的标签
        label2 = new JLabel("目前刻度: " + slider2.getValue());
        panel2.add(label2);
        panel2.add(slider2);
        panel2.setBorder(BorderFactory.createTitledBorder(BorderFactory

```

```

        .createEtchedBorder(), "Slider 2", TitledBorder.LEFT,
        TitledBorder.TOP));
...
    }
    public static void main(String[] args) {
        new SliderTest3();           //创建 SliderTest3 类对象
    }
    public void stateChanged(ChangeEvent e) { //事件监听器
...
    }
}

```

运行 SliderTest3.java 类, 会出现如图 15.12 所示的界面。

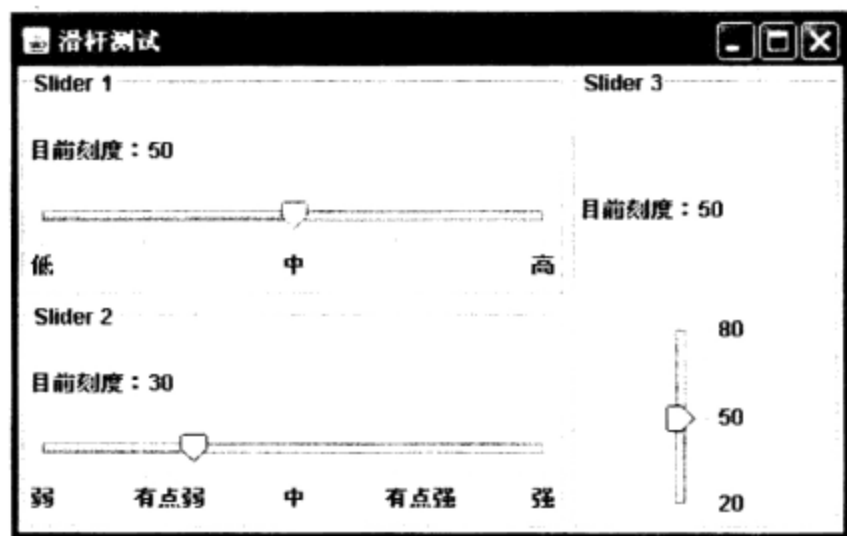


图 15.12 运行结果

【代码解析】

- ❑ 对于滑杆对象 slider1, 通过 setLabelTable (table) 方法设置刻度标签, 该方法的参数 (集合对象 table) 里存在 3 对键/值, 分别为 0/低、50/中和 100/高。
- ❑ 对于滑杆对象 slider2, 通过 setLabelTable (table) 方法设置刻度标签, 该方法的参数 (集合对象 table) 里存在 5 对键/值, 分别为 0/弱、25/有点弱、50/中、75/有点强和 100/强。

通过查看 API 帮助文档可以发现, 如果想设置 JSlider 组件对象刻度的显示标签, 可以通过相应方法来实现, 具体语法如下:

```
void setLabelTable(Dictionary labels)
```

该方法用于实现在给定值处绘制指定的标签。

最后, JSlider 组件还支持通过鼠标和键盘与用户进行交互。如果 JSlider 获得焦点, 左箭头键和右箭头键分别使 JSlider 的游标减少或增加一个刻度。PgDn 键和 PaUp 键分别使 JSlider 的游标减少或增加总刻度范围内的十分之一。Home 键使游标移动到 JSlider 的最小值, End 键使游标移动到 JSlider 的最大值。

15.3.4 JSlider 组件的高级应用

在 Swing 中提供了 JSlider 组件, 允许用户以图形方式在有界区间内通过移动滑块来选择一个整数值。该组件常用于如播放器中的音量设定等领域中, 但是因其提供的 UI 样式

很单调, 根本满足不了用户的审美需求, 所以在具体编程时经常需要自行重构 UI。下面通过一个实例来讲解如何重构组件 JSlider 的 UI, 具体步骤如下。

(1) 自定义滑杆的 UI, 该类的具体内容如代码 15.6 所示, 该类的类图如图 15.13 所示。

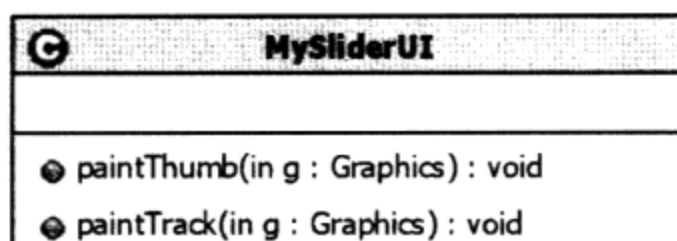


图 15.13 自定义 UI 的类图

代码 15.6 自定义 UI: MySliderUI.java

```

class MySliderUI extends javax.swing.plaf.metal.MetalSliderUI {
    public void paintThumb(Graphics g) {                //绘制指示物方法
        Graphics2D g2d = (Graphics2D) g;              //获取对象 g2d
        //避免绘制时有很多锯齿
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
        //填充椭圆框为当前 thumb 位置
        g2d.fillOval(thumbRect.x, thumbRect.y, thumbRect.width,
            thumbRect.height);
    }
    public void paintTrack(Graphics g) {                //绘制刻度轨迹方法
        int cy, cw;
        Rectangle trackBounds = trackRect;
        if (slider.getOrientation() == JSlider.HORIZONTAL) {
            Graphics2D g2 = (Graphics2D) g;
            cy = (trackBounds.height / 2) - 2;
            cw = trackBounds.width;
            g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON);
            g2.translate(trackBounds.x, trackBounds.y + cy);
            //背景设为灰色
            g2.setPaint(Color.GRAY);
            g2.fillRect(0, -cy, cw, cy * 2);
            int trackLeft = 0;
            int trackRight = 0;
            trackRight = trackRect.width - 1;
            int middleOfThumb = 0;
            int fillLeft = 0;
            int fillRight = 0;
            //坐标换算
            middleOfThumb = thumbRect.x + (thumbRect.width / 2);
            middleOfThumb -= trackRect.x;
            if (!drawInverted()) {
                fillLeft = !slider.isEnabled() ? trackLeft : trackLeft + 1;
                fillRight = middleOfThumb;
            } else {
                fillLeft = middleOfThumb;
                fillRight = !slider.isEnabled() ? trackRight - 1
                    : trackRight - 2;
            }
            //设定渐变
            g2.setPaint(new GradientPaint(0, 0, new Color(0, 100, 100), cw, 0,

```

```

        new Color(0, 255, 100), true));
g2.fillRect(0, -cy, fillRight - fillLeft, cy * 2);
g2.setPaint(slider.getBackground());
Polygon polygon = new Polygon();
polygon.addPoint(0, cy);
polygon.addPoint(0, -cy);
polygon.addPoint(cw, -cy);
g2.fillPolygon(polygon);
polygon.reset();
g2.setPaint(Color.WHITE);
g2.drawLine(0, cy, cw - 1, cy);
g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
    RenderingHints.VALUE_ANTIALIAS_OFF);
g2.translate(-trackBounds.x, -(trackBounds.y + cy));
} else {
    super.paintTrack(g);
}
}
}

```

【代码解析】

在上述代码中存在两个方法，实现绘制指示物的 `paintThumb()` 方法和绘制刻度轨迹的 `paintTrack()` 方法。对于前者，首先把画笔转换成 2D 图形的画笔，然后通过 `SetRenderingHint()` 方法设置图形的边界更光滑，最后通过 `fillOver()` 方法使用椭圆实现填存。对于后者，则主要用来实现绘制刻度轨迹。

(2) 在类 `ReconstructionUI` 中通过调用自定义的 UI 类实现滑杆组件的 UI 重构，具体内容如代码 15.7 所示，该类的类图如图 15.14 所示。

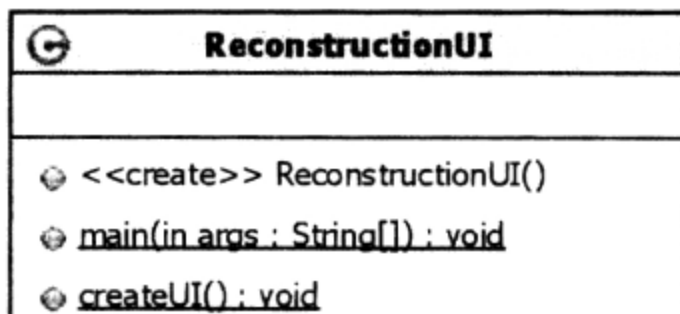


图 15.14 重构滑杆的类图

代码 15.7 重构滑杆的 UI: `ReconstructionUI.java`

```

public class ReconstructionUI extends JPanel {
    public ExampleSlider() {                                //构造函数
        super(new BorderLayout());                          //设定布局器
        ChangeListener listener = new ChangeListener() {   //设定监听器
            public void stateChanged(ChangeEvent e) {
                if (e.getSource() instanceof JSlider) {
                    //输出相应的信息
                    System.out.println("所指示的刻度: "
                        + ((JSlider) e.getSource()).getValue());
                }
            }
        };
        //创建和设置滑杆对象 s1
        JSlider s1 = new JSlider(0, 100, 0);                //创建滑杆对象 s1
    }
}

```

```

s1.setUI(new MySliderUI()); //设置对象 s1 的 UI
//设置对象 s1 的刻度
s1.setMajorTickSpacing(10); //设置主刻度
s1.setMinorTickSpacing(5); //设置次刻度
//显示刻度
s1.setPaintTicks(true);
s1.setPaintLabels(true);
s1.addChangeListener(listener); //添加事件监听器
//创建和设置滑杆对象 s2
JSlider s2 = new JSlider(0, 100, 0);
//设置对象 s2 的 UI
s2.setUI(new javax.swing.plaf.metal.MetalSliderUI() {
    protected void paintHorizontalLabel(Graphics g, int v, Component l) {
        JLabel lbl = (JLabel) l;
        lbl.setForeground(Color.green);
        super.paintHorizontalLabel(g, v, lbl);
    }
});
s2.setForeground(Color.BLUE); //设置颜色
s2.setMajorTickSpacing(10);
s2.setMinorTickSpacing(5);
s2.setPaintTicks(true);
s2.setPaintLabels(true);
s2.addChangeListener(listener); //添加事件监听器
//创建和设置盒式容器
Box box = Box.createVerticalBox(); //创建 Box 对象
box.add(Box.createVerticalStrut(5));
box.add(s1); //添加对象 s1 到对象 box 里
box.add(Box.createVerticalStrut(5));
box.add(s2); //添加对象 s2 到对象 box 里
box.add(Box.createVerticalGlue());
add(box, BorderLayout.CENTER);
add(Box.createHorizontalStrut(5), BorderLayout.WEST);
add(Box.createHorizontalStrut(5), BorderLayout.EAST);
setPreferredSize(new Dimension(240, 100)); //设定窗体大小
}
public static void main(String[] args) { //主方法
    EventQueue.invokeLater(new Runnable() { //重构 UI
        public void run() {
            createUI(); //调用 createUI() 方法
        }
    });
}
public static void createUI() { //创建 UI 方法
    JFrame frame = new JFrame("自定义滑杆 UI"); //创建主窗口
    //设置关闭方法
    frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    //添加 ExampleSlider 类对象到窗口里
    frame.getContentPane().add(new ReconstructionUI ());
    frame.setResizable(false); //设置不能够修改大小
    frame.pack();
    frame.setLocationRelativeTo(null);
    frame.setVisible(true); //显示窗口
}
}

```


运行 MySliderUI.java 类，会出现如图 15.15 所示的界面。

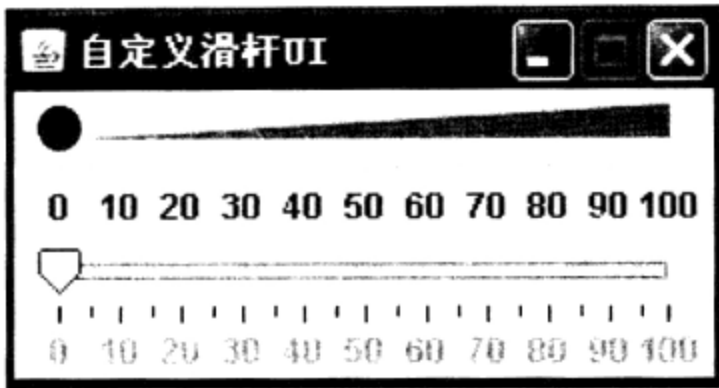


图 15.15 运行结果

【代码解析】

- ❑ 在构造函数中，首先创建了两个 JSlider 对象，然后设置这两个组件的基本属性并通过 setUI()方法设置这两个对象的 UI 为 MySliderUI 类对象，最后再把两个滑杆对象添加到盒式容器对象里。
- ❑ 在重新设置组件的 UI 时，必须通过多线程来实现，于是创建了一个界面的方法 createUI()，在该方法中实现了主界面，同时在主方法中通过反射来调用该方法。

15.3.5 Swing 中的多线程

通过查看 API 帮助文档可以发现，如果在 Swing 控件显示后需要对该控件外观进行修改，则必须使用多线程，这是因为 Swing 组件是不支持线程安全的。除了上述情况外，图形用户界面程序中的响应按钮、文本框、菜单等事件的编写，也必须要使用多线程，可以说，多线程是图形用户界面程序的本质特征。

本节将通过一个具体的实例，来讲解在 Swing 图形用户界面中如何实现多线程机制，该实例主要实现这样的功能，即 Swing 图形用户界面程序在启动后，首先会访问网络，然后将网络返回的信息在自己界面中显示出来。为了避免用户在访问网络的过程中关闭该 Swing 图形用户界面程序，在访问网络的过程中，也会在该程序自己的界面中显示出相应的信息。具体步骤如下。

(1) 利用以前学过的多线程知识，可以很容易地实现上述功能，实现上述功能的类的具体内容如代码 15.8 所示，该类的类图如图 15.16 所示。

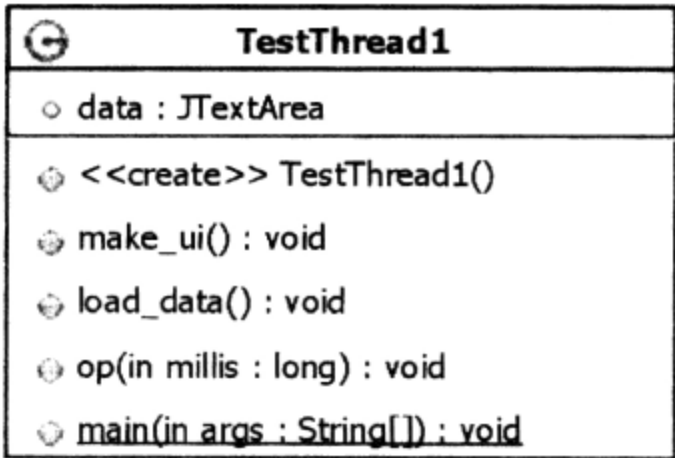


图 15.16 TestThread1 的类图

代码 15.8 测试多线程: TestThread1.java

```

public class TestThread1 {
    JTextArea data;                                //用来显示信息的组件对象 data
    TestThread1() {
        Thread t_ui = new Thread() {               //初始化界面的线程
            public void run() {
                make_ui();                           //调用 make_ui() 方法
            }
        };
        Thread t_load = new Thread() {              //获取和显示返回信息的线程
            public void run() {
                load_data();                          //调用 load_data() 方法
            }
        };
        //启动线程
        t_ui.start();
        t_load.start();
    }
    void make_ui() {                                //初始化界面的方法
        //为组件对象 data 赋值
        data = new JTextArea("正在获取数据...");
        //创建窗口对象 frame
        JFrame frame = new JFrame("实现多线程机制");
        frame.setContentPane(data);                  //添加对象 data 到窗口里
        //设置关闭功能
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //设置窗口的大小及显示
        frame.setSize(300, 200);
        frame.setVisible(true);
    }
    void load_data() {                              //设置返回的信息
        //通过使当前线程休眠 3 秒时间来模拟访问网络过程
        op(3000);                                    //调用 op() 方法
        data.setText("这是用户需要的数据");          //显示返回的信息
    }
    void op(long millis) {                          //线程休眠的方法
        try {
            Thread.sleep(millis);                    //使线程休眠
        } catch (Exception exc) {
        }
    }
    //主方法
    public static void main(String[] args) throws Exception {
        //修改组件的 UI
        String lnf = UIManager.getCrossPlatformLookAndFeelClassName();
        UIManager.setLookAndFeel(lnf);
        JFrame.setDefaultLookAndFeelDecorated(true);
        new TestThread1();                          //创建类 TestThread1 对象
    }
}

```

运行 TestThread1.java 类, 过程如图 15.17 所示, 达到了预期的效果。

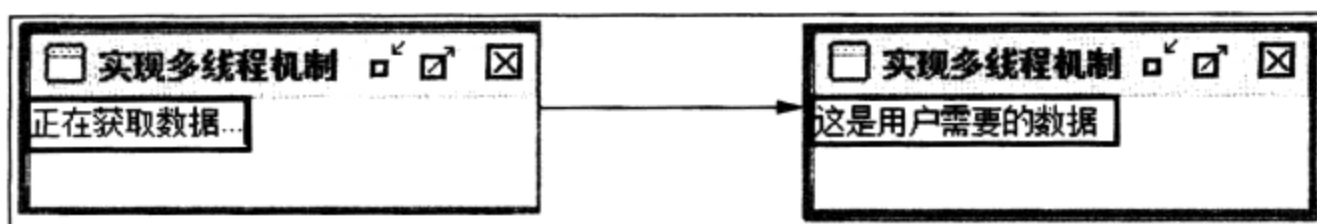


图 15.17 运行结果

【代码解析】

在上述代码中，分别创建了两个线程对象 `t_ui` 和 `t_load`，线程对象 `t_ui` 用来实现初始化界面过程；而线程对象 `t_load` 用来模拟获取和显示网络数据。为了使程序的运行效果达到预期的效果，在线程对象 `t_load` 中首先通过调用 `op()` 方法使线程休眠一段时间，然后才修改组件对象 `data` 的内容。

(2) 类 `TestThread1` 是不稳定的，即随着该类运行次数的增加会出现意想不到的错误，例如图 15.18 所示的错误。

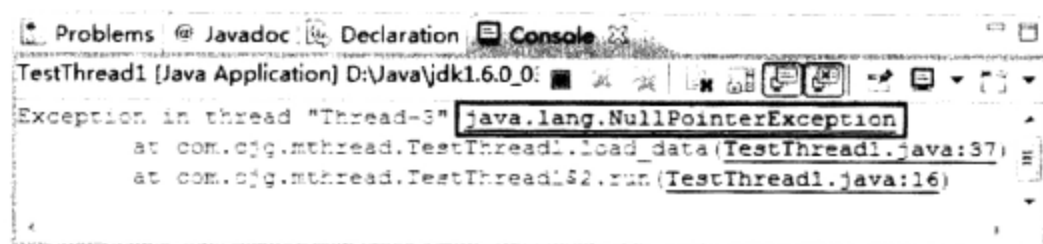


图 15.18 错误提示

为了模拟该错误，在原来的功能中增加一个新的需求，即在访问网络资源之前需要确定用户的权限。为了模拟该功能，只需在初始化界面的方法里增加语句 `op()` 方法。修改后的具体内容如代码 15.9 所示，该类的类图如图 15.19 所示。

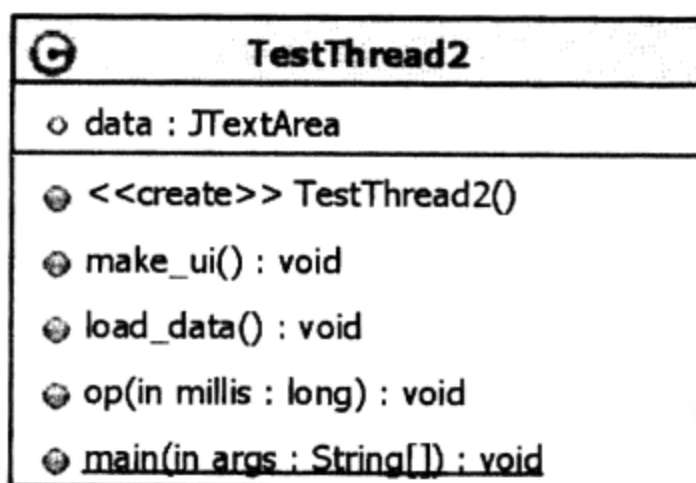


图 15.19 TestThread2 的类图

代码 15.9 测试线程: TestThread2.java

```
public class TestThread2 {
    JTextArea data; //用来显示信息的组件对象 data
    TestThread1() {
        ...
    };
    //启动线程
    t_ui.start();
    t_load.start();
}
```

```
}
void make_ui() { //初始化界面的方法
    //通过使当前线程休眠 4 秒时间来模拟验证用户权限功能
    op(4000); //调用 op() 方法
    //为组件对象 data 赋值
    data = new JTextArea("正在获取数据...");
    //创建窗口对象 frame
    JFrame frame = new JFrame("实现多线程机制");
    frame.setContentPane(data); //添加对象 data 到窗口里
    //设置关闭功能
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    //设置窗口的大小及显示
    frame.setSize(300, 200);
    frame.setVisible(true);
}
void load_data() { //设置返回的信息
    //通过使当前线程休眠 3 秒时间来模拟访问网络过程
    op(3000); //调用 op() 方法
    data.setText("这是用户需要的数据"); //显示返回的信息
}
void op(long millis) { //线程休眠的方法
    //省略部分代码
...
}
//主方法
public static void main(String[] args) throws Exception {
    //省略部分代码
...
}
```

运行 TestThread2.java 类，会出现如图 15.20 所示的错误。这是因为 make_ui()方法执行时间延长了，当得到网络返回的数据后需要显示时，用来显示信息的对象 data 还未完成初始化。

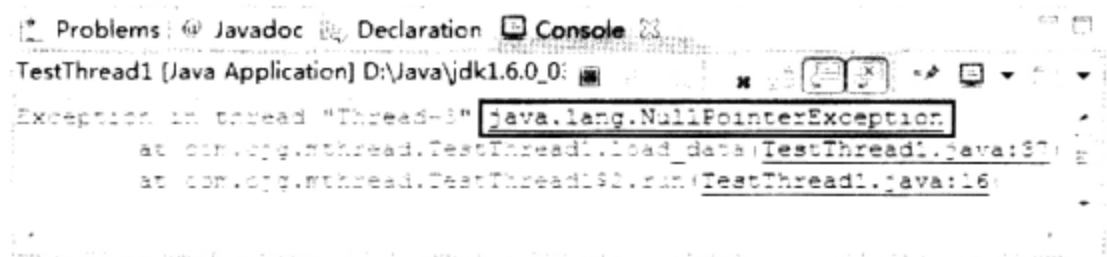


图 15.20 运行结果

(3) 为了避免出现前面意想不到的错误，可以修改 TestThread3 类的内容如代码 15.10 所示，该类的类图如图 15.21 所示。

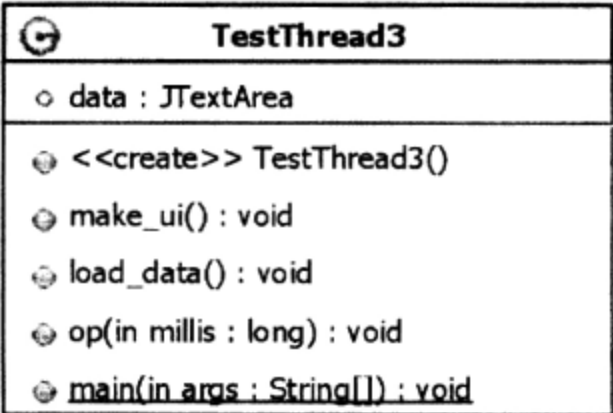


图 15.21 TestThread3 的类图

代码 15.10 测试线程: TestThread3.java

```

public class TestThread3 {
    JTextArea data;
    TestThread3() {
        Thread t_ui = new Thread() {                //创建对象 t_ui
            public void run() {
                make_ui();
            }
        };
        Thread t_load = new Thread() {              //创建对象 t_load
            public void run() {
                load_data();
            }
        };
        SwingUtilities.invokeLater(t_ui);            //实现异步执行
        t_load.start();
    }
    void make_ui() {                                  //初始化界面
        //通过使当前线程休眠 4 秒时间来模拟验证用户权限功能
        op(4000);                                     //调用 op() 方法
        //为组件对象 data 赋值
        data = new JTextArea("正在获取数据...");
        //创建窗口对象 frame
        JFrame frame = new JFrame("实现多线程机制");
        frame.setContentPane(data);                   //添加对象 data 到窗口里
        //设置关闭功能
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //设置窗口的大小及显示
        frame.setSize(300, 200);
        frame.setVisible(true);
    }
    void load_data() {
        op(3000);
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                data.setText("这是用户需要的数据");
            }
        });
    }
    void op(long millis) {
        ...
    }
    public static void main(String[] args) throws Exception {
        ...
    }
}

```

运行 TestThread3.java 类, 过程如图 15.22 所示, 达到了预期的效果。

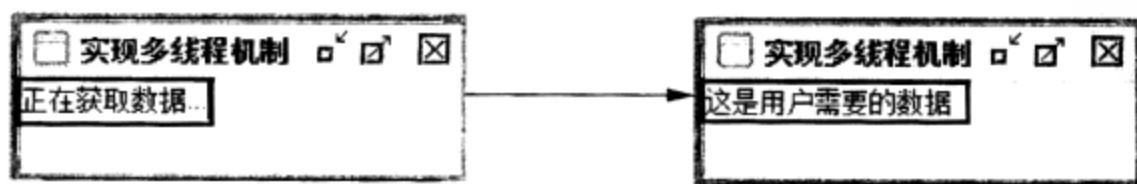


图 15.22 运行过程

【代码解析】

- ❑ 对于 Swing 组件，如果想修改后显示组件的外观，则必须使用如下的形式。所以需要在 load_data()方法中对修改组件 data 内容的语句进行修改。

```
SwingUtilities.invokeLater(new Runnable() {
    public void run() {
        //修改 GUI 控件的外观
    }
});
```

- ❑ 为了实现用户界面初始化后，才能修改用户界面中的组件 data 的内容，需要把与用户界面相关的指令序列线程（t_ui）在 AWT 事件线程上实现异步执行，即修改代码为：

```
SwingUtilities.invokeLater(t_ui);
```

15.3.6 Timer 组件的基础知识

通过查看 API 帮助文档可以发现，除了在包 java.util 中存在一个 Timer 类外，在包 javax.swing 中也存在一个 Timer 类。其实在 swing 包中的 Timer 类可以看作是 Swing 中的一个组件。作为 Swing 中的计时器组件，Timer 组件与其他组件有些不一样，即其没有直观外观，但是其能按相当的时间间隔来执行事件。由于 Timer 组件的上述特性，其经常用来实现组件内容的更新。

下面通过显示当前时间的类，来详细讲解如何使用 Timer 组件，该类的具体内容如代码 15.11 所示，该类的类图如图 15.23 所示。

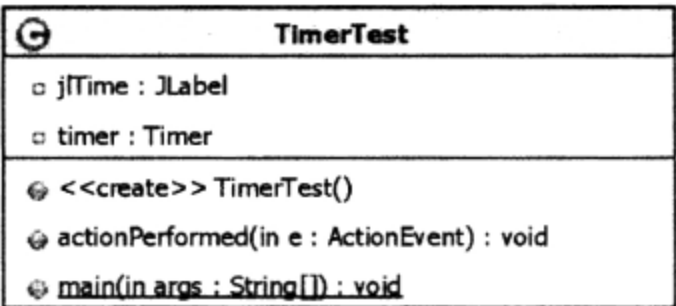


图 15.23 TimerTest 的类图

代码 15.11 显示当前时间：TimerTest.java

```
public class TimerTest extends JFrame implements ActionListener {
    //创建成员变量
    private JLabel jlTime = new JLabel(); //显示时间的标签对象
    private Timer timer; //计时器对象 timer
    public TimerTest() { //构造函数
        setTitle("Swing 中 Timer 组件的测试"); //设置窗口标题
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //实现关闭功能
        //设置窗口大小
        setSize(180, 80);
        add(jlTime); //添加对象 jlTime 到窗口
        //创建和设置计时器组件对象 timer 定时器
        timer = new Timer(500, this);
    }
}
```



```

        timer.start(); //启动计时器组件
        setVisible(true);
    }
    //计算器执行的任务
    @Override
    public void actionPerformed(ActionEvent e) {
        //创建时间格式
        DateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        Date date = new Date(); //获取当前的时间
        j1Time.setText(format.format(date)); //更改标签对象的内容
    }
    public static void main(String[] args) { //主方法
        new TimerTest(); //创建类 TimerTest 对象
    }
}

```

运行 TimerTest.java 类, 过程如图 15.24 所示。

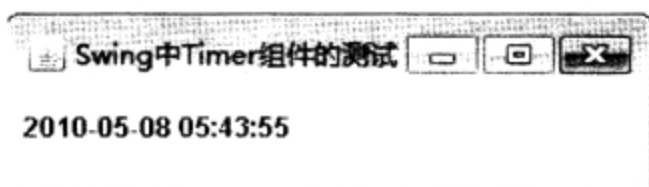


图 15.24 运行结果

【代码解析】

在上述代码中, 计时器组件对象 Timer 会每隔 0.5 秒通知 ActionListener 监听器, 即每隔 0.5 秒, 执行监听器 actionPerformed()方法里的代码。

最后, 查看 API 帮助文档可以发现 Timer 组件使用的一段代码, 具体内容如下:

```

Int delay=1000;
ActionListener taskPerformer = new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
    }
};
new Timer(delay, taskPerformer).start();

```

在上述代码中, 为了创建计时器组件对象, 需要传入两个参数, delay 为计时器对象的间隔时间; taskPerformer 为计划任务, 其为接收计时器操作事件的侦听器, 必须为 ActionListener 类型。

15.3.7 Timer 组件的应用

通过前面内容的讲解可以知道, 定时器类 javax.swing.Timer 就是一个按预定频率触发 ActionEvent 事件的源组件。更准确地说, 当调用定时器组件的 start()方法时, 对象将调用监听器的 actionPerformed()方法, 即监听器只能是 ActionListener 的实例或者是实现了 ActionListener 接口的组件对象, 例如 JPanel 对象。

下面将讲解一个 Timer 组件的具体应用——定时按不同顺序显示图片, 具体步骤如下。

(1) 创建自定义 JPanel 类, 其实现根据单击不同按钮以不同的方式不断地显示图片, 该类的具体内容如代码 15.12 所示, 该类与其内部类的 UML 如图 15.25、图 15.26 和图 15.27 所示。

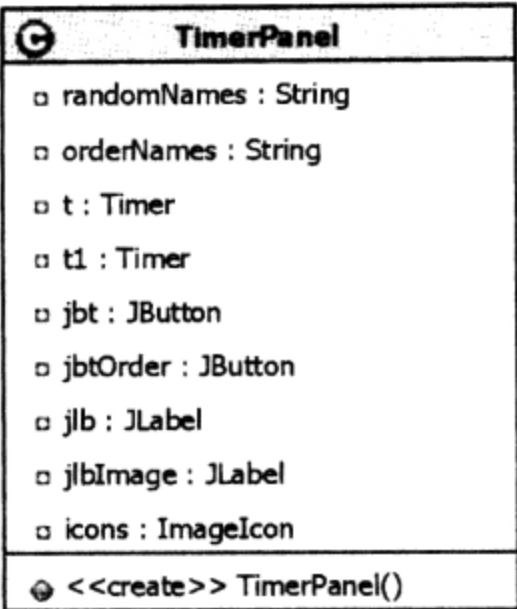


图 15.25 TimerPanel 类图

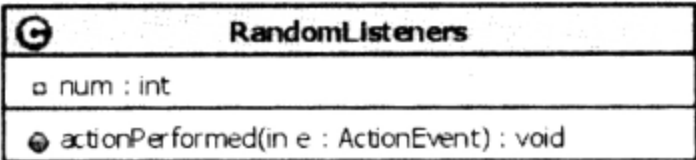


图 15.26 顺序按钮类图

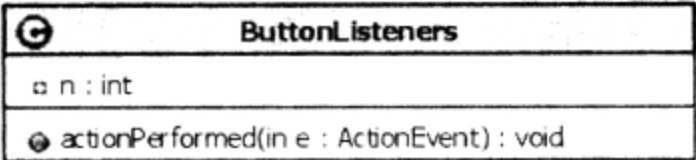


图 15.27 随机按钮类图

代码 15.12 自定义面板类: TimerPanel.java

```
class TimerPanel extends JPanel {
    //创建成员变量
    //图片的名字数组
    private String[] randomNames = { "p/Randol5.jpg", "p/Random2.jpg", "p/
    Random3.jpg",
        "p/Random4.jpg", "p/Random5.jpg" };
    private String[] orderNames = { "p/Order1.jpg", "p/Order2.jpg", "p/
    Order3.jpg",
        "p/Order4.jpg", "p/Order5.jpg", "p/Order6.jpg" };
    //创建两个计时器对象
    private Timer t;
    private Timer t1;
    //创建两个按钮对象
    private JButton jbt;
    private JButton jbtOrder;
    //创建两个标签对象
    private JLabel jlb = new JLabel();
    private JLabel jlbImage = new JLabel();
    private ImageIcon icons;
    public ViewerPanel() {
        //设置对象 jlb 对象
        jlb.setText("Pictures");
        //设置字体
        jlb.setFont(new Font("SansSerif", Font.BOLD, 26));
        jlb.setHorizontalAlignment(JLabel.CENTER);
        //设置组件对象 jlbImage 的图片
        jlbImage.setIcon(new ImageIcon("p/Order2.jpg"));
        //创建和设置对象 p3
    }
}
```

//创建 ImageIcon 对象
//构造函数

//设置内容

```

jbt = new JButton("Random");           //为按钮对象 jbt 赋值
jbtOrder = new JButton("Order");       //为按钮对象 jbtOrder 赋值
JPanel p3 = new JPanel();              //创建标签对象 p3
p3.setLayout(new FlowLayout());        //设置对象 p3 的布局管理器
//添加对象 jbt 和 jbtOrder 到对象 p3 里
p3.add(jbt);
p3.add(jbtOrder);
//设置类 ViewerPanel
setLayout(new BorderLayout());         //设置布局管理器
//添加对象 jlb、jlbImage 和 p3 到类 ViewerPanel 界面里
add(jlbImage, BorderLayout.CENTER);
add(jlb, BorderLayout.SOUTH);
add(p3, BorderLayout.NORTH);
//创建监听器的类对象
ButtonListeners listener = new ButtonListeners();
RandomListeners randomListener = new RandomListeners();
jbtOrder.addActionListener(listener); //为组件对象 jbtOrder 添加事件监听器
jbt.addActionListener(randomListener); //为组件对象 jbtOrder 添加事件监听器

//为计时器对象 t 和 t1 赋值
t = new Timer(1000, listener);
t1 = new Timer(1000, randomListener);
}

private class RandomListeners implements ActionListener {
    //顺序按钮的监听器
    //创建变量 num
    private int num = 0;
    public void actionPerformed(ActionEvent e) {
        //启动和停止相应的计时器对象
        t.start();
        t1.stop();
        num++; //实现变量 num 的自增
        if (num >= 5) {
            num = 0;
        }
        //设置组件 jlbImage 所显示的图片
        jlbImage.setIcon(new ImageIcon(randomNames[num]));
    }
}

private class ButtonListeners implements ActionListener {
    //随机按钮的监听器
    //创建变量 n
    private int n;
    public void actionPerformed(ActionEvent e) {
        //启动和停止相应的计算器对象
        t1.start();
        t.stop();
        n = (int) (Math.random() * 6); //获取 1~6 范围中的随机数
        //设置组件 jlbImage 所显示的图片
        jlbImage.setIcon(new ImageIcon(orderNames[n]));
    }
}
}

```

【代码解析】

- 上述代码实现了控制动画速度项目界面，该用户界面涉及的具体容器、对象和布局如图 15.28 所示。
- 对于按钮对象 jbt，当发生单击事件后，就会调用监听器 ButtonListeners 中的 actionPerformed()方法。在该方法中会重新设置组件对象 jlblImage 的内容，即以随机产生一个数字作为下标数组元素对应的图片。
- 对于按钮对象 jbtorder，当发生单击事件后，会调用监听器 RandomListeners 中的 actionPerformed()方法。在该方法中重新设置组件对象 jlblImage 的内容，即当前图片的下一张图片。
- 为了能够实现单击按钮后图片会不断变化的效果，创建了两个以监听器 ButtonListeners 和 RandomListeners 为参数的计时器对象 t 和 t1。

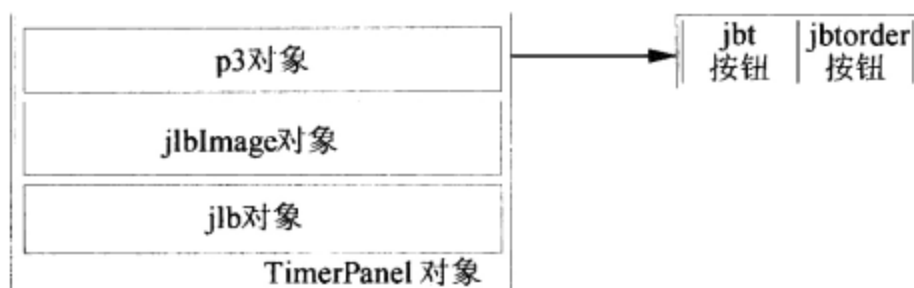


图 15.28 布局

(2) 创建一个测试类 TimerFrame，查看 TimerPanel 类是否达到预期效果，该类的具体内容如代码 15.13 所示。

代码 15.13 测试类：TimerFrame.java

```

class TimerFrame extends JFrame {
    //创建窗口宽度和高度的变量
    private final int WIDTHS = 500;
    private final int HEIGHTS = 750;
    TimerFrame() {
        setTitle("Timer 应用"); //构造函数
        setSize(WIDTHS, HEIGHTS); //设置标题
        TimerPanel imagePanel = new TimerPanel(); //设置窗口的宽度和高度
        setLayout(new BorderLayout()); //创建 TimerPanel 对象
        add(imagePanel); //设置布局管理器
        pack(); //添加对象 imagePanel 到窗口
    }
    public static void main(String[] args) { //主方法
        TimerFrame frames = new TimerFrame(); //创建 TimerFrame 类对象
        //设置关闭功能
        frames.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frames.setVisible(true); //显示窗口
    }
}
  
```

运行 TimerFrame.java 类，会出现如图 15.29 所示的过程。

【代码解析】

上述代码非常简单，即首先创建 TimerPanel 类对象，然后把该对象添加到窗口对象

TimerFrame 中，最后在将窗口对象中显示出来。

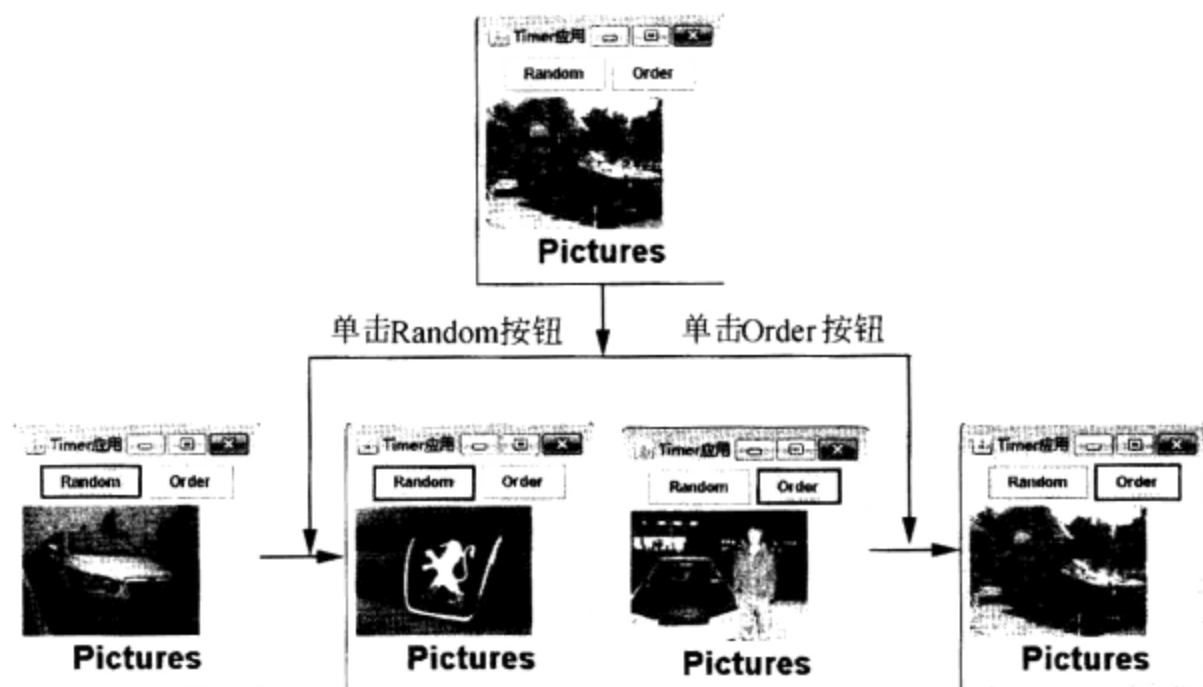


图 15.29 运行结果

15.4 小 结

本章主要通过 Java 语言中的 Swing 滑杆组件 JSlider, 以及计算器组件 Timer 实现控制显示图片的速度, 从而达到动画的效果。在具体实现该项目时, 主要需监听 JSlider 组件的状态变化。在本章最后的扩展部分, 不仅详细介绍了组件 JSlider 和 Timer 的基础知识, 而且还讲解了修改 Swing 组件 UI 的知识。

第 16 章 记事本（对话框组件）

在 Swing 组件中存在许多对话框组件，例如打开文件对话框组件、保存文件对话框等，本章将通过 Swing 组件的对话框组件和菜单组件来模拟 Windows 系统中的记事本。对于一个项目来说，有好的图形用户界面固然重要，但是各种组件的相互操作更重要，因为其相当于该项目的逻辑功能。

本章的学习目标如下：

- ❑ 掌握组件和面板的使用方法；
- ❑ 掌握日记簿项目；
- ❑ 熟练掌握各种对话框的使用。

16.1 记事本原理

“记事本”项目用来模拟计算机中的记事本（notepad）功能，在具体运行该项目时，其不仅将模拟 notepad.exe 的界面，还将模拟该程序的所有菜单功能。

16.1.1 项目结构框架分析

对于记事本项目，根据面向对象的思想，需要创建一个对象——记事本。记事本项目目录如图 16.1 所示，该项目中存在 3 个类，Notepad 类用来实现记事本界面；FontChooserDialog 类用来实现字体选择对话框；MyFileFilter 类用来实现文件类型过滤。

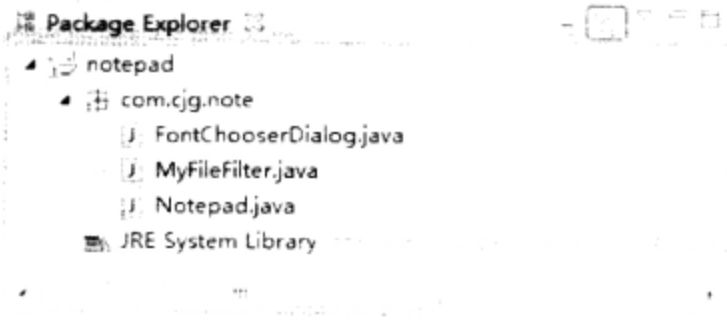


图 16.1 项目目录

16.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括记事本项目的初始化界面和各种菜单选项要实现的功能。

1. 记事本项目的初始化界面

当运行 Notepad 类后，会出现如图 16.2 所示的初始界面。在该界面工具栏里存在 6 个菜单，而主体部分为输入区。

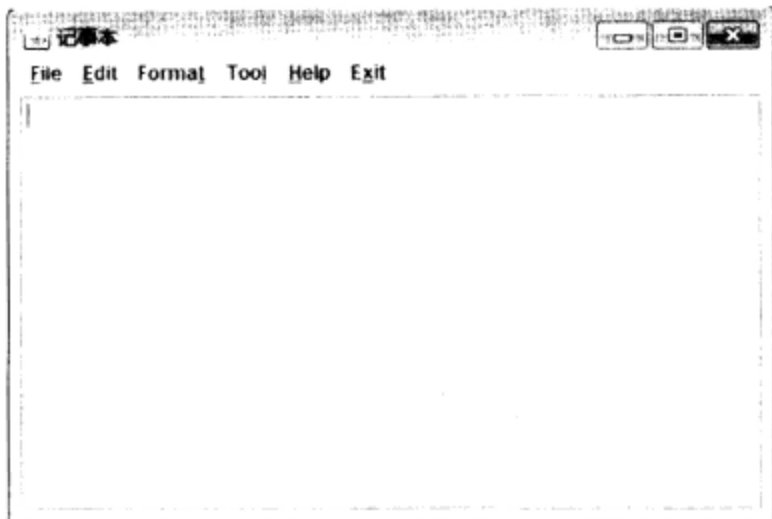


图 16.2 记事本的初始界面

2. File 菜单实现的功能

在记事本项目的初始界面中，如果想实现文件新建、保存和打开等功能，可以选择 File 菜单下的菜单选项，各选项的作用如下。

- ☐ New 菜单选项：通过选择如图 16.3 所示的菜单选项，会新建一个文件。
- ☐ Open 菜单选项：通过选择该菜单选项，如图 16.4 所示，会打开“打开”文件对话框。
- ☐ Save 菜单选项：通过选择该菜单选项如图 16.5 所示，会打开“保存”文件对话框。



图 16.3 新建菜单项



图 16.4 打开菜单项



图 16.5 保存菜单项

新建功能：如果想新建一个文件，可以选择菜单 File | New 命令，这时就会创建一个新的文件，具体过程如图 16.6 所示。

保存功能：如果想把当前输入区的内容保存到文件里，首先选择菜单 File | Save 命令打开“保存”文件对话框。然后在该对话框中选择相应的保存目录，填写相应的文件名并单击“保存”按钮，这时相应的目录（c:/save）里就会创建出新的文件（save.txt），具体过程如图 16.7 所示。

打开功能：如果想打开文件并把该文件的内容显示在输入区，首先选择菜单 File | Open 命令，打开“打开”文件对话框。然后在该对话框中选择文件存放的目录，选择该文件并单击“打开”按钮，这时相应目录（c:/save）里文件（save.txt）的内容就会显示在输入区，具体过程如图 16.8 所示。



图 16.6 新建功能

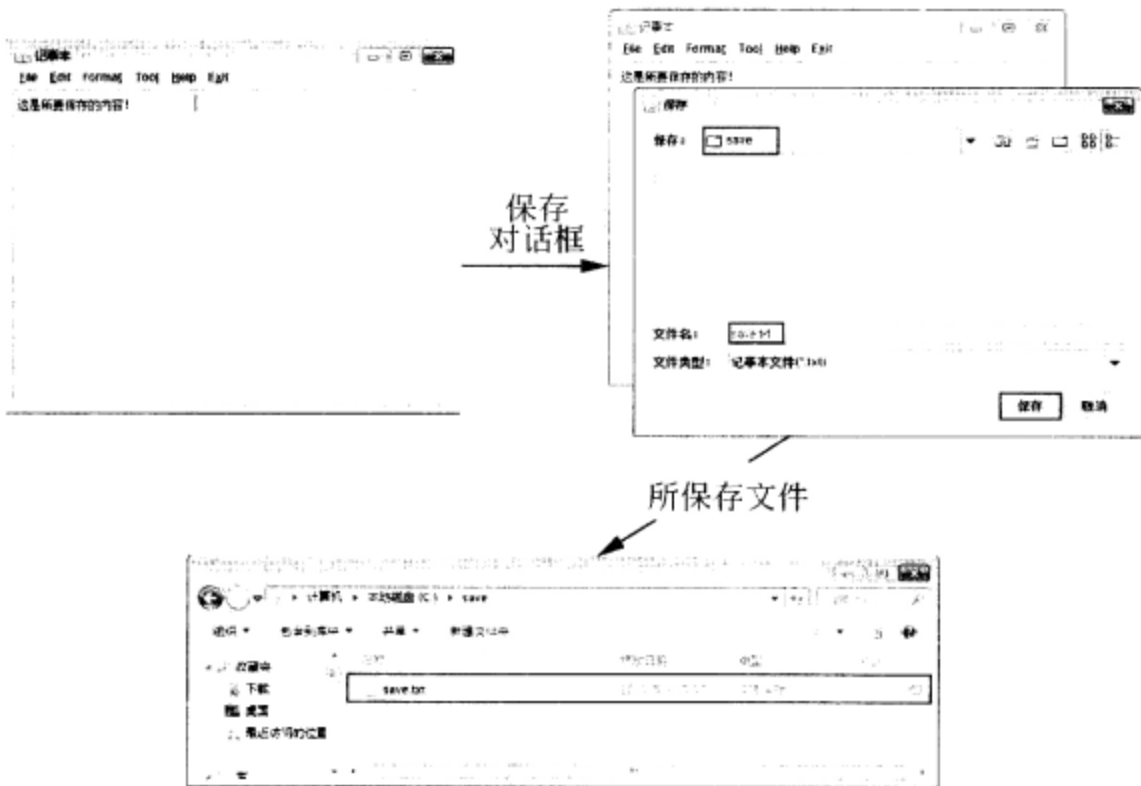


图 16.7 保存文件功能



图 16.8 打开文件功能

3. Edit菜单实现的功能

在记事本项目的初始界面中，可以在输入区输入任何内容。如果想实现一些编辑方面的操作，可以选择 Edit 菜单下的菜单选项，各选项的作用如下。

- ❑ Cut 菜单选项：该菜单选项会实现“剪切”功能，其快捷键为 Ctrl+X，如图 16.9 所示。
- ❑ Copy 菜单选项：该菜单选项会实现“复制”功能，其快捷键为 Ctrl+C，如图 16.10 所示。
- ❑ Plaster 菜单选项：该菜单选项会实现“粘贴”功能，其快捷键为 Ctrl+V，如图 16.11

所示。



图 16.9 Cut 菜单项



图 16.10 Copy 菜单项



图 16.11 Plaster 菜单项

剪切和粘贴功能：首先用鼠标选择“这是剪切的内容！”内容，然后通过快捷键 Ctrl+X 剪切所选内容，最后再通过快捷键 Ctrl+V 粘贴所剪切的内容，具体过程如图 16.12 所示。



图 16.12 剪切和粘贴过程

复制和粘贴功能：首先用鼠标选择“这是复制的内容！”内容，然后通过快捷键 Ctrl+C 复制所选内容，最后再通过快捷键 Ctrl+V 粘贴所复制的内容，具体过程如图 16.13 所示。



图 16.13 复制和粘贴过程

全选功能：可以通过快捷键 Ctrl+A 中输入区的所有内容，具体过程如图 16.14 所示。



图 16.14 全选功能

4. Format菜单实现的功能

在记事本项目的初始界面中，可以在输入区输入任何内容。这时如果想实现格式操作，可以选择 Format 菜单下的菜单选项，各菜单选项的作用如下。

- ❑ Font 菜单选项：选择该菜单选项会打开“字体选择对话框”，如图 16.15 所示。
- ❑ Color 菜单选项：选择该菜单选项会打开“颜色选择对话框”，如图 16.16 所示。

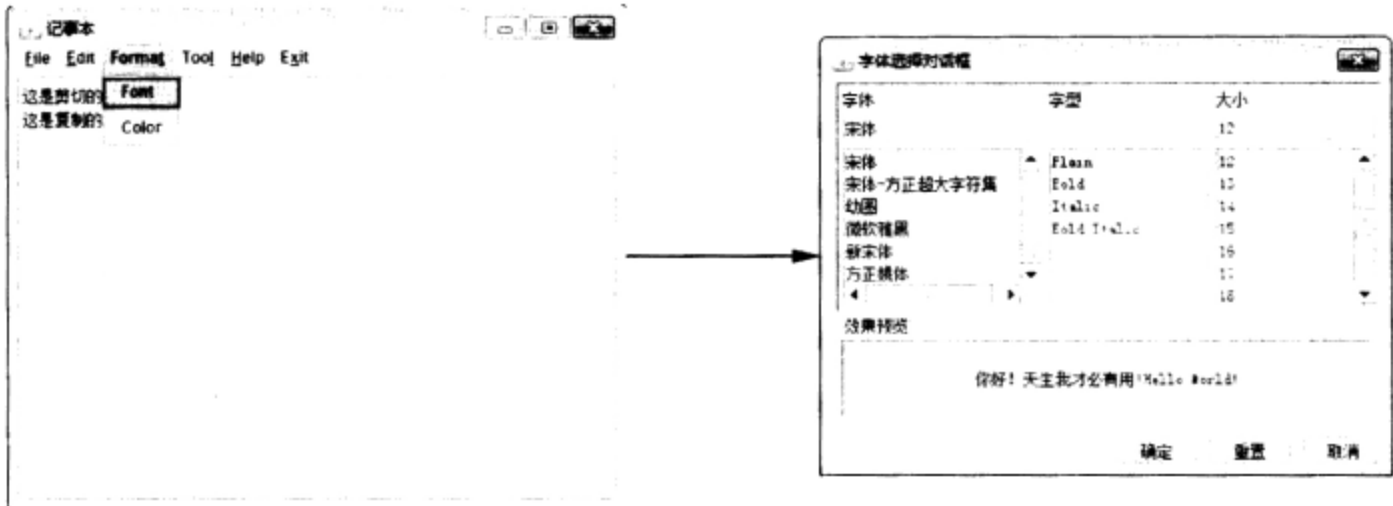


图 16.15 字体对话框

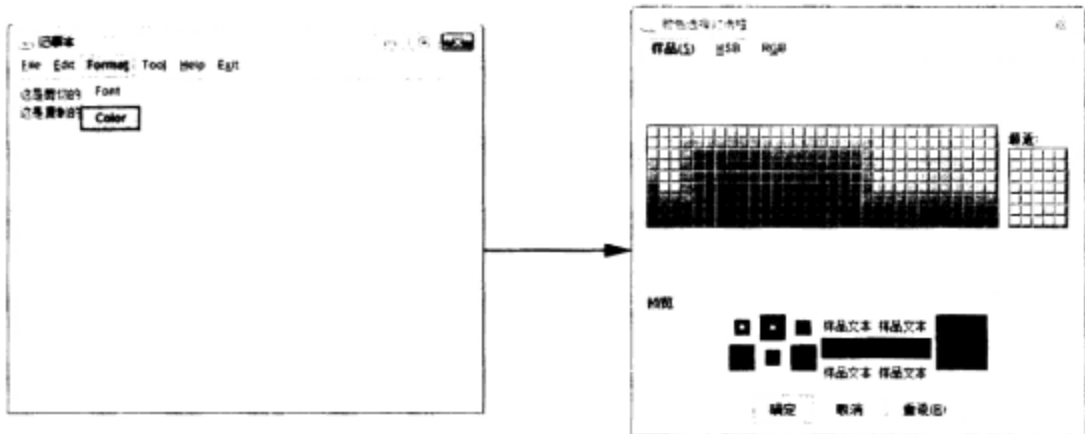


图 16.16 颜色对话框

设置字体功能：首先选择菜单 Format | Font 命令，打开“字体选择对话框”。然后在该对话框中选择相应的字体格式并单击“确定”按钮，这时记事本输入区的字体会发生变化，具体过程如图 16.17 所示。



图 16.17 字体设置过程

设置颜色功能：首先选择菜单 Format | Color 命令，打开“颜色选择对话框”。然后在该对话框中选择相应的颜色并单击“确定”按钮，这时记事本输入区的内容就会发生颜色上的变化，具体过程如图 16.18 所示。

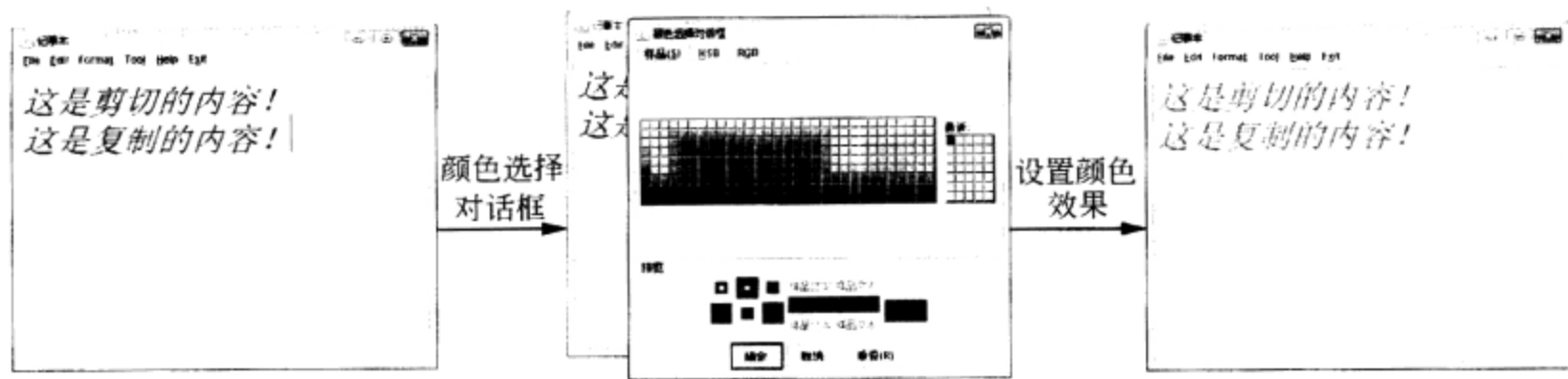


图 16.18 颜色设置过程

5. Tool菜单实现的功能

在具体运行记事本项目时，如果想运行 Notepad 程序，可以选择 Tool | MS Notepad 命令，具体过程如图 16.19 所示；如果想运行 Calculator 程序，可以选择 Tool | MS Calculator 命令，具体过程如图 16.20 所示。

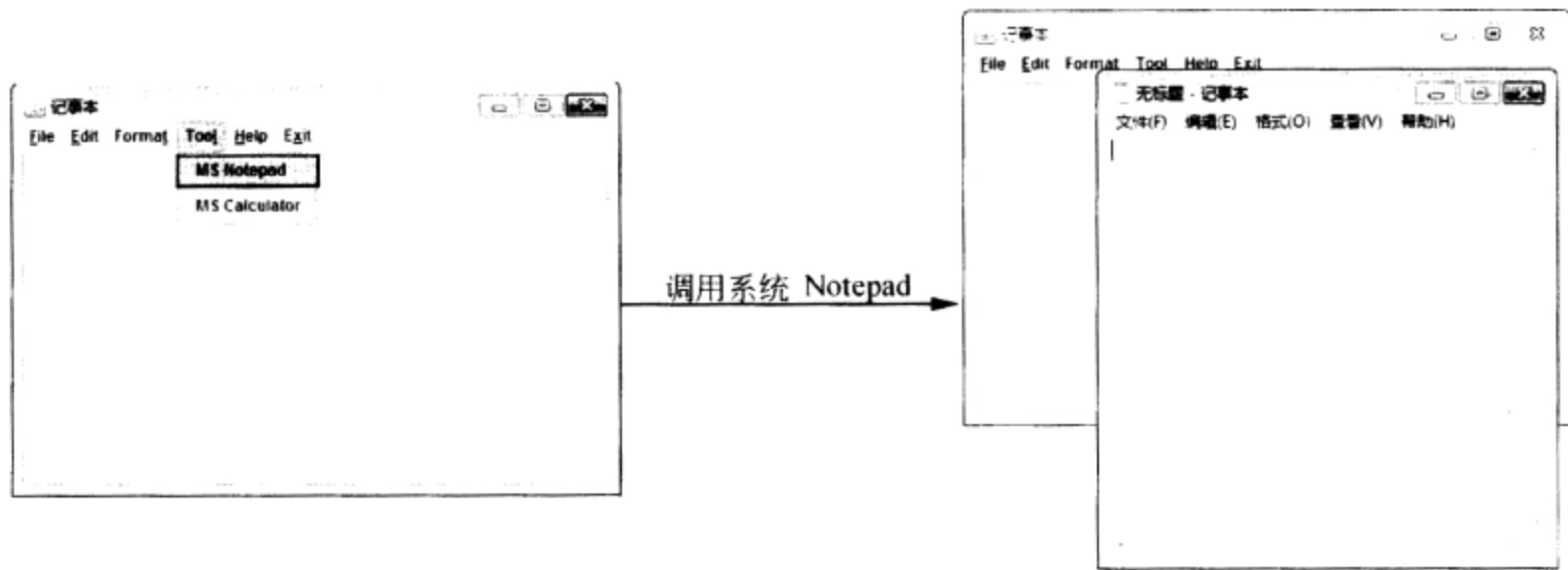


图 16.19 调用 Notepad 程序

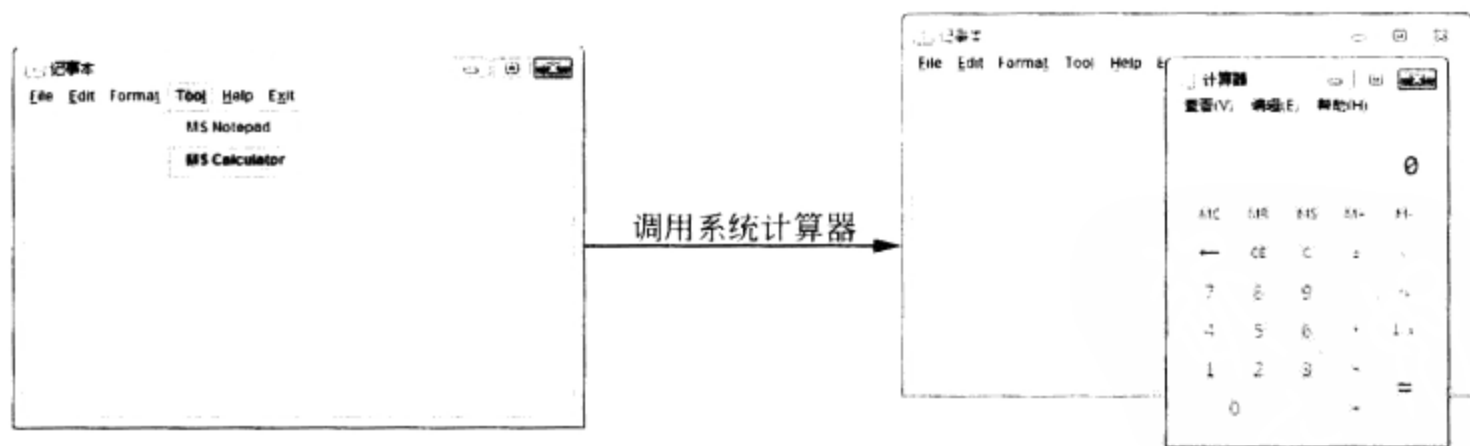


图 16.20 调用 Calculator 程序

6. Help菜单和Exit菜单实现的功能

在具体运行记事本项目时，当选择 Help | About 菜单命令时，就会弹出 About 对话框，具体过程如图 16.21 所示。如果想退出记事本程序，可以选择 Exit 菜单，具体过程如图 16.22 所示。



图 16.21 打开 About 对话框



图 16.22 退出系统

16.2 记事本的实现过程

为了更好地模拟记事本（notepad）功能，该项目不仅实现各个菜单的功能，还会通过布局管理器模拟记事本（notepad）的界面。该项目存在一个实现主要功能的类 Notepad，该类的类图如图 16.23 所示。



图 16.23 Notepad 的类图

16.2.1 实现记事本的界面

Notepad.java 类为记事本类，在该类中通过布局管理器来统一管理各个组件的大小和位置，该类的具体内容如代码 16.1 所示。

代码 16.1 记事本的界面：Notepad.java

```

public class Notepad extends JFrame implements ActionListener {
    //创建成员变量
    File file = null; //文件对象变量 file
    Color color = Color.black; //颜色对象变量 color
    JTextPane text = new JTextPane(); //文本框对象变量 text
    JDialog about = new JDialog(this); //实现“关于”功能的对话框对象 about
    JFileChooser filechooser = new JFileChooser(); //打开/保存文件对话框变量 filechooser
    GraphicsEnvironment getFont = GraphicsEnvironment
        .getLocalGraphicsEnvironment(); //环境变量 getFont
    Font[] fonts = getFont.getAllFonts(); //获取本地所有的字体
    JColorChooser colorchooser = new JColorChooser(); //颜色选择器对象 colorchooser

    //创建菜单的成员变量
    //File 菜单的菜单选项
    private JMenuItem jminew, jmiopen, jmisave, jmisaveas;
    //Edit 菜单的菜单选项
    private JMenuItem jmicut, jmicopy, jmiplaster, jmiall;
    //Format 菜单的菜单选项
    private JMenuItem jmifont, jmicolor;
    //Tool 菜单的菜单选项
    private JMenuItem jminotepad, jmicalculator;
    //Help 菜单的菜单选项
    private JMenuItem jmiabout;
    //Exit 菜单的菜单选项
    private JMenuItem jmiexit;
    public static void main(String[] args) { //主方法
        Notepad frame = new Notepad(); //创建 Notepad 对象
        //设置窗口的退出功能
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //显示窗口
        frame.pack();
        frame.setVisible(true);
    }
    public Notepad() { //无参构造函数
        //设置窗口对象
        setTitle("记事本"); //设置标题
        setLocation(100, 50); //设置位置
        //创建和设置菜单栏
        JMenuBar jmb = new JMenuBar(); //创建菜单栏对象 jmb
        setJMenuBar(jmb); //设置菜单栏到窗口中
        //创建工具栏上的 6 个菜单
        //创建和设置 File 菜单
        JMenu filemenu = new JMenu("File"); //创建菜单 filemenu
        filemenu.setMnemonic('F'); //设置该菜单的 hotkey 键
        jmb.add(filemenu); //添加菜单到菜单栏对象 jmb 中
        //创建和设置 Edit 菜单
        JMenu editmenu = new JMenu("Edit");
        editmenu.setMnemonic('E'); // set hotkey
        jmb.add(editmenu);
        //创建和设置 Format 菜单
        JMenu formatmenu = new JMenu("Format");
        formatmenu.setMnemonic('T'); // set hotkey
        jmb.add(formatmenu);
    }
}

```

```

//创建和设置 Tool 菜单
JMenu toolmenu = new JMenu("Tool");
toolmenu.setMnemonic('L');
jmb.add(toolmenu);
//创建和设置 Help 菜单
JMenu helpmenu = new JMenu("Help");
helpmenu.setMnemonic('H');// set hotkey
jmb.add(helpmenu);
//创建和设置 Exit 菜单
JMenu exitmenu = new JMenu("Exit");
exitmenu.setMnemonic('X');
//设置工具栏上 6 个菜单的菜单选项
//设置 File 菜单的菜单选项
//创建和设置 File 菜单的菜单选项 New
filemenu.add(jminew = new JMenuItem("New", 'N')); //添加 New 选项
jminew.setIcon(new ImageIcon("/images/Handle.gif")); //设置图片
//创建和设置 File 菜单的菜单选项 Open
filemenu.add(jmiopen = new JMenuItem("Open", 'O'));
jmiopen.setIcon(new ImageIcon("/images/folderOpen.gif"));
//创建和设置 File 菜单的菜单选项 Save
filemenu.add(jmisave = new JMenuItem("Save", 'S'));
jmisave.setIcon(new ImageIcon("/images/3.gif"));
filemenu.addSeparator(); //添加“分隔线”
//创建和设置 File 菜单的菜单选项 Save
filemenu.add(jmisaveas = new JMenuItem("Save as"));
jmisaveas.setIcon(new ImageIcon("images/7.gif"));
//设置前 3 个菜单选项的快捷键
jminew.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N,
        ActionEvent.CTRL_MASK)); //设置菜单选项 jminew 快捷方式
jmiopen.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O,
        ActionEvent.CTRL_MASK));
jmisave.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
        ActionEvent.CTRL_MASK));
//设置 Edit 菜单的菜单选项
editmenu.add(jmicut = new JMenuItem("Cut", 'X'));
jmicut.setIcon(new ImageIcon("images/face1.gif"));
editmenu.add(jmicopy = new JMenuItem("Copy", 'C'));
jmicopy.setIcon(new ImageIcon("images/face2.gif"));
editmenu.add(jmiplaster = new JMenuItem("Plaster", 'V'));
jmiplaster.setIcon(new ImageIcon("images/face3.gif"));
editmenu.addSeparator();
editmenu.add(jmiall = new JMenuItem("All"));
jmiall.setIcon(new ImageIcon("images/face4.gif"));
...
//设置和初始化文本框对象 text
setFont(new Font("Times New Roman", Font.PLAIN, 12)); //设置字体
JScrollPane scrollpane = new JScrollPane(text); //创建包含 text 的滚动面板
//设置滚动面板的大小
scrollpane.setPreferredSize(new Dimension(600, 500));
getContentPane().add(scrollpane); //添加滚动面板到窗口中
//为所有的菜单选项注册事件
//为 File 菜单选项注册事件
jminew.addActionListener(this);
jmiopen.addActionListener(this);
jmisave.addActionListener(this);
jmisaveas.addActionListener(this);
...
}

```

```
...
}
```

【代码解析】

上述代码主要用来实现记事本的主界面，在该界面中主要存在两个组件，即菜单工具栏的对象 `jmb` 和输入文本框对象 `text`，主界面的布局如图 16.24 所示。

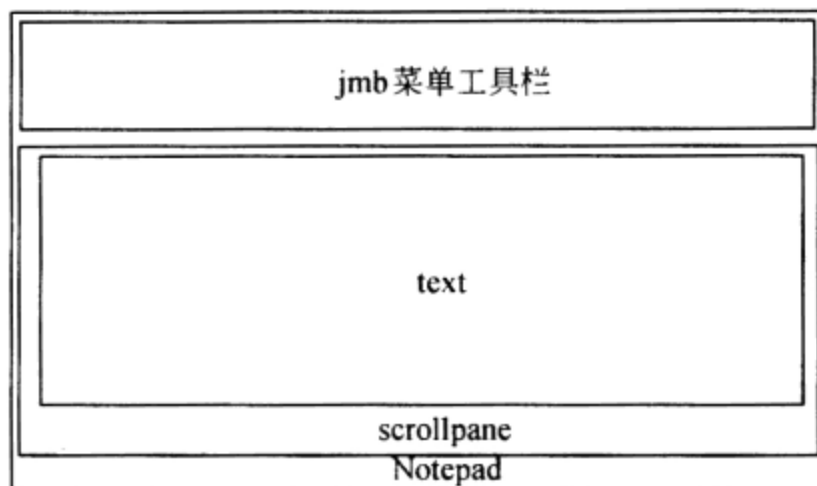


图 16.24 主界面布局

16.2.2 实现菜单功能

`Notepad.java` 类为记事本类，在该类中不仅会实现该项目的主界面，还会实现多数菜单的功能，菜单功能的具体内容如代码 16.2 所示。

代码 16.2 实现菜单功能：Notepad.java

```
public class Notepad extends JFrame implements ActionListener {
    ...
    public void actionPerformed(ActionEvent e) {           //事件监听器
        String name = e.getActionCommand();               //获取发生事件的组件显示字符
        //判断发生事件的组件
        if (e.getSource() instanceof JMenuItem) {
            //当发生事件的组件为 New 菜单选项时
            if ("New".equals(name)) {                       //New 菜单选项发生事件
                text.setText("");                           //清空 text 文本框
                file = null;                                 //为 file 变量赋值
            }
            //当发生事件的组件为 Open 菜单选项时
            if ("Open".equals(name)) {
                if (file != null)                            //当 file 对象不为 null 时
                    filechooser.setSelectedFile(file);       //设置对话框的选择文件属性
                //打开“打开文件对话框”
                int returnVal = filechooser.showOpenDialog(Notepad.this);
                //当单击“打开”按钮时
                if (returnVal == JFileChooser.APPROVE_OPTION) {
                    file = filechooser.getSelectedFile();    //设置 file 对象的值
                    try {                                     //读取 file 对象的内容
                        //创建输入流对象
                        FileReader fr = new FileReader(file);
                        int len = (int) file.length();       //获取 file 的长度
                    }
                }
            }
        }
    }
}
```

```

        char[] buffer = new char[len]; //创建字符数组
        fr.read(buffer, 0, len);      //实现读取功能
        fr.close();                    //关闭输入流
        //设置文本框的内容
        text.setText(new String(buffer));
    } catch (Exception e_open) {
        e_open.printStackTrace();
    }
}
//当发生事件的组件为 Save 菜单选项时
if ("Save".equals(name)) {
    if (file != null)
        filechooser.setSelectedFile(file);
    //打开“保存文件对话框”
    int returnVal = filechooser.showSaveDialog(Notepad.this);
    //当单击“保存”按钮时
    if (returnVal == JFileChooser.APPROVE_OPTION) {
        file = filechooser.getSelectedFile(); //设置 file 对象的值
    }
    try {                                //写入 file 对象的相应内容
        FileWriter fw = new FileWriter(file); //创建输出流对象
        fw.write(text.getText()); //把文本框的内容输出
        fw.close();                //关闭输出流
    } catch (Exception e_save) {
        e_save.printStackTrace();
    }
}
//当发生事件的组件为 Cut 菜单选项时
if ("Cut".equals(name)) {
    text.cut(); //调用 cut() 方法
}
//当发生事件的组件为 Copy 菜单选项时
if ("Copy".equals(name)) { //调用 copy() 方法
    text.copy();
}
//当发生事件的组件为 Plaster 菜单选项时
if ("Plaster".equals(name)) {
    text.paste(); //调用 paste() 方法
}
//当发生事件的组件为 All 菜单选项时
if ("All".equals(name)) {
    text.selectAll(); //调用 selectAll() 方法
}
//当发生事件的组件为 Font 菜单选项时
if ("Font".equals(name)) {
}
//当发生事件的组件为 Color 菜单选项时
if ("Color".equals(name)) {
    //为变量 color 赋值
    color = JColorChooser.showDialog(Notepad.this, "颜色选择对话框", color);
    text.setForeground(color); //设置文件的颜色
}
//当发生事件的组件为 MS Notepad 菜单选项时
if ("MS Notepad".equals(name)) {
    try {

```

```

        String command = "notepad.exe"; //创建 notepad 的命令
        //执行命令
        Process child = Runtime.getRuntime().exec(command);
    } catch (IOException ex) {
    }
}
//当发生事件的组件为 MS Calculator 菜单选项时
if ("MS Calculator".equals(name)) {
    try {
        String command = calc.exe;      //创建 calc 的命令
        //执行命令
        Process child = Runtime.getRuntime().exec(command);
    } catch (IOException ex) {
    }
}
//当发生事件的组件为 About 菜单选项时
if ("About".equals(name)) {
    about.setLayout(new GridLayout(6, 1));
                                //设置对话框 About 的布局管理器
    about.setTitle("About..."); //设置对话框的标题
    about.setSize(320, 280);    //设置对话框的大小
    //设置对话框的背景颜色
    about.getContentPane().setBackground(Color.BLUE);
    JLabel jlbfirst = new JLabel(); //创建面板对象 jlbfirst
    jlbfirst.setIcon(new ImageIcon("images/wx2.gif"));
    about.getContentPane().add(jlbfirst);
                                //添加面板 jlbfirst 到对话框中

    //添加相应的信息
    about.getContentPane().add(new JLabel("Star"));
    about.getContentPane().add(
        new JLabel("Edition 2.0 (author:cjgong)"));
    about.getContentPane().add(
        new JLabel(
            "copyright possession (C) xxxx.x.
            x Star Corp."));
    about.getContentPane().add(
        new JLabel(
            "Thank you to use!"));
    about.setModal(true);      //设置对话框模式
    about.show();              //显示对话框
}
//当发生事件的组件为 Exit 菜单选项时
if ("Exit".equals(name)) {
    System.exit(0);            //退出系统
}
}
}
...
}

```

【代码解析】

- 对于 File 菜单，当发生事件的菜单选项为 File | New 时，则清空组件 text 的内容，然后设置对象 file 的值为 null；当发生事件的菜单选项为 File | Open 时，则通过组件 text 里设置打开文件对话框所返回的文件流；当发生事件的菜单选项为 File | Save 命令时，则通过保存对话框把组件 text 里的内容保存到文件里。

- ❑ 对于 Edit 菜单，当发生事件的菜单选项为 Edit | Cut 时，则通过组件 text 的 cut() 方法实现剪切功能；当发生事件的菜单选项为 Edit | Copy 时，则通过组件 text 的 copy() 方法实现复制功能；当发生事件的菜单选项为 Edit | Plaster 时，则通过组件 text 的 paste() 方法实现粘贴功能；当发生事件的菜单选项为 Edit | All 时，则通过组件 text 的 selectAll() 方法实现全选功能。
- ❑ 对于菜单 Format，当发生事件的菜单选项为 Formal | Font 时，则会创建出文件格式对象 fontchooser；当发生事件的菜单选项为 Formal | Color 时，则会打开调色板对象。
- ❑ 对于菜单 Tool，当发生事件的菜单选项为 Tool | MS Notepad 时，则会打开系统中的记事本对象；当发生事件的菜单选项为 Tool | MS Calculator 时，则会打开系统中的计算器对象。
- ❑ 对于菜单 Help，当发生事件的菜单选项为 Help | About 时，则会打开该项目的帮助对话框；对于菜单 Exit，当发生事件的菜单选项为 Exit | Exit 时，则会退出该项目。

16.2.3 文件类型的过滤

在 Windows 系统中当使用各种文件对话框时，可以通过该对话框中的类型选择来过滤文件的类型，具体操作如图 16.25 所示。为了实现对话框组件中文件类型的过滤，专门创建了实现 txt 文件类型过滤的类 MyFileFilter，其具体内容如代码 16.3 所示。



图 16.25 类型的过滤

代码 16.3 过滤文件类型类：MyFileFilter.java

```
public class MyFileFilter extends javax.swing.filechooser.FileFilter {
    @Override
    public String getDescription() {                //重写 getDescription() 方法
        return "记事本文件 (*.txt)";
    }
    @Override
    public boolean accept(File f) {                //重写 accept() 方法
        if (f.isDirectory() || f.getName().endsWith(".txt")) {
            return true;
        } else {

```



```

        return false;
    }
}
}

```

【代码解析】

在 Java 语言中，如果想实现文件类型过滤类，只需要继承 `FileFilter` 类同时重写该类中的两个方法 `getDescription()` 和 `accept()` 就可以。

16.3 记事本的实现过程——字体设置对话框

在模拟记事本（notepad）的项目中，当选择菜单 `Format|Font` 命令时，会弹出字体设置对话框，通过对该对话框的设置，可以修改记事本 `text` 组件里的内容。字体设置对话框的属性如图 16.26 所示，方法如图 16.27 所示。



图 16.26 字体设置对话框的属性



图 16.27 字体设置对话框的方法

16.3.1 字体设置对话框——主界面

`FontChooserDialog.java` 类为字体设置对话框类，在该类中实现了对字体的各种设置，首先查看一下该对话框的主界面，具体内容如代码 16.4 所示。

代码 16.4 字体对话框类主界面：FontChooserDialog.java

```

public class FontChooserDialog extends JDialog {
    //创建成员变量
    private JPanel jPanel = null;           //创建面板对象 jPanel
    private JScrollPane jScrollPane = null; //创建滚动面板对象 jScrollPane
    private JPanel jPanel1 = null;          //创建面板对象 jPanel1

```

```

private JLabel jLabel = null;           //创建标签对象 jLabel
private JLabel jLabel1 = null;          //创建标签对象 jLabel1
private JLabel jLabel2 = null;          //创建标签对象 jLabel2
...
//创建字体默认变量
private Font defaultFont = new Font("\u5b8b\u4f53", Font.PLAIN, 12);
private static Font returnFont = null;  //返回字体变量
private static boolean judge = false;   //是否正常返回变量
//两个判断循环的变量
private boolean nameJuge = true;
private boolean sizeJuge = true;
//构造函数
public FontChooserDialog() {             //无参构造函数
    this(null);
}
public FontChooserDialog(JFrame jframe) { //1 个参数构造函数
    this(jframe, true);
}
//2 个参数构造函数
public FontChooserDialog(JFrame jframe, boolean boo) {
    this(jframe, boo, null);
}
//3 个参数构造函数
public FontChooserDialog(JFrame jframe, boolean boo, Font font) {
    super(jframe, boo);
    initialize();
    initializeFont(font);
    this.setLocationRelativeTo(jframe);
}
private void initialize() {              //初始化方法
    this.setContentPane(getJPanel());    //添加 jPanel 面板到窗口中
    //创建字体
    this.setFont(new Font("\u5b8b\u4f53", Font.PLAIN, 12));
    this.setBounds(new Rectangle(0, 0, 430, 335));
    this.setTitle("字体选择对话框");      //设置窗口标题
    this.addWindowListener(new WindowAdapter() { //添加窗口监听器
        public void windowClosing(WindowEvent e) {
            judge = false;
            closeWindow();
        }
    });
}
//创建显示对话框方法
public static Font showDialog(JFrame jframe, boolean boo) {
    return showDialog(jframe, boo, null); //调用带有3个参数的showDialog()方法
}
//带有3个参数的showDialog()方法
public static Font showDialog(JFrame jframe, boolean boo, Font font) {
    JDialog jd = new FontChooserDialog(jframe, boo, font);
    jd.setVisible(true);
    if (judge) {
        returnFont = fontStyle.getFont();
    }
    jd.dispose();
    return returnFont;
}
private JPanel getJPanel() {             //初始化 jPanel 对象
    if (jPanel == null) {

```

```

        jPanel = new JPanel();           //为 jPanel 对象赋值
        jPanel.setLayout(null);          //设置布局管理器
        //设置字体
        jPanel.setFont(new Font("Dialog", Font.PLAIN, 12));
        jPanel.add(getJPanel1(), null); //添加 jPanel1 对象到 jPanel 对象中
        jPanel.add(getJPanel2(), null); //添加 jPanel2 对象到 jPanel 对象中
        jPanel.add(getOkButton(), null); //添加“确定”按钮到 jPanel 对象中
        jPanel.add(getRegitButton(), null);
                                   //添加“重置”按钮到 jPanel 对象中
        jPanel.add(getCancleButton(), null);
                                   //添加“取消”按钮到 jPanel 对象中
    }
    return jPanel;                      //返回 jPanel 对象
}
...
private void initializeFont(Font font) { //默认字体初始化方法
    if (font != null) {
        defaultFont = font;
        fontStyle.setFont(defaultFont);
    }
    fontStyle.setFont(defaultFont);
    fontNameList.setSelectedValue(defaultFont.getFontName(), true);
    fontSizeList.setSelectedValue(new Integer(defaultFont.getSize())
        .toString(), true);
    fontItalicList.setSelectedIndex(defaultFont.getStyle());
}
private Object getLateIndex(JList jlist, String str) {
                                   //判断里给定的值最近的索引
    ListModel list = jlist.getModel();
    ...
    return list.getElementAt(0);
}
public Font returnfonttype() { //所选择的字体
    Font f = new Font(fontStyle.getFont().getFontName(), fontStyle
        .getFont().getStyle(), Integer.parseInt(fontSizeList
        .getSelectedValue().toString()));
    return f;
}
private void closeWindow() { //窗体关闭方法
    this.setVisible(false);
}
}

```

【代码解析】

- 在上述代码中，为了方便创建出该类，设置了该类的 3 个构造函数，同时还创建了一个初始化 `initialize()` 方法和两个返回该对话框所设置的字体格式的 `showDialog()` 方法。最后通过 `getJPanel()` 方法设置了该对话框的布局，组件 `jpanel` 对象的布局如图 16.28 所示。
- 在上述代码中，还存在一个根据给定的值判断最近索引的 `getLateIndex()` 方法，该方法在获取文字名字方法 `getFontNameText()` 和获取文字大小方法 `getFontSizeText()` 中调用。

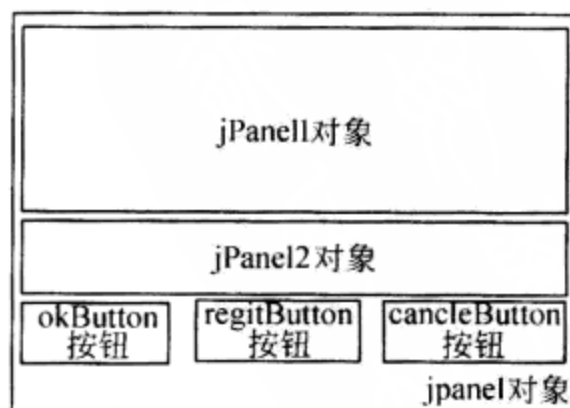


图 16.28 jPanel 对象的布局

16.3.2 字体设置对话框——JPanel1 组件界面

FontChooserDialog.java 类为字体设置对话框类, 16.3.1 节已经讲解了该对话框的主界面, 本节将接着 16.3.1 节的内容继续讲解 JPanel1 组件的界面, 具体内容如代码 16.5 所示。

代码 16.5 字体对话框中的 JPanel1 用户界面: FontChooserDialog.java

```
public class FontChooserDialog extends JDialog {
    ...
    private JPanel getJPanel1() { //初始化 jPanel1 对象
        if (jPanel1 == null) {
            //创建和设置 jLabel 对象
            jLabel = new JLabel();
            jLabel.setBounds(new Rectangle(5, 5, 150, 15));
            jLabel.setDisplayedMnemonic(KeyEvent.VK_UNDEFINED);
            //设置字体
            jLabel.setFont(new Font("\u5b8b\u4f53", Font.PLAIN, 12));
            jLabel.setText("字体:"); //设置内容
            //创建和设置 jLabel1 对象
            jLabel1 = new JLabel();
            jLabel1.setBounds(new Rectangle(160, 5, 120, 15));
            //设置字体
            jLabel1.setFont(new Font("\u5b8b\u4f53", Font.PLAIN, 12));
            jLabel1.setText("字型:"); //设置内容
            //创建和设置 jLabel2 对象
            jLabel2 = new JLabel(); //为对象 jLabel2 赋值
            jLabel2.setBounds(new Rectangle(285, 5, 120, 15));
            //设置字体
            jLabel2.setFont(new Font("\u5b8b\u4f53", Font.PLAIN, 12));
            jLabel2.setText("大小:"); //设置内容
            //创建和设置 jPanel1 对象
            jPanel1 = new JPanel();
            jPanel1.setLayout(null); //设置布局管理器
            jPanel1.setBounds(new Rectangle(5, 5, 410, 175));
            jPanel1.setBorder(new SoftBevelBorder(SoftBevelBorder.
                LOWERED));
            //添加 3 个标签对象 jLabel、jLabel1 和 jLabel2 到对象 jPanel1 里
            jPanel1.add(jLabel, null);
            jPanel1.add(jLabel1, null);
            jPanel1.add(jLabel2, null);
            //添加 3 个文本框对象 fontNameText、fontItalicText 和 fontSizeText
            到对象 jPanel1 里
            jPanel1.add(getFontNameText(), null);
            jPanel1.add(getFontItalicText(), null);
            jPanel1.add(getFontSizeText(), null);
            //添加 3 个滚动面板对象 jScrollPane1、jScrollPane2 和 jScrollPane3
            到对象 jPanel1 里
            jPanel1.add(getJScrollPane1(), null);
            jPanel1.add(getJScrollPane2(), null);
            jPanel1.add(getJScrollPane3(), null);
        }
        return jPanel1; //返回对象 jPanel1
    }
}
```

```

private JTextField getFontNameText() { //初始化 fontNameText 对象
    if (fontNameText == null) {
        //创建和设置 fontNameText 对象
        fontNameText = new JTextField(); //为 fontNameText 对象赋值
        fontNameText.setBounds(new Rectangle(5, 25, 150, 20));
        //设置字体
        fontNameText.setFont(new Font("\u5b8b\u4f53", Font.PLAIN,
        12));
        fontNameText.addKeyListener(new KeyAdapter() { //添加键盘监听器
            public void keyTyped(KeyEvent e) {
                String oldText = fontNameText.getText();
                //获取原来的内容
                String newText = ""; //创建一个空字符串对象
                if ("".equals(fontNameText.getSelectedText())
                    && null == fontNameText.getSelectedText()) {
                    //当没有选择, 通过键盘输入时
                    newText = fontNameText.getText() + e.getKeyChar();
                } else {
                    newText = oldText.substring(0, fontNameText
                        .getSelectionStart())
                        + e.getKeyChar()
                        + oldText.substring(fontNameText
                            .getSelectionEnd());
                }
                nameJuge = false; //设置变量 nameJuge 的值
                //设置对象 fontNameList 的选择项
                fontNameList.setSelectedValue(getLateIndex(font-
                NameList,newText), true);
                nameJuge = true; //设置变量 nameJuge 的值
            }
        });
    }
    return fontNameText; //返回 fontNameText 对象
}

private JTextField getFontItalicText() { //初始化 fontItalicText 对象
    if (fontItalicText == null) {
        fontItalicText = new JTextField(); //为对象 fontItalicText 赋值
        fontItalicText.setBounds(new Rectangle(160, 25, 120, 20));
        //设置字体
        fontItalicText.setFont(new Font("\u5b8b\u4f53", Font.PLAIN,
        12));
        fontItalicText.setEnabled(false); //设置模式
    }
    return fontItalicText; //返回 fontItalicText 对象
}

...
private JScrollPane getJScrollPane() { //初始化 jScrollPane1() 方法
    ...
}

...
}

```

【代码解析】

在上述代码中, 首先通过 `getJPanel1()` 方法获取面板对象 `jPanel1`, 在该方法中通过布局管理器组织该组件中的各种组件, 而各种组件则通过 `getXXX()` 方法来初始化。`jPanel1`

组件的基本布局如图 16.29 所示。

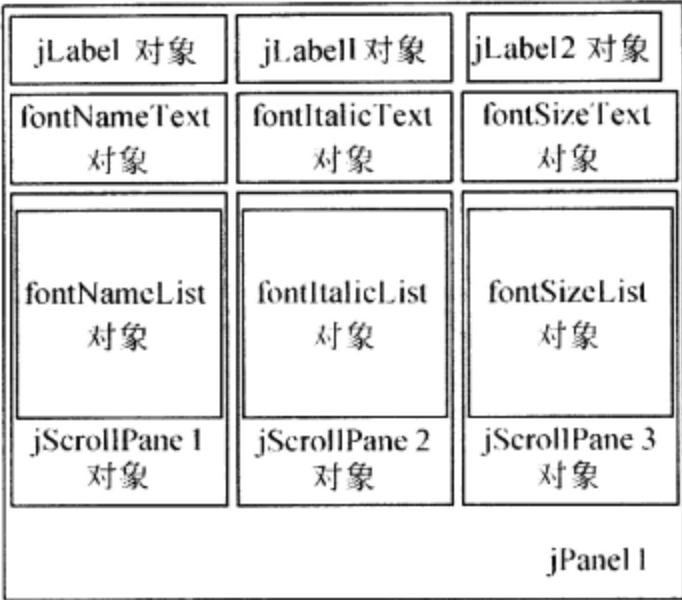


图 16.29 JPanel1 组件的布局

16.3.3 字体设置对话框——其他组件

FontChooserDialog.java 类为字体设置对话框类，前面已经讲解了该对话框的主界面和 JPanel1 组件界面，本节将接着前面的内容继续讲解其他组件的界面，具体内容如代码 16.6 所示。

代码 16.6 字体对话框中用户界面的其他对象：FontChooserDialog.java

```
public class FontChooserDialog extends JDialog {
    ...
    private JPanel getJPanel2() { //初始化 jPanel2 对象
        if (jPanel2 == null) {
            jPanel2 = new JPanel();
            jPanel2.setLayout(null);
            jPanel2.setBounds(new Rectangle(3, 180, 414, 90));
            jPanel2.setBorder(BorderFactory.createTitledBorder(null,
                "\u6548\u679c\u9884\u89c8",
                TitledBorder.DEFAULT_JUSTIFICATION,
                TitledBorder.DEFAULT_POSITION, new Font("\u5b8b\u4f53",
                    Font.PLAIN, 12), new Color(51, 51, 51)));
            jPanel2.add(getJScrollPane(), null);
        }
        return jPanel2;
    }
    private JScrollPane getJScrollPane() { //初始化 jScrollPane 对象
        if (jScrollPane == null) {
            //创建和设置标签对象 fontStyle
            fontStyle = new JLabel(); //为标签对象 fontStyle 赋值
            //设置标签对象的内容
            fontStyle.setText("你好！天生我才必有用！Hello World!");
            //设置标签对象的字体
            fontStyle.setFont(new Font("\u5b8b\u4f53", Font.PLAIN, 12));
            //设置标签对象的对齐方式
            fontStyle.setHorizontalAlignment(SwingConstants.CENTER);
            fontStyle.setHorizontalTextPosition(SwingConstants.CENTER);
            //创建和设置滚动面板
```



```

        jScrollPane = new JScrollPane(); //为滚动面板对象 jScrollPane 赋值
        jScrollPane.setBorder(BorderFactory
            .createBevelBorder(BevelBorder.LOWERED));
        jScrollPane.setViewportView(fontStyle);
            //添加对象 fontStyle 到对象 jScrollPane 中
        jScrollPane.setBounds(new Rectangle(5, 20, 400, 60));
    }
    return jScrollPane; //返回对象 jScrollPane
}
private JButton getOkButton() { //初始化 okButton 对象
    if (okButton == null) {
        okButton = new JButton();
        okButton.setBounds(new Rectangle(215, 275, 60, 20));
        okButton.setFont(new Font("\u5b8b\u4f53", Font.PLAIN, 12));
        okButton.setText("确定");
        okButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                judge = true;
                closeWindow();
            }
        });
    }
    return okButton;
}
private JButton getRegitButton() { //初始化 regitButton 对象
    if (regitButton == null) {
        regitButton = new JButton();
        regitButton.setBounds(new Rectangle(285, 275, 60, 20));
        regitButton.setFont(new Font("\u5b8b\u4f53", Font.PLAIN, 12));
        regitButton.setText("重置");
        regitButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                initializeFont(null);
            }
        });
    }
    return regitButton;
}
...
}

```

【代码解析】

在上述代码中，首先通过 `getJPanel2()` 方法获取面板对象 `jPanel2`，在该方法中通过布局管理器组织该组件里的各种对象，然后通过 `getXXXButton()` 方法获取各种按钮对象。组件 `jPanel2` 的基本布局如图 16.30 所示。

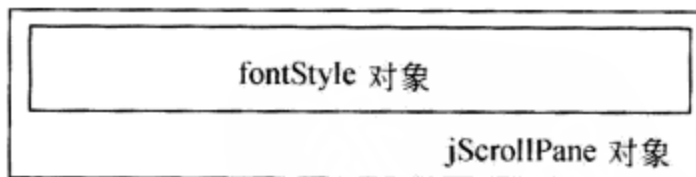


图 16.30 `jPanel2` 对象的布局

16.4 小 结

本章主要通过菜单组件和对话框组件实现模拟记事本项目，不仅该项目的界面非常接近于 Windows 系统的记事本，而且还通过对话框组件实现了各个菜单项功能。同时为了实现字体设置功能，还专门创建了一个字体设置对话框的类。

第 17 章 拼图游戏（GUI 综合应用）

本章将通过图形用户界面的相关知识编写一个复杂的拼图游戏，在该游戏中主要使用事件机制、布局机制和 Swing 组件，即相当于对于图形用户界面中所有知识的一个综合应用。本章将详细讲解“拼图游戏”项目，并详细介绍事件的处理过程。

本章的学习目标如下：

- ❑ 掌握“拼图游戏”项目；
- ❑ 理解事件处理的内部机制；
- ❑ 掌握实现线程同步的两种方式。

17.1 拼图游戏原理

所谓“拼图游戏”，是指将一个完整的图片分割成若干个规则的小图片，然后将这些小图片随机地拼接在一起，然后由玩家按照原图重新拼接出正确的图片。

17.1.1 项目结构框架分析

对于拼图游戏，根据面向对象的思想，需要创建移动按钮、主面板和主窗口 3 个对象。拼图游戏项目目录如图 17.1 所示，该项目中的 3 个类，分别为移动按钮 Cell、主面板 MyCanvas 和主窗口 MyMainFrame。

17.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括拼图游戏的初始化界面、预览拼图、设置拼图和通过鼠标单击按钮模拟的“拼图”效果。

1. 拼图游戏的初始化界面

当运行测试定义按钮的类 MyMainFrame 后，会出现如图 17.2 所示的初始界面。在该界面的最上方有 3 个按钮（“开始”按钮、“预览”按钮和“设置”按钮）的面板对象，而下面则会存在主面板对象。

2. 预览拼图

在具体玩拼图游戏时，首先需要预览所设置的图片。单击“预览”按钮，主面板的地方就会由预览面板（显示所设置图片的面板）代替，同时“预览”按钮变成“返回”按钮。

单击“返回”按钮不仅实现预览图片由主面板代替，而且“返回”按钮变成“预览”按钮，具体过程如图 17.3 所示。

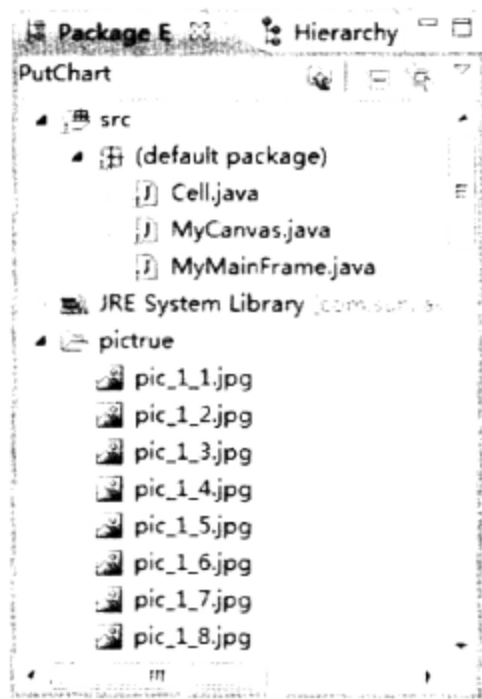


图 17.1 项目目录



图 17.2 初始界面

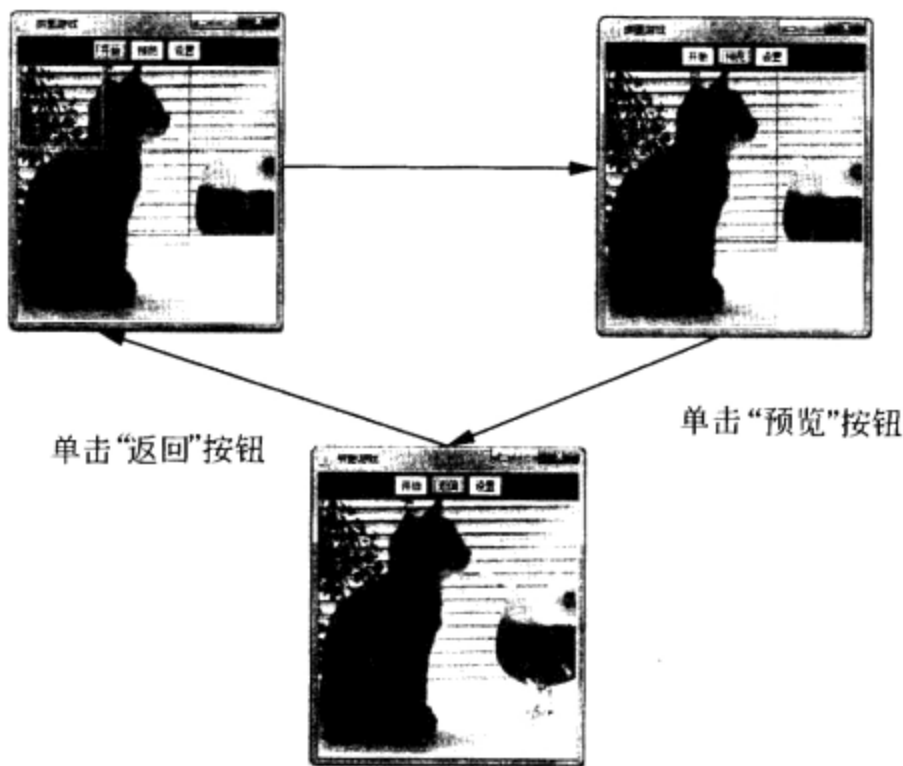


图 17.3 预览过程

3. 设置拼图

在具体玩拼图游戏时，如果想更换图片，可以单击“设置”按钮会弹出如图 17.4 所示的“选择图片”对话框。在该对话框中选择相应选项就可以实现图片的更换。

4. 开始拼图

在开始玩拼图游戏时，首先需要单击“开始”按钮，这时图片就会被打乱摆放。然后通过单击与空白网格相邻的图片实现图片的移动，直到各个图片拼接完成完整图片为止，具体过程如图 17.5 所示。

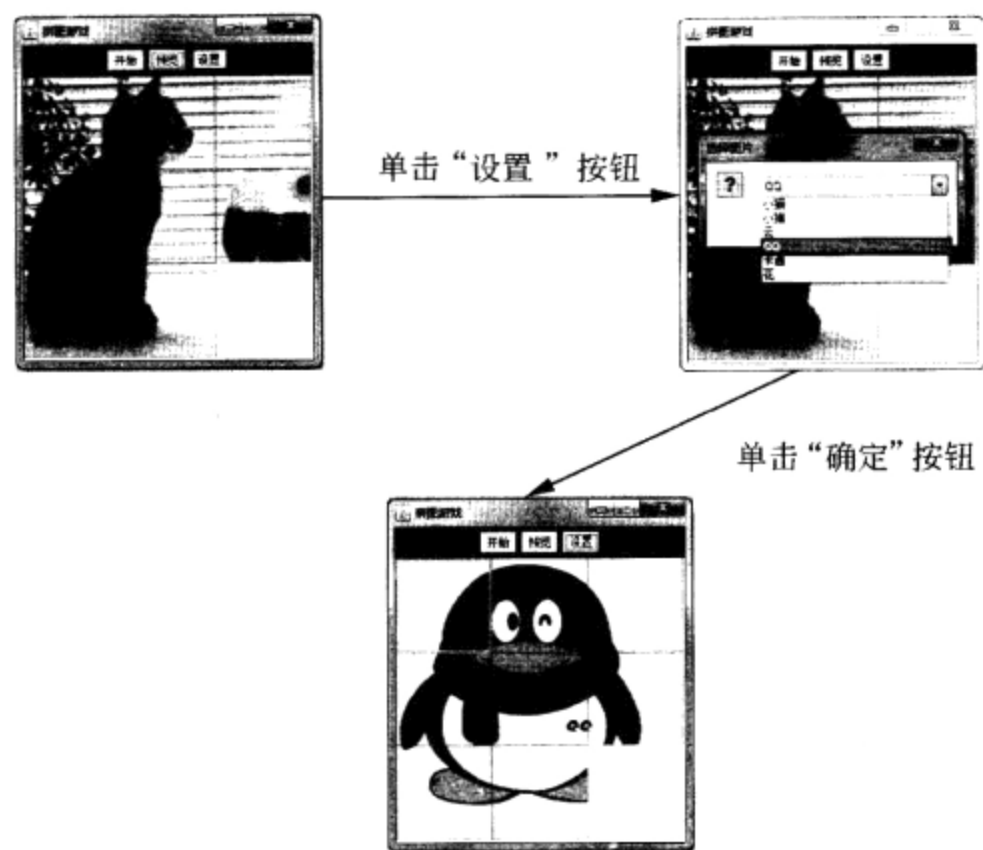


图 17.4 设置按钮

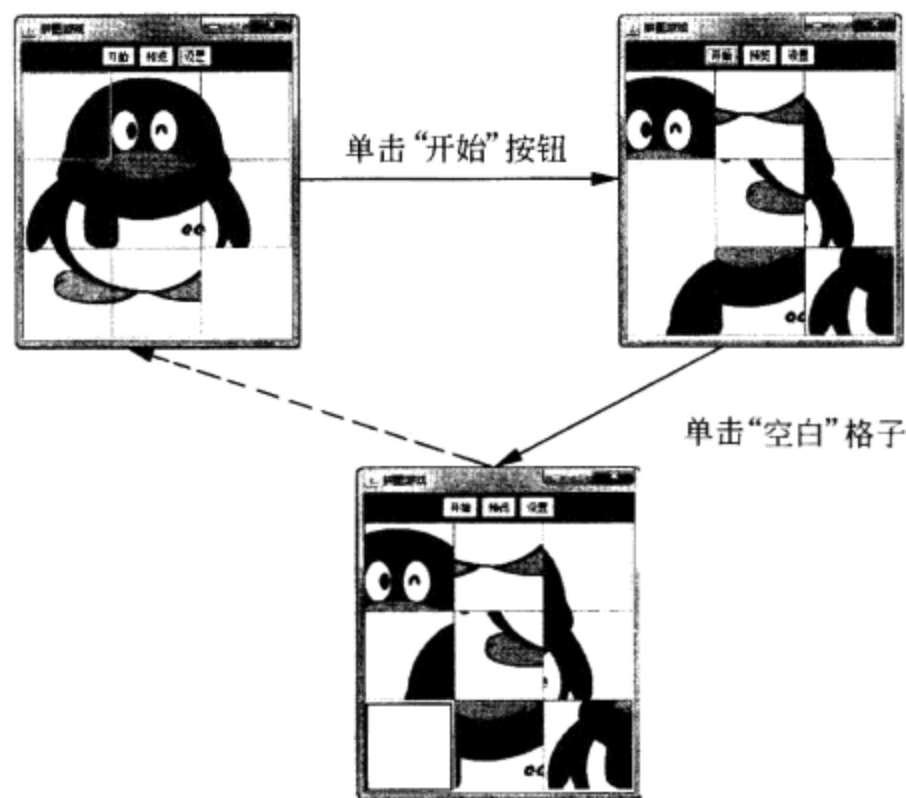


图 17.5 具体玩拼图游戏

17.1.3 拼图游戏项目的原理

游戏玩家在玩拼图游戏时，单击与空白网格相邻的网格，可以将该网格显示的图片移动到空白的网格中，重复这种操作直到把随机的图片拼接成正确的图片。

如果想实现上述的操作，需要有以下几个问题需要解决。

- (1) 如何实现图片的移动。
- (2) 如何判断被单击的网格与空白的网格是否相邻。
- (3) 如何实现图片的随机摆放。

实现图片的移动有两种方案：一种是将图片和按钮绑定，然后改变按钮在面板中的位置；另一种是固定按钮在面板中的位置，然后改变按钮上显示的图片。本节采用第一种方式。

如何判断被单击的按钮是否与空白网格相邻？如果两个按钮相邻，一种是它们在同一行，并且在水平方向上相差一列；另一种是它们在同一列，并且在垂直方向上相差一行。本节中，由于每个按钮大小都为 100×100，所以对于水平方向上的相邻有两种情况，第一种情况发生事件的按钮比空白网格的 x 轴坐标多 100，同时 y 轴坐标相同，则需要向左移动；第二种情况发生事件的按钮比空白网格的 x 轴坐标少 100，同时 y 轴坐标相同，则需要向右移动。对于垂直方向上的相邻也有两种情况，第一种情况发生事件的按钮比空白网格的 y 轴坐标多 100，同时 x 轴坐标相同，则需要向上移动；第二种情况发生事件的按钮比空白网格的 y 轴坐标少 100，同时 x 轴坐标相同，则需要向下移动。

最后，如何实现图片的随机摆放呢？可以利用 java.lang.Math 类的 Math.random()方法获取一个随机数，然后通过表达式 “Math.random() * 4” 获取 0~4 的任意一个数，这 4 个数字表示 “上下左右” 4 个方向。获取到随机产生的方向后，空白网格就会与在该方向上相邻的按钮进行交换。

17.2 拼图游戏的实现过程

本章通过事件机制的相关知识来实现拼图游戏项目，具体程序架构如图 17.6 所示。其中 Cell 为自定义按钮类，MyCanvas 为面板类，主要用来加载按钮类，窗口类 MyMainFrame 则用来显示所有组件。

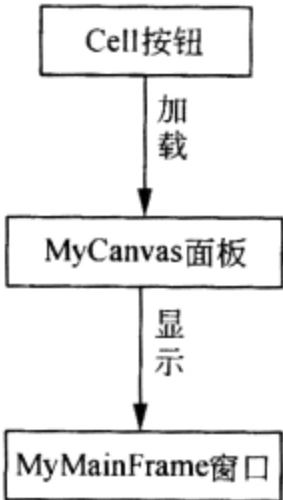


图 17.6 程序关系图

17.2.1 实现移动功能的按钮类

Cell.java 类为自定义的按钮类，在该类中实现了向 4 个方向移动的功能。具体内容如代码 17.1 所示，该类的 UML 如图 17.7 所示。

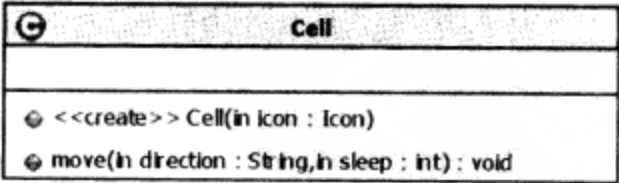


图 17.7 按钮类图

代码 17.1 按钮类：Cell.java

```
public class Cell extends JButton {
    Cell(Icon icon) {                                //构造函数
```

```

        super(icon);
        this.setSize(100, 100); //设置按钮的大小
    }
    public void move(String direction, int sleep) { //方格的移动
        if (direction == "UP") { //当向上移动时,x 坐标不变,y 坐标减少 100
            this.setLocation(this.getBounds().x, this.getBounds().y - 100);
        } else if (direction == "DOWN") { //当向下移动时, x 坐标不变, y 坐标增加 100
            this.setLocation(this.getBounds().x, this.getBounds().y + 100);
        } else if (direction == "LEFT") { //当向左移动时, y 坐标不变, x 坐标减少 100
            this.setLocation(this.getBounds().x - 100, this.getBounds().y);
        } else { //当向右移动时,y 坐标不变,x 坐标增加 100
            this.setLocation(this.getBounds().x + 100, this.getBounds().y);
        }
    }
}

```

【代码解析】

在上述代码中, 由于该类是自定义按钮, 所以继承了 `JButton` 类。由于该类需要移动功能, 所以创建了 `move()` 方法。

17.2.2 主面板的类

`MyCanvas.java` 类为自定义的面板, 该面板主要用来实现拼图游戏的主要功能。该类的具体内容如代码 17.2 所示, 该类的 UML 如图 17.8 所示。

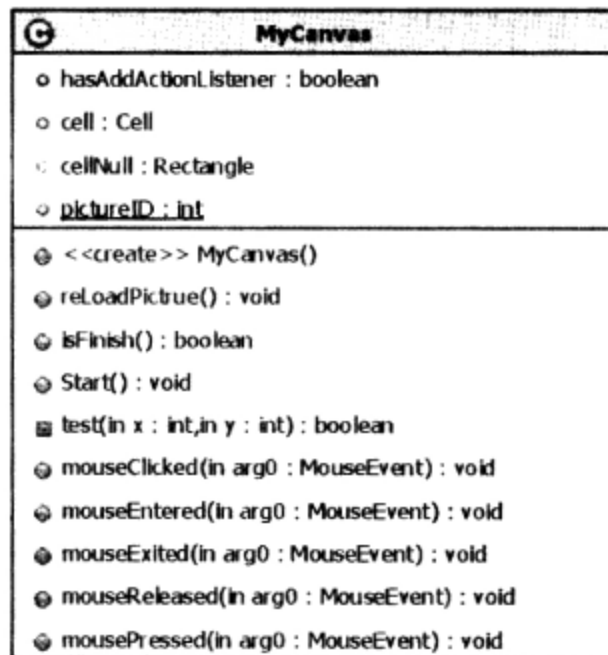


图 17.8 主面板类图

代码 17.2 主面板类: `MyCanvas.java`

```

public class MyCanvas extends JPanel implements MouseListener {
    //动作监听器的标志位, TRUE 为已经添加的动作事件, FALSE 是尚未添加的动作事件
    boolean hasAddActionListener = false;
    Cell cell //按钮数组
    Rectangle cellNull; //空方格区域数组
    public static int pictureID = 1; //当前选择的图片代号
    public MyCanvas() { //构造函数
        this.setLayout(null); //设置布局管理器
        this.setSize(400, 400); //设置大小
        cellNull = new Rectangle(200, 200, 100, 100); //空方格区域在第三行第三列
        cell = new Cell[9]; //为按钮数组赋值
        Icon icon; //创建 icon 对象
        //为 9 个方格加载图片, 并初始化坐标, 形成 3 行 3 列
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                icon = new ImageIcon("pictrue/pic_" + pictureID + "_"
                    + (i * 3 + j + 1) + ".jpg"); //为对象 icon 赋值
                cell[i * 3 + j] = new Cell(icon); //为按钮对象赋值
                //设置按钮的位置
                cell[i * 3 + j].setLocation(j * 100, i * 100);
                this.add(cell[i * 3 + j]); //添加按钮到当前对象上
            }
        }
        this.remove(cell[8]); //移除最后一个多余的方格
    }
}

```



```

    }
    //当选择其他图形进行拼图时,需重新加载新图片
    public void reLoadPictrue() {
        Icon icon; //创建 icon 对象
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                icon = new ImageIcon("pictrue/pic_" + pictureID + "_"
                    + (i * 3 + j + 1) + ".jpg");
                cell[i * 3 + j].setIcon(icon); //设置按钮的 icon 对象
            }
        }
    }
    //判断拼接是否成功
    public boolean isFinish() {
        for (int i = 0; i < 8; i++) {
            int x = cell[i].getBounds().x; //获取按钮的 x 坐标
            int y = cell[i].getBounds().y; //获取按钮的 y 坐标
            if (y / 100 * 3 + x / 100 != i) //判断是否在正确的位置
                return false;
        }
        return true;
    }
    //对方格进行重新排列,打乱顺序
    public void Start() {
        //当第一个方格距左上角较近时
        while (cell[0].getBounds().x <= 100 && cell[0].getBounds().y <= 100) {
            int x = cellNull.getBounds().x; //获取空格的 x 坐标
            int y = cellNull.getBounds().y; //获取空格的 y 坐标
            //产生 0~4, 对应空方格的上下左右移动
            int direction = (int) (Math.random() * 4);
            //空方格左移动,与左侧方格互换位置
            if (direction == 0) {
                x -= 100;
                if (test(x, y)) {
                    for (int j = 0; j < 8; j++) {
                        //依次寻找左侧的按钮
                        if ((cell[j].getBounds().x == x)
                            && (cell[j].getBounds().y == y)) {
                            cell[j].move("RIGHT", 100); //左边的按钮右移
                            cellNull.setLocation(x, y); //右边的按钮左移
                            break;
                        }
                    }
                }
            }
        }
    }
    ...
    }
    //为按钮添加事件
    if (!hasAddActionListener) //如果尚未添加动作事件,则添加
        for (int i = 0; i < 8; i++)
            cell[i].addMouseListener(this);
    hasAddActionListener = true;
}
//判断按钮是否到了面板的边界
private boolean test(int x, int y) {
    if ((x >= 0 && x <= 200) || (y >= 0 && y <= 200))
        return true;
    else

```

```

        return false;
    }
    ...
    //鼠标按下时的事件
    public void mousePressed(MouseEvent arg0) {
        Cell button = (Cell) arg0.getSource();           //获取事件源
        int x1 = button.getBounds().x;                   //获取事件源的坐标
        int y1 = button.getBounds().y;
        //获取空格的坐标
        int x2 = cellNull.getBounds().x;
        int y2 = cellNull.getBounds().y;
        //进行方向移动判断
        if (x1 == x2 && y1 - y2 == 100)
            button.move("UP", 100);
        else if (x1 == x2 && y1 - y2 == -100)
            button.move("DOWN", 100);
        else if (x1 - x2 == 100 & y1 == y2)
            button.move("LEFT", 100);
        else if (x1 - x2 == -100 && y1 == y2)
            button.move("RIGHT", 100);
        else
            return;
        cellNull.setLocation(x1, y1);                     //空格移动
        this.repaint();
        if (this.isFinish()) {                             //进行是否完成的判断
            JOptionPane.showMessageDialog(this, "恭喜你完成拼图,加油!");
            //如果已完成,撤销鼠标事件,鼠标单击方格不再起作用
            for (int i = 0; i < 8; i++)
                cell[i].removeMouseListener(this);
            hasAddActionListener = false;                 //设置事件状态
        }
    }
}

```

【代码解析】

在上述代码中,由于该类是自定义面板,所以继承了 JPanel 类。在该类中主要存在 4 个方法,即实现重新加载图片的 reLoadPictrue()方法;实现判断是否拼合成功的 isFinish()方法;实现对方格进行重新排列的 Start()方法;鼠标按下事件的 mousePressed(MouseEvent arg0)方法。

17.2.3 主窗口的类

MyMainFrame.java 类为自定义的窗口类,该窗口主要用来实现对所有子组件的布局功能。该类的具体内容如代码 17.3 所示,该类的 UML 如图 17.9 所示。

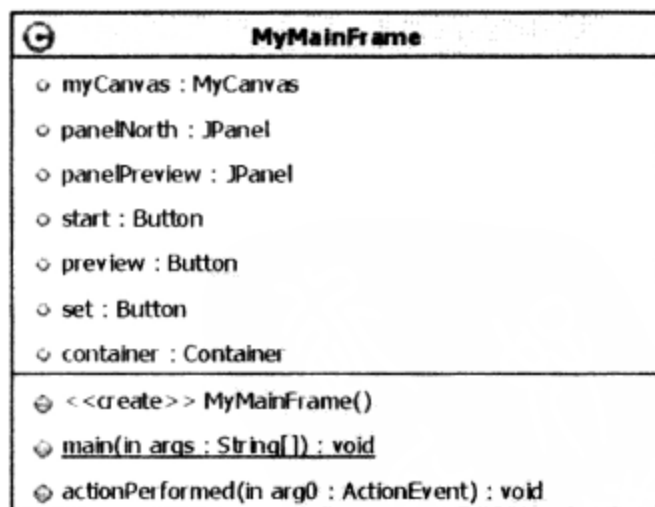


图 17.9 主窗口类图

代码 17.3 主窗口类: MyMainFrame.java

```

public class MyMainFrame extends JFrame implements ActionListener {
    MyCanvas myCanvas;           //创建主面板对象
    JPanel panelNorth, panelPreview; //创建上方的面板,并预览所需的面板
    Button start, preview, set;   //创建开始、预览和设定按钮
}

```

```

Container container;           //创建窗口的子对象
public MyMainFrame() {         //构造函数
    container = this.getContentPane(); //为对象 container 赋值
    //为 3 个按钮赋值并设置事件
    start = new Button("开始");
    start.addActionListener(this);
    preview = new Button("预览");
    preview.addActionListener(this);
    set = new Button("设置");
    set.addActionListener(this);
    //为预览面板赋值并进行设置
    panelPreview = new JPanel();
    panelPreview.setLayout(null);
    Icon icon = new ImageIcon("pictrue/pic_" + MyCanvas.pictureID + ".jpg");
    JLabel label = new JLabel(icon); //创建一个显示图片的面板
    label.setBounds(0, 0, 300, 300);
    panelPreview.add(label);
    //为上面面板赋值并进行设置
    panelNorth = new JPanel();
    panelNorth.setBackground(Color.red);
    panelNorth.add(start);           //添加 3 个按钮
    panelNorth.add(preview);
    panelNorth.add(set);
    myCanvas = new MyCanvas();       //为主面板赋值
    container.add(myCanvas, BorderLayout.CENTER);
    container.add(panelNorth, BorderLayout.NORTH);
    this.setTitle("拼图小游戏");     //设置标题
    this.setLocation(300, 200);
    this.setSize(308, 365);
    this.setResizable(false);
    this.setVisible(true);
    this.setDefaultCloseOperation(3); //设置关闭事件
}
public static void main(String[] args) {
    new MyMainFrame();               //创建 MyMainFrame 对象
}
//对 3 个按钮事件的处理
public void actionPerformed(ActionEvent arg0) {
    Button button = (Button) arg0.getSource(); //获取事件源对象
    if (button == start) {               //为游戏开始按钮
        myCanvas.Start();
    } else if (button == preview) {      //为预览按钮
        if (button.getLabel() == "预览") { //根据按钮的内容进行判断
            //由主面板转向预览面板
            container.remove(myCanvas);
            container.add(panelPreview);
            panelPreview.updateUI();
            container.repaint();
            button.setLabel("返回");
        } else {                         //由预览面板转向主面板
            container.remove(panelPreview);
            container.add(myCanvas);
            container.repaint();
            button.setLabel("预览");
        }
    } else if (button == set) {          //设置按钮

```

```

        Choice pic = new Choice();           //创建和设置选择框
        pic.add("小猫");
        pic.add("小猪");
        pic.add("云");
        pic.add("QQ");
        pic.add("卡通");
        pic.add("花");
        //创建一个对话框
        int i = JOptionPane.showConfirmDialog(this, pic, "选择图片",
            JOptionPane.OK_CANCEL_OPTION);
        if (i == JOptionPane.YES_OPTION) {
            MyCanvas.pictureID = pic.getSelectedIndex() + 1;
            myCanvas.reLoadPictrue();
            Icon icon = new ImageIcon("pictrue/pic_" + MyCanvas.pictureID
                + ".jpg");
            JLabel label = new JLabel(icon);    //设置面板
            label.setBounds(0, 0, 300, 300);    //设置面板的大小
            panelPreview.removeAll();           //移除预览面板中的内容
            panelPreview.add(label);            //添加label 面板
            panelPreview.repaint();             //重画
        }
    }
}

```

【代码解析】

在上述代码中, 由于该类是游戏的主界面, 所以继承了 `JFrame` 类。在该类的构造函数中主要实现界面的初始化, 然后在 `actionPerformed()` 方法中定义了 3 个按钮的处理事件。

17.3 小 结

本章主要通过综合 `Swing` 组件和 `Java` 语言中的事件处理机制, 来实现拼图游戏。在具体实现时首先创建了单元格的自定义按钮类 `Cell`, 然后创建了加载按钮类 `Cell` 的面板类 `MyCanvas`, 同时在该类中实现单选按钮的移动功能, 最后实现该项目的窗口类 `MyMainFrame`。

第4篇 文件操作和访问

- ▶▶ 第18章 文件属性查看器（GUI+文件操作）
- ▶▶ 第19章 文件内容查看器（GUI+文件访问）
- ▶▶ 第20章 日记簿（GUI+文件访问和操作）
- ▶▶ 第21章 查找和替换项目（GUI+字符串处理）

第 18 章 文件属性查看器（GUI+文件操作）

本章通过 Swing 组件实现文件属性查看器界面，通过文件的操作来获取文件的相关属性并显示在界面中。“文件属性查看器”项目的实现，综合了图形用户界面的相关知识点和文件的操作。

本章的学习目标如下：

- ❑ 掌握组件和面板的使用方法；
- ❑ 了解文件的操作；
- ❑ 熟悉文件操作和访问的类。

18.1 文件属性查看器原理

“文件属性查看器”项目通过单击“查看”按钮，打开显示文件或目录的属性表格，这些文件或目录的具体位置为文本框中表示地址的字符串。

18.1.1 项目结构框架分析

文件属性查看器项目可利用 Swing 组件实现图形用户界面。文件属性查看器项目目录如图 18.1 所示，各个包的功能如下。

- ❑ 类 FileAttrView：自定义窗口组件类。
- ❑ 类 FileAttrFrame：利用 Swing 组件实现界面。

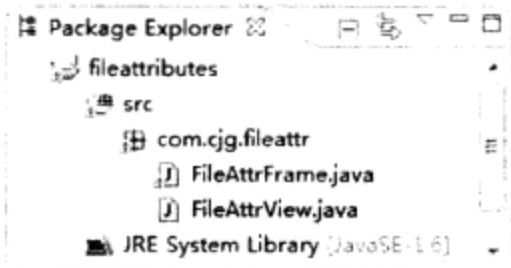


图 18.1 项目目录

18.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括文件属性查看器的初始化界面、查看已存在文件属性、查看已存在的目录属性，以及查看不存在的文件、目录属性和退出功能。

1. 初始化界面

当运行文件属性查看器项目中的 FileViewer 类后，会出现如图 18.2 所示的初始界面——文件属性查看器界面。



图 18.2 初始化界面

2. 查看已存在的文件属性

当出现初始化界面后，在“文件的地址”文本框中输入 D:\cjgong.txt 字符串（已存在的文件地址），然后单击“查看”按钮，主界面的中间会显示出该文件的所有属性信息，具体过程如图 18.3 所示。

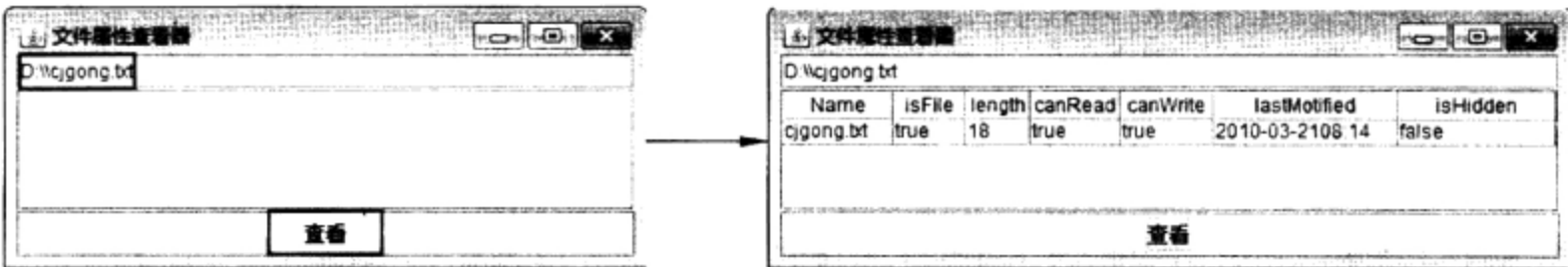


图 18.3 查看已存在的文件

3. 查看已存在的目录属性

当出现初始化界面后，在“文件的地址”文本框中输入 D:\cjgong 字符串（已存在的目录地址），然后单击“查看”按钮，主界面的中间会显示出该目录的所有属性信息，具体过程如图 18.4 所示。



图 18.4 查看已存在的目录

4. 查看不存在的文件和目录属性

当出现初始化界面后，在“文件的地址”文本框中输入 D:\test 字符串（不存在的目录地址），然后单击“查看”按钮，主界面的中间会显示出该目录的所有属性信息，具体过程如图 18.5 所示。

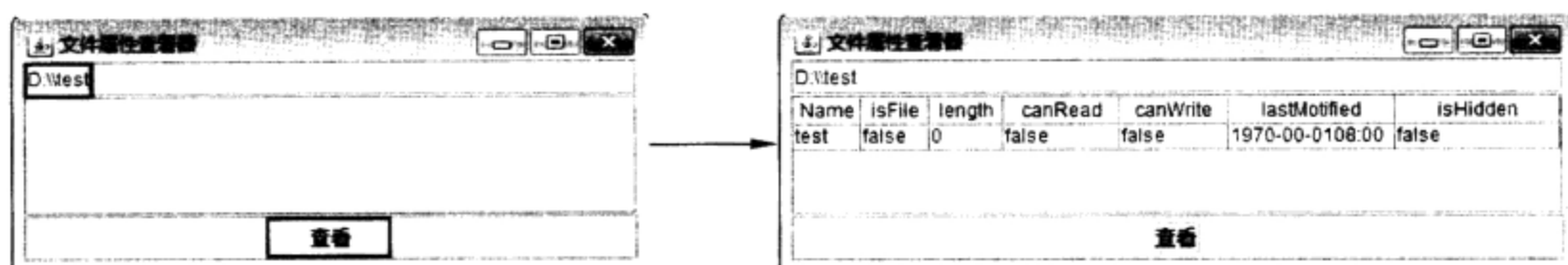


图 18.5 查看不存在的目录过程

5. 退出功能


当出现初始界面后, 如果想实现退出功能, 可以单击右上角的  按钮, 如图 18.6 所示。



图 18.6 退出功能

18.2 文件属性查看器项目

文件属性查看器项目具体程序架构如图 18.7 所示, 它包含一个“文件属性查看器输入界面”的自定义窗口类 `FileAttrFrame.java`, 以及自定义窗口显示位置的类 `FileAttrView.java`。

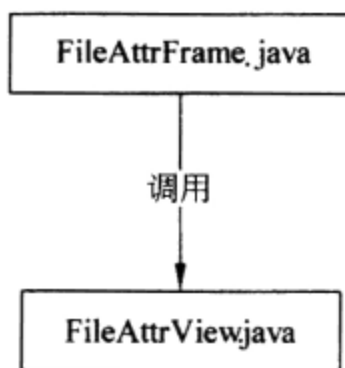


图 18.7 程序关系图

18.2.1 实现显示文件信息的自定义窗口

`FileViewer` 为“文件属性查看器”项目中的自定义窗口类, 该类不仅继承了 `JFrame` 类, 而且还实现了各个组件的相应功能, 具体内容如代码 18.1 所示。

代码 18.1 自定义窗口类: `FileAttrFrame.java`

```

public class FileAttrFrame extends JFrame {
    //创建成员变量
    private JPanel contentPane;
    private BorderLayout borderLayout1 = new BorderLayout();
    //创建布局管理器对象
  
```

```

private JTextField jTextField1 = new JTextField();           //创建文本域对象 jTextField1
private JScrollPane jScrollPane1 = new JScrollPane();        //创建滚动面板对象 jScrollPane1

private JTable jTable1;                                     //创建表格对象
private JButton jButton1 = new JButton();                   //创建按钮对象
File file;                                                  //创建文件对象
public FileAttrFrame() {                                    //构造函数
    //注册 window 事件
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try {
        jbInit();                                           //调用 jbInit() 方法
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void jbInit() throws Exception {                    //创建初始化方法
    contentPane = (JPanel) this.getContentPane();           //为 contentPane 赋值
    jTextField1.setText("文件的地址");                      //为文本域对象设置文本
    contentPane.setLayout(borderLayout1);                   //设置布局管理器
    this.setSize(new Dimension(394, 164));                 //设置大小
    this.setTitle("文件属性查看器");                        //设置标题
    jScrollPane1.setAutoscrolls(true);                      //设置滚动面板对象
    jButton1.setText("查看");                                //设置按钮文本
    //按钮的处理事件
    jButton1.addActionListener(new java.awt.event.ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            jButton1_actionPerformed(e);                    //调用相应的方法
        }
    });
    //添加各个组件到主窗口中
    contentPane.add(jButton1, BorderLayout.SOUTH);
    contentPane.add(jTextField1, BorderLayout.NORTH);
    contentPane.add(jScrollPane1, BorderLayout.CENTER);
}

//处理窗口的各种事件
protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);                            //处理窗口关闭方法
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
        System.exit(0);                                     //退出窗口
    }
}

Object[] getfileInfoType() {                               //创建文件属性的标题名字数组
    return new Object[] { "Name", "isFile", "length", "canRead",
        "canWrite", "lastMotified", "isHidden" };
}

Object[][] getFileInfo(File file) {                        //获取文件属性信息数组
    File theFile = file;                                    //创建 theFile 对象
    Object[][] data = new Object[1][7];                    //文件信息数组变量 data
    if (theFile.exists())
        return null;
    //用 File 类的各种方法获取文件属性为数组 data 赋值
    data[0][0] = theFile.getName();                        //文件的名字
    data[0][1] = String.valueOf(theFile.isFile());          //是否为文件
    data[0][2] = String.valueOf(theFile.length());          //文件的大小

```

```

        data[0][3] = String.valueOf(theFile.canRead());    //是否可读
        data[0][4] = String.valueOf(theFile.canWrite());  //是否可写
        data[0][5] = getDateString(theFile.lastModified()); //最后修改时间
        data[0][6] = String.valueOf(theFile.isHidden());  //是否隐藏属性
        return data;                                     //返回数组对象
    }
    public static String getDateString(long mill) {        //日期格式化方法
        if (mill < 0)
            return "";
        Date date = new Date(mill);
        Calendar rightNow = Calendar.getInstance();
        rightNow.setTime(date);
        int year = rightNow.get(Calendar.YEAR);
        int month = rightNow.get(Calendar.MONTH);
        int day = rightNow.get(Calendar.DAY_OF_MONTH);
        int hour = rightNow.get(Calendar.HOUR_OF_DAY);
        int min = rightNow.get(Calendar.MINUTE);
        return year + "-" + (month < 10 ? "0" + month : "" + month) + "-"
            + (day < 10 ? "0" + day : "" + day)
            + (hour < 10 ? "0" + hour : "" + hour) + ":"
            + (min < 10 ? "0" + min : "" + min);
    }
    void jButton1_actionPerformed(ActionEvent e) {        //按钮的事件监听器
        file = new File(this.jTextField1.getText());    //为 file 对象赋值
        //显示相应信息的表格
        this.jTable1 = new JTable(this.getFileInfo(file), this
            .getFileInfoType());
        jScrollPane1.getViewport().add(jTable1, null); //添加到滚动面板上
    }
}

```

【代码解析】

- 上述代码实现了文件属性查看器中的自定义窗口类，该用户界面涉及的具体容器、对象和布局如图 18.8 所示。

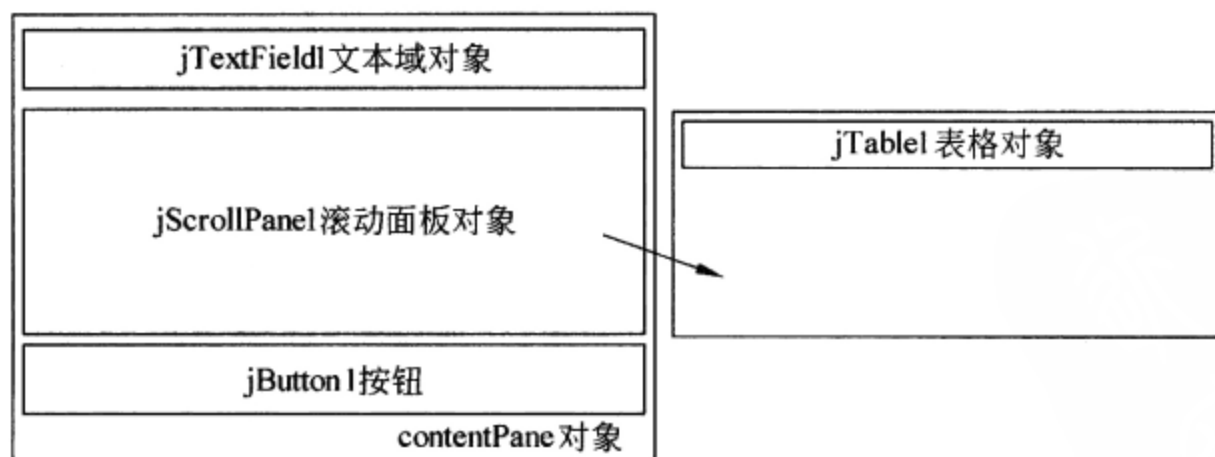


图 18.8 布局

- 在上述代码中存在一个处理按钮的事件监听器方法 `jButton1_actionPerformed()`，在该方法中首先通过获取到的地址创建一个文件对象 `file`，然后把该文件的所有信息通过 `getFileInfo()` 方法存储到数组 `data` 中并同时显示在表格组件 `jTable1` 中，最后把该表格组件显示在滚动面板 `jScrollPane1` 上。

18.2.2 自定义窗口的显示

在 18.2.1 节创建了自定义窗口类, 该窗口实现了“文件属性查看器”项目的全部功能。本节将讲解如何将自定义窗口在计算机屏幕上居中显示出来, 具体内容如代码 18.2 所示。

代码 18.2 实现窗口的显示: FileAttrView.java

```
public class FileAttrView {
    private boolean packFrame = false;           //创建布尔变量
    public FileAttrView() {                       //构造函数
        FileAttrFrame frame = new FileAttrFrame(); //创建 FileAttrFrame 对象
        if (packFrame) {                          //根据值来调用相应方法
            frame.pack();
        } else {
            frame.validate();
        }
        //用来实现居中显示
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreen-
        Size();                                     //获取屏幕大小
        Dimension frameSize = frame.getSize();    //获取窗口大小
        if (frameSize.height > screenSize.height) {
            frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
            frameSize.width = screenSize.width;
        }
        //设置窗口的位置
        frame.setLocation((screenSize.width - frameSize.width) / 2,
            (screenSize.height - frameSize.height) / 2);
        frame.setVisible(true);                   //显示窗口
    }
    public static void main(String[] args) {
        new FileAttrView();                       //创建 FileAttrView 对象
    }
}
```

【代码解析】

在上述代码中, 为了使自定义窗口居中显示, 首先获取计算机屏幕大小 `screenSize` 及自定义窗口的大小 `frameSize`, 然后通过表达式 $(screenSize.width - frameSize.width) / 2$ 和 $(screenSize.height - frameSize.height) / 2$ 来获取中心位置的坐标。

18.3 知识点扩展——文件的操作和访问

在 Java 语言中存在 I/O (Input/Output) 机制, 即输入和输出机制。通过 I/O 处理技术可以将数据保持到文本文件、二进制文件甚至 ZIP 压缩文件, 以达到数据永远保存的要求。

18.3.1 通过 FileOp 类实现文件创建和删除功能

在 Java 语言中, 目录是被当作一种特殊的文件来使用的。查看 API 帮助文档可以发现,

File 类是唯一代表磁盘文件对象的类。这里的文件本身不包含文件的内容部分。既然 **File** 类是文件的实体类，那么该类拥有了许多文件方面的属性和方法。

对于 **File** 类中提供的属性和方法，有一些是针对文件处理的，有一些是针对目录处理的，还有一些属于共用。由于该类的属性和方法太多，所以不能一一列举，下面通过 **FileOp** 类的实例，来讲解 **File** 类的一些主要属性和方法，具体内容如代码 18.3 所示。

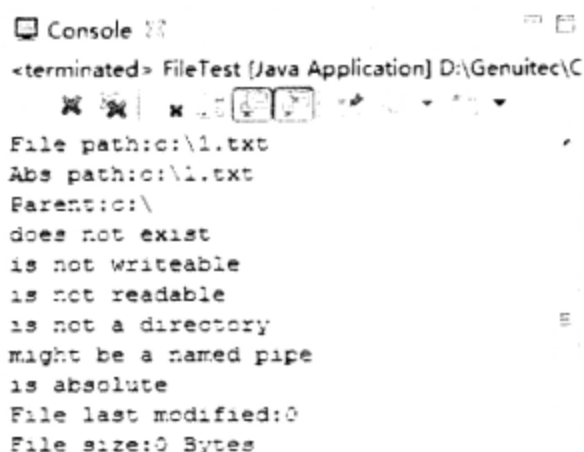
代码 18.3 文件的基本操作: FileOp.java

```
public class FileOp {
    public static void main(String[] args) {
        File f = new File("c:\\1.txt");           //创建文件对象
        if (f.exists())                           //判断文件是否存在
            f.delete();                           //如存在则删除
        else                                       //如不存在则创建
        {
            try {
                f.createNewFile();
            } catch (Exception e) {
                System.out.println(e.getMessage());
            }
        }
        System.out.println("File name:" + f.getName()); //输出文件的名称
        System.out.println("File path:" + f.getPath()); //输出文件的路径
        System.out.println("Abs path:" + f.getAbsolutePath()); //输出文件的绝对路径
        System.out.println("Parent:" + f.getParent()); //输出文件的父路径
        System.out.println(f.exists() ? "exists" : "does not exist"); //文件是否存在于磁盘中
        System.out.println(f.canWrite() ? "is writeable" : "is not writeable"); //是否可写
        System.out.println(f.canRead() ? "is readable" : "is not readable"); //是否可读
        System.out.println(f.isDirectory() ? "is " : "is not" + " a directory"); //是否是目录
        System.out.println(f.isFile() ? "is normal file" : "might be a named pipe"); //是否是文件
        //是否绝对路径
        System.out.println(f.isAbsolute() ? "is absolute" : "is not absolute");
        System.out.println("File last modified:" + f.lastModified()); //输出文件的最后修改时间
        System.out.println("File size:" + f.length() + " Bytes"); //输出文件的大小
    }
}
```

运行 **FileTest.java** 类，当文件 **c:\1.txt** 存在时，控制台窗口如图 18.9 所示；当文件不存在时，控制台窗口如图 18.10 所示。

【代码解析】

在上述代码中，首先创建了与文件 **1.txt** 相关联的 **File** 类对象 **f**，然后通过该对象的 **exists()** 方法判断 **1.txt** 文件是否存在。如果存在就通过该对象的 **delete()** 方法删除；否则通过该对象的 **createNewFile()** 方法实现该文件的创建。最后，通过对象的各种属性和方法输出所创建文件 **1.txt** 的各种属性。



```

Console
<terminated> FileTest [Java Application] D:\Genuitec\C
File path:c:\1.txt
Abs path:c:\1.txt
Parent:c:\
does not exist
is not writeable
is not readable
is not a directory
might be a named pipe
is absolute
File last modified:0
File size:0 Bytes

```

图 18.9 文件存在时控制台窗口显示



```

Console
<terminated> FileTest [Java Application] D:\Genuitec\C
File name:1.txt
File path:c:\1.txt
Abs path:c:\1.txt
Parent:c:\
exists
is writeable
is readable
is not a directory
is normal file
is absolute
File last modified:1270962460261
File size:0 Bytes

```

图 18.10 文件不存在时控制台窗口显示

18.3.2 通过 FileDir 类实现列举文件和目录的功能

Java 使用 FileDir 类来列举文件和目录，下面举例说明，如代码 18.4 所示。

代码 18.4 文件列举的基本操作: FileDir.java

```

public class FileDir {
    public static void main(String argv[]) {
        File f = new File(System.getProperty("user.dir")); //创建 File 对象
        File files[] = f.listFiles(); //获取 File 对象下的所有文件数组
        for (int i = 0; i < files.length; i++) { //遍历文件数组
            if (files[i].isDirectory()) //判断是否为目录
                System.out.print("<Dir>\t");
            else
                System.out.print("\t" + files[i].length()); //输出文件的大小
            System.out.println("\t" + files[i].getName()); //输出文件的名字
        }
    }
}

```

运行 FileDir.java 类，控制台窗口如图 18.11 所示，而用来列举文件的文件夹目录如图 18.12 所示。



```

Console
<terminated> FileDir [Java Application] D:\Genuitec\C
350 .project
301 .classpath
<Dir> src
<Dir> bin
<Dir> .settings

```

图 18.11 运行结果



图 18.12 文件夹目录

【代码解析】

在上述代码中，首先创建了与系统属性变量 user 相对应的文件夹相连的 File 对象 f，

然后通过该对象的 `listFiles()` 方法获取文件数组，最后遍历该数组，如果为文件类型，则输出文件的长度和名字；如果为目录，则输出目录名字。

18.3.3 File 类提供的属性和方法


通过 API 帮助文档可以查看 File 类提供的属性和方法，如下所示。

1. 构造函数

对于文件操作方法，首先从 File 类的构造方法说起，该类具有 3 种构造函数，它们的语法格式分别如下：

(1) File (String pathname)

参数 `pathname` 表示文件的路径名（包含文件名），该函数用于创建一个指定路径的文件。

 **注意：**对于参数 `pathname`，一般格式为“路径\文件名”格式，如果没有路径名称，则在程序运行的目录下创建文件。


(2) File (String path, String filename)

参数 `path` 表示文件的父路径名（不包含文件名），参数 `filename` 为文件的名称，该函数用于创建一个指定路径和名称的文件。

(3) File (File parent, String filename)

参数 `parent` 表示文件的父路径，参数 `filename` 为文件的名称（包含子路径），该函数用于创建一个指定路径和名称的文件。

在 Windows 系统中目录的分隔符号为“\”（separator），而在其他系统中却不是这样，例如 Linux 系统中是“/”。为了写出通用的程序，在具体设置目录时，首先需要通过 `System.getProperty("os.name")` 获取操作系统，然后通过 File 类中分隔符的常量具体设置。目录和路径要区分开，目录里的是目录分隔符，而路径里的是路径分隔符，分别对应于 File 类中的 4 个常量，即 `separator`、`separatorChar` 和 `pathseparator`、`pathseparatorChar`。

 **注意：**`separator` 是 String 类型，而 `separatorChar` 是 Char 类型。同理，`pathseparator` 是 String 类型，而 `pathseparatorChar` 是 Char 类型。

2. 路径的方法

在构造函数中存在一个重要的参数，即路径的参数。当获取 File 类对象后，如何获取路径的属性呢？查看 API 帮助文档，可以发现如下方法。

- ☐ `getName()` 方法：获取与 File 对象相连接的文件或目录的名称（不包含路径名称）。
- ☐ `getPath()` 方法：获取与 File 对象相连接的文件或目录的名称（包含路径名称）。
- ☐ `getAbsolutePath()` 方法：获取与 File 对象相连接的文件或目录的绝对路径名称。
- ☐ `getParent()` 方法：获取与 File 对象相连接的文件或目录的父路径名称。
- ☐ `isAbsolute()` 方法：判断与 File 对象相连接的文件或目录的父路径是否绝对路径。

例如创建一个 File 类对象, 该程序是在 D:\Java 目录里运行, 具体代码如下:

```
File test=new File("test","testfile.text");
```


对于对象 test, 调用 getAbsolutePath()方法将返回 D:\Java\test\testfile.text 绝对路径; 调用 getPath()方法则返回 test\testfile.text 绝对路径; 而调用 getName()方法和 getParent()方法将分别返回 testfile.text 和 testfile.text 路径。有时还需要判断 File 类对象是否使用了绝对路径, 这时就需要通过 isAbsolute()方法来判断, 对于 test 对象将返回 false, 即不是绝对路径。如果修改 test 对象如下, 则返回 true (使用了绝对路径)。

```
File test=new File("D:\\Java\\test\\testfile.text");
```

3. 操作文件和目录——创建文件方法

如果想创建一个文件, 首先需要判断该文件是否存在, 然后才能创建。查看 API 帮助文档可以发现如下方法。


- ❑ exists()方法: 检查与 File 对象相连接的文件和目录是否存在于磁盘中。
- ❑ createNewFile()方法: 如果与 File 对象相连接的文件不存在, 则创建一个空文件。
- ❑ createTempFile()方法: 创建一个 File 对象并同时在磁盘上创建指定的文件。

 注意: createNewFile()方法通常与 exists()方法相配合使用, 而 createTempFile()方法是一个类方法, 其参数也具有特殊的规定。

4. 操作文件和目录——创建目录方法

如果想创建一个文件, 首先需要判断该文件是否存在, 然后才能创建。查看 API 帮助文档可以发现如下方法。


- ❑ mkdir()方法: 创建与 File 对象相连接的目录名称。
- ❑ mkdirs()方法: 创建与 File 对象相连接的目录名称, 如果父目录不存在, 系统会自动生成。

 注意: 上述两个方法的区别在于, 如果要创建 D:\Java\test 这个目录, 但是 D:\Java 不存在, 这时如果用 mkdir()方法创建, 则不会成功; 用 mkdirs()方法创建会成功。

5. 操作文件和目录——删除方法

如果想删除一个文件和目录, 首先需要判断该对象是目录还是文件, 然后才能删除。查看 API 帮助文档可以发现如下方法。

- ❑ isDirectory()方法: 检查与 File 对象相连接的对象是否为目录。
- ❑ isFile()方法: 检查与 File 对象相连接的对象是否为文件。
- ❑ delete()方法: 删除与 File 对象相连接的文件和目录。
- ❑ deleteOnExit()方法: 删除与 File 对象相连接的文件和目录, 其不会立即运行, 而是在整个程序结束时才会被执行。

 注意: 在删除之前之所以要判断是目录还是文件, 因为如果是目录, 则需要判断目录下是否有文件和子目录, 只有没有文件或子目录的情况下才可以正常删除。

6. 操作文件和目录——列举方法

有时需要实现列举一个目录下的所有子目录和文件的功能，查看 API 帮助文档可以发现如下方法。

- ❑ `list` 方法：返回与 `File` 对象相连接的目录下的所有子目录和文件。
- ❑ `listFile()` 方法：返回与 `File` 对象相连接的目录下的所有文件。
- ❑ `listRoots()` 方法：返回与 `File` 对象相连接的对象所属的根目录，即磁盘符号。

7. 文件的属性

如果想删除一个文件和目录，首先需要判断该对象是目录还是文件，然后才能删除。查看 API 帮助文档可以发现如下方法。

- ❑ `canRead()` 方法：判断与 `File` 对象相连接的文件是否可以读取里面的数据。
- ❑ `canWrite()` 方法：判断与 `File` 对象相连接的文件是否可以写入数据。
- ❑ `isHidden()` 方法：判断与 `File` 对象相连接的文件和目录是否隐藏。
- ❑ `length()` 方法：返回与 `File` 对象相连接的文件的大小。
- ❑ `lastModified()` 方法：返回与 `File` 对象相连接的文件最后修改时间。
- ❑ `setLastModified()` 方法：设置与 `File` 对象相连接的文件最后修改时间。

18.3.4 文件访问的基本概念

如果想彻底理解文件的访问功能，必须先理解 `Stream`（流）这个概念。当用水管浇花、洗澡等时，水是从水龙头经水管的一端流向另一端，而且只要水龙头不断的有水送出来，那么水就会接连不断地经水管流向另一端。这就是所谓的流概念。

文件的访问就是以上述概念来处数据的输入和输出的，`InputStream` 和 `OutputStream` 是所有以字节为单位的输入和输出类的父类，而 `Reader` 和 `Writer` 是所有以字符为单位读取和输出类的父类。下面详细讲解这 4 个类的基础知识。

1. `InputStream` 类

`InputStream` 是用来实现输入字节的类，即读取数据的意思。如果要想实现读取功能，就需要用到 `read()` 方法，该类提供了 3 种重载的方法，分别如下。

```
abstract int read()
```

该函数无任何参数，用来实现一次读取一个字节的的数据，并以 `int` 类型返回读取的数据。如果没有数据，将返回 -1。

```
int read(byte[] b)
```

参数 `b` 为缓冲区数组，该函数用来实现一次读取一个字节的的数据，读取的数据会存储到缓冲区数组里，其返回的是真正读取的字节数目。

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

参数 `off` 为缓冲区数组的起始位置，参数 `len` 为一次要读取多少个字节数组。该函数用


法与第 2 种重载方法一样。

由于有些数据源一次只能允许一个流来传送数据, 所以如果用完了 `InputStream` 对象, 则通过 `close()` 方法来关闭对象。

如果想知道流中还有多少个字节的数据, 可以通过 `available()` 方法来实现, 如下所示。

```
int available()
```

该方法会返回一个 `int` 类型数据, 表示还有多少个字节的数据可以读取。

 **注意:** 如果用 `InputStream` 对象调用该方法, 只会返回 0。该方法必须由继承 `InputStream` 的子类对象调用才能返回正确的数据。

如果想跳过流中的某些数据, 可以通过 `skip()` 方法来实现, 如下所示。

```
long skip(long n)
```

参数 `n` 表示要跳过几个字节的数据, 该函数用来实现跳过指定字节的数据, 返回真正跳过的字节数据。


对于流中的数据, 如果想返回重新读取, 可以先通过 `mark()` 方法设置回头的位置, 然后通过 `reset()` 方法使流返回到设置的位置处。

```
void mark(int readlimit)
```

参数 `readlimit` 为返回的位置, 在此输入流中标记指定的位置。

```
void reset()
```

该函数用来实现将流重新定位到对输入流最后调用 `mark()` 方法时的位置。

 **注意:** 对于上述两个方法, 不是每个 `InputStream` 子类都可以使用的, 所以需要通过 `markSupported()` 方法来判断类是否支持这两个类。

2. OutputStream类

`OutputStream` 是用来实现输出字节的类, 即写入数据的意思。如果要实现写入功能, 就需要用到 `write()` 方法, 该类提供了 3 种重载的方法, 分别如下。

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

参数 `b` 为字节的数目, 该函数用来实现把 `b.length` 个字节写入输出流。

```
void write (byte[] b)
```

参数 `b` 为指定的字节数组, 该函数用来实现把指定的字节数组写入输出流。

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

参数 `off` 为缓冲区数组的起始位置, 参数 `len` 为一次要读取多少个字节数组。该函数用法与第 2 种重载方法一样。


由于有些数据源一次只能允许一个流来传送数据, 所以如果用完 `OutputStream` 对象后, 同样需要通过 `close()` 方法来关闭对象。

当用 `write()` 方法输出数据时, 这些数据并不会马上输出到指定的目的文件里, 而是先

存储到内存的缓冲中。如果想把数据立即输出到目的文件中，可以调用 `flush()` 方法，该方法的语法如下：


```
void flush()
```

该函数用来实现刷新输出流并强制输出缓冲中的所有字节。

 **注意：**对于输出流，在调用 `close()` 方法前，一般会先调用 `flush()` 方法确保把所有的数据都输出到目的文件里。

3. Reader类

`Reader` 是用来实现输入字符的类，即读取字符数据的意思。与 `InputStream` 类相比，`Reader` 类操作的对象为 `char` 类型，而 `InputStream` 类为 `byte` 类型。所以只有把 `InputStream` 类提供的方法中，在用到 `byte` 类型的地方改为 `char` 类型就可以被使用。

 **注意：**在 `Reader` 类中用 `ready()` 方法代替 `InputStream` 类中的 `available()` 方法，表示已经准备好输入数据。

4. Writer类

`Writer` 是用来实现输出字符的类，即写入字符数据的意思。与 `OutputStream` 类相比，`Writer` 类操作的对象为 `char` 类型，而 `OutputStream` 类为 `byte` 类型。所以只有把 `OutputStream` 类提供的方法中，在用到 `byte` 类型的地方改为 `char` 类型就可以被使用。对于 `write()` 方法，`Writer` 类比 `OutputStream` 类多了两个重载的方法，分别如下：

```
void write(String str)
```

参数 `str` 为写入的字符串，该函数用来实现把指定的字符串写入输出流。

```
void write(String str, int off, int len)
```

参数 `off` 为写入字符串的起始位置，参数 `len` 为一次要读取多少个字符。该函数用法与第2种重载方法一样。

18.3.5 文件的基本访问方式——字节方式

18.3.4 节讲解了文件的操作方法和文件访问的父类，本节将详细讲解如何实现对文件的访问。查看 API 帮助文档，可以发现有 3 种方式来访问文件，本节将详细讲解以字节方式（类 `FileInputStream` 和 `FileOutputStream`）访问文件的内容。

下面通过实现文件访问功能的 `ByteAccess` 类来讲解 `FileInputStream` 和 `FileOutputStream` 类的基本用法，具体内容如代码 18.5 所示。

代码 18.5 文件的访问：ByteAccess.java

```
public class ByteAccess {
    //创建成员变量
    public File file = new File("C:\\", "Byte.txt");
    public byte bytes[] = new byte[512];
    public static void main(String args[]) {           //主函数
```



```

ByteAccess test = new ByteAccess();           //创建 ByteAccess 方式
test.writemethod();                           //调用写入方法
test.readmethod();                           //调用读取方法
}
public void writemethod() {                   //写入方法
    int b;                                   //定义一个变量
    System.out.println("请输入存入文本的内容:"); //输出相应提示信息
    try {
        if (!file.exists())                 //判断文件是否存在
            file.createNewFile();           //创建新文件
        b = System.in.read(bytes);          //把从键盘输入的字符存入 bytes 里
        //创建文件输出流
        FileOutputStream fos = new FileOutputStream(file, true);
        fos.write(bytes, 0, b);              //把 bytes 写入到指定文件中
        fos.close();                         //关闭输出流
    } catch (IOException e) {
        e.printStackTrace();
    }
}
public void readmethod() {                   //读取方法
    try {
        FileInputStream fis = new FileInputStream(file);
        //创建文件字节输入流
        int rs = 0;                          //创建变量 rs
        System.out.println("开始读取文件内容:"); //输出相应提示
        while ((rs = fis.read(bytes, 0, 512)) > 0) { //读取文件内容
            //在循环中读取输入流的数据
            String s = new String(bytes, 0, rs);
            System.out.println(s);
        }
        fis.close();                         //关闭输入流
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

运行 ByteAccess.java 类之前, C 盘里 Byte.txt 文件的内容如图 18.13 所示, 按照如图 18.14 所示的运行过程运行 ByteAccess 类后, C 盘里 Byte.txt 文件的内容如图 18.15 所示。

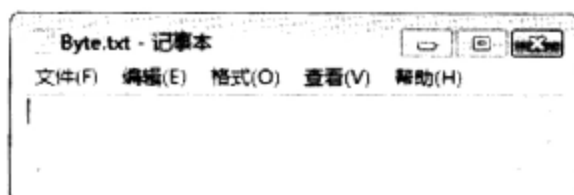


图 18.13 程序运行前的文件



图 18.14 运行过程

【代码解析】

- 在写入数据的方法 `writemethod()` 中，首先判断指定的文件是否存在，如果不存在，则调用 `createNewFile()` 方法进行创建，然后获取从键盘中输入字符，最后通过创建文件输出流的 `write()` 方法，把字符字节数组输出到文件里。
- 在读取数据的 `readmethod()` 方法中，首先创建与文件相关联的读取流，然后在循环中通过文件流的 `read()` 方法把文件中的内容读取到字节数组中，最后再输出字节数组中的内容。

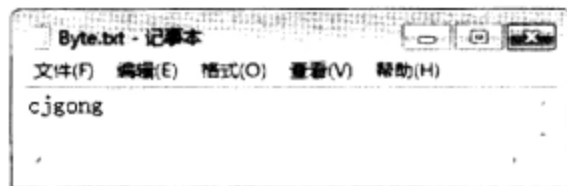


图 18.15 程序运行后的文件

18.3.6 文件的基本访问方式——字符方式

18.3.5 节讲解了如何通过字节方式来访问文件，本节将详细讲解如何通过字符方式来访问文件，即通过 `FileInputStream` 和 `FileOutputStream` 类实现对文件的访问。

下面通过实现文件访问功能的 `CharAccess` 类，来讲解 `FileReader` 和 `FileWriter` 类的基本使用方法，具体内容如代码 18.6 所示。

代码 18.6 文件的访问: CharAccess.java

```
public class CharAccess {
    //创建成员变量
    File filein = new File("C:\\char.txt"); //创建原文件对象 filein
    File fileout = new File("C:\\char-1.txt"); //创建复制后的文件对象 fileout
    public char chars[] = new char[512]; //创建字符数组 chars
    public static void main(String args[]) { //主函数
        CharAccess test = new CharAccess (); //创建 CharAccess 对象
        test.readmethod(); //调用读取方法
        test.writermethod(); //调用写入方法
    }
    public void writermethod() { //复制一个文件的内容到另一个文件中
        try {
            //创建新文件
            if (!fileout.exists()) //如果文件不存在
                fileout.createNewFile();
            //创建源文件的读取流和目的文件的写入流
            FileReader fin = new FileReader(filein);
            FileWriter fos = new FileWriter(fileout);
            int is; //变量
            while ((is = fin.read()) != -1) { //遍历读取流
                fos.write(is);
            }
            fin.close(); //关闭读取流
            fos.close(); //关闭写入流
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public void readmethod() { //读取方法
        try {
            if (!filein.exists()) //如果文件不存在
```

```

        filein.createNewFile();           //创建新文件
        FileReader fin = new FileReader(filein); //创建源文件的读取流
        int is;                           //创建变量
        while ((is = fin.read(chars)) != -1) { //遍历读取
            String str = new String(chars, 0, is); //创建字符串
            //输出字符串
            System.out.println("char 文件的内容为: "+str);
        }
        fin.close();                       //关闭读取流
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

打开 C 盘可以发现 char.text 文件, 该文件的内容如图 18.16 所示。运行 CharAccess.java 类, 控制台窗口如图 18.17 所示。这时如果再次打开 C 盘, 可以发现多出了一个名为 char-1.text 的文件, 该文件的内容如图 18.18 所示。

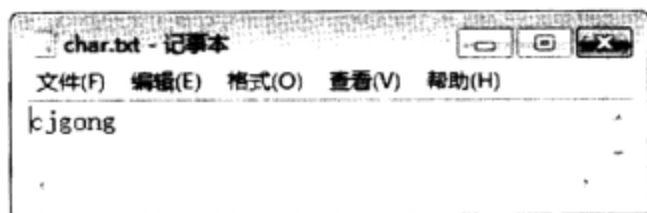


图 18.16 char.text 文件的内容



图 18.17 控制台窗口

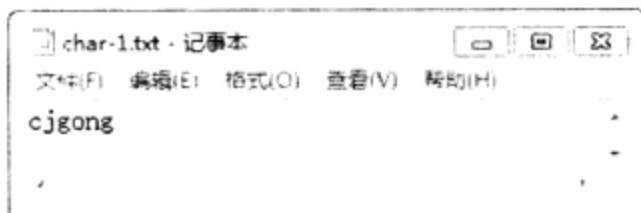


图 18.18 char-1.text 文件的内容

【代码解析】

- ❑ 在读取数据的 readmethod()方法中, 首先创建与文件相关联的读取流, 然后在循环中通过文件流的 read()方法把文件中的内容读取到字符数组中, 最后再输出字符数组组成的字符串内容。
- ❑ 在写入数据的 writemethod()方法中, 首先判断源文件和目的文件是否存在, 如果不存在, 则调用 createNewFile()方法进行创建, 然后获取该文件的输入流和输出流, 最后在循环中把读取流中的字符通过输入流中的 write()方法写入目的文件中, 从而达到从源文件复制目标文件的效果。

18.3.7 文件的高级访问方式

18.3.6 节讲解了文件访问的两种基本方式, 本节将详细讲解文件访问的最后一种方式——RandomAccessFile 类的方式。查看 API 帮助文档, 可以发现 RandomAccessFile 类可以实现对文件更高级的访问。下面将通过一个具体实例来讲解, 具体步骤如下。

(1) 下面通过实现文件访问功能的 RandomAccess 类来讲解 RandomAccessFile 类的基本使用方法, 具体内容如代码 18.7 所示。

代码 18.7 文件的访问: RandomAccess.java

```
public class RandomAccess {
    public static void main(String[] args) throws Exception {
        //创建 3 个雇员变量
        Employee e1 = new Employee("zhangsan", 23);
        Employee e2 = new Employee("cjgong", 24);
        Employee e3 = new Employee("Wangwu", 25);
        //创建一个文件的 RandomAccessFile 对象
        RandomAccessFile ra = new RandomAccessFile("c:\\Random.txt", "rw");
        //把 3 个雇员的信息写入文件里
        ra.write(e1.name.getBytes());
        ra.writeInt(e1.age);
        ra.write(e2.name.getBytes());
        ra.writeInt(e2.age);
        ra.write(e3.name.getBytes());
        ra.writeInt(e3.age);
        ra.close(); //关闭 ra 流对象
        //创建一个文件的 RandomAccessFile 对象
        RandomAccessFile raf = new RandomAccessFile("c:\\Random.txt", "r");
        int len = 8; //名字长度变量
        //跳过第一个员工的信息, 其中姓名 8 字节, 年龄 4 字节
        raf.skipBytes(12); //跳过 12 字节
        System.out.println("第二个员工信息: ");
        String str = ""; //雇员信息变量
        //通过循环为 str 赋值
        for (int i = 0; i < len; i++)
            str = str + (char) raf.readByte();
        System.out.println("name:" + str.trim()); //输出雇员名字变量的值 str
        System.out.println("age:" + raf.readInt()); //输出雇员的年龄
        //输出第一个员工的信息
        raf.seek(0); //将文件指针移动到文件开始位置
        System.out.println("第一个员工的信息: ");
        str = "";
        for (int i = 0; i < len; i++)
            str = str + (char) raf.readByte();
        System.out.println("name:" + str.trim());
        System.out.println("age:" + raf.readInt());
        //输出第三个员工的信息
        raf.skipBytes(12); //跳过第二个员工的信息
        str = "";
        System.out.println("第三个员工的信息: ");
        for (int i = 0; i < len; i++)
            str = str + (char) raf.readByte();
        System.out.println("name:" + str.trim());
        System.out.println("age:" + raf.readInt());
        raf.close(); //关闭流对象 raf
    }
}
```

打开 C 盘可以发现 Random.txt 文件, 该文件的内容如图 18.19 所示。运行 RandomAccess.java 类, 控制台窗口如图 18.20 所示。这时如果再次打开 C 盘的 char.txt 文件, 该文件的内容如图 18.21 所示。

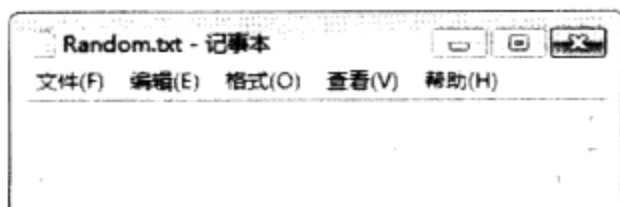


图 18.19 程序运行前的文件

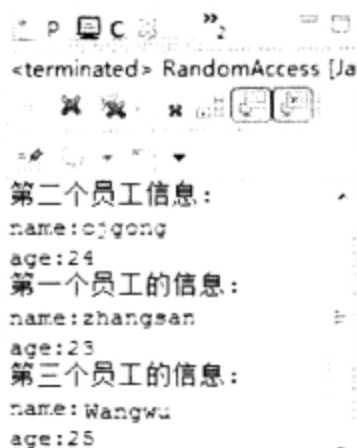


图 18.20 控制台窗口

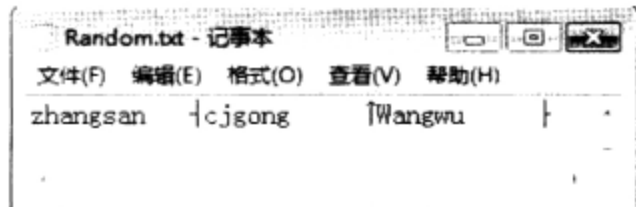


图 18.21 程序运行后的文件

【代码解析】

- ❑ 对于文件 Random.txt, 首先通过与其关联的 RandomAccessFile 类型对象 ra 把雇员信息按照“第一雇员、第二雇员和第三雇员”的顺序写入文件。然后通过与其关联的 RandomAccessFile 类型对象 raf 把雇员信息按照“第二雇员、第一雇员和第三雇员”的顺序读取并输出到控制台窗口。
- ❑ 在具体写信息到文件的过程中, 首先创建一个“读写”方式的 RandomAccessFile 类型对象 ra, 然后通过 write()方法把所有“雇员”的姓名(name)信息写入文件中, writeInt()方法把所有“雇员”的年龄(age)信息写入文件中。
- ❑ 在具体从文件读取信息的过程中, 需要创建一个“只读”方式的 RandomAccessFile 类型对象 raf。对于第二个雇员信息的读取, 首先需要通过 skipBytes()方法跳过第一个雇员的信息, 然后在遍历过程中通过 readByte()方法把姓名(name)信息读出, 最后通过 readInt()方法把年龄(age)信息读出。对于第一个雇员信息的读取, 首先需要通过 seek()方法移动到文件的开始位置, 然后在遍历过程中通过 readByte()方法把姓名(name)信息读出, 最后通过 readInt()方法把年龄(age)信息读出。对于第三个雇员信息, 同读取第二个雇员的过程基本一样, 只是通过 skipBytes()方法跳过第一个和第二个雇员的信息。

(2) 创建一个封装“雇员”信息名为 Employee 的类, 该类的具体内容如代码 18.8 所示。

代码 18.8 雇员的类: Employee.java

```
class Employee {
    String name;                //雇员的名字
    int age;                    //雇员的年龄
    final static int LEN = 8;   //雇员名字长度的常量
    public Employee(String name, int age) { //构造函数
        if (name.length() > LEN) { //当名字长度大于 8 个字符时
            name = name.substring(0, 8); //截取前 8 个字符
        } else { //当名字长度小于 8 个字符时
            while (name.length() < LEN) //用空格填补成 8 个字符
                name = name + "\u0000";
        }
        //为成员变量赋值
        this.name = name;
        this.age = age;
    }
}
```

【代码解析】

- 由于一个“雇员”信息就是文件中的一条记录，而对于记录来说需要保证其大小相同，因此每个“雇员”的姓名（name）和年龄（age）属性值在文件中的长度是一样的。
- 对于年龄（age）属性，由于其是 int 类型，所以每个成员的年龄（age）属性值长度相同。对于姓名（name）属性，由于其是 String 类型，即长度不确定，所以需要在构造函数中通过判断语句对其值进行操作。当长度大于 8 个字符时，则截取前 8 个字符；当长度小于 8 个字符时，则补空格（\u0000）。

(3) 通过 API 帮助文档来查看，以高级方式访问文件涉及的 RandomAccessFile 类的一些基础知识，分别如下：

类 RandomAccessFile 存在两种构造函数，分别如下所示。

RandomAccessFile(File file, String mode)

参数 file 表示文件对象，参数 mode 为操作文件的方式，该函数用于创建一个与指定文件对象相关联的写入或读取数据的随机存取文件流。

RandomAccessFile(String name, String mode)

参数 name 表示文件的路径（包含文件名），该函数用于创建一个与指定文件对象相关联的写入或读取数据的随机存取文件流。

上述构造函数中的参数 mode 表示访问文件的权限，其取值如表 18.1 所示。

表 18.1 mode 的值

值	意 义
r	只读方式
rw	可读写
rws	同步写入
rwd	更新同步写入

18.4 小 结

本章主要通过对文件的操作来获取文件的属性，并把相应的文件属性内容显示在相应的 Swing 组件里。虽然设计图形用户界面很重要，但是实现图形用户界面中组件的互访更重要。

在本章的最后还详细介绍了文件操作和访问的基础知识，对于文件的访问介绍了 3 种方式，分别为通过字节方式访问、通过字符方式访问和通过高级方式访问。

第 19 章 文件内容查看器(GUI+文件访问)

本章将通过 Swing 组件实现文件属性查看器界面，通过对文件的访问获取文件里的内容并显示在界面中。“文件内容查看器”项目的实现综合了图形用户界面的相关知识和文件访问的相关知识。

本章的学习目标如下：

- ❑ 掌握组件和面板的使用方法；
- ❑ 掌握文件内容查看器项目；
- ❑ 熟悉文件访问的各种类和方法。

19.1 文件内容查看器原理

“文件内容查看器”项目通过“文件”菜单和列表框来实现对文件的选择，并把所选文件的内容显示在文本框中。

19.1.1 项目结构框架分析

对于文件内容查看器，虽然内容很简单，但是实现的功能却非常典型。在具体实现时，利用 Swing 组件实现图形用户界面，该项目目录如图 19.1 所示。

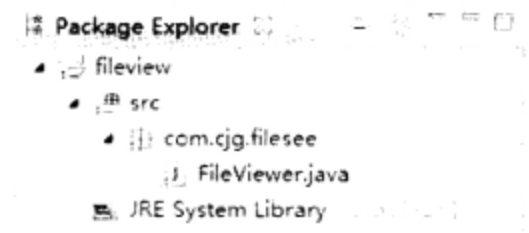


图 19.1 项目目录

19.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括文件内容查看器的初始化界面、通过菜单打开文件的功能、通过下拉列表框中的选项打开文件的功能和退出功能。

1. 初始化界面

当运行文件内容查看器项目中的 FileViewer 类后，会出现如图 19.2 所示的初始界面——文件内容查看器界面。

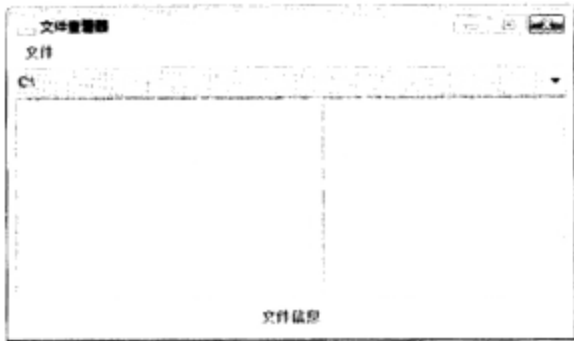


图 19.2 初始界面

2. 通过菜单打开文件的功能

当出现初始界面后，为了实现打开文件的功能，首先通过选择“文件”|“打开”命令会弹出“打开”对话框。然后在该对话框中选择需要打开的文件目录，最后单击“打开”按钮就会使该对话框消失，并在主界面的右边文本域中显示出文件的内容，具体过程如图 19.3 所示。

3. 通过下拉列表框中的选项打开文件的功能

当出现初始界面后，为了实现打开文件功能，首先通过选择下拉列表框中的盘符就会在主界面的左边文本域中显示出该盘符里的所有文件夹和文件。然后在左边文本域中选择需要打开的文件，最后单击选择的文件，在右边文本域中就会显示出文件的内容，具体过程如图 19.4 所示。

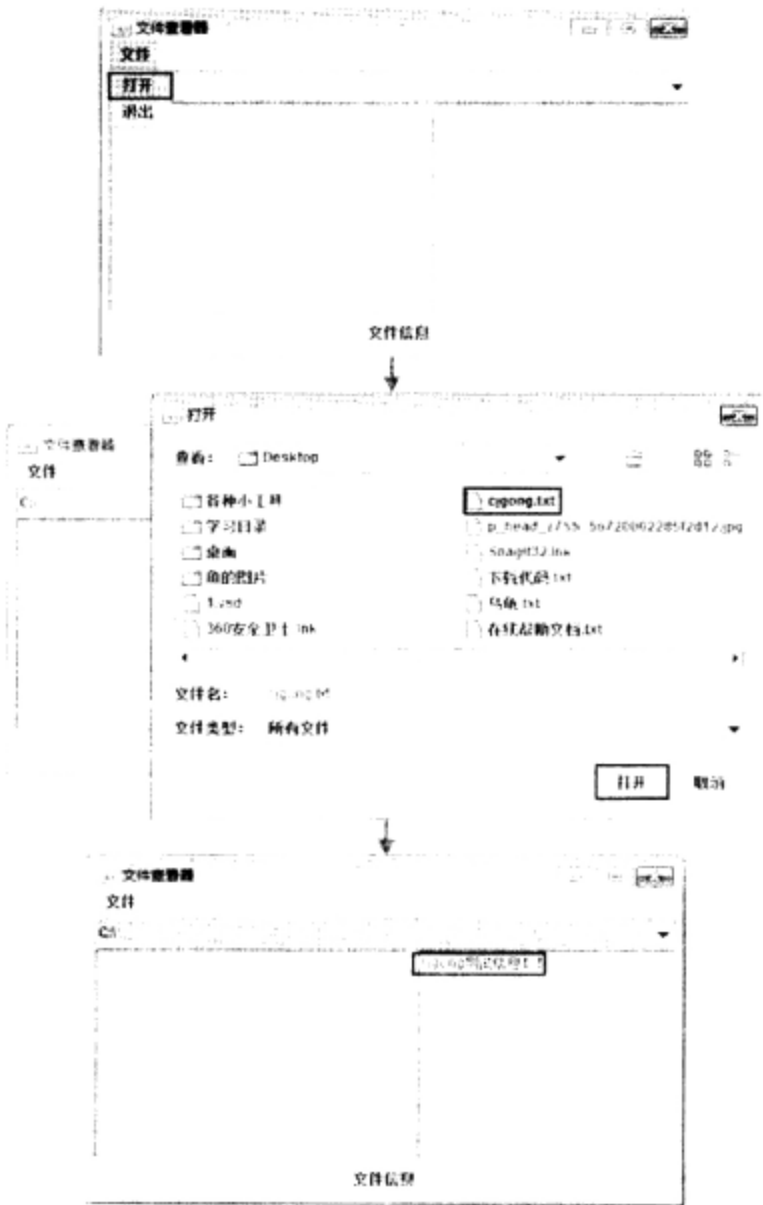


图 19.3 打开文件的过程

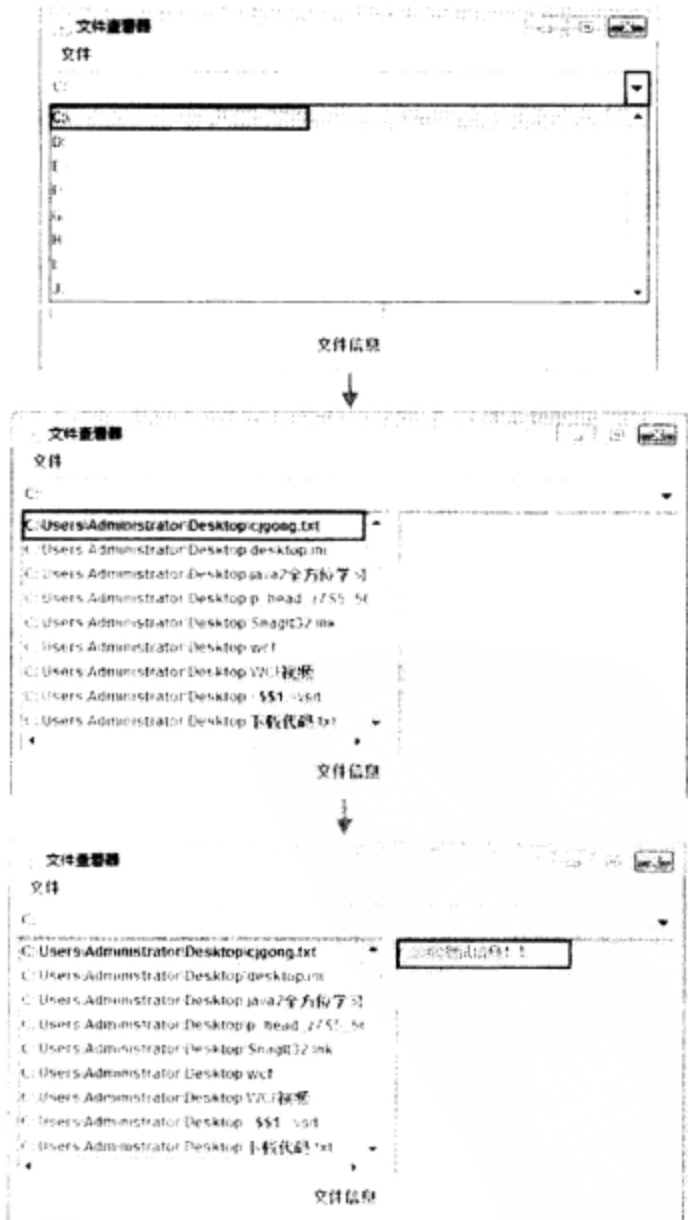


图 19.4 打开文件的过程

4. 退出功能


当出现初始界面后，如果想实现退出功能，可以选择菜单“文件”|“退出”命令，或单击窗口右上角的按钮，如图 19.5 所示。



图 19.5 退出功能

19.2 文件内容查看器项目

文件内容查看器项目内容非常简单，整个项目中只有一个名叫 FileViewer 的类，该类的类图如图 19.6 所示。对于该类的实现，主要通过两种方式选择所要查看内容的文件，并把文件的内容显示在文本框中。



图 19.6 FileViewer 类图

19.2.1 设计项目的界面——文件内容查看器输入界面

FileViewer 为文件内容查看器项目界面的类，本节主要讲解如何实现该项目的界面，具体内容如代码 19.1 所示。

代码 19.1 主界面: FileViewer.java

```

public class FileViewer {
    //创建成员变量
    JFrame frame = null;
    File[] s = new File[7];
    JList list;
    JTextArea textArea;
    JLabel jlbXinxi;
    JPanel panel;
    JScrollPane jpnsroll;
    JScrollPane jpnsrollTxt;
    File dir;
    File[] fileList;
    JComboBox combo;

    //创建主界面对象
    //创建文件数组对象(盘符)
    //创建用来显示盘符下的文件的列表框
    //文本编辑区
    //显示文件信息
    //JLabel 的容器
    //JList 容器的滚动面板
    //textArea 容器的滚动面板
    //带盘符根目录的 File 实例
    //某路径下的所有文件(抽象路径名)
    //下拉式菜单变量

    JMenuBar jMenuBar1 = new JMenuBar(); //创建菜单栏对象 jMenuBar1
    JMenu jMenu1 = new JMenu(); //创建菜单对象 jMenu1
    JMenuItem jMenuItem19 = new JMenuItem(); //创建菜单对象 jMenuItem19
    JMenuItem jMenuItem2 = new JMenuItem(); //创建菜单选项 jMenuItem2
    JFileChooser jFileChooser1 = new JFileChooser(); //打开文件对话框
    private File file = null; //文件对话框返回的对象

    public FileViewer() //构造方法, 实现界面
    {
        frame = new JFrame("文件内容查看器");
        Container contentPane = frame.getContentPane(); //获得容器对象
        contentPane.setLayout(new BorderLayout()); //设置布局管理器
        //菜单的设置
        frame.setJMenuBar(jMenuBar1); //添加菜单栏对象 jMenu1
        jMenu1.setText("文件"); //设置菜单对象 jMenu1 的名字
        jMenuItem19.setText("打开"); //设置菜单选项 jMenuItem19 的名字
        //“打开”菜单选项单击事件
        jMenuItem19.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                ... //省略部分代码
            }
        });
        //“退出”菜单选项单击事件
        jMenuItem2.setText("退出"); //设置菜单选项 jMenuItem2 的名字
        jMenuItem2.addActionListener(new ActionListener() {
            ... //省略部分代码
        });
        jMenuBar1.add(jMenu1); //添加菜单对象到菜单工具栏中
        jMenu1.add(jMenuItem19); //添加菜单选项到菜单中
        jMenu1.add(jMenuItem2); //添加菜单选项到菜单中
        //设置主窗口的北面部分
        //“下拉式菜单”的设置
        File file = new File("我的电脑"); //创建文件对象
        s = file.listFiles(); //获得盘符
        //把各盘符(文件数组)作为组合框的数据项
        combo = new JComboBox(s);
        contentPane.add(combo, BorderLayout.NORTH); //添加到主窗口的北面部分
        //设置主窗口的中间部分
        list = new JList(); //为 list 对象
        textArea = new JTextArea(); //文本编辑区
    }
}

```

```

jpnsroll = new JScrollPane(list);           //把 list 加入滚动面板
jpnsrollTxt = new JScrollPane(textArea);    //把 textArea 加入滚动面板
//拆分面板的设置
JSplitPane jsp = new JSplitPane(1, true, jpnsroll, jpnsrollTxt);
contentPane.add(jsp, BorderLayout.CENTER);
                                           //添加拆分面板到主窗口的中间部分

//设置主界面的南面部分
//设置信息显示面板
jlbXinxi = new JLabel("文件信息");,
panel = new JPanel();
panel.add(jlbXinxi);
contentPane.add(panel, BorderLayout.SOUTH);
                                           //添加拆分面板到主窗口的南面部分

frame.setSize(500, 300);                   //设置主界面大小
frame.setVisible(true);                     //显示主界面
//frame 的事件监听处理 */
frame.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);                     //程序退出
    }
});
//对组合框事件监听处理
combo.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String str = combo.getSelectedItem() + "\\.";
        try {
            dir = new File(str);
            fileList = dir.listFiles();
            list.setListData(fileList);
        } catch (Exception ee) {
        }
    }
});
...                                         //省略部分代码
}
public static void main(String[] args) {    //主函数
    new FileViewer();                       //创建 FileViewer 类对象
}
}

```

【代码解析】

上述代码实现了文件内容查看器项目界面，该用户界面涉及的具体容器、对象和布局如图 19.7 所示。

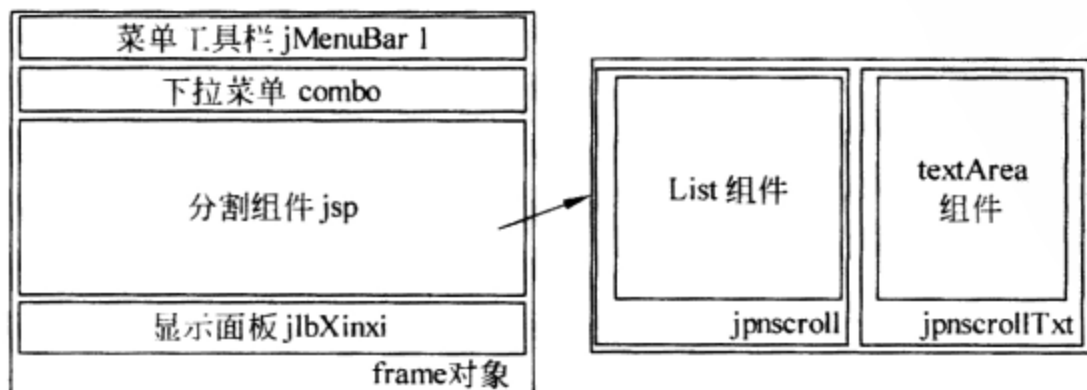


图 19.7 布局

19.2.2 “打开”菜单项的处理方法

ActionListener 事件监听器中的 actionPerformed()方法为处理“打开”菜单项的方法，为了使代码简单和易读，该监听器通过在用户界面中的匿名类方式来实现，具体内容如代码 19.2 所示。

代码 19.2 “打开”菜单项的处理方法：ActionListener

```
jMenuItem19.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        String str;                                //创建字符串变量  
        jFileChooser1.showOpenDialog(frame);        //显示对话框  
        file = jFileChooser1.getSelectedFile();      //获取对话框返回的文件  
        try {  
            //创建带有缓冲功能的输入流对象  
            BufferedReader reader = new BufferedReader(  
                new java.io.FileReader(file));  
            while (true) {                          //通过循环实现内容的显示  
                str = reader.readLine();             //读取一行内容  
                if (str == null)  
                    break;  
                textArea.append(str + "\n");         //显示读取的内容  
            }  
        } catch (Exception ex) {  
            ex.printStackTrace();  
        }  
    }  
});
```

【代码解析】

上述代码中主要实现文件读取功能，即将“打开”文件对话框返回的文件内容显示在对象 textArea 里。在具体实现读取文件功能时，通过如图 19.8 所示的过程获取输入流。

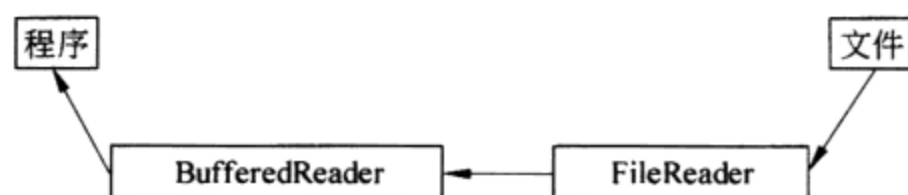


图 19.8 输入流操作

19.2.3 单击列表选项的处理方法

MouseListener 事件监听器中的 mouseClicked()方法，是处理单击列表文本框中的列表选项方法。为了使代码简单易读，该监听器通过在用户界面中的匿名类方式来实现，具体内容如代码 19.3 所示。

代码 19.3 单击列表选项处理方法：MouseListener

```
list.addMouseListener(new MouseAdapter() {  
    public void mouseClicked(MouseEvent me) {
```



```

String fileStr = list.getSelectedValue().toString();
//获取所选择的文件名
File file= new File(fileStr);
//创建文件变量
//文件读取功能
try {
    if (!file.isDirectory()) {
        //当为文件时
        //创建字符方式的读入流对象 fin
        FileReader fin = new FileReader(file);
        //创建带有缓冲功能的读入流对象 bin
        BufferedReader bin = new BufferedReader(fin);
        //创建字符串变量
        String txt = "", strTemp = "";
        while (true) {
            strTemp = bin.readLine();
            //读取文件内容
            if (strTemp == null)
                break;
            txt += strTemp + "\n\r";
        }
        bin.close();
        fin.close();
        textArea.setText(txt);
        //为文本域赋值
    } else {
        //当为文件夹时
        fileList = new File(fileStr).listFiles();
        //获取文件夹里的文件列表
        list.setListData(fileList);
        //设置文本域的内容
    }
} catch (IOException ioex) {
}
});
}

```

【代码解析】

上述代码中主要实现文件读取的功能，即首先返回所单击列表框中列表选项的文件名，然后通过如图 19.9 所示的过程获取该文件的输入流。

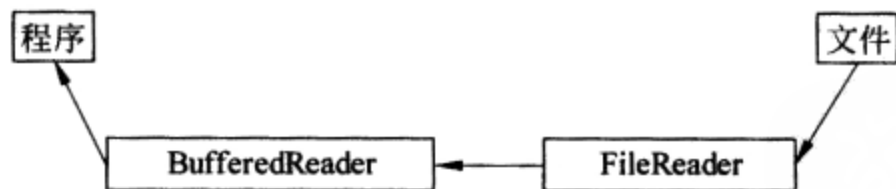


图 19.9 输入流操作

19.3 知识点扩展——管道的访问

为了使 Java 应用程序具有“强内聚，弱耦合”的特性，可以对 Java 应用程序先进行功能模块划分，然后再通过管理流通信对所有功能模块进行整合。所谓管道流通信，就是线程之间的通信，在 I/O 包中存在两种方式的访问来对应，即字节方式的访问 `PipeInputStream` 和 `PipeOutputStream` 类；字符方式的访问 `PipeReader` 和 `PipeWriter`。

19.3.1 管道的访问——字节方式

本节通过实现线程间通信功能的事例来讲解 `PipeInputStream` 和 `PipeOutputStream` 类的基本用法，具体步骤如下。

(1) 创建一个实现向管道写入信息功能的类 `Sender`，具体内容如代码 19.4 所示。

代码 19.4 写入的管道流: `Sender.java`

```
class Sender extends Thread {
    private PipedOutputStream out = new PipedOutputStream();
                                                    //线程的写入流属性 out
    public PipedOutputStream getOutputStream() {    //属性 out 的 get 方法
        return out;
    }
    public void run() {                                //重写 run() 方法
        String s = new String("hello,receiver,how are you!");
                                                    //创建字符串变量 s
        try {
            out.write(s.getBytes());                //写入信息
            out.close();                             //关闭输入流
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

【代码解析】

在上述代码中继承了 `Thread` 类，所以该类的主体功能在 `run()` 方法中实现；由于用来实现向管道中写入信息，所以创建了一个 `PipedOutputStream` 类型的属性 `out`。在 `run()` 方法中，首先创建了一个所要写入内容的变量 `s`，然后通过 `out.write()` 方法把传入的参数写入管道。

(2) 创建一个实现读取管道里信息功能的类 `Receiver`，具体内容如代码 19.5 所示。

代码 19.5 读取的管道流: `Receiver.java`

```
class Receiver extends Thread {
    private PipedInputStream in = new PipedInputStream();
                                                    //线程的读取流属性 in
    public PipedInputStream getInputStream() {    //属性 in 的 get 方法
        return in;
    }
    public void run() {                                //重写 run() 方法
        String s = null;                                //创建一个变量
        byte[] buf = new byte[1024];                  //创建一个字节数组
        try {
            int len = in.read(buf);                    //读取相应字节数据到数组 buf 中
            s = new String(buf, 0, len);                //把字节数组转换成字符串
            //输出相应的信息
            System.out
                .println("the following message comes from sender:\n"+s);
            in.close();                                //关闭读取流
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

```

    }
}

```

【代码解析】

上述代码与 Sender 类的内容基本相同，由于实现读取功能，所以用 PipedInputStream 类对象 in 代替了 PipeOutputStream 类型，而在 run() 方法中，通过 in.read() 方法来读取管道里的信息。

(3) 创建用来实现管道通信功能的 PipeAccess 类，具体内容如代码 19.6 所示。

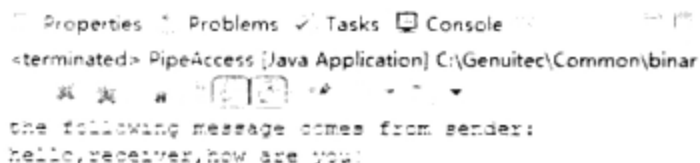
代码 19.6 管道通信: PipeAccess.java

```

public class PipeAccess {
    public static void main(String args[]) {
        try {
            Sender t1 = new Sender();           //创建 Sender 对象 t1
            Receiver t2 = new Receiver();       //创建 Receiver 对象 t2
            //获取管道输入流对象 out
            PipedOutputStream out = t1.getOutputStream();
            //获取管道读取流对象 in
            PipedInputStream in = t2.getInputStream();
            out.connect(in);                     //实现管道输入流与输出流的相连
            //启动相应线程
            t1.start();
            t2.start();
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}

```

运行 PipeAccess.java 类，其控制台窗口如图 19.10 所示。



```

<terminated> PipeAccess [Java Application] C:\Genuitec\Common\binar
the following message comes from sender:
hello,receiver,how are you!

```

图 19.10 运行结果

【代码解析】

- 只有将 PipedInputStream 对象和 PipeOutputStream 对象通过 connect() 方法连接起来才能产生一个通信管道，PipeOutputStream 类的对象用来从管道中读取类 PipedInputStream 对象写入该管道的内容。
- 在上述代码中，首先通过 Sender 和 Receiver 类获取 PipeOutputStream 和 PipedInputStream 对象，然后通过线程的 start() 方法运行程序。

19.3.2 管道的访问——字符方式

查看 API 帮助文档可以发现，与 PipedInputStream 和 PipedOutputStream 相对应的操作管

道并以字符数组为操作单位的类为 `PipeReader` 和 `PipeWriter`。

下面通过实现与 19.3.1 节相同功能的事例，来讲解类 `PipeReader` 和 `PipeWriter` 的基本使用，具体步骤如下。

(1) 创建一个实现向管道写入信息功能的类 `Sender`，具体内容如代码 19.7 所示。

代码 19.7 写入的管道流: `Sender.java`

```
class Sender extends Thread {
    private PipedWriter out = new PipedWriter(); //线程的写入流属性 out
    public PipedWriter getOutputStream() { //属性 out 的 get 方法
        return out;
    }
    public void run() { //重写 run() 方法
        //创建字符串变量 s
        String s = new String("hello,receiver,how are you!");
        try {
            out.write(s.toCharArray()); //写入信息
            out.close(); //关闭输入流
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

【代码解析】

上述代码与原来 `Sender` 类的内容基本相同，由于操作的单位为 `char` 而不是 `byte`，所以用 `PipedWriter` 类对象 `out` 代替了 `PipeOutputStream` 类型。而在 `run()` 方法中，首先通过 `toCharArray()` 方法将字符串转换成字符数组，然后才通过 `write()` 方法将字符数组中的内容写入管道。

(2) 创建一个实现读取管道里信息功能的类 `Receiver`，具体内容如代码 19.8 所示。

代码 19.8 读取的管道流: `Receiver.java`

```
class Receiver extends Thread {
    private PipedReader in = new PipedReader(); //线程的读取流属性 in
    public PipedReader getInputStream() { //属性 in 的 get 方法
        return in;
    }
    public void run() { //重写 run() 方法
        String s = null; //创建一个变量
        char[] buf = new char[1024]; //创建一个字符数组
        try {
            int len = in.read(buf); //读取相应字符数据到数组 buf 中
            s = new String(buf, 0, len); //把字符数组转换成字符串
            System.out
                .println("the following message comes from sender:\n"+s);
            in.close();
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

【代码解析】

上述代码与 Receiver 类的内容基本相同, 由于操作的单位为 char 而不是 byte, 所以用 Pipedreader 类对象 in 代替了 PipeInputStream 类型。而在 run() 方法中, 首先创建了用来存储读取信息的容器——字符数组而不是字节数组, 然后才通过 read() 方法将管道中的内容存储到字符数组中, 最后再把字符数组转换成字符串输出到控制台窗口。

由于类 PipeAccess 的内容完全相同, 所以本节不再详细讲解。

19.4 知识点扩展——内存的访问

在 Java 语言中对数组的操作是非常简单的, 那么在 I/O 包中为什么还要出现内存访问的类? 所谓内存访问, 就是通过相应的类来间接访问内存中的各种类型数组, 即 ByteArrayInputStream 和 ByteArrayOutputStream 这两个类用来操作 byte 的数组; CharArrayReader 和 CharArrayWriter 这两个类用来操作 char 的数组; StringReader 和 StringWriter 这两个类用来操作字符串。

19.4.1 内存的访问——字节方式

在操作系统中存在内存虚拟文件或内存映像文件的概念, 它们主要是把一块内存虚拟成磁盘中的文件, 如果把程序运行过程中要产生的一些临时文件内容存储到内存虚拟文件里, 那么在访问时就不需要访问磁盘了, 直接访问内存从而提高应用程序的效率。下面将通过一个具体的实例来讲解以字节方式来访问内存。

创建实现“内存的访问——字节”访问功能的 ByteArrayAccess 类, 该类主要演示 ByteArrayInputStream 和 ByteArrayOutputStream 类的基本用法, 具体内容如代码 19.9 所示。

代码 19.9 内存的字节访问方式: ByteArrayAccess.java

```
public class ByteArrayAccess {
    public static void main(String[] args) throws Exception {
        String tmp = "abcdefghijklmnopqrstuvwxyz"; //创建字符串变量
        byte[] src = tmp.getBytes();              //创建转换前的内存块变量
        //创建与 src 相关联的输入流对象 input
        ByteArrayInputStream input = new ByteArrayInputStream(src);
        //创建输出流对象 output
        ByteArrayOutputStream output = new ByteArrayOutputStream();
        //调用 transform() 方法
        new ByteArrayAccess.transform(input, output);
        byte[] result = output.toByteArray();     //获取写出流中的字节数据
        System.out.println(new String(result));   //输出转换后的字节数据
    }
    //转换方法
    public void transform(InputStream in, OutputStream out) {
        int c = 0;                                //创建一个变量
        try {                                       //实现转换
            while ((c = in.read()) != -1)         //判断是否读取完成
            {
                out.write(c);
            }
        }
    }
}
```

```

        //对每个字节数据进行转换
        int C = (int) Character.toUpperCase((char) c);
        out.write(C);           //输出字符
    }
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

运行 `ByteArrayAccess.java` 类，其控制台窗口如图 19.11 所示。

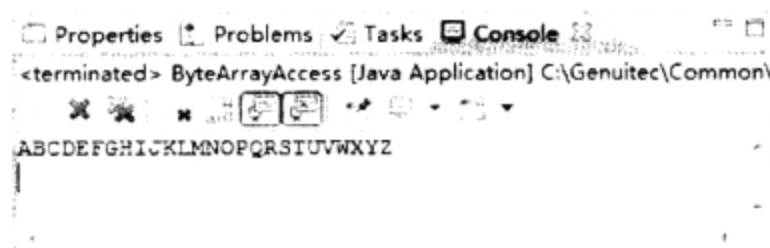


图 19.11 运行结果

【代码解析】

- 在上述代码中存在一个实现转换功能的方法 `transform(InputStream in, OutputStream out)`，在该方法中首先遍历读取流中的每一个字节，然后不仅通过 `Character.toUpperCase()` 方法把字节的字符转换成大写，最后把转换后的字符写入到输出流中。
- 在上述代码的主方法 `main()` 中，首先创建了一个“内存虚拟文件——字符串”变量，然后不仅创建该字符串字节数组的读取流和输出流，而且还把它们作为参数传递给 `Character.toUpperCase()` 方法，最后再把输出流中的字节在控制窗口输出。

19.4.2 内存的访问——字符和字符串方式

查看 API 帮助文档可以发现，与 `ByteArrayInputStream` 和 `ByteArrayOutputStream` 类对应的操作内存方法，为以字符数组为单位的 `CharArrayReader` 和 `CharArrayWriter` 类，还有以字符串为单位的 `StringReader` 和 `StringWriter` 类。下面将通过一个具体的实例来讲解以字符和字符串方式访问内存的过程，具体步骤如下。

(1) 创建实现“内存的访问——字符”访问功能的 `CharArrayAccess` 类，该类主要演示 `CharArrayReader` 和 `CharArrayWriter` 类的基本使用，具体内容如代码 19.10 所示。

代码 19.10 内存的字符访问方式: `CharArrayAccess.java`

```

public class CharArrayAccess {
    public static void main(String[] args) throws Exception {
        String tmp = "abcdefghijklmnopqrstuvwxyz";
        char[] src = tmp.toCharArray();           //把字符串转换成字符数组
        //创建字符数组的读取流对象 input
        CharArrayReader input = new CharArrayReader(src);
        //创建输出流对象 output
        CharArrayWriter output = new CharArrayWriter();
        new CharArrayAccess().transform(input, output);
        String result = output.toString();         //获取输出流中的字符串
    }
}

```



```

        System.out.println(new String(result));
    }
    public void transform(Reader in, Writer out) {    //实现转换功能的方法
        int c = 0;
        try {
            while ((c = in.read()) != -1)
            {
                int C = (int) Character.toUpperCase((char) c);
                out.write(C);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

运行 CharArrayAccess.java 类，其控制台窗口如图 19.12 所示。



图 19.12 运行结果

【代码解析】

- 由于上述代码实现的功能与 ByteArrayAccess 类实现的功能相同，所以只需把用到 ByteArrayInputStream 和 ByteArrayOutputStream 类的地方用 CharArrayReader 和 CharArrayWriter 类代替就可以。
- 由于 CharArrayReader 和 CharArrayWriter 类操作的对象为字符对象，所以需要对“内存访问——字符”变量 tmp 先转换成字符数组 src。

(2) 创建实现“内存的访问——字符串”访问功能的 StingAccess 类，该类主要演示 StringReader 和 StringWriter 类的基本用法，具体内容如代码 19.11 所示。

代码 19.11 内存的字符串访问方式: StingAccess.java

```

public class StingAccess {
    public static void main(String[] args) throws Exception {
        String tmp = "abcdefghijklmnopqrstuvwxyz";
        //创建字符串的读取流对象 input
        StringReader input = new StringReader(tmp);
        //创建输出流对象 output
        StringWriter output = new StringWriter();
        new StingAccess().transform(input, output);
        String result = output.toString();
        System.out.println(new String(result));
    }
    public void transform(Reader in, Writer out) {    //实现转换功能方法
        int c = 0;
        try {
            while ((c = in.read()) != -1)
            {
                int C = (int) Character.toUpperCase((char) c);
                out.write(C);
            }
        }
    }
}

```

```
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

运行 `StingAccess.java` 类，其控制台窗口如图 19.13 所示。

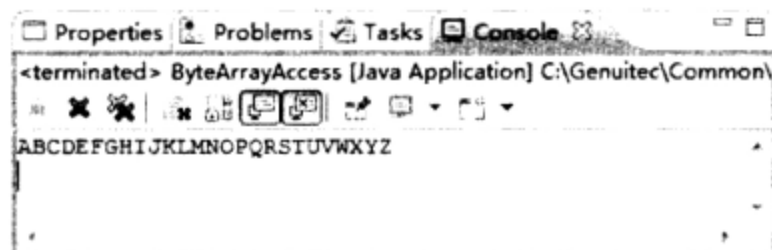


图 19.13 运行结果

【代码解析】

- 由于上述代码实现的功能与 `ByteArrayAccess` 类实现的功能相同，所以只需把用到 `ByteArrayInputStream` 和 `ByteArrayOutputStream` 类的地方用 `StringReader` 和 `StringWriter` 类代替即可。
- 由于 `StringReader` 和 `StringWriter` 类操作的对象为字符串对象，所以不需要对“内存虚拟文件——字符串”变量 `tmp` 进行字符数组的转换或字节数组的转换。

19.5 小 结

本章主要通过对文件的访问来获取文件的内容，并把相应的文件内容显示在相应的 `Swing` 中的文本框中。对于该项目，如果想选择文件，可以通过“文件”菜单或下拉列表框中的选项来实现。

在本章的最后还详细介绍了内存和管道访问的基础知识，对于管道访问，可以以字节和字符为单位进行访问，而对于内存访问，则可以以字节、字符和字符串为单位进行访问。

第 20 章 日记簿（GUI+文件访问和操作）

本章将通过 Swing 组件实现日记簿界面,通过文件访问和操作实现该项目的后台功能。日记簿的实现综合了图形用户界面的所有知识点和 Java 语言中的 I/O 操作。

本章的学习目标如下:

- ❑ 掌握组件和面板的使用方法;
- ❑ 掌握日记簿项目;
- ❑ 熟练掌握文件的操作和访问。

20.1 日记簿原理

“日记簿”项目不仅实现日记文件的创建,而且还实现对日记的查看和删除功能。为了便于用户的操作,本章还专门通过 Swing 组件设计了图形用户界面。

20.1.1 项目结构框架分析

对于日记簿项目,首先需要创建两个界面,即日记输入界面和日记列表对话框。日记簿项目目录如图 20.1 所示,各个目录的功能如下。

- ❑ DiaryInputInterface 类:日记簿输入界面的类。
- ❑ DiaryList 类:日记列表对话框的类。

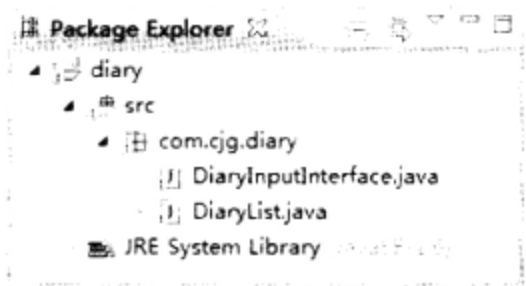


图 20.1 项目目录

20.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括日记簿的初始化界面、输入日记簿记录输入、清空日记簿记录、查看日记簿的所有记录和操作日记簿记录。

1. 初始化界面

当运行日记簿项目中的 DiaryInputInterface 类后,会出现如图 20.2 所示的初始界

面——日记簿输入界面。

2. 输入日记簿记录输入

当出现初始界面后，如果想实现日记簿记录的保存功能，可以先输入如图 20.3 所示的信息，然后单击“保存”按钮。

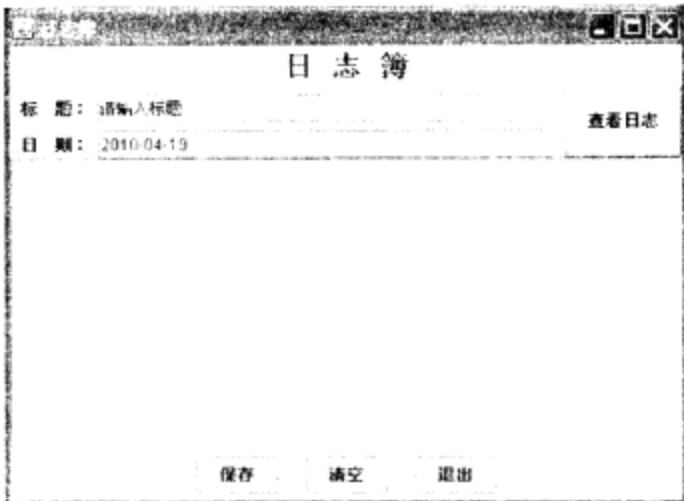


图 20.2 初始界面



图 20.3 保存日记簿记录

3. 清空日记簿记录

当保存完日记簿记录后，如果还想继续保存其他记录信息，可以先单击“清空”按钮，这时就会出现初始界面，具体过程如图 20.4 所示。



图 20.4 清空过程

4. 查看日记簿的所有记录

如果想查看日记簿的所有记录信息，可以在日记簿输入界面中单击“查看日志”按钮，这时就会出现“日志列表”对话框，具体过程如图 20.5 所示。在该对话框中将显示出保存的所有日记记录。

5. 日记簿记录的操作

在“日志列表”对话框中可以实现对日记簿记录的所有操作，例如，如果想实现删除功能，可以单击“删除”按钮，具体过程如图 20.6 所示；如果想实现查看该日记记录的功能，可以单击“查看”按钮，具体过程如图 20.7 所示；如果想返回到日记簿输入界面，可以单击“返回”按钮，具体过程如图 20.8 所示。

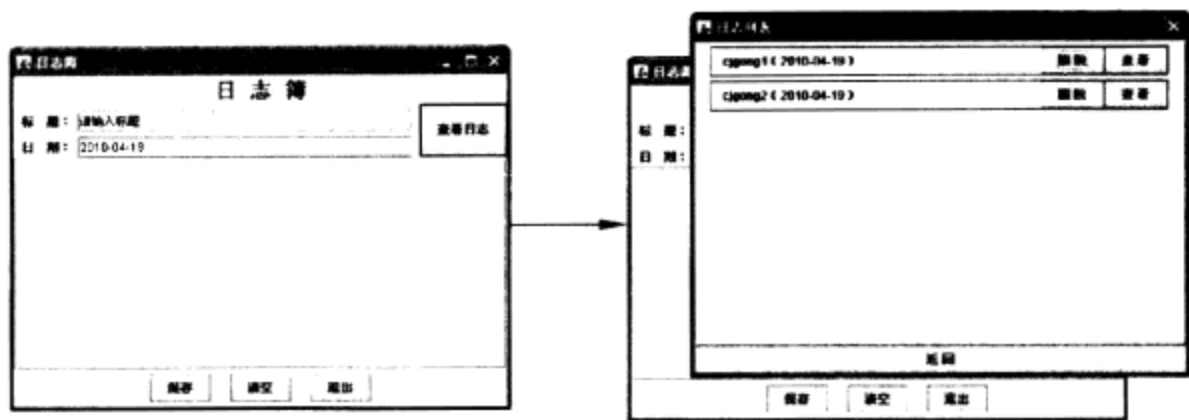


图 20.5 查看日记过程

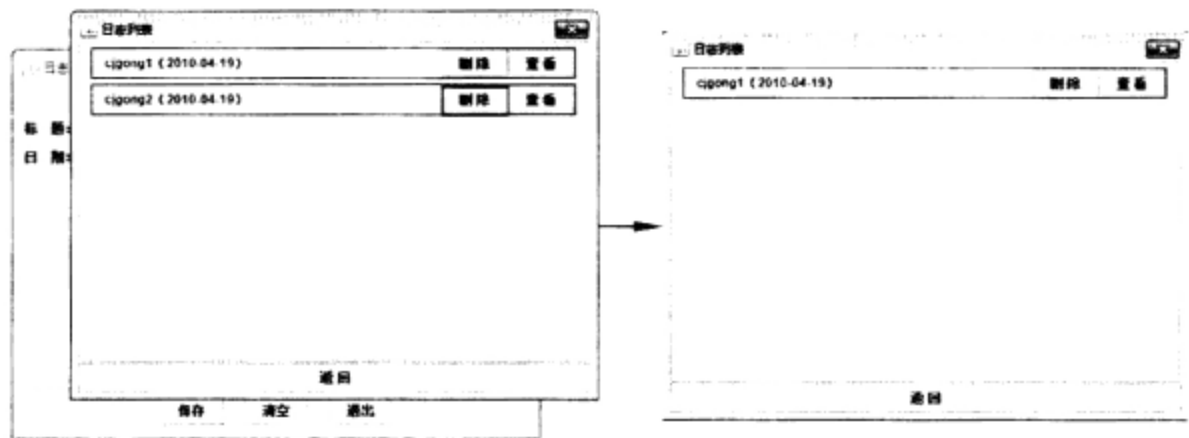


图 20.6 删除日记记录过程

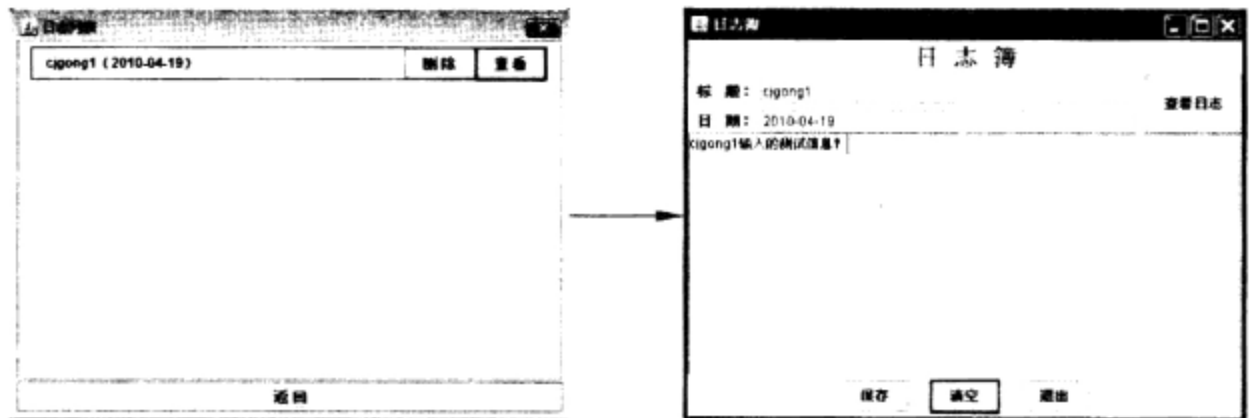


图 20.7 查看日记记录过程

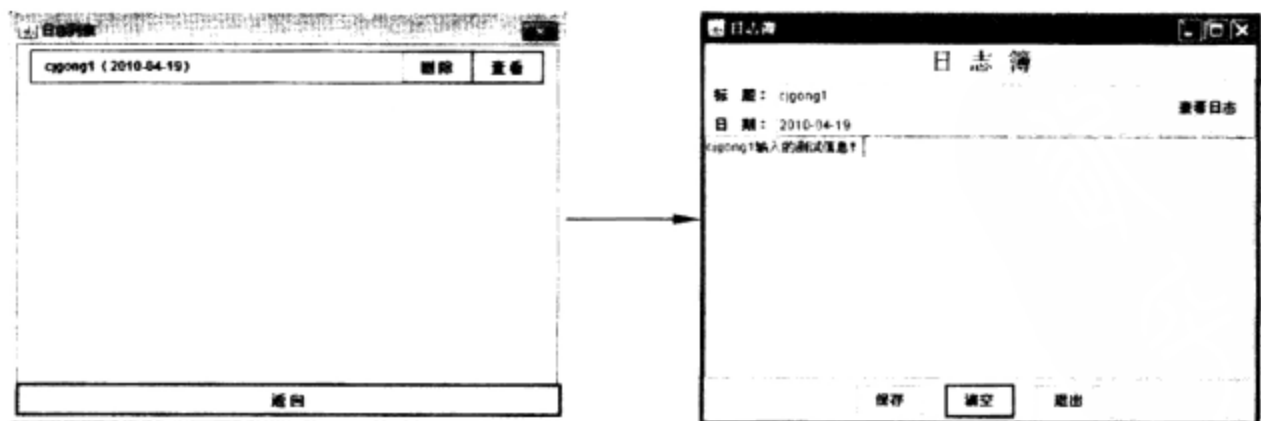


图 20.8 返回过程

20.2 日记簿项目

日记簿项目具体程序架构如图 20.9 所示，它包含一个“日记簿输入界面”的类和“日

记列表对话框”的类 `DiaryInputInterface.java` 和 `DiaryList.java`。在日记簿输入界面中将日记以文本文件的形式保存到本地计算机中，在日记列表对话框中获取计算机中的日记文件信息。

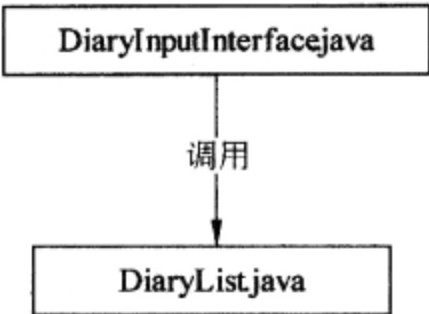


图 20.9 程序关系图

20.2.1 设计项目的界面——日记簿输入界面

`DiaryInputInterface` 为日记簿输入界面的类，在该类中不仅实现将日记内容保存为文件，而且还会实现打开“日记列表”对话框，具体内容如代码 20.1 所示，该类的类图如图 20.10 所示。

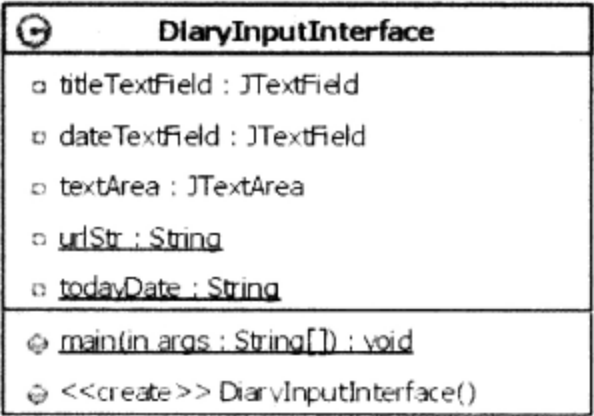


图 20.10 输入界面的类图

代码 20.1 日记内容输入界面: `DiaryInputInterface.java`

```
public class DiaryInputInterface extends JFrame {
    private JTextField titleTextField;           //标题文本框
    private JTextField dateTextField;           //日期文本框
    private JTextArea textArea;                 //内容文本域
    private final static String urlStr = "C:/cjgong /"; //文本文件存放路径
    //将当前日期格式化为“xxxx-xx-xx”格式
    private final static String todayDate = String.format("%tF", new
Date());
    static {                                     //在静态代码块中初始化文本文件的存放路径
        File file = new File(urlStr);
        if (!file.exists())
            file.mkdirs();
    }
    public static void main(String args[]) {    //主方法
        try {
            //创建 DiaryInputInterface 对象
            DiaryInputInterface frame = new DiaryInputInterface();
            frame.setVisible(true);            //显示界面
        } catch (Exception e) {
```



```

        e.printStackTrace();
    }
}

public DiaryInputInterface() { //构造函数
    super();
    //设置主窗口
    //设置对象 DiaryInputInterface
    setTitle("日记簿"); //设置标题
    setBounds(100, 100, 500, 375); //设置大小
    //设置关闭功能
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    //主窗口 NORTH 面部分
    //创建和设置显示框对象 softLabel
    final JLabel softLabel = new JLabel(); //创建显示框对象
    //设置容器的前景色
    softLabel.setForeground(new Color(255, 0, 0));
    //设置字体
    softLabel.setFont(new Font("", Font.BOLD, 22));
    //设置水平对齐方式
    softLabel.setHorizontalAlignment(SwingConstants.CENTER);
    softLabel.setText("日 志 簿"); //设置内容
    //添加到主窗口中
    getContentPane().add(softLabel, BorderLayout.NORTH);
    //主窗口 CENTER 面部分
    //创建和设置内容容器对象
    final JPanel contentPanel = new JPanel(); //创建内容容器对象
    //设置布局管理器
    contentPanel.setLayout(new BorderLayout());
    //添加到主窗口中
    getContentPane().add(contentPanel, BorderLayout.CENTER);
    //内容容器 contentPanel 的 CENTER 部分
    //创建和设置 infoPanel 容器
    final JPanel infoPanel = new JPanel(); //创建容器 infoPanel
    //添加到内容容器对象中
    contentPanel.add(infoPanel, BorderLayout.CENTER);
    //日记输入标题时涉及的组件
    //创建和设置日记标题的显示框
    final JLabel titleLabel = new JLabel(); //创建显示框
    titleLabel.setText("标 题: "); //显示的文本
    infoPanel.add(titleLabel); //添加到容器 infoPanel 中
    //创建和设置日记标题的输入框
    titleTextField = new JTextField(); //为标题文本框赋值
    titleTextField.setColumns(30); //设置标题文本框的长度
    titleTextField.setText("请输入标题"); //设置标题文本框的默认标题
    //添加焦点事件
    titleTextField.addFocusListener(new FocusListener() {
        public void focusGained(FocusEvent e) { //焦点获取时
            titleTextField.setText(""); //清空
        }
        public void focusLost(FocusEvent e) { //焦点失去时
            //获取文本框中的内容
            String date = titleTextField.getText().trim();
            if (date.length() == 0) //判断内容的长度
                titleTextField.setText("请输入标题");
        }
    });
}

```

```

    });
    infoPanel.add(titleTextField); //添加到容器 infoPanel 中
    //日记输入日期时涉及的组件
    //创建和设置日记日期的显示框
    final JLabel dateLabel = new JLabel(); //创建显示框
    dateLabel.setText("日期:"); //显示的文本
    infoPanel.add(dateLabel); //添加到容器 infoPanel 中
    //创建和设置日记时间的输入框
    dateTextField = new JTextField(); //为日期文本框赋值
    dateTextField.setColumns(30); //设置日期文本框的长度
    dateTextField.setText(todayDate); //设置日期文本框的默认时间
    //添加焦点事件
    dateTextField.addFocusListener(new FocusListener() {
        public void focusGained(FocusEvent e) { //焦点获取时
            dateTextField.setText("");
        }
        public void focusLost(FocusEvent e) { //焦点失去时
            String date = dateTextField.getText().trim();
            if (date.length() != 10)
                dateTextField.setText(todayDate);
        }
    });
    infoPanel.add(dateTextField); //添加到容器 infoPanel 中
    ... //省略部分代码

    //主窗口 SOUTH 面部分
    //创建和设置操作日记输入的按钮
    //创建和设置一个采用 FlowLayout 布局管理器的容器
    final JPanel buttonPanel = new JPanel(); //创建一个容器（存放按钮）
    //创建布局管理器对象
    final FlowLayout flowLayout = new FlowLayout();
    flowLayout.setHgap(20);
    buttonPanel.setLayout(flowLayout); //设置布局管理器
    //添加到容器 contentPanel 的南面
    getContentPane().add(buttonPanel, BorderLayout.SOUTH);
    //创建和设置“保存”按钮
    final JButton saveButton = new JButton(); //创建“保存”按钮对象
    saveButton.setText("保存"); //设置按钮显示文本
    //设置按钮的监听事件
    saveButton.addActionListener(new SaveButtonActionListener());
    buttonPanel.add(saveButton); //添加到容器 buttonPanel 中
    //创建和设置“清空”按钮
    final JButton clearButton = new JButton(); //创建“清空”按钮对象
    clearButton.setText("清空"); //设置按钮显示文本
    //设置按钮的监听事件
    clearButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            titleTextField.setText("请输入标题");
            dateTextField.setText(todayDate);
            textArea.setText("");
        }
    });
    ... //省略部分代码
}

//“查看日记”按钮的事件
private class SeeButtonActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {

```

```
... //省略部分代码
    }
}
// “保存” 按钮的事件
private class SaveButtonActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
... //省略部分代码
    }
}
}
```

【代码解析】

上述代码实现了日记簿输入界面，该用户界面涉及的具体容器、对象和布局如图 20.11 所示。

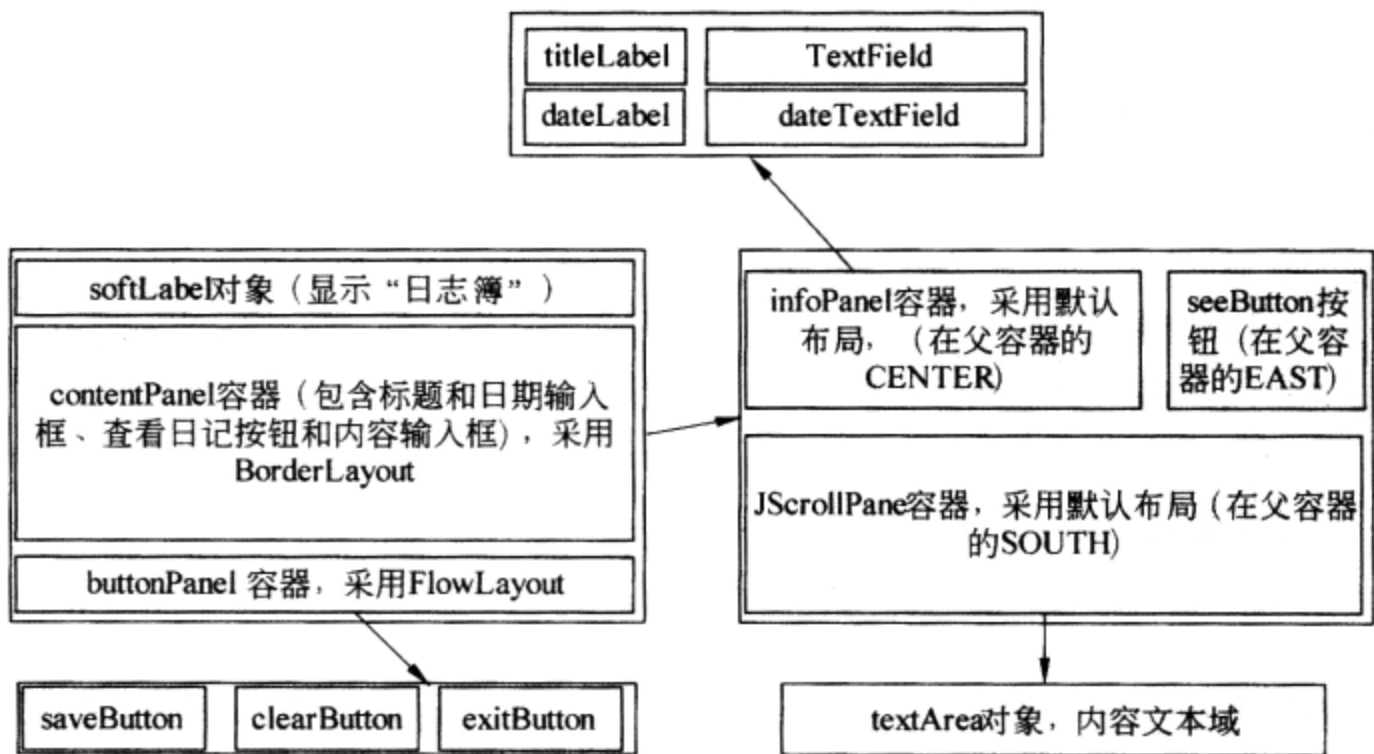


图 20.11 布局

20.2.2 “保存”按钮的事件处理

SaveButtonActionListener.java 类为 DiaryInputInterface 用户界面的内部类，主要用来实现将日记内容保存为一个名称格式为“标题（日期）.text”的文本文件，具体内容如代码 20.2 所示，该类的类图如图 20.12 所示。

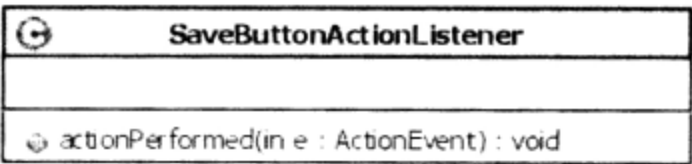


图 20.12 SaveButtonActionListener 的类图

代码 20.2 保存按钮的事件处理：SaveButtonActionListener.java

```
private class SaveButtonActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        String title = titleTextField.getText(); //获得日记标题
        String date = dateTextField.getText(); //获得日记日期
    }
}
```

```

String name = title + "(" + date + ").txt"; //组织文本文件名称
File file = new File(urlStr + name); //创建文本文件对象
if (!file.exists()) { //判断文件是否存在
    try {
        file.createNewFile(); //如果不存在则创建文件
    } catch (IOException e1) {
        e1.printStackTrace();
    }
}
try {
    FileWriter fileWriter = new FileWriter(file); //创建字符输出流
    fileWriter.write(textArea.getText()); //将内容写入文本文件中
    fileWriter.close(); //关闭字符输出流
} catch (IOException e1) {
    e1.printStackTrace();
}
}
}

```

【代码解析】

上述代码首先创建文件，然后向文件写入内容。在具体创建时，用日记的标题和日期的组合作为文件的名称；在具体向文件写入内容时，通过 `FileWriter` 类型对象 `fileWriter` 的方法 `write()` 写入“内容文本域”中的内容。

20.2.3 “查看日记”按钮的事件处理

`SeeButtonActionListener.java` 类为 `DiaryInputInterface` 用户界面的内部类，主要用来打开“日志列表”对话框，具体内容如代码 20.3 所示，该类的类图如图 20.13 所示。

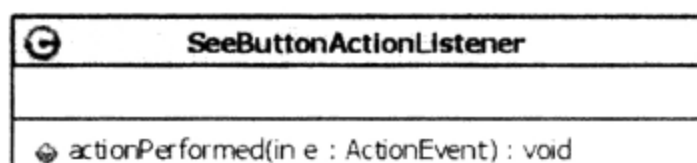


图 20.13 `SeeButtonActionListener` 的类图

代码 20.3 查看日记按钮的事件处理：SeeButtonActionListener.java

```

private class SeeButtonActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        DiaryList listFrame = new DiaryList(); //创建日记列表对象
        listFrame.setVisible(true); //显示日记列表窗体
        File text = listFrame.getText(); //日记对象
        listFrame.dispose(); //销毁日记列表窗体
        if (text != null) { //查看日记对象是否为空
            //分割日记文件的名称
            String[] infos = text.getName().split("(");
            titleTextField.setText(infos[0]); //设置日记标题
            dateTextField.setText(infos[1]); //设置日记日期
            try {
                //创建字符输入流
                FileReader fileReader = new FileReader(text);
                char[] cbuf = new char[(int) text.length()];
            }
        }
    }
}

```

```

//创建字符型数组
fileReader.read(cbuf); //读入文件内容到字符型数组中
fileReader.close(); //关闭字符输入流
textArea.setText(String.valueOf(cbuf)); //设置日记内容
} catch (FileNotFoundException e1) {
    e1.printStackTrace();
} catch (IOException e2) {
    e2.printStackTrace();
}
}
}
}

```

【代码解析】

在上述代码中, 首先创建 `DiaryList` 类用来实现打开日记列表对话框, 然后通过该类的 `getText()` 方法返回 `File` 对象, 最后通过文件访问操作创建日记列表的内容。

20.2.4 设计项目的界面——日记列表界面

`DiaryList` 为日记列表界面的类, 在该类中不仅实现日记列表界面, 还会实现日记记录的操作, 具体内容如代码 20.4 所示, 该类的类图如图 20.14 所示。

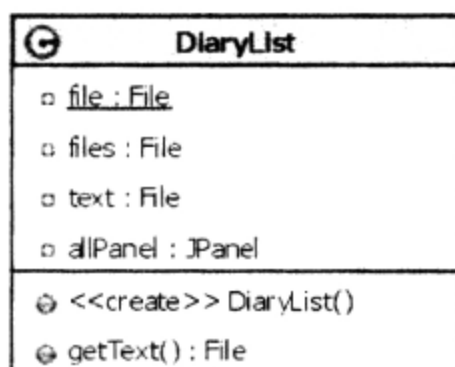


图 20.14 日记列表的类图

代码 20.4 日记列表界面: `DiaryList.java`

```

public class DiaryList extends JDialog {
    //创建成员变量
    private static File file = null; //文本文件存放文件夹对象
    private File[] files = null; //文本文件对象数组
    private File text = null; //查看的文本文件对象
    private JPanel allPanel; //显示文件信息的面板
    static { //在静态代码块中实现初始化文本文件的功能
        file = new File("C:/cjgong");
    }
    public DiaryList() { //构造函数
        super();
        setModal(true); //设置对话框模式
        setTitle("日记列表"); //设置对话框名称
        setBounds(100, 100, 500, 375); //设置对话框大小
        //创建和设置主对话框中的北面部分
        //创建和设置滚动列表
        final JScrollPane scrollPane = new JScrollPane();
        getContentPane().add(scrollPane);
    }
}

```

```

//为文件数组赋值
files = file.listFiles();
//为面板对象赋值
allPanel = new JPanel();
//设置文件信息列表大小
allPanel.setPreferredSize(new Dimension(450, files.length * 36));
scrollPane.setViewportView(allPanel); //添加到容器 scrollPane 中
//通过循环创建容器 scrollPane 中 onePanel 组建的个数和内容
for (int i = 0; i < files.length; i++) {
    //获取文件的名字
    String name = "    " + files[i].getName();
    //对文件名字进行处理
    name = name.substring(0, name.length() - 4);
    //创建一个容器对象 onePanel
    final JPanel onePanel = new JPanel();
    allPanel.add(onePanel); //添加对象到 onePanel 容器中
    //设置容器的边框
    onePanel.setBorder(new LineBorder(Color.black, 1, false));
    //设置容器的布局
    onePanel.setLayout(new BorderLayout());
    final JLabel label = new JLabel(); //创建面板 label 容器
    //设置面板的大小
    label.setPreferredSize(new Dimension(330, 0));
    label.setText(name); //显示变量 name
    onePanel.add(label, BorderLayout.WEST);
    //“删除”按钮
    final JButton delButton = new JButton(); //创建按钮对象
    delButton.setText("删除"); //设置按钮的文本
    delButton.setName("" + i); //设置按钮的名字
    //设置按钮的事件监听器
    delButton.addActionListener(new DelButtonActionListener());
    //添加按钮到容器 onePanel 中
    onePanel.add(delButton, BorderLayout.CENTER);
    //“查看”按钮
    final JButton seeButton = new JButton(); //创建按钮对象
    seeButton.setText("查看"); //设置按钮的文本
    seeButton.setName("" + i); //设置按钮的名字
    //设置按钮的事件监听器
    seeButton.addActionListener(new SeeButtonActionListener());
    //添加按钮到容器 onePanel 中
    onePanel.add(seeButton, BorderLayout.EAST);
}
//创建和设置主对话框中的南面部分
//设置“返回”按钮
final JButton returnButton = new JButton(); //创建按钮对象
returnButton.setText("返回"); //设置按钮文本
//设置事件监听器
returnButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        setVisible(false);
    }
});
//添加到对话框的南面
getContentPane().add(returnButton, BorderLayout.SOUTH);
}
private class SeeButtonActionListener implements ActionListener {

```



```
public void actionPerformed(ActionEvent e) {           //事件处理方法
...                                                    //省略部分代码
}
}

private class DelButtonActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {       //事件处理方法
...                                                    //省略部分代码
    }
}

public File getText() {                                //获取文件对象
    return text;
}
}
```

【代码解析】

上述代码实现了日记列表界面，该用户界面涉及的具体容器、对象和布局如图 20.15 所示。

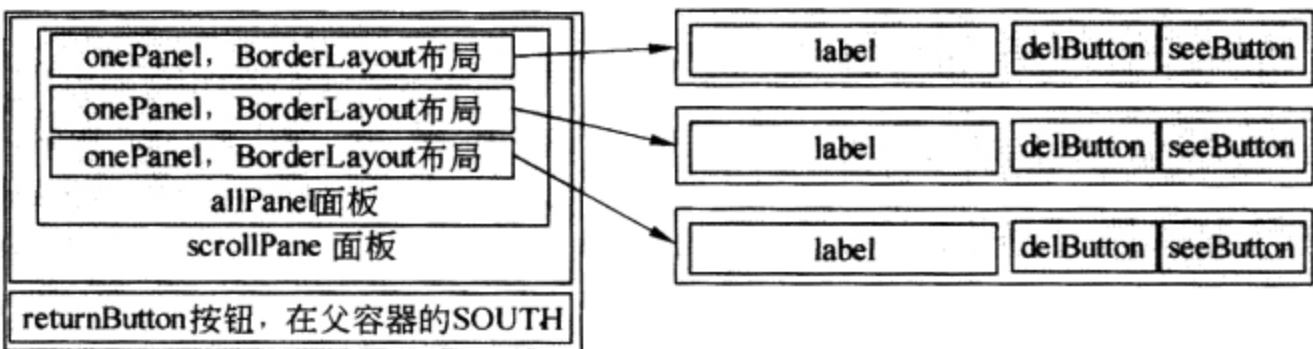


图 20.15 布局

20.2.5 “查看”按钮的事件处理

SeeButtonActionListener.java 类为 DiaryList 用户界面的内部类，主要用来实现查看日记记录的功能，具体内容如代码 20.5 所示，该类的类图如图 20.16 所示。

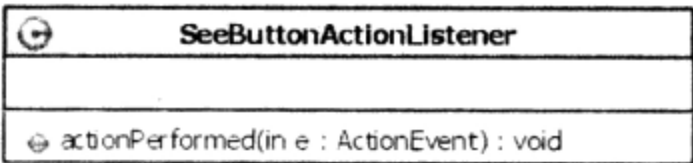


图 20.16 SeeButtonActionListener 类图

代码 20.5 查看按钮的事件处理：SeeButtonActionListener.java

```
private class SeeButtonActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        JButton button = (JButton) e.getSource(); //获取发生事件源的组件
        String name = button.getName();           //获取按钮的名字
        text = files[Integer.valueOf(name)];       //获取日记记录内容
        setVisible(false);                        //隐藏对话框
    }
}
```

【代码解析】

在上述代码中，首先获取查看日记记录所对应按钮的名字，然后根据该按钮的名字通

过文件类的相应方法获取该名字所对应的内容。

20.2.6 “删除”按钮的事件处理

DelButtonActionListener.java 类为 DiaryList 用户界面的内部类，主要用来实现删除日记记录的功能，具体内容如代码 20.6 所示，该类的类图如图 20.17 所示。

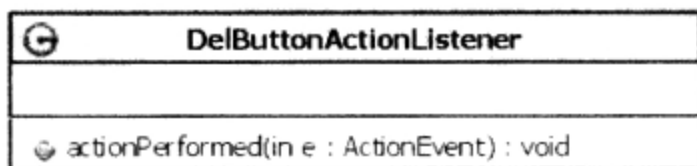


图 20.17 DelButtonActionListener 类图

代码 20.6 删除按钮的事件处理：DelButtonActionListener.java

```
private class DelButtonActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        JButton button = (JButton) e.getSource(); //获取发生事件的组件
        int index = Integer.valueOf(button.getName()); //获取日记记录的索引号
        files[index].delete(); //删除日记文件内容
        allPanel.remove(index); //从界面中删除日记
        SwingUtilities.updateComponentTreeUI(allPanel); //刷新窗体
    }
}
```

【代码解析】

在上述代码中，首先获取日记文件的索引号，然后通过 delete() 方法对该日记文件进行删除，最后再从界面中删除该日记记录，并通过工具类 SwingUtilities 的相应方法刷新窗体。

20.3 知识点扩展——过滤流的基础知识

Java 应用程序中如果只以字节或字符为单位，对数据做输入或输出还是远远不够的，例如有时需要一行一行地读取数据，有时需要读取特定格式的对象等。为了实现上述功能，Java 语言提供了一种过滤机制，即让原本没有特殊访问方法的数据流，经过相应的过滤流后，变得可以用特定方法来访问数据。在 Java 中提供了过滤流的一系列类，本节将一一介绍这些类。

20.3.1 过滤流的缓存（Buffering）类

在 Java 语言中为了提高应用程序的性能，增加了缓存机制。对于 I/O 增加缓存区，使得在流上执行 skip()、mark() 和 reset() 方法成为可能。在 I/O 包中存在两种方式实现缓存功能的过滤器流，即字节方式类 BufferedInputStream 和 BufferedOutputStream；字符方式的类 BufferedReader 和 BufferedWriter。

下面通过实现文件内容复制功能的事例，来讲解 BufferedInputStream、

BufferedOutputStream 和 BufferedReader、BufferedWriter 类的基本用法，具体步骤如下。

(1) 在类 BufferedStreamByte 中，通过以字节为单位并带有缓存功能的方式实现内容复制功能，具体内容如代码 20.7 所示。

代码 20.7 文件内容复制: BufferedStreamByte.java

```
public class BufferedStreamByte {
    public static void main(String[] args) {
        try {
            byte[] data = new byte[1];           //创建字节数组
            //创建相关文件的 File 对象
            File srcFile = new File("C:\\\\BufferedByte.txt");
            File desFile = new File("C:\\\\BufferedByte_1.txt");
            //创建文件输入流的 BufferedInputStream 对象
            BufferedInputStream bufferedInputStream = new BufferedInput-
                Stream(new FileInputStream(srcFile));
            //创建文件输出流的 BufferedOutputStream 对象
            BufferedOutputStream bufferedOutputStream = new BufferedOut-
                putStream(new FileOutputStream(desFile));
            //输出相关信息
            System.out.println("复制文件: " + srcFile.length() + "字节");
            //实现复制功能
            while (bufferedInputStream.read(data) != -1) {
                bufferedOutputStream.write(data);
            }

            bufferedOutputStream.flush();         //将缓存区中的数据全部写出
            System.out.println("复制完成");       //输出相关信息
            //显示输出 BufferedByte_1.txt 文件的内容
            //为缓存输入流对象赋值
            bufferedInputStream = new BufferedInputStream(new FileInput-
                Stream(new File("C:\\\\BufferedByte_1.txt")));
            //获取文件中的内容
            while (bufferedInputStream.read(data) != -1) {
                String str = new String(data); //字节数组转换成字符串
                System.out.print(str);         //输出字符串
            }
            //关闭流对象
            bufferedInputStream.close();
            bufferedOutputStream.close();
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("using: java useFileStream src des");
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

在具体运行 BufferedStreamByte 类之前，首先创建名为 BufferedByte.txt 的文件，其内容如图 20.18 所示。具体运行时，控制台窗口如图 20.19 所示。对于该文件的复制文本 BufferedByte_1.txt，其内容如图 20.20 所示。



图 20.18 BufferedByte 文件的内容

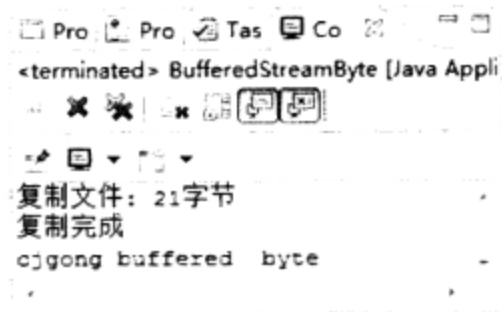


图 20.19 控制台窗口



图 20.20 BufferedByte_1 文件的内容

【代码解析】

- ❑ 在上述代码中主要实现两种功能：即将 BufferedByte 文件的内容复制到 BufferedByte_1 文件里；将 BufferedByte_1 文件里的内容输出到控制台窗口。
- ❑ 上述代码中通过如图 20.21 所示的过程获取输入流和输出流。首先创建了与文件 BufferedByte 和 BufferedByte_1 相关连的文件输入流对象和文件输出流对象，然后通过 BufferedInputStream 和 BufferedInputStreamr 类使得输入流（bufferedInputStream）和输出流（bufferedOutputStream）分别具有了缓存功能。
- ❑ 上述代码的第一种功能，在循环中通过 bufferedInputStream.read()方法读取内容，通过 bufferedOutputStream.write()方法写入内容，从而达到复制的效果。
- ❑ 上述代码的第二种功能，首先获取 BufferedByte_1 文件带有缓存功能的输入流对象 bufferedInputStream，然后通过该对象的 read()方法把文件的内容读取到字节数组 data 中，最后再把字节数组转换成字符串并输出到控制台窗口中。

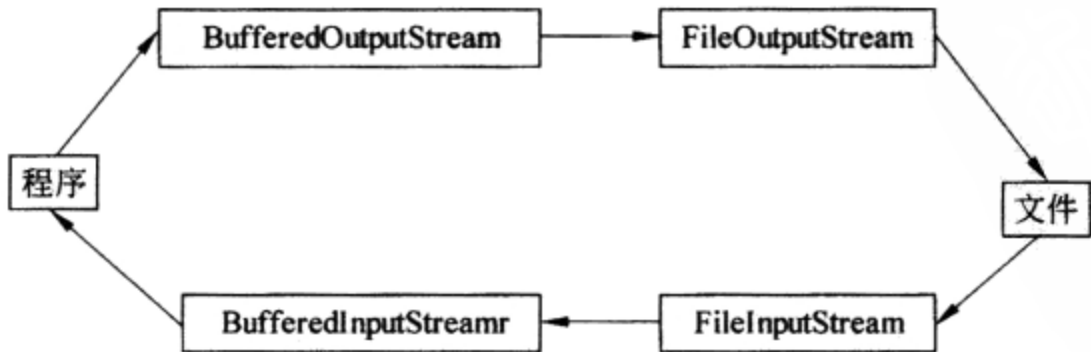


图 20.21 输入流和输出流操作

(2) 在 BufferedStreamChar 类中，通过以字符为单位并带有缓存功能的方式实现内容复制功能，具体内容如代码 20.8 所示。

代码 20.8 文件内容复制: BufferedStreamChar.java

```

public class BufferedStreamChar {
    public static void main(String[] args) {
        try {
            String str; //创建字符串变量
            //创建相关文件的 File 对象
            File srcFile = new File("C:\\\\BufferedByte.txt");
            File desFile = new File("C:\\\\BufferedByte_1.txt");
            //创建文件输入流的 BufferedReader 对象
            BufferedReader in = new BufferedReader(new FileReader(srcFile));
            //创建文件输入流的 BufferedWriter 对象
            BufferedWriter out = new BufferedWriter(new FileWriter(desFile));
            //输出相应信息
            System.out.println("复制文件: " + srcFile.length() + "字节");
            while ((str = in.readLine()) != null) { //实现复制功能
                out.write(str);
                out.newLine(); //实现换行功能
            }
            out.flush(); //将缓存区中的数据全部写出
            System.out.println("复制完成");
            //显示输出 BufferedByte_1.txt 文件的内容
            in = new BufferedReader(new FileReader(new File(
                "C:\\\\BufferedByte_1.txt")));
            while ((str = in.readLine()) != null) {
                System.out.println(str); //输出文件的信息
            }
            //关闭流对象
            in.close();
            out.close();
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("using: java useFileStream src des");
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

【代码解析】

由于上述代码实现的功能与 `BufferedStreamByte` 类实现的功能相同, 所以只需把用到 `BufferedInputStream` 和 `BufferedOutputStream` 类的地方, 用 `BufferedReader` 和 `BufferedWriter` 类来代替就可以。在 `BufferedReader` 类中存在一个名为 `ReaderLine()` 的方法, 实现读取文本行的功能; 在 `BufferedWriter` 类中存在一个名为 `newLine()` 的方法, 实现换行的功能, 文件流的具体过程如图 20.22 所示。

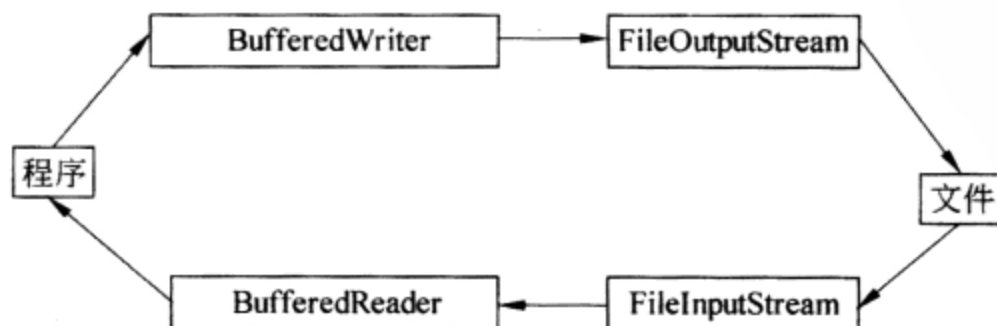



图 20.22 输入流和输出流操作

20.3.2 过滤流实现字节和字符相互转换类

在 Java 语言中支持字节流和字符流,那么如何实现它们之间的转换呢?查看 API 帮助文档可以发现,在 I/O 包中存两个类 `InputStreamReader` 和 `OutputStreamWriter`,前者用来实现将一个字节流中的字节解码成字符,而后者用来将写入的字符编码成字节后写入一个字节流。

 **注意:** 也可以这样理解, `InputStreamReader` 类的作用就是将 `InputStream` 转化成 `Reader`; 而 `OutputStreamWriter` 类的作用就是将 `OutputStream` 转化成 `Writer`。

下面通过实现文件内容复制功能的事例,来讲解 `InputStreamReader` 和 `OutputStreamWriter` 类的基本用法。

在 `Transform` 类中,通过以字符为单位并同时带有缓存功能方式实现内容复制功能,具体内容如代码 20.9 所示。

代码 20.9 文件内容复制: `Transform.java`

```
public class Transform {
    public static void main(String[] args) {
        try {
            String str;                                //字符串变量
            //输入流 br 的设置
            File srcFile = new File("C:\\chang.txt");    //创建 File 对象
            //创建文件输入流对象(字节形式)
            FileInputStream filein = new FileInputStream(srcFile);
            //实现由字节形式转化成字符形式
            InputStreamReader isr = new InputStreamReader(filein);
            BufferedReader br = new BufferedReader(isr); //增加缓存功能
            //输出流 bw 的设置
            File desFile = new File("C:\\chang_1.txt"); //创建 File 对象
            //创建文件输出流对象(字节形式)
            FileOutputStream fileout = new FileOutputStream(desFile);
            //实现由字节形式转化成字符形式
            OutputStreamWriter osw = new OutputStreamWriter(fileout);
            BufferedWriter bw = new BufferedWriter(osw); //增加缓存功能
            //输出相应信息
            System.out.println("复制文件: " + srcFile.length() + "字节");
            while ((str = br.readLine()) != null) {      //实现复制功能
                bw.write(str);                            //写入目标文件
                bw.newLine();                             //换行符号
            }
            bw.flush();                                  //将缓存区中的数据全部写出
            System.out.println("复制完成");              //输出相应信息
            // chang_1.txt 文件的内容
            //创建文件 chang_1.txt 的输入流对象 br
            br = new BufferedReader(new InputStreamReader(new FileInputStream(
                new File("C:\\chang_1.txt"))));
            while ((str = br.readLine()) != null) {      //实现信息的输出
                System.out.println(str);                  //输出文件内容
            }
        }
    }
}
```



```

    }
    //关闭流对象
    br.close();
    bw.close();
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("using: java useFileStream src des");
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
}

```

在具体运行 Transform 类之前, 首先创建名为 chang.txt 的文件, 其内容如图 20.23 所示。具体运行时, 控制台窗口如图 20.24 所示。该文件的复制文本 chang_1.txt 内容如图 20.25 所示。

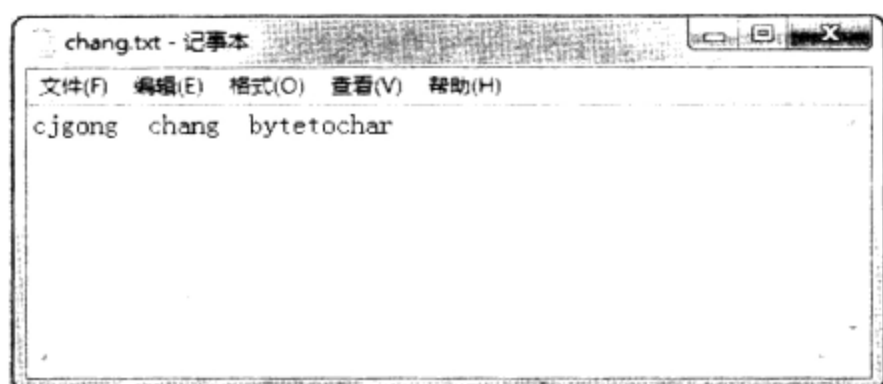


图 20.23 chang 文件的内容

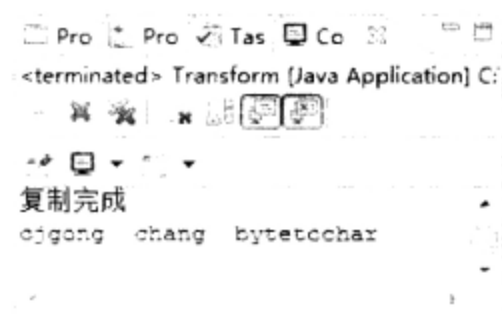


图 20.24 控制台窗口



图 20.25 chang_1 文件的内容

【代码解析】

- ❑ 在上述代码中主要实现两个功能: 将文件 chang 的内容复制到 chang_1 文件里; 将文件 chang_1 里的内容输出到控制台窗口。文件流的具体过程如图 20.26 所示。
- ❑ 在上述代码中通过图 20.26 所示的过程获取输入流和输出流, 首先创建了与文件 chang 和 chang_1 相关连的文件输入流对象 (filein) 和文件输出流对象 (fileout)。然后通过类 InputStreamReader 和 OutputStreamWriter 使得输入流 (isr) 和输出流 (osw) 分别具有了操作字符的功能, 最后通过 BufferedInputStream 和 BufferedInputStreamr 类使输入流 (br) 和输出流 (bw) 分别具有了缓存功能。
- ❑ 在功能一中, 首先创建了与文件 chang 相关连 srcFile 对象的文件输入流对象, 以及与文件 chang_1 相关连 desFile 对象的文件输出流对象。接着再把这两个对象分别作为参数传入 FileInputStream 和 FileOutputStream 类中获取以字节为单位的输入流 (filein) 和输出流 (fileout)。最后为了实现以字符为单位, 于是通过过滤流类

InputStreamReader 和 OutputStreamWriter 使输入流 (filein) 和输出流 (fileout) 具有字符操作的功能, 为了方便字符的操作, 通过过滤流类 BufferedReader 和 BufferedWriter, 使输入流 (filein) 和输出流 (fileout) 具有缓存的功能,

- 带有缓存功能后的输入流 (br) 和输出流 (bw) 存在如下的好处: 可以通过 br.readLine() 方法直接读取一行字符, 而不需要一个字符一个字符的读取; 同时也可以通过 bw.writer() 方法直接输出字符串, 而不要由其他类型 (Int、byte 等) 转化成字符串。

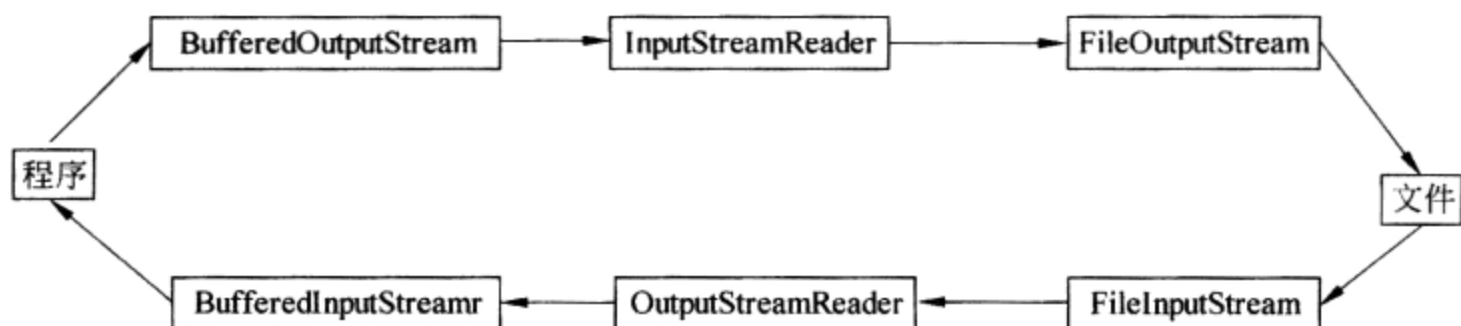


图 20.26 输入流和输出流操作

20.3.3 过滤流特定数据类型类

在 Java 语言中支持各种基本类型数据的直接读写, 所谓基本类型就是 int、double 等类型。查看 API 帮助文档, 可以发现在 I/O 包中存两个类 DataInputStream 和 DataOutputStream, 前者用来实现直接输入各种基本类型数据, 后者用来实现直接输出各种基本类型数据。

下面通过实现文件内容复制功能的事例, 来讲解 DataInputStream 和 DataOutputStream 类的基本使用, 具体步骤如下。

(1) 在 BaseType 类中, 通过以特定数据类型访问实现内容复制功能, 具体内容如代码 20.10 所示。

代码 20.10 文件内容复制: BaseType.java

```

public class BaseType {
    public static void main(String[] args) {
        try {
            byte[] data = new byte[1];           //创建字节数组对象
            String str;                           //创建字符串对象
            //输入流的设置
            File srcFile = new File("C:\\base.txt");
            FileInputStream filein = new FileInputStream(srcFile);
            BufferedInputStream bi = new BufferedInputStream(filein);
            //创建 DataInputStream 对象
            DataInputStream di = new DataInputStream(bi);
            //输出流的设置
            File desFile = new File("C:\\base_1.txt");
            FileOutputStream fileout = new FileOutputStream(desFile);
            BufferedOutputStream bo = new BufferedOutputStream(fileout);
            //创建 DataOutputStream 对象
            DataOutputStream doo = new DataOutputStream(bo);
            System.out.println("复制文件: " + srcFile.length() + "字节");
        }
    }
}
  
```

```

//实现复制功能
while (di.read(data) != -1) {
    doo.write(data);
}
doo.flush();
System.out.println("复制完成");
di = new DataInputStream(new BufferedInputStream(
    new FileInputStream(srcFile)));
while (di.read(data) != -1) {
    String str1 = data.toString();
    System.out.println(str1);
}
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("using: java useFileStream src des");
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
}

```

在具体运行 BaseType 类之前, 首先创建名为 base.txt 的文件, 其内容如图 20.27 所示。具体运行时, 控制台窗口如图 20.28 所示。该文件的复制文本 base_1.txt 内容如图 20.29 所示。

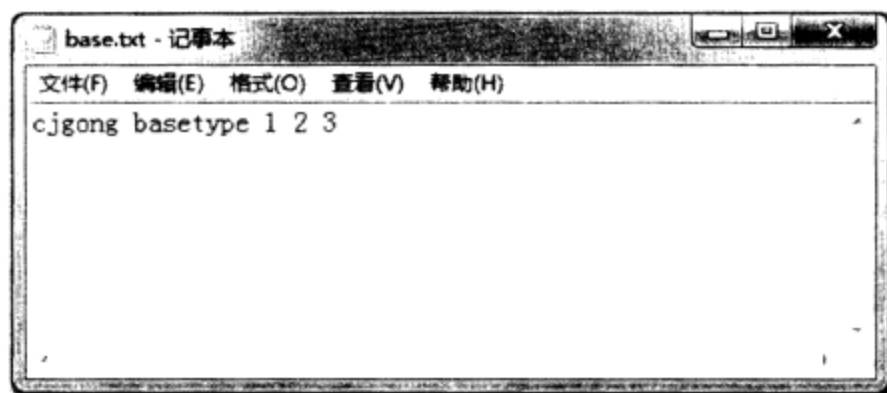


图 20.27 base 文件的内容

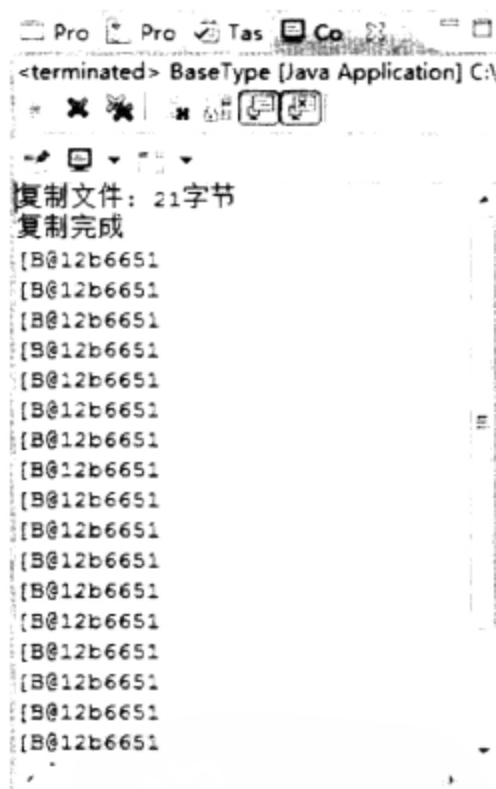


图 20.28 控制台窗口

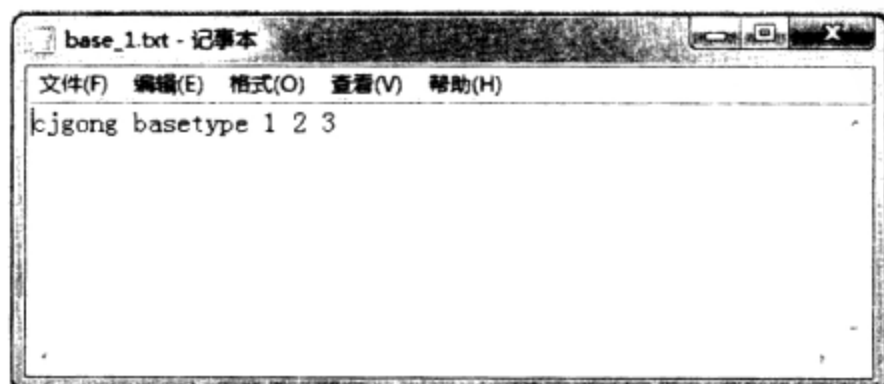


图 20.29 chang_1 文件的内容

注意：从运行结果可以看出，虽然可以从 serial.txt 文件中读取并还原写入的内容，但是在控制台输出的是字节。

【代码解析】

- 在上述代码中经过如下过程获取输入流和输出流：首先创建与文件 base 相关连 srcFile 对象的文件输入流对象，以及与文件 base _1 相关连 desFile 对象的文件输出流对象。接着再把这两个对象分别作为参数传入 FileInputStream 和 FileOutputStream 类中获取以字节为单位的输入流（filein）和输出流（fileout）。最后再通过 BufferedInputStream 和 BufferedInputStreamr 类使得输入流（bi）和输出流（bo）分别具有了缓存功能，通过 DataInputStream 和 DataOutputStream 类使得输入流（di）和输出流（doo）分别具有了操作基本类型的功能。文件流的具体过程如图 20.30 所示。
- 在上述代码中，通过 di.read(data)代码获取源文件内容的字节数组，然后通过 doo.write(data)把内容的字节数组内容输出到目标文件中。

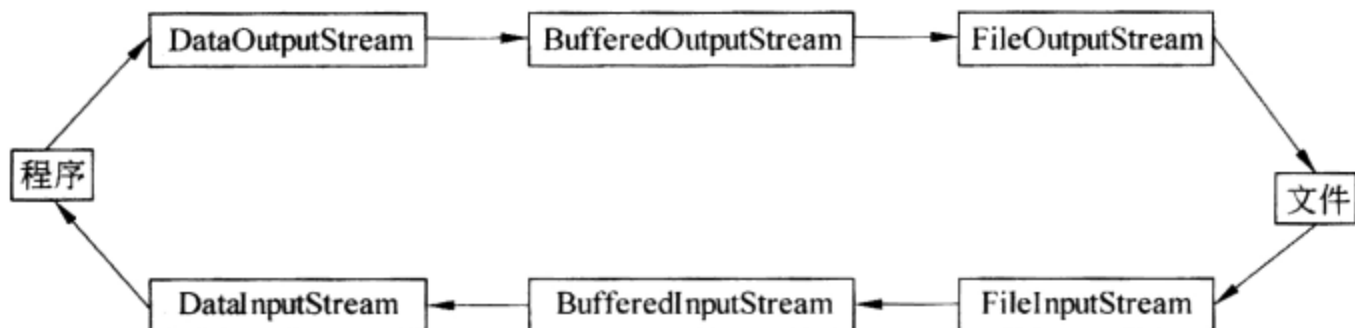


图 20.30 输入流和输出流操作

(2) 在 DataInputStream 和 DataOutputStream 这两个类中存在许多方法，下面通过一个具体的实例名为 BaseTypeMehtod 类，来讲解这些类的方法，具体内容如代码 20.11 所示。

代码 20.11 文件内容复制：BaseTypeMehtod.java

```

public class BaseTypeMehtod {
    public static void main(String[] args) {
        try {
            byte[] data = new byte[1];           //创建字节数组对象
            String str;                          //创建字符串变量
            //设置文件输出流
            File desFile = new File("C:\\\\bianma.txt");
            FileOutputStream fileout = new FileOutputStream(desFile);
            BufferedOutputStream bo = new BufferedOutputStream(fileout);
            //获取 DataOutputStream 类对象
            DataOutputStream doo = new DataOutputStream(bo);
            //利用 doo 的各种方法向文件写入信息
            doo.writeUTF("cj 常功");
            doo.writeBytes("cj 常功");
            doo.writeChars("cj 常功");
            doo.flush();                         //缓存区的操作
            //输出相应的信息
            System.out.println("一共写入" + desFile.length() + "个字节");
            //设置文件输入流
            FileInputStream filein = new FileInputStream(desFile);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
  
```

```

BufferedInputStream bi = new BufferedInputStream(filein);
//获取 DataInputStream 类对象
DataInputStream di = new DataInputStream(bi);
//利用 di 的各种方法读取文件的信息
System.out.println("-----");
System.out.println("writeUTF 方法写入的字符: " + di.readUTF());
System.out.println("-----");
for (int i = 0; i < 12; i++) { //循环读出文件的信息
    int il = di.read(data);
    if ((il != -1) && i < 4) {
        String str1 = data.toString();
        System.out.println("前四个为 writeBytes 方法写入的字符: " +
            str1);
    } else {
        String str2 = data.toString();
        System.out.println("后八个为 writeChars 方法写入的字符: " +
            str2);
    }
}
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("using: java useFileStream src des");
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
}

```

具体运行上述代码时, 控制台窗口如图 20.31 所示。所创建的名为 bianma.txt 的文件, 内容如图 20.32 所示。

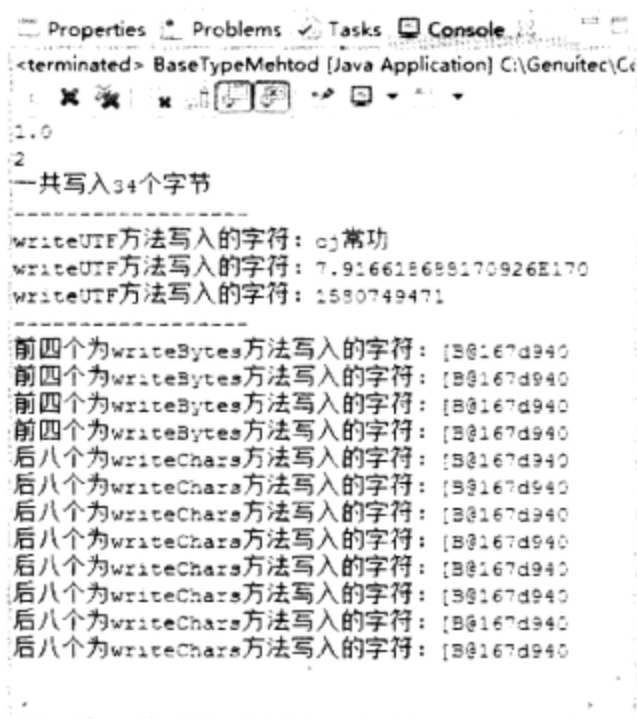


图 20.31 运行结果

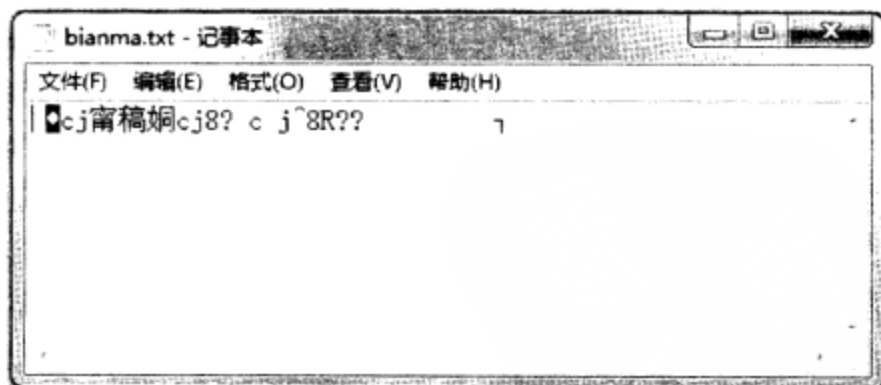


图 20.32 文件内容

注意: 从运行结果可以看出虽然用记事本打开 serial.txt 是乱码, 但是在具体写入时却显示正常。

【代码解析】

□ 在上述代码中主要实现用文件输出流对象 doo 向文件 bianma 写入信息, 然后再通

过文件输入流对象 `bi` 从文件 `bianma` 读取信息，最后在控制台窗口输出信息。文件流的具体过程如图 20.33 所示。

- ❑ 在上述代码中通过图 20.33 所示的过程获取输入流和输出流，首先创建了与文件 `bianma` 相关连的文件输入流对象（`fileout`）和文件输出流对象（`filein`）。然后通过类 `BufferedInputStream` 和 `BufferedInputStreamr` 使得输入流（`bi`）和输出流（`bo`）分别具有了缓存功能。最后通过类 `DataInputStream` 和 `DataOutputStream` 使得输入流（`di`）和输出流（`doo`）分别具有了操作基本类型的功能。
- ❑ 在 `DataOutputStream` 中存在 3 种直接写入字符串的方式，它们分别为 `writeBytes()` 方法只将字符串中每一个字符中低字节的内容写入目标设备；`writeChars()` 方法将字符串中每一个字符的两个字节的内容都写入目标设备；`writeUTF` 对字符串按照 UTF 格式写入目标设备。而在 `DataInputStream` 中只存在一个名为 `readUTF()` 的方法返回字符串。对于字符串“cj 常功”，当用 UTF 编码时为 10 个字节；当用 `writeBytes()` 方法写入时，由于每个字符为 2 个字符，而其只写入低字节即 4 个字节，因此最后 `writeChars()` 方法写入为 8 个字节。

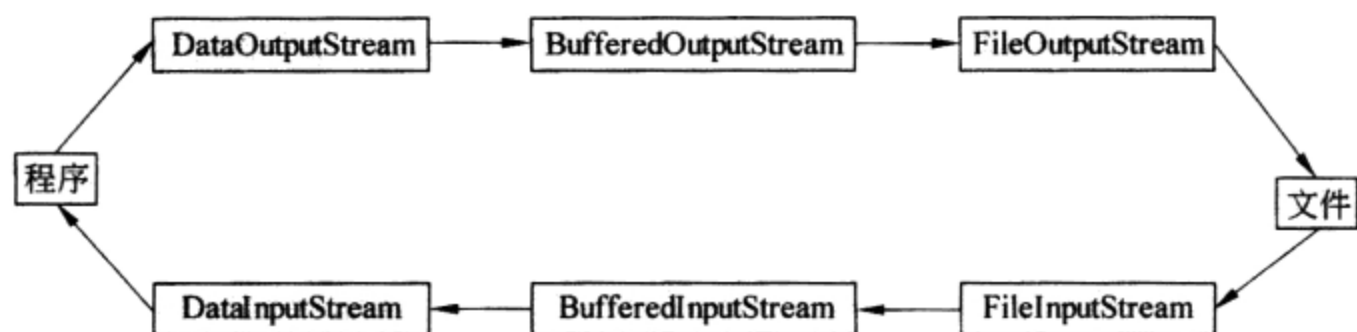


图 20.33 输入流和输出流操作

注意：通过一个连续的字节流来读取一个字符串（流中的一段内容）时，如果没有特殊的标记作为一个字符串的结尾或者不知道字符串中字节流的长度，程序是没有办法读取该字符串的。前面 3 个写入方法中，只有 `writeUTF()` 方法会向目标设备写入字符串的长度，所以只能通过 `readUTF()` 方法读取写入的字符串，而不存在 `readBytes()` 和 `readChars()` 方法。

20.3.4 过滤流对象序列化类

在 Java 语言中可以通过特殊数据类型的过滤器流类，把各种基本数据类型如 `int`、`float` 等保存到文件中并从文件中读取。但如果是对象呢？也可以直接保存到文件并从文件中读取吗？查看 API 帮助文档可以发现，在 I/O 包中有两个类 `ObjectInputStream` 和 `ObjectOutputStream`，前者用来实现直接输入序列化的对象，而后者用来实现直接输出序列化的对象。

所谓序列化，就是将实现了 `Serializable` 接口的对象转换为连续的字节数据，这些数据能够被还原回来。上句话中所说的对象是指对象的属性数据，因为只有属性才会存放一些数据，而方法本身就是程序代码，做输入输出没有任何意义。

下面通过具体的事例来讲解 `ObjectInputStream` 和 `ObjectOutputStream` 类的基本用法，具体步骤如下。

(1) 首先创建写入文件的雇员对象类型, 具体内容如代码 20.12 所示。

代码 20.12 雇员类: Empoly.java

```
class Empoly implements Serializable {
    //雇员的成员变量
    int id;                //Id 属性
    String name;           //姓名属性
    int age;               //年龄属性
    String department;     //部分属性
    //带有参数的构造函数
    public Empoly(int id, String name, int age, String department) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.department = department;
    }
}
```

【代码解析】

由于该对象需要做为序列化操作的对象, 所以其实现了 Serializable 接口。该接口是一个标记接口, 即没有定义任何成员。

(2) 在 Serialization 类中, 通过特殊对象类型方式实现 Empoly 对象的写入和读取, 具体内容如代码 20.13 所示。

代码 20.13 雇员信息的写入和读取: serialization1.java

```
public class serialization1 {
    public static void main(String args[]) throws IOException,
        ClassNotFoundException {
        Empoly emp = new Empoly(1, "cjgong", 50, "kezhang");
                                //创建雇员对象 emp
        File srcFile = new File("serial.txt"); //创建 File 对象
        //设置文件输出流
        FileOutputStream fos = new FileOutputStream(srcFile);
                                //创建 FileOutputStream 对象
        //创建 ObjectOutputStream 对象
        ObjectOutputStream os = new ObjectOutputStream(fos);
        //设置文件输入流
        FileInputStream fi = new FileInputStream(srcFile);
                                //创建 FileInputStream 对象
        //创建 ObjectInputStream 对象
        ObjectInputStream si = new ObjectInputStream(fi);
        //输出相应信息
        System.out.println("开始写入文件, 该文件的大小为: " + srcFile.length()
            + "字节");
        try {
            os.writeObject(emp); //向文件写入相应信息
            os.close();          //关闭输出流对象
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
        //输出相应信息
        System.out.println("写入文件完成, 该文件大小为: " + srcFile.length() +
            "字节");
    }
}
```

```

emp = null; //为对象 emp 赋值
try {
    emp = (Empoly) si.readObject(); //获取文件中的信息
    si.close(); //关闭输入流
} catch (IOException e) {
    System.out.println(e.getMessage());
}
//输出获取到的信息
System.out.println("输出文件里的信息: ");
System.out.println("雇员的 ID 为:" + emp.id);
System.out.println("雇员的名字为:" + emp.name);
System.out.println("雇员的 age 为:" + emp.age);
System.out.println("雇员的 department 为:" + emp.department);
}
}

```

运行 serialization1 类，控制台窗口如图 20.34 所示。

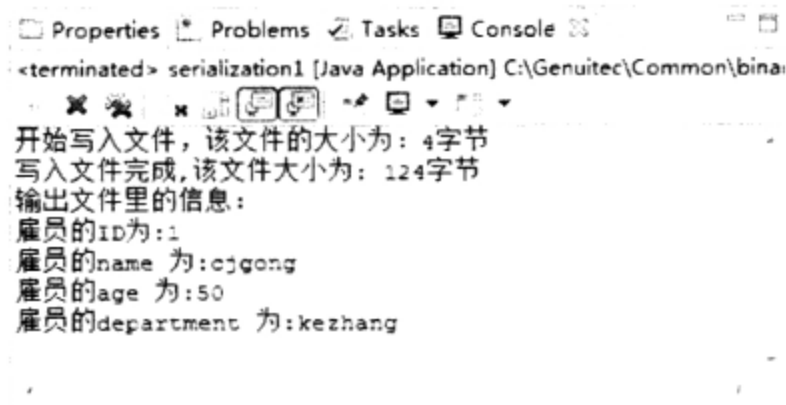


图 20.34 运行结果

【代码解析】

- ❑ 在上述代码中主要实现用文件输出流对象 `os` 向文件 `serial` 写入信息，然后再通过文件输入流对象 `si` 从文件 `serial` 读取信息并在控制台窗口输出信息，文件流的具体过程如图 20.35 所示。
- ❑ 在上述代码中通过图 20.35 所示的过程获取输入流和输出流，首先创建了与文件 `serial` 相关连的文件输入流对象（`fos`）和文件输出流对象（`fi`）。然后通过 `ObjectInputStream` 和 `ObjectOutputStream` 类使得输入流（`os`）和输出流（`si`）分别具有了直接操作对象的功能。
- ❑ 在具体写入文件时，通过 `os.writeObject()` 方法直接把雇员对象（`emp`）写入文件；在具体读取文件时，通过 `si.readObject()` 方法获取 `Object` 对象，然后再强制转换成雇员类型（`Empoly`）。

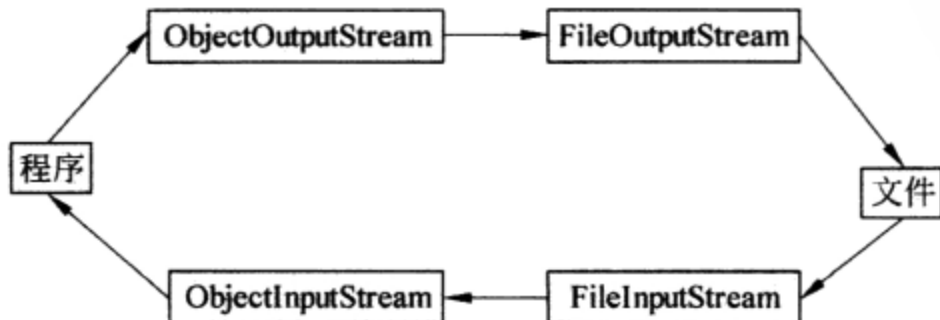


图 20.35 输入流和输出流操作

20.3.5 过滤流打印类

查看 API 帮助文档，可以发现在 I/O 包中存两个打印的类 `PrintStream` 和 `PrintWriter`，前者可以直接输出各种类型的数据，而后者可以把 Java 语言中的内置类型以字符形式送到相应的输出流中，如果浏览可以通过文本形式来实现。

下面通过具体的实例来讲解 `PrintStream` 和 `PrintWriter` 类的基本用法，具体步骤如下。

(1) 实现向文件写入随机数据功能的类 `StreamTest`，具体内容如代码 20.14 所示。

代码 20.14 向文件写入内容: `StreamTest.java`

```
public class StreamTest {
    public static void main(String args[]) {
        PrintStream ps;                                //创建 PrintStream 对象
        try {
            File file = new File("C:\\\\print.txt"); //创建 File 对象
            if (!file.exists())                     //如果文件不存在
                file.createNewFile();               //创建新文件
            //为对象 ps 赋值
            ps = new PrintStream(new FileOutputStream(file));
            Random r = new Random();                 //创建 Random 类对象
            int rs;                                  //创建变量 rs
            for (int i = 0; i < 5; i++) {             //创建 5 个随机整数
                rs = r.nextInt(100);
                System.out.println(rs);
                ps.println(rs + "\\t");               //写入文件
            }
            ps.close();                               //关闭
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

在具体运行 `StreamTest` 类之前，首先创建名为 `print.txt` 的文件，其内容如图 20.36 所示。具体运行时，控制台窗口如图 20.37 所示。该文件的复制文本 `print_1.txt`，内容如图 20.38 所示。

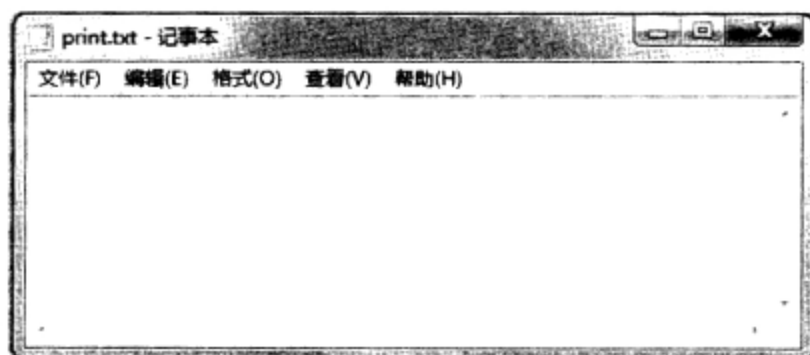


图 20.36 print 文件写入前内容

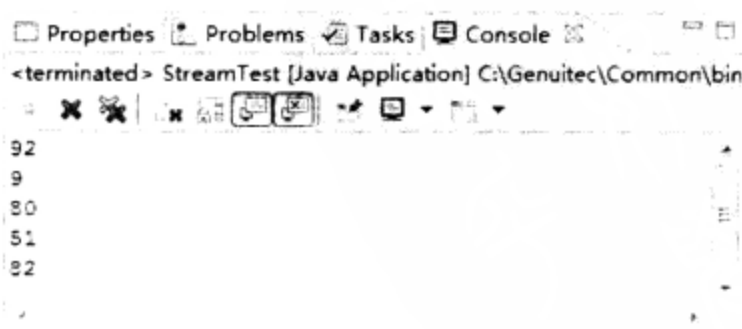


图 20.37 运行结果

【代码解析】

上述代码中首先获取文件对象 `file`，接着把该文件对象的 `FileOutputStream` 流对象作为参数获取 `PrintStream` 对象 `ps`，最后通过 `ps.println()` 方法把随机产生的 5 个数字写入文件中。



图 20.38 print 文件写入后内容

(2) 实现文件内容复制功能的类 WriteTest，具体内容如代码 20.15 所示。

代码 20.15 实现文件复制: WriteTest.java

```
public class WriteTest {
    public static void main(String[] args) {
        try {
            int data; //创建 int 类型变量
            //设置输入流对象
            File srcFile = new File("C:\\\\print.txt"); //创建 File 类型对象
            FileReader filein = new FileReader(srcFile);
            //创建文件输入流对象
            //创建带有缓存功能的输入流对象
            BufferedReader br = new BufferedReader(filein);
            //设置输出流对象
            File desFile = new File("C:\\\\print_1.txt"); //创建 File 类型对象
            FileWriter fileout = new FileWriter(desFile);
            //创建文件输出流对象
            //创建带有打印功能的输出流对象
            PrintWriter pw = new PrintWriter(fileout);
            //输出相应信息
            System.out.println("复制文件: " + srcFile.length() + "字节");
            //实现复制功能
            while ((data = br.read()) != -1) {
                pw.print(data);
                System.out.println(data); //输出相应信息
            }
            pw.flush(); //将缓存区中的数据全部写出
            System.out.println("复制完成"); //输出相应信息
            //关闭流对象
            br.close();
            pw.close();
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("using: java useFileStream src des");
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

在具体运行 WriteTest 类之前，首先创建名为 print.txt 的文件，其内容如图 20.39 所示。具体运行时，控制台窗口如图 20.40 所示。print.txt 文件的复制文本 print_1.txt 其内容如图 20.41 所示。

【代码解析】

在上述代码中，首先创建了与文件 print 相关联的 filein 对象的文件输入流对象，以及

与文件 print_1 相关联的 fileout 对象的文件输出流对象。接着再把这两个对象分别作为参数传入 `BufferedReader` 和 `PrintWriter` 类中, 获取以字符为单位带有缓存功能的输入流(`br`), 以及以字符为单位带有打印功能的输出流(`pw`)。最后通过遍历把文件 `print` 中的内容复制到文件 `print_1` 里。文件流的具体过程如图 20.42 所示。



图 20.39 print 文件的内容

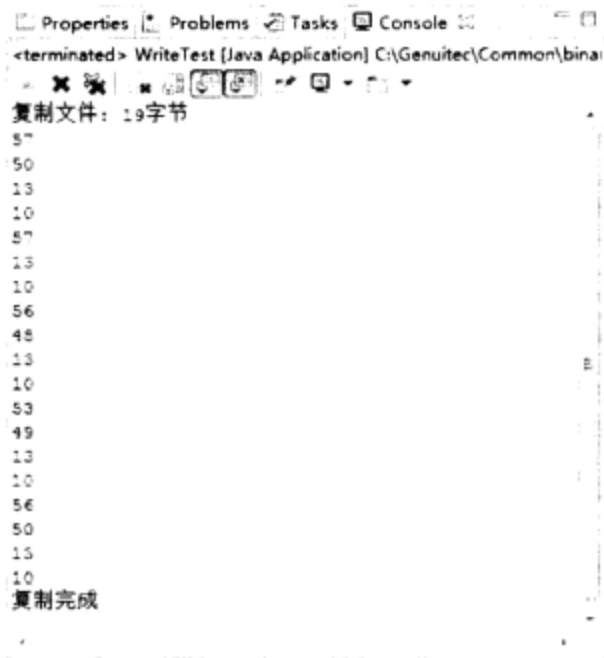


图 20.40 输出结果



图 20.41 print_1 文件的内容

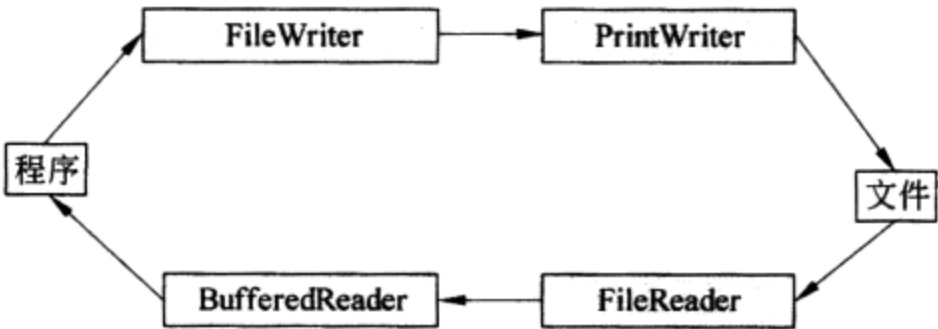


图 20.42 输入流和输出流操作

20.4 小 结

本章首先通过对文件的操作来创建对日记文件的创建, 然后通过对文件的访问来向文件里输入内容, 最后通过对文件的操作来实现文件的查看和删除功能。

在本章的最后还详细介绍了 Java 语言中 I/O 的高级知识——过滤流类, 分别为实现缓存功能的过滤类、实现字节和字符转换的过滤类、实现特定数据类型的过滤类、实现对象序列化的过滤类和实现打印功能的过滤类。

第 21 章 查找和替换项目

(GUI+字符串处理)

本章将通过 AWT 和 Swing 组件实现查找和替换界面,通过字符串操作来模拟 Windows 系统的查找和替换功能。“查找和替换”项目综合了图形用户界面的所有知识点和字符串的操作。

本章的学习目标如下:

- ❑ 掌握组件和面板的使用方法;
- ❑ 理解布局管理器 and 事件处理的使用方法;
- ❑ 了解字符串操作。

21.1 查找和替换原理

“查找和替换”项目通过字符串的相关操作方法和图形用户界面,模拟 Windows 系统中的查找和替换功能,在具体运行时,只要单击相应按钮就可以实现查找和替换的功能。

21.1.1 项目结构框架分析

“查找和替换”项目利用 AWT 组件和 Swing 两种类型的组件来实现。查找和替换项目目录如图 21.1 所示,各个包的功能如下。

- ❑ 包 com.cjg.awt: 利用 AWT 组件实现界面。
- ❑ 包 com.cjg.swing: 利用 Swing 组件实现界面。

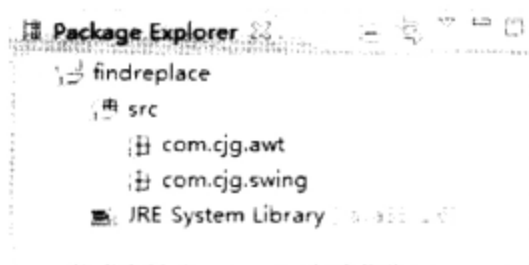


图 21.1 项目目录

21.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括查找和替换的初始化界面、输入查找和替换记录、清空查找和替换记录、查看查找和替换的所有记录及查找和替换记

录的操作。

1. 初始化界面

当运行查找和替换项目中的 FindReplace 类后，会出现如图 21.2 所示的初始界面——文本查找和替换输入界面。

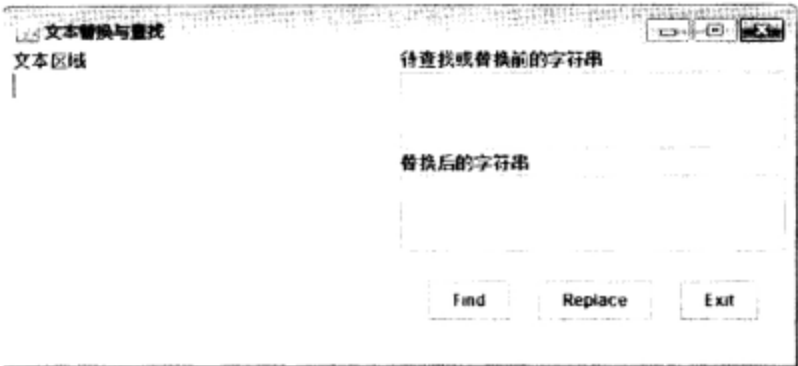


图 21.2 初始界面

2. 查找功能

当出现初始界面后，如果想实现查找的功能，即查找“文本区域”内容中的某个字符，首先在“文本区域”中输入内容，然后在“待查找或替换前的字符串”区域中输入相应字符，最后单击 Find 按钮弹出查找结果的对话框，具体过程如图 21.3 所示。

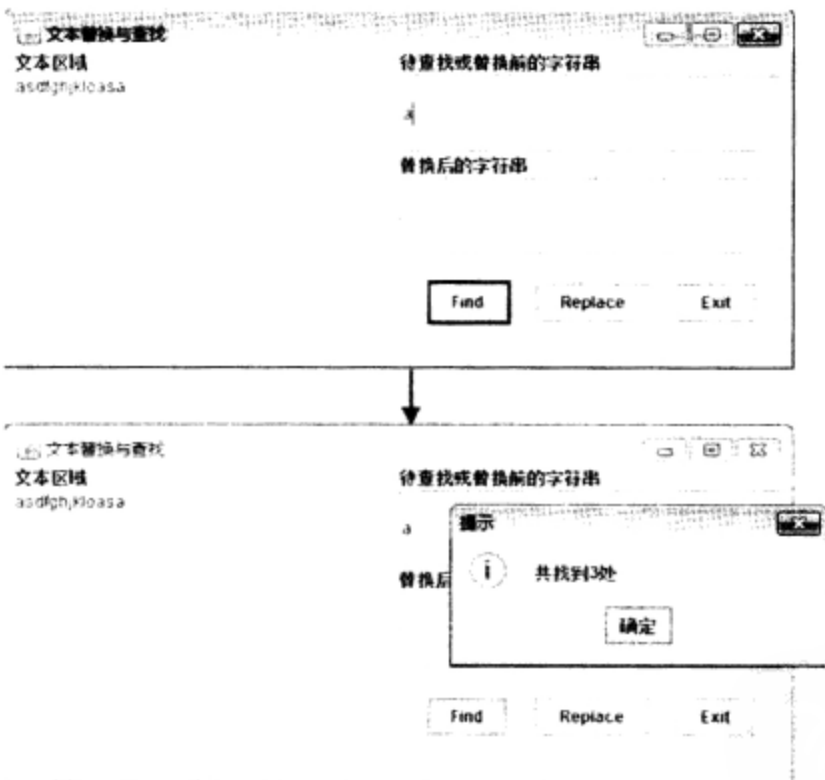



图 21.3 “查找”过程

3. 替换功能

当出现初始界面后，如果想实现替换的功能，即用“替换后的字符串”区域中的字符替换“文本区域”内容中的“待查找或替换前的字符串”的内容，除了“查找”功能中的步骤外，还需要在“替换后的字符串”区域中输入需要替换的字符，最后单击 Replace 按钮会弹出替换结果的对话框。如果单击该对话框的“确定”按钮，会在“文本区域”中显示出更新后的内容，具体过程如图 21.4 所示。

4. 退出功能

如果想实现退出功能，可以单击 Exit 按钮或窗口的右上角  按钮，如图 21.5 所示。

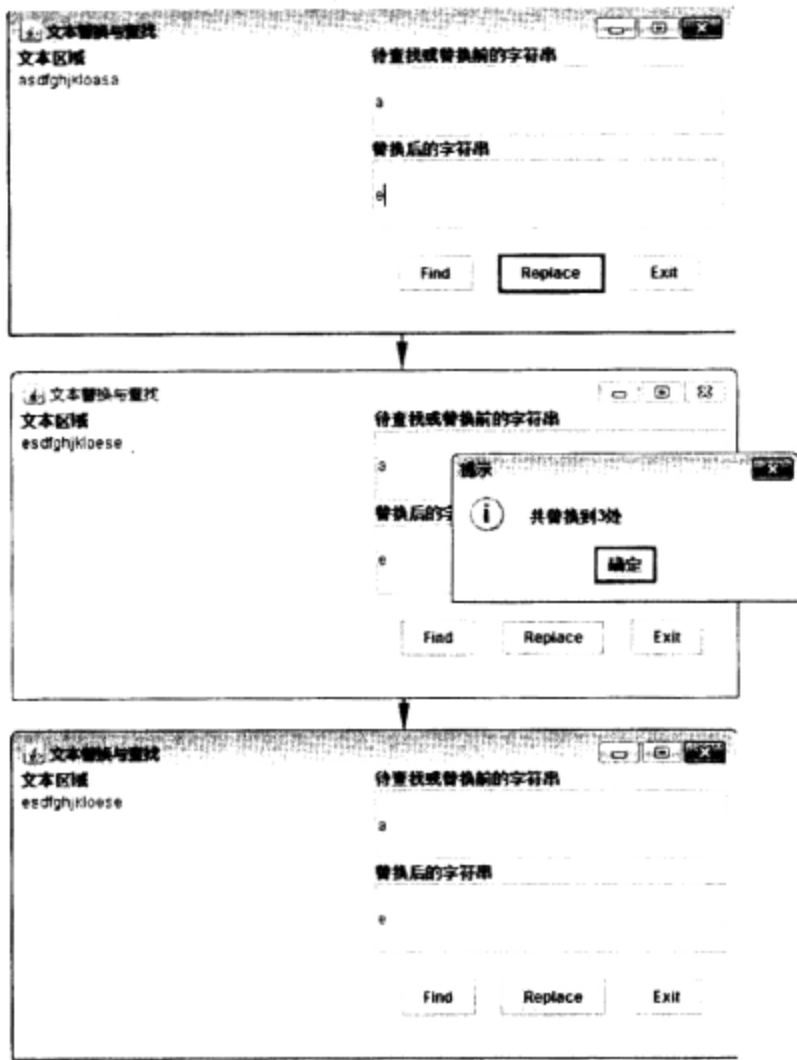


图 21.4 “替换”过程

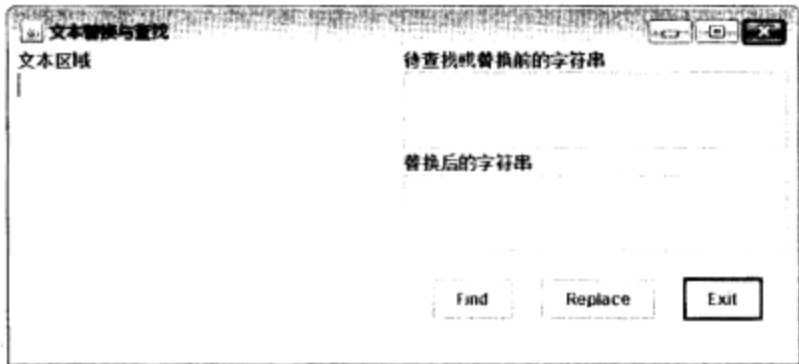


图 21.5 退出功能

21.2 查找和替换项目——利用 AWT 组件

查找和替换项目具体程序架构，如图 21.6 所示，即包含一个查找和替换输入界面的类和实现字符串处理的工具类。在该项目中，主要通过 AWT 组件来实现该项目的界面。

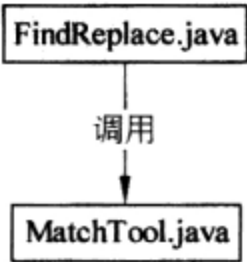


图 21.6 程序关系图

21.2.1 设计项目的界面——查找和替换输入界面

FindReplace 为查找和替换项目界面的类，在该类中不仅实现了主界面，还会实现显示“查找”和“替换”结果的对话框，具体内容如代码 21.1 所示，该类的成员变量如图 21.7 所示。

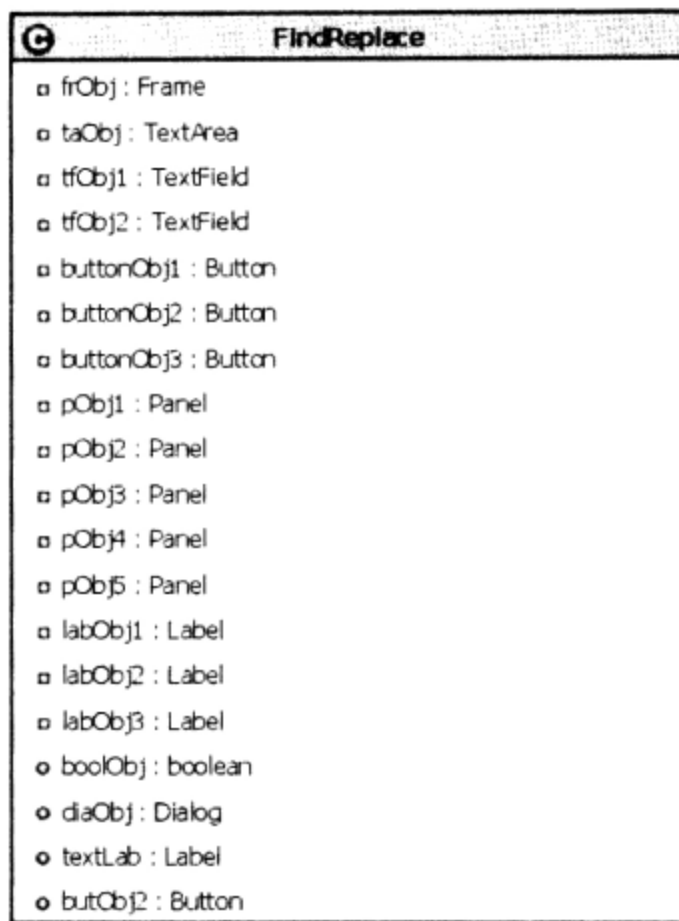


图 21.7 成员变量的 UML 图

代码 21.1 主界面: FindReplace.java

```

public class FindReplace implements MouseListener, WindowListener {
    //创建各种成员变量
    private Frame frObj;                                //主窗口对象
    private TextArea taObj;                             //“文本区域”变量
    //“待查找或替换前的字符串”和“替换后的字符串”变量
    private TextField tfObj1, tfObj2;
    //“查找”、“替换”和“退出”按钮变量
    private Button buttonObj1;
    private Button buttonObj2;
    private Button buttonObj3;
    private Panel pObj1, pObj2, pObj3, pObj4, pObj5;
    private Label labObj1, labObj2, labObj3;            //显示信息面板变量
    boolean boolObj = false;                          //布尔变量
    Dialog diaObj;                                     //对话框对象
    Label textLab;
    Button butObj2 = new Button("OK");                 //对话框中的按钮对象
    public static void main(String args[]) {           //主方法
        FindReplace examObj = new FindReplace();      //创建 FindReplace 对象
        examObj.create();                             //调用 create() 方法
    }
    //实现创建主界面及创建查找和替换字符串结果信息的对话框方法
    public void create() {
        frObj = new Frame("文本查找与替换");          //创建主界面
        taObj = new TextArea();                       //创建“文本区域”对象
        //创建待查找或替换的字符串的文本框
        tfObj1 = new TextField();
        tfObj2 = new TextField();                     //创建替换后的字符串的文本框
        //初始化 3 个按钮对象, 分别用来实现查找、替换和退出操作
        buttonObj1 = new Button("Find");              //创建 Find 按钮
        buttonObj2 = new Button("Replace");            //创建 Replace 按钮
    }
}
  
```

```

buttonObj3 = new Button("Exit");           //创建 Exit 按钮
//初始化 3 个 Label 对象, 用于显示提示信息
labObj1 = new Label("文本区域");
labObj2 = new Label("待查找或替换前的字符串");
labObj3 = new Label("替换后的字符串");
//初始化 5 个 Panel 对象, 用于控制主界面上各组件的位置和大小
pObj1 = new Panel();
pObj2 = new Panel();
pObj3 = new Panel();
pObj4 = new Panel();
pObj5 = new Panel();
// pObj1 用于控制文件区域和相关提示信息的相对位置
pObj1.setLayout(new BorderLayout()); //设置布局管理器
pObj1.add("North", labObj1);         //添加“文件区域提示信息”到北面
pObj1.add("Center", taObj);          //添加“文件区域”到中间
// pObj2 用于控制“待查找或替换的字符串”文本框和相关提示信息的相对位置
pObj2.setLayout(new BorderLayout()); //设置布局管理器
pObj2.add("North", labObj2);
pObj2.add("Center", tfObj1);
// pObj3 用于“替换后的字符串”文本框和相关提示信息的相对位置
pObj3.setLayout(new BorderLayout()); //设置布局管理器
pObj3.add("North", labObj3);
pObj3.add("Center", tfObj2);
// pObj4 用于控制 Find 按钮、Replace 按钮和 Exit 按钮的相对位置
//设置布局管理器
pObj4.setLayout(new FlowLayout(FlowLayout.CENTER, 20, 20));
pObj4.add(buttonObj1);
pObj4.add(buttonObj2);
pObj4.add(buttonObj3);
// pObj5 用于控制 pObj2, pObj3 和 pObj4 的相对位置
pObj5.setLayout(new GridLayout(3, 1)); //设置布局管理器
pObj5.add(pObj2);
pObj5.add(pObj3);
pObj5.add(pObj4);
//最后添加到主窗口中
frObj.setLayout(new GridLayout(1, 2)); //设置主窗口布局管理器
frObj.add(pObj1);
frObj.add(pObj5);
//为 Find 按钮、Replace 按钮和 Exit 按钮添加事件监听器
buttonObj1.addMouseListener(this);
buttonObj2.addMouseListener(this);
buttonObj3.addMouseListener(this);
//下面两个语句设置主界面的大小并让主界面可见
frObj.setSize(560, 260);
frObj.setVisible(true);
//设置对话框对象
diaObj = new Dialog(frObj);           //创建对话框对象
//设置对话框布局管理器
diaObj.setLayout(new FlowLayout(FlowLayout.CENTER, 40, 20));
//创建用于显示查找或替换字符串的次数的 Label 组件
textLab = new Label(" ");
diaObj.add(textLab);                  //添加 textLab 到对话框中
diaObj.add(butObj2);                  //添加按钮对象到对话框中
butObj2.addMouseListener(this);     //添加事件监听器
diaObj.setSize(200, 100);            //设置对话框的大小
}

```

```
//鼠标监听的方法
public void mouseClicked(MouseEvent e) { //鼠标单击的事件
...
}
...
}
```

【代码解析】

上述代码实现了查找和替换项目界面，该用户界面涉及的具体容器、对象和布局如图 21.8 所示。

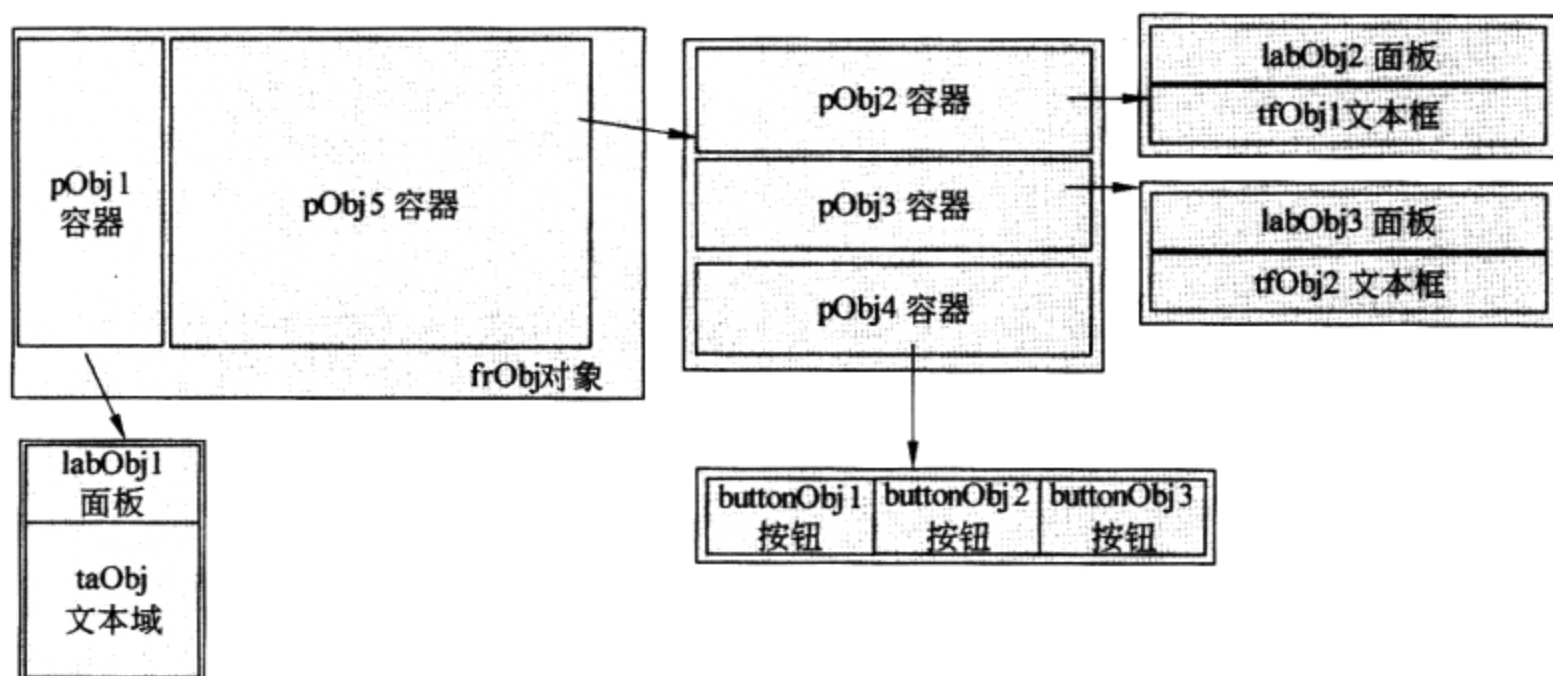


图 21.8 布局

21.2.2 各种按钮的事件处理

mouseClicked()方法主要用来实现处理鼠标单击事件，即分别实现单击 Find 按钮、Replace 按钮和 Exit 按钮后所要调用的代码，具体内容如代码 21.2 所示，主界面中涉及的方法如图 21.9 所示。

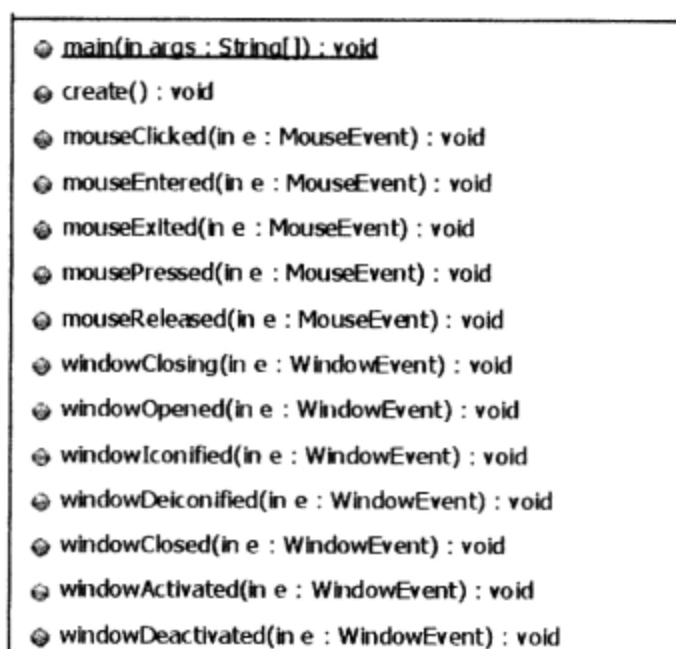


图 21.9 方法的 UML 图

代码 21.2 单击按钮的事件处理方法: mouseClicked

```

public void mouseClicked(MouseEvent e) {           //鼠标单击的事件
    //下面这个语句用于获得事件源按钮
    Button butObj = (Button) (e.getSource());
    //各种按钮单击方法
    if (butObj.getLabel() == "Exit") {           //当发生的事件源为退出按钮
        System.exit(0);                          //退出程序功能
    }
    //当单击按钮为查找或替换按钮时
    if (butObj.getLabel() == "Find" || butObj.getLabel() == "Replace") {
        String strObj1 = taObj.getText();         //文本区域的内容
        String strObj2 = tfObj1.getText();        //第一个文本框中的内容
        int matchNum = 0;                        //字符串匹配的次数
        int cursorPos = taObj.getCaretPosition(); //光标当前的位置
        MatchTool classObj = new MatchTool();    //创建 matchFunod 对象
        //处理 Find 按钮事件
        if (butObj.getLabel() == "Find") {
            //通过调用 matchFun 类的 strFind() 方法计算出字符串匹配的次数
            matchNum = classObj.strFind(strObj1, strObj2, cursorPos);
            //重新设置对话框上 Label 对象的文本内容
            textLab.setText("共找到" + matchNum + "处");
            diaObj.show();                        //显示对话框
        }
        //处理 Replace 按钮事件
        if (butObj.getLabel() == "Replace") {
            String strObj3 = tfObj2.getText();    //第二个文本框中的字符串
            //通过调用 matchFun 类中的 strReplace 按钮计算字符串匹配次数
            matchNum = classObj.strReplace(strObj1, strObj2, strObj3,
                cursorPos);
            //重新设置对话框上 Label 对象的文本内容
            textLab.setText("共替换到" + matchNum + "处");
            //刷新字符串替换后文本区域的文字显示
            StringBuffer taText = classObj.repStr;
            taObj.setText(taText.toString());
            //设置对话框上 Label 对象的文本内容
            diaObj.show();                        //显示对话框
        }
    }
    if (butObj.getLabel() == "OK") {             //OK 按钮的处理
        //单击 OK 按钮后, 信息提示对话框消失, 主界面显示
        diaObj.hide();                          //隐藏对话框
        frObj.show();                           //显示主界面
    }
}

```

【代码解析】

当单击 Exit 按钮时, 通过调用 exit() 方法退出项目。单击 Find 按钮时, 通过处理字符串工具类的 strFind() 方法获取所匹配的次数, 然后在相应的地方显示出来。当单击 Replace 按钮时, 通过处理字符串工具类的 strReplace() 方法获取所匹配的次数, 然后在相应的地方显示出来。

21.2.3 字符串处理的类

为了方便其他类的调用, 在该项目中专门创建了一个字符串操作的类, 在该类中主要实现查找和替换的方法, 该类的具体内容如代码 21.3 所示, 其类图如图 21.10 所示。

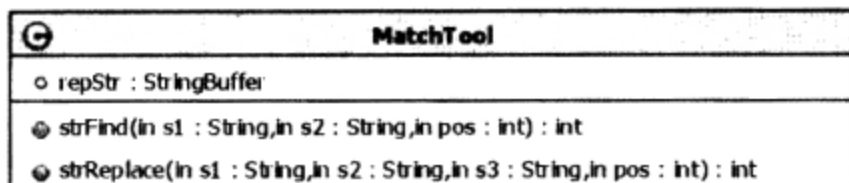


图 21.10 字符串处理的类图

代码 21.3 字符串处理的类: MatchTool.java

```

class MatchTool {
    StringBuffer repStr; //创建一个 StringBuffer 变量
    //用于创建实现字符串查找, 返回匹配的次数方法
    public int strFind(String s1, String s2, int pos) {
        //创建表示主串和模式串中当前字符串位置的变量 i、j, 以及表示匹配次数的变量 k
        int i, j, k = 0;
        //为 i 赋值, 表示直接从主字符串头开始找, i 决定着开始查找的位置
        i = 0;
        //为 j 赋值, 表示模式串中当前的位置
        j = 0;
        while (i < s1.length() && j < s2.length()) { //实现查找功能
            if (s1.charAt(i) == s2.charAt(j)) {
                ++i;
                ++j;
                if (j == s2.length()) {
                    // j=s2.length() 表示字符串匹配成功, 匹配成功次数加 1
                    k = k + 1;
                    //将指示主串和模式中当前字符的变量 i 和 j 进行回退
                    i = i - j + 1;
                    j = 0;
                }
            } else {
                i = i - j + 1;
                j = 0;
            }
        }
        i++;
    }
    return k; //表示匹配成功的次数
}

//strReplace() 方法用于实现字符串替换操作, 返回替换的次数
public int strReplace(String s1, String s2, String s3, int pos) {
    //创建表示主串和模式串中当前字符串位置的变量 i、j, 以及表示匹配次数的变量 k
    int i, j, k = 0;
    //为 i 赋值, 表示直接从主字符串头开始找, i 决定着开始查找的位置
    i = 0;
    //为 j 赋值, 表示模式串中当前的位置
    j = 0;
    repStr = new StringBuffer(s1); //将 s1 转化成 StringBuffer 型进行操作
    //实现替换功能
    while (i < repStr.length() && j < s2.length()) {
  
```

```

        if (repStr.charAt(i) == s2.charAt(j)) {
            ++i;
            ++j;
            if (j == s2.length()) {
                k = k + 1;
                repStr.replace(i - j, i, s3);
                j = 0; //将 j 进行重新赋值开始新的比较
            }
        }
        else {
            i = i - j + 1;
            j = 0;
        }
    }
    return k; //返回替换成功的次数
}
}

```

【代码解析】

在上述代码中主要存在两个方法，分别为实现字符串查找，返回匹配次数的 `strFind()` 方法；实现字符串替换操作，返回匹配次数的 `strReplace()` 方法。

21.3 查找和替换项目——利用 Swing 组件

21.2 节通过 AWT 组件实现查找和替换模块的主界面，本章将通过 Swing 组件重写该项目，其项目结构与 21.2 节相同，主要是 `FindReplace` 这个类的内容不同。

21.3.1 设计项目的界面——查找和替换输入界面

`FindReplace` 为查找和替换项目界面的类，在该类中不仅通过 Swing 组件实现主界面，还会实现显示“查找”和“替换”结果的对话框。具体内容如代码 21.4 所示，该类的类图如图 21.11 所示。

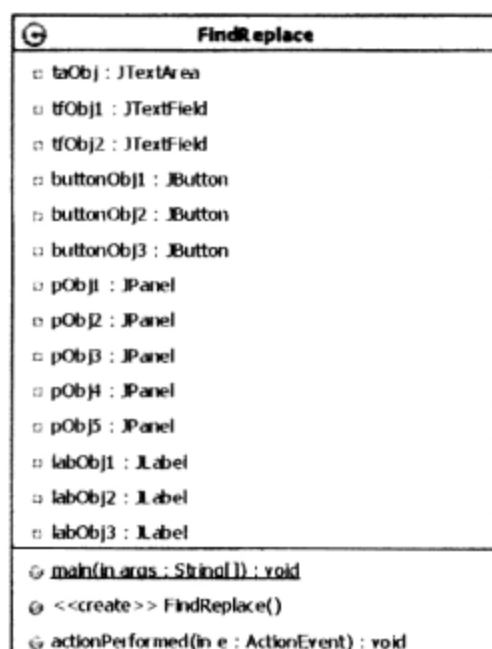


图 21.11 主界面的类图

代码 21.4 主界面: FindReplace.java

```

public class FindReplace extends JFrame implements ActionListener {
    private JTextArea taObj;           //创建文本域对象
    private JTextField tfObj1, tfObj2; //创建文本框对象
    //创建按钮对象
    private JButton buttonObj1;
    private JButton buttonObj2;
    private JButton buttonObj3;
    //创建容器对象
    private JPanel pObj1, pObj2, pObj3, pObj4, pObj5;
    //创建显示信息面板
    private JLabel labObj1, labObj2, labObj3;
    public FindReplace() {             //构造函数
        super("文本替换与查找");       //设置标题
        taObj = new JTextArea();       //为文本域赋值

        //创建用于输入待查找或替换字符串的文本框
        tfObj1 = new JTextField();
        //创建用于输入替换字符串的文本框
        tfObj2 = new JTextField();

        //初始化 3 个按钮对象, 分别用来实现查找、替换和退出操作
        buttonObj1 = new JButton("Find");
        buttonObj2 = new JButton("Replace");
        buttonObj3 = new JButton("Exit");

        //初始化 3 个 Label 对象用于显示有关的提示信息
        labObj1 = new JLabel("文本区域");
        labObj2 = new JLabel("待查找或替换前的字符串");
        labObj3 = new JLabel("替换后的字符串");

        //通过 5 个 Panel 对象用于控制主界面上各组件的位置和大小
        //为 5 个 Panel 对象赋值
        pObj1 = new JPanel();
        pObj2 = new JPanel();
        pObj3 = new JPanel();
        pObj4 = new JPanel();
        pObj5 = new JPanel();
        //控制文件区域和相关提示信息的相对位置
        pObj1.setLayout(new BorderLayout());
        pObj1.add("North", labObj1);
        pObj1.add("Center", taObj);
        //控制第一个文本框和相关提示信息的相对位置
        pObj2.setLayout(new BorderLayout());/**/
        pObj2.add("North", labObj2);
        pObj2.add("Center", tfObj1);
        //控制第二个文本框和相关提示信息的相对位置
        pObj3.setLayout(new BorderLayout());/**/
        pObj3.add("North", labObj3);
        pObj3.add("Center", tfObj2);
        //控制 Find 按钮和 Replace 按钮的相对位置
        pObj4.setLayout(new FlowLayout(FlowLayout.CENTER, 20, 20));/**/
        pObj4.add(buttonObj1);
        pObj4.add(buttonObj2);
        pObj4.add(buttonObj3);
        //控制 pObj2, pObj3 和 pObj4 的相对位置

```

```

    pObj5.setLayout(new GridLayout(3, 1));/**/
    pObj5.add(pObj2);
    pObj5.add(pObj3);
    pObj5.add(pObj4);
    //最后添加到主窗口上
    setLayout(new GridLayout(1, 2));
    Container container = getContentPane();           //获取容器对象
    container.setVisible(true);
    container.add(pObj1);
    container.add(pObj5);
    //为 Find 按钮、Replace 按钮和主窗口添加事件监听器
    buttonObj1.addActionListener(this);
    buttonObj2.addActionListener(this);
    buttonObj3.addActionListener(this);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);    //创建退出方法
    setSize(560, 260);                                //设置窗口大小
    setVisible(true);                                  //显示窗口
}
public static void main(String args[]) {              //主方法
    FindReplace my = new FindReplace();                //创建 FindReplace 对象
}
//处理鼠标单击的事件方法
public void actionPerformed(ActionEvent e) {
...                                                    //省略部分代码
}
}

```

【代码解析】

上述代码实现了查找和替换项目界面，该用户界面涉及的具体容器、对象和布局如图 21.12 所示。

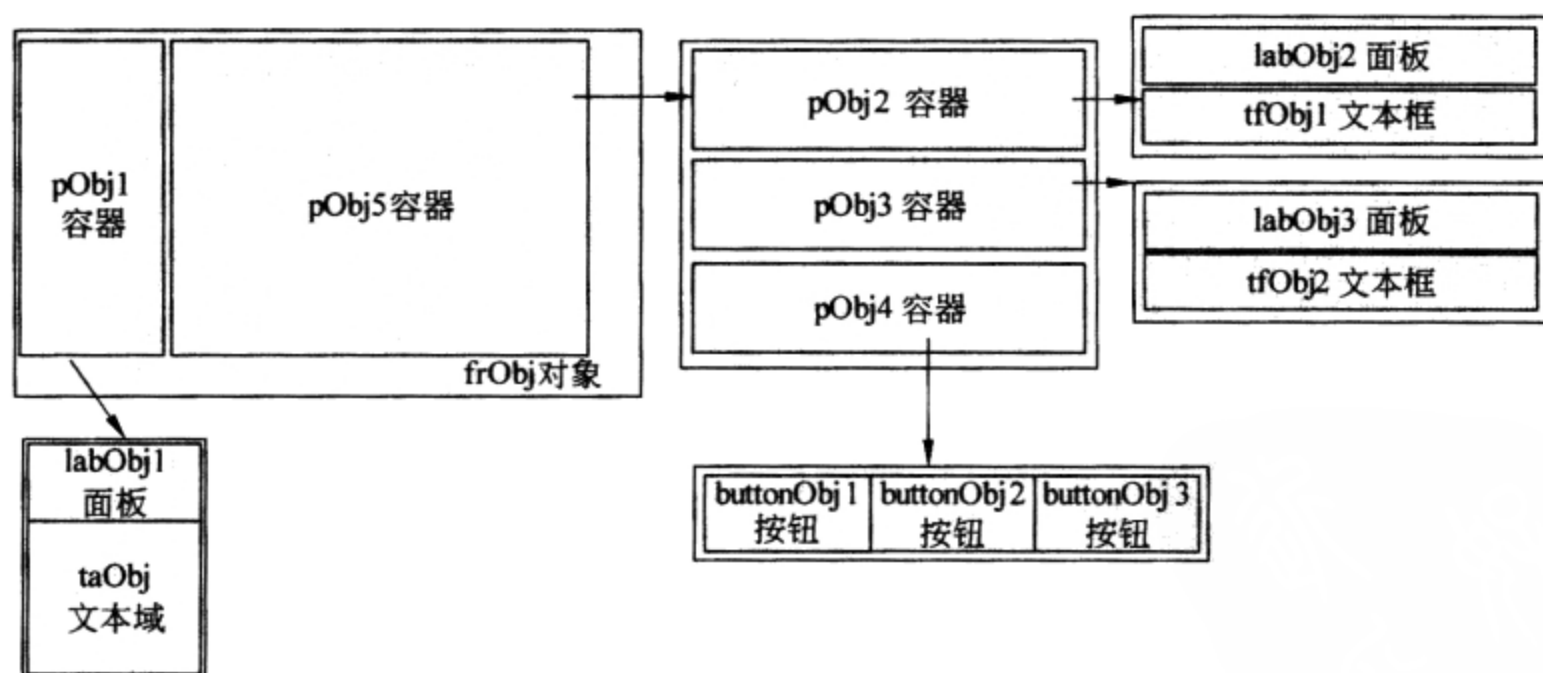


图 21.12 布局

21.3.2 各种按钮的事件处理

mouseClicked()方法主要用来实现处理鼠标单击事件，即分别实现单击 Find 按钮、Replace 按钮和 Exit 按钮后所要调用的代码，具体内容如代码 21.5 所示。

代码 21.5 处理按钮事件: actionPerformed

```

public void actionPerformed(ActionEvent e) {
    JButton butObj = (JButton) (e.getSource()); //获取事件源对象
    //处理单击 Find 按钮、Replace 按钮和 Exit 按钮的事件
    if (butObj.getLabel() == "Exit") { //如果单击 Exit 按钮
        System.exit(0); //退出系统
    }
    //如果单击 Find 按钮和 Replace 按钮
    if (butObj.getLabel() == "Find" || butObj.getLabel() == "Replace") {
        //初始化一些信息
        String strObj1 = taObj.getText(); //获取文本区域内容
        String strObj2 = tfObj1.getText(); //获取第一个文本框内容
        //变量 matchNum 代表字符串匹配的个数, 初始值为 0
        int matchNum = 0;
        int cursorPos = taObj.getCaretPosition(); //初始化光标当前的位置
        MatchTool classObj = new MatchTool(); //创建 matchFunod 类的对象
        if (butObj.getLabel() == "Find") { //单击 Find 按钮
            //通过调用 matchFun 类的 strFind() 方法, 计算出字符串匹配的个数
            matchNum = classObj.strFind(strObj1, strObj2, cursorPos);
            //设置对话框上 Label 对象的文本内容
            JOptionPane.showMessageDialog(null, "共找到" + matchNum + "处",
                "提示", JOptionPane.INFORMATION_MESSAGE);
        }
        if (butObj.getLabel() == "Replace") { //单击 Replace 按钮
            String strObj3 = tfObj2.getText(); //获取第二个文本框中的字符串
            //通过调用 matchFun 类中的 strReplace 按钮计算字符串替换次数
            matchNum = classObj.strReplace(strObj1, strObj2, strObj3,
                cursorPos);
            //重新设置文本域的内容
            StringBuffer taText = classObj.repStr;
            taObj.setText(taText.toString());
            //设置对话框上 Label 对象的文本内容
            JOptionPane.showMessageDialog(null, "共替换到" + matchNum + "处",
                "提示",
                JOptionPane.INFORMATION_MESSAGE);
        }
    }
}

```

【代码解析】

在上述代码中主要存在两个方法, 分别为实现字符串查找, 返回匹配次数的 `strFind()` 方法; 实现字符串替换操作, 返回匹配次数的 `strReplace()` 方法。

21.4 小 结

本章主要通过对字符串的操作来模拟 Windows 系统中的查找和替换功能, 为了让用户更容易使用这些功能, 还设计了非常友好的图形用户界面, 即通过 AWT 组件实现的界面和通过 Swing 组件实现的界面。

第 5 篇 Applet 程序开发

- ▶▶ 第 22 章 图像轮显动画项目（显示图像+多线程）
- ▶▶ 第 23 章 Applet 事件监听项目（事件处理机制）
- ▶▶ 第 24 章 动画播放项目（音频操作+多线程）

第 22 章 图像轮显动画项目 (显示图像+多线程)

在 Java 语言中不仅可以在 Application 项目中操作图像，在 Applet 程序中同样可以显示图像。本章不仅通过“图像轮显动画”项目详细讲解如何在 Applet 程序中显示图像，还会讲解 Applet 的基础知识。

本章的学习目标如下：

- ❑ 掌握图像轮显动画项目；
- ❑ 理解在 Applet 程序中如何显示图像；
- ❑ 了解 Applet 程序的基础知识。

22.1 图像轮显动画原理

“图像轮显动画”项目用来通过不断地显示不同图像来模拟一种图像动画。在具体运行程序时，会在 Applet 程序的界面上循环显示 0~9 的图像。

22.1.1 项目结构框架分析

对于图像轮显动画项目，由于其是 Applet 程序，所以主类需要继承 Applet 类。图像轮显动画项目目录如图 22.1 所示，只有一个实现动画功能的 PictureAnimation 类和承载该 Applet 程序的网页。

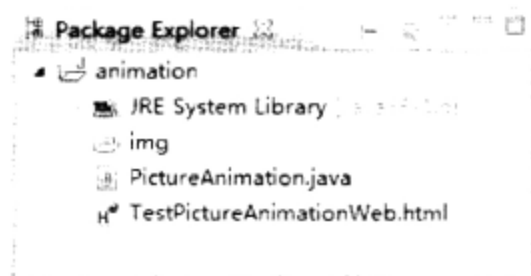


图 22.1 项目目录

22.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括图像轮显动画的初始化、图像轮显功能和图像重画功能。

1. 初始化界面

当运行“图像轮显动画”项目中的 `PictureAnimation` 类后，会出现如图 22.2 所示的初始界面。



图 22.2 初始界面

2. 动画的运行

当出现初始界面后，随着时间的推移，界面中会循环显示相应的数值图像，同时也会循环显示相应的信息，具体过程如图 22.3 所示。

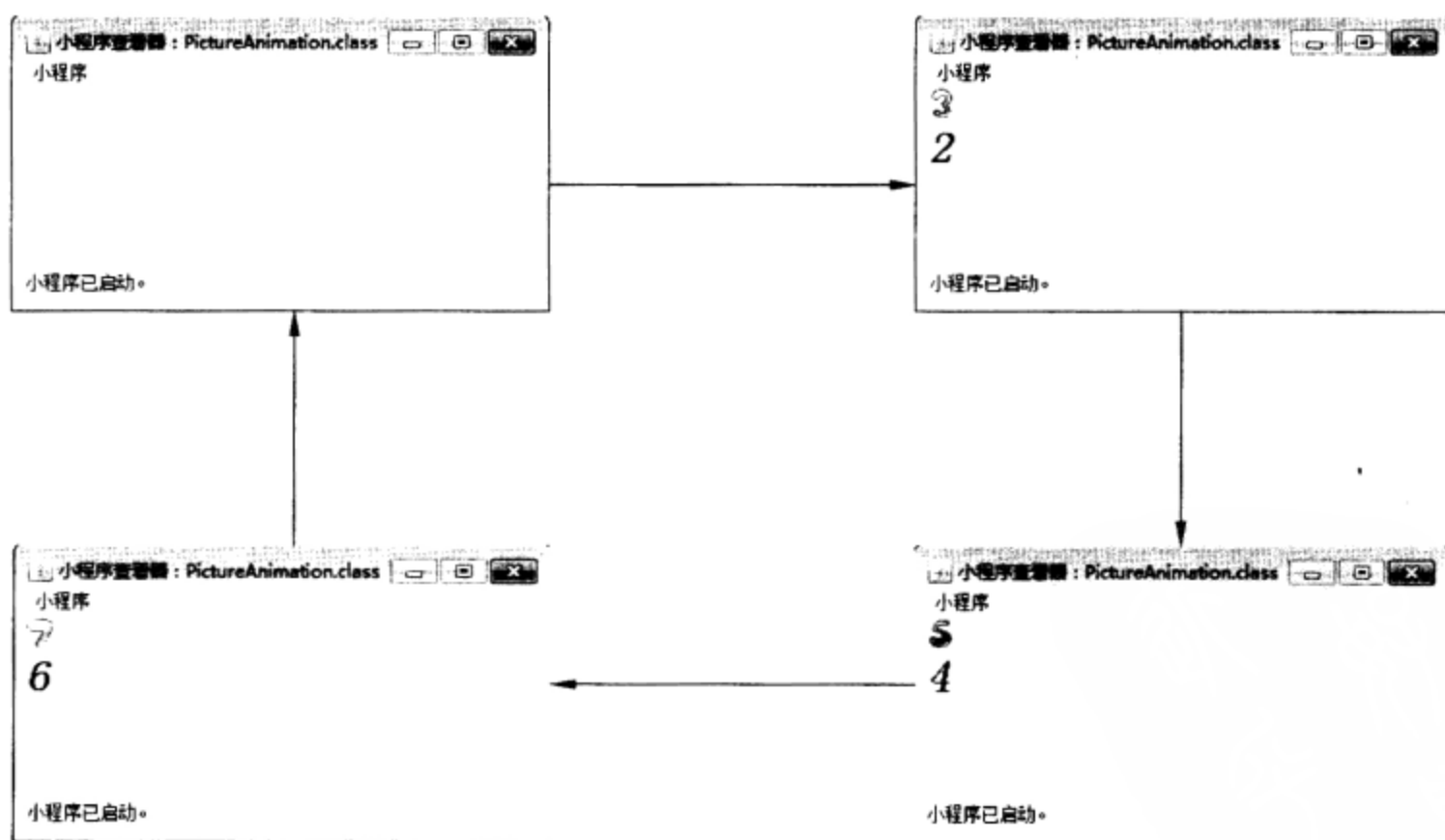


图 22.3 动画的运行过程

3. 重绘功能

将窗口最小化后再恢复正常化显示的过程，其实就是重绘的过程。如果考虑重绘功能，恢复正常化显示就会出现正常的窗口；否则就会出现空白窗口，具体过程如图 22.4 所示。

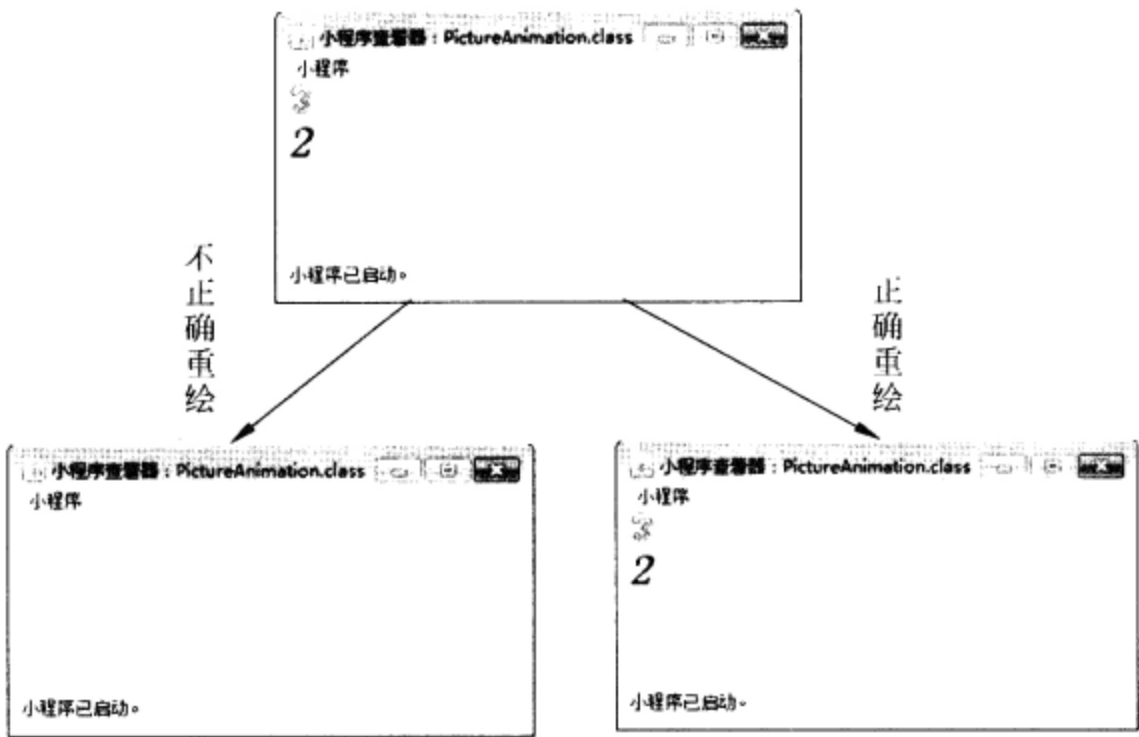


图 22.4 重绘的过程

22.2 图像轮显动画项目

图像轮显动画项目具体程序架构如图 22.5 所示，它包含一个动画的类和一个承载该程序的网页 `PictureAnimation.java` 和 `TestPictureAnimationWeb.html`。该项目通过定时循环显示图片的功能模拟动画效果，主要利用了 Java 语言中的线程机制和显示图像技术。动画功能的实现分为以下两个阶段来实现。

- (1) 实现定时功能，让一个变量的取值在每隔 100 毫秒于 0~9 之间循环变化显示。
- (2) 显示图像动画功能，改变变量值后，只需要显示数组中相对应的图形，就可以产生动画效果。

`PictureAnimation.java` 类为图像动画，该类除了重写了 `Applet` 类的 `init()`和 `paint()`方法外，还实现了线程接口 `Runnable` 的 `run()`方法，该类的具体内容如代码 22.1 所示，该类的 UML 如图 22.6 所示。

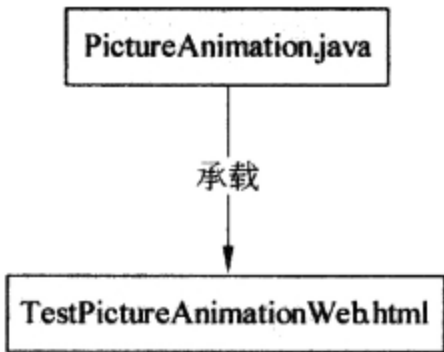


图 22.5 程序关系图

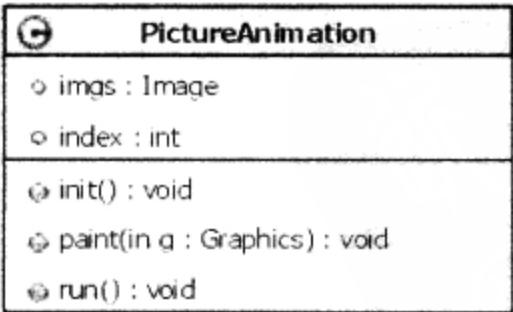


图 22.6 动画类图

代码 22.1 显示动画类: `PictureAnimation.java`

```
public class PictureAnimation extends Applet implements Runnable {
    //创建成员变量
    Image[] imgs = new Image[10];
    int index = 0;
    //创建图像数组
    //创建循环变量
```

```

//初始化方法
public void init() {
    try {
        //为图像数组赋值
        for (int i = 0; i < 10; i++) {
            imgs[i] = getImage(new URL(getCodeBase(), "img\\" + i + ".gif"));
        }
        new Thread(this).start(); //启动线程
    } catch (Exception e) {
        e.printStackTrace();
    }
}

//重画的方法
public void paint(Graphics g) { //重画过程
    g.drawImage(imgs[index], 0, 0, this); //显示图像
    //设置字体
    g.setFont(new Font("隶书", Font.ITALIC | Font.BOLD, 30));
    g.drawString("" + index, 0, 50); //显示相应信息
}

public void run() { //实现 run() 方法
    while (true) { //线程休眠 100 毫秒
        try {
            Thread.sleep(100);
        } catch (Exception e) {
        }
        repaint(); //调用 repaint() 方法
        index = (index + 1) % 10; //在数字 0~9 间循环
    }
}
}

```

【代码解析】

- 在上述代码中，先通过 `start()` 方法启动一个线程，执行 `run()` 方法里的代码，即每隔 100 毫秒调用 `repaint()` 方法及通过代码 “`index = (index + 1) % 10`”，让成员变量 `index` 的取值在 0~9 之间循环变化。根据图形用户界面知识可以知道，`repaint()` 方法会调用 `paint()` 方法，该方法会显示变量 `index` 变量值所对应的图像并输出相应的信息。
- 在上述代码中，对表示图像数组的变量 `imgs` 的赋值是在初始化方法 `init()` 中实现的。由于上述类是 `Applet` 程序，所以在具体加载图像时不能实现绝对地址，而应该使用相对地址，因此创建了表示地址的 `URL` 对象。

接着再创建一个承载该 `Applet` 程序的网页，该网页的具体内容如代码 22.2 所示。

代码 22.2 承载网页：TestPictureAnimationWeb.html

```

<body>
    <Applet code="PictureAnimation.class" width=400 height=50>
    </Applet>
</body>

```

如果用 IE 浏览器打开该页面，同样出现如图 22.7 所示的页面。

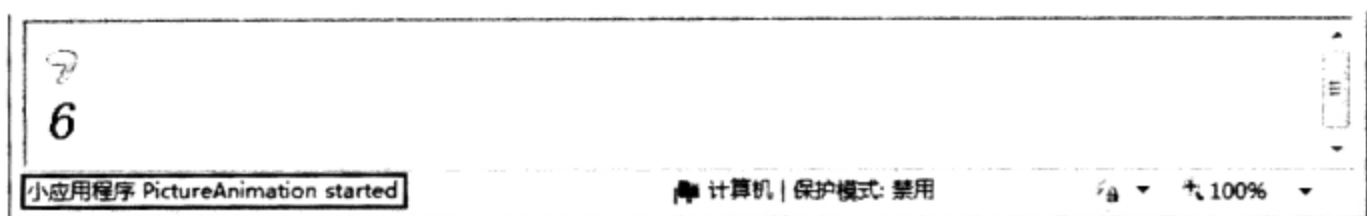


图 22.7 测试页面

22.3 知识点扩展——Applet 程序的基础知识

在 Java 语言中存在两种基本程序，Application 和 Applet，Applet 程序就是在浏览器上运行的小型 Java 程序。当用户访问包含 Applet 的网页时，该 Applet 程序会被下载到本地计算机里，然后才会执行。

22.3.1 Applet 程序的执行过程

Applet 程序被用户称为是比较平易近人的程序，之所以这样，是因为该类程序是通过浏览器来显示的，即利用网页作为载体。

所谓 Applet 程序就是在浏览器中执行的 Java 程序，是按照下面的过程执行的：

- ☐ 浏览器载入要访问的 HTML 文件的 URL 地址。
- ☐ 浏览器载入 HTML 文件。
- ☐ 浏览器载入 Applet 的类字节代码。
- ☐ 启动 Java 虚拟机执行 Applet。

22.3.2 Applet 程序的执行环境

Applet 与 Application 的区别主要在于执行方式不同，Application 程序主要是从其中的 main()方法开始执行的，而 Applet 程序是无法单独运行的，必须以一个网页作为载体。

在 Applet 程序的运行环境中，当用户利用浏览器显示 Applet 程序时，如果显示不出正确的运行结果，是因为该浏览器没有内嵌 Java 解释器。为了解决该问题，可以在安装 JDK 的具体过程中，选择“浏览器的默认 Java”选项来支持浏览器，具体界面如图 22.8 所示。



图 22.8 选择插件

当正确安装了 JDK 后，并且也选择了“浏览器的默认 Java”选项，但是在浏览器中仍然不能正确显示 Applet 程序时，这时就需要配置浏览器。通过选择“工具”|“Internet 选项”命令，打开如图 22.9 所示的 Internet 选项对话框。在该对话框的“高级”选项卡中选择“将 JRE 1.6.0_03 用于<applet>（需要重新启动）”选项即可，如图 22.10 所示。

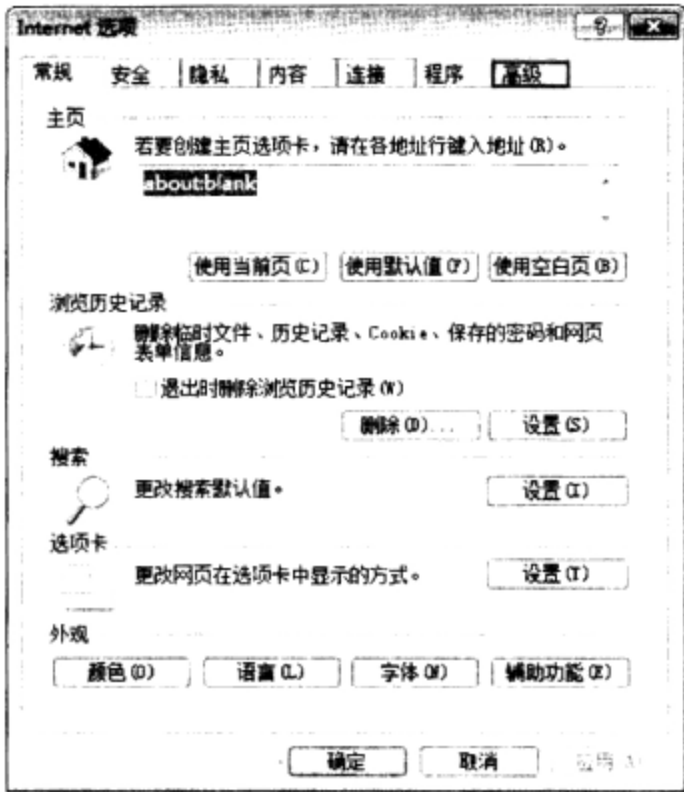


图 22.9 Internet 选项对话框

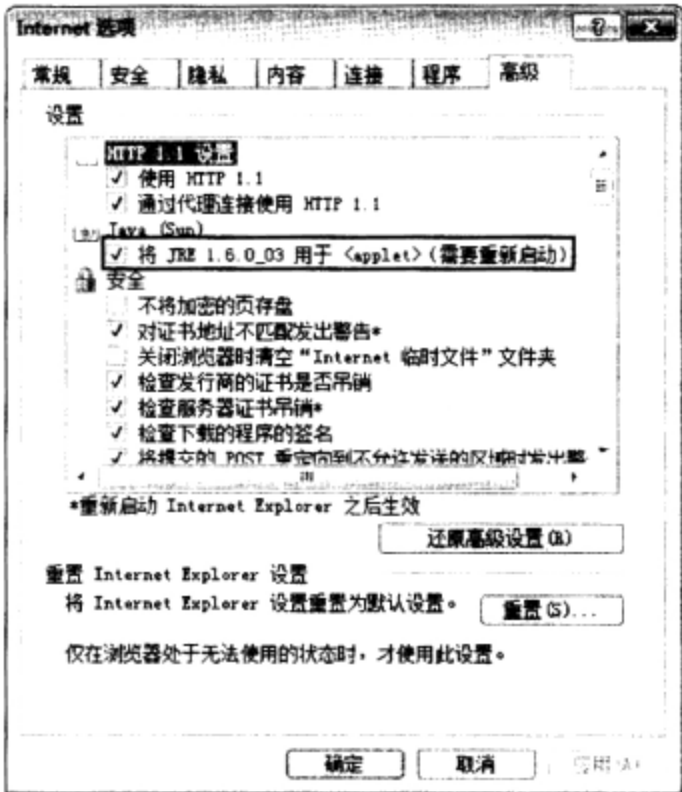


图 22.10 “高级”选项卡

22.3.3 Applet 程序的输出

Applet 类继承了 Panel 类，因此是一个容器，具体的继承关系如图 22.11 所示。在具体编写一个 Applet 程序时，最好继承类 java.applet.Applet。

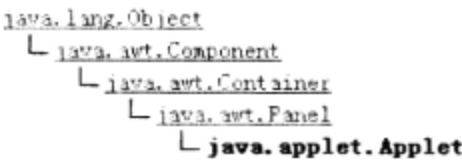



图 22.11 类的关系

在图形用户界面中如果使用 System.out.println()语句是不能打印字符文本的，只有使用 drawString()语句才能在图形窗口中打印出字符文本。对于 Applet 程序，如果利用 appletviewer 程序执行，System.out.println()语句的输出会显示在启动该程序的控制台窗口中。其实 IE 浏览器也提供了专门用于显示 System.out.println()语句的输出小窗口，只要右击桌面任务栏的右下角 Java 小图标，就会打开如图 22.12 所示的菜单。在该菜单中选择“打开控制面板”选项就可以打开如图 22.13 所示的控制台窗口。

22.3.4 Applet 程序的标记

如果要把 Applet 程序应用于网页中，需要使用 HTML 语言中的<applet>标记。<applet>标记被用来从网页中启动一个 Applet 程序，该标记的具体语法如下：

```
<applet archive="" code="" width="" height="" alt="" name="" align=""
vspace="" hspace="">
    <param name="" value=""/>
    <param name="" value=""/>
</applet>
```

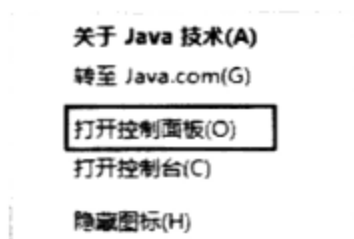



图 22.12 右键菜单



图 22.13 Java 控制台

上述各种属性的含义如下。

- ☐ archive 属性：指定 Applet 程序执行前预载入的类字节码，或其他资源压缩文件（后缀名为 jar 的文件）。
- ☐ code 属性：指定所要执行的 Applet 程序名称，对于 Applet 名称包含不包含.class 后缀都可以，至于 Applet 程序的路径，则是由 HTML 文件的地址和 codebase 属性的值来决定。
- ☐ alt 属性：用于指定 Applet 不能正常执行时显示的替换文本。
- ☐ name 属性：用于指定 Applet 的实例名称。

 注意：只有设置了该属性，同一网页上的所有 Applet 程序才能互相访问并通信。

- ☐ align 属性：用于指定 Applet 显示的对齐方法。
- ☐ vspace、hspace 属性：用于指定与边框之间的垂直和水平距离。
- ☐ width、height 属性：用于指定 Applet 程序执行时显示的长度和高度。

22.3.5 参数的传递

对于 HTML 语言中的<applet>标记，比较复杂的属性就是参数<param>标记。如果 Applet 程序需要根据参数来初始化自己，可以利用<param>标记来实现，从而实现从网页传递参数到 Applet 程序的功能。Applet 程序如果要获取参数，需要通过 Applet 类的 getParameter()方法，该方法的语法格式如下：

```
String getParameter(String name)
```

参数为标记<param>设置的参数名称，返回指定名称参数的值。

下面通过一个具体的实例讲解如何实现参数的传递，具体步骤如下。

(1) 创建一个实现滚动字幕的类 MoveString，具体内容如代码 22.3 所示。

代码 22.3 实现字幕的滚动: MoveString.java

```
public class MoveString extends JApplet implements Runnable {
    //创建静态成员
    String str;                                //滚动字符变量
    int time;                                  //事件间隔变量
    private Thread thread;                     //线程变量
    public void init() {                       //初始化方法
        setBackground(Color.PINK);            //设置背景颜色
        str = getParameter("message");        //获取参数为变量 str 赋值
        String timeArg = getParameter("time"); //获取参数 time 值
        time = Integer.parseInt(timeArg);      //为变量 time 赋值
        thread = new Thread(this);            //创建线程对象
    }
    public void start() {                      //实现 start() 方法
        thread.start();                        //启动线程
    }
    public void run() {                        //实现 run() 方法
        int x = 0;                             //字幕的 x 坐标
        Graphics g = getGraphics();            //获取 Graphics 对象
        while (true) {
            try {
                Thread.sleep(time);            //线程休眠
            } catch (Exception e) {
                e.printStackTrace();
            }
            g.clearRect(0, 0, getWidth(), getHeight()); //清屏
            g.drawString(str, x, 30);          //绘制字幕
            x += 2;                             //实现字幕的移动
            if (x >= getWidth())               //字幕出界
                x = 0;
        }
    }
}
```

(2) 创建一个 MoveString 类的载体网页 TestMoveStringWeb.html，具体内容如代码 22.4 所示。

代码 22.4 网页载体: TestMoveStringWeb.html

```
<body>
    <!--设置 code 属性-->
    <Applet code="MoveString.class" width=400 height=60>
        <param name="message" value="cjgong">    <!--参数 message-->
        <param name="time" value="100">         <!--参数 time-->
    </Applet>
</body>
```

如果用 IE 浏览器打开该页面，出现如图 22.14 所示的运行过程，即页面里的字符串 cjgong 会随着时间的推移而在页面中移动。

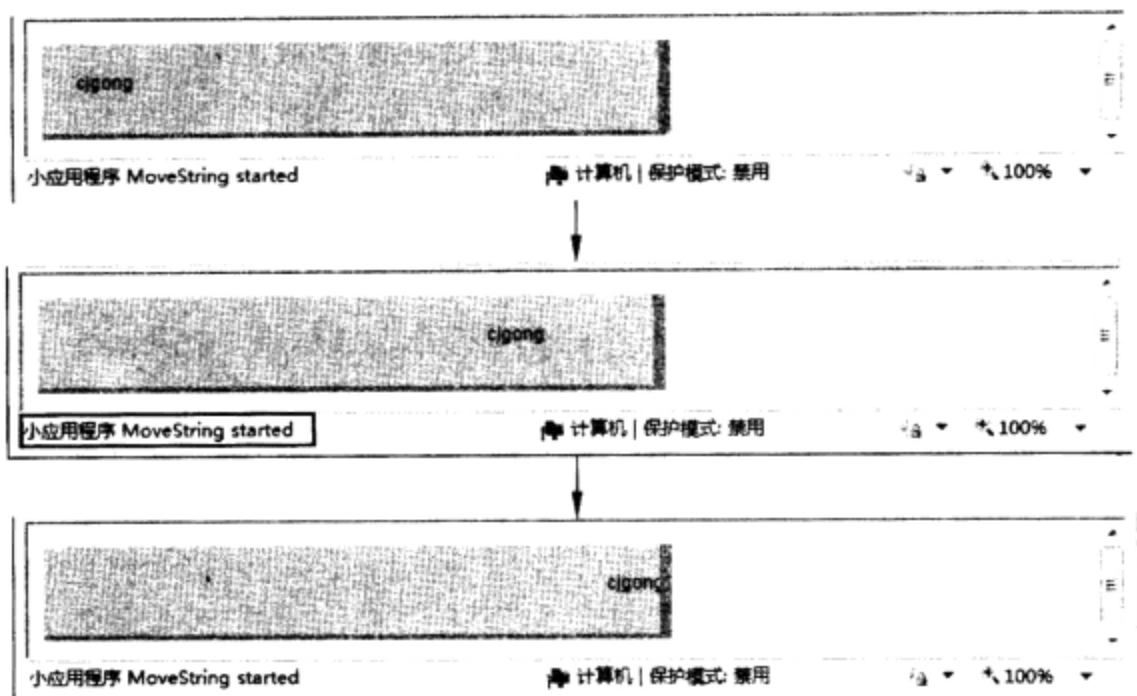


图 22.14 运行结果

22.3.6 Applet 程序的相关方法

Applet 程序与 Application 程序有很大的不同, Application 程序必须有一个 `main()` 方法, 而 Applet 程序必须继承 `java.applet.Applet` 这个类; Application 程序运行的起始点为 `main()` 方法, 而 Applet 程序运行的起始点为 `init()` 方法。

由于 Applet 程序的载体为网页, 所以 Applet 类中存在相应的方法来对应浏览器的具体动作, 例如浏览器放到最大动作、缩到最小动作、跳到其他网页动作、关闭的动作。Applet 类的具体方法如下。

1. `init()` 方法

该方法用来实现 Applet 程序初始化功能, 例如图片的加载、成员变量的初始值、对象实例化等。当浏览器打开网页中的 Applet 程序时, 第一个调用的方法就是该方法。

2. `start()` 方法

该方法为 Applet 程序的开始阶段, 只有该方法运行完后, 整个 Applet 程序才算“活”起来。当 Applet 程序调用完 `init()` 方法后就会调用该方法; 当用户从其他页面返回到包含该 Applet 程序的页面时, 也会调用该方法。在具体编程时, 会放置一些工作量比较重的初始化动作, 例如线程的产生和打开。

注意: 在具体运行 Applet 程序时, 初始化方法 `init()` 只会执行一次, 而开始 `start()` 方法则会被调用多次。

3. `stop()` 方法

该方法为 Applet 程序的暂停阶段, 当离开包含该 Applet 程序的页面时, 就会调用该方法。在具体编程时, 一般会把实现节省系统资源、保证系统运行速度的功能代码放在该方法里。

4. destroy()方法

该方法为 Applet 程序的销毁阶段，当用户正常关闭浏览器时，Applet 程序就会调用该方法。在具体编程时，一般会把实现释放程序所占用的资源的代码放在该方法里。

注意：对于 Applet 程序，除了上述 4 个方法外，还经常调用 paint()方法。该方法不是在 java.applet.Applet 类里定义的方法，而是继承于 java.awt.Container 类里的 paint()方法。

下面通过一个具体的实例来学习 Applet 程序中的各种方法，具体步骤如下。

(1) 创建实现类 Applet 各种方法的 AppletMethod 类，来感受 Applet 程序的各种动作，该类的具体内容如代码 22.5 所示。

代码 22.5 Applet 各种方法：AppletMethod.java

```
public class AppletMethod extends Applet {
    public void init() {                                //初始化方法
        System.out.println("初始化阶段,init");
    }
    public void start() {                                //开始阶段
        showMsg("开始阶段, start");
    }
    public void stop() {                                //暂停阶段
        showMsg("暂停阶段, stop");
    }
    public void destroy() {                              //销毁阶段
        showMsg("销毁阶段, destory");
    }
    public void showMsg(String msg) {                    //输出方法
        System.out.println(msg);
    }
}
```

(2) 再编写一个承载 TestAppletMethod.java 类的页面 TestAppletMethodWeb.html，该页面的具体内容如代码 22.6 所示。

代码 22.6 承载网页：TestAppletMethodWeb.html

```
<html>
  <body>
    <applet code="TestAppletMethod.class" width=200 height=100></applet>
  </body>
</html>
```

(3) 当用浏览器打开 TestAppletMethodWeb.html 页面时，显示的页面和控制台结果分别如图 22.15 和图 22.16 所示。当切换到其他网页时，控制台的输出结果如图 22.17 所示。当又切换到原始页面时，控制台的输出结果如图 22.18 所示。



图 22.15 运行页面

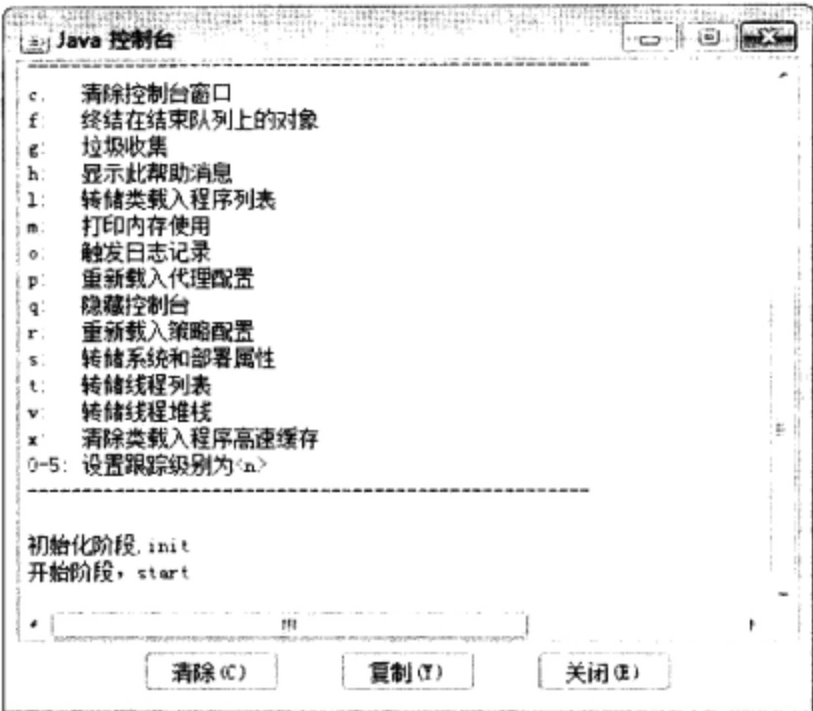


图 22.16 控制台运行结果

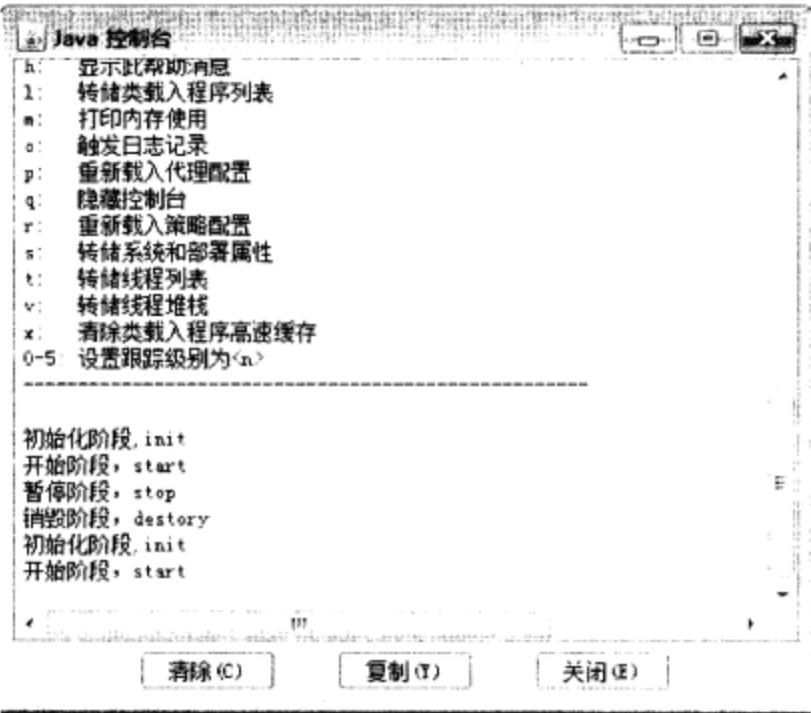


图 22.17 控制台运行结果

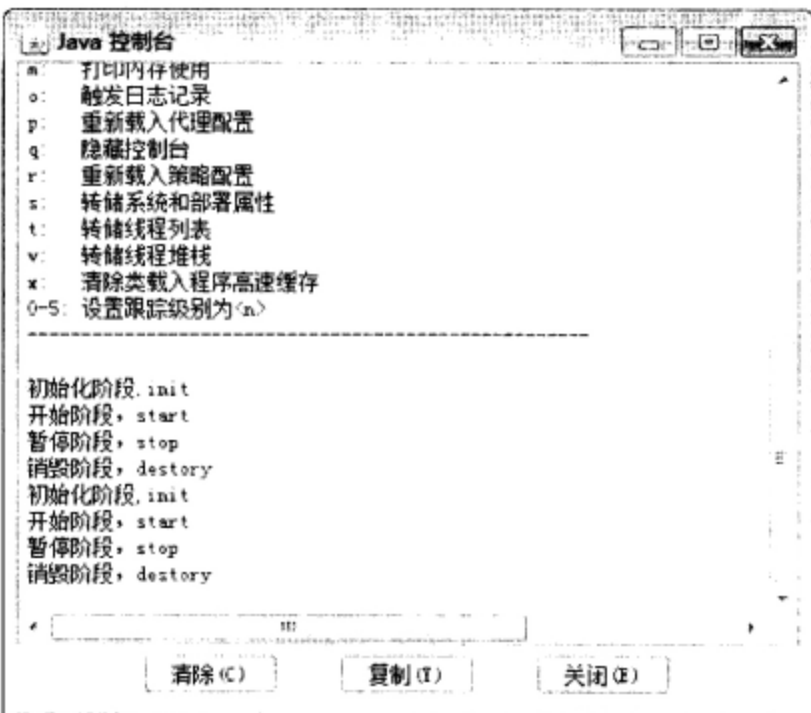


图 22.18 控制台运行结果

(4)当用 appletviewer 运行该程序时,程序的运行界面和控制台输出结果分别如图 22.19 和图 22.20 所示。

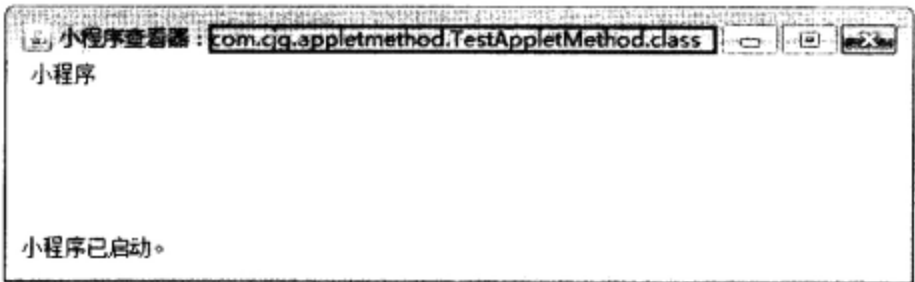


图 22.19 Applet 程序界面



图 22.20 控制台输出信息

为了方便操作“小程序查看器”提供了许多菜单,如图 22.21 所示。当选择 appletviewer 程序菜单的“重新启动”或“重新载入”选项时,Java 输出窗口如图 22.22 所示。

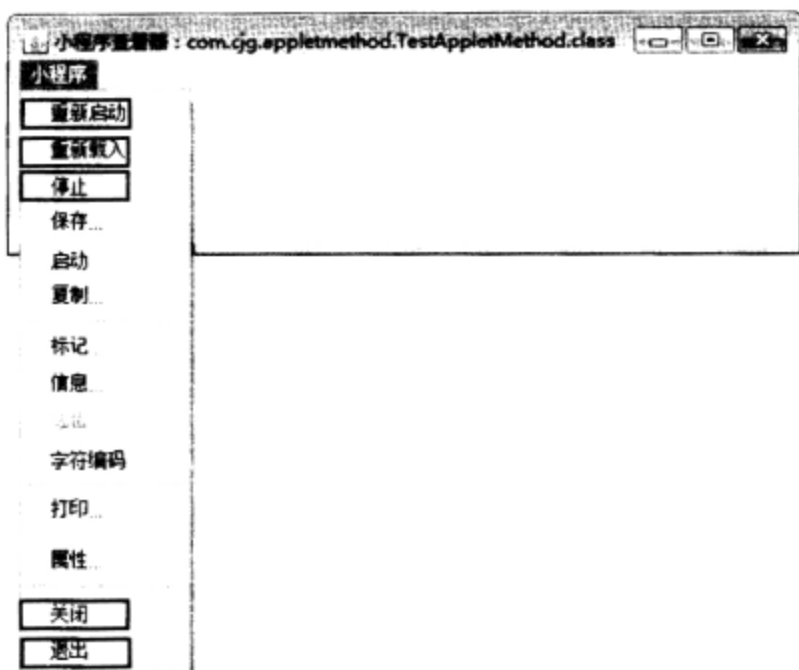


图 22.21 appletviewer 程序窗口

当选择 appletviewer 程序菜单的“停止”选项时，Java 输出窗口如图 22.23 所示。

当选择 appletviewer 程序菜单的“关闭”和“退出”选项时，Java 输出窗口如图 22.24 所示。



图 22.22 输出结果 1



图 22.23 输出结果 2



图 22.24 输出结果 3

22.4 小 结

本章主要通过 Applet 程序中的显示图像技术、图像重绘技术和多线程技术实现在 Applet 程序中图像的轮显。虽然该项目比较小，只包含一个实现图像轮显的 PictureAnimation 类和承载 Applet 的 TestPictureAnimationWeb.html 页面，但是该项目却涉及了 Applet 程序中图像操作的所有知识点。

在本章的最后还详细介绍了 Applet 程序的一些基础知识，通过这些知识的学习，可以更好地掌握 Applet 程序的编写。

第 23 章 Applet 事件监听项目 (事件处理机制)

在 Java 语言中不仅可以在 Application 项目中监听鼠标和键盘事件，在 Applet 程序中同样也可以监听。本章不仅通过 Applet 事件监听项目详细讲解鼠标事件的监听内容，还会讲解键盘事件的监听内容。

本章的学习目标如下：

- ❑ 掌握 Applet 事件监听项目；
- ❑ 理解在 Applet 程序中如何实现鼠标和键盘事件的监听。

23.1 Applet 事件监听原理

“Applet 事件监听”项目通过不停地监听鼠标单击的位置来达到监听功能，即显示鼠标单击点坐标和确定横线和纵线的交叉点。

23.1.1 项目结构框架分析

对于 Applet 事件监听项目，根据面向对象的思想，需要创建一个动画的对象。Applet 事件监听项目目录如图 23.1 所示，各目录的功能如下。

- ❑ 包 com.cjg.mousekey：包含了事件监听的类。
- ❑ 网页 TestMouseKeyWeb.html：测试事件监听项目的网页。

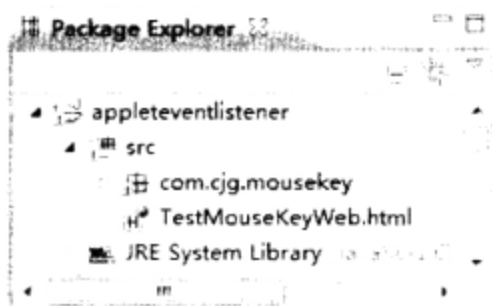


图 23.1 项目目录

23.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括 Applet 事件监听初始化，鼠标单击功能和重绘功能。

1. 初始化界面

当通过 IE 浏览器浏览 TestMouseKeyWeb.html 页面时，会出现如图 23.2 所示的初始界面。

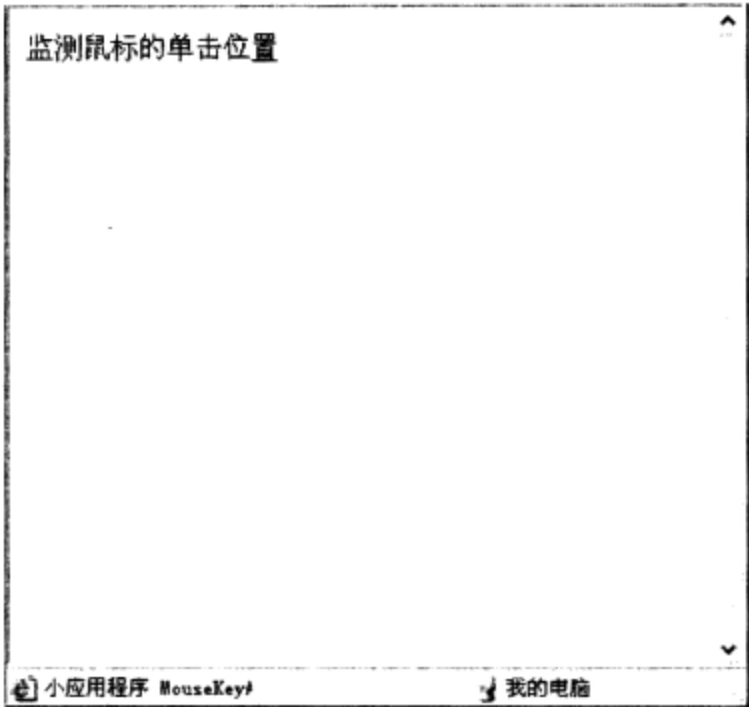


图 23.2 初始界面

2. 鼠标单击功能

当出现初始界面后，如果鼠标在页面的空白处随便一点(77, 117)，不仅会显示出该单击点的坐标(x=77, y=117)，而且还会通过横线和纵线的交叉点来指示单击点。这时如果再单击其他点(173, 163)，同样也会出现上面的情况，具体过程如图 23.3 所示。

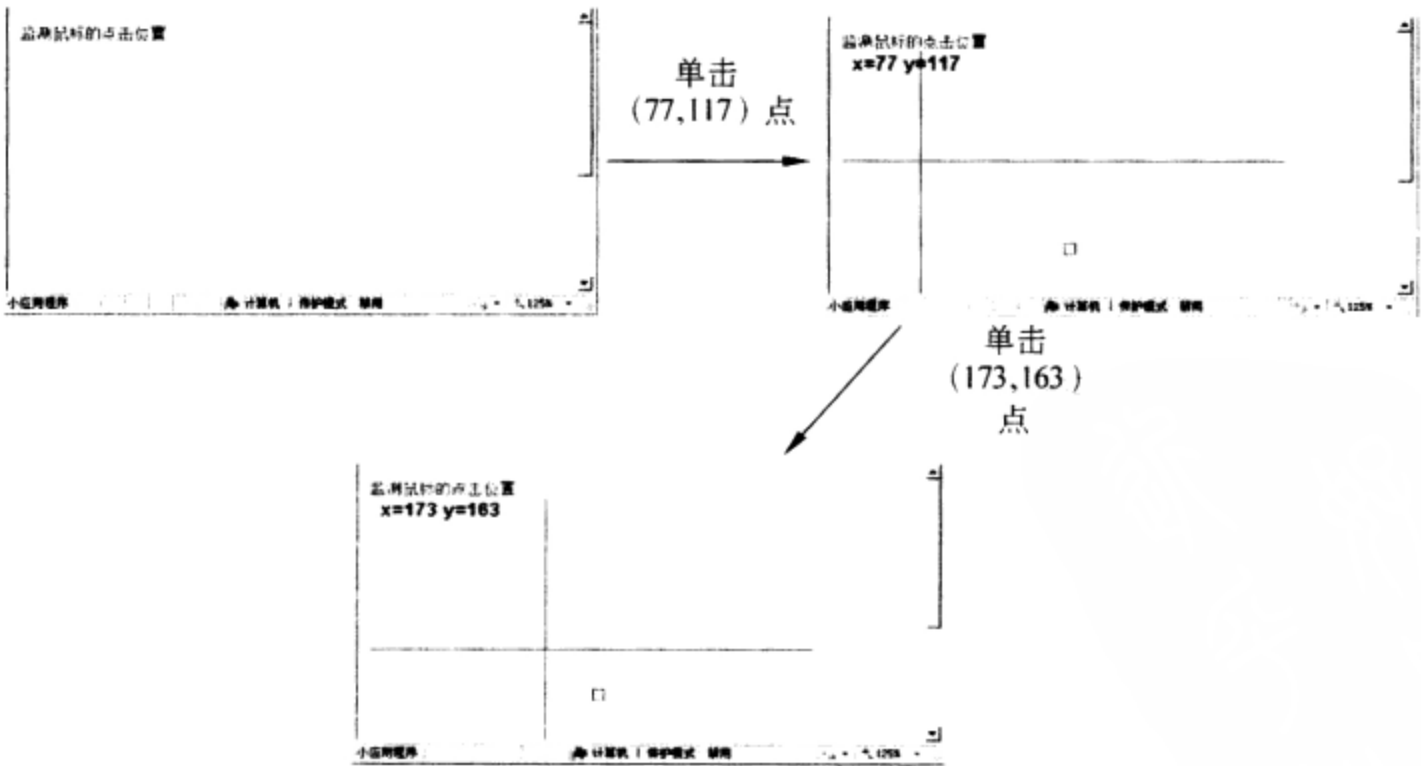


图 23.3 动画的具体过程

3. 重绘功能

将窗口最小化后再恢复正常化显示的过程，其实就是重绘的过程。如果考虑重绘功

能，恢复正常化显示就会出现正常的窗口；否则就会出现空白窗口，具体过程如图 23.4 所示。

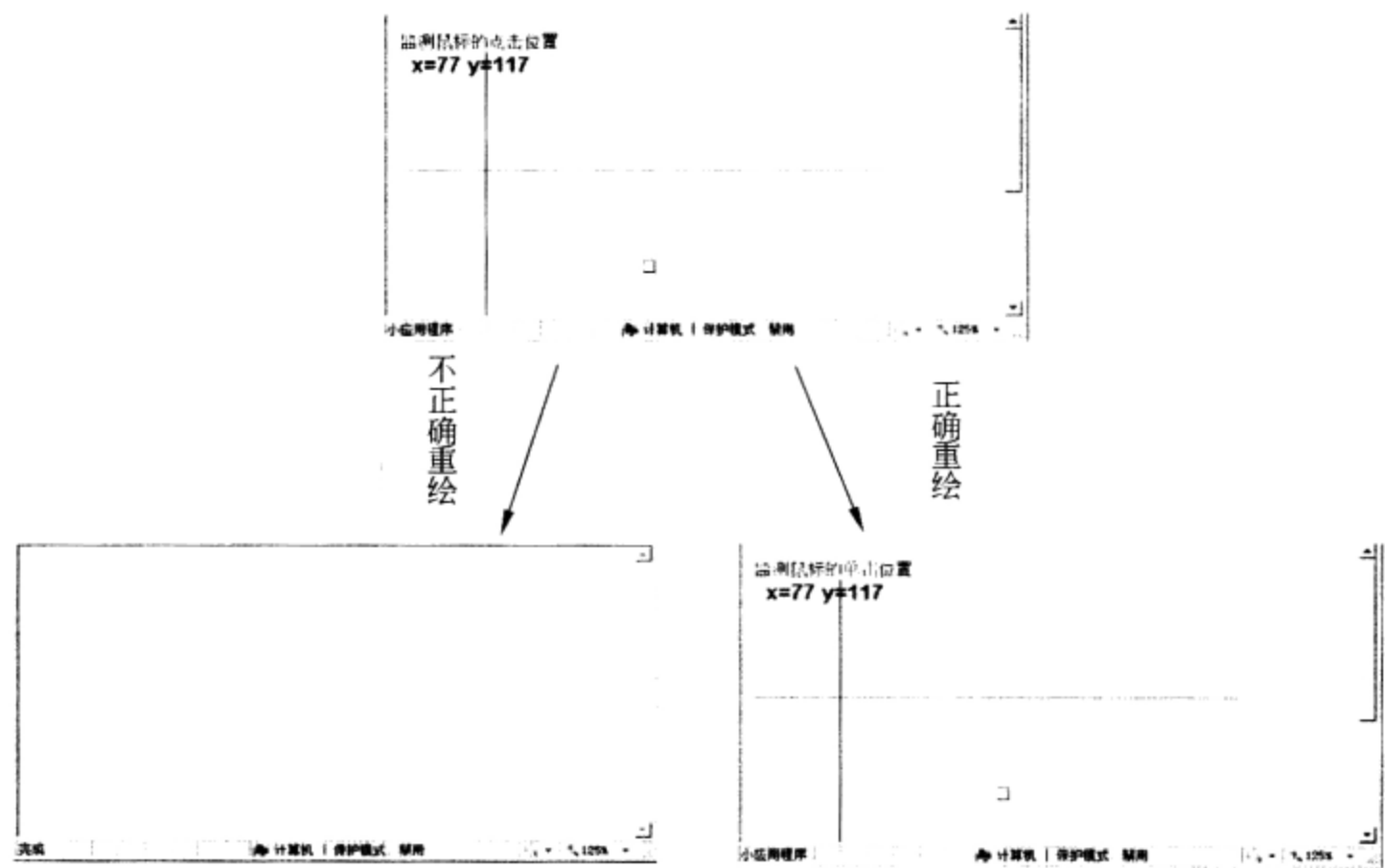


图 23.4 重绘的过程

23.2 Applet 事件监听项目

Applet 事件监听项目具体程序架构如图 23.5 所示，它包含一个实现事件监听的类 `MouseKeyApplet.java` 和一个承载该类的页面 `TestMouseKeyWeb.html`。该项目主要通过 Applet 程序中的事件机制实现“Applet 事件监听项目”。

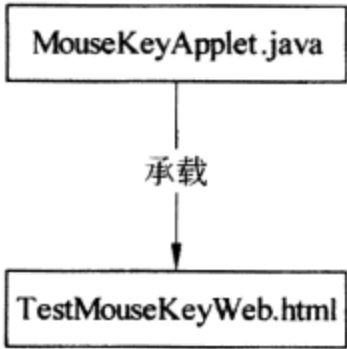


图 23.5 程序关系图

23.2.1 事件监听的类

`MouseKeyApplet.java` 为事件监听类，该类拥有几个属性和初始化的方法 `init()`，在该方法中主要实现添加鼠标和键盘的监听事件。该类的具体内容如代码 23.1 所示，该类的 UML 如图 23.6 所示。

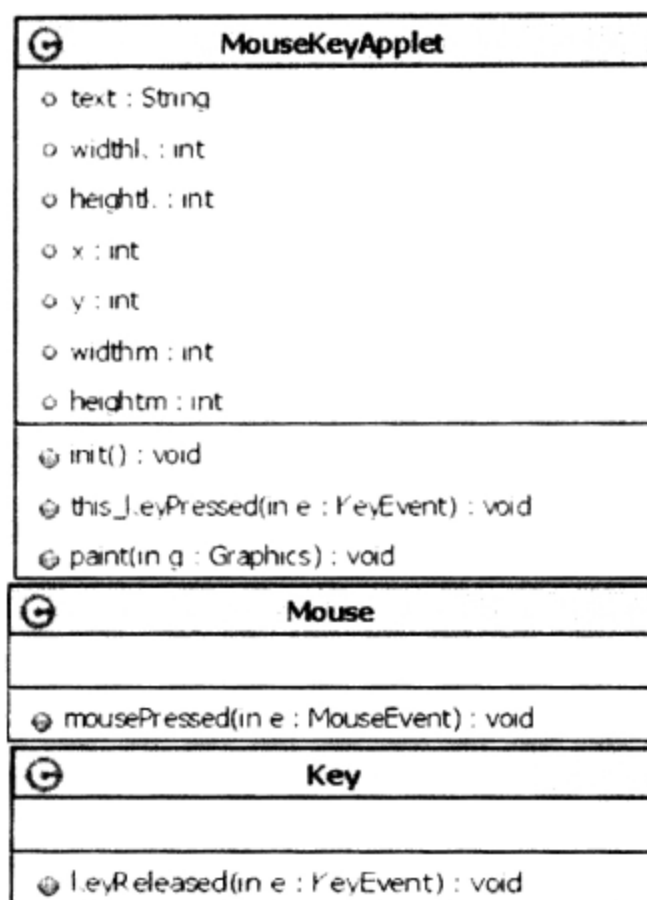


图 23.6 动画类图

代码 23.1 事件监听的类：MouseKeyApplet.java

```

public class MouseKeyApplet extends Applet {
    String text = "";
    int widthk, heightk;
    int x = 0, y = 0;
    int widthm, heightm;
    public void init() {
        addKeyListener(new Key());
        addMouseListener(new Mouse());
    }
    class Key extends KeyAdapter {
        public void keyReleased(KeyEvent e) {
            String s = "";
            text = s += e.getKeyChar();
            //为键盘字符的长宽赋值
            widthk = size().width;
            heightk = size().height;
            repaint();
        }
    }
    class Mouse extends MouseAdapter {
        public void mousePressed(MouseEvent e) {
            x = e.getX();
            y = e.getY();
            //为鼠标绘制区域的长宽赋值
            widthm = size().width;
            heightm = size().height;
            repaint();
        }
    }
    public void paint(Graphics g) {

```

//创建显示键盘字符的变量
//键盘字符的大小
//创建鼠标坐标的变量
//鼠标的绘制区域大小
//初始化方法
//添加键盘监听事件
//添加鼠标监听事件

//创建一个字符串变量
//强制转换成字符型

//重绘方法

//鼠标的监听器
//鼠标按下时的方法
//获取 x 轴坐标
//获取 y 轴坐标

//重绘方法

//绘制 Applet 屏幕

```

        setFont(new Font("Dialog", 1, 20));           //设置字体格式
        //绘制键盘的信息
        g.drawString(text, heightk / 2, widthk / 2);   //画出相应的键盘按钮
        //显示鼠标单击信息
        g.drawString("x=" + x + " y=" + y, 10, 20);     //绘制出鼠标单击的坐标
        g.drawLine(x, 0, x, heightm);                 //绘制 x 轴坐标
        g.drawLine(0, y, widthm, y);                  //绘制 y 轴坐标
    }
}

```

【代码解析】

- 上述代码由于是 Applet 程序,所以继承了 Applet 类。在该类中重写了 init()和 paint()方法, init()方法主要用来实现添加鼠标和键盘事件监听器; paint()方法主要用来实现显示键盘被按下时键的字符和鼠标的坐标。
- 在上述代码中存在两个内部类: 键盘事件的监听器类 KeyAdapterhehe 和鼠标事件的监听器类 MouseAdapter。

23.2.2 承载事件监听的页面

由于 Applet 程序一般都会通过网页来承载,所以在“Applet 事件监听”项目中,实现事件监听功能的类由名为 TestMouseKeyWeb.html 的网页来承载,该页面的具体内容如代码 23.2 所示。

代码 23.2 承载事件监听的页面: TestMouseKeyWeb.html

```

<html>
<head>
    <title>监测鼠标</title>                                <!--标题设置-->
</head>
<body>
    <h>
        监测鼠标的点击位置                                <!--显示相应信息-->
    </h>
    <br>
    <!--显示 Applet 小程序-->
    <applet code="MouseKeyApplet.class" width=350 height=350 ALIGN=middle>
    </applet>
</body>
</html>

```

【代码解析】

在上述代码中,主要通过<applet>标签来显示 Applet 程序。

23.3 知识点扩展——MyEclipse 开发环境对 Applet 程序的支持

在 Java 语言中存在两种基本程序: Application 和 Applet, MyEclipse 开发环境不仅对

Application 项目进行支持，而且还会对 Applet 项目进行支持。

23.3.1 MyEclipse 开发环境对 Applet 项目的支持

如果没有安装 MyEclipse 开发工具，创建和运行 Applet 程序是一个很复杂的事情，因为有时需要传递很多命令参数。作为一种功能强大的 Java 方面的开发工具，MyEclipse 开发工具不仅提供了 Applet 源代码的代码辅助、智能纠错等功能，而且还提供了如何直接创建和运行 Applet 程序功能。

1. 创建Applet程序

(1) 创建 Java Project 项目，首先通过选择菜单 File | New | Java Project 命令打开如图 23.7 所示的 Create a Java Project 对话框。在该对话中，除了需要填写 Project name 项外，还需要设置 Project layout 选项区域。

(2) Project layout 选项区域存在两个选项，默认选择 Create separate folders for sources and class files 选项，而对于 Applet 项目则需要选择 Use project folder as root for source and class files，具体选择如图 23.8 所示。这是因为该选项可以使 Java 源文件和编译后的目标文件存放在一起，HTML 文件可以直接应用编译后的 Applet 程序，不需要指定路径。

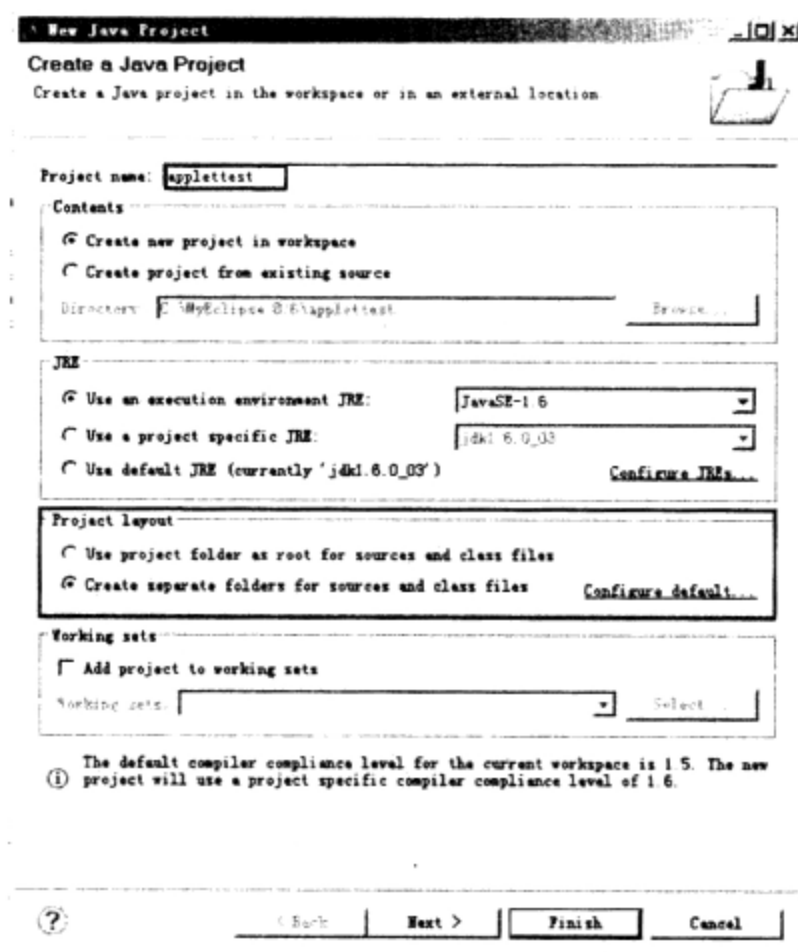


图 23.7 New Java Project 对话框

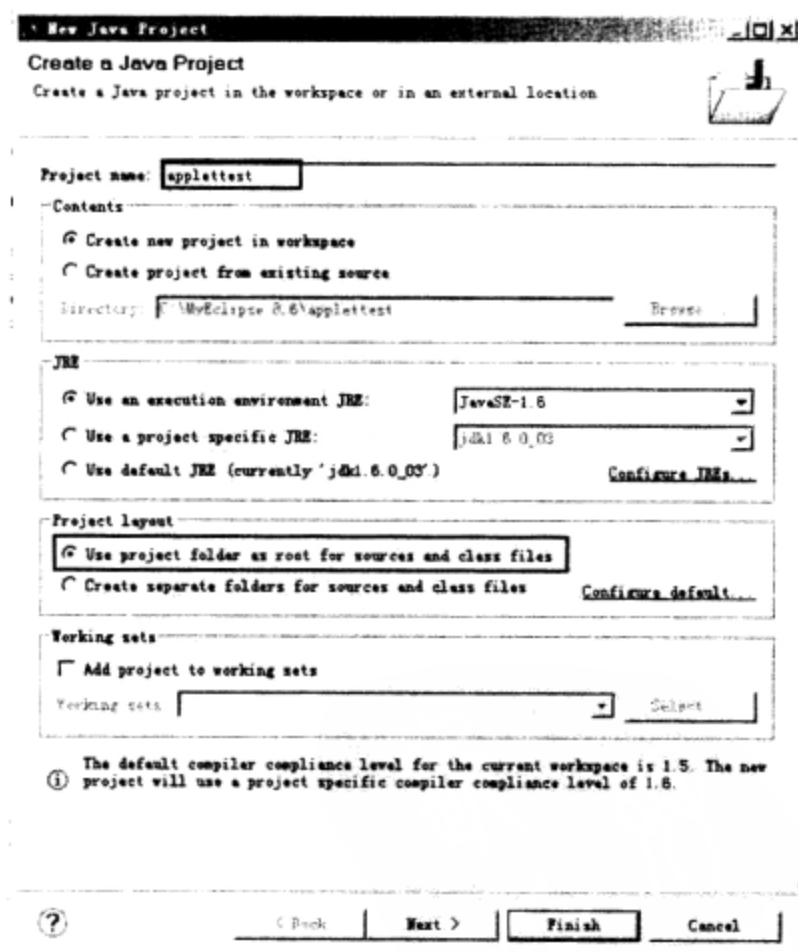


图 23.8 具体选择值

(3) 创建 Applet 文件，首先右击所创建的项目，然后在弹出的快捷菜单中选择 New | Applet 结点打开如图 23.9 所示的 Create a new Applet 对话框。在该对话中进行必要的设置后，单击 Next 按钮就可以打开 Applet HTML Wizard 设置对话框，如图 23.10 所示。当通过上述配置后，一般会取消 Generate HTML page 复选框的选择，最后单击 Finish 按钮就可

以实现 Applet 文件的创建。

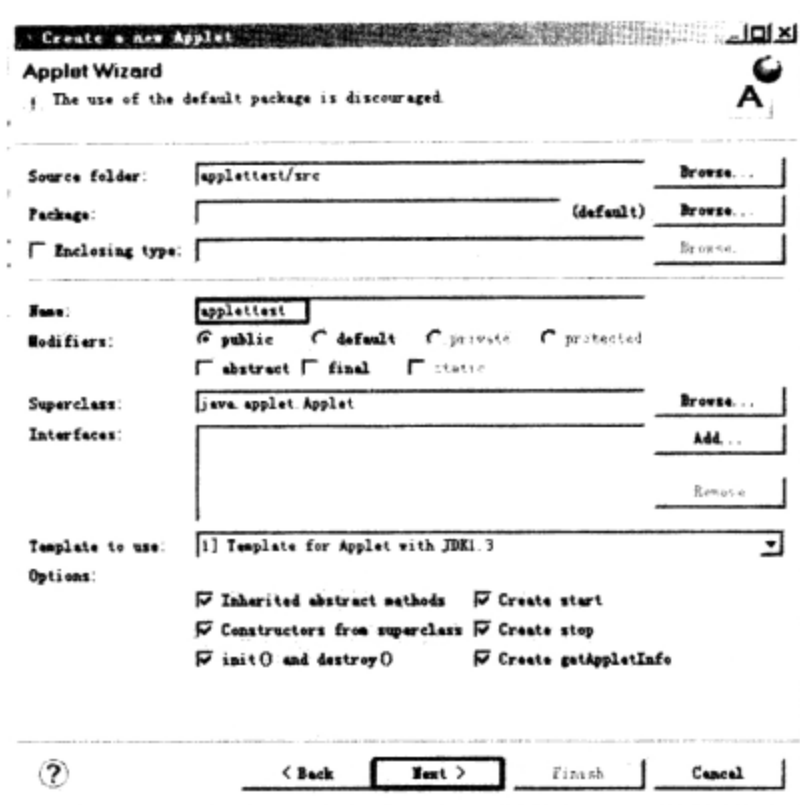


图 23.9 Create a new Applet 对话框

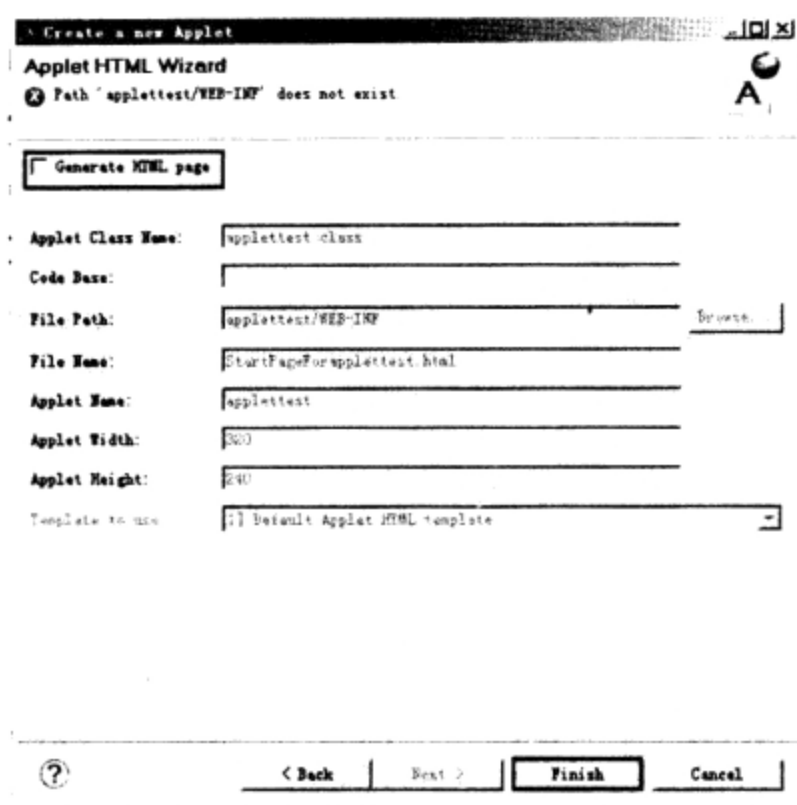


图 23.10 Applet HTML Wizard 设置

(4) 创建 HTML 文件，首先右击所创建的项目，然后在弹出的快捷菜单中选择 New | HTML 命令，打开如图 23.11 所示的 Html Wizard 对话框，在该对话中进行必要的设置后，单击 Finish 按钮就可以实现 HTML 文件的创建了。

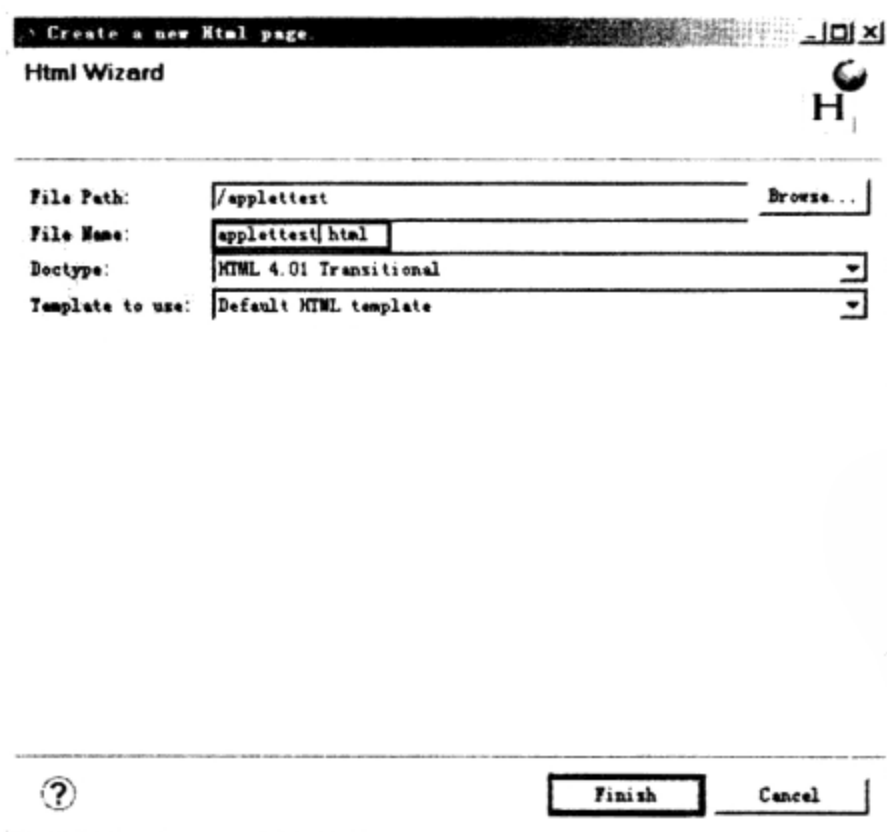


图 23.11 Html Wizard 对话框

2. 运行Applet程序

如果要使用 Appletviewer 查看 Applet 程序的运行结果，根据该 Applet 程序的情况可以分成以下两种情况。

(1) 无参数情况，首先右击所要查看的 Applet 程序，然后在弹出的快捷菜单中选择

Run As | Java Applet 结点就可以实现，如图 23.12 所示。

（2）带参数情况，首先右击选择所要查看的 Applet 程序，然后在弹出的快捷菜单中选择 Run As | Run Configurations 结点，如图 23.13 所示，打开如图 23.14 所示的 Create manage and run configurations 对话框。

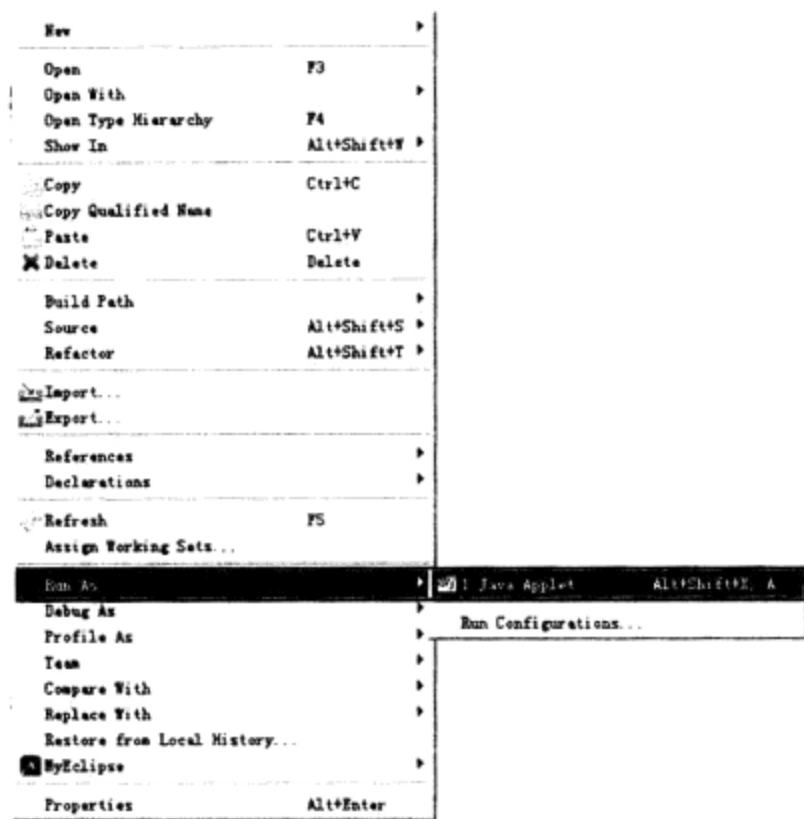


图 23.12 运行选项

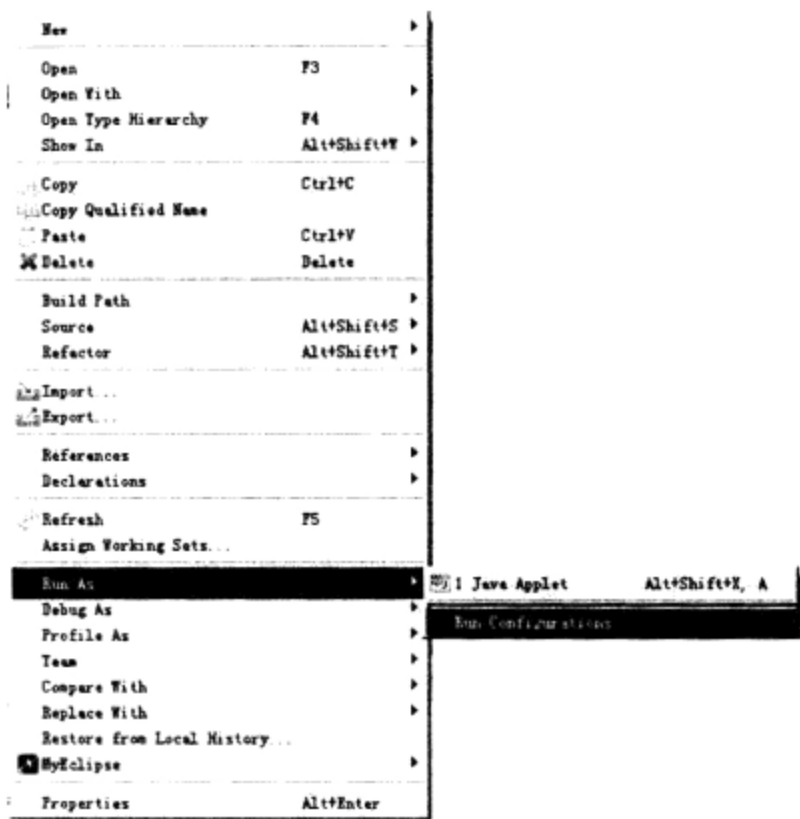


图 23.13 运行配置选项

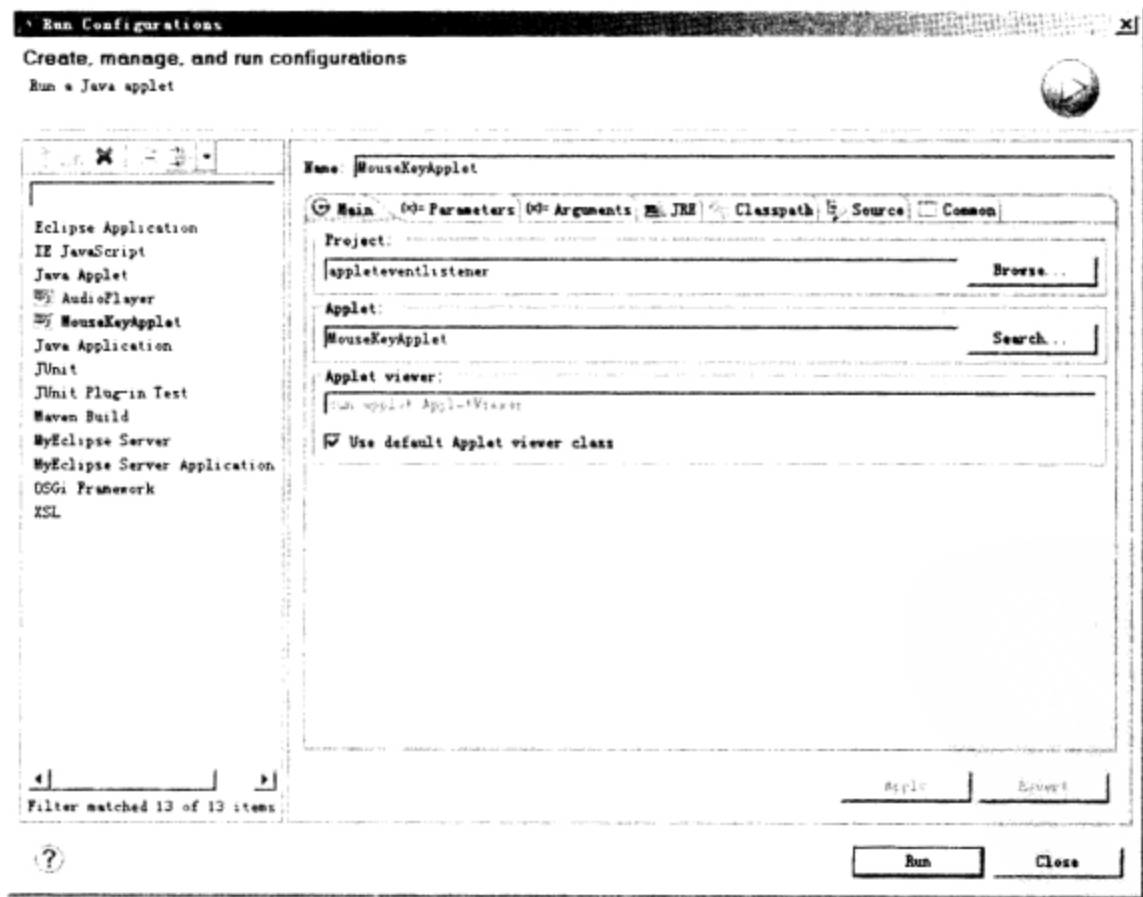


图 23.14 Run Configurations 对话框

对于该对话框的 Main 选项卡的设置，需要确认 Project 和 Applet 这两个选项的值是为所测试的 Applet 程序，具体设置参数如图 23.15 所示。

对于该对话框的 Parameters 选项卡的设置，如图 23.16 所示，主要是在 Parameters 区

域中为 Applet 程序添加参数值。通过单击 Add 按钮就可以打开 Add Parameter Variable 对话框,如图 23.17 所示,只要在该对话框中的 Name 选项中填写相应的参数名,Value 选项中填写相对应的参数值,然后单击 OK 按钮就可以在 Parameters 区域中显示出所设置的参数和参数值,如图 23.18 所示。

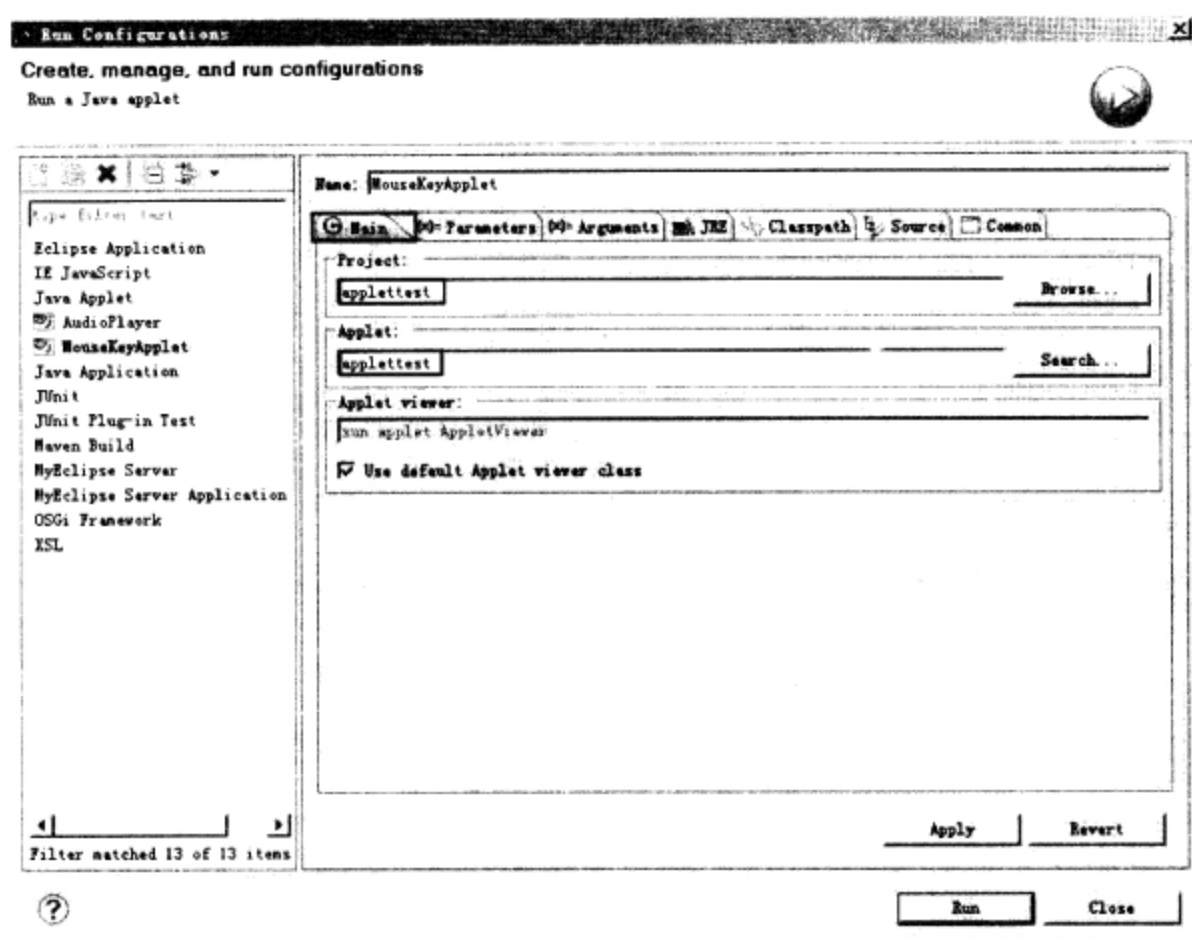


图 23.15 设置 Main 选项卡

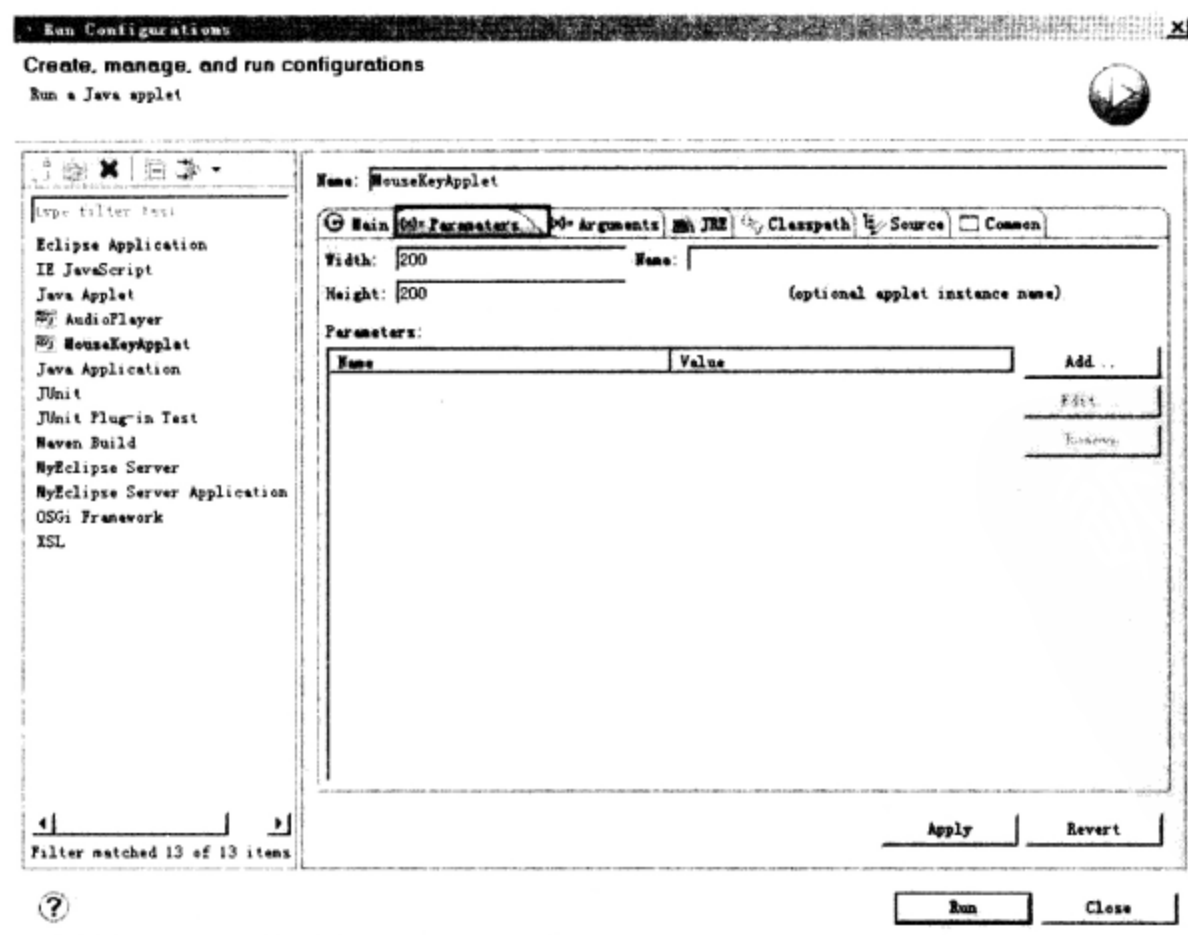


图 23.16 Parameters 选项卡

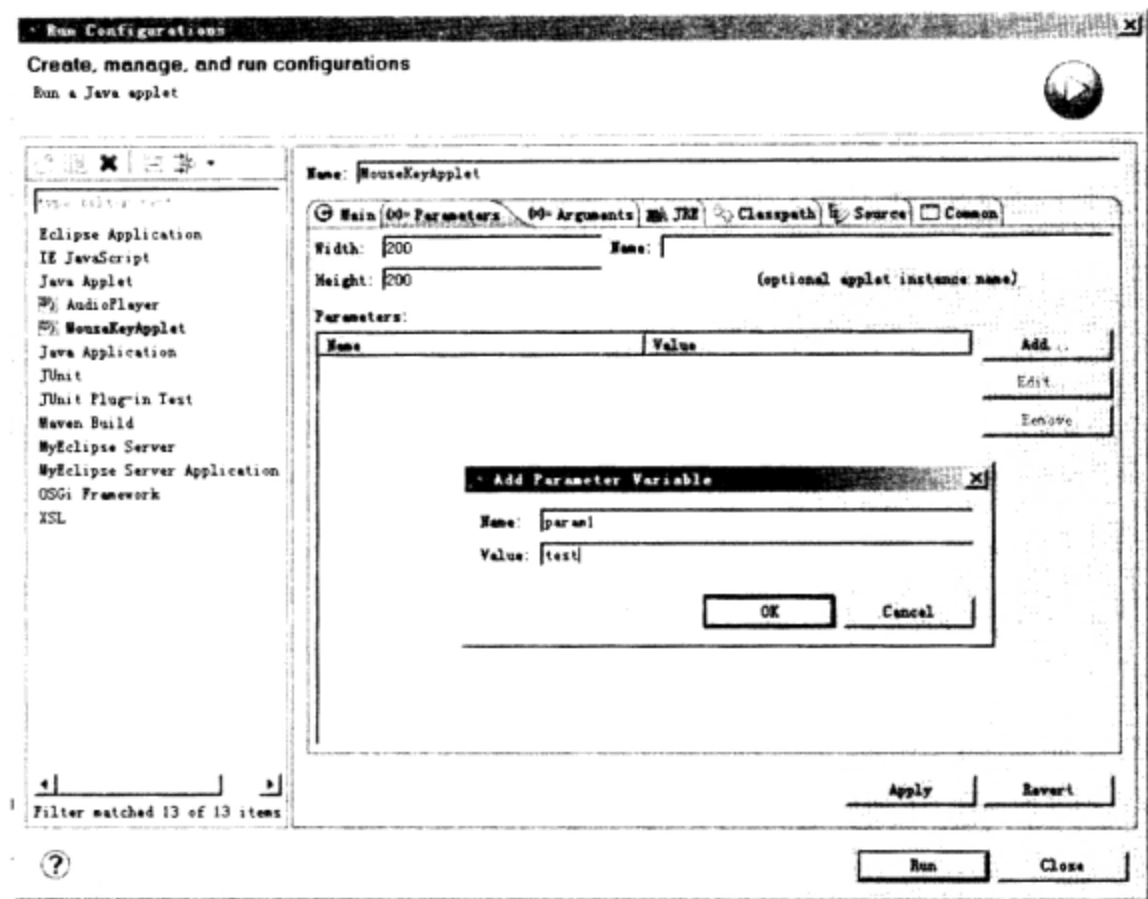


图 23.17 Add Parameter Variable 对话框

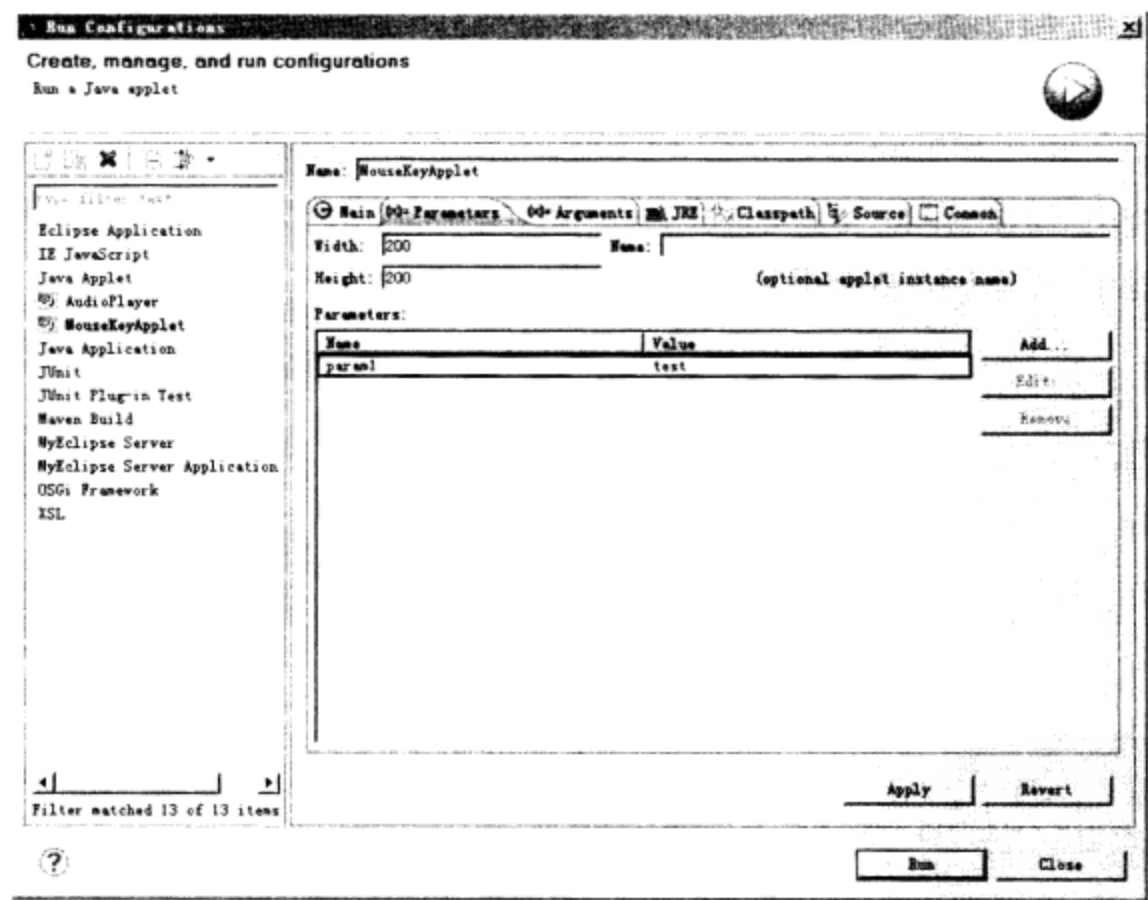


图 23.18 添加完参数值

添加完参数值后，单击对话框中的 Run 按钮，即实现 Applet 程序的查看功能，如图 23.19 所示。

23.3.2 MyEclipse 开发环境对 JAR 的支持

如果没有安装 MyEclipse 开发工具，生成 JAR 文件是一个很复杂的事情。作为一种强

大的 Java 开发工具，MyEclipse 开发工具也提供了导出成 JAR 文件的功能。之所以要获取相关代码的 JAR 文件，是因为在网页中要显示 Applet 程序时，可以直接设置 JAR 文件路径。

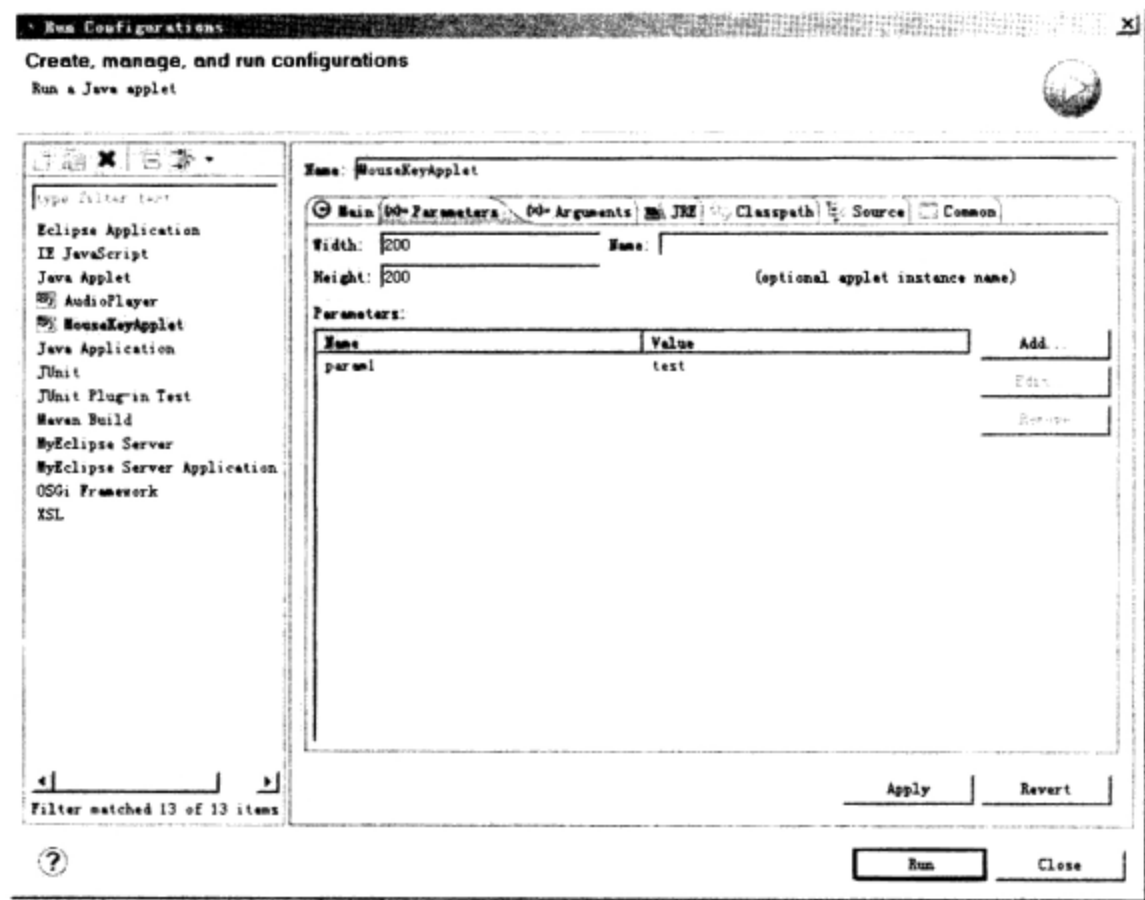


图 23.19 单击 Run 按钮

MyEclipse 开发环境要生成 JAR 文件，需要经历如下步骤。

(1) 右击需要导出成 JAR 文件的项目，在弹出的如图 23.20 所示的菜单中选择 Export 结点，会出现如图 23.21 所示的 Select 对话框。

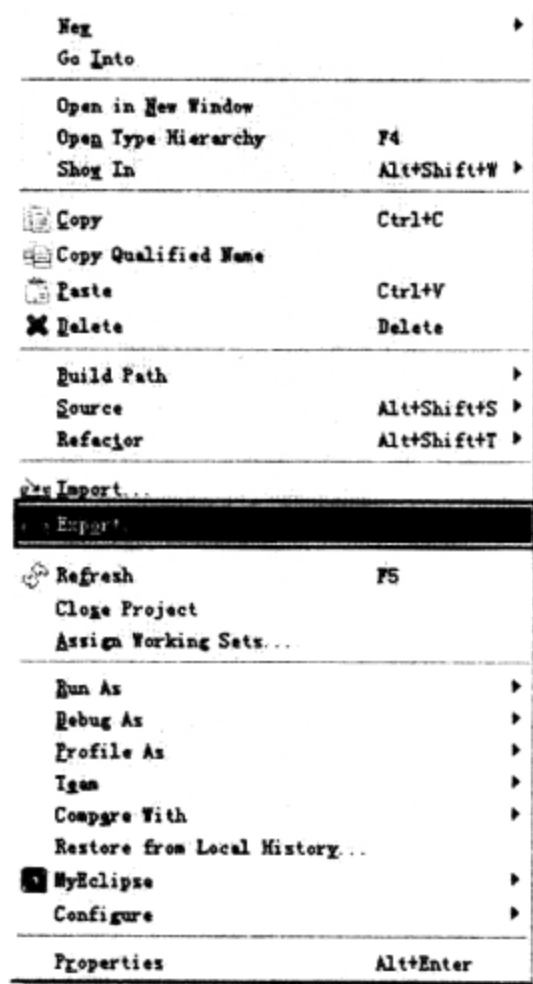


图 23.20 Export 菜单



图 23.21 Select 对话框

(2) 选择 Java | JAR file 文件结点，单击 Next 按钮就会出现如图 23.23 所示的 JAR File Specification 对话框。

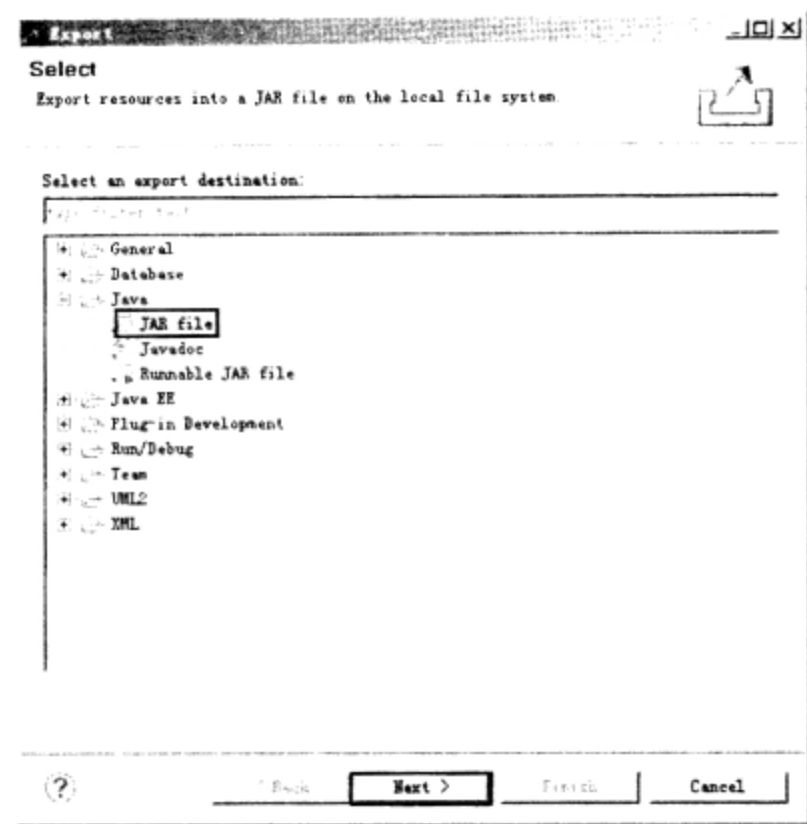


图 23.22 选择 JAR file 选项

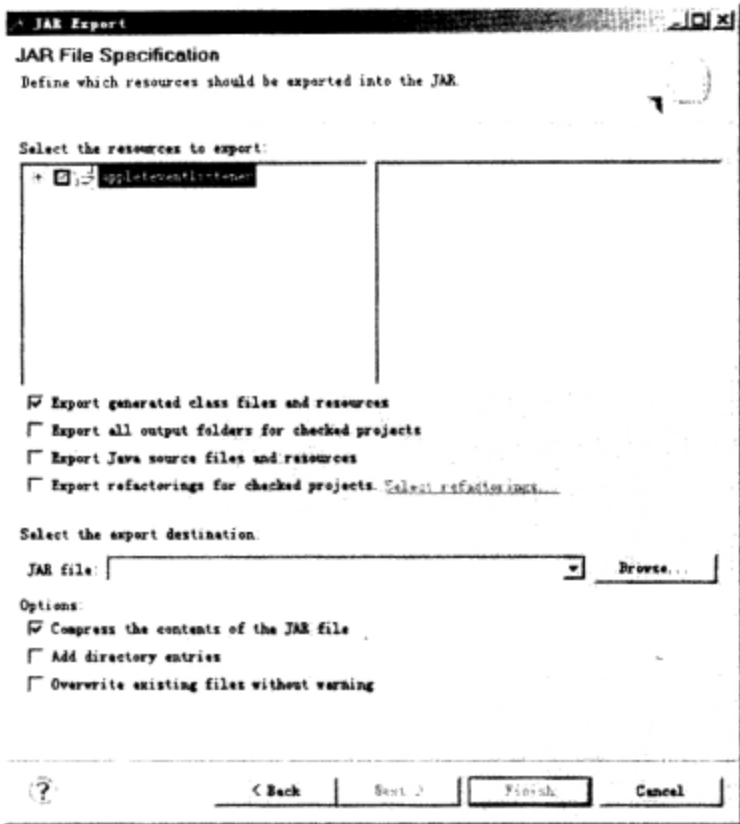


图 23.23 JAR File Specification 对话框

(3) 在 Select the resources to export 区域中选择将要导出的 Java 程序代码和各种资源文件。在 Select the export destination 区域通过单击 Browse 按钮来选择保存 JAR 文件的路径和文件名称，最后 JAR Export 对话框的设置如图 23.24 所示。

(4) 最后，单击 Next 按钮进入如图 23.25 所示的 JAR Packaging Options 对话框。在该对话框的 Select options for handling problems 区域中，一般会选择 Export class files with compile errors 和 Export class files with compile warnings 这两个选项，然后单击 Finish 按钮就可以完成 JAR 文件的导出。

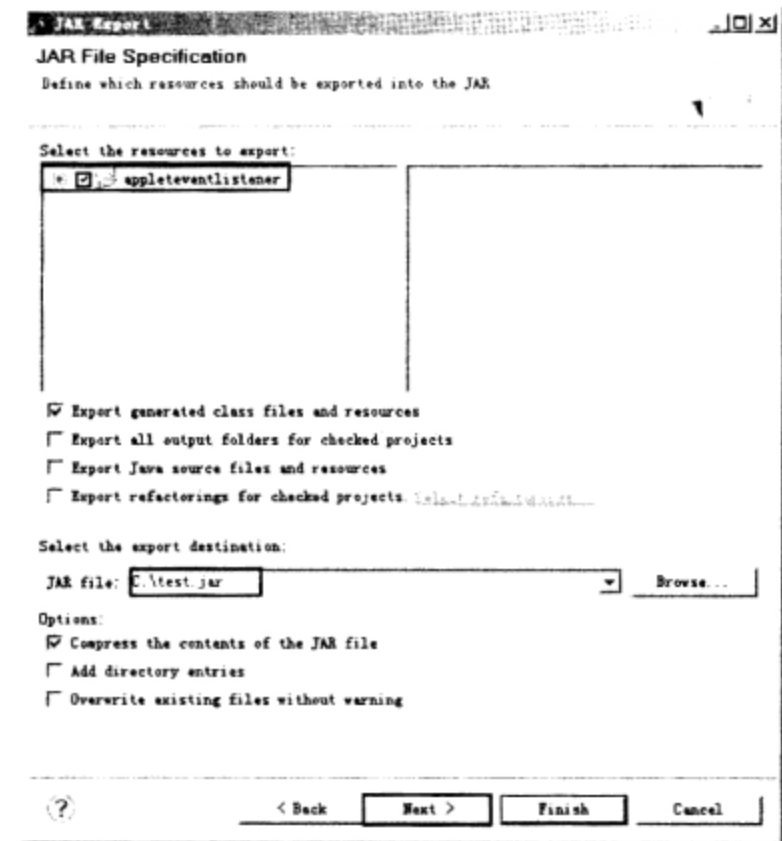


图 23.24 相应配置

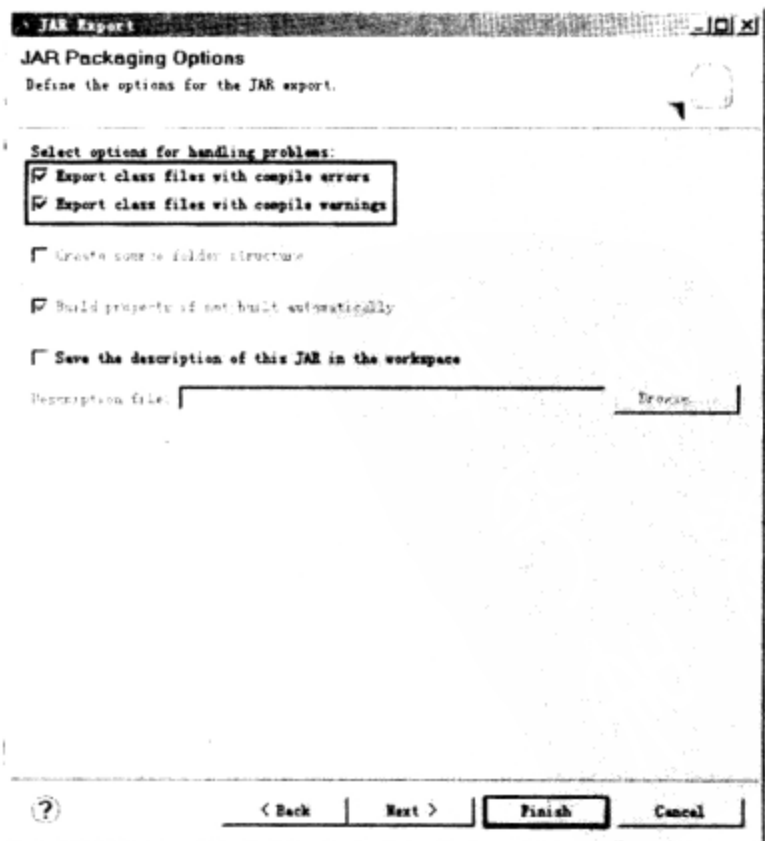


图 23.25 JAR Packaging Options 对话框

⚠注意：随着 JDK 版本的不断更新，一些老版本中的方法虽然能够正常完成业务处理，但是已经不被新版本的 JDK 所推荐。如果不选择上述对话框中的选项，则会导出带有编译警告的 JAR 文件。

23.4 小 结

本章主要通过 Applet 程序中的事件处理机制来实现在 Applet 程序中监听鼠标和键盘的事件，虽然该项目比较小，只包含一个实现监听功能的 `MouseKeyApplet` 类和承载 Applet 的 `TestMouseKeyWeb.html` 页面，但是该项目却涉及了 Applet 程序中事件处理的所有知识点。在本章的最后还详细介绍了 MyEclipse 开发环境对 Applet 程序的各种支持。

第 24 章 动画播放项目（音频操作+多线程）

在 Applet 程序中不仅可以操作图像，而且还可以实现对音频的操作。本章不仅通过“动画播放项目”讲解如何实现动画，还会详细讲解 Applet 程序中音频操作的基础知识。

本章的学习目标如下：

- ❑ 掌握动画播放项目；
- ❑ 了解 Applet 程序中音频的操作。

24.1 动画播放原理

“动画播放”项目通过每隔一定时间就修改图像相同的位置来达到移动的动画效果，在具体运行程序时，会通过界面中的相应按钮实现动画的播放和停止。

24.1.1 项目结构框架分析

对于“动画播放”项目，根据面向对象的思想，需要创建一个动画的对象。“动画播放”项目目录如图 24.1 所示，各个目录的功能如下。

- ❑ MoveImage：动画类的对象。
- ❑ Controler：操作动画类的类。

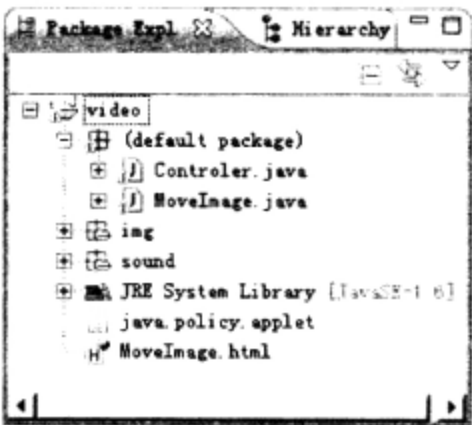


图 24.1 项目目录

24.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括动画播放的初始化，按住鼠标功能和放开鼠标后的功能。

1. 初始化界面

当运行“动画播放”项目中的 Controler 类后，会出现如图 24.2 所示的初始界面。



图 24.2 初始界面

2. 动画的运行

当出现初始界面后，界面中的云彩图片会随着时间的推移而从左向右移动，当单击“停止”按钮时，云彩图片就不会移动；如果单击“播放”按钮，云彩图片还会继续移动。最后当云彩图片超出动画界限时，该图片会从左边重新开始移动，如图 24.3 所示。

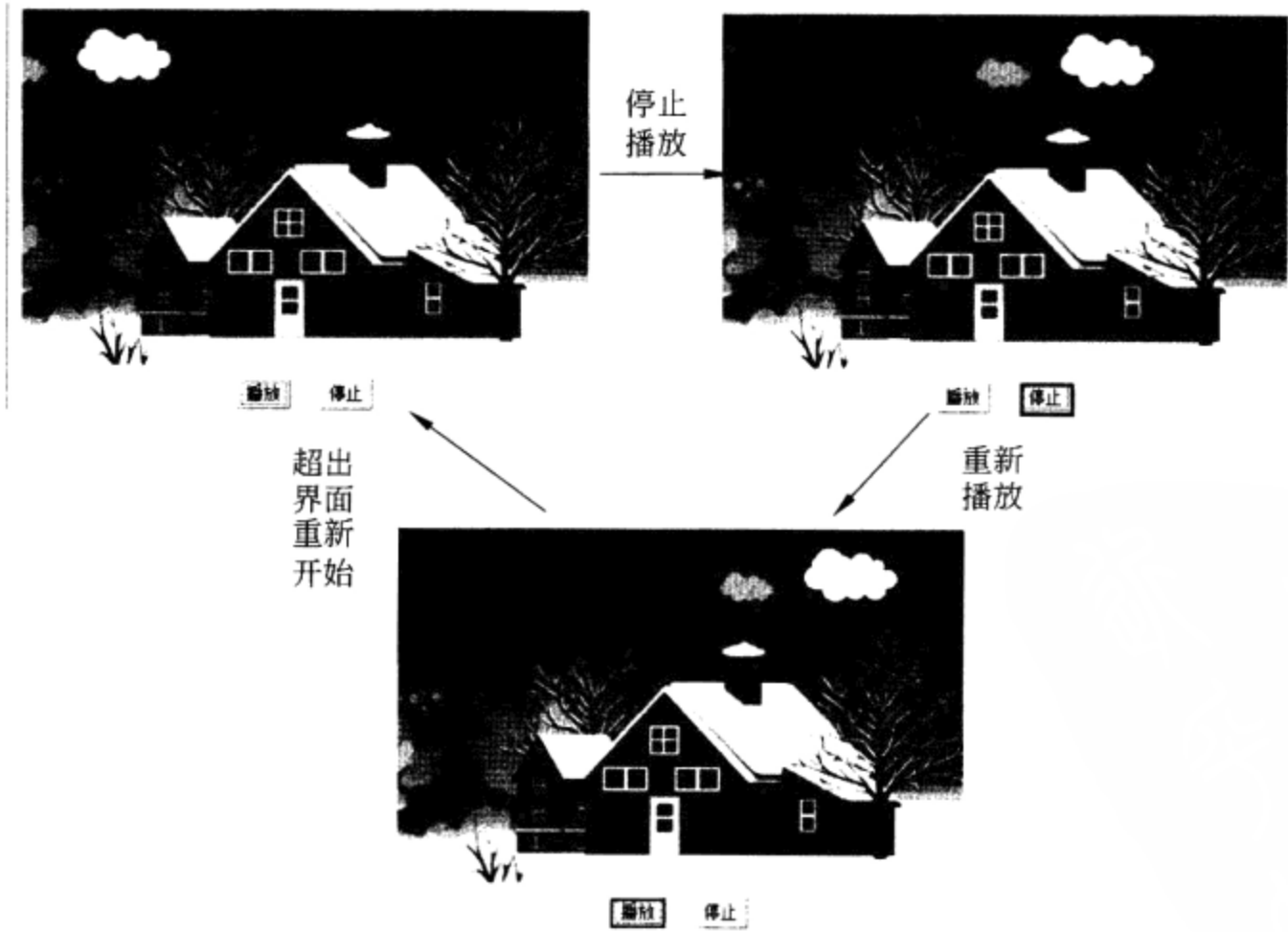


图 24.3 动画的具体过程

3. 重绘功能

将窗口最小化后再恢复正常化显示的过程，其实就是重绘的过程。如果考虑重绘功能，

恢复正常化显示就会出现正常的窗口；否则就会出现空白窗口，具体过程如图 24.4 所示。

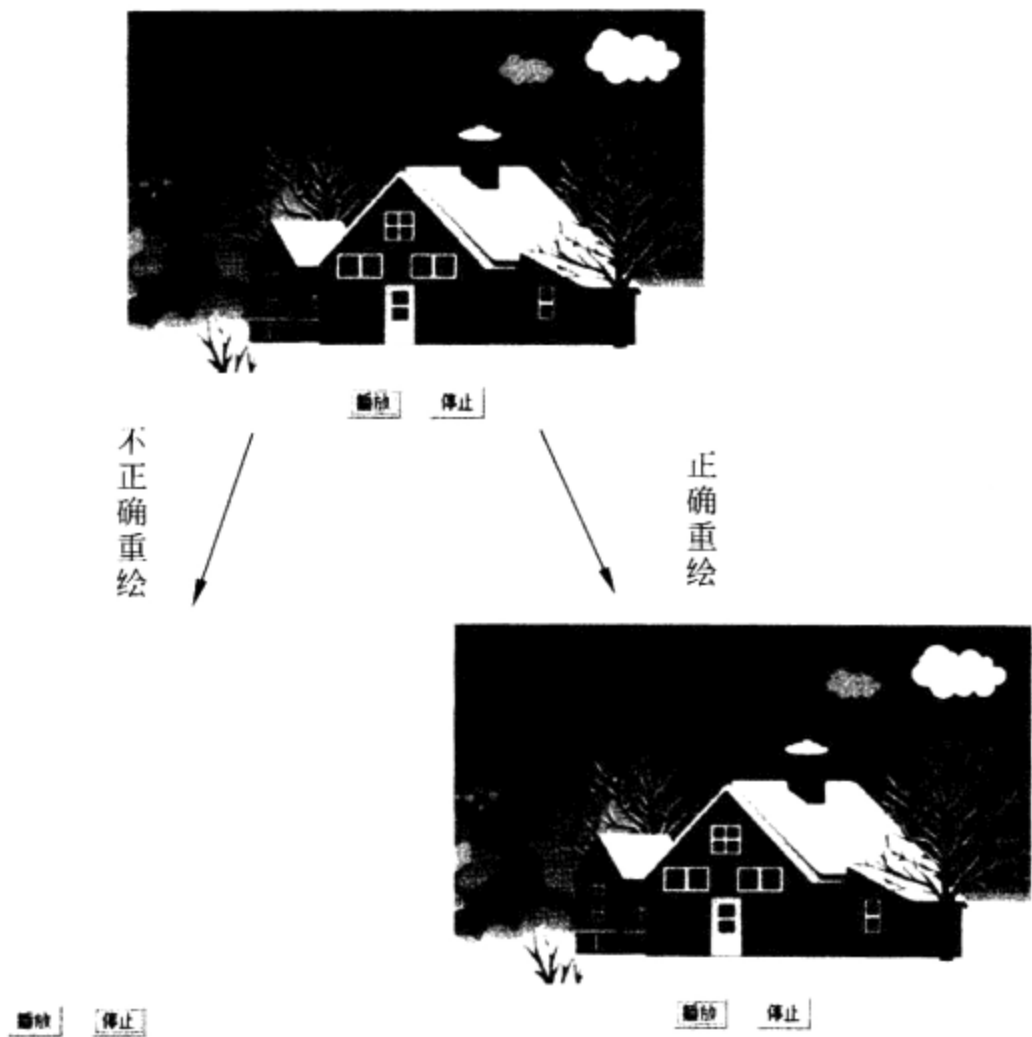


图 24.4 重绘的过程

24.2 动画播放项目

动画播放项目具体程序架构如图 24.5 所示，它包含一个动画对象的类 MoveImage.java 和一个控制动画的类 Controler.java。该项目通过定时设置图像的坐标来模拟云彩图片移动的动画效果，主要利用了 Java 语言中的线程机制和图形双缓冲技术。

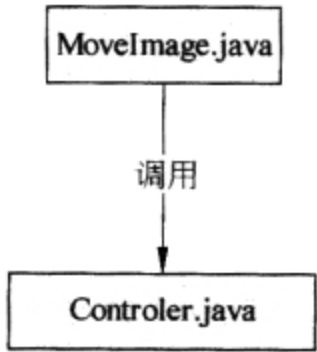


图 24.5 程序关系图

24.2.1 动画的类

MoveImage.java 为动画对象类，该对象拥有几个属性和操作自己的方法。该类的具体内容如代码 24.1 所示，该类的 UML 如图 24.6 所示。

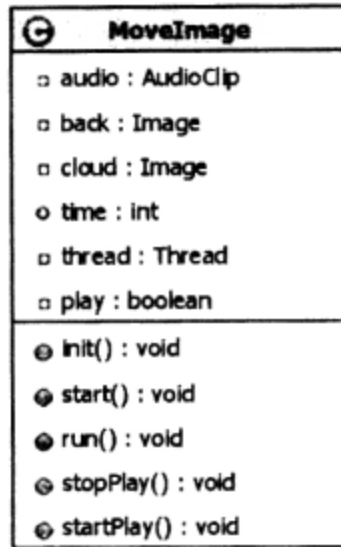


图 24.6 动画类图

代码 24.1 动画类: MoveImage.java

```

public class MoveImage extends JApplet implements Runnable {
    //创建成员变量
    private Image back; //背景图像变量
    private Image cloud; //云彩图像的变量
    int time; //云彩移动速度变量
    private Thread thread; //线程变量
    private boolean play = true; //播放标记的变量

    public void init() {
        setBackground(Color.PINK); //设置背景颜色
        //获取背景和云彩的图片
        back = getImage(getCodeBase(), "img/back.jpg");
        cloud = getImage(getCodeBase(), "img/cloud.png");
        String paramStr = getParameter("time"); //获取时间字符串
        //为变量time赋值
        if (paramStr != null)
            time = Integer.parseInt(paramStr);
        else
            time = 100;
    }

    public void start() { //实现 start() 方法
        startPlay(); //调用 startPlay() 方法
    }

    public void run() { //实现 run() 方法
        Graphics g = getGraphics(); //获取 Graphics 对象
        //获取长度和宽度
        int width = getWidth();
        int height = getHeight();
        int x = width; //云的显示位置
        Image buff = createImage(width, height); //双缓冲对象
        Graphics cache = buff.getGraphics(); //缓冲对象的图片对象
        while (play) {
            //为缓冲图片对象绘制背景和云
            cache.drawImage(back, 0, 0, width, height, this);
            cache.drawImage(cloud, x, 10, this);
            try {
                Thread.sleep(time); //线程休眠
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
  
```



```

        g.drawImage(buff, 0, 0, this);           //把缓冲对象画在 Applet
        x -= 2;                                   //改变云的位置
        if (x <= -cloud.getWidth(this))          //防止云过界
            x = width;
    }
}
public void stopPlay() {                         //停止播放动画的方法
    play = false;                                //设置播放标记位
}
public void startPlay() {                        //开始播放动画的方法
    play = true;                                 //设置播放标记位
    if (thread == null || !thread.isAlive())     //判断线程
        thread = new Thread(this);              //创建新线程
    thread.start();                              //启动新线程
}
}
```

【代码解析】

- ❑ 动画类拥有两个属性：背景图片和云彩图片。由于该类主要实现云彩动的效果，所以有两个动画，开始播放云彩动的方法 `startPlay()`和停止云彩动的方法 `stopPlay()`。
- ❑ 由于动画类继承了 `JApplet` 类并实现了 `Runnable` 接口，所以该类中会重写 `init()`方法，实现 `start()`和`run()`方法。`init()`方法主要为图片对象赋值及设置线程休眠的时间。而在 `run()`方法中，主要是实现云彩图片移动的效果，让线程每休眠一次就让云彩的位置移动一次。
- ❑ 在动画播放的 `startPlay()`方法中，首先设置播放标志，然后创建新的线程，最后通过线程的 `start()`方法调用 `run()`方法实现动画效果。在动画停止的 `stopPlay()`方法里，只要设置播放标记，就可以实现动画的停止。

24.2.2 控制动画的类

`Controler.java` 类用来控制动画，该类首先通过 `Applet` 程序的上下文对象获取动画的 `Applet` 程序对象，然后通过动画对象的相应方法实现对其的控制，该类的具体内容如代码 24.2 所示，该类的 UML 如图 24.7 所示。

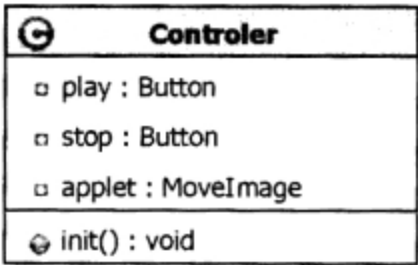


图 24.7 控制动画类图

代码 24.2 控制动画类：Controler.java

```

public class Controler extends Applet {
    private Button play;           //播放按钮
    private Button stop;          //停止按钮
    private MoveImage applet;      //动画对象
}
```

```

public void init() {                                     //初始化方法
    setLayout(new FlowLayout(FlowLayout.CENTER, 20, 5)); //设置布局管理器

    //为按钮对象赋值和设置
    play = new Button("播放");
    stop = new Button("停止");
    add(play);
    add(stop);
    final AppletContext context = getAppletContext();    //获取 Applet 的上下文

    play.addActionListener(new ActionListener() {       //添加事件监听器
        public void actionPerformed(ActionEvent e) {
            //通过上下文对象为动画对象赋值
            if (applet == null)
                applet = (MoveImage) context.getApplet("imagePlayer");
            applet.startPlay();                          //播放动画
        }
    });
    stop.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            //通过上下文对象为动画对象赋值
            if (applet == null)
                applet = (MoveImage) context.getApplet("imagePlayer");
            applet.stopPlay();                          //停止动画
        }
    });
}
}

```

【代码解析】

上述代码由于是 Applet 程序，所以继承了 Applet 类。在 Applet 程序界面中，有两个按钮对象来控制动画的播放和停止，在具体编写播放的按钮监听器方法中，通过动画对象的 startPlay()方法就可以实现动画的播放；在具体编写停止的按钮监听器方法中，通过动画对象的 stopPlay()方法就可以实现动画播放的停止。

24.3 知识点扩展——Applet 程序的高级知识

在动画播放项目中，不仅可以实现动画播放，还可以实现音频的播放。那么在 Applet 程序中如何实现音频播放呢？细心的读者可能会发现，在该项目中有两个 Applet 程序，但是却被承接在同一个网页中，并实现相互控制，这些功能的实现使用了 Applet 程序中的哪些对象呢？

24.3.1 音频播放

在 Java 语言中除了可以处理图片资源外，而且还可以处理音频等多媒体资源。在 Applet 程序中，可以通过 AudioClip 接口的相关方法来操作音频。对于 Applet 程序，支持的主要音频格式有 AIFF、AU、MIDI、WAV、RMF 等。

在 Applet 程序中，播放音频是个很简单的事情，首先加载音频文件，然后调用 play()

方法播放就可以。查看 API 帮助文档，可以发现 Java 提供了两种播放声音的方式，通过 Applet 的 play()方法实现音频文件的加载和播放；通过 AudioClip 接口中的相应方法。

1. play()方法

Applet 类的 play()方法语法格式如下：

```
void play(URL url)
```

参数 url 为声音资源的地址，该函数用于加载和播放指定地址的声音文件。

```
void play(URL url,String name)
```

参数 name 为声音文件的名称，该函数用于加载和播放指定地址及文件名的声音文件。

上述方法加载完声音文件后就立即播放，如果找不到指定的声音文件，也不会产生异常。该种方式的播放只是一次性的，若要重播，则必须重新加载声音文件。

2. getAudioClip()方法

Applet 类的 getAudioClip()方法会返回一个接口 AudioClip 的实例，该方法的语法格式如下：

```
AudioClip getAudioClip(URL url)
```

参数 url 为音频资源的地址，该函数用于加载指定地址的音频文件并返回 AudioClip 对象。

```
AudioClip getAudioClip(URL url,String name)
```

参数 name 为音频文件的名称，该函数用于加载指定地址和文件名的音频文件并返回 AudioClip 对象。

上述方法首先创建 AudioClip 对象，然后加载音频文件。如果该方法没有找到指定的音频文件，将返回 Null 对象。此时所返回的 AudioClip 对象将不能被引用。

3. newAudioClip()方法

除了利用 Applet 类的 getAudioClip()方法获取 AudioClip 对象外，还可以通过该类的 newAudioClip()方法来实现，该方法的语法格式如下：

```
AudioClip newAudioClip(URL url)
```

参数 url 为音频资源的地址，该函数用于加载指定地址的音频文件并返回 AudioClip 对象。

由于该方法是静态方法，所以其不仅可以应用在 Applet 应用程序中，而且还可以应用到其他 Java 程序中。

获取到 AudioClip 对象后，就可以通过该实例中的相应方法实现相应的功能，例如可以利用 play()方法播放音频文件，利用 stop()方法停止当前播放的音频文件及利用 loop()方法循环播放音频文件。

注意：由于 play()方法并不能返回 AudioClip 对象，所以该方法只能操作声音文件，而不能操作视频文件。

下面通过一个名为 `AudioPlayer` 的类来具体讲解如何实现音频的播放，该类的具体内容如代码 24.3 所示。

代码 24.3 播放声音的类: `AudioPlayer.java`

```
public class AudioPlayer extends JApplet {  
    public void start() {  
        play(getCodeBase(), "08.WAV");  
    }  
}
```

【代码解析】

上述代码中，在 `start()` 方法中通过 `play()` 方法播放名称为 `08.WAV` 的音乐文件。

接着再编写一个承载 `AudioPlayer` 类的页面 `TestAudioPlayerWeb.html`，该页面的具体内容如代码 24.4 所示。

代码 24.4 承载网页: `TestAudioPlayerWeb.html`

```
<body>  
    <Applet code="AudioPlayer.class" width=200 height=100>  
    </Applet>  
</body>
```

当通过 IE 浏览器浏览上述页面时，就会自动播放音乐。

24.3.2 Applet 的上下文对象

在一个网页中可以存在多个 Applet 程序，如果想在同一个 Applet 程序中操作同网页中的其他 Applet 程序，可以通过实现接口 `AppletContext` 的上下文对象来实现。通过上下文对象可以获取同一个网页中的所有 Applet 程序、在浏览器的状态栏显示信息等功能。与上下文相关的方法如下。

1. `getAppletContext()` 方法

`getAppletContext()` 方法语法格式如下：

```
AppletContext getAppletContext()
```

该函数用于返回 Applet 程序的上下文对象。获取上下文对象后，就可以通过该对象的方法来获取同一个上下文中其他 Applet 程序，从而实现对其他 Applet 程序的控制。

2. `getApplets()` 方法

`getApplets()` 方法语法格式如下：

```
Enumeration<Applet> getApplets()
```

该函数返回属于同一个上下文中的所有 Applet 程序。

3. getApplets()方法

getApplet(String name)方法语法格式如下：

```
Applet getApplet(String name)
```

参数 name 为 Applet 程序的名称，该函数返回属于同一个上下文中指定名称的 Applet 程序。

24.4 小 结

本章主要通过 Applet 程序中的音频操作技术、显示图像技术、图像重绘技术和多线程技术来实现在 Applet 程序中动画播放的功能，虽然该项目比较小，只包含实现动画的 MoveImage 类、实现控制动画的 Controler 类和承载 Applet 的 MoveImage.html 页面，但是该项目却涉及了 Applet 程序中音频操作的所有知识点。

在本章的最后还详细介绍了 Applet 程序的高级知识点：音频操作和上下文对象，通过这些知识的学习，可以更好地掌握 Applet 程序的编写。

第 6 篇 网络编程

- ▶▶ 第 25 章 网络聊天室（UDP 协议+多线程）
- ▶▶ 第 26 章 FTP 服务器客户端（FtpClient+I/O 处理）
- ▶▶ 第 27 章 Web 服务器（HTTP 协议）
- ▶▶ 第 28 章 QQ 聊天工具（Swing+多线程+网络编程）

第 25 章 网络聊天室

(UDP 协议+多线程)

在 Java 语言的网络编程中经常会涉及两种协议 UDP 协议（无连接协议）和 TCP（Transmission Control Protocol，传输控制协议），在具体编写相应的网络程序时，经常会用到 UDP 协议相关的类，所以掌握该协议对于程序员来说是必不可少的。本章将通过模拟网络聊天室的功能介绍 Java 语言中的 UDP 协议和事件机制，还会详细介绍网络编程的基础知识和 UDP 协议。

本章的学习目标如下：

- ❑ 掌握网络聊天室项目原理；
- ❑ 理解网络编程的 Socket 协议和 UDP 协议；
- ❑ 掌握多线程机制。

25.1 网络聊天室原理

“网络聊天室项目”用来模拟现实生活中网络聊天室的功能，在具体使用时如果想计时开始则按住鼠标，如果想停止计时则放开鼠标。由于该系统需要处理鼠标的事件，所以必须了解事件机制。

25.1.1 项目结构框架分析

对于网络聊天室项目，根据 C/S 软件框架可以知道，需要创建服务器端程序和客户端程序，但是由于该项目中服务器端功能和客户端功能是集成在一起的，所以该项目只存在一个类，其目录如图 25.1 所示。

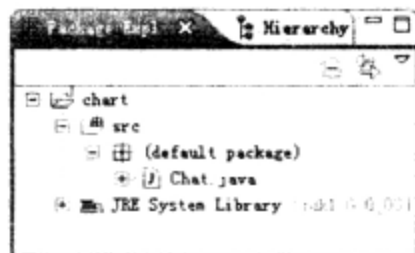


图 25.1 项目目录

25.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能，这些功能包括网络聊天室的初始化，聊天

过程的功能。

1. 网络聊天室的初始化

当运行网络聊天室的 Chat 类后，会出现如图 25.2 所示的初始界面。在该界面的上部分为显示接收信息的列表框，下部分的左边用来输入对方的 IP 地址，右边为所要发送的信息，如图 25.2 所示。



图 25.2 网络聊天室的初始界面

2. 聊天过程

网络聊天室要实现聊天功能，首先需要在 IP 地址为 192.168.1.15 的初始化界面里，输入需要聊天对象的 IP (192.168.1.19) 地址和聊天内容 (hello,19)，然后按下 Enter 键就可以在目标地址的初始化界面里出现聊天内容 (hello,19:from 192.168.1.15)，具体过程如图 25.3 所示。

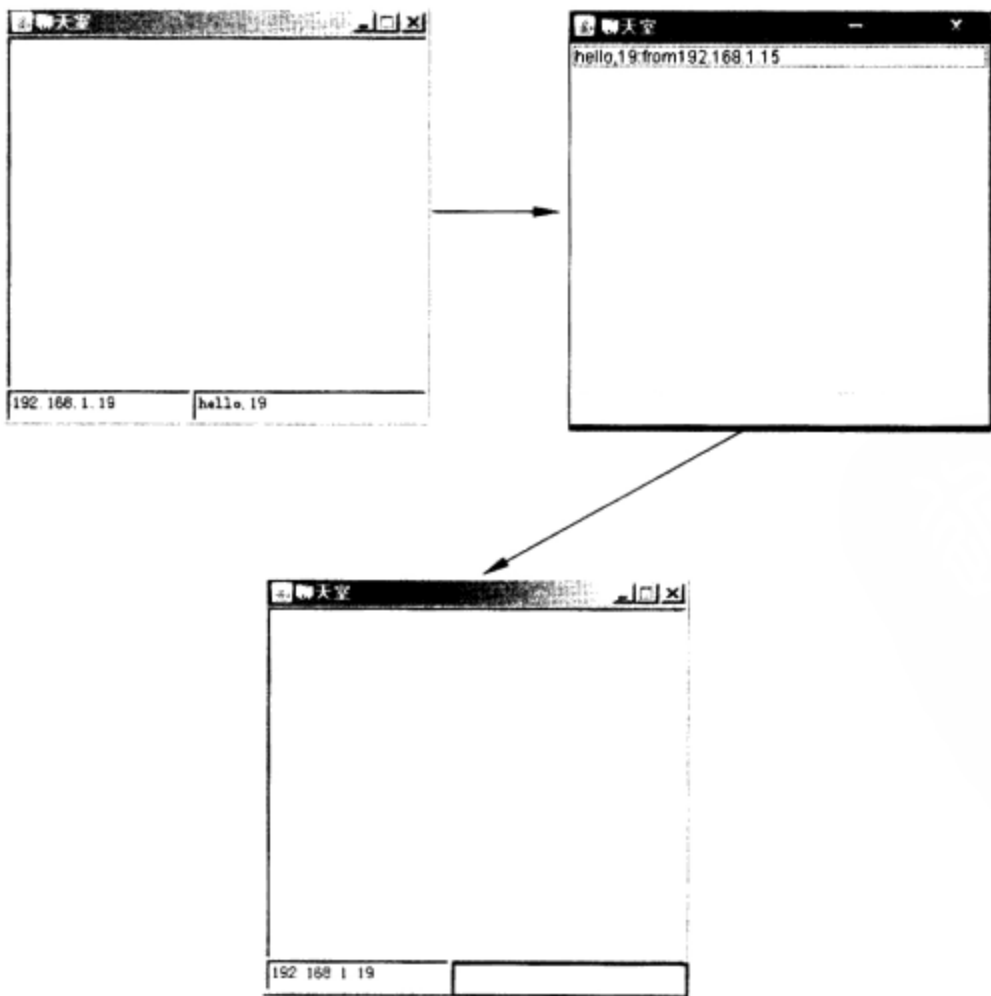


图 25.3 聊天过程

3. 聊天过程

当 IP 地址为 192.168.1.19 的初始化界面接收到聊天内容后，如果想回复，同样可以在该初始化界面的相应地方输入需要聊天对象的 IP（192.168.1.15）地址和聊天内容（hello,15），然后按下 Enter 键就可以在目标地址的初始化界面里出现聊天内容（hello,15:from 192.168.1.19），具体过程如图 25.4 所示。

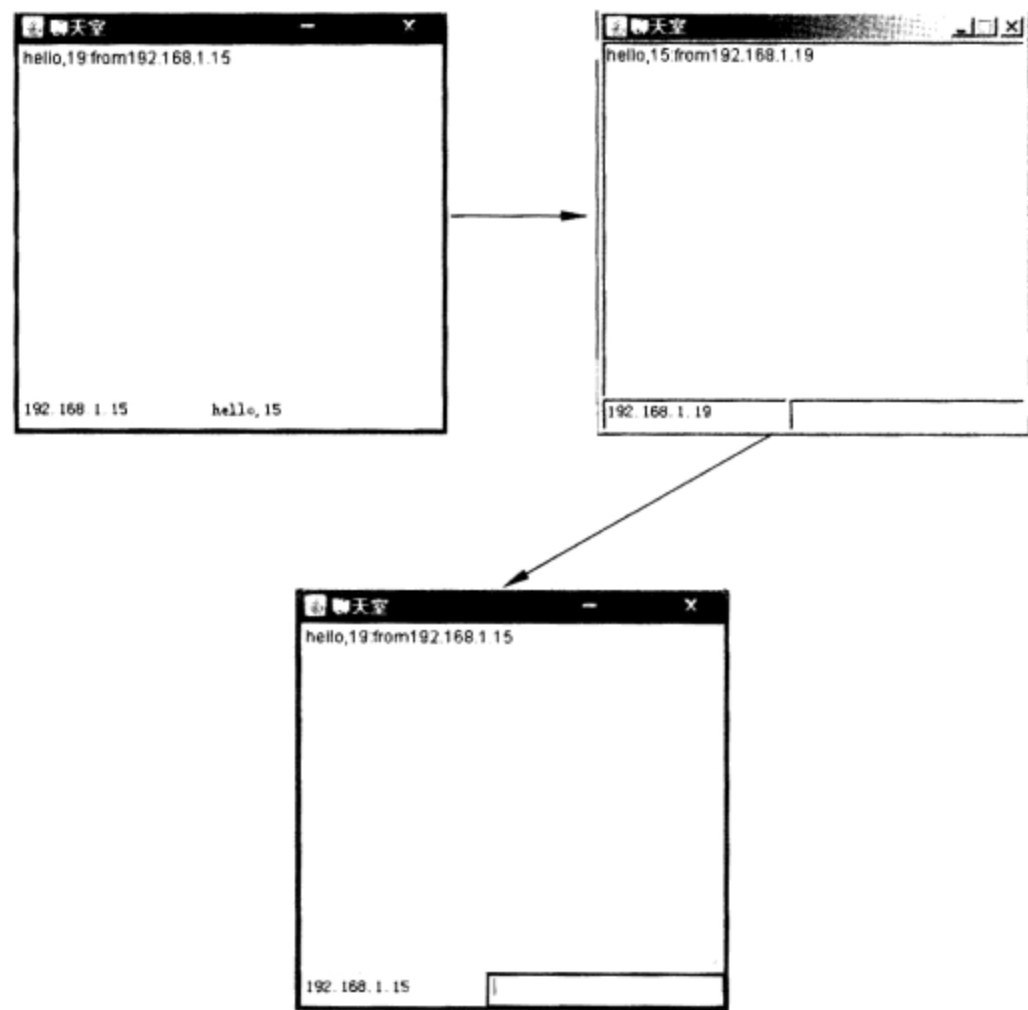


图 25.4 聊天过程

25.2 网络聊天室的实现过程

本章通过 UDP 协议类和多线程的相关知识来实现网络聊天室项目，由于该项目不仅要实现发送信息功能，而且还需要实现接收信息功能，所以服务器端程序和客户端程序就集成到一个类——Chat 里。

Chat.java 类用来模拟网络聊天室，该类涉及了 UDP 协议的类和多线程知识，其具体内容如代码 25.1 所示，该类的 UML 如图 25.5 所示。

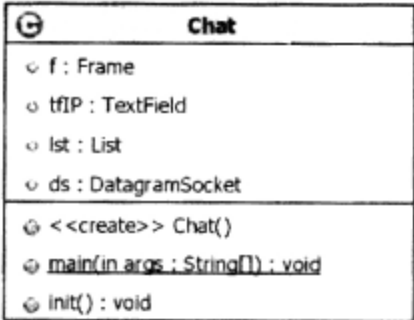


图 25.5 网络聊天室类图

代码 25.1 网络聊天室的类: Chat.java

```

public class Chat {
    Frame f = new Frame("聊天室");           //创建 Frame 对象
    TextField tfIP = new TextField(15);        //创建 TextField 对象
    List lst = new List(6);                    //创建列表框对象
    DatagramSocket ds;                         //创建 DatagramSocket 对象
    public Chat() {                            //构造函数
        try {
            ds = new DatagramSocket(3000);     //为 DatagramSocket 对象赋值
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    public static void main(String[] args) {   //主方法
        Chat chat = new Chat();               //创建 Chat 类对象
        chat.init();                           //初始化方法
    }
    public void init() {                       //初始化方法
        f.setSize(300, 300);                  //设置窗口大小
        f.add(lst);
        Panel p = new Panel();                //创建一个面板对象 p
        p.setLayout(new BorderLayout());        //设置面板对象的布局管理器
        p.add("West", tfIP);                   //添加对象 tfIP 到对象 p 里
        TextField tfData = new TextField(20); //创建 TextField 对象
        p.add("East", tfData);                 //添加对象 tfData 到对象 p 里
        f.add("South", p);                     //添加面板对象 p 到窗口里
        new Thread(new Runnable() {
            public void run() {
                byte buf[] = new byte[1024];
                //创建 DatagramPacket 对象
                DatagramPacket dp = new DatagramPacket(buf, 1024);
                while (true) {
                    try {
                        ds.receive(dp);          //接收信息
                        //在列表框里显示相应的信息
                        lst.add(new String(buf, 0, dp.getLength()) + ":from"
                                + dp.getAddress().getHostAddress(), 0);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
        }).start();
        f.setVisible(true);                    //显示窗口
        f.setResizable(false);                 //禁止用户修改窗口大小
        //关闭窗口的事件处理代码
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                ds.close();                     //关闭 Socket, 释放相关资源
                //不显示窗口并释放资源
                f.setVisible(false);
                f.dispose();
                System.exit(0);                 //退出程序
            }
        });
        //为消息文本框中按下回车键增加事件处理代码

```

```

tfData.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        byte[] buf; //创建字节数组
        //取出文本框中的消息字符串，并将其转换成字节数组
        buf = e.getActionCommand().getBytes();
        try {
            //创建 DatagramPacket 类对象
            DatagramPacket dp = new DatagramPacket(buf, buf.length,
                InetAddress.getByName(tfIP.getText()), 3000);
            ds.send(dp); //发送对象 dp
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        //清空消息文本框内容
        ((TextField) e.getSource()).setText("");
    }
});
}
}

```

【代码解析】

在对象 `tfData` 的动作监听事件中，主要实现了消息发送功能。即首先获取文本框中的字符串，然后为获取到的字符串创建数据包对象 `dp`，最后再通过对象 `ds` 的 `send()` 方法把该数据包发送出去。完成消息发送功能后，还需要清空文本框中的内容。

对于消息的接收功能，需要在一个新的线程中完成。之所以这样，是因为在没有客户端连接的情况下，接收代码会处于阻塞状态，如果与发送消息代码在同一个线程中，那么就会影响该功能的运行。接收完发送过来的信息后，会把接收到的相应内容在对象 `lst` 中显示出来。

网络聊天室的界面布局如图 25.6 所示。

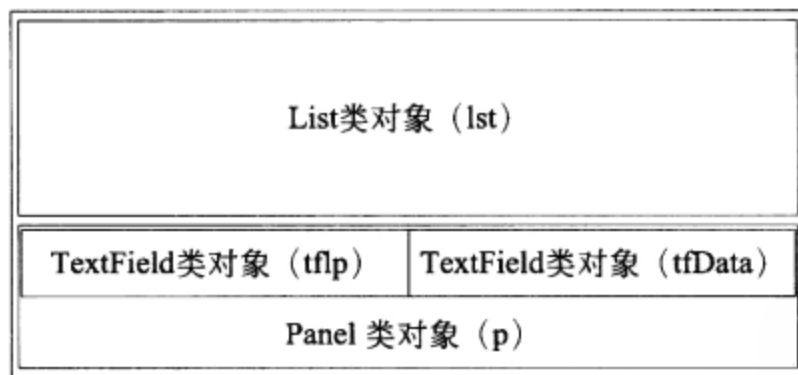


图 25.6 布局

注意：由于 `DatagramSocket` 的构造函数声明可能抛出异常，所以不能在声明时直接赋值，而是在构造函数中进行初始化。

25.3 知识点扩展——网络编程和 UDP 协议

随着时间的推移，网络已经成了工作、娱乐、生活和休闲的一部分，因此网络程序设计受到越来越多的重视。所谓网络程序设计，就是为用户提供网络服务的实用程序，例如网络通信、股票行情、新闻资讯等。另外网络程序设计也是游戏开发的必修课。

25.3.1 网络编程涉及的基本概念

要想开发网络应用程序,就必须对网络的基础知识有一定的了解,即首先必须了解 Java 的网络通信可以使用的协议——TCP、IP、UDP。同时还需要对网络应用程序的架构有一定的了解。

1. 网络协议——TCP/IP 协议

当许多计算机通过网络组建成一个互联系统后,就不能简单的叫做计算机互联,而应该叫做计算机网络。没有网络协议的计算机网络,就像是没有交通规则的城市道路系统,所以网络协议的出现主要是为了使相互连接的计算机之间能够更好地进行数据交换。随着时间的推移出现了许多网络协议,其中 TCP/IP 协议就是一个非常实用的网络协议。

TCP/IP 协议用来实现 Internet 数据传送的协议,其中 IP 为 Internet Protocol 的缩写。在 TCP/IP 协议中存在两种高级协议,即 TCP 协议和 UDP 协议。

2. 面向连接协议——TCP 协议

TCP 协议是面向连接的协议,可以实现两台计算机之间可靠无差错的数据传输。所谓面向连接协议,是指两台计算机在传输数据前,会先创建一个专属的双向虚拟连接。就像打电话一样,当两个人打电话时,电信局的交换机会创建一条只属于这两个人的虚拟连接,当打完电话后,这条专属连接就会消失。所谓双向,即就像打电话时,双方即能相互听到对方的声音,也能听见对方的回应。

3. 无连接协议——UDP 协议

UDP (User Datagram Protocol) 协议是无连接的协议,可以实现计算机间非持续连接的数据传输。所谓非连接协议,是指计算机间发送数据时,发送者会不管接收者是否已准备接收数据,发送完后也不管接收者是否接收到数据。就像发送短信一样,当发送短信时,其会由电信局的短信平台转发给所接收的用户,但是发送者并不知道这条短信能否被接收者收到。

4. IP 地址和端口号

既然 TCP/IP 协议用来实现 Internet 的数据传送,即实现 Internet 中计算机的互相通信,那么在 Internet 中如何标识计算机呢?在 TCP/IP 协议中,这个标识号就是 IP 地址。IP 地址在计算机中用 4 个字节(32 位二进制)表示,所以其也被称为 IPv4。为了便于记忆和使用,通常取每个字节的十进制数,并且每个字节之间用圆点隔开的形式表示,例如:

166. 111. 0. 255

随着 Internet 规模的不断扩大,32 位的 IP 地址不够使用,于是就出现了 128 位的 IP 地址,由于其由六个字节组成所以叫 IPv6。到目前为止,IPv6 还没有投入使用,Internet

上用的还是 IPv4。

在一台计算机上可以同时运行多个程序，但是 IP 地址只能保证数据送到某个计算机，但是不能确定把这些数据传送给计算机上的哪个应用程序。那么如何标识计算机上的应用程序呢？通过计算机组成原理的学习可以知道，在计算机上不能有两个使用同一端口的程序运行，所以端口被用来标识计算机里的应用程序。

5. 网络应用程序的架构——C/S

网络应用程序最常见的架构就是 C/S（Client-Server），以常见的万维网（www）为例，首先在服务器端会运行一个 Web Server 的程序，客户端会通过浏览器（IE 或 Netscape）应用程序连接到 Web Server，然后服务器端程序会根据客户端的请求把指定的内容返回给客户端，具体过程如图 25.7 所示。即服务器端（Server）的工作就是用来处理客户端（Client）的请求。

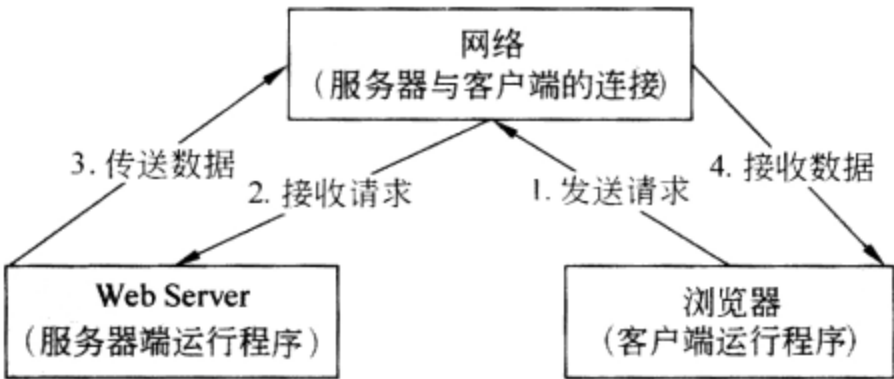


图 25.7 线程状态转换

25.3.2 套接字（Socket）机制

如果要想编写网络程序，就必须要了解 and 掌握 Java 语言中的 Socket 机制，所谓 Socket 机制就是网络驱动层提供给应用程序编程的接口和一种机制。

在具体运行网络程序时，会在网络应用程序中创建 Socket，其会通过一种绑定机制与网络驱动程序建立关系，通知自己所对应的 IP 地址和端口号。如果运行网络应用程序的计算机需要发送数据时，该应用程序首先会把数据送给 Socket，然后该 Socket 会把数据交给网络驱动程序，最后网络驱动程序把数据通过网络发送给目的计算机，具体过程如图 25.8 所示。如果运行网络应用程序的计算机需要接收数据时，计算机首先会从网络上收到与网络应用程序中 Socket 绑定的 IP 地址+端口号相关的数据，然后由网络驱动程序交给 Socket，最后该网络应用程序就会从该 Socket 中提取接收到的数据，具体过程如图 25.9 所示。

对于 Socket 机制也可以这样理解，Socket 其实就是网络应用程序的一个港口码头。如果要发送货物（数据），只需把装有货物的集装箱（数据包）放在港口码头（Socket），就算完成了货物发送。剩下的工作就由货运中间公司（计算机中的驱动程序）来处理。如果要接收货物（数据），只需要在港口码头（Socket）等待，然后把该港口码头（Socket）上的集装箱（数据包）取走。这是因为货运中间公司（计算机中的驱动程序）会自动送集装箱（数据包）到相应港口码头（Socket）。

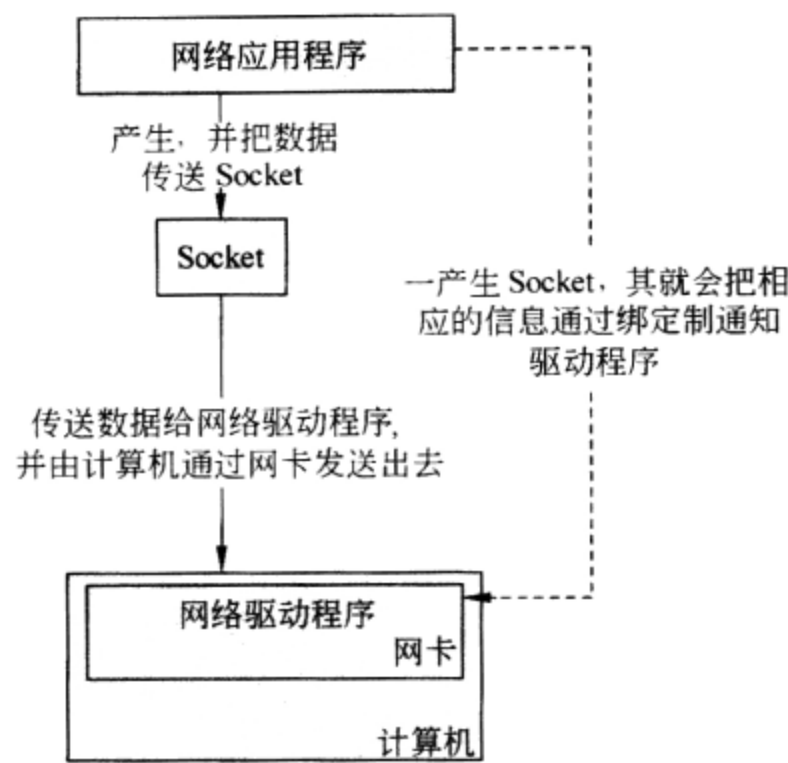


图 25.8 发送数据过程

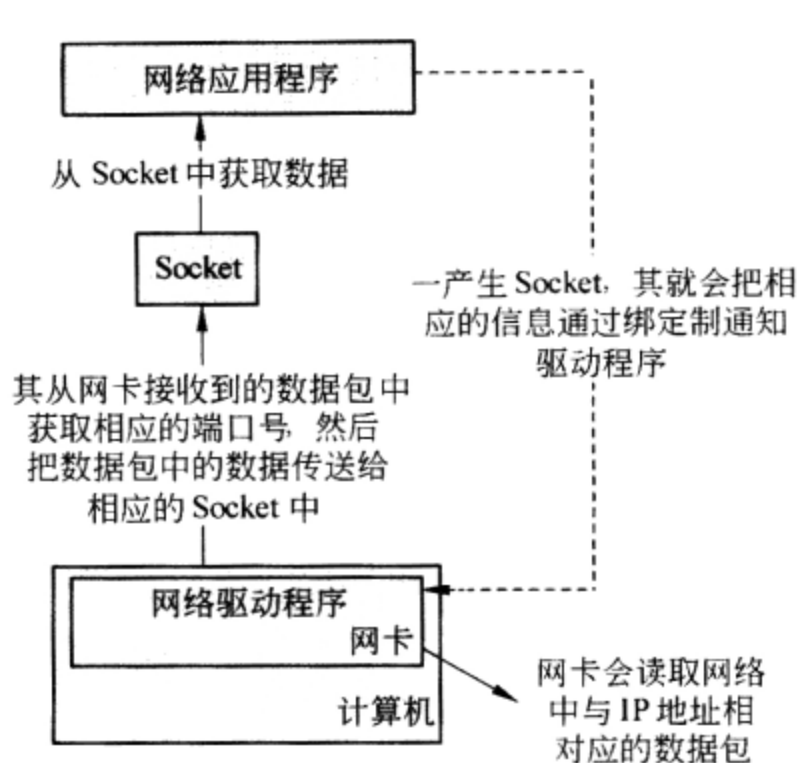


图 25.9 接收数据过程

25.3.3 UDP 协议类

在 Java 语言中分别为 UDP 和 TCP 两种通信协议提供了相关的编程类，这些类全部放在 java.net 包中，它们分别为与 UDP 相对应的 DatagramSocket；与 TCP 服务器端和客户端相对应的 ServerSocket 和 Socket。

为了更好地掌握 UDP 协议的类，下面通过一个具体的实例来讲解，该实例主要实现在一台计算机上发送和接收信息的功能。具体步骤如下。

(1) 创建一个实现信息发送功能的类 UDPSend.java，该类的具体内容如代码 25.2 所示。

代码 25.2 实现信息的发送：UDPSend.java

```
public class UDPSend {
    public static void main(String[] args) throws Exception {
        DatagramSocket ds = new DatagramSocket(); //创建DatagramSocket对象
        String str = "cjgong"; //所要发送的信息
        //创建 DatagramPacket 类对象
        DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(),
            InetAddress.getByName("172.168.1.100"), 3500);
        ds.send(dp); //实现发送信息
        ds.close(); //关闭对象 ds
    }
}
```

【代码解析】

在上述代码中，DatagramSocket 类对象（ds）主要用来创建 UDP 的 Socket，而 DatagramPacket 类对象（dp）主要用来包装数据，最后通过对象 ds 的 sent()方法把数据包（dp）发送出去。

⚠注意：IP 地址 172.168.1.100 为本地计算机的 IP 号，如果不知道本地计算机的 IP 地址，可以在 DOS 命令窗口通过命令 ipconfig 来查看，如图 25.10 所示。而端口号 3500 只要在 1024~65535 范围之内就可以。



图 25.10 测试 IP 地址

(2) 创建一个实现接收信息功能的类 UDPRecv.java, 该类的具体内容如代码 25.3 所示。

代码 25.3 实现信息的接收: UDPRecv.java

```
public class UDPRecv {
    public static void main(String[] args) throws Exception {
        DatagramSocket ds = new DatagramSocket(3500);
        //创建 DatagramSocket 对象

        byte[] buf = new byte[1024];
        //存储接收信息的数组变量
        DatagramPacket dp = new DatagramPacket(buf, 1024);
        //创建 DatagramPacket 对象

        ds.receive(dp);
        //接收信息
        //获取数据包中的数据信息
        String strRecv = new String(dp.getData(), 0, dp.getLength());
        String strRecv1 = "上述信息来自 IP 地址为:" + dp.getAddress().getHost-
        Address()
            + "; 端口号为:" + dp.getPort() + "的计算机。";
        //输出相应的信息
        System.out.println(strRecv);
        System.out.println("-----");
        System.out.println(strRecv1);
        ds.close();
    }
}
```

【代码解析】

在上述代码中, 首先创建了绑定端口号的 DatagramSocket 类对象 (ds) 和设置了存储空间大小的 DatagramPacket 类对象 (dp), 然后通过 ds 的 receive() 方法把发送给该 ds 对象的信息存储到数据包, 最后通过 dp 对象的 getData() 方法获取接收到的数据; dp.getAddress() 方法获取发送信息的计算机 IP 地址, dp.getPort() 方法获取发送信息的计算机上发送信息的应用程序的端口号, 同时把这些获取到的信息输入到屏幕里。

注意: 对象 ds 绑定的端口号必须为发送信息类 UDPSend 中对象 dp 里所设置的端口号。

(3) 查看运行效果。对于发送信息类 UDPSend, 其发送完信息后就会结束程序; 而对于接收信息类 UDPRecv, 虽然接收完信息也会结束程序, 但是在没有接收信息时就会处于阻塞状态, 一直到信息发送过来。因此需要先运行 UDPRecv 类, 使该程序处于阻塞状态,

然后再运行 UDPSend 类来发送信息，运行结果如图 25.11 所示。

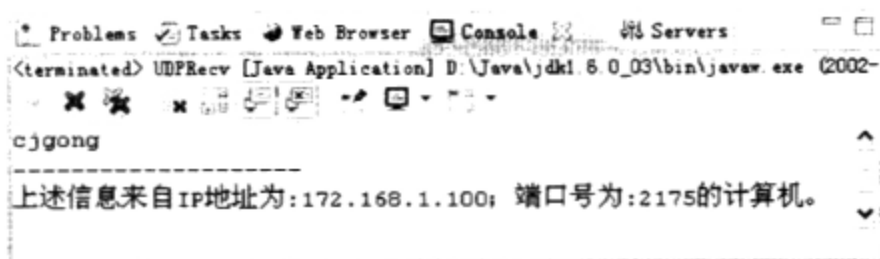


图 25.11 运行结果

通过 MyEclipse 的编译和运行功能虽然也可以查看到最后的运行结果,但是运行时的具体过程却不明显。这时可以通过在 DOS 命令窗口中查看运行结果,具体步骤如下。

(1) 需要进入 UDPSend 类和 UDPRecv 类编译后的文件里，具体命令如图 25.12 所示。



图 25.12 进入相应文件夹

在上述命令窗口中，命令 `cd` 用来实现进入相应的文件夹，而 `dir` 用来显示当前文件夹里的文件。

(2) 接着通过 `start` 命令再启动一个继承上一步骤打开命令窗口全部特性的窗口，最后的效果如图 25.13 所示。



图 25.13 打开继承窗口

(3) 先在继承窗口中启动接收信息的类 UDPRecv, 如图 25.14 所示, 然后在主窗口中启动发送信息的类 UDPSend, 如图 25.15 所示。当类 UDPSend 程序运行结束后, 继承窗口就会显示出发送的信息并同时结束接收信息类 UDPRecv, 如图 25.16 所示。

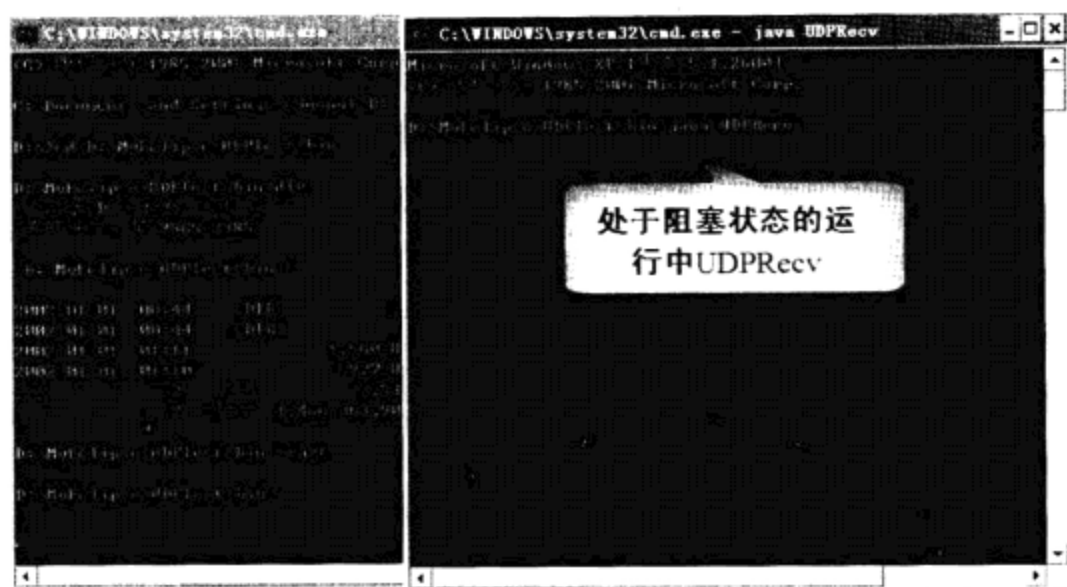


图 25.14 运行程序 UDPRecv

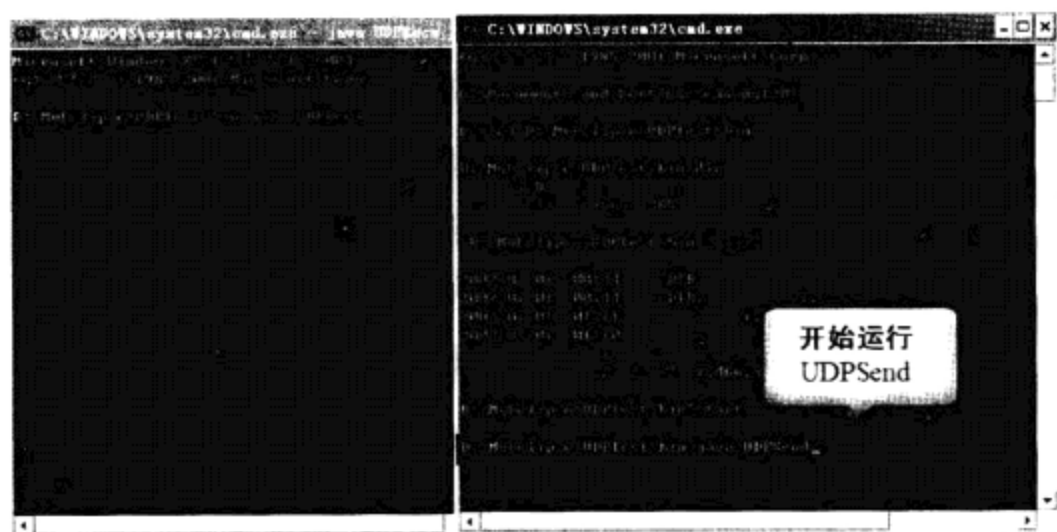


图 25.15 启动类 UDPSend



图 25.16 运行结束

注意: 由于 UDP 协议就像发送短信一样, 发送者将数据发送出去就算完成, 不管接收者是否接收或发送过程中数据是否丢失。所以当项目运行结束后, 如果再次运行发送信息类 UDPSend, 该程序不会因为接收信息类 UDPRecv 没有接收信息而发生错误, 具体过程如图 25.17 所示。

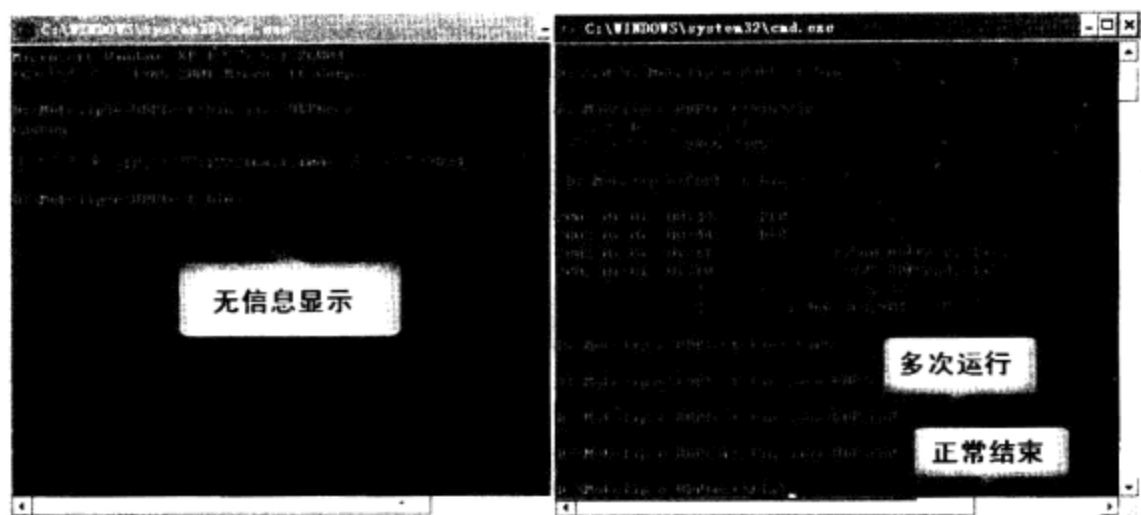


图 25.17 运行过程

25.3.4 TCP 协议类

在 Java 语言中的 java.net 包中, 除了 UDP 协议的类外, 还存在 TCP 服务器端和客户端相对应的 ServerSocket 和 Socket。为了更好地掌握 TCP 协议的类, 下面通过一些具体的实例来讲解, 这些实例主要用来实现服务器端和客户端之间的通信, 具体步骤如下:

(1) 首先创建一个运行在服务器端的服务器类 TCPServer1.java, 该类的具体内容如代码 25.4 所示。

代码 25.4 服务器端类: TCPServer1.java

```
public class TCPServer1{
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(8002);
            //创建 ServerSocket 类对象
            Socket s = ss.accept();
            //等待客户端的连接
            //创建输入和输出流对象 ips 和 ops
            InputStream ips = s.getInputStream();
            OutputStream ops = s.getOutputStream();
            ops.write("cjgong!".getBytes()); //向客户端输出信息
            //创建和设置获取客户端信息的输入流对象
            byte[] buf = new byte[1024]; //字节数组变量对象 buf
            int len = ips.read(buf); //读取的字节数变量
            System.out.println(new String(buf, 0, len));
            //输出客户端发出的信息
            //关闭相应的对象
            ips.close();
            ops.close();
            s.close();
            ss.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

【代码解析】

在上述代码中, 首先创建了一个绑定 8002 端口的 ServerSocket1 对象, 该对象主要用来实现监听客户端的请求。当监听到客户端的连接请求后, ServerSocket1 对象就会创建出

Socket 对象。为了能够实现服务器端和客户端之间信息交互,通过 Socket 对象获取输入流和输出流对象。对于输出流对象 ops,主要用来实现服务器端的发送信息;对于输入流对象 ips,主要用来实现获取客户端的发送信息。

⚠注意:由于输入流和输出流对象 ips 和 ops 都是依附于 Socket 类对象 s,而对象 s 又依附于 ServerSocket 类对象 ss,因此,首先需要先关闭输入流和输出流对象 ips 和 ops,然后再关闭对象 s,最后才关闭对象 ss。

(2) 如果想测试服务器端类 TCPServer1,可以通过 Windows 系统中的 Telnet 工具来代替客户端程序。首先运行类 TCPServer1,然后通过命令 telnet 172.168.1.100 8002 连接服务器程序,这时该命令窗口就会显示服务器端发出的信息,具体过程如图 25.18 所示。如果想通过 Telnet 工具向服务器发送信息,可以在连接服务器成员的命令窗口中直接输入想发送的信息,这时服务器运行的输出窗口就会显示接收到的信息,具体过程如图 25.19 所示。



图 25.18 连接服务器

⚠注意: TCPServer1 程序只能接收 Telnet 程序发送的一个字符。

(3) Telnet 客户端程序有一个特点,只要有输入就会发送,而不管有没有按下 Enter 键。所以如果输入 cjgong 字符串,服务器端程序 TCPServer1 只会接收到字符 c 而不是字符串 cjogng,这是因为 TCPServer1 类每接收到一个发送就会显示出该发送的内容并退出程序。为了使程序 TCPServer1 能够接收到字符串 abc,修改其内容如代码 25.5 所示。

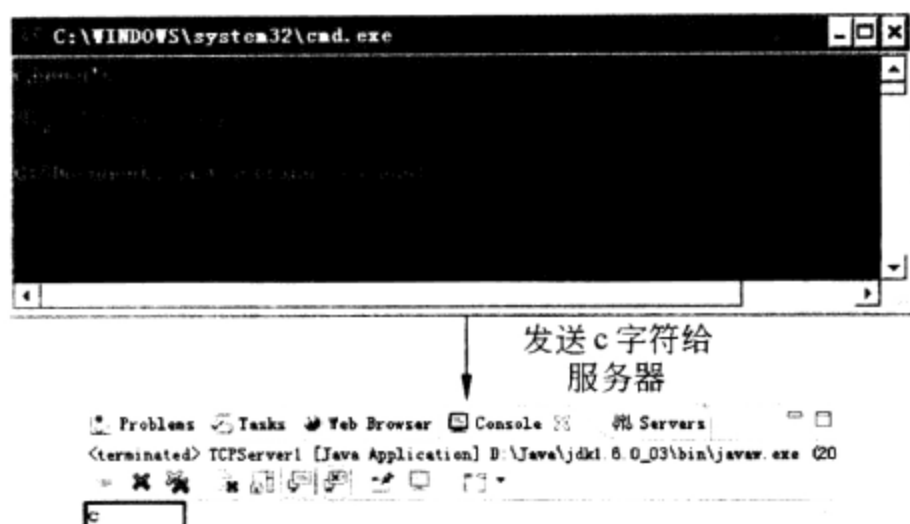


图 25.19 发送信息给服务器

代码 25.5 服务器端类: TCPServer2.java

```
public class TCPServer2 {
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(8050);
            Socket s = ss.accept();
            InputStream ips = s.getInputStream();
            OutputStream ops = s.getOutputStream();
            ops.write("cjgong!".getBytes());
            //创建过滤器对象 br
            BufferedReader br = new BufferedReader(new InputStreamReader
                (ips));
            System.out.println(br.readLine());
            br.close();
            ops.close();
            s.close();
            ss.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

通过 Telnet 工具再次连接服务器程序 TCPServer2, 其具体运行过程如图 25.20 所示, 达到预期要求。

【代码解析】

在上述代码中, 由于利用了过滤类 `BufferedReader` 的 `readLine()` 方法, 所以不管客户端程序是逐个发送一行中的所有字符, 还是一次发一行, 该程序的处理结果都是一样的。

注意: 关闭包装类, 会自动关闭包装类中所包装的底层类。所以不需要调用 `ips.close()` 方法。

(4) 服务器端程序 TCPServer2 虽然可以接收字符串, 但是却只能够接受一个客户端的连接请求。为了使程序能够接受多个客户端的多次连接请求, 首先创建一个实现数据交换功能的类, 具体内容如代码 25.6 所示。

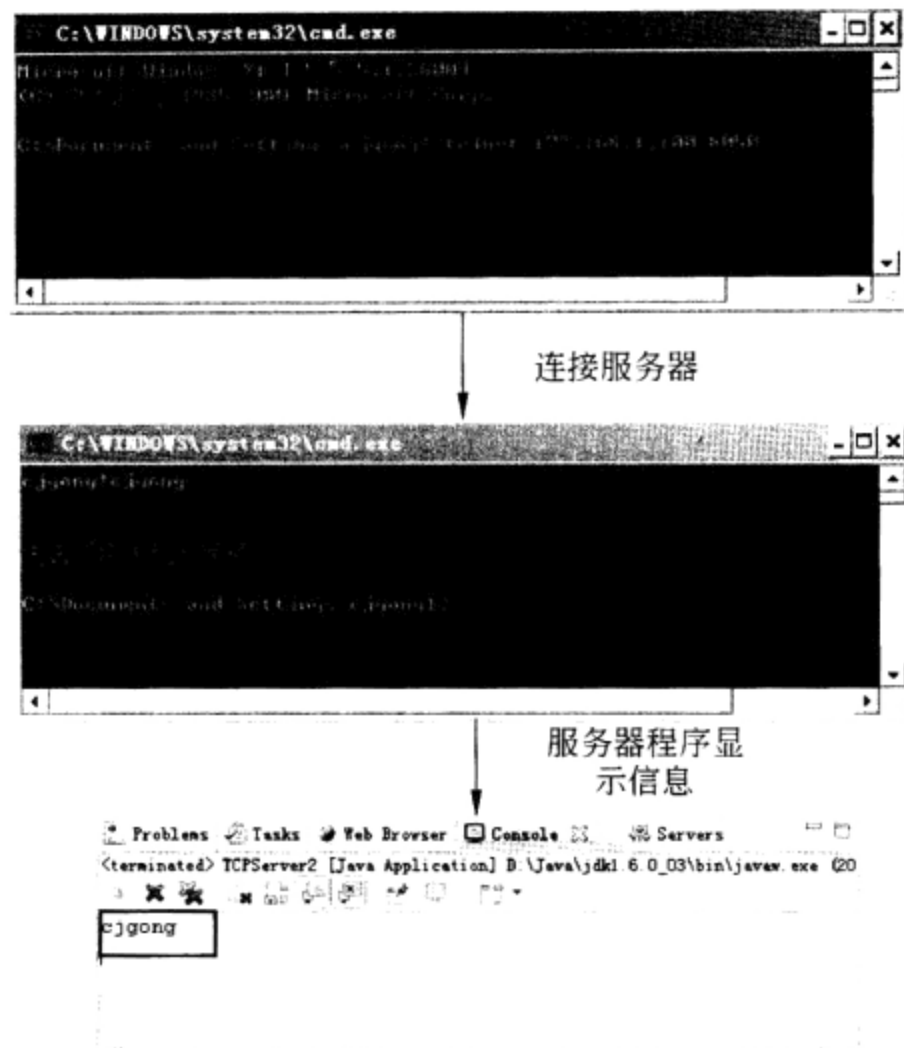


图 25.20 运行过程

代码 25.6 实现数据交换的服务: Servicer.java

```

class Servicer implements Runnable {
    Socket s; //创建一个 Socket 类成员
    public Servicer(Socket s) { //构造函数
        this.s = s;
    }
    public void run() { //执行任务
        try {
            //获取输入流和输出流
            InputStream ips = s.getInputStream();
            OutputStream ops = s.getOutputStream();
            //过滤类
            //可以方便读取一行字符串
            BufferedReader br = new BufferedReader(new InputStreamReader(
                ips));
            //可以方便字符串的写入
            DataOutputStream dos = new DataOutputStream(ops);
            while (true) { //通过循环实现多次数据相互交互
                String strWord = br.readLine(); //客户端发送的内容
                System.out.println("客户端发送的字符串为: " + strWord + "; 其长度
                    为: " + strWord.length());
                if (strWord.equalsIgnoreCase("quit")) //判断是否退出
                    break; //退出循环
                //反转后的字符串
                String strEcho = (new StringBuffer(strWord).reverse())
                    .toString();
                //输出相应的信息
                dos.writeBytes(strWord + "---->" + strEcho

```

```

        + System.getProperty("line.separator"));
    }
    br.close();
    dos.close();
    s.close();
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

【代码解析】

- ❑ 实现数据交换的代码之所以要放在独立的线程中，是为了保证可以实现多个客户端的连接，即如果该代码在运行时，其他的程序代码就得不到调用，同理 `accept()` 方法也得不到调用，新的连接也无法创建。所以需要有一个单独的类来实现服务器端与客户端的对话功能。
- ❑ 实现数据交换的代码之所以要放在循环中，是为了保证在一次连接中可以实现多次的相互交换。循环中的代码主要实现客户端每向服务器发送一个字符串，服务器会将这个字符串中的所有字符方向排列好传送给客户端，直到客户端向服务器端发送 `quit` 命令，才结束两者间的对话。
- ❑ 为了简化程序的编写，对于读取流对象，过滤类 `BufferedReader` 可以方便地从底层字节输入流中以整行的形式读取一个字符串；对于写入流对象，过滤类 `DataOutputStream` 可以方便地将一个字符串以字节数组的形式写入底层字节输出流中。
- ❑ 代码 `System.getProperty("line.separator")` 可以根据不同的操作系统，返回相应的换行符，如果是 Windows 系统可以用 `"\r\n"` 代替；如果是 Linux 系统可以用 `"\n"` 代替。发送给客户端的字符串中之所以要加上换行符，是为了方便客户端的处理，服务器端最好以行的形式发送。

接着创建类 `TcpServer3`，该类用来实现服务器端和客户端的信息相互交互，具体内容如代码 25.7 所示。

代码 25.7 服务器端类: `TcpServer3.java`

```

public class TcpServer3 {
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(8052);
                                //创建 ServerSocket 类对象
            while (true) {      //通过循环不停地监听客户端请求
                Socket s = ss.accept();    //获取 Socket 类对象
                Servicer servicer = new Servicer(s); //创建 Servicer 类对象
                new Thread(servicer).start();    //启动对象 servicer
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

通过 `Telnet` 工具再次连接服务器程序 `TCPServer3`，可以发现该类不仅实现了在一个 `Telnet` 命令窗口中多次发送信息，而且还实现了多个 `Telnet` 命令窗口连接服务器端程序的功能，具体过程如图 25.21 所示，达到了预期要求。



图 25.21 运行过程

【代码解析】

对于 `ServerSocket` 对象, `accept()` 方法用于接收连接, 即创建一个连接调用一次该方法。为了实现接收多个连接, 所以该方法调用的语句需要放在循环中。

假如在连接服务器的窗口中, 首先输入 `cjpgong1` 信息, 这时服务器端程序的回送信息 `lgnogjc` 和输出的信息“客户端发送的字符串为: `cjpgong1`; 其长度为: 7”, 完全正确。可是在接着输入 `cjpgong2` 时, 却发现最后一个字符 `2` 错敲成了 `1`, 于是用 `BackSpace` 键将 `1` 删除, 然后重新输入 `2`, 最后回车发送给服务器。这时发现服务器回送的信息为 `lgnogjc`, 而显示的信息为“客户端发送的字符串为: `cjpgong1_2`; 其长度为: 9”, 具体过程如图 25.22 所示。

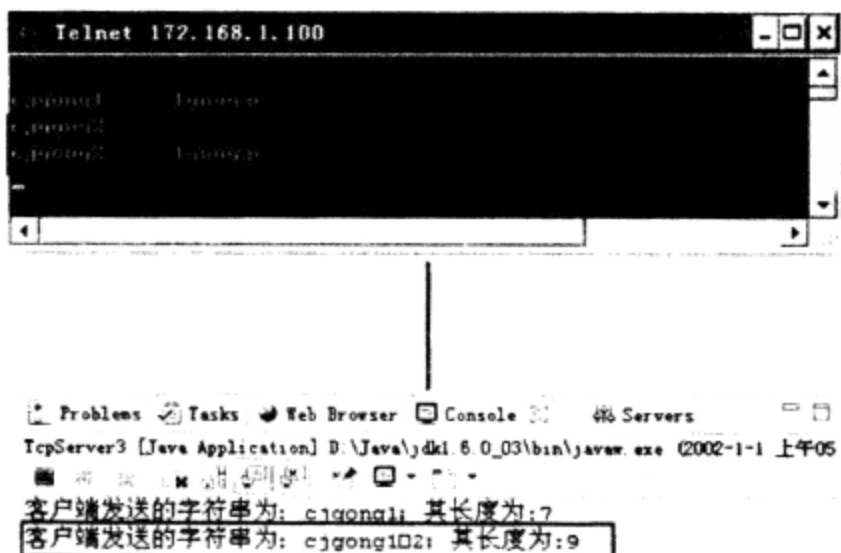


图 25.22 运行过程

之所以会这样,可以理解为当发送信息 `cjgong2` 时,服务器程序实际接收到了 9 个字符,其内容为 `cjgong1{BackSpace}2`,由于字符 `{BackSpace}` 的特殊显示效果,所以这 9 个字符在窗口上的显示结果就是 `cjgong2`。同理服务器的回送的信息为 `2{BackSpace}1gnogjc`,在窗口上的显示结果就是 `1gnogjc`。

25.3.5 TCP 协议客户端类

在 Java 语言中的 `java.net` 包中,除了 TCP 服务器端的类 `ServerSocket` 外,还有客户端的类 `Socket`。虽然在 Windows 系统里可以通过命令 `Telnet` 来模拟客户端程序测试服务器程序,但是在具体运行网络应用程序时却必须通过客户端程序连接服务器端程序。

为了更好地掌握 TCP 协议的客户端类 `Socket`,下面通过一个具体的实例来讲解,该实例主要用来实现服务器端和客户端之间的通信,具体步骤如下。

创建一个运行在服务器端的服务器类 `TCPServer1.java`,该类的具体内容如代码 25.8 所示。

代码 25.8 客户端类: `TcpClient.java`

```
public class TcpClient {
    public static void main(String[] args) {
        try {
            // Socket s=new Socket(InetAddress.getByName("192.168.0.213"),
            // 8001);
            //判断传入参数的个数
            if (args.length < 2) {
                System.out.println("请输入服务器端的 IP 地址和端口号");
                return; //退出程序
            }
            //创建 Socket 类对象
            Socket s = new Socket(InetAddress.getByName(args[0]), Integer
                .parseInt(args[1]));
            InputStream ips = s.getInputStream(); //获取网络输入流
            OutputStream ops = s.getOutputStream(); //获取网络输出流
            //过滤键盘输入流
            BufferedReader brKey = new BufferedReader(new InputStreamReader(
                System.in));
            //过滤网络输出流
            DataOutputStream dos = new DataOutputStream(ops);
            //过滤网络输入流
            BufferedReader brNet = new BufferedReader(
                new InputStreamReader(ips));
            while (true) {
                String strWord = brKey.readLine(); //获取键盘输入
                //发送信息给服务程序
                dos.writeBytes(strWord + System.getProperty("line.
                    separator"));
                if (strWord.equalsIgnoreCase("quit"))
                    break; //退出循环
                else
                    System.out.println(brNet.readLine()); //输出读取信息
            }
            //关闭各种流对象
        }
    }
}
```


```

        dos.close();
        brNet.close();
        brKey.close();
        s.close(); //关闭对象 s
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

【代码解析】

在上述代码中，首先创建了一个绑定 8002 端口的 `ServerSocket1` 对象，该对象主要用来实现监听客户端的请求。当监听到客户端的连接请求后，`ServerSocket1` 对象就会创建出 `Socket` 对象。为了能够实现服务器端和客户端之间的信息交互，通过 `Socket` 对象获取输入流和输出流对象。对于输出流对象 `ops`，主要用来实现服务器端的发送信息；对于输入流对象 `ips`，主要用来实现获取客户端的发送信息。

 **注意：**由于输入流和输出流对象 `ips` 和 `ops` 都是依附于 `Socket` 类对象 `s`，而对象 `s` 依附于 `ServerSocket` 类对象 `ss`，因此，首先需要先关闭输入流和输出流对象 `ips` 和 `ops`，然后再关闭对象 `s`，最后才关闭对象 `ss`。

25.4 小 结

本章主要通过 Java 语言中的网络编程技术和多线程技术实现网络聊天室项目，在具体实现该项目时，主要利用了 Java 语言中的 `Socket` 机制和 `UDP` 协议。

为了让读者能够更好地掌握 Java 语言的网络编程技术，在本章的最后还详细介绍了网络编程涉及的基本概念、`Socket`、`UDP` 和 `TCP`。

第 26 章 FTP 服务器客户端

(FtpClient+I/O 处理)

FTP (File Transfer Protocol) 是 Internet 上用来实现文件传送的协议, 其是在 Internet 上实现相传文件的标准, 规定了 Internet 上如何实现文件传送。通过 FTP 协议, 可以与 Internet 上的 FTP 服务器进行文件的上传和下载。本章将实现一个简单易用的 FTP 服务器客户端, 还会详细介绍类 FtpClient 的各种成员变量和各种方法。

本章的学习目标如下:

- ❑ 掌握 FTP 服务器客户端项目;
- ❑ 掌握类 FtpClient 的用法。

26.1 FTP 服务器客户端原理

在众多的网络应用中, 虽然存在许多 FTP 服务器客户端软件, 但是由于这些软件界面复杂、操作烦琐, 所以不被大多数用户接受。本章将通过 Java 语言中的类 FtpClient 来实现一个简单易用的 FTP 服务器客户端软件。

26.1.1 项目结构框架分析

对于 FTP 服务器客户端项目, 首先创建工具类, 即通过类 FtpClient 实现操作 FTP 服务器的各种方法, 然后通过调用工具类中的相应方法来实现文件上传和下载的功能。FTP 服务器客户端项目目录如图 26.1 所示, 各目录功能如下。

- ❑ FtpTools: 实现 FTP 服务器各种操作的类。
- ❑ FtpUpFile: 实现文件上传的类。
- ❑ FtpDownFile: 实现文件下载的类。

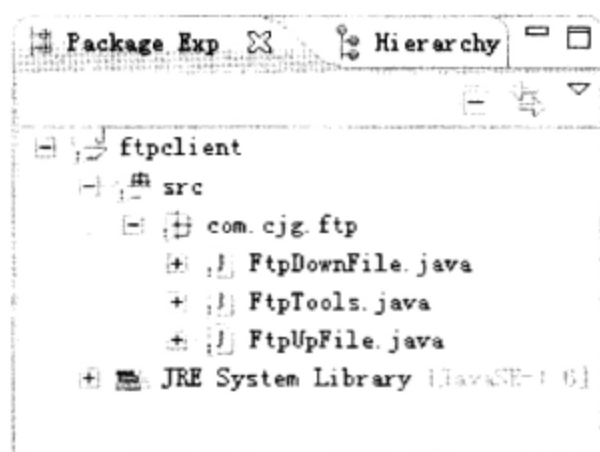


图 26.1 项目目录

26.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括本地和服务器文件夹初始化、文件上传功能和文件下载功能。

1. 本地和服务器文件夹

在运行程序之前，需要检查要上传的文件夹和所上传的 FTP 服务器目录。对于所要上传的文件夹和文件如图 26.2 所示，即在 C:\wwwroot 文件夹里存在 1.txt 和 2.txt 两个文件。对于 FTP 服务器目录如图 26.3 所示，该服务器的网络地址为 98.126.133.34。

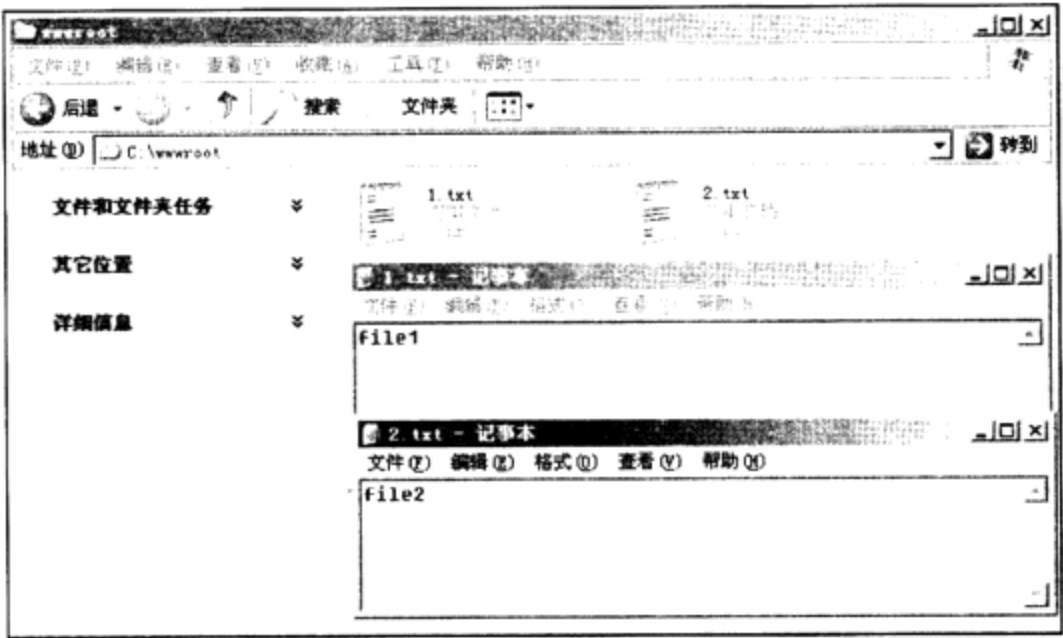


图 26.2 本地文件夹和文件

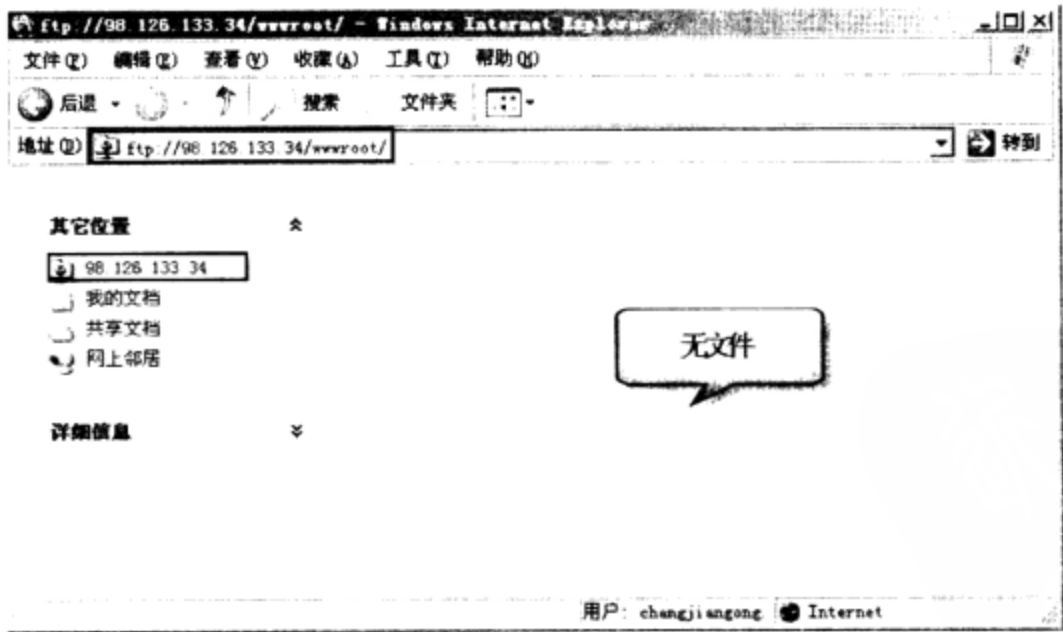


图 26.3 服务器 wwwroot 文件夹

2. 文件上传功能

当运行实现文件上传方法的类时，输出窗口如图 26.4 所示，这时服务器端的文件夹内容就会发生变化，具体内容如图 26.5 所示。



图 26.4 输出窗口

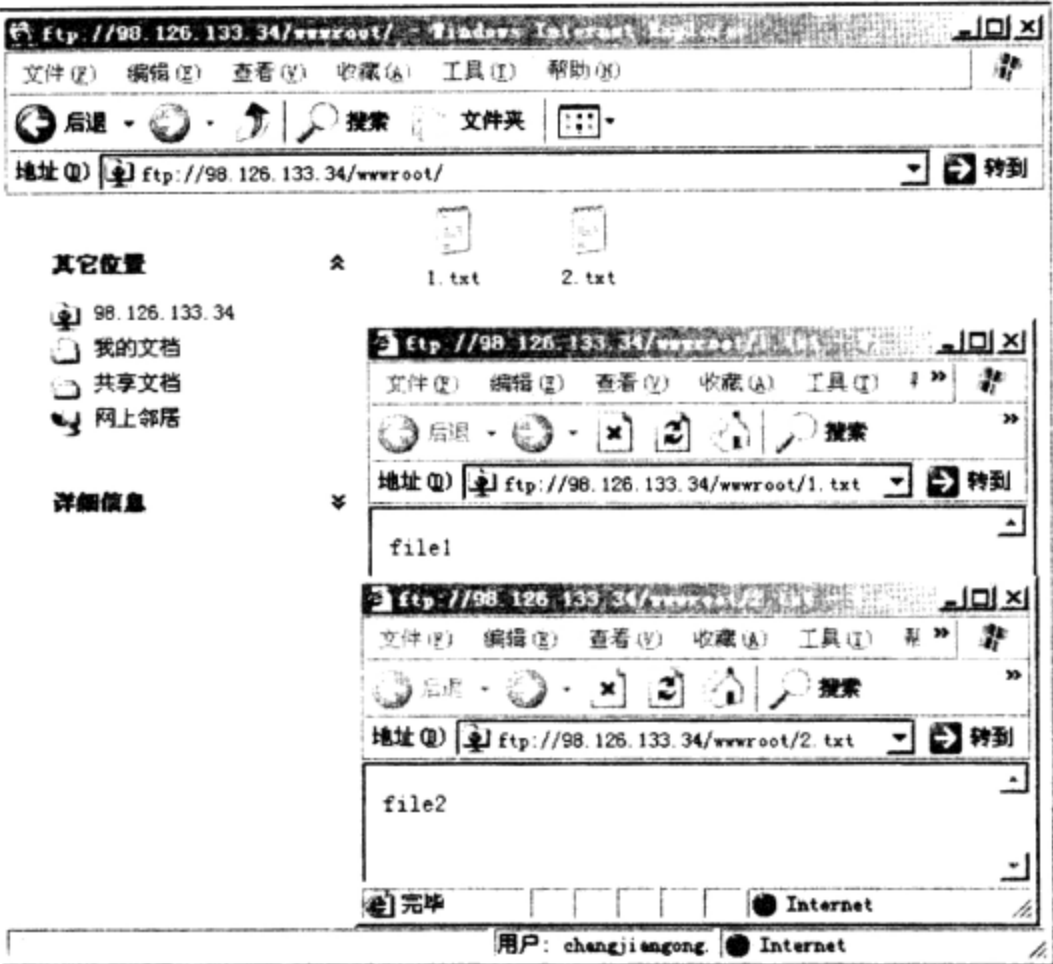


图 26.5 服务器的 wwwroot 文件夹内容

3. 文件下载功能

当运行实现文件下载方法的类时，输出窗口如图 26.6 所示，这时客户端的文件夹内容就会发生变化，具体内容如图 26.7 所示。



图 26.6 输出窗口

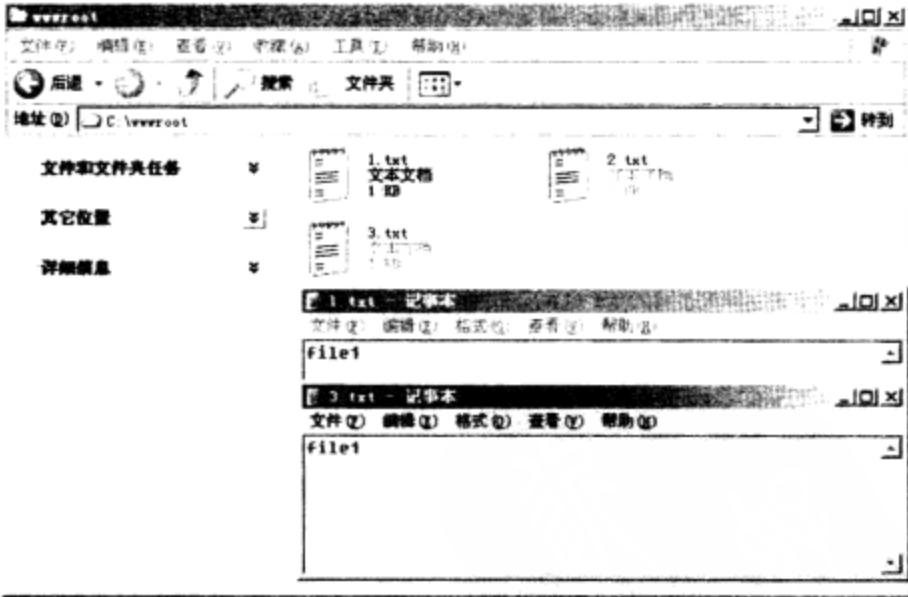


图 26.7 客户端 C 盘中的 wwwroot 文件夹内容

26.2 FTP 服务器客户端的实现过程

本章通过 FtpClient 类和 I/O 的相关知识来实现 FTP 服务器项目，在该项目中存在 3 个类，即工具类 FtpTools、实现文件上传功能 FtpUpFile 类和文件下载功能的 FtpDownFile 类，

这 3 个类的关系如图 26.8 所示。

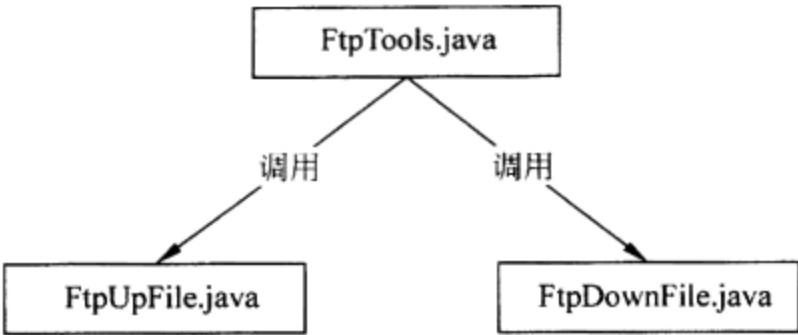


图 26.8 程序关系

26.2.1 FTP 服务器操作的工具类

为了方便设计，本节专门编写了一个名为 FtpTools 的工具类，该类主用实现操作 FTP 服务器的各种方法。FtpTools 类的具体内容如代码 26.1 所示，该类的 UML 如图 26.9 所示。

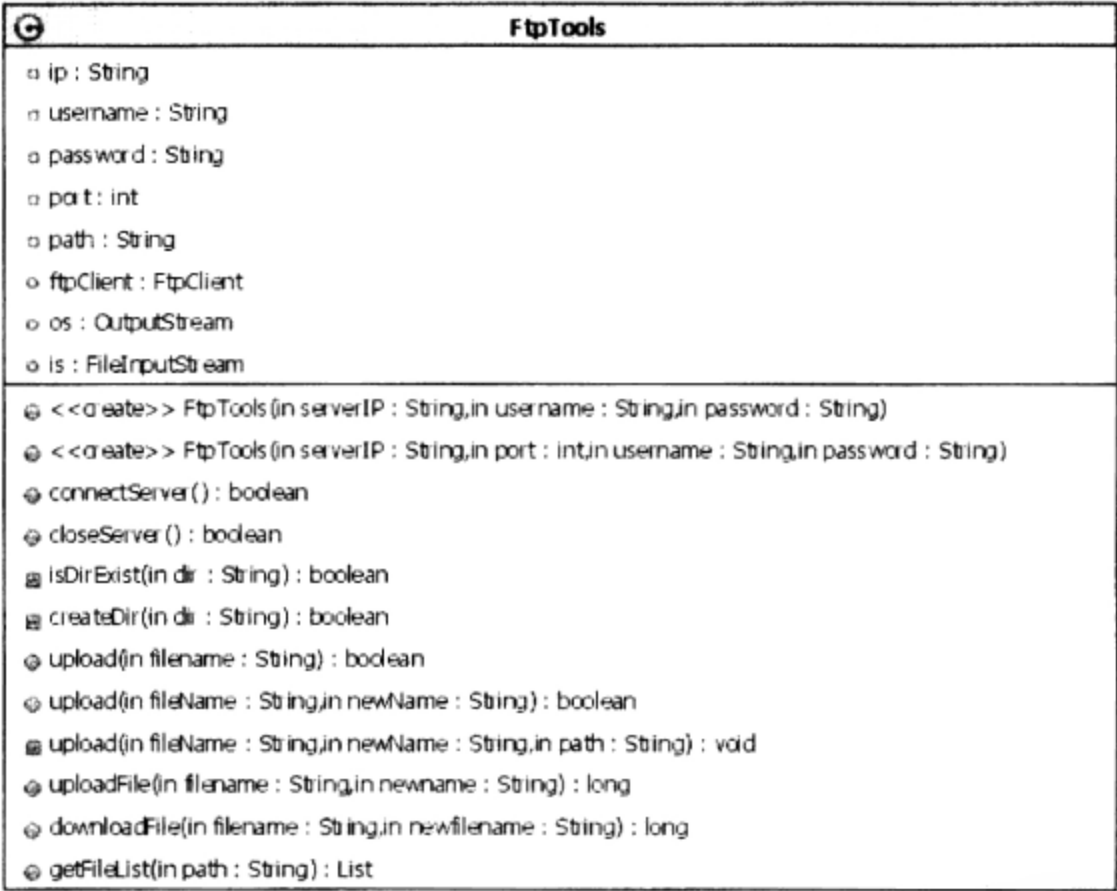


图 26.9 工具类 UML 图

代码 26.1 FTP 服务器操作的工具类：FtpTools.java

```
public class FtpTools {
    //创建字段
    private String ip = "";
    private String username = "";
    private String password = "";
    private int port = -1;
    //FTP 服务器下主目录的子目录路径成员变量
    private String path = "";
    FtpClient ftpClient = null;
    OutputStream os = null;

    //服务器端地址的成员变量
    //用户名成员变量
    //密码成员变量
    //端口号成员变量

    //路径成员变量
    //输出流成员变量
}
```



```

FileInputStream is = null; //输入流成员变量
//构造函数
public FtpTools(String serverIP, String username, String password) {
    this.ip = serverIP; //初始化 IP
    this.username = username; //初始化 username
    this.password = password; //初始化 password
}
//构造函数
public FtpTools(String serverIP, int port, String username, String
password) {
    this.ip = serverIP; //初始化 IP
    this.username = username; //初始化 username
    this.password = password; //初始化 password
    this.port = port; //初始化 port
}
//实现连接 FTP 服务器功能
public boolean connectServer() { //连接服务器方法
    ftpClient = new FtpClient(); //初始化变量 ftpClient
    try {
        if (this.port != -1) { //当端口号不为-1 时
            //通过新端口号打开连接
            ftpClient.openServer(this.ip, this.port);
        } else {
            ftpClient.openServer(this.ip); //以默认端口号打开连接
        }
        //通过用户名和密码登录 FTP 服务器
        ftpClient.login(this.username, this.password);
        if (this.path.length() != 0) { //判断 path 的值
            ftpClient.cd(this.path); // path 是 FTP 服务下主目录的子目录
        }
        ftpClient.binary(); //用二进制实现上传、下载
        //输出相应信息
        System.out.println("已登录到\"" + ftpClient.pwd() + "\"目录");
        return true;
    } catch (IOException e) {
        e.printStackTrace();
        return false;
    }
}
//断开与 FTP 服务器的连接
public boolean closeServer() { //断开服务器方法
    try {
        if (is != null) { //关闭输入流
            is.close();
        }
        if (os != null) { //关闭输出流
            os.close();
        }
        if (ftpClient != null) { //关闭服务器
            ftpClient.closeServer();
        }
        //输出相应信息
        System.out.println("已从服务器断开");
        return true;
    } catch (IOException e) {
        e.printStackTrace();
        return false;
    }
}

```

```

    }
}
//实现检查文件夹在当前目录下是否存在
private boolean isDirExist(String dir) { //判断当前目录下是否存在文件夹
    String pwd = ""; //创建空字符串变量 pwd
    try {
        pwd = ftpClient.pwd(); //初始化变量 pwd
        ftpClient.cd(dir);
        ftpClient.cd(pwd);
    } catch (Exception e) {
        return false;
    }
    return true;
}
//在当前目录下创建文件夹
private boolean createDir(String dir) {
... //省略部分代码
}
//取得相对于当前连接目录的某个目录下的所有文件列表
public List getFileList(String path) {
... //省略部分代码
}
//实现上传同名文件方法
public boolean upload(String filename) {
... //省略部分代码
}
//实现上传同名文件方法
public boolean upload(String fileName, String newName) {
... //省略部分代码
}
//实现上传文件
private void upload(String fileName, String newName, String path)
    throws Exception {
    //创建变量 savefilename
    String savefilename = new String(fileName.getBytes("ISO-8859-1"),
        "GBK");
    File file_in = new File(savefilename); //打开本地待上传的文件
    //判断文件是否存在
...
    ftpClient.cd(path);
}
//实现文件上传功能
public long uploadFile(String filename, String newname) throws Exception {
    long result = 0; //创建返回值变量
...
    return result;
}
//实现文件下载功能
public long downloadFile(String filename, String newfilename) {
    //创建各种成员变量
    long result = 0;
... //省略部分代码
    return result;
}
}

```

【代码解析】

在上述代码中主要创建了操作服务器的相关方法, 分别是实现连接和断开 FTP 服务器的方法 `connectServer()` 和 `closeServer()`、实现检查文件夹是否存在的方法 `isDirExist()`、实现创建文件夹的方法 `isDirExist()`、实现文件上传和下载的方法 `uploadFile()` 和 `downloadFile()`。

26.2.2 实现文件上传的类

`FtpUpFile.java` 类用来实现上传文件的功能, 该类的具体内容如代码 26.2 所示, 该类的 UML 如图 26.10 所示。

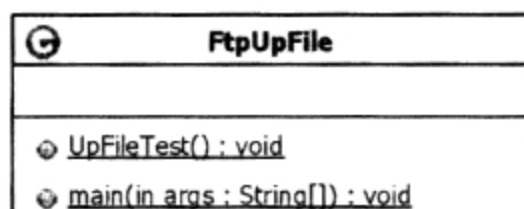


图 26.10 文件上传类 UML 图

代码 26.2 实现文件上传: `FtpUpFile.java`

```

public class FtpUpFile {
    public static void UpFileTest() {
        //创建 FtpTools 类对象
        FtpTools ftp = new FtpTools("98.126.133.34", "changjiangong.com",
            "Y58n0WZ3cX");
        ftp.connectServer(); //实现连接服务器
        boolean result = ftp.upload("C:/wwwroot", "wwwroot"); //调用 upload() 方法
        //输出相应信息
        System.out.println(result ? "上传成功!" : "上传失败!");
        List list = ftp.getFileList("wwwroot"); //获取集合 list 镀锡
        //通过循环输出相应文件
        for (int i = 0; i < list.size(); i++) {
            String name = list.get(i).toString();
            System.out.println(name);
        }
        ftp.closeServer(); //关闭服务器
    }
    public static void main(String[] args) {
        UpFileTest(); //调用 UpFileTest() 方法
    }
}
  
```

【代码解析】

在上述代码中为了实现上传文件的功能, 创建了一个实现文件上传功能的 `UpFileTest()` 方法。在该方法中, 首先通过 `connectServer()` 方法连接服务器, 然后通过 `upload()` 方法把相应文件上传到服务器上, 最后输出上传的文件名并关闭连接服务器。

26.2.3 实现文件下载的类

`FtpDownFile.java` 类用来实现下载文件的功能, 该类的具体内容如代码 26.3 所示, 该

类的 UML 如图 26.11 所示。

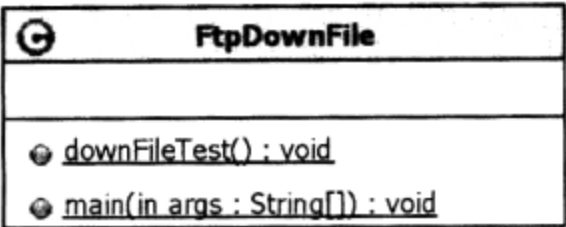


图 26.11 文件下载类 UML 图

代码 26.3 实现文件下载: FtpDownFile.java

```
public class FtpDownFile {
    public static void downFileTest() { //实现文件下载方法
        //创建 FtpTools 类对象 ftp
        FtpTools ftp = new FtpTools("98.126.133.34", "changjiangong.com",
            "Y58n0WZ3cX");
        ftp.connectServer(); //连接服务器
        //实现下载文件功能
        long result = ftp.downloadFile("wwwroot/1.txt", "C:/wwwroot/3.
            txt");
        System.out.println("下载成功"); //输出相应信息
        List list = ftp.getFileList("wwwroot"); //获取文件目录
        //通过循环输出文件名字
        for (int i = 0; i < list.size(); i++) {
            String name = list.get(i).toString();
            System.out.println(name); //输出文件名字
        }
        ftp.closeServer(); //关闭服务器
    }
    public static void main(String[] args) {
        downFileTest(); //调用 downFileTest() 方法
    }
}
```

【代码解析】

在上述代码中为了实现下载文件功能，创建了一个实现文件下载功能的 downFileTest() 方法。在该方法中，首先通过 connectServer()方法连接服务器，然后通过 downloadFile ()方法把服务器上的文件下载到本地，最后输出下载的文件名并关闭连接服务器。

26.3 知识点扩展——FtpClient 类的相关知识

在具体编写 FTP 服务器客户端项目时，主要使用了 Java 类库中的 sun.net.ftp.FtpClient 类。FtpClient 类封装了 FTP 协议的相关指令和实现细节，并提供了一系列方法来操作 FTP 服务器。本节将详细介绍 FtpClient 类的相关知识。

26.3.1 实现 FTP 服务器相关操作类

FtpClient 类存放在 sun.net.ftp 包里，该类主要用于实现建立 FTP 连接等功能。利用

FtpClient 类的方法, 编程人员不仅可以远程登录到 FTP 服务器, 而且还可以实现各种操作功能, 即列举该服务器上的目录、设置传输协议以及传送文件。

为了能够涵盖 FTP 所有的功能, FtpClient 类设置了各种成员变量和方法。下面将详细介绍该类的成员变量和方法, 具体步骤如下。

1. 成员变量

FtpClient 类中有 3 个比较重要的成员变量, 分别如下所示。

(1) useFtpProxy: 该变量用于表明 FTP 传输过程中是否使用了一个代理, 因此, 它实际上是一个标记, 此标记若为 true, 表明使用了一个代理主机。

(2) public static String ftpProxyHost: 该变量只有在变量 useFtpProxy 为 true 时才有效, 用于保存代理主机名。

(3) public static int ftpProxyPort: 该变量只有在变量 useFtpProxy 为 true 时才有效, 用于保存代理主机的端口地址。

2. 构造函数

FtpClient 中有 3 种不同形式的构造函数, 分别如下所示。

(1) public FtpClient(String hostname,int port): 该构造函数利用给出的主机名和端口号建立一条 FTP 连接。

(2) public FtpClient(String hostname): 该构造函数利用给出的主机名建立一条 FTP 连接, 使用默认端口号。

(3) FtpClient(): 该构造函数将创建一个 FtpClient 类, 但不建立 FTP 连接。

3. 实现连接服务器功能函数

当建立了 FtpClient 类后, 就可以用该类的方法打开与 FTP 服务器的连接。FtpClient 类提供了如下两个可用于打开与 FTP 服务器之间的连接的方法。

(1) public void openServer(String hostname): 该方法用于建立一条与指定主机上的 FTP 服务器的连接, 使用默认端口号。

(2) public void openServer(String host,int port): 该方法用于建立一条与指定主机、指定端口上的 FTP 服务器的连接。

4. 实现登录服务器功能函数

当通过 openServer()方法打开连接之后, 就可以登录 FTP 服务器, 这时需要利用到下面的方法。

public void login(String username, String password): 该法利用参数 username 和 password 登录到 FTP 服务器。使用过 Internet 的用户应该知道, 匿名 FTP 服务器的登录用户名为 anonymous, 密码一般用自己的电子邮件地址。

5. 实现命令功能函数

当登录到 FTP 服务器后, 有可能使用到下面的方法。

(1) public void cd(String remoteDirectory): 该命令用于把远程系统上的目录切换到参

数 `remoteDirectory` 所指定的目录。

(2) `public void cdUp()`: 该命令用于把远程系统上的目录切换到上一级目录。

(3) `public String pwd()`: 该命令可显示远程系统上的目录状态。

(4) `public void binary()`: 该命令可把传输格式设置为二进制格式。

(5) `public void ascii()`: 该命令可把传输协议设置为 ASCII 码格式。

(6) `public void rename(String string, String string1)`: 该命令可对远程系统上的目录或者文件进行重命名操作。

6. 实现输入和输出函数

当登录到 FTP 服务器后, 就可以实现上传和下载文件的功能了, `FtpClient` 类提供了可用于传递并检索目录清单和文件的若干方法, 这些方法返回的是可供读或写的输入、输出流, 分别如下所示。

(1) `public TelnetInputStream list()`: 该方法返回与远程机器上当前目录相对应的一条输入流。

(2) `public TelnetInputStream get(String filename)`: 该方法以写方式打开一条输入流, 通过该输入流可以把远程机器上的文件 `filename` 下载到本地。

(3) `public TelnetOutputStream put(String filename)`: 该方法以写方式打开一条输出流, 通过该输出流可以把文件 `filename` 传送到远程计算机上。

26.3.2 相关 JAR 包导入问题

Sun 公司提供的 JDK 包中, 虽然包含了进行 FTP 操作的各种类和方法, 但是在具体开发时, 就会出现导入 `sun.net.ftp` 出错现象。即“Access restriction: The type `TelnetOutputStream` is not accessible due to restriction on required library `D:\Java\jdk1.6.0_10\jre\lib\rt.jar`”错误, 如图 26.12 所示。

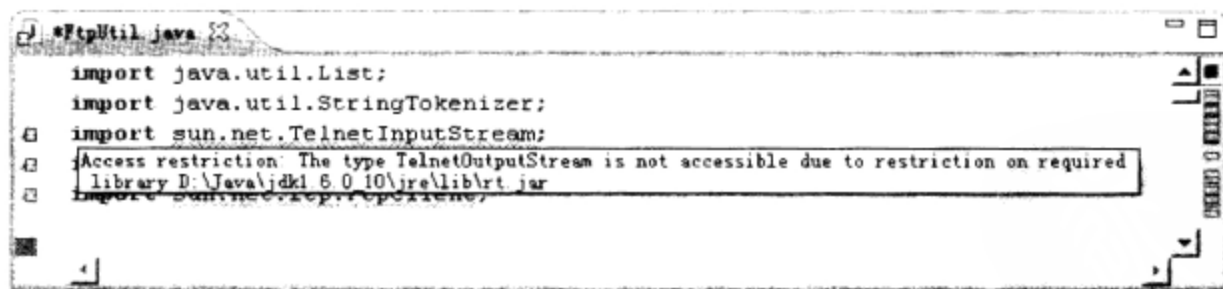


图 26.12 Web 服务器类图

之所以会出现上面的错误, 查看 JDK 帮助文档可以发现, 对于 JDK 中以 `sun` 或 `com.sun` 开头的包都是非公开的, 不能保证在其他版本中继续一致。所以不能直接使用 JRE 中 `sun` 或者 `com.sun` 开头的包, 这些包类只能在 Sun 公司提供的 JRE 中使用, 而在其他的 JRE 中不保证能够使用。

在 MyEclipse 开发环境中, 如果想使用 `sun` 或 `com.sun` 开头包中的类, 只需要自定义 Access Rules 就可以, 具体步骤如下。

(1) 右击 `ftptest` 项目, 在弹出的快捷菜单中选择 `Properties` 选项, 如图 26.13 所示, 会打开如图 26.14 所示的对话框。

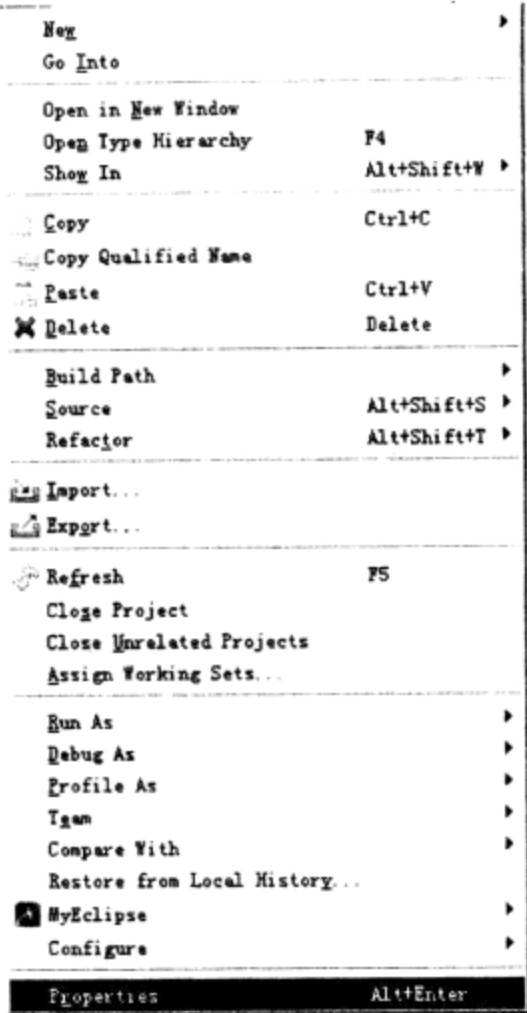


图 26.13 菜单选项

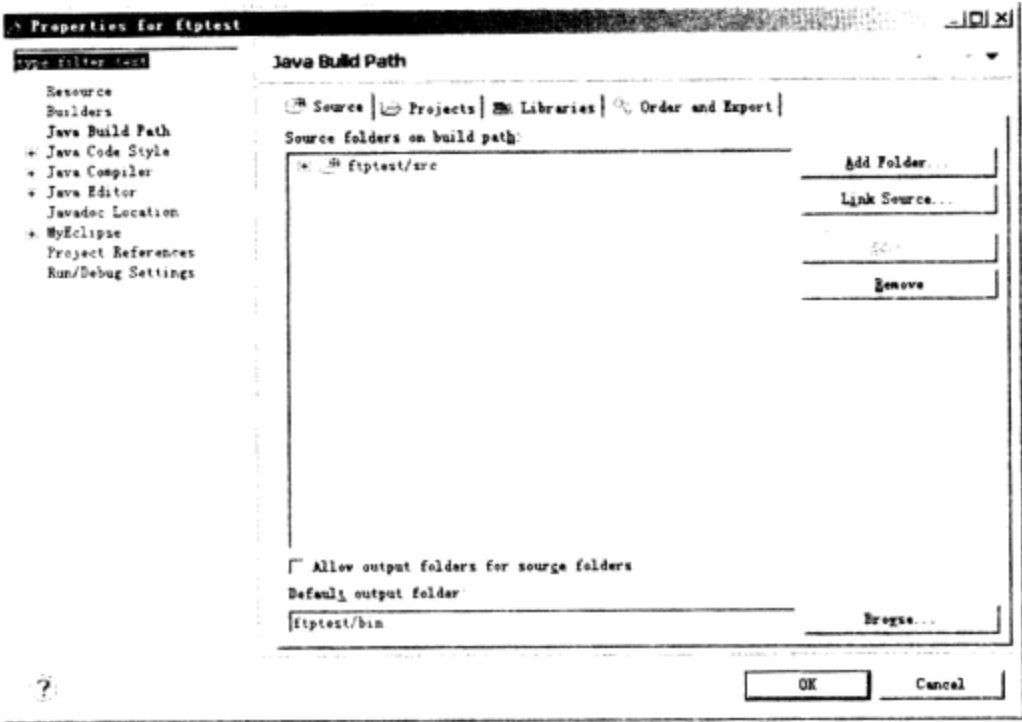


图 26.14 Properties 对话框

(2) 在 Properties 对话框中，选择 Libraries 标签，如图 26.15 所示。双击 JRE System Library 项下的 Access rules 选项，打开如图 26.16 所示的 Type Access Rules 对话框。

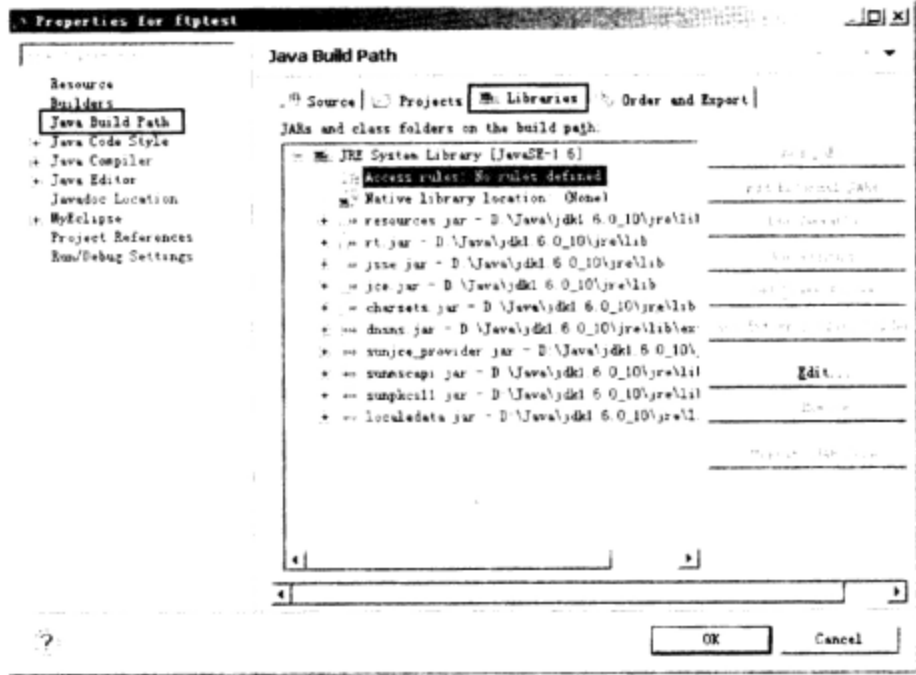


图 26.15 相应设置

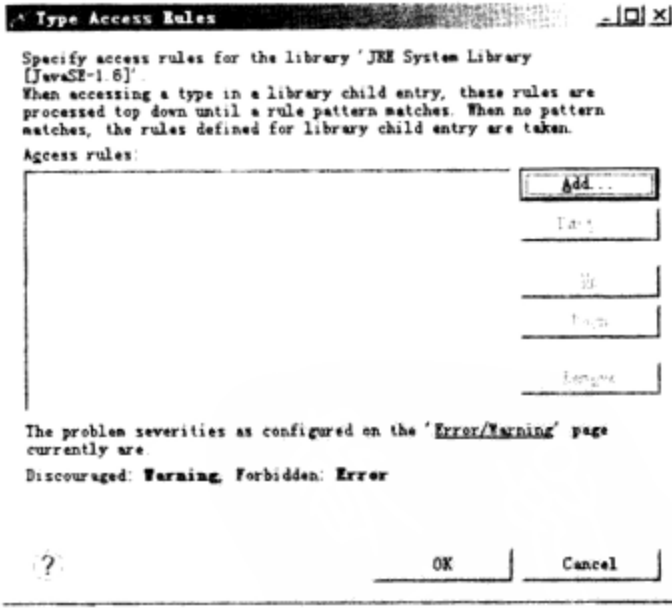


图 26.16 Type Access Rules 对话框

(3) 如果想自定义 Access Rules，只需要单击 Type Access Rules 对话框中的 Add 按钮，会弹出 Add Access Rule 对话框，如图 26.17 所示，然后修改和设置 Resolution 和 Rule Pattern 文本框中的值，如图 26.18 所示。

通过上述步骤的设置就可以完全消除 “Access restriction: The type TelnetOutputStream is not accessible due to restriction on required library D:\Java\jak1.6_10\jre\lib\rt.jar” 错误。

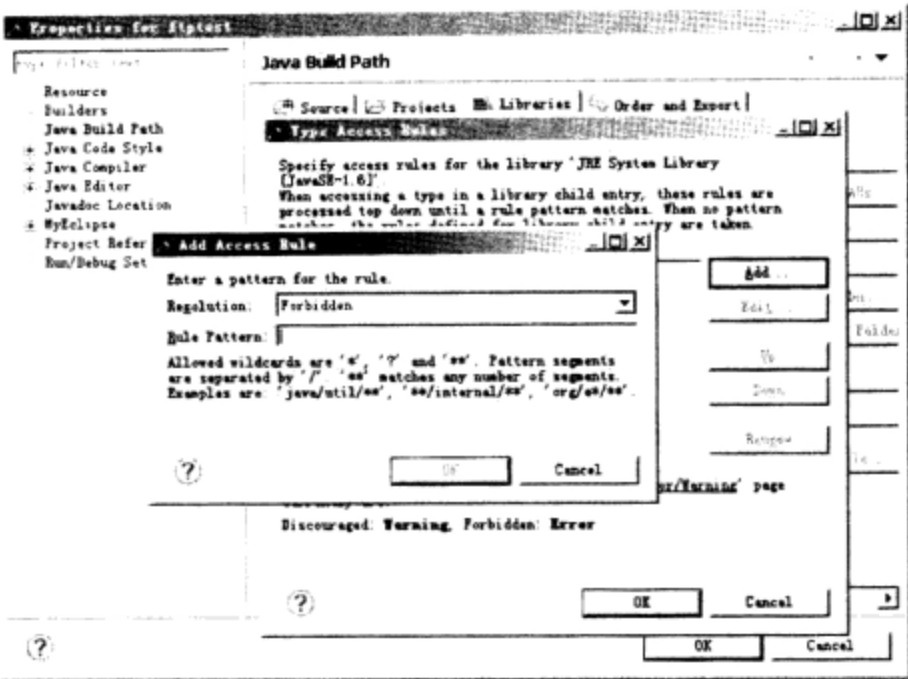


图 26.17 Add Access Rule 对话框

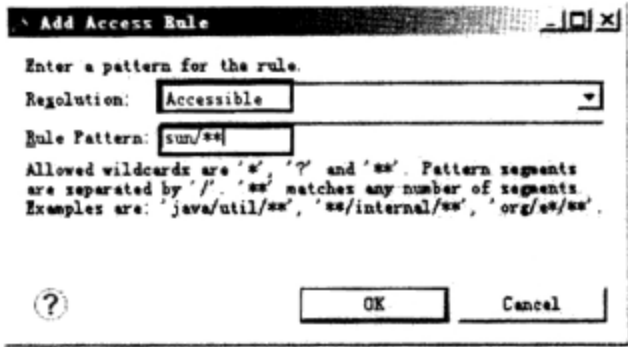


图 26.18 具体设置

26.4 小 结

本章主要通过 FtpClient 类和 I/O 处理机制实现了 FTP 服务器客户端项目，虽然该项目比较小，只包含 FTP 服务器操作的工具类、实现上传文件功能的类和实现下载文件功能的这 3 类，但是却涉及了网络开发中的流概念。

在本章的最后还详细介绍了类 FtpClient 的主要成员变量和方法，以及在具体开发时遇到的一些问题。

第 27 章 Web 服务器（HTTP 协议）

应用程序可以分为两种模式，一种为 C/S 模式，另一种为 B/S 模式。所谓 B/S 模式，就是由 IE 浏览器（B）和 Web 服务器（S）来构成，本章将通过 Java 语言中的事件机制和 HTTP 协议来实现 Web 服务器。本章不仅通过线程机制实现 Web 服务器，还会详细介绍 HTTP 协议的基本知识。

本章的学习目标如下：

- ❑ 掌握 Web 服务器项目；
- ❑ 理解和熟悉 HTTP 协议；
- ❑ 理解和熟悉线程机制。

27.1 Web 服务器原理

“Web 服务器”项目用来模拟 Web 服务器的功能，在具体实现时，为了能够同时连接多个客户端，所以使用 Java 语言中的多线程机制；为了能够实现与客户端连接，会利用 HTTP 协议。

27.1.1 项目结构框架分析

对于“Web 服务器”项目，根据 C/S 软件框架可以知道，需要创建服务器端程序和客户端程序，由于 IE 浏览器可以作为客户端程序，所以该项目只存在一个实现服务器功能的类，如图 27.1 所示。

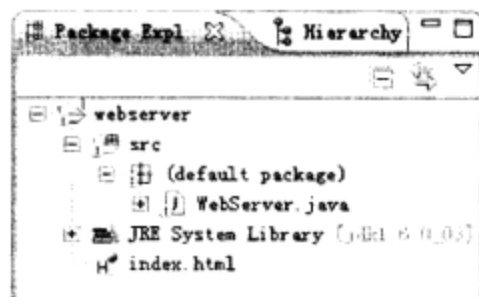


图 27.1 项目目录

27.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括启动 Web 服务器和测试 Web 服务器。

1. 启动Web服务器

为了测试 Web 服务器，需要通过运行 WebServer.java 类启动 Web 服务器，如果运行成功，则输出窗口如图 27.2 所示。

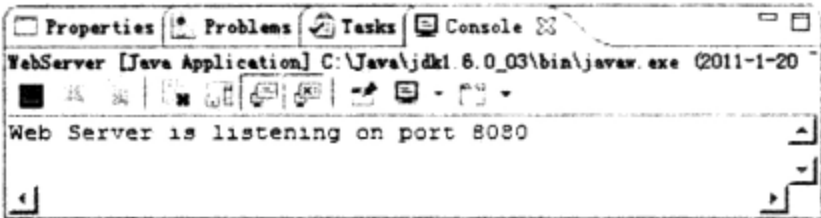


图 27.2 启动服务器

2. 测试Web服务器

当启动 Web 服务器成功后，可以通过在 IE 浏览器中访问网页来进行测试，运行结果如图 27.3 所示。



图 27.3 浏览网页

27.2 Web 服务器的实现过程

本章通过 HTTP 协议类和多线程的相关知识来实现 Web 服务器项目，该项目中存在两个类，实现与浏览器通信功能的类 CommunicateThread 和 Web 服务器类 WebServer，这两个类的关系如图 27.4 所示。

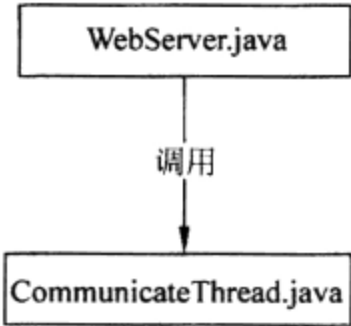


图 27.4 程序关系图

27.2.1 实现与浏览器通信的类

CommunicateThread.java 类用来实现与浏览器的通信功能，该类涉及 HTTP 协议原理

和多线程知识, 其具体内容如代码 27.1 所示, 该类的 UML 如图 27.5 所示。

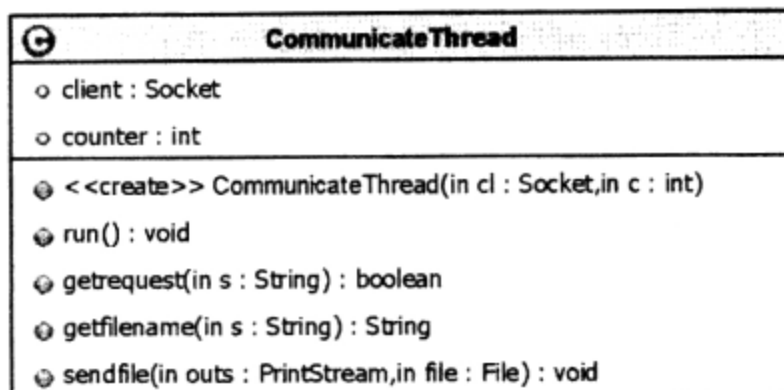


图 27.5 CommunicateThread 类的 UML 图

代码 27.1 实现与浏览器的通信: CommunicateThread.java

```

public class CommunicateThread extends Thread {
    //创建成员变量
    Socket client;                //创建连接 Web 浏览器的 Socket 类对象
    int counter;                  //创建计数器对象
    //构造函数
    public CommunicateThread(Socket cl, int c) {
        //为成员变量赋值
        client = cl;
        counter = c;
    }
    public void run()              //实现 run() 方法
    {
        try {
            //获取客户端的 IP 地址变量
            String destIP = client.getInetAddress().toString();
            int destport = client.getPort();        //获取客户机端口号
            //创建 PrintStream 输出流对象
            PrintStream outstream = new PrintStream(client.getOutputStream());
            //创建输入流的过滤对象
            DataInputStream instream = new DataInputStream(client
                .getInputStream());
            //读取 Web 浏览器提交的请求信息
            String inline = instream.readLine();
            System.out.println("Received:" + inline);
            //判断请求信息
            if (getrequest(inline)) {                //当为 GET 请求时
                String filename = getfilename(inline); //获取文件的名称
                File file = new File(filename);        //创建网页的 file 对象
                if (file.exists()) {                    //当文件存在, 则回送给浏览器
                    //设置返回信息
                    System.out.println(filename + " requested.");
                    outstream.println("HTTP/1.0 200 OK");
                    outstream.println("MIME_version:1.0");
                    outstream.println("Content_Type:text/html");
                    int len = (int) file.length();
                    outstream.println("Content_Length:" + len);
                    outstream.println("");
                    sendfile(outstream, file); //调用 sendfile() 方法发送文件
                    outstream.flush();
                }
            }
        }
    }
}
  
```

```

        } else {
            //当文件不存在时
            //创建显示的内容
            String notfound = "<html><head><title>Not Found</title></head><body><h1>Error 404-file not found</h1></body></html>";
            //设置返回的信息
            ostream.println("HTTP/1.0 404 no found");
            ostream.println("Content_Type:text/html");
            ostream
                .println("Content_Length:"+notfound.length()+2);
            ostream.println("");
            ostream.println(notfound);
            ostream.flush();
        }
    }
    long 27 = 1;
    while (27 < 11100000) {
        27++;
    } //延时
    client.close();
} catch (IOException e) {
    System.out.println("Exception:" + e);
}
}

boolean getrequest(String s) {
    //判断请求的类型是否为 GET
    if (s.length() > 0) {
        if (s.substring(0, 3).equalsIgnoreCase("GET"))
            return true;
    }
    return false;
}

String getfilename(String s) {
    //实现文件名的获取
    String f = s.substring(s.indexOf(' ') + 1);
    f = f.substring(0, f.indexOf(' '));
    try {
        if (f.charAt(0) == '/')
            f = f.substring(1);
    } catch (StringIndexOutOfBoundsException e) {
        System.out.println("Exception:" + e);
    }
    if (f.equals(""))
        f = "index.html";
    return f;
}

void sendfile(PrintStream outs, File file) { //实现发送文件到浏览器的功能
    try {
        //对象 file 的输入流对象
        DataInputStream in = new DataInputStream(new FileInputStream(file));
        int len = (int) file.length(); //获取文件的长度
        byte buf[] = new byte[len]; //创建字节数组
        in.readFully(buf); //读取内容存储到数组 buf 中
        outs.write(buf, 0, len); //输出数组 buf 中的内容
        //输入流和输出流
        outs.flush();
        in.close();
    } catch (Exception e) {
        System.out.println("Error retrieving file.");
    }
}

```



```

        System.exit(1);
    }
}

```

【代码解析】

- ❑ 为了便于实现相应功能, 代码中创建了 3 个方法, `getrequest()` 方法用来检测客户的请求是否为 GET; `getfilename(s)` 方法是从客户请求信息 `s` 中获取要访问的 HTML 文件名; `sendfile()` 方法把指定文件内容通过 Socket 传回给 Web 浏览器。
- ❑ 线程子类主要用来实现两个功能, 即分析 Web 浏览器提交的请求及把应答信息传回给 Web 浏览器。

27.2.2 实现 Web 服务器的类

`WebServer.java` 类用来实现 Web 服务器, 该类主要通过调用 `CommunicateThread` 类来实现, 其具体内容如代码 27.2 所示, 该类的 UML 如图 27.6 所示。

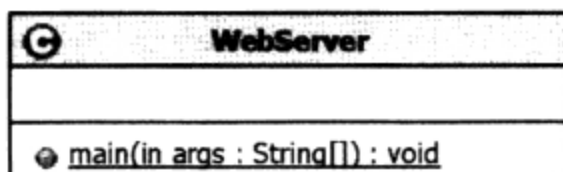


图 27.6 WebServer 类的 UML 图

代码 27.2 Web 服务器对象: `WebServer.java`

```

public class WebServer {
    public static void main(String args[]) {
        //创建两个端口号和连接次数的变量
        int i = 1, PORT = 8080;
        ServerSocket server = null;           //创建 ServerSocket 对象
        Socket client = null;                 //创建 Socket 对象
        try {
            server = new ServerSocket(PORT);   //为对象 ServerSocket 赋值
            //输出相应的信息
            System.out.println("Web Server is listening on port "
                               + server.getLocalPort());
            for (;;) {                         //通过无限循环来不断的接收监听
                client = server.accept();      //获取客户机的连接请求
                //创建 CommunicateThread 对象并启动
                new CommunicateThread(client, i).start();
                i++;                           //实现变量 i 的自增
            }
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}

```

【代码解析】

在上述代码中首先创建 `ServerSocket` 对象, 然后通过该对象的 `accept()` 方法不停地接收请求, 如果连接成功, 则调用 `CommunicateThread` 类的 `start()` 方法来处理请求。

27.2.3 浏览器所请求的页面

当浏览器发送请求到 Web 服务器时, 实际情况是 Web 服务器会根据浏览器的请求返回相应的内容, 本项目只是简单地返回固定的名为 index.html 的网页, 该网页的内容如代码 27.3 所示。

代码 27.3 返回的页面: index.html

```
<HTML>
  <HEAD>
    <META HTTP-EQUIV="Content-Type" content="text/html; charset=utf-8">
    <TITLE>Java Web 服务器</TITLE>
  </HEAD>
  <BODY>
    <!--设置主题-->
    <h3>
      WEB 服务器主页
    </h3>
    <hr>                                <!--设置横线-->
  </BODY>
</HTML>
```

【代码解析】

在上述代码只是一个普通的网页, 在该网页中只通过标签<h3>和<hr>设置了标题和横线。

27.3 知识点扩展——HTTP 协议知识

通过前面两章的学习可以知道, UDP 和 TCP 协议属于网络通信协议, 而 FTP 协议属于网络应用协议。其实, HTTP 协议与 FTP 协议一样, 也属于网络应用协议。

27.3.1 HTTP 协议原理

在网络上经常会遇见 WWW 服务应用系统, 该应用系统最基本的传输单位是 Web 网页。WWW 服务应用系统的工作方式是基于 C/S 模式, 由 Web 浏览器(浏览器)和 Web 服务器(服务器)构成, 两者之间采用超文本传送协议(HTTP)进行通信。

HTTP 协议是基于 TCP/IP 协议之上的协议, 为浏览器和 Web 服务器之间的应用层协议, 是一种通用的、无状态的和面向对象的协议。当通过 HTTP 协议连接服务器时, 一般会经历以下 4 个步骤。

(1) 连接: Web 浏览器与 Web 服务器建立连接, 打开一个名为 socket(套接字)的虚拟文件, 此文件的建立标志着连接建立成功。

(2) 请求: Web 浏览器通过 Socket 向 Web 服务器提交请求。HTTP 的请求一般是 GET 或 POST 命令(POST 用于 FORM 参数的传递)。

(3) 应答: Web 浏览器提交请求后, 通过 HTTP 协议传送给 Web 服务器。Web 服务器接到后, 进行事务处理, 处理结果又通过 HTTP 传回给 Web 浏览器, 从而在 Web 浏览

器上显示出所请求的页面。

(4) 关闭连接: 当应答结束后, Web 浏览器与 Web 服务器必须断开, 以保证其他 Web 浏览器能够与 Web 服务器建立连接。

例如当 IE 浏览器与 `www.test.com:8080/mydir/index.html` 建立了连接, 就会发送 GET 命令 `GET /mydir/index.html HTTP/1.0`。主机名为 `www.test.com` 的 Web 服务器就会从它的文档空间中搜索子目录 `mydir` 的文件 `index.html`。如果找到该文件, Web 服务器把该文件内容传送给相应的 Web 浏览器。

为了能够让浏览器知道传送回文件的类型, Web 服务器首先传送一些 HTTP 头信息, 然后传送具体内容 (即 HTTP 体信息), HTTP 头信息和 HTTP 体信息之间用一个空行分开。

常用的 HTTP 头信息如下。

(1) `HTTP 1.0 200 OK`: 这是 Web 服务器应答的第一行, 列出服务器正在运行的 HTTP 版本号和应答代码。代码 “200 OK” 表示请求完成。

(2) `MIME_Version:1.0`: 表示 MIME 类型的版本。

(3) `content_type`: 这个头信息非常重要, 表示 HTTP 体信息的 MIME 类型。如 `content_type:text/html` 表示传送的数据是 HTML 文档。

(4) `content_length`: 长度值, 表示 HTTP 体信息的长度 (字节)。

27.3.2 实现 HTTP 协议服务器的原理

当创建实现 HTTP 协议的 Web 服务器时, 一般都会根据 HTTP 协议的工作原理, 经历固定的步骤。具体步骤如下。

(1) 创建 `ServerSocket` 类对象, 监听端口 8080。这是为了区别于 HTTP 的标准 TCP/IP 端口 80 而取的。

(2) 等待、接受客户机连接到端口 8080, 得到与客户机连接的 `socket`。

(3) 创建与 `socket` 套接字相关联的输入流 `instream` 和输出流 `outstream`。

(4) 从与 `socket` 关联的输入流 `instream` 中读取一行客户机提交的请求信息, 请求信息的格式为 `GET 路径/文件名 HTTP/1.0`。

(5) 从请求信息中获取请求类型。如果请求类型是 GET, 则从请求信息中获取所访问的 HTML 文件名。没有 HTML 文件名时, 则以 `index.html` 作为文件名。

(6) 如果 HTML 文件存在, 则打开 HTML 文件, 把 HTTP 头信息和 HTML 文件内容通过 `Socket` 传回给 Web 浏览器, 然后关闭文件; 否则发送错误信息给 Web 浏览器。

(7) 关闭与相应 Web 浏览器连接的 `socket` 套接字。

27.4 小 结

本章主要根据 HTTP 协议的工作原理来实现 Web 服务器项目, 虽然该项目比较小, 只包含一个名为 `WebServer` 的类, 但是却实现了目前比较流行的 Web 服务器的基本功能。

在具体设置 Web 服务器项目时, 首先创建了一个实现与浏览器通信的工具类, 然后再创建了调用工具类的 Web 服务器类, 最后为了测试 Web 服务器的功能, 专门创建了一个简单的网页。

第 28 章 QQ 聊天工具

（Swing+多线程+网络编程）

本章将通过 J2SE 的基础知识来实现 QQ 项目，该项目包含所有的基础知识，如 Swing、多线程、网络编程等。为了让读者能够清晰地学习该项目，本章将通过三层结构（MVC）的模式来讲解该项目。

本章的学习目标如下：

- ❑ 掌握 Swing 的基本组件；
- ❑ 熟悉多线程机制；
- ❑ 了解网络编程。

28.1 QQ 聊天工具原理

QQ 软件是深圳市腾讯计算机系统有限公司开发的一款基于 Internet 的即时通信（IM）软件。腾讯 QQ 支持在线聊天、视频电话、点对点断点续传文件、共享文件、网络硬盘、自定义面板等多种功能，是目前使用最广泛的聊天软件之一。

28.1.1 项目结构框架分析

对于 QQ 项目，主要通过三层结构（MVC）来设计目录结构。由于 QQ 项目是即时通信软件，所以分为服务器端程序和客户端程序。QQ 项目服务器端项目目录如图 28.1 所示，各个类的功能如下。

- ❑ com.qq.common 包：该包主要用于存放对象模型类，其中 Message 类表示信息对象、MessageType 类表示信息类型对象、User 类表示用户对象。
- ❑ com.qq.server.model 包：该包主要用于存放 QQ 项目中业务逻辑类，其中 MyQqServer 类表示 QQ 服务器连接功能、SerConClientThread 类表示服务器与客户端的通信线程；ManageClientThread 类表示管理服务器与客户端的通信线程。
- ❑ com.qq.server.view 包：该包主要用于存放视图类，其中 MyServerFrame 类为服务器端界面。
- ❑ com.qq.server.db 包：该包主要用于存放工具类，SqlHelper 类主要实现连接数据库。

QQ 项目客户端项目目录如图 28.2 所示，各个类的功能如下。

- ❑ com.qq.common 包：该包主要用于存放对象模型类，其中 Message 类表示信息对象、MessageType 类表示信息类型对象；User 类表示用户对象。
- ❑ com.qq.server.model 包：该包主要用于存放 QQ 项目中的业务逻辑类，其中 QqClientConServer 类表示 QQ 客户端连接服务器功能；QqClientUser 类表示 QQ

客户端实现用户登录功能。

- ❑ com.qq.server.view 包：该包主要用于存放视图类，其中 QqClientLogin 类为客户端登录界面；QqFriendList 类为客户端的成员列表界面；QqChat 类为 QQ 聊天界面。
- ❑ com.qq.client.tools 包：该包主要用于存放工具类，其中 ClientConServerThread 类表示客户端连接服务器端线程；ManageClientConServerThread 类表示管理客户端连接服务器端线程；ManageQqFriendList 类表示成员列表的操作；ManageQqChat 类表示 QQ 聊天的操作。

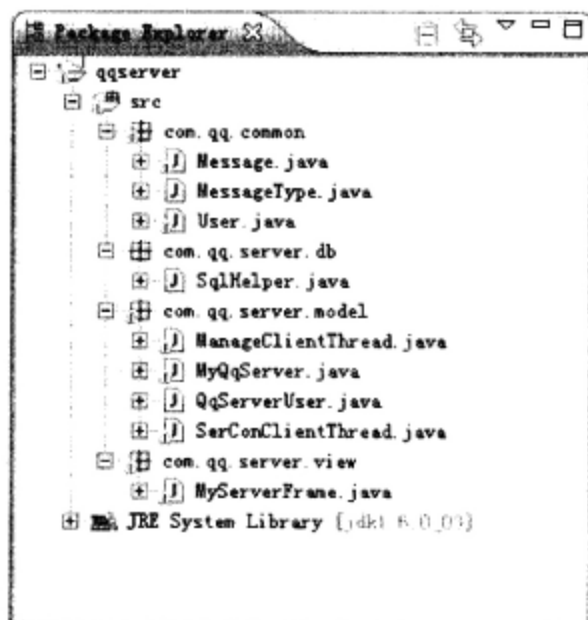


图 28.1 客户端项目目录

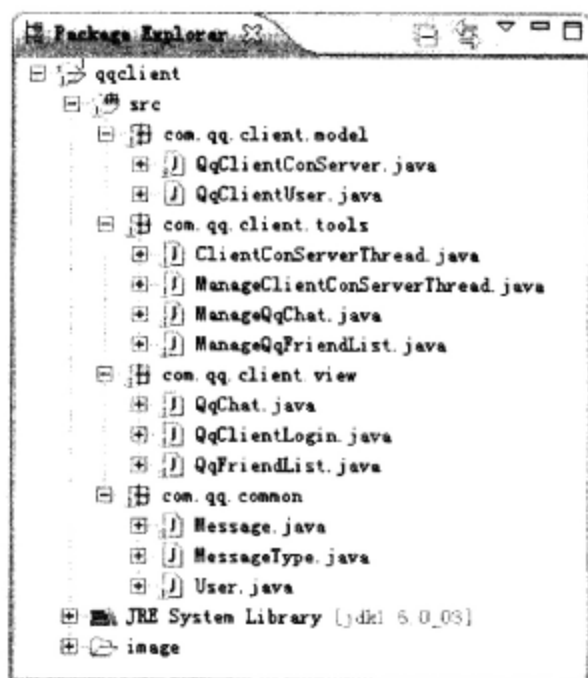


图 28.2 服务器端项目目录

28.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括启动 QQ 服务器、QQ 客户端初始化界面、QQ 客户端登录功能和聊天功能。

1. 启动QQ服务器

当运行 QQ 服务器端项目中的 MyServerFrame 类后，会出现如图 28.3 所示的初始界面——QQ 服务器界面。如果单击“启动服务器”按钮，则表示运行 QQ 服务器；如果想关闭 QQ 服务器，则可以直接单击“关闭服务器”按钮。

2. QQ客户端初始化界面

当运行 QQ 客户端项目中的 QqClientLogin 类后，会出现如图 28.4 所示的 QQ 客户端登录界面。在 QQ 客户端登录界面的“QQ 号码”标签中，输入相应的信息，单击“登录”按钮就可以实现登录功能。

3. QQ客户端登录功能

在 QQ 客户端登录界面中，只要在“QQ 号码”标签的 QQ 号码文本框和 QQ 密码框

中输入相应信息，单击“登录”按钮就可以实现登录功能。当登录成功后，则会进入“成员列表”面板，具体过程如图 28.5 所示。

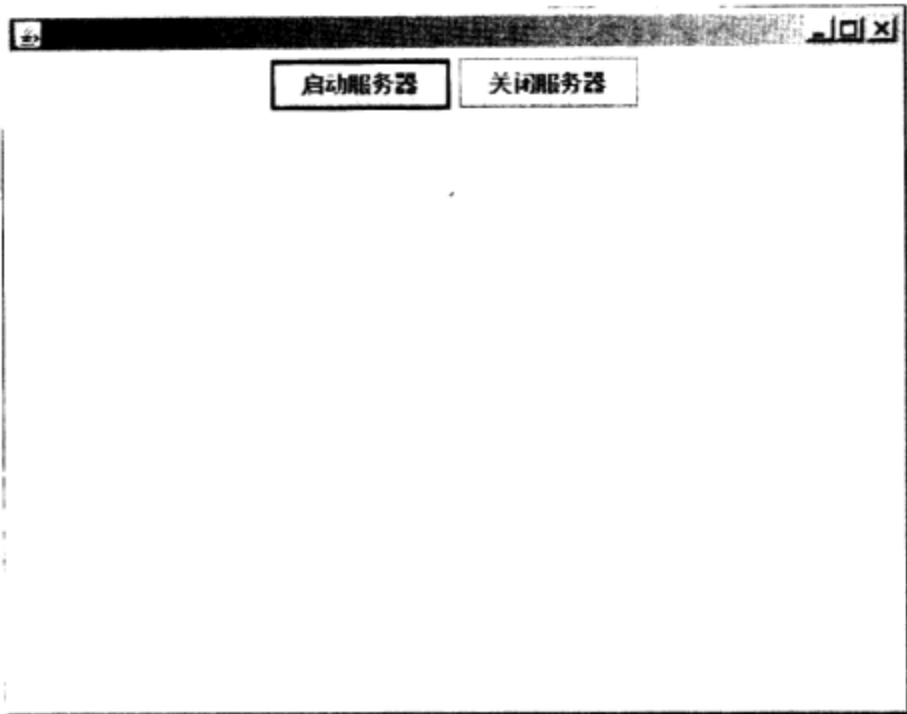


图 28.3 初始化界面

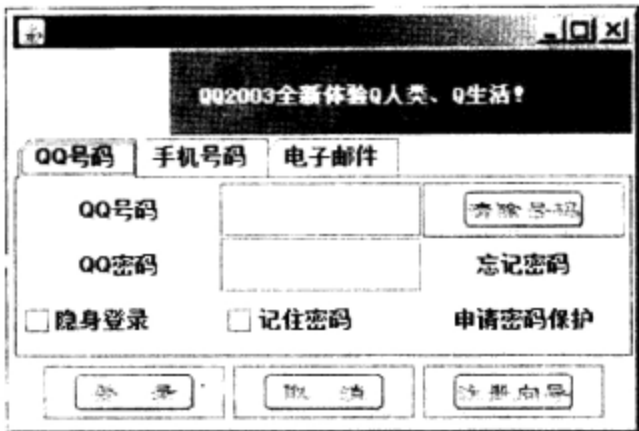


图 28.4 客户端登录界面

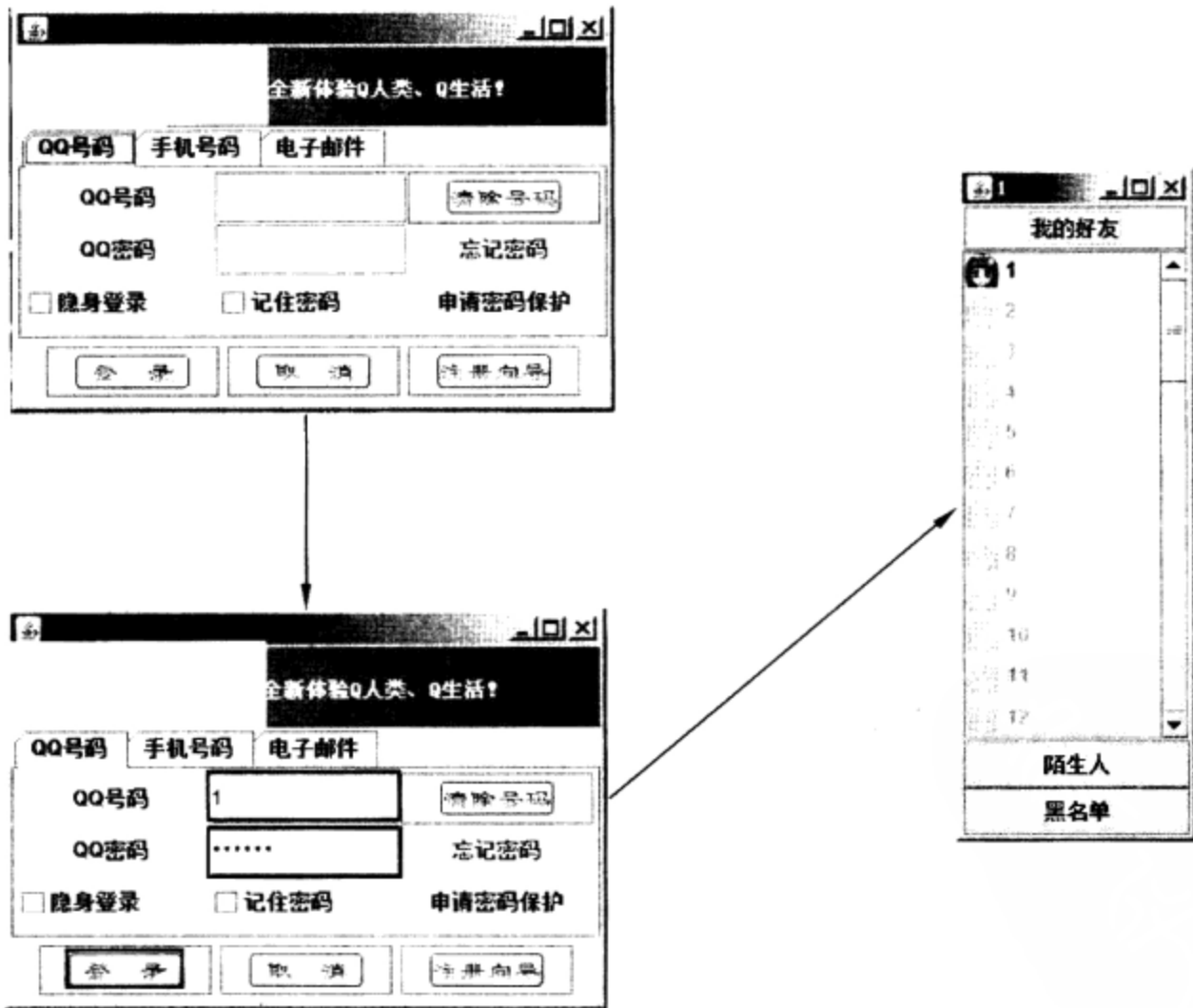


图 28.5 用户登录

为了便于操作，用户 1 和用户 2 都登录 QQ，最后成员列表面板如图 28.6 所示。

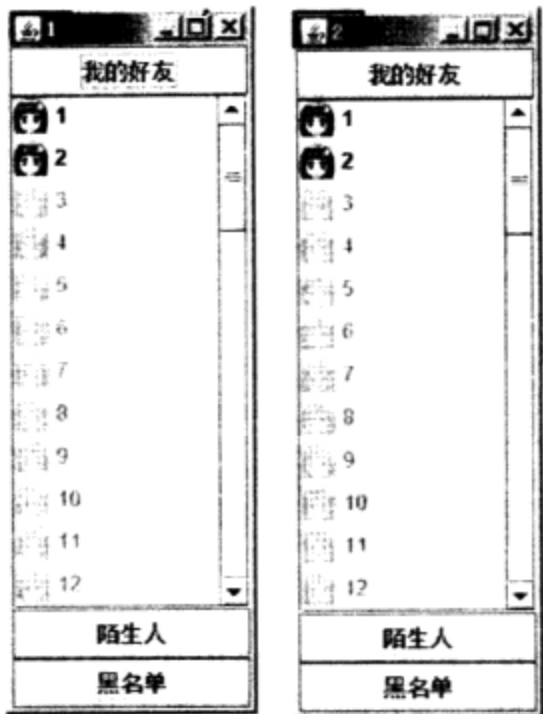


图 28.6 成员列表面板

4. 聊天对话框

在成员列表面板中，如果想聊天，双击成员面板中的成员图标，会打开聊天对话框，具体过程如图 28.7 所示。



图 28.7 聊天对话框

5. 聊天功能

打开“1 正在和 2 聊天”对话框和“2 正在和 1 聊天”对话框。在“1 正在和 2 聊天”对话框的文本输入框中输入“hello, I am NO1”字符串，然后单击“发送”按钮，“2 正在和 1 聊天”对话框的文本域里就会显示出“hello, I am NO1”字符串。同理，如果在“2 正在和 1 聊天”对话框的文本输入框中输入“hello, I am NO2”字符串，然后单击“发送”按钮，“1 正在和 2 聊天”对话框的文本域里就会显示出“hello, I am NO2”字符串，具体过程如图 28.8 所示。



图 28.8 用户聊天过程

28.2 QQ 项目——对象模型的类

QQ 项目运行的过程中会涉及一些对象，如聊天对话框中发送的信息对象、信息类型的对象和用户信息的对象。

28.2.1 信息的类

Message 为 QQ 项目中信息对象类，该类包含了表示信息特性的属性和这些属性的 getter 和 setter 方法。Message 类的 UML 如图 28.9 所示，具体内容如代码 28.1 所示。

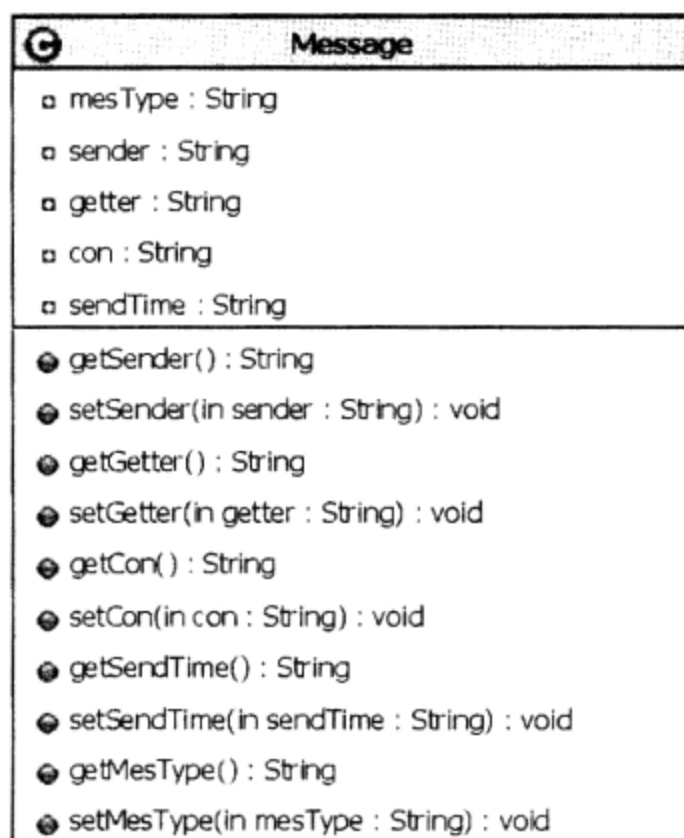


图 28.9 Message 类的类图

代码 28.1 信息对象：Message.java

```
public class Message implements java.io.Serializable {
    //创建成员变量
    private String mesType;                //信息类型对象
```

```

private String sender;           //发送者对象
private String getter;           //接收者对象
private String con;              //发送的内容
private String sendTime;         //信息发送时间

public String getSender() {       //属性 sender 的 getter 和 setter 方法
    return sender;
}
public void setSender(String sender) {
    this.sender = sender;
}
public String getGetter() {       //属性 getter 的 getter 和 setter 方法
    return getter;
}
public void setGetter(String getter) {
    this.getter = getter;
}
public String getCon() {          //属性 conr 的 getter 和 setter 方法
    return con;
}
public void setCon(String con) {
    this.con = con;
}
public String getSendTime() {     //属性 sendTime 的 getter 和 setter 方法
    return sendTime;
}
public void setSendTime(String sendTime) {
    this.sendTime = sendTime;
}
public String getMesType() {      //属性 mesType 的 getter 和 setter 方法
    return mesType;
}
public void setMesType(String mesType) {
    this.mesType = mesType;
}
}

```

【代码解析】

在上述代码中为了便于开发,在 Message 类中专门创建了表示信息的特性,mesType 表示信息类型;sender 表示信息发送者;getter 表示信息的接收者;sendTime 表示信息的发送时间和 con 表示发送的内容。

Message 类会涉及 MessageType 类型,其主要用来表示信息的类型,MessageType 类的具体内容如代码 28.2 所示。

代码 28.2 信息类型类: MessageType.java

```

public interface MessageType {
    String message_succeed = "1";           //表明登录成功
    String message_login_fail = "2";         //表明登录失败
    String message_comm_mes = "3";           //普通信息包
    String message_get_onLineFriend = "4";    //要求在线好友的包
    String message_ret_onLineFriend = "5";    //返回在线好友的包
}

```

【代码解析】

上述代码中定义了一些成员常量，它们的作用主要是为了便于项目的开发。

28.2.2 “用户”的类

User 为 QQ 项目中用户对象类，该类包含了表示用户特性的属性和这些属性的 getter 及 setter 方法。User 类的 UML 如图 28.10 所示，具体内容如代码 28.3 所示。

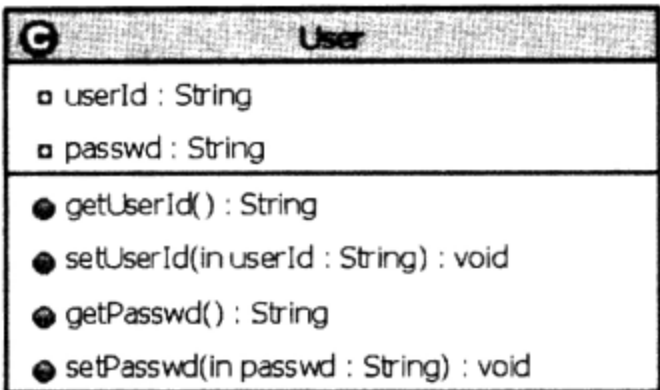


图 28.10 User 类的类图

代码 28.3 用户信息的类：User.java

```
public class User implements java.io.Serializable {
    //创建成员变量
    private String userId;           //用户 ID 变量
    private String passwd;          //用户密码变量
    public String getUserId() {      //userId 属性的 getter 和 setter 方法
        return userId;
    }
    public void setUserId(String userId) {
        this.userId = userId;
    }
    public String getPassword() {   //passwd 属性的 getter 和 setter 方法
        return passwd;
    }
    public void setPasswd(String passwd) {
        this.passwd = passwd;
    }
}
```

【代码解析】

上述代码中为了便于开发，在 User 类中专门创建了表示用户的一些特性，userId 表示用户编号，passwd 表示用户的秘密。

28.3 QQ 项目——登录功能

在 QQ 项目中，如果想实现聊天功能，需要先通过客户端连接到服务器。具体过程是用户先在登录界面输入相应信息，然后把登录界面里的信息发送到服务器，如果服务器端验证成功则会显示出成员列表界面。

28.3.1 QQ 服务器界面的设计

在具体设置 QQ 项目的服务器端程序时, 首先设计服务器的控制窗口, 在该窗口中可以实现启动和关闭服务器的功能, MyServerFrame 类的 UML 如图 28.11 所示, 具体内容如代码 28.4 所示。

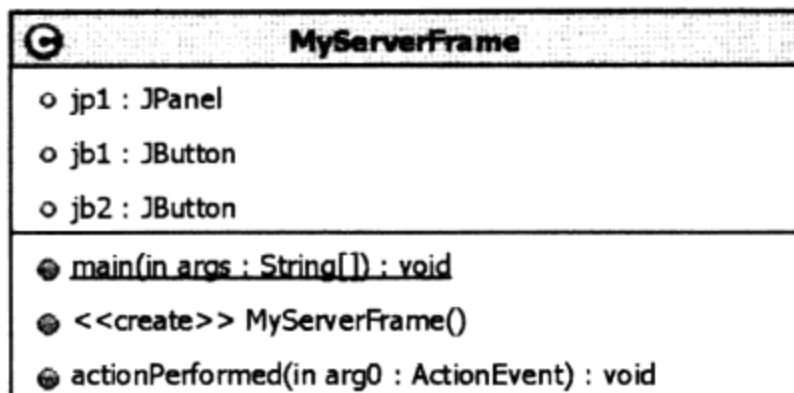


图 28.11 MyServerFrame 类的类图

代码 28.4 服务器端的控制界面类: MyServerFrame.java

```

public class MyServerFrame extends JFrame implements ActionListener {
    //创建成员变量
    JPanel jp1; //面板对象 jp1
    JButton jb1, jb2; //按钮对象 jb1, jb2
    public static void main(String[] args) { //主方法
        MyServerFrame mysf = new MyServerFrame(); //创建 MyServerFrame 类对象
    }
    public MyServerFrame() { //构造函数
        //初始化和设置对象 jp1
        jp1 = new JPanel(); //初始化对象 jp1
        jb1 = new JButton("启动服务器"); //设置 jp1 的文本
        jb1.addActionListener(this); //为 jp1 注册监听器
        jb2 = new JButton("关闭服务器"); //初始化对象 jb2
        //添加按钮对象 jb1 及 jb2 到对象 jp1 中
        jp1.add(jb1);
        jp1.add(jb2);
        this.add(jp1); //添加 jp1 到 MyServerFrame 对象中
        this.setSize(500, 400); //设置窗体大小
        //设置关闭方法
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true); //显示窗体
    }
    public void actionPerformed(ActionEvent arg0) { //事件处理方法
        if (arg0.getSource() == jb1) { //当发生事件的按钮为 jb1
            new MyQqServer(); //启动服务器
        }
    }
}
  
```

【代码解析】

在上述代码中, 由于 QQ 项目服务器端的控制界面是窗口对象, 所以继承了 JFrame

类。在该窗口中有两个按钮对象，即 jb1 和 jb2，jb1 实现启动服务器功能而 jb2 实现关闭服务器的功能。最后该窗口的布局如图 28.12 所示。

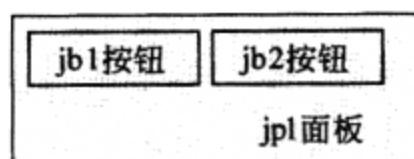


图 28.12 布局

28.3.2 QQ 服务器后台程序

当单击服务器端控制窗口中的“启动”按钮时，会启动服务器。当启动服务器后，服务器会不停地监听等待 QQ 客户端的连接。为了实现该功能，本节专门创建了一个名为 MyQqServer 的类，该类的 UML 如图 28.13 所示，具体内容如代码 28.5 所示。

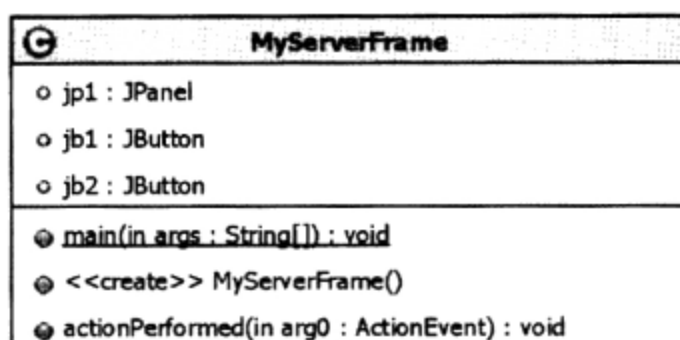


图 28.13 类 MyServerFrame 的类图

代码 28.5 QQ 服务器后台程序：MyQqServer.java

```
public class MyQqServer {
    public MyQqServer() { //构造函数
        try {
            //在 9999 监听
            ServerSocket ss = new ServerSocket(9999);
            //阻塞，等待连接
            while (true) {
                Socket s = ss.accept();
                //接收客户端发来的信息
                ObjectInputStream ois = new ObjectInputStream(s
                    .getInputStream());
                User u = (User) ois.readObject();
                System.out.println("服务器接收到用户 id:" + u.getUserId() + " 密码:"
                    + u.getPasswd());
                Message m = new Message();
                ObjectOutputStream oos = new ObjectOutputStream(s
                    .getOutputStream());
                if (u.getPasswd().equals("123456")) {
                    //返回一个成功登录的信息报
                    m.setMesType("1");
                    oos.writeObject(m);
                    //这里就单开一个线程，让该线程与该客户端保持通信
                    SerConClientThread scct = new SerConClientThread(s);
                    ManageClientThread.addClientThread(u.getUserId(), scct);
                    scct.start(); // 启动与该客户端通信的线程
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



```

        scct.notifyOther(u.getUserId()); //通知其他在线用户

    } else {
        m.setMesType("2");
        oos.writeObject(m);
        s.close(); //关闭 Socket
    }
}
} catch (Exception e) {
    e.printStackTrace();
} finally {
}
}
}

```

【代码解析】

- 在上述代码中，由于需要不停地监听 QQ 客户端，所以需要在死循环中实现监听功能。在具体实现监听功能时，首先创建服务器套接字对象 ss，然后通过 accept() 方法接受客户端的连接，最后获取连接的对象输入流并从该流中获取相关信息，实现校验登录功能。
- 在具体获取输入流时，由于登录时传递的信息由用户名和密码组成，所以需要用到对象输入流类 (ObjectOutputStream)。在具体实现登录功能时，由于该项目主要是讲解网络编程，所以用户的秘密都固定为“123456”。

28.3.3 QQ 客户端登录界面的设计

在 QQ 客户端的登录界面中，可以实现登录功能。QqClientLogin 类的 UML 如图 28.14 所示，具体内容如代码 28.6 所示。



图 28.14 QqClientLogin 类的类图

代码 28.6 QQ 客户端登录界面: QqClientLogin.java

```

public class QqClientLogin extends JFrame implements ActionListener {
    //创建成员变量
    JLabel jbl1; //定义北部需要的组件
    //定义中部需要的组件
    JTabbedPane jtp; //定义选项卡窗口管理对象
    JPanel jp2, jp3, jp4; //定义 3 个 JPanel 对象
    JLabel jp2_jbl1, jp2_jbl2, jp2_jbl3, jp2_jbl4; //定义 4 个标签对象
    JButton jp2_jb1; //定义按钮对象
    JTextField jp2_jtf; //定义文本域对象
    JPasswordField jp2_jpf; //定义密码框对象
    JCheckBox jp2_jcb1, jp2_jcb2; //定义复选框对象
    //定义南部需要的组件
    JPanel jp1; //定义主面板对象
    JButton jp1_jb1, jp1_jb2, jp1_jb3; //定义按钮对象
    public static void main(String[] args) { //主方法
        //创建 QqClientLogin 类对象
        QqClientLogin qqClientLogin = new QqClientLogin();
    }
    public QqClientLogin() { //构造函数
        //处理北部
        jbl1 = new JLabel(new ImageIcon("image/tou.gif"));

        //处理中部
        jp2 = new JPanel(new GridLayout(3, 3));
        jp2_jbl1 = new JLabel("QQ 号码", JLabel.CENTER);
        jp2_jbl2 = new JLabel("QQ 密码", JLabel.CENTER);
        jp2_jbl3 = new JLabel("忘记密码", JLabel.CENTER);
        jp2_jbl3.setForeground(Color.blue);
        ...
        //把控件按照顺序加入到 jp2 中
        jp2.add(jp2_jbl1);
        jp2.add(jp2_jtf);
        jp2.add(jp2_jb1);
        ...
        //创建选项卡窗口
        jtp = new JTabbedPane();
        jtp.add("QQ 号码", jp2);
        jp3 = new JPanel();
        jtp.add("手机号码", jp3);
        jp4 = new JPanel();
        jtp.add("电子邮件", jp4);
        //处理南部
        jp1 = new JPanel();
        jp1_jb1 = new JButton(new ImageIcon("image/denglu.gif"));
        //响应用户单击“登录”按钮
        jp1_jb1.addActionListener(this);
        jp1_jb2 = new JButton(new ImageIcon("image/quxiao.gif"));
        jp1_jb3 = new JButton(new ImageIcon("image/xiangdao.gif"));
        //把 3 个按钮放入到 jp1 中
        jp1.add(jp1_jb1);
        jp1.add(jp1_jb2);
        jp1.add(jp1_jb3);
        this.add(jbl1, "North");
        this.add(jtp, "Center");
    }
}

```

```

//把 jp1 放在南部
this.add(jp1, "South");
this.setSize(350, 240);
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
this.setVisible(true);
}
public void actionPerformed(ActionEvent arg0) {
    //如果用户单击“登录”按钮
    if (arg0.getSource() == jp1_jb1) {
        QqClientUser qqClientUser = new QqClientUser();
        User u = new User();
        u.setUserId(jp2_jtf.getText().trim());
        u.setPasswd(new String(jp2_jpf.getPassword()));
        if (qqClientUser.checkUser(u)) {
            try {
                //把创建好友列表的语句提前
                QqFriendList qqList = new QqFriendList(u.getUserId());
                ManageQqFriendList.addQqFriendList(u.getUserId(), qqList);
                //发送一个要求返回在线好友的请求包
                ObjectOutputStream oos = new ObjectOutputStream(
                    ManageClientConServerThread
                        .getClientConServerThread(u.getUserId())
                        .getS().getOutputStream());
                //做一个 Message
                Message m = new Message();
                m.setMesType(MessageType.message_get_onLineFriend);
                //指明我要的是这个 QQ 号的好友情况
                m.setSender(u.getUserId());
                oos.writeObject(m);
            } catch (Exception e) {
                e.printStackTrace();
            }
            this.dispose(); //关闭登录界面
        } else {
            JOptionPane.showMessageDialog(this, "用户名密码错误");
        }
    }
}
}
}

```

【代码解析】

在上述代码中, 由于 QQ 项目客户端的登录界面是窗口对象, 所以继承了 JFrame 类。在该窗口中存在一个包含输入框和密码框的选项卡, 在该选项卡中输入相应内容单击“登录”按钮后, 就可以实现登录功能。最后该窗口的布局如图 28.15 所示。

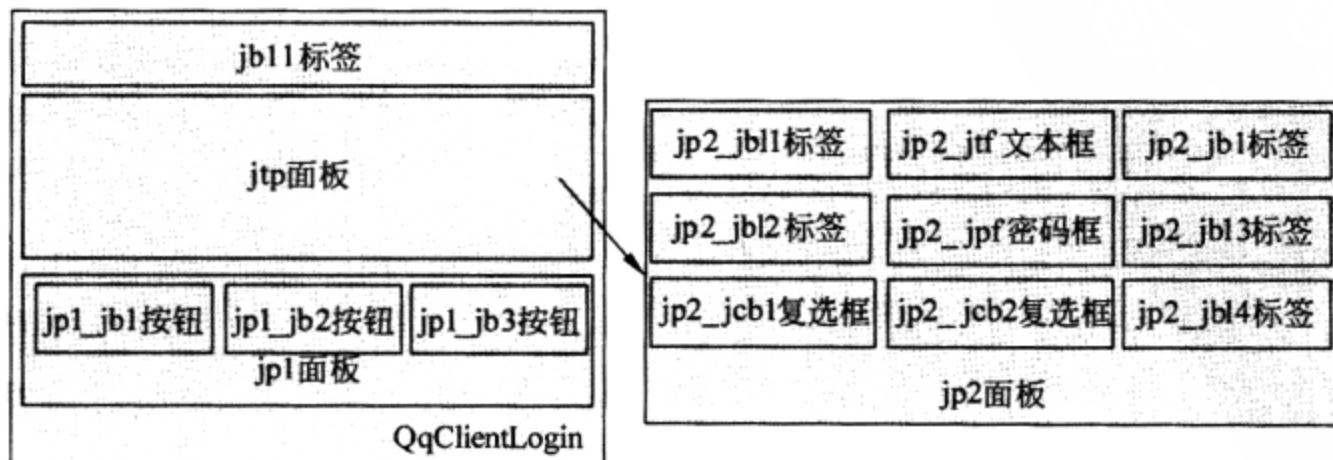


图 28.15 布局

28.3.4 QQ 客户端后台程序

在 QQ 客户端的登录界面中输入相应的信息后，单击“登录”按钮就会调用连接 QQ 服务器的 QqClientConServer 类，该类的 UML 如图 28.16 所示，具体内容如代码 28.7 所示。

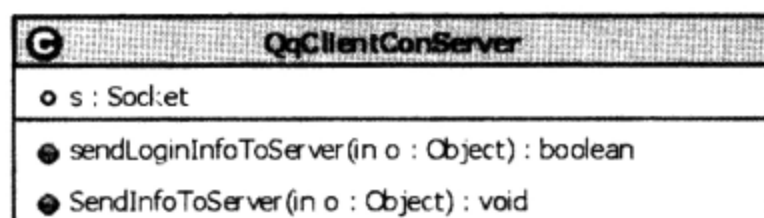


图 28.16 QqClientConServer 类的类图

代码 28.7 QQ 客户端后台：QqClientConServer.java

```

public class QqClientConServer {
    public Socket s;                                //创建成员变量 s
    //发送第一次请求
    public boolean sendLoginInfoToServer(Object o) {
        boolean b = false;                          //创建成员变量 b
        try {
            s = new Socket("127.0.0.1", 9999);      //初始化对象 s
            //创建对象输出流 oos
            ObjectOutputStream oos = new ObjectOutputStream(s.getOutputStream());
            oos.writeObject(o);                      //写入方法
            //创建对象输入流 ois
            ObjectInputStream ois = new ObjectInputStream(s.getInputStream());
            Message ms = (Message) ois.readObject(); //获取消息对象
            //这里就是验证用户登录的地方
            if (ms.getMesType().equals("1")) {
                //创建一个该 QQ 号和服务器端保持通信连接的线程
                ClientConServerThread ccst = new ClientConServerThread(s);
                //启动该通信线程
                ccst.start();
                ManageClientConServerThread.addClientConServerThread-
                    (((User) o)
                        .getUserId(), ccst);
                b = true;
            } else {
                s.close();                            //关闭 Socket
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
        }
        return b;                                    //返回对象 b
    }
}
  
```

【代码解析】

在上述代码中首先创建了客户端套接字对象 s，然后通过对象输出流把对象输出到

套接字里，最后通过对象输入流获取服务器端返回的信息，同时判断信息以决定是否连接成功。

QqClientConServer 类主要实现连接服务器的功能，单击“登录”按钮则会调用发送用户信息到服务器的 QqClientUser 类，该类的 UML 如图 28.17 所示，具体内容如代码 28.8 所示。

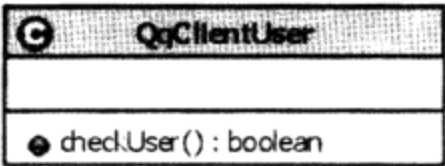


图 28.17 QqClientUser 类的类图

代码 28.8 发送信息类: QqClientUser.java

```
public class QqClientUser {
    public boolean checkUser(User u) { //校验用户方法
        //调用 sendLoginInfoToServer() 方法
        return new QqClientConServer().sendLoginInfoToServer(u);
    }
}
```

【代码解析】

在上述代码中为了发送对象到服务器，直接调用了 QqClientConServer 类的 sendLoginInfoToServer()方法来完成。

28.3.5 成员列表窗口

当通过 QQ 客户端的登录界面登录成功后，会出现成员列表窗口 QqFriendList，其 UML 如图 28.18 所示，具体内容如代码 28.9 所示。

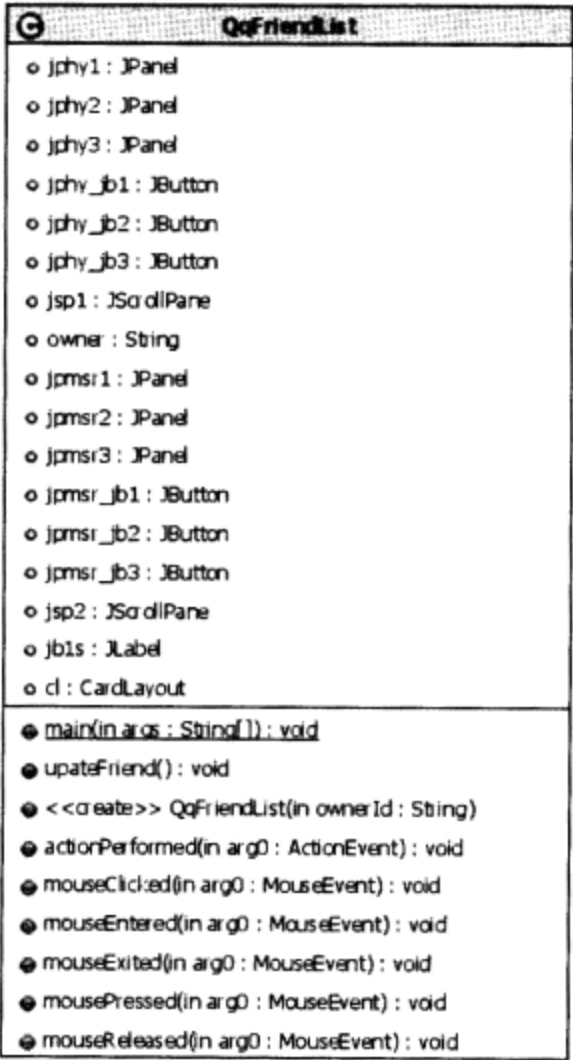


图 28.18 QqFriendList 类的类图

代码 28.9 QQ 成员列表界面: QqFriendList.java

```

public class QqFriendList extends JFrame implements ActionListener,
    MouseListener {
    //创建成员变量
    //处理第一张卡片 (好友)
    JPanel jphy1, jphy2, jphy3; //创建面板对象
    JButton jphy_jb1, jphy_jb2, jphy_jb3; //创建按钮对象
    JScrollPane jsp1; //创建滚动条对象
    String owner; //创建字符串
    //处理第二张卡片 (陌生人)
    JPanel jpmsr1, jpmsr2, jpmsr3; //创建面板对象
    JButton jpmsr_jb1, jpmsr_jb2, jpmsr_jb3; //创建按钮对象
    JScrollPane jsp2; //创建滚动条对象
    JLabel[] jbls; //创建标签对象数组
    CardLayout cl; //创建 CardLayout 布局管理器
    public void upateFriend(Message m) { //更新在线的好友情况
        String onLineFriend[] = m.getCon().split(" ");
        for (int i = 0; i < onLineFriend.length; i++) {
            jbls[Integer.parseInt(onLineFriend[i]) - 1].setEnabled(true);
        }
    }
    public QqFriendList(String ownerId) { //构造函数
        this.owner = ownerId;
        //处理第一张卡片 (显示好友列表)
        jphy_jb1 = new JButton("我的好友");
        jphy_jb2 = new JButton("陌生人");
        jphy_jb2.addActionListener(this);
        jphy_jb3 = new JButton("黑名单");
        jphy1 = new JPanel(new BorderLayout());
        //假定有 50 个好友
        jphy2 = new JPanel(new GridLayout(50, 1, 4, 4));
        //给 jphy2, 初始化 50 个好友
        jbls = new JLabel[50];
        for (int i = 0; i < jbls.length; i++) {
            jbls[i] = new JLabel(i + 1 + "", new ImageIcon("image/mm.jpg"),
                JLabel.LEFT);
            jbls[i].setEnabled(false);
            if (jbls[i].getText().equals(ownerId)) {
                jbls[i].setEnabled(true);
            }
            jbls[i].addMouseListener(this);
            jphy2.add(jbls[i]);
        }
        jphy3 = new JPanel(new GridLayout(2, 1));
        ...
        //在窗口显示自己的编号
        this.setTitle(ownerId);
        this.setSize(140, 400);
        this.setVisible(true);
    }
    public void actionPerformed(ActionEvent arg0) { //动作处理方法
        if (arg0.getSource() == jphy_jb2) {
            //如果单击了“陌生人”按钮, 就显示第二张卡片
            cl.show(this.getContentPane(), "2");
        } else if (arg0.getSource() == jpmsr_jb1) {
            //如果单击了“我的好友”按钮, 就显示第一张卡片

```

省略部分代码


```

        cl.show(this.getContentPane(), "1");
    }
}
public void mouseClicked(MouseEvent arg0) {    //鼠标单击组件处理方法
    //响应用户双击的事件, 并得到好友的编号
    if (arg0.getClickCount() == 2) {
        //得到该好友的编号
        String friendNo = ((JLabel) arg0.getSource()).getText();
        //System.out.println("你希望和 "+friendNo+" 聊天");
        QqChat qqChat = new QqChat(this.owner, friendNo);
        //将聊天界面加入到管理类中
        ManageQqChat.addQqChat(this.owner + " " + friendNo, qqChat);
    }
}
public void mouseEntered(MouseEvent arg0) {    //鼠标进入组件处理方法
    JLabel jl = (JLabel) arg0.getSource();    //获取发生事件的事件源
    jl.setForeground(Color.red);              //设置颜色
}
public void mouseExited(MouseEvent arg0) {    //鼠标离开组件处理方法
    JLabel jl = (JLabel) arg0.getSource();    //获取发生事件的事件源
    jl.setForeground(Color.black);            //设置颜色
}
public void mousePressed(MouseEvent arg0) {    //鼠标按下组件处理方法
}
public void mouseReleased(MouseEvent arg0) {    //鼠标释放组件处理方法
}
}

```

【代码解析】

在上述代码中通过构造函数来初始化成员列表窗口, 为了便于操作还创建了更新在线好友的方法 `upateFriend()`, 同时还实现了动作处理事件, 即面板的显示, 该窗口的布局如图 28.19 所示。

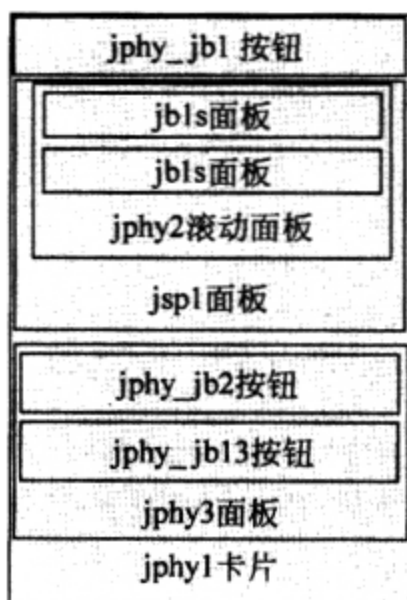


图 28.19 布局

28.4 QQ 项目——聊天功能

当用户通过 QQ 客户端连接服务器后, 如果验证成功就出现成员列表窗口。在成员列

表窗口中，如果双击显示成员就会出现聊天窗口，通过聊天窗口可以实现相关成员间的即时通信。成员即时通信的原理如图 28.20 所示。

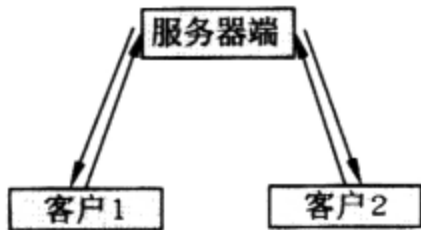


图 28.20 客户端聊天原理

28.4.1 服务器端的信息转发

为了实现客户端的即时通信功能，服务器必须实现信息的转发功能。为了实现该功能，服务器首先要把每个 Socket 套接字保存在 HashMap 集合中，同时利用客户端的 ID 标示各个套接字。

Socket 套接字存储容器的类为 ManageClientThread，其 UML 如图 28.21 所示，具体内容如代码 28.10 所示。

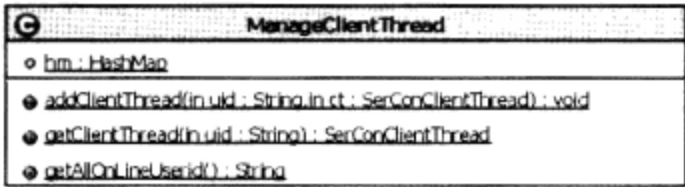


图 28.21 ManageClientThread 类的类图

代码 28.10 存储套接字容器：ManageClientThread.java

```
public class ManageClientThread {
    //创建集合对象 hm
    public static HashMap hm = new HashMap<String, SerConClientThread>();
    //向 hm 中添加一个客户端通信线程
    public static void addClientThread(String uid, SerConClientThread ct) {
        hm.put(uid, ct);
    }
    //获取 SerConClientThread 类对象
    public static SerConClientThread getClientThread(String uid) {
        return (SerConClientThread) hm.get(uid);
    }
    public static String getAllOnLineUserid() { //返回当前在线的人数情况
        Iterator it = hm.keySet().iterator(); //使用迭代器完成
        String res = ""; //创建空字符串
        while (it.hasNext()) {
            res += it.next().toString() + " ";
        }
        return res; //返回字符串对象
    }
}
```

【代码解析】

在上述代码中首先创建了一个名为 hm 的集合类对象，然后通过 getter 和 setter 方法来

获取和存储 SerConClientThread 类对象，最后创建了实现显示当前在线人数的方法 getAllOnLineUserid()。

每当 QQ 服务器端与 QQ 客户器端形成一个套接字时，QQ 服务器端就会单独开辟一个线程，并通过该套接字实现与客户端的通信。其实，ManageClientThread 类中的集合存储的就是 QQ 服务器端与 QQ 客户器端的通信线程类 SerConClientThread，该类的 UML 如图 28.22 所示，具体内容如代码 28.11 所示。

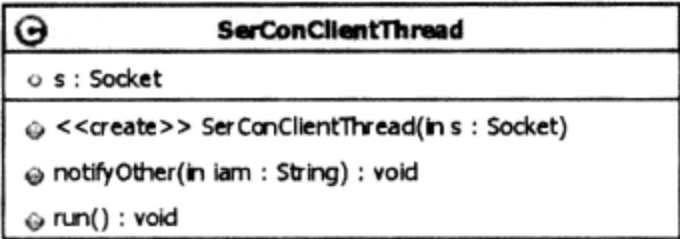


图 28.22 SerConClientThread 类的类图

代码 28.11 服务器和某个客户端的通信线程: SerConClientThread.java

```
public class SerConClientThread extends Thread {
    Socket s; //创建套接字对象
    public SerConClientThread(Socket s) {
        this.s = s; //把服务器和该客户端的连接赋给 s
    }
    public void notifyOther(String iam) { //让该线程去通知其他用户
        HashMap hm = ManageClientThread.hm; //得到所有在线人员的线程
        Iterator it = hm.keySet().iterator(); //迭代器
        while (it.hasNext()) {
            Message m = new Message(); //创建消息类对象
            //设置对象 m
            m.setCon(iam);
            m.setMesType(MessageType.message_ret_onLineFriend);
            //取出在线人的 ID
            String onLineUserId = it.next().toString();
            try {
                //创建对象输出流对象
                ObjectOutputStream oos = new ObjectOutputStream(
                    ManageClientThread.getClientThread(onLineUserId).s
                        .getOutputStream());
                m.setGetter(onLineUserId); //设置对象 m
                oos.writeObject(m); //写入对象
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
    public void run() { //这里该线程就可以接收客户端的信息
        while (true) {
            try {
                ObjectInputStream ois = new ObjectInputStream(s
                    .getInputStream()); //创建对象输入流对象 ois
                //创建消息对象
                Message m = (Message) ois.readObject();
                //对从客户端取得的消息进行类型判断，然后做相应的处理
                if (m.getMesType().equals(MessageType.message_comm_mes)) {
```

```

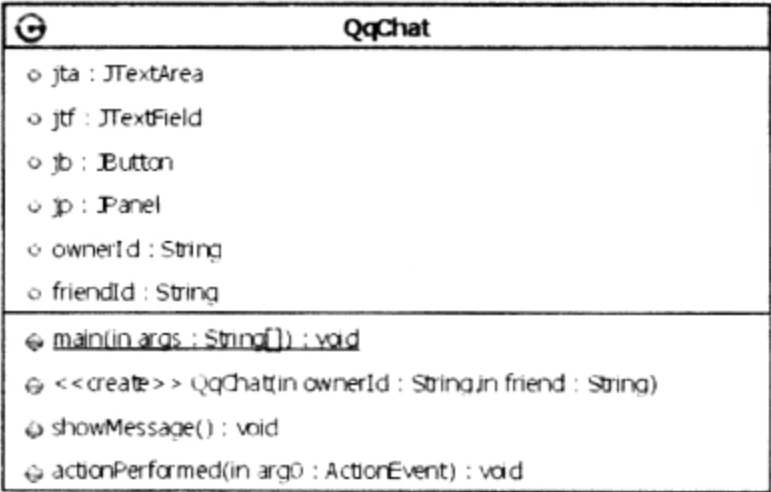
        //一会完成转发
        //取得接收人的通信线程
        SerConClientThread sc = ManageClientThread
            .getClientThread(m.getGetter());
        ObjectOutputStream oos = new ObjectOutputStream(sc.s
            .getOutputStream());
        oos.writeObject(m);
    } else if (m.getMesType().equals(
        MessageType.message_get_onLineFriend)) {
        //将在服务器的好友返回给客户端
        String res = ManageClientThread.getAllOnLineUserid();
        Message m2 = new Message(); //创建消息对象 m2
        //设置消息对象
        m2.setMesType(MessageType.message_ret_onLineFriend);
        m2.setCon(res);
        m2.setGetter(m.getSender());
        //创建输出流对象 oos
        ObjectOutputStream oos = new ObjectOutputStream(s
            .getOutputStream());
        oos.writeObject(m2); //写入对象
    }
} catch (Exception e) {
    e.printStackTrace();
}
}
}
}
}
```

【代码解析】

上述代码首先在 `run()` 方法中实现与客户端的连接，为了便于后面内容的编写，这里专门创建了让该线程去通知其他用户的方法 `notifyOther()`。

28.4.2 客户端信息的发送和接收

在成员列表面板中，如果想聊天，双击成员面板中的成员图标，会打开聊天对话框（`QqChat` 类），该类的 UML 如图 28.23 所示，具体内容如代码 28.12 所示。



• 图 28.23 QqChat 类的类图

代码 28.12 聊天界面：QqChat.java

```

public class QqChat extends JFrame implements ActionListener {
    //创建成员变量
```

```

        JTextArea jta;                //创建文本域对象
        JTextField jtf;              //创建文本框对象
        JButton jb;                  //创建按钮对象
        JPanel jp;                   //创建面板对象
        String ownerId;              //创建字符串对象
        String friendId;             //创建字符串对象
        public QqChat(StringownerId,Stringfriend) { //构造函数
            this.ownerId = ownerId;   //初始化对象 ownerId
            this.friendId = friend;    //初始化对象 friendId
            jta = new JTextArea();     //初始化对象 jta
            jtf = new JTextField(15);  //初始化对象 jtf
            jb = new JButton("发送");  //初始化对象 jb
            jb.addActionListener(this); //为对象 jb 添加事件监听器
            jp = new JPanel();         //初始化对象 jp
            jp.add(jtf);               //添加 jtf 到 jp
            jp.add(jb);               //添加 jb 到 jp
            this.add(jta, "Center");   //添加 jta 到 QqChat
            this.add(jp, "South");     //添加 jp 到 QqChat
            //设置标题
            this.setTitle(ownerId + " 正在和 " + friend + " 聊天");
            //设置标题图片
            this.setIconImage((new ImageIcon("image/qq.gif").getImage()));
            this.setSize(300, 200);    //设置大小
            this.setVisible(true);     //设置可见
        }
        public void showMessage(Message m) { //显示消息的方法
            //创建对象 info
            String info = m.getSender() + " 对 " + m.getGetter() + " 说:"
                + m.getCon() + "\r\n";
            this.jta.append(info);      //追加字符串
        }
        public void actionPerformed(ActionEventarg0) { //处理动作的方法
            if (arg0.getSource() == jb) { //如果用户单击了, 发送按钮
                Message m = new Message(); //创建 m 对象
                //初始化对象 m
                m.setMesType(MessageType.message_comm_mes);
                m.setSender(this.ownerId);
                m.setGetter(this.friendId);
                m.setCon(jtf.getText());
                m.setSendTime(new java.util.Date().toString());
                // 发送给服务器
                try {
                    ObjectOutputStream oos = new ObjectOutputStream(
                        ManageClientConServerThread.getClientConServer
                        Thread(
                            ownerId).getS().getOutputStream());
                    oos.writeObject(m); //写入
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

```

【代码解析】

在上述代码中, 由于 QQ 项目的聊天界面是窗口对象, 所以继承了 JFrame 类。在该窗

口中有两个界面对象，即文本输入域对象 jta 和面板对象 jp，而在面板对象 jp 中则有文本输入框对象 jtf 和按钮对象 jb。最后该窗口的布局如图 28.24 所示。

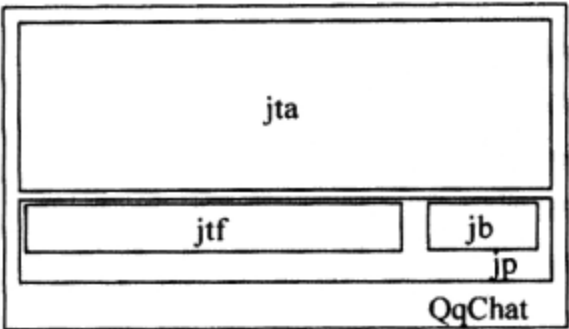


图 28.24 布局

28.4.3 客户端信息转发类

为了实现客户端的即时通信，客户端间需要通过服务器实现信息的转发。为了实现客户端间信息的转发，需要保持客户端和服务端间的通信并管理该通信的类。

客户端和服务端间通信类存储容器的类为 ManageClientConServerThread，其 UML 如图 28.25 所示，具体内容如代码 28.13 所示。

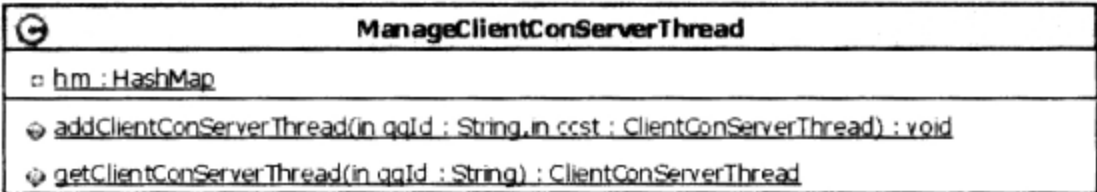


图 28.25 ManageClientConServerThread 类图

代码 28.13 客户端连接服务器的类：ManageClientConServerThread.java

```
public class ManageClientConServerThread {
    //创建成员变量 hm
    private static HashMap hm = new HashMap<String, ClientConServerThread>();
    //把创建好的 ClientConServerThread 放入到 hm 中
    public static void addClientConServerThread(String qqId,
        ClientConServerThread ccst) {
        hm.put(qqId, ccst);
    }
    //可以通过 qqId 取得该线程
    public static ClientConServerThread getClientConServerThread(String qqId) {
        return (ClientConServerThread) hm.get(qqId);
    }
}
```

【代码解析】

上述代码中首先创建了一个名为 hm 的集合类对象，然后通过 getter 和 adder 方法来获取和存储 ClientConServerThread 类对象。

每当 QQ 客户器端与 QQ 服务器端形成一个套接字时，QQ 客户器端就会单独开辟一个线程，并通过该套接字实现与客户端通信。其实 ManageClientConServerThread 类中的集

合存储的就是 QQ 客户器端与 QQ 服务器端的通信线程类 ClientConServerThread，该类的 UML 如图 28.26 所示，具体内容如代码 28.14 所示。

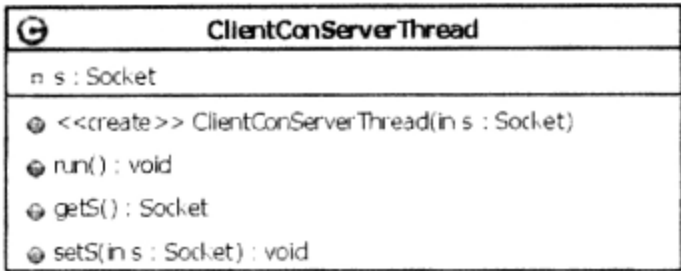


图 28.26 ClientConServerThread 类图

代码 28.14 实现关于客户端连接服务器的线程类：ClientConServerThread.java

```
public class ClientConServerThread extends Thread {
    private Socket s; //创建属性 s
    public ClientConServerThread(Socket s) { //构造函数
        this.s = s;
    }
    public void run() { //实现 run() 方法
        while (true) {
            //不停地读取从服务器端发来的消息
            try {
                //创建对象输入流对象 ois
                ObjectInputStream ois = new ObjectInputStream(s
                    .getInputStream());
                //获取信息对象 m
                Message m = (Message) ois.readObject();
                if (m.getMesType().equals(MessageType.message_comm_mes)) {
                    //把从服务器获得的消息，显示到该显示的聊天界面
                    QqChat qqChat = ManageQqChat.getQqChat(m.getGetter() + " "
                        + m.getSender());
                    //显示
                    qqChat.showMessage(m); //显示信息
                } else if (m.getMesType().equals(
                    MessageType.message_ret_onLineFriend)) {
                    String con = m.getCon(); //获取字符串变量 con
                    String friends[] = con.split(" ");
                    String getter = m.getGetter();
                    //修改相应的好友列表
                    QqFriendList qqFriendList = ManageQqFriendList
                        .getQqFriendList(getter);
                    //更新在线好友
                    if (qqFriendList != null) {
                        qqFriendList.upateFriend(m);
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
    public Socket getS() { //属性 s 的 getter 和 setter 方法
        return s;
    }
    public void setS(Socket s) {
```

```
        this.s = s;  
    }  
}
```

【代码解析】

在上述代码中首先在创建的构造函数中实现初始化变量 s 的功能,然后在 run()方法中实现 QQ 客户端与 QQ 服务器端之间的连接,最后还创建了成员变量 s 的 getter 和 setter 方法。

28.5 小 结

本章主要介绍了一个模拟腾讯计算机系统有限公司的 QQ 软件项目,该项目与腾讯公司的 QQ 软件相比,虽然功能比较简单,但是却实现了软件的最主要也是最基本的功能,同时完全基于 Java 语言的网络编程构建而成。

在具体实现 QQ 项目时,由于该项目是 C/S 类型,即需要一个服务器 (Server) 和客户端 (Client),所以完整 QQ 项目由服务器端 QqServer 和客户端 QqClient 来组成,但无论实现服务器端还是客户端都通过 MVC 模式来实现。

在具体实现客户端时,首先编程 model 包,其中 QqClientConServer 类表示 QQ 客户端连接服务器功能、QqClientUser 类表示 QQ 客户端实现用户登录功能;在具体实现 Model 层时,需要操作相应的对象,所以在 common 包中,创建了表示信息对象的类 Message、表示信息类型的类 MessageType 和表示用户对象的类 User。最后根据项目的需要创建了客户端视图 view 包,其中 QqClientLogin 类为客户端登录界面;QqFriendList 类为客户端的成员列表界面;QqChat 类为 QQ 聊天界面。

与客户端代码一样,服务器端也分为三层,并且表示对象层的 common 包的类与客户端代码完全一样。因此,其中的 model 包表示服务器端的逻辑功能,view 包表示服务器端的视图。

第 7 篇 项目案例实战

- ▶▶ 第 29 章 人员信息管理项目（接口设计模式+MySQL 数据库）
- ▶▶ 第 30 章 中国象棋游戏（GUI+游戏规则算法）
- ▶▶ 第 31 章 俄罗斯方块游戏网络版(Swing+多线程+网络编程)
- ▶▶ 第 32 章 图书管理系统项目（GUI+Oracle 数据库）

第 29 章 人员信息管理项目

（接口设计模式+MySQL 数据库）

本章将综合 J2SE 接口的相关设计模式，实现一个名为“人员信息管理”的项目，即要求通过该项目实现用户信息的统一管理（增加、删除、修改和查找）。由于该项目主要用在接口在开发中的具体应用和数据库的相关操作方面，所以该项目的界面都使用命令行方式来完成。

本章的学习目标如下：

- ❑ 掌握面向对象思想中的接口；
- ❑ 熟悉工厂设计模式和代理设计模式；
- ❑ 熟练操作数据库。

29.1 人员信息管理原理

“人员信息管理”是一个很大的信息管理系统，本章的项目只是实现基本的业务功能应用。在本章的人员信息管理项目中主要实现增加用户、修改用户、删除用户、查询单个用户、查询全部用户和退出系统的功能。

29.1.1 项目结构框架分析

对于人员信息管理项目，主要利用面向对象思想中的接口来实现，即接口编程模型。所谓接口编程模型，就是把整个项目中涉及的对象分别利用接口表示出来。人员信息管理项目目录如图 29.1 所示，各个包的功能如下。

- ❑ com.cjgong.useradmin.dao 包：数据连接操作的包。
- ❑ com.cjgong.useradmin.dao.factory 包：数据库连接操作的工厂包
- ❑ com.cjgong.useradmin.dao.impl 包：实现数据连接操作的包。
- ❑ com.cjgong.useradmin.dao.proxy 包：数据连接操作的代理包。
- ❑ com.cjgong.useradmin.dbc 包：数据库连接操作的包。
- ❑ com.cjgong.useradmin.menu 包：项目菜单的包。
- ❑ com.cjgong.useradmin.operate 包：业务层的包。
- ❑ com.cjgong.useradmin.test 包：测试的包。
- ❑ com.cjgong.useradmin.util 包：工具类的包。
- ❑ com.cjgong.useradmin.vo 包：用户模型的包。

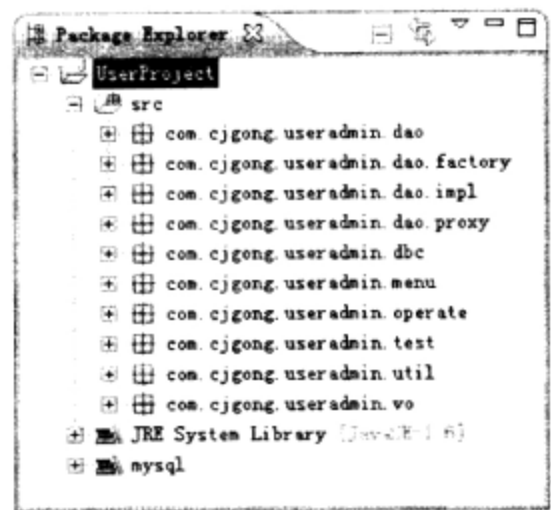


图 29.1 项目目录

29.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括初始化界面、增加用户功能、修改用户功能、查询单个用户功能、查询全部用户功能、删除用户功能和退出系统功能。

1. 初始化界面

当运行“坦克大战”项目中的 TestUserAdmin 类后，就会出现如图 29.2 所示的初始界面——人员信息管理系统界面，数据库中名为 users 表格的初始化数据如图 29.3 所示。

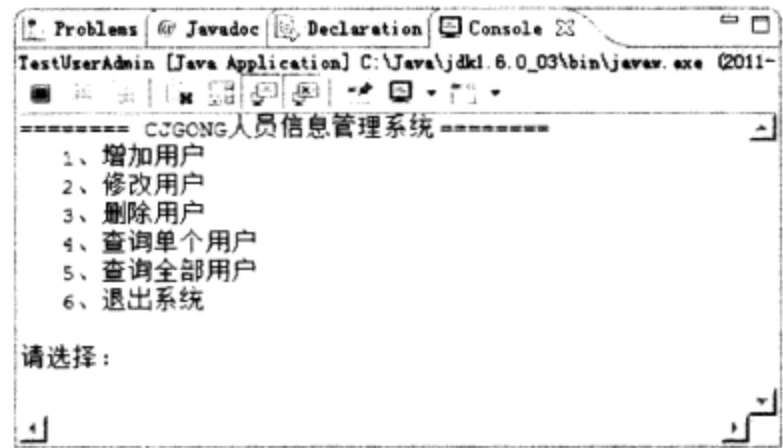


图 29.2 初始化界面

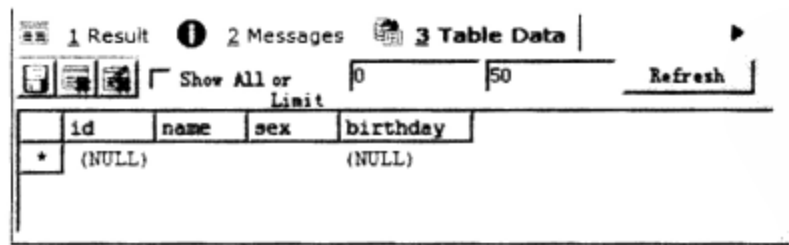


图 29.3 初始化数据

2. 增加用户的功能

在初始化界面中，如果想增加用户，首先需要选择相应菜单（数字 1），然后按下 Enter 键就会显示出相应的项。每次输入相应的信息后，需要按下 Enter 键。输入完成后，再次按下 Enter 键就会实现添加用户功能并显示出默认菜单，具体过程如图 29.4 所示，而这时

数据库中名为 users 表格的初始化数据如图 29.5 所示。

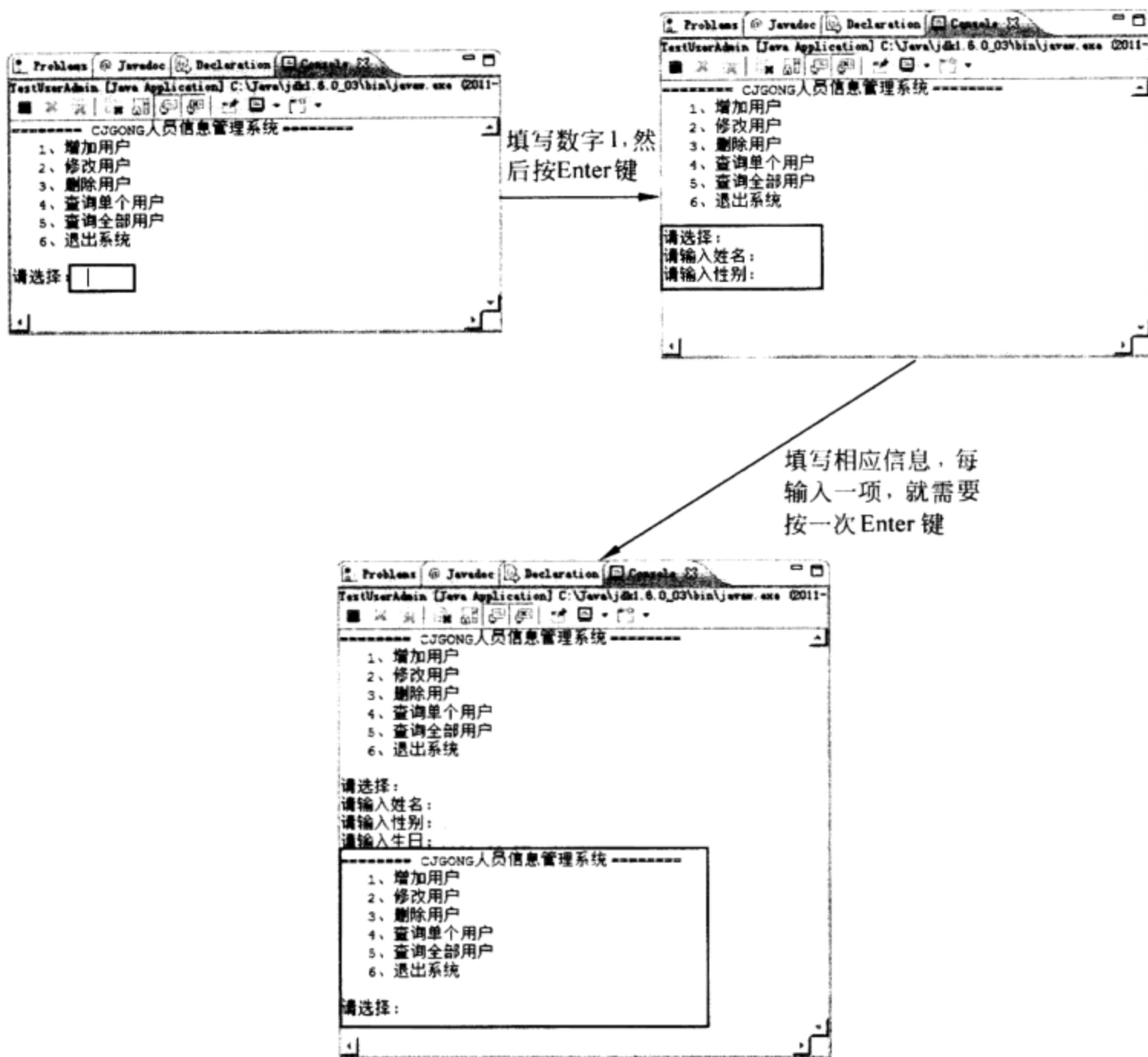


图 29.4 增加用户过程

1 Result 2 Messages 3 Table Data				
Show All or Limit 0 50 Refresh				
id	name	sex	birthday	
1	cjgong	man	2010-01-12	
(NULL)			(NULL)	

图 29.5 users 表格数据

最后再次添加一条记录，数据库 users 表格的数据如图 29.6 所示。

2 Messages 3 Table Data 4 Objects				
Show All or Limit 0 50 Refresh				
id	name	sex	birthday	
1	cjgong	man	2010-01-12	
2	cjgong2	sex	2010-01-24	
(NULL)			(NULL)	

图 29.6 users 表格最后的数据

4. 查询单个用户的功能

在初始化界面中, 如果想查询单个用户信息, 首先需要选择相应菜单(数字 4), 然后按 Enter 键就会显示“请输入要查询的编号:”项, 这时输入所需修改记录的编号(编号 2)同时按 Enter 键, 就会出现该记录相应字段的值并显示出默认菜单。具体过程如图 29.9 所示。



图 29.9 查询单个用户的过程

5. 查询全部用户的功能

在初始化界面中, 如果想查询全部用户信息, 首先需要选择相应菜单(数字 5), 然后按 Enter 键就会显示“请输入要查询的关键字:”项, 这时输入关键字(cjgong)同时按 Enter 键, 就会出现所包含该关键字的所有记录, 具体过程如图 29.10 所示。

6. 删除用户的功能

在初始化界面中, 如果想查询全部用户信息, 首先需要选择相应菜单(数字 3), 然后按 Enter 键就会显示“请输入要删除的用户编号:”项, 这时输入编号(1)同时按 Enter 键, 就会删除该编号的用户, 具体过程如图 29.11 所示, 而这时数据库中名为 users 表格的数据如图 29.12 所示。

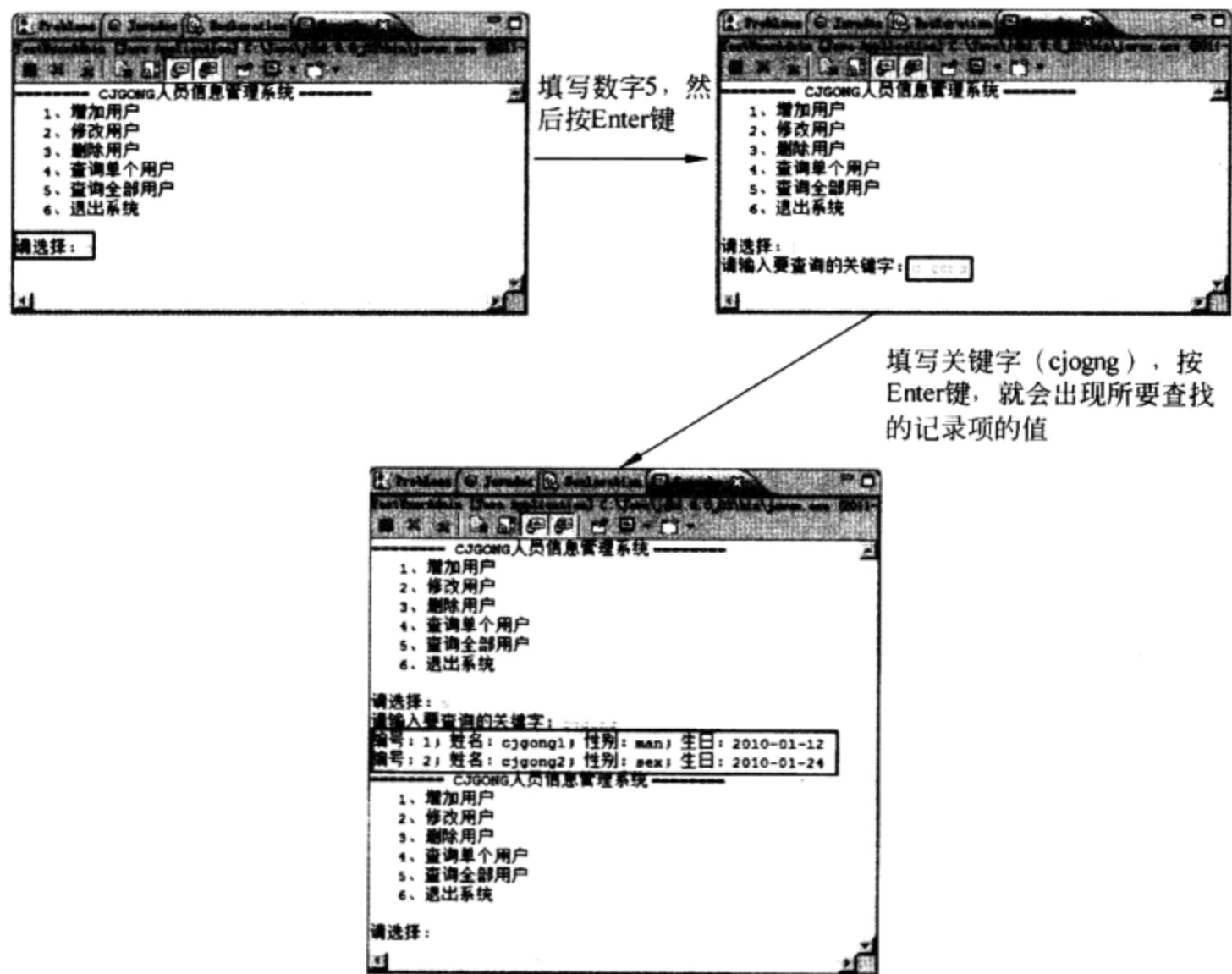


图 29.10 查询全部用户的过程

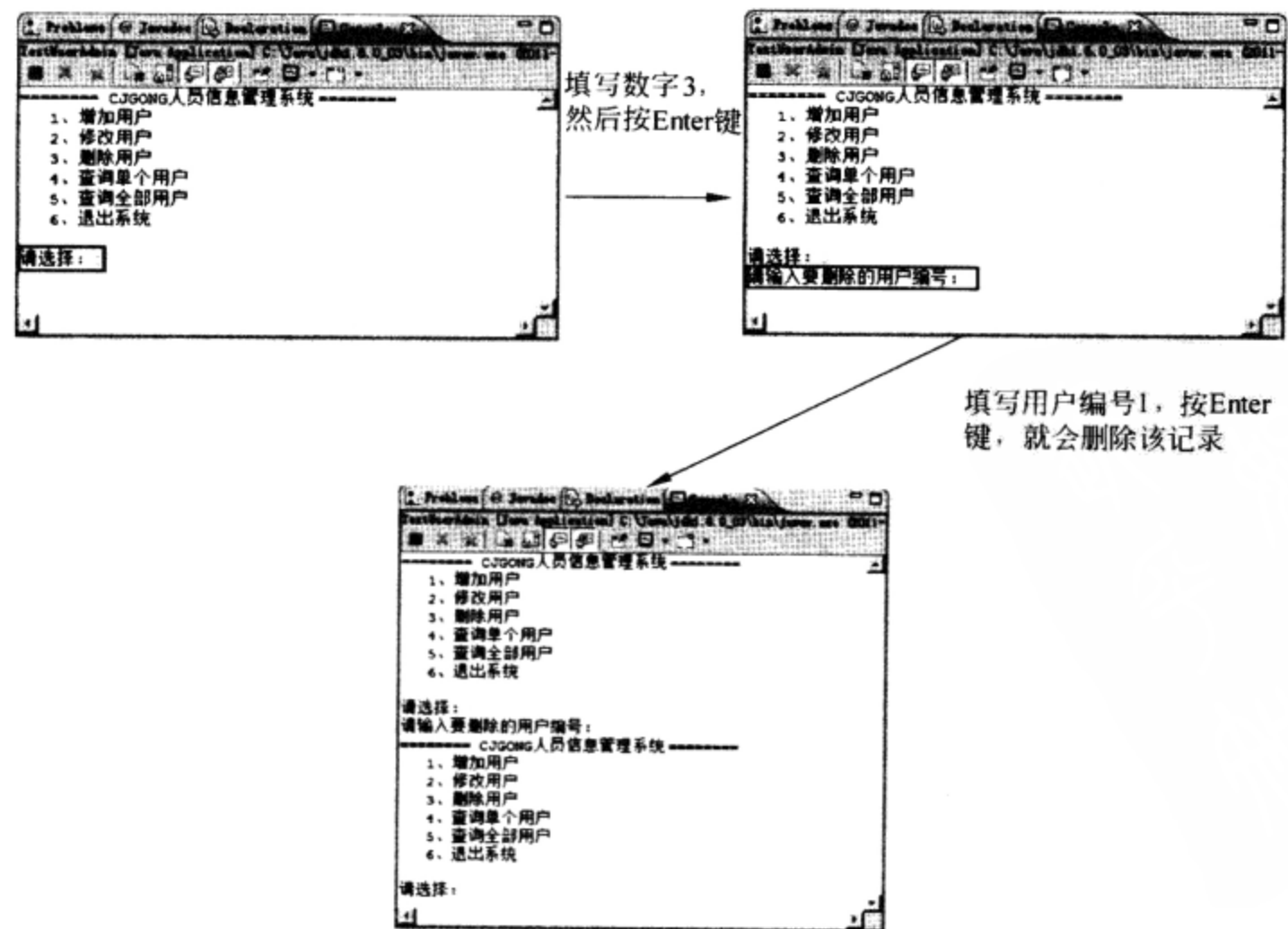
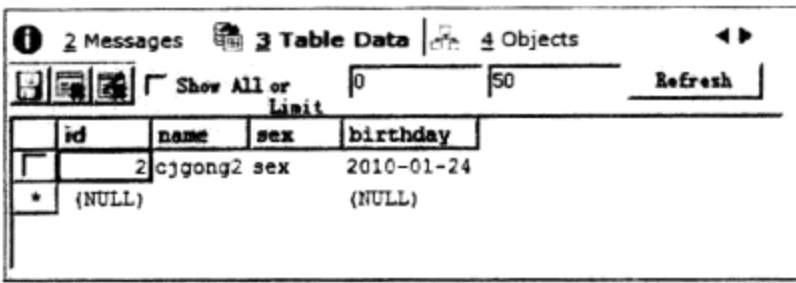


图 29.11 删除用户的过程



	id	name	sex	birthday
	2	cjgong2	sex	2010-01-24
*	{NULL}			{NULL}

图 29.12 users 表格数据

7. 退出系统的功能

在初始化界面中，如果想退出系统，首先需要选择相应菜单（数字 6），然后按 Enter 键就会实现退出系统的功能，具体过程如图 29.13 所示。

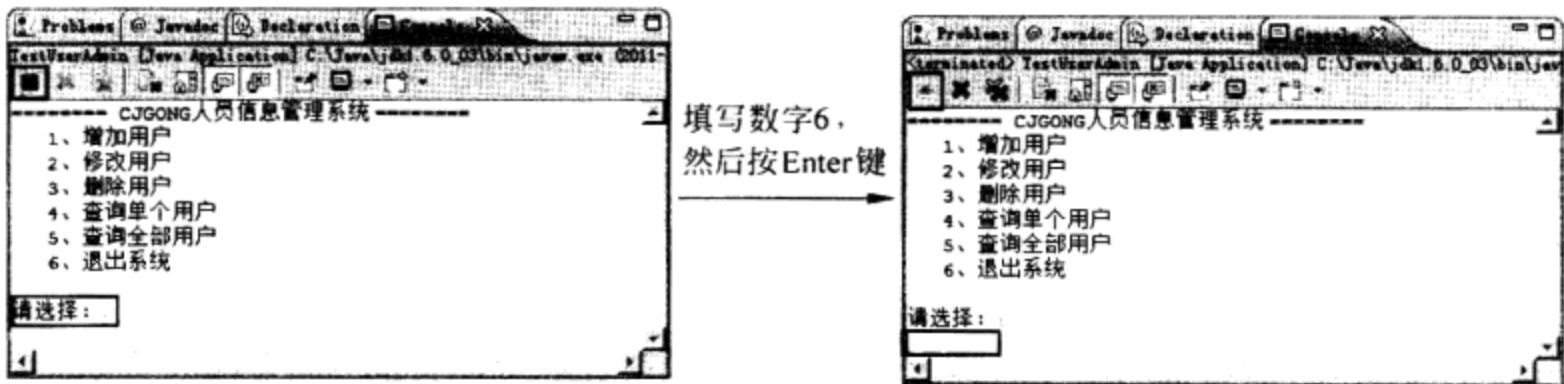


图 29.13 退出系统的过程

29.2 人员信息管理项目前期准备

本节除了将详细介绍如何设计人员信息管理项目的数据库和表外，还将实现该系统的环境搭建，其中 MySQL 数据库的版本号为 MySQL 5.0。

29.2.1 设计数据库

人员信息管理项目需要建立一个数据库并在该数据库中建立一张表，存放表的数据库 users 和存放用户信息的表 user。

1. 创建数据库users

如果想创建出数据库 users，可以在 MySQL 的命令窗口中输入如下命令：

```
CREATE DATABASE 'users'
```

2. 创建表user

如果想创建出表 user，可以在 MySQL 的命令窗口中输入相关命令。该表的具体信息如表 29.1 所示。

表 29.1 表user信息

字 段 名 称	数 据 类 型	字 段 说 明
id	int	编号
name	varchar(50)	用户名
sex	varchar(10)	性别
birthday	date	生日

在具体创建接口类时会涉及谁要操作对象——用户信息，所以需要将表中的字段进行抽象设计出一个名为 User 的类，该类主要表示用户的信息，具体内容如代码 29.1 所示，该类的 UML 如图 29.14 所示。

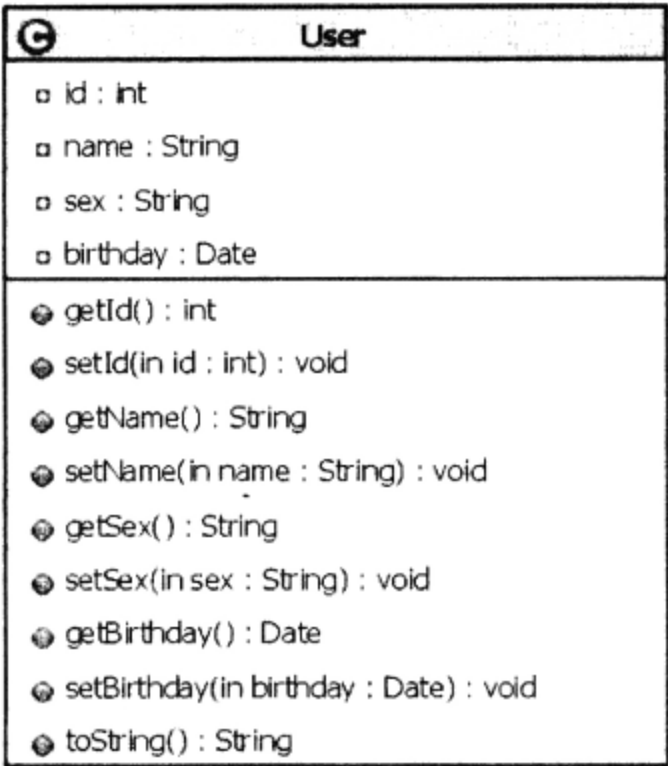


图 29.14 用户类图

代码 29.1 用户信息类：User.java

```
public class User {
    //创建字段
    private int id;
    private String name;
    private String sex;
    private Date birthday;
    //属性id的getter和setter方法
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    //属性name的getter和setter方法
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

//用户编号属性
 //用户姓名属性
 //用户性别属性
 //用户生日属性

```

//属性 sex 的 getter 和 setter 方法
public String getSex() {
    return sex;
}
public void setSex(String sex) {
    this.sex = sex;
}
//属性 birthday 的 getter 和 setter 方法
public Date getBirthday() {
    return birthday;
}
public void setBirthday(Date birthday) {
    this.birthday = birthday;
}
@Override
public String toString() {                //重写 toString() 方法
    return "编号: " + this.id + "; 姓名: " + this.name + "; 性别: " + this.sex
        + "; 生日: " + this.birthday;
}
}

```

【代码解析】

在具体设计 User 对象信息类时, 该类的所有属性都是按照数据库 users 中的 user 表字段设计出来的。其中属性 id 与表格字段 id 相对应; 属性 name 与表格字段 name 相对应; 属性 sex 与表格字段 sex 相对应; 属性 birthday 与表格字段 birthday 相对应。

29.2.2 数据库操作相关类

在人员信息管理项目中, 为了方便操作数据库, 专门创建了一个名为 DataBaseConnection 的连接数据库类, 该类的具体内容如代码 29.2 所示, 该类的 UML 如图 29.15 所示。

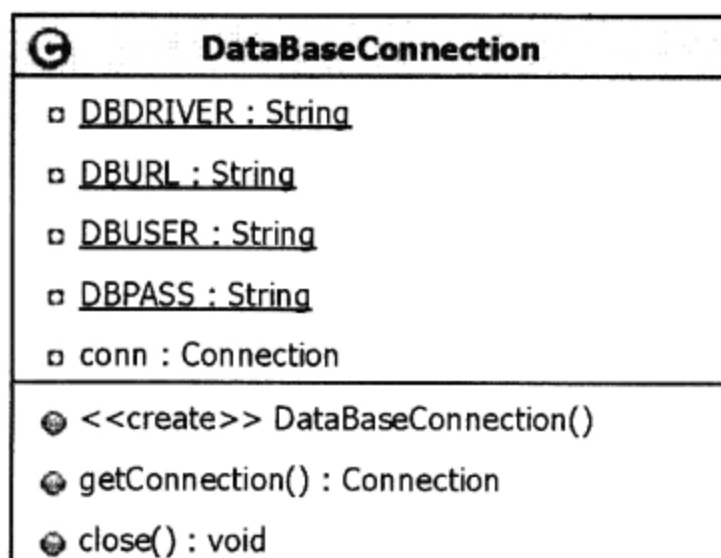


图 29.15 数据库连接类图

代码 29.2 数据库连接类: DataBaseConnection.java

```

public class DataBaseConnection {
    //创建成员变量
    //创建数据库连接驱动的成员变量

```



```

private static final String DBDRIVER = "org.gjt.mm.mysql.Driver";
//创建数据库 URL 成员变量
private static final String DBURL = "jdbc:mysql://localhost:3306/Users";
//创建数据库连接用户名成员变量
private static final String DBUSER = "root";
//创建数据库连接密码成员变量
private static final String DBPASS = "root";
private Connection conn = null;                                //创建数据库连接成员变量
public DataBaseConnection() {                                  //构造函数
    try {
        Class.forName(DBDRIVER);                              //加载驱动
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
    try {
        //获取连接
        conn = DriverManager.getConnection(DBURL, DBUSER, DBPASS);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
public Connection getConnection() {                            //获取连接方法
    return this.conn;
}
public void close() {                                          //关闭数据库连接
    if (this.conn != null) {                                    //判断变量 conn 的值
        try {
            this.conn.close();                                  //关闭连接
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
}

```

【代码解析】

在上述代码中，首先创建了 5 个成员变量，DBDRIVER 变量表示连接数据库驱动；DBURL 变量表示连接数据库的 URL 地址；DBUSER 变量表示连接数据库的用户名；DBPASS 变量表示连接数据库的密码；Connection 变量表示数据库的连接对象。然后在构造函数中设置 Connection 对象的值；最后设置了关闭数据库连接的 close()方法。

29.3 人员信息管理项目——DAO 层

本节将详细介绍如何设计人员信息管理项目的 DAO 层，在该层中首先设计一个 IUserDAO 接口，然后创建实现接口的 IUserDAOImpl 类。为了使程序更具可读性，使用了 IUserDAO 接口的代理模式，创建了代理类 IUserDAOProxy；同时使用了 IUserDAO 接口及实现类 IUserDAOProxy 的工厂模式，创建了工厂类 DAOFactory。上述类的关系如图 29.16 所示。

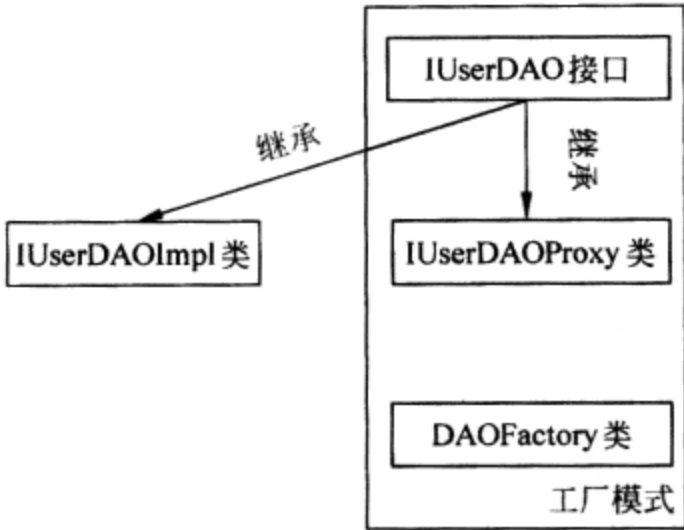


图 29.16 程序关系图

29.3.1 实现数据连接操作（DAO）的接口

在进行任何操作的时候首先需要完成的就是接口，在这些接口中主要定义操作数据的各种方法，即该项目中关于用户操作的增加、删除、修改和查找等方法。

IUserDAO 为人员信息管理项目中的接口类，该类主要定义用户的一些操作。IUserDAO 类的具体内容如代码 29.3 所示，该类的 UML 如图 29.17 所示。

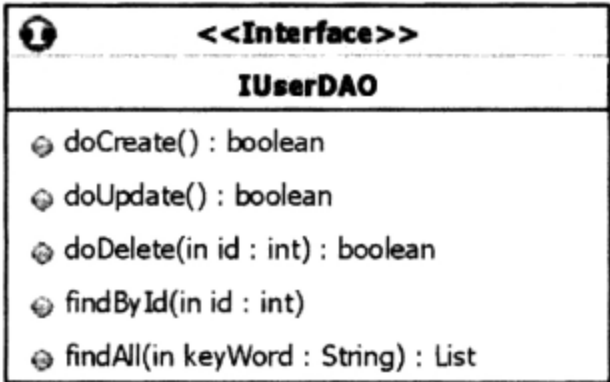


图 29.17 IUserDAO 接口的类图

代码 29.3 用户信息操作接口：IUserDAO.java

```
public interface IUserDAO {
    //表示数据库的增加操作
    public boolean doCreate(User user) throws Exception;
    //表示数据库的更新操作
    public boolean doUpdate(User user) throws Exception;
    //表示删除操作，按编号删除
    public boolean doDelete(int id) throws Exception;
    //表示数据库的查询操作
    public User findById(int id) throws Exception;
    //表示数据库的查询，查询的时候将返回一组对象
    List<User> findAll(String keyWord) throws Exception;
}
```

【代码解析】

□ 在具体设计接口名字时，最好能够让接口名称与要操作的代码进行关联。由于所操作的对象为 user 表格，所以名字为 IUserDAO，其中 I 表示接口的意思，DAO

为数据库操作对象，表示的是操作数据。

- 在具体设计接口中的方法名时，凡是数据库更新操作方法都是 doXxx()格式形式，凡是涉及数据库操作的方法都是 findXxx()格式形式。

29.3.2 实现数据连接操作（DAO）的实现类

创建好接口后，实际上就是制定了对数据库 users 中表 user 的一个完整操作标准。设计好接口后，然后就需要实现该接口，实现 IUserDAOImpl 类的具体内容如代码 29.4 所示，该类的 UML 如图 29.18 所示。

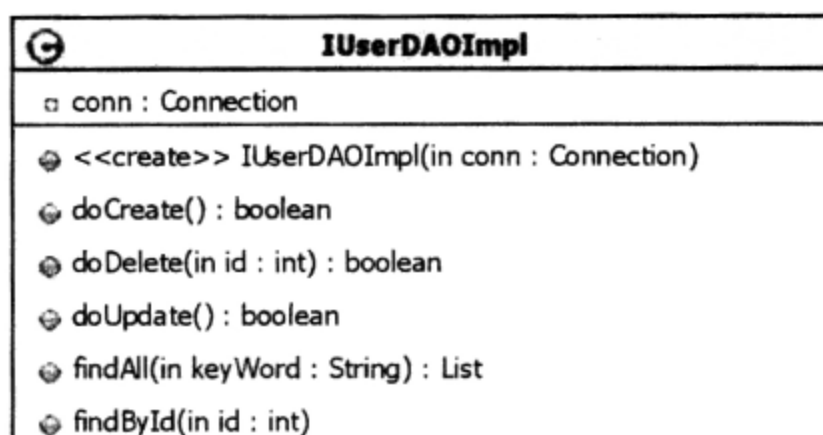


图 29.18 IUserDAOImpl 类的类图

代码 29.4 实现接口类：IUserDAOImpl.java

```

public class IUserDAOImpl implements IUserDAO {
    private Connection conn = null; //创建数据库连接字段
    public IUserDAOImpl(Connection conn) { //构造函数
        this.conn = conn;
    }
    //实现 doCreate() 方法
    public boolean doCreate(User user) throws Exception {
        boolean flag = false;
        PreparedStatement pstmt = null;
        String sql = "INSERT INTO user(name,sex,birthday) VALUES (?, ?, ?) ";
        try {
            pstmt = this.conn.prepareStatement(sql);
            pstmt.setString(1, user.getName()); //从user类中取出所有的内容
            pstmt.setString(2, user.getSex()); //从user类中取出所有的内容
            pstmt.setDate(3, new java.sql.Date(user.getBirthday().getTime()));
            if (pstmt.executeUpdate() > 0) { //至少已经更新了一行
                flag = true;
            }
        } catch (Exception e) {
            throw e;
        }
        //不管如何抛出，最终肯定是要进行数据库的关闭操作的
    } finally {
        if (pstmt != null) {
            try {
                pstmt.close();
            } catch (Exception e1) {
            }
        }
    }
}
  
```

```

        return flag;
    }
    //实现 doDelete() 方法
    public boolean doDelete(int id) throws Exception {
        boolean flag = false;
        PreparedStatement pstmt = null;
        String sql = "DELETE FROM user WHERE id=? ";
        try {
            pstmt = this.conn.prepareStatement(sql);
            pstmt.setInt(1, id); //从 user 类中取出所有的内容
            if (pstmt.executeUpdate() > 0) { //至少已经更新了一行
                flag = true;
            }
        } catch (Exception e) {
            throw e;
        }
        //不管如何抛出, 最终肯定是要进行数据库的关闭操作的
        finally {
            if (pstmt != null) {
                try {
                    pstmt.close();
                } catch (Exception e1) {
                }
            }
        }
        return flag;
    }
    //实现 doUpdate() 方法
    public boolean doUpdate(User user) throws Exception {
        boolean flag = false;
        PreparedStatement pstmt = null;
        String sql = "UPDATE user SET name=?,sex=?,birthday=? WHERE id=?";
        try {
            pstmt = this.conn.prepareStatement(sql);
            pstmt.setString(1, user.getName()); //从 user 类中取出所有的内容
            pstmt.setString(2, user.getSex()); //从 user 类中取出所有的内容
            pstmt.setDate(3, new java.sql.Date(user.getBirthday().getTime()));
            pstmt.setInt(4, user.getId());
            if (pstmt.executeUpdate() > 0) { //至少已经更新了一行
                flag = true;
            }
        } catch (Exception e) {
            throw e;
        }
        //不管如何抛出, 最终肯定是要进行数据库的关闭操作的
        finally {
            if (pstmt != null) {
                try {
                    pstmt.close();
                } catch (Exception e1) {
                }
            }
        }
        return flag;
    }
    //实现 findAll() 方法
    public List<User> findAll(String keyWord) throws Exception {
        List<User> all = new ArrayList<User>();
        PreparedStatement pstmt = null;
        String sql = "SELECT id,name,sex,birthday FROM user WHERE name LIKE ? OR sex LIKE ? OR birthday LIKE ?";
    }

```

```

    try {
        pstmt = this.conn.prepareStatement(sql);
        pstmt.setString(1, "%" + keyWord + "%");
        pstmt.setString(2, "%" + keyWord + "%");
        pstmt.setString(3, "%" + keyWord + "%");
        ResultSet rs = pstmt.executeQuery();    //执行查询操作
        while (rs.next()) {
            User user = new User();
            user.setId(rs.getInt(1));
            user.setName(rs.getString(2));
            user.setSex(rs.getString(3));
            user.setBirthday(rs.getDate(4));
            all.add(user);    //所有的内容向集合中插入
        }
        rs.close();
    } catch (Exception e) {
        throw e;
    }
    //不管如何抛出，最终肯定是要进行数据库的关闭操作的
    finally {
        if (pstmt != null) {
            try {
                pstmt.close();
            } catch (Exception e1) {
            }
        }
    }
    return all;
}

//实现 findById() 方法
public User findById(int id) throws Exception {
    User user = null;
    PreparedStatement pstmt = null;
    String sql = "SELECT id,name,sex,birthday FROM user WHERE id=?";
    try {
        pstmt = this.conn.prepareStatement(sql);
        pstmt.setInt(1, id);
        ResultSet rs = pstmt.executeQuery();    //执行查询操作
        if (rs.next()) {
            user = new User();
            user.setId(rs.getInt(1));
            user.setName(rs.getString(2));
            user.setSex(rs.getString(3));
            user.setBirthday(rs.getDate(4));
        }
        rs.close();
    } catch (Exception e) {
        throw e;
    }
    //不管如何抛出，最终肯定是要进行数据库的关闭操作的
    finally {
        if (pstmt != null) {
            try {
                pstmt.close();
            } catch (Exception e1) {
            }
        }
    }
    return user;
}
}
}

```

【代码解析】

在上述代码中，首先创建了成员变量 conn，然后在构造函数中初始化该变量，最后分别实现接口 IUserDAO 中的各个方法。在具体实现接口中的方法时，首先创建相关的 SQL 语句，然后通过连接对象 conn 的 prepareStatement()方法，获取上述 SQL 语句的 PreparedStatement 对象 pstmt，同时通过对象 pstmt 的 setString()方法设置 SQL 语句中的参数，最后通过对象 pstmt 的各种方法实现增加、删除、查找和修改功能。

29.3.3 实现数据连接操作（DAO）的代理类

在具体设计人员信息管理项目时，接口实现类的主要功能就是完成数据库的具体操作，但是对于数据库的打开和关闭操作与具体的业务操作时没有任何关系。

为了解决这些问题，可以通过代理设计模式来实现。即所有数据库的打开和连接由代理类 IUserDAOProxy 完成，而具体的操作由接口实现类来完成。IUserDAOProxy 类的具体内容如代码 29.5 所示，该类的 UML 如图 29.19 所示。

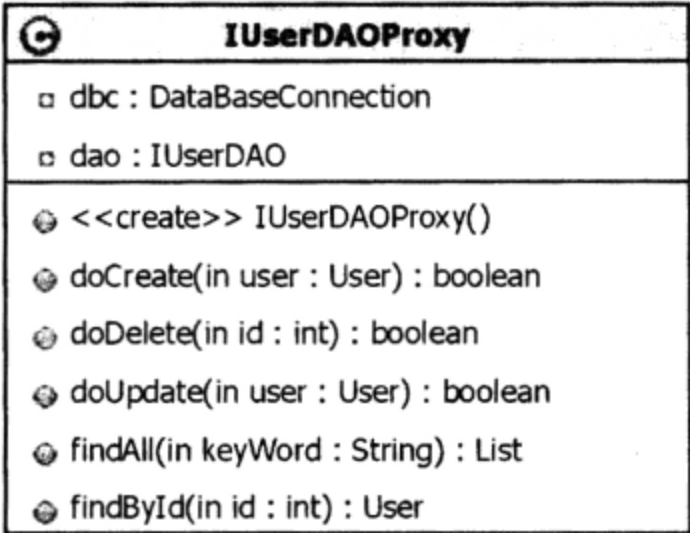


图 29.19 IUserDAOProxy 类的类图

代码 29.5 代理类：IUserDAOProxy.java

```
public class IUserDAOProxy implements IUserDAO {
    private DataBaseConnection dbc = null; //创建数据库工具类对象
    private IUserDAO dao = null; //创建接口实现类对象
    public IUserDAOProxy() { //构造函数
        this.dbc = new DataBaseConnection(); //初始化对象 dbc
        //初始化对象 dao
        this.dao = new IUserDAOImpl(this.dbc.getConnection());
    }
    @Override
    public boolean doCreate(User user) throws Exception { //实现 doCreate() 方法
        //创建标记 flag
        boolean flag = true;
        try {
            flag = this.dao.doCreate(user); //调用 dao.doCreate() 方法
        } catch (Exception e) {
            throw e;
        } finally {
            this.dbc.close(); //实现关闭功能
        }
    }
}
```



```

    }
    return flag;
}
@Override
public boolean doDelete(int id) throws Exception {
    //实现 doDelete() 方法
    //创建标记 flag
    boolean flag = true;
    try {
        flag = this.dao.doDelete(id); //调用 dao. doDelete() 方法
    } catch (Exception e) {
        throw e;
    } finally {
        this.dbc.close(); //实现关闭功能
    }
    return flag;
}
@Override
public boolean doUpdate(User user) throws Exception {
    //实现 doUpdate() 方法
    //创建标记 flag
    boolean flag = true;
    try {
        flag = this.dao.doUpdate(user); //调用 dao. doUpdate() 方法
    } catch (Exception e) {
        throw e;
    } finally {
        this.dbc.close(); //实现关闭功能
    }
    return flag;
}
@Override
public List<User> findAll(String keyWord) throws Exception {
    //实现 findAll() 方法
    //创建对象 all
    List<User> all = null;
    try {
        all = this.dao.findAll(keyWord); //调用 dao.findAll() 方法
    } catch (Exception e) {
        throw e;
    } finally {
        this.dbc.close(); //实现关闭功能
    }
    return all;
}
@Override
public User findById(int id) throws Exception {
    //实现 findById() 方法
    //创建对象 user
    User user = null;
    try {
        user = this.dao.findById(id); //调用 dao.findById() 方法
    } catch (Exception e) {
        throw e;
    } finally {
        this.dbc.close(); //实现关闭功能
    }
    return user;
}
}
}

```

【代码解析】

在上述代码中,首先创建了两个成员变量:DataBaseConnection 类对象 dbc 和 IUserDAO 类对象 dao,然后在构造函数中初始化这两个成员变量值,最后再具体实现接口 IUserDAO 中的各种方法。在具体实现接口方法时,通过调用对象 dao 中的相应方法来实现,最后通过对象 dbc 的 close()方法实现关闭数据库连接的功能。

29.3.4 实现数据连接操作 (DAO) 的工厂类

创建好接口并实现接口后,为了能够便于操作,专门根据工厂设计模式创建了一个名为 DAOFactory 的工厂类,该类主要在项目中实现耦合的操作。DAOFactory 类的具体内容如代码 29.6 所示,该类的 UML 如图 29.20 所示。

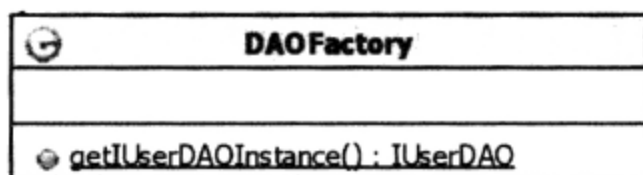


图 29.20 DAOFactory 类的类图

代码 29.6 工厂类: DAOFactory.java

```
public class DAOFactory {
    public static IUserDAO getUserDAOInstance() { //创建静态方法
        return new IUserDAOProxy();             //返回 IUserDAO 类对象
    }
}
```

【代码解析】

在上述代码中,首先创建了静态方法 getUserDAOInstance(),该方法主要返回 IUserDAO 类对象。在具体实现返回实例方法时,主要通过代理类 IUserDAOProxy()来实现。

29.4 人员信息管理项目——服务层和表示层

本节除了介绍如何设计人员信息管理项目的服务层外,还详细介绍了该项目的表示层。对于“人员信息管理”项目的服务层,主要实现该项目的业务逻辑,而该项目的表示层则是通过模拟菜单来实现。

29.4.1 人员信息管理项目的服务层

在具体实现人员信息管理项目服务层的 UserOperate 类时,主要实现用户的 4 种方法:增加用户方法、更新用户方法、删除用户方法、根据 ID 实现查找方法和查找所有用户方法。UserOperate 类的具体内容如代码 29.7 所示,该类的 UML 如图 29.21 所示。

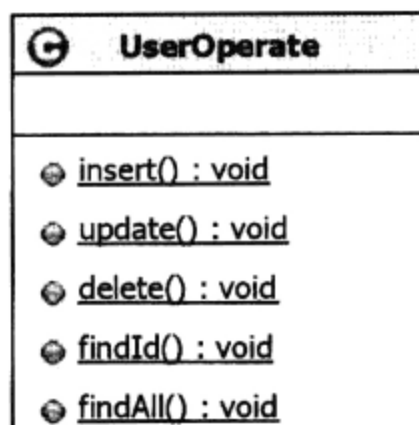


图 29.21 UserOperate 类的类图

代码 29.7 用户信息的操作：UserOperate.java

```

public class UserOperate {
    public static void insert() {                                //添加用户操作方法
        User user = new User();                                  //创建用户对象 user
        InputData input = new InputData();                      //创建 InputData 对象
        //设置用户的姓名
        user.setName(input.getString("请输入姓名: "));
        //设置用户的性别
        user.setSex(input.getString("请输入性别: "));
        //设置用户的生日
        user.setBirthday(input.getDate("请输入生日: ", "内容必须是日期 (yyyy-
        mm-dd), "));
        try {
            //实现添加用户的功能
            DAOFactory.getIUserDAOInstance().doCreate(user);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void update() {                                //更新用户信息
        InputData input = new InputData();                      //创建 InputData 对象
        //获取用户编号
        int id = input.getInt("请输入要修改用户的编号: ", "编号必须是数字, ");
        User user = null;                                         //创建用户对象 user
        try {
            //查找相应 ID 的用户对象
            user = DAOFactory.getIUserDAOInstance().findById(id);
        } catch (Exception e) {
            e.printStackTrace();
        }
        if (user != null) {                                       //当对象 user 不为空时
            //设置用户姓名
            user.setName(input.getString("请输入姓名 (原姓名: " + user.getName()
            + "): "));
            //设置用户性别
            user.setSex(input.getString("请输入性别 (原性别: " + user.getSex()
            + "): "));
            //设置用户生日
            user.setBirthday(input.getDate("请输入生日 (原生日: " + user.get-
            Birthday()
            + "): ", "内容必须是日期 (yyyy-mm-dd), "));
        }
    }
}
  
```

```

        try {
            //实现更新用户功能
            DAOFactory.getIUserDAOInstance().doUpdate(user);
        } catch (Exception e) {
            e.printStackTrace();
        }
    } else {
        System.out.println("要查找的用户不存在!"); //输出相应信息
    }
}

public static void delete() { //删除用户信息
    InputData input = new InputData(); //创建 InputData 对象
    //获取用户编号
    int id = input.getInt("请输入要删除的用户编号: ", "编号必须是数字, ");
    try {
        //实现删除用户的功能
        DAOFactory.getIUserDAOInstance().doDelete(id);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public static void findId() { //根据 ID 查找用户信息
    InputData input = new InputData(); //创建 InputData 类对象
    //获取用户编号
    int id = input.getInt("请输入要查询的编号: ", "编号必须是数字, ");
    User user = null; //创建 user 对象
    try {
        //实现根据 ID 查找用户功能
        user = DAOFactory.getIUserDAOInstance().findById(id);
    } catch (Exception e) {
        e.printStackTrace();
    }
    if (user != null) {
        System.out.println(user); //输出对象 user
    } else {
        //输出相应信息
        System.out.println("要查找的用户不存在!");
    }
}

public static void findAll() { //查找所有用户信息
    InputData input = new InputData(); //创建 InputData 类对象
    //获取关键字字符串
    String keyWord = input.getString("请输入要查询的关键字: ");
    List<User> allUser = null; //创建存储 user 类型的 allUser 对象
    try {
        //实现查找所有用户的功能
        allUser = DAOFactory.getIUserDAOInstance().findAll(keyWord);
    } catch (Exception e) {
        e.printStackTrace();
    }
    Iterator<User> iter = allUser.iterator(); //获取allUser对象的迭代器
    while (iter.hasNext()) { //遍历迭代器
        User user = iter.next();
        System.out.println(user); //输出对象 user
    }
}
}
}

```

【代码解析】

- 在上述代码中主要实现了人员信息管理项目的业务逻辑：实现添加用户的方法 `insert()`；实现更新用户信息的方法 `update()`；实现删除用户信息的方法 `delete()`；实现根据 ID 实现查找用户方法 `findById()`和实现查找所有用户方法 `findAll()`。
- 在具体实现各种业务逻辑方法时，首先通过工厂类 `DAOFactory` 的 `getIUserDAO-Instance()`方法获取代理类 `IUserDAOProxy` 对象，然后调用代理类的相应方法实现相应的业务功能。

29.4.2 人员信息管理项目的表示层

在具体实现人员信息管理项目表示层的类 `Menu` 时，主要通过 `System.out.println()`方法输出相应信息来模拟菜单，同时通过“switch—case”分支结构调用相应的业务方法，`Menu` 类的具体内容如代码 29.8 所示，该类的 UML 如图 29.22 所示。

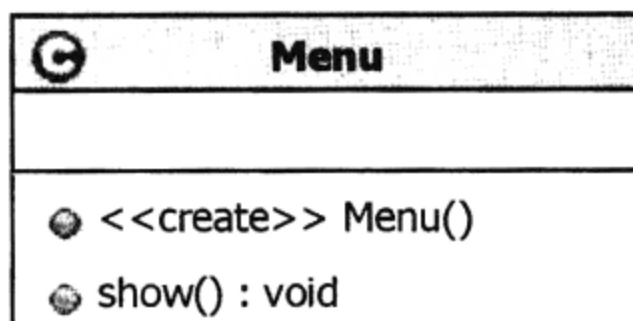


图 29.22 Menu 类的类图

代码 29.8 表示类：Menu.java

```

public class Menu {
    public Menu() {
        while (true) {
            this.show();
        }
    }
    public void show() {
        //输出相应信息
        System.out.println("===== CJGONG 人员信息管理系统 =====");
        System.out.println("  1、增加用户");
        System.out.println("  2、修改用户");
        System.out.println("  3、删除用户");
        System.out.println("  4、查询单个用户");
        System.out.println("  5、查询全部用户");
        System.out.println("  6、退出系统");
        InputData input = new InputData();
        //获取功能编号变量 ch
        int ch = input.getInt("\n 请选择：", "请输入正确的选项，");
        switch (ch) {
            case 1: {
                UserOperate.insert();
                break;
            }
            case 2: {
                //构造函数
                //死循环
                //调用 show() 方法
                //实现显示方法
                //创建 InputData 对象
                //判断变量 ch 的值
                //调用添加用户的方法
            }
        }
    }
}
  
```

```
        UserOperate.update(); //调用更新用户信息的方法
        break;
    }
    case 3: {
        UserOperate.delete(); //调用删除用户信息的方法
        break;
    }
    case 4: {
        UserOperate.findId(); //调用根据 ID 查找用户的方法
        break;
    }
    case 5: {
        UserOperate.findAll(); //调用查找所有用户的方法
        break;
    }
    case 6: {
        System.exit(1); //实现退出功能
        break;
    }
    default: {
        System.out.println("请选择正确的选项！"); //输出相应信息
        break;
    }
}
}
```

【代码解析】

在上述代码中首先通过输出语句输出各种功能的信息，然后通过分支结构判断用户所输入的值，从而调用相应的业务功能。

29.4.3 工具类

为了便于开发人员信息管理项目，专门创建一个名为 `InputData` 的类，该类主要实现格式化输入字符串功能，`InputData` 类的具体内容如代码 29.9 所示，该类的 UML 如图 29.23 所示。

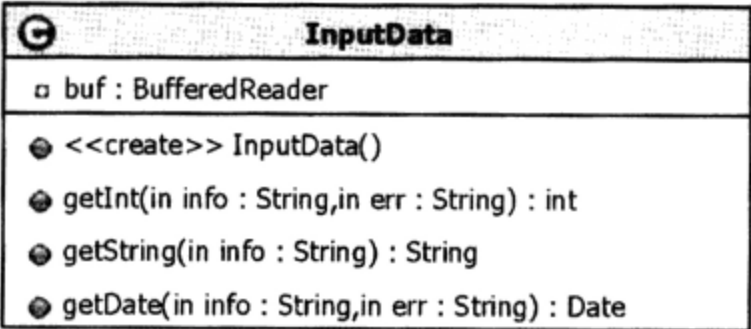


图 29.23 `InputData` 类的类图

代码 29.9 数据格式的类: `InputData.java`

```
public class InputData {
    private BufferedReader buf = null; //创建带有缓存功能的输入流对象 buf
    public InputData() { //构造函数
        //为对象 buf 赋值
    }
}
```



```

        this.buf = new BufferedReader(new InputStreamReader(System.in));
    }
    public int getInt(String info, String err) {           //获取整型数据
        int i = 0;
        boolean flag = true;
        while (flag) {
            String str = this.getString(info);
            if (str.matches("\\d+")) {
                i = Integer.parseInt(str);
                flag = false;
            } else {
                System.out.print(err);
            }
        }
        return i;
    }
    public String getString(String info) {                 //获取字符串
        String str = null;
        System.out.print(info);
        try {
            str = this.buf.readLine();
        } catch (IOException e) {
        }
        return str;
    }
    public Date getDate(String info, String err) {         //获取时间
        Date date = null;
        boolean flag = true;
        while (flag) {
            String str = this.getString(info);
            if (str.matches("\\d{4}-\\d{2}-\\d{2}")) {
                try {
                    date = new SimpleDateFormat("yyyy-MM-dd").parse(str);
                    flag = false;
                } catch (ParseException e) {
                }
            } else {
                System.out.print(err);
            }
        }
        return date;
    }
}

```

【代码解析】

在上述代码中主要实现把输入的字符串转换成各种类型数据，其中 `getInt()` 方法主要实现把字符串类型转换成整型类型；`getString()` 方法主要实现把字符串类型转换成字符串类型；`getDate()` 方法主要实现把字符串类型转换成时间类型。

29.5 人员信息管理项目——代理类测试

在具体编写 `IUserDAOProxy` 时，该类中的各种方法主要实现了“人员信息管理”项目中的业务逻辑。为了便于测试 `IUserDAOProxy` 类中的业务逻辑方法，专门创建了一个存储各个测试类的名为 `com.cjgong.useradmin.dao.proxy` 的包。

29.5.1 测试实现业务功能的各种方法

在人员信息管理项目中，IUserDAOProxy 类主要用来实现该项目的业务功能。对于一个项目，最主要的就是业务功能的代码是正确的，即测试业务功能的方法正确。所需要测试的方法如图 29.24 所示。

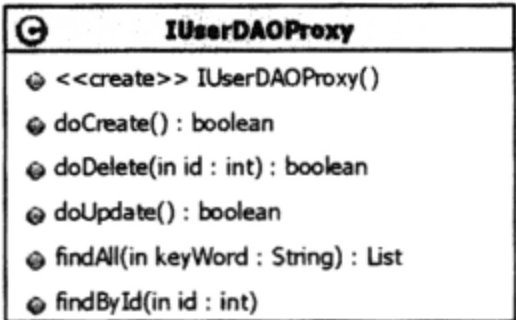


图 29.24 业务逻辑方法

1. 测试findById()方法

创建测试方法 findById()的 TestId 类，该类的具体内容如代码 29.10 所示。

代码 29.10 测试根据编号查找方法：TestId.java

```
public class TestId {
    public static void main(String[] args) throws Exception {
        //调用 IUserDAOProxy 类中的 findById() 方法
        User user = DAOFactory.getIUserDAOInstance().findById(2);
        //输出对象 user 信息
        System.out.println(user);
    }
}
```

【运行结果】

当出现如图 29.25 所示的运行结果时，则表明 IUserDAOProxy 类中的 findById()方法是正确的。

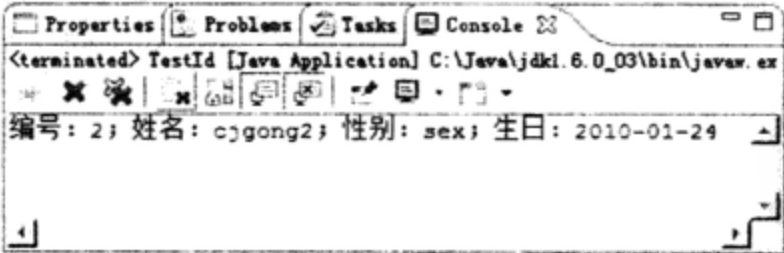


图 29.25 运行结果

2. 测试findAll()方法

创建测试方法 findAll()的 TestAll 类，该类的具体内容如代码 29.11 所示。

代码 29.11 测试查找所有用户方法：TestAll.java

```
public class TestAll {
    public static void main(String[] args) throws Exception {
```

```
//调用 IUserDAOProxy 类中的 findAll() 方法
List<User> allUser = DAOFactory.getUserDAOInstance().findAll("");
Iterator<User> iter = allUser.iterator(); //获取对象allUser 的迭代器
while (iter.hasNext()) { //遍历迭代器
    User user = iter.next();
    System.out.println(user); //输出对象 user
}
}
```

【运行结果】

当出现如图 29.26 所示的运行结果时，则表明 IUserDAOProxy 类中的 findAll()方法是正确的。

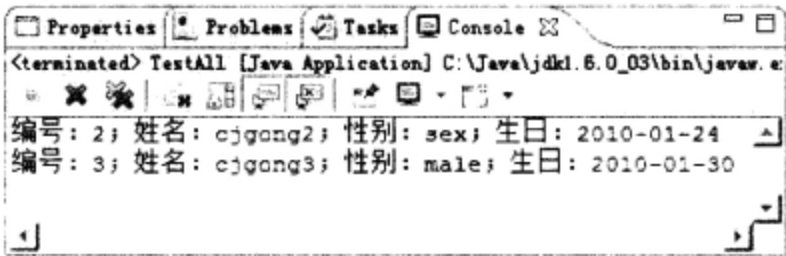


图 29.26 运行结果

3. 测试doDelete()方法

创建测试方法 doDelete()的 TestDelete 类，该类的具体内容如代码 29.12 所示。

代码 29.12 测试删除方法：TestDelete.java

```
public class TestDelete {
    public static void main(String[] args) throws Exception {
        //调用 IUserDAOProxy 类中的 doDelete() 方法
        DAOFactory.getUserDAOInstance().doDelete(2);
    }
}
```

【运行结果】

当运行上述代码后，则数据库 users 中的表格 user 的记录变化过程如图 29.27 所示。

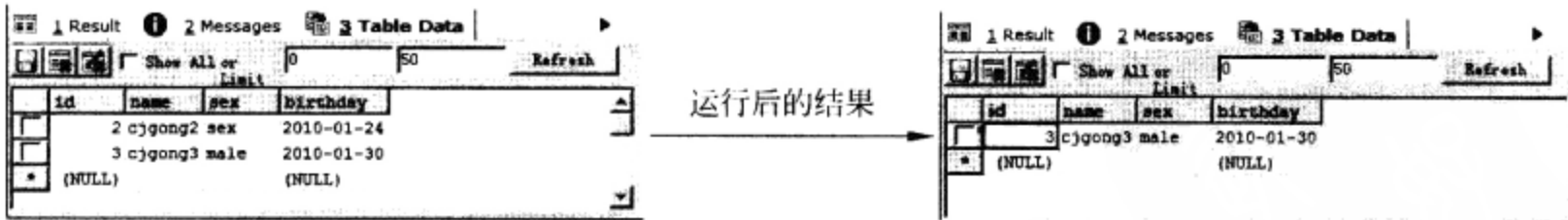


图 29.27 数据库表格变化

4. 测试doCreate()方法

创建测试方法 doCreate()的 TestInsert 类，该类的具体内容如代码 29.13 所示。

代码 29.13 测试添加用户方法：TestInsert.java

```
public class TestInsert {
    public static void main(String[] args) throws Exception {
```

```

        User user = new User();           //创建对象 user
        user.setName("常建功");           //设置用户名
        user.setSex("male");              //设置性别
        user.setBirthday(new Date());      //设置时间
        //调用 IUserDAOProxy 类中的 doCreate() 方法
        DAOFactory.getUserDAOInstance().doCreate(user);
    }
}

```

【运行结果】

当运行上述代码后，则数据库 users 中的表格 user 的记录变化过程如图 29.28 所示。

运行后的结果

id	name	sex	birthday
3	cjgong3	male	2010-01-30
(NULL)	(NULL)	(NULL)	(NULL)

id	name	sex	birthday
3	cjgong3	male	2010-01-30
4	常建功	male	2011-03-07
(NULL)	(NULL)	(NULL)	(NULL)

图 29.28 数据库表格变化

5. 测试doUpdate()方法

创建测试方法 doUpdate() 的 TestUpdate 类，该类的具体内容如代码 29.14 所示。

代码 29.14 测试更新用户方法：TestUpdate.java

```

public class TestUpdate {
    public static void main(String[] args) throws Exception {
        User user = new User();           //创建对象 user
        user.setName("cjgong4");           //设置用户名
        user.setSex("sex");                //设置性别
        user.setId(2);                     //设置编号
        user.setBirthday(new Date());      //设置时间
        //调用 IUserDAOProxy 类中的 doUpdate() 方法
        DAOFactory.getUserDAOInstance().doUpdate(user);
    }
}

```

【运行结果】

当运行上述代码后，则数据库 users 中的表格 user 的记录变化过程如图 29.29 所示。

运行后的结果

id	name	sex	birthday
1	cjgong1	male	2010-01-02
2	cjgong2	sex	2010-02-03
3	cjgong3	male	2010-03-04
(NULL)	(NULL)	(NULL)	(NULL)

id	name	sex	birthday
1	cjgong1	male	2010-01-02
2	cjgong4	sex	2011-03-07
3	cjgong3	male	2010-03-04
(NULL)	(NULL)	(NULL)	(NULL)

图 29.29 数据库表格变化

29.5.2 人员信息管理入口类

在人员信息管理项目中，TestUserAdmin 类为该项目的入口类，该类的具体内容如代码 29.15 所示。

代码 29.15 入口类: TestUserAdmin.java

```
public class TestUserAdmin {
    public static void main(String[] args) {
        new Menu();           //创建菜单
    }
}
```

29.6 知识点扩展——设计模式的基础知识

随着时间的推移,使用 Java 语言的程序员越来越多,但是一直徘徊在语言层次的程序员却不在少数。对于这些程序员来说,虽然他们经常使用接口和抽象类,但是并没有真正掌握。如果想了解接口和抽象类,则需要学习设计模式。

29.6.1 工厂设计模式

在 Java 应用程序中经常会用到接口,每用到接口就会应用其的工厂设计模式,那么什么叫工厂设计模式呢?工厂设计模式有哪些作用呢?

为了解决上述问题,可以先复习一下接口的简单的例子,具体步骤如下。

(1) 创建一个“吃饭”的接口,具体内容如代码 29.16 所示。

代码 29.16 吃饭类: Meat.java

```
interface Meat{           //定义一个吃饭接口
    public void eat() ;    //吃饭
}
```

(2) 分别创建两个类, Noodles 类实现“吃面条”功能,具体内容如代码 29.17 所示; Rice 类实现“吃米饭”功能,具体内容如代码 29.18 所示。

代码 29.17 吃面条类: Noodles.java

```
class Noodles implements Meat{
    public void eat(){      //实现吃饭接口
        System.out.println("** 吃面条。");
    }
}
```

代码 29.18 吃米饭类: Rice.java

```
class Rice implements Meat{
    public void eat(){      //实现吃饭接口
        System.out.println("** 吃米饭。");
    }
}
```

(3) 创建测试接口实现类的 TestInterface 类,具体内容如代码 29.19 所示。

代码 29.19 吃米饭类: TestInterface.java

```
public class TestInterface {
    public static void main(String args[]) {
        Meat f = new Noodles();    //实例化接口
        f.eat();                   //调用吃饭的方法
    }
}
```

【运行结果】

运行上述代码，就会出现如图 29.30 所示的运行结果。



图 29.30 运行结果

上述几段代码非常简单，子类实现接口后，调用子类实现接口的方法，但是以上的操作是否存在问题呢？类实际上相当于一个客户端，如果要更换一个子类，则必须要修改该主类，这样实际中就会存在问题，如何解决该问题呢？这就用到了工厂设计模式，即在接口和具体实现类之间可以加入一个过渡端，通过该过渡端取得接口的实现，具体关系如图 29.31 所示，这个过渡端就是工厂类。

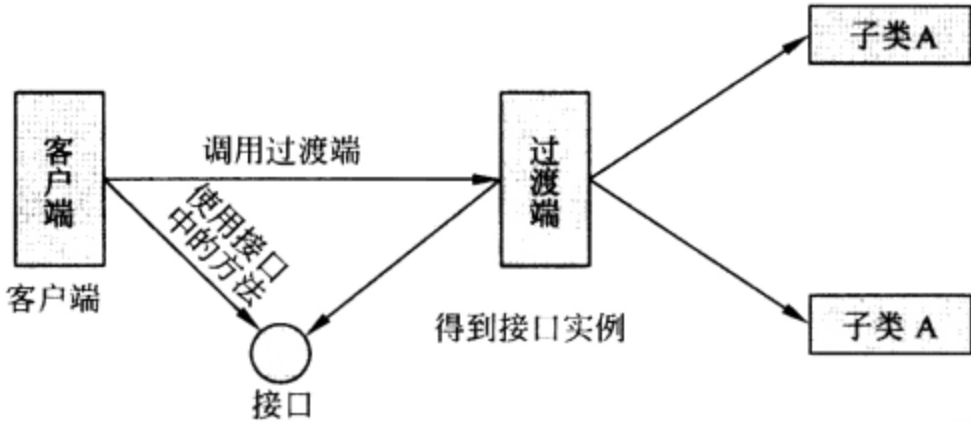


图 29.31 工厂设计模式的实现

下面通过工厂设计模式修改上述的“吃饭”代码，具体步骤如下。

(1) 创建工厂类 Factory，具体内容如代码 29.20 所示。

代码 29.20 工厂类: Factory.java

```
public class Factory {
    public static Meat getInstance(String className) {
        Meat f = null;
        if ("Noodles".equals(className)) {    //判断要的是否是吃面条的子类
            f = new Noodles();
        }
    }
}
```



```

        if ("Rice".equals(className)) { //判断要的是否是吃大米的子类
            f = new Rice();
        }
        return f;
    }
}

```

(2) 创建测试工厂类 TestFactory，具体内容如代码 29.21 所示。

代码 29.21 工厂类：TestFactory.java

```

public class TestFactory {
    public static void main(String args[]) {
        Meat f = Factory.getInstance("Noodles") ; //实例化接口
        f.eat() ;
    }
}

```

【运行结果】

运行上述代码，会出现如图 29.32 所示的运行结果。



图 29.32 运行结果

(3) 如果使用初始化参数的方式，则可以实现更大的灵活性，具体内容如代码 29.22 所示。

代码 29.22 工厂类：TestFactory1.java

```

public class TestFactory1 {
    public static void main(String args[]) {
        Meat f = null;
        f = Factory.getInstance(args[0]); //实例化接口
        if (null != f) { //判断输入参数是否为空
            f.eat();
        }
    }
}

```

【运行结果】

右击上述代码，在弹出的快捷菜单中选择 Run As | Run Configurations 菜单项，弹出 Run Configurations 对话框。该对话框的 Main 选项卡的具体设置如图 29.33 所示，Arguments 选项卡的具体设置如图 29.34 所示，之后会出现如图 29.35 所示的运行结果。如果 Arguments 选项卡的具体设置如图 29.36 所示，则会出现如图 29.37 所示的运行结果。

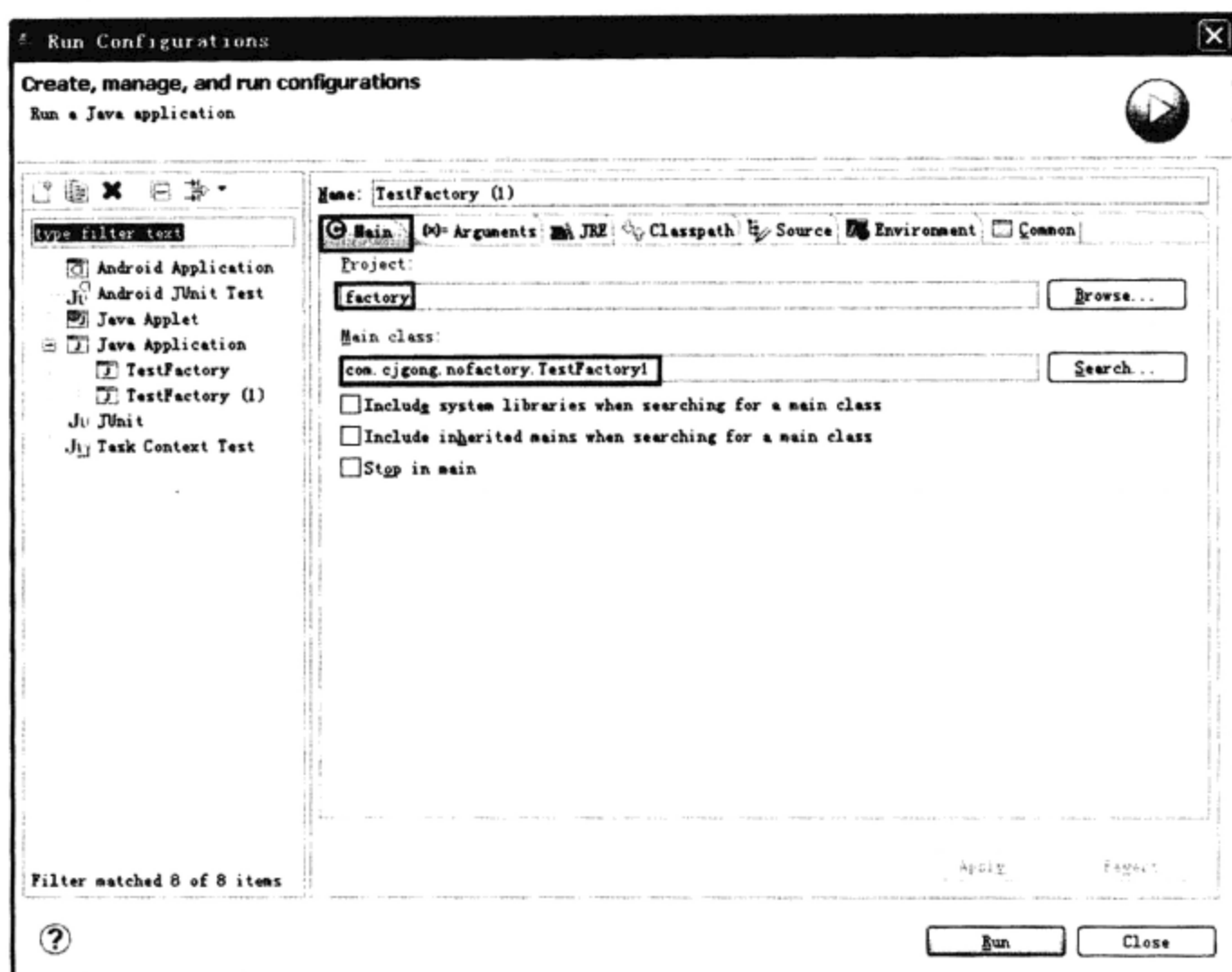


图 29.33 main 选项卡的配置

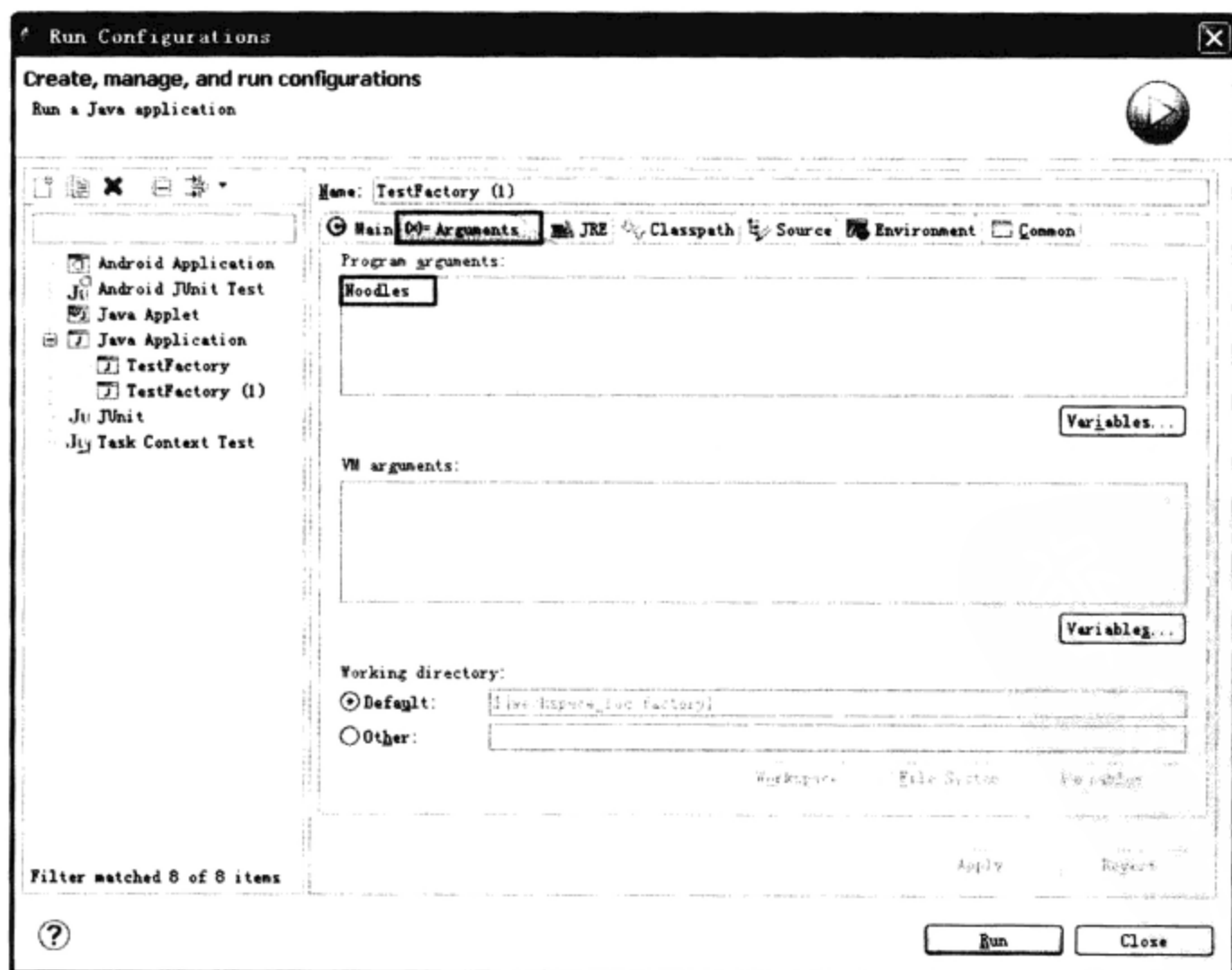


图 29.34 Arguments 选项卡的配置



图 29.35 运行结果

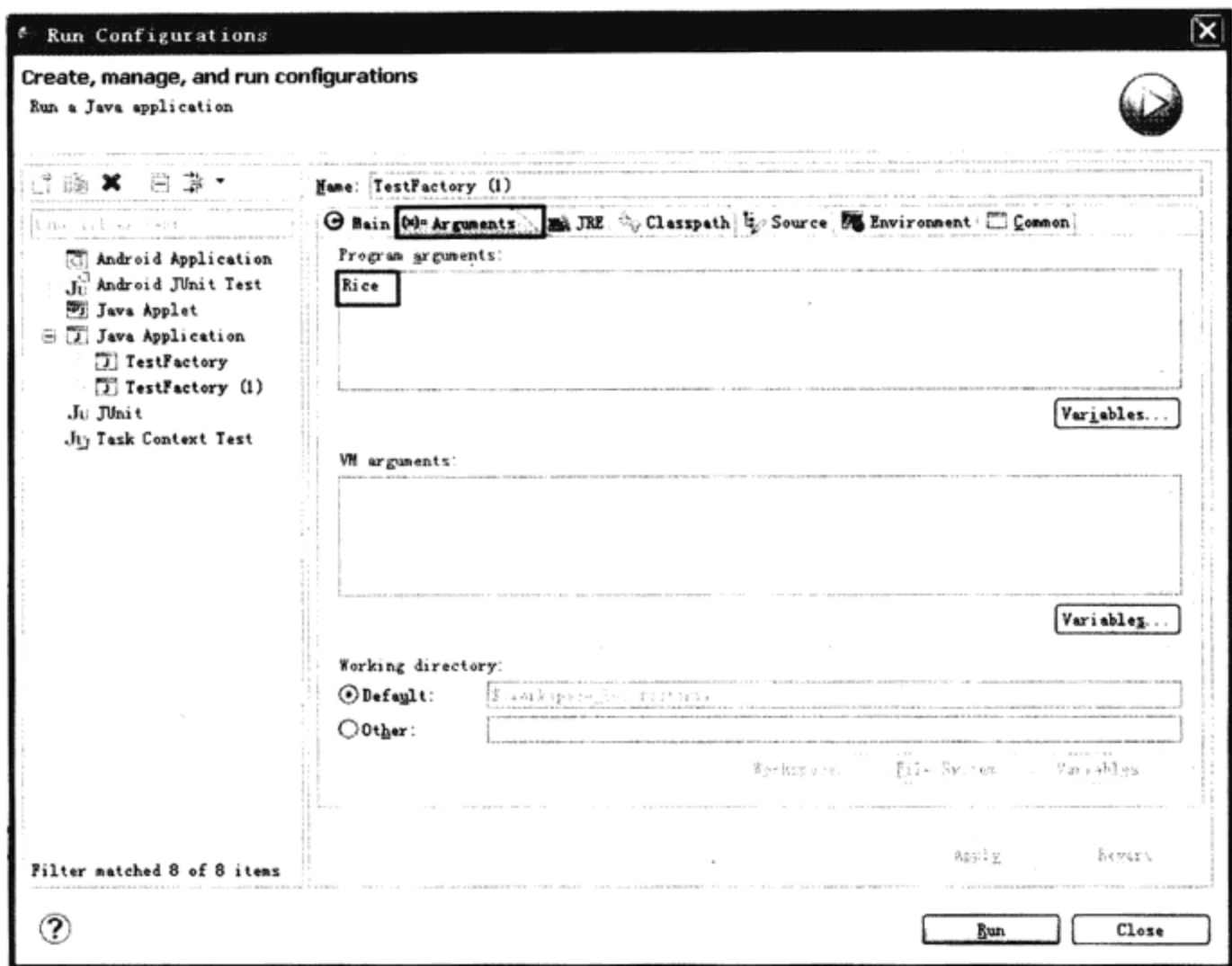


图 29.36 Arguments 选项卡的配置

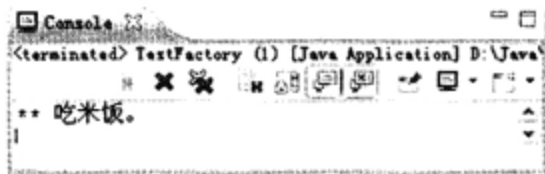


图 29.37 运行结果

总之通过上述两段代码的对比可以发现，虽然取得实例的过程不太一样，但是运行结果却是一样的。当通过工厂模式来实现功能时，如果再有子类需要扩充，则直接修改工厂类客户端就可以根据标记得得到相应的实例，灵活性更高。

29.6.2 代理设计模式

在 Java 应用程序中如果想使用接口，除了会使用到工厂设计模式，还经常会用到代理设计模式，那么什么叫代理设计模式呢？代理设计模式有哪些作用呢？所谓代理设计模式就是指由一个代理主题来操作真实主题，真实主题执行具体的业务操作，而代理主题负责其他相关业务的处理。

为了能够彻底理解代理设计模式的作用，下面通过代码来模拟生活中的代理上网，即客户通过网络代理连接网络，由代理服务器完成用户权限和访问限制等与上网相关的操作，具体过程如图 29.38 所示。

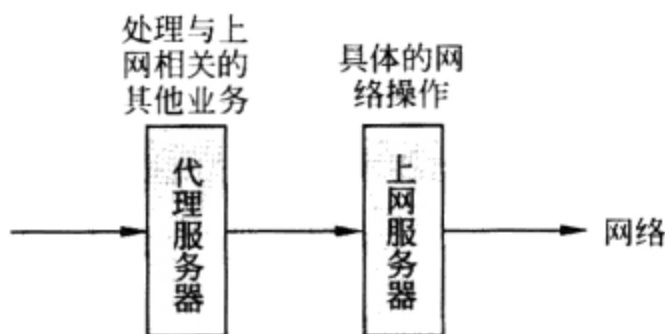


图 29.38 代理上网

在具体实现时，不管是代理操作还是真实的操作，其共同的目的就是上网，所以用户只需要关心如何上网即可，至于具体是如何操作的用户不需要关心，具体实现步骤如下。

(1) 创建一个“上网”的接口 `Internate`，具体内容如代码 29.23 所示。

代码 29.23 上网类: `Internate.java`

```
interface Internate {
    public void browse();           //浏览网页
}
```

(2) 分别创建两个类，`Real` 类为真实主题，具体内容如代码 29.24 所示；`Proxy` 类为代理主题类，具体内容如代码 29.25 所示。

代码 29.24 真实操作类: `Real.java`

```
class Real implements Internate{
    public void browse(){
        System.out.println("上网浏览信息") ;    //实现接口
    }
}
```

代码 29.25 代理操作类: `Proxy.java`

```
class Proxy implements Internate{
    private Internate network ;           //创建 network 对象
    public Proxy(Internate network){      //构造函数
        this.network = network ;         //初始化对象 network
    }
    public void check(){                  //其他操作
        System.out.println("检查用户是否合法。") ;
    }
    public void browse(){                 //实现接口方法
        this.check() ;
        this.network.browse() ;          //调用真实的主题操作
    }
}
```

(3) 创建“上网”的类 `TestProxy`，具体内容如代码 29.26 所示。

代码 29.26 上网类：TestProxy.java

```
public class TestProxy{
    public static void main(String args[]){
        Internate net = null ;
        net = new Proxy(new Real()) ;           //指定代理操作
        net.browse() ;                          //客户只关心上网浏览的操作
    }
}
```

【运行结果】

运行上述代码，就会出现如图 29.39 所示的运行结果。

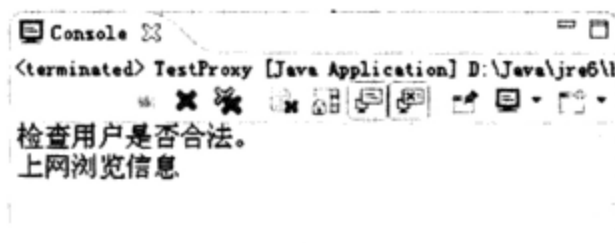


图 29.39 运行结果

通过上述流程可以看出，真实主题完成的只是上网的最基本功能，而代理主题要做比真实主题更多的业务相关操作。

29.7 小 结

本章主要介绍一个完整的人员信息管理项目，该项目与目前流行的人员信息管理项目相比，虽然没有实现任何界面，但是却实现了一些基本功能，基于 Java 语言构建而成。

在具体实现人员信息管理项目时，详细讲解了接口编程。人员信息管理项目一共分成 4 层，首先创建 DAO 层和 DAO 实现层，在具体创建该两层时会涉及所要操作的对象即对象模型层 vo，项目的逻辑功能主要在业务层（operate）实现，在具体实现时会通过组合 DAO 实现层的相关方法来实现，最后创建该项目的表示层（menu）。

第 30 章 中国象棋游戏

（GUI+游戏规则算法）

中国象棋是我国国粹，历史悠久，普及流行。在具体对战时，下棋双方根据对棋局形势的理解和对棋艺规律的掌握，调动车马，组织兵力，协调作战在棋盘这块特定的战场上，进行着象征性的军事战斗。本章将通过 Swing 组件实现象棋游戏的界面，通过事件监听机制实现该游戏的逻辑。

本章的学习目标如下：

- ❑ 掌握组件和面板的使用方法；
- ❑ 掌握 Java 语言的事件监听机制。

30.1 象棋游戏原理

“象棋游戏”是一个博弈软件系统，本章的系统只是定位于小型、简单的应用，仅实现各种棋子的相应规则、下棋双方对战及棋局的打开和保存等功能，而没有实现人机对战、人工智能等功能。

30.1.1 象棋游戏的基本规则

对弈双方通过调动棋子在棋盘上进行对决，因此必须要懂游戏得规则，否则将无法进行。本节将详细介绍该游戏的规则，具体如下所示。

对于象棋游戏来说，包含棋盘和棋子两个对象。棋子共 32 个，分为红黑两组，各 16 个，由对弈双方各执一组，兵种是一样的，分为 7 种：

- ❑ 红方：将、士、象、车、马、炮、卒；
- ❑ 黑方：将、士、象、车、马、炮、卒。

棋盘为棋子活动的场所，在长方形的平面上，由绘有 9 条平行的竖线和 10 条平行的横线相交组成，共 90 个交叉点，棋子就摆在这些交叉点上。中间第 5、6 两横线之间未画竖线的空白地带，称为“河界”，整个棋盘就以“河界”分为相等的两部分；两方将坐镇、画“米”字方格的地方，叫做“九宫”。各种棋子的移动规则如下所示。

1. 将

移动范围：只能在王宫内移动。

移动规则：每一步只可以水平或垂直移动一点。

2. 士

移动范围：只能在王宫内移动。

移动规则：每一步只可以沿对角线方向移动一点。

3. 象

移动范围：河界的一侧。

移动规则：每一步只可以沿对角线方向移动两点。另外，在移动的过程中不能穿越障碍。

4. 马

移动范围：任何位置。

移动规则：每一步只可以水平或垂直移动一点，再按对角线方向向左或者右移动。另外，移动的过程中不能穿越障碍。

5. 车

移动范围：任何位置。

移动规则：可以水平或垂直方向移动任意个无阻碍的点。

6. 炮

移动范围：任何位置。

移动规则：移动起来和车很相似，但它必须跳过一个棋子来吃掉对方的一个棋子。

7. 兵

移动范围：任何位置。

移动规则：每步只能向前移动一点。过河以后，它便增加了向左右移动的能力，兵不允许向后移动。

30.1.2 项目结构框架分析

对于象棋游戏项目，主要利用 Swing 组件实现图形用户界面，以及利用事件监听机制实现游戏逻辑。“象棋游戏”项目目录如图 30.1 所示，各个包的功能如下。

- ❑ ChessMainFrame 类：象棋游戏主类。
- ❑ ChineseChess 类：象棋游戏入口。

30.1.3 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括象棋游戏的初始化界面、查看已存在的文件属性、查看已存在的目录属性，以及查看不存在的文件和目录属性。

1. 初始化界面

当运行“象棋游戏”项目中的 ChineseChess 类后，会出现如图 30.2 所示的初始界面——象棋游戏界面。

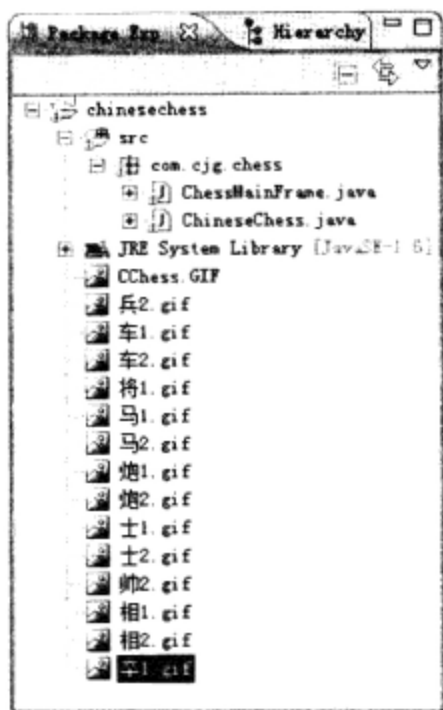


图 30.1 项目目录

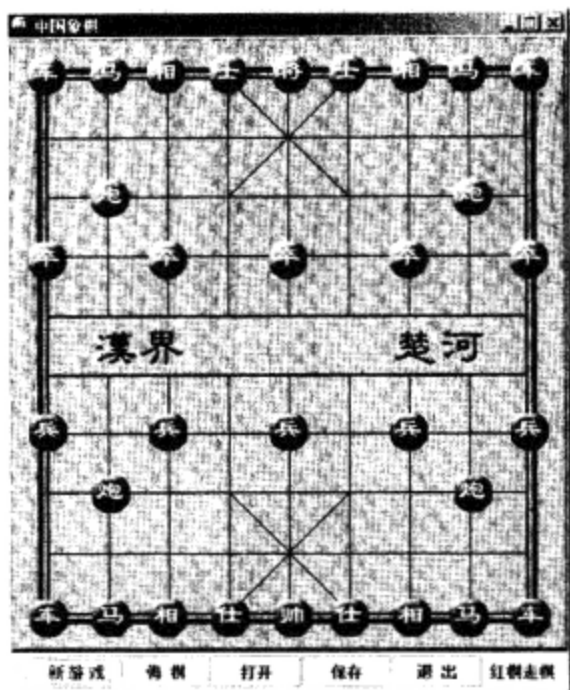


图 30.2 初始化界面

2. 走棋过程

在初始化界面中如果需要走棋，需要查看界面最下面的菜单。如果为“红棋走棋”，则需要先单击所要走的棋子（兵），这时该棋子就会出现闪烁效果。然后在规则内所能出现的地方单击棋子，这时该棋子就会在所单击的地方出现，而在原来的地方消失，具体过程如图 30.3 所示。

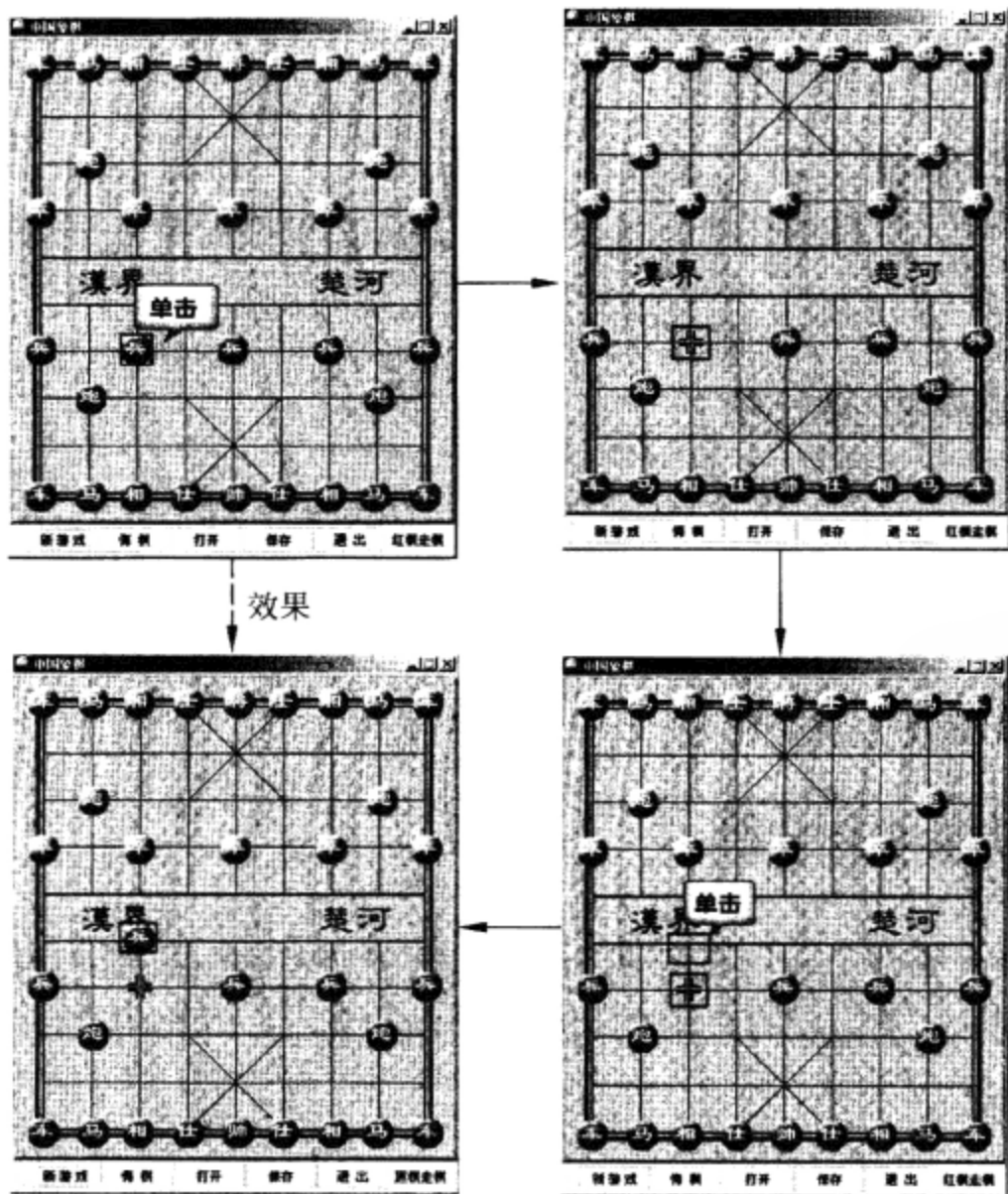


图 30.3 走棋过程

3. 吃棋过程

为了模拟吃棋过程，本例专门创建了红棋兵吃黑棋兵的局面。首先单击所要走的红棋（兵），这时该棋子就会出现闪烁效果。然后在规则内所能出现黑棋（兵）的地方单击，这时红棋就会出现在黑棋所在的地方，而黑棋则消失。具体过程如图 30.4 所示。

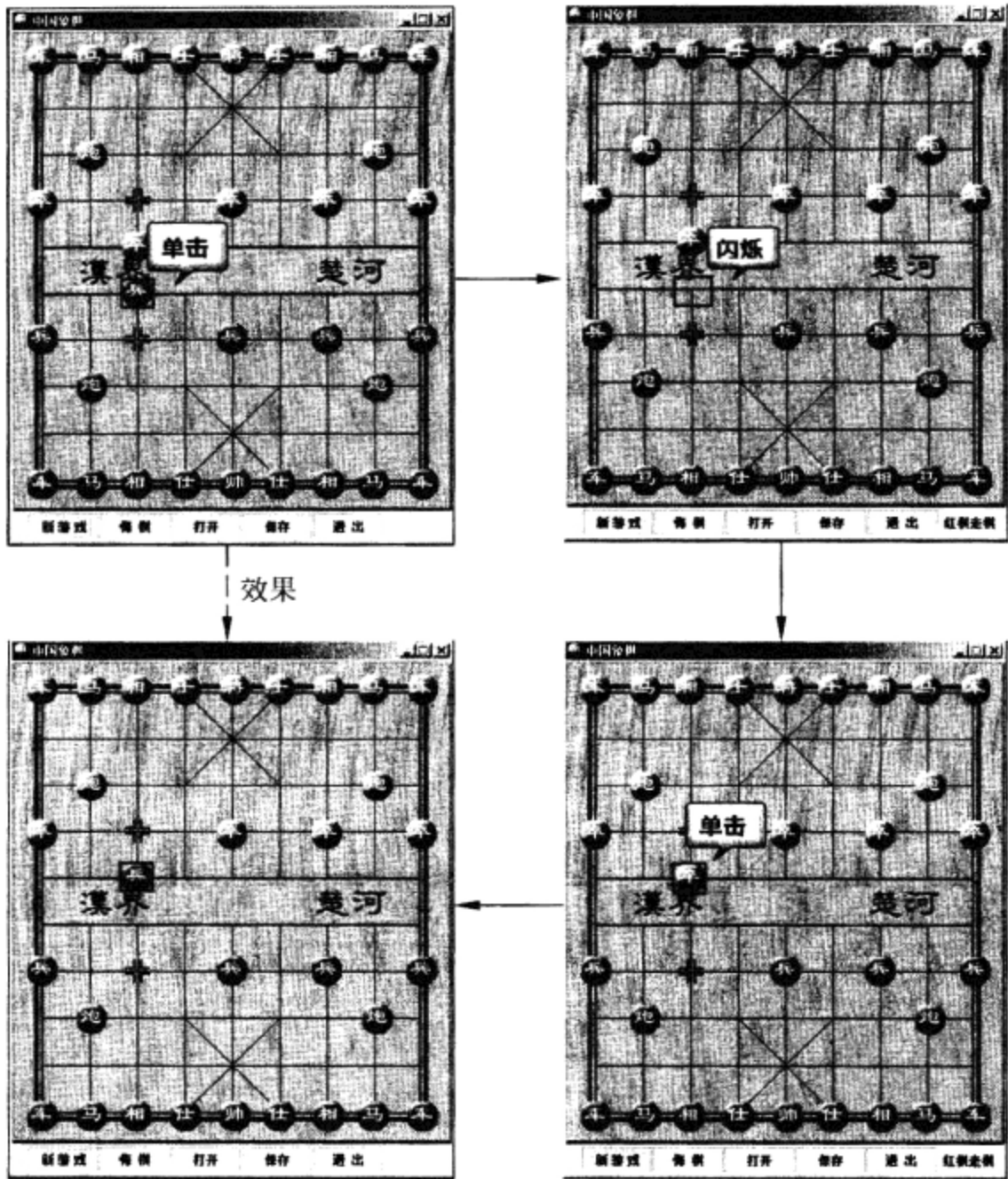


图 30.4 吃棋过程

4. 悔棋菜单

象棋具体对战时经常会悔棋，为了实现该功能，本例专门在界面的最下面设置一个“悔棋”按钮。为了验证该菜单的功能，在“吃棋过程”后的棋面中，单击几次“悔棋”按钮就会返回到“吃棋过程”之前的棋局，具体过程如图 30.5 所示。

5. 保存菜单

象棋具体对战时经常会出现保存对战棋局，为了实现该功能，专门在界面的最下面出现一个“保存”菜单。为了验证该菜单的功能，对于“走棋过程”后的棋面，单击“保存”按钮就会把该棋局保存到相应的文件里，具体过程如图 30.6 所示。

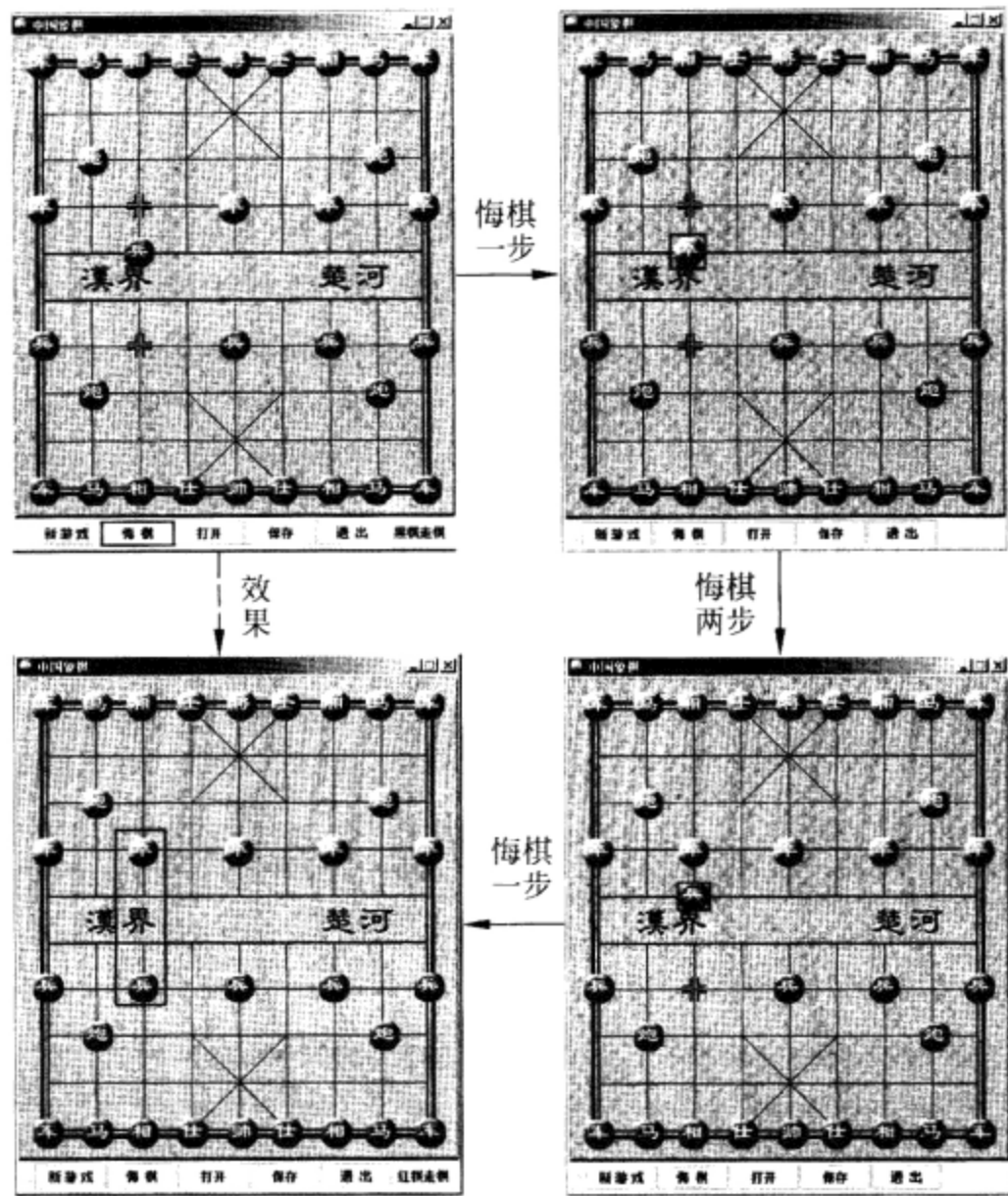


图 30.5 悔棋过程

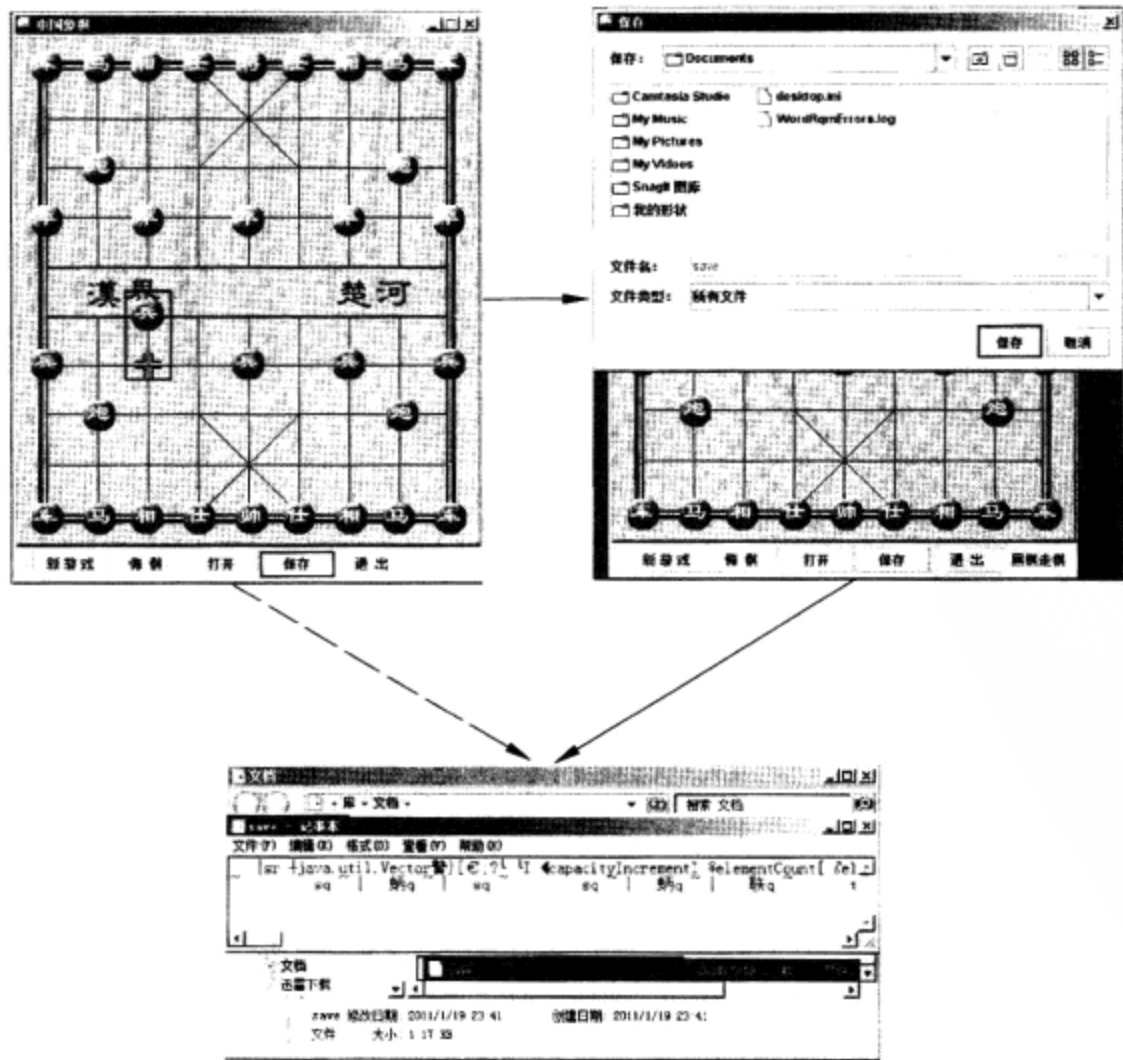


图 30.6 保存过程

6. 新游戏菜单

象棋具体对战过程中有时需要重启游戏, 为了实现该功能, 本例专门在界面的最下面设置了一个“新游戏”菜单。该菜单的作用如图 30.7 所示。

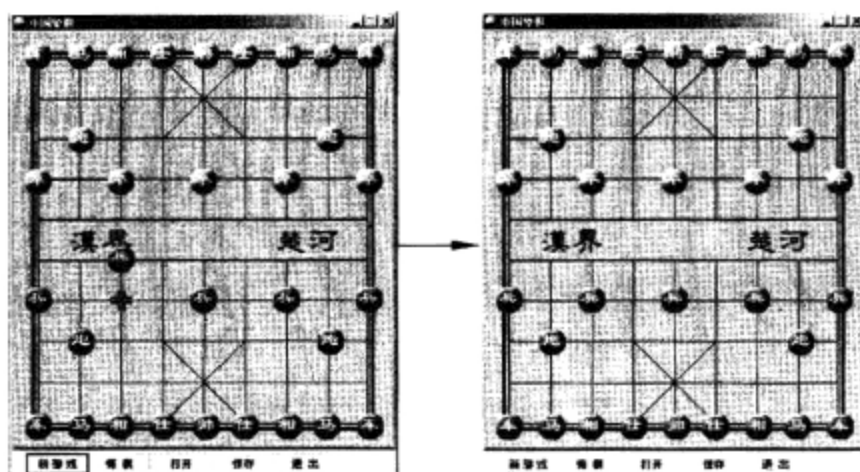



图 30.7 “新游戏”菜单

7. 打开菜单

当出现初始化界面后, 如果从以前的棋局开始, 可以通过“打开”菜单布局以前的棋局, 具体过程如图 30.8 所示。

8. 退出菜单

如果想退出象棋游戏, 可以单击右上角的  按钮, 如图 30.9 所示, 或者选择“退出”菜单, 具体过程如图 30.10 所示。

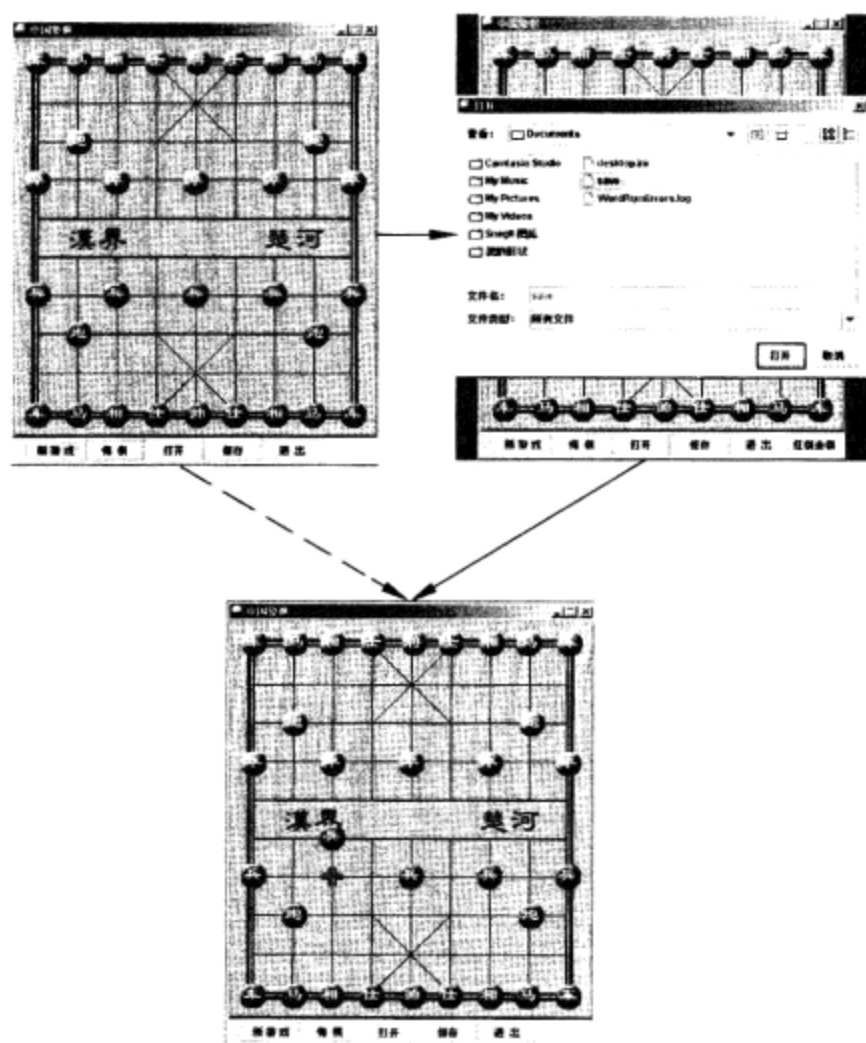


图 30.8 打开菜单

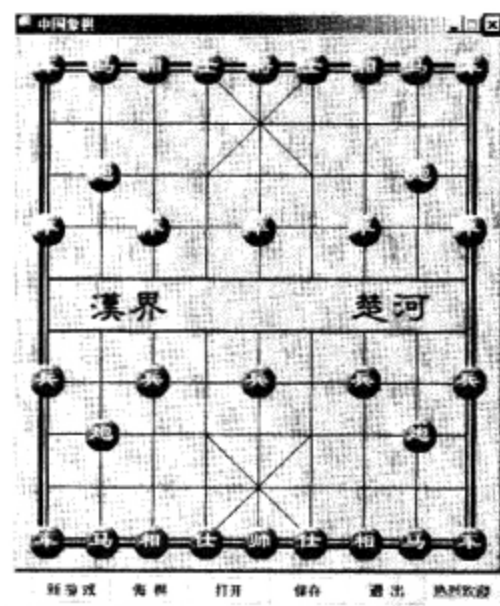


图 30.9 退出功能

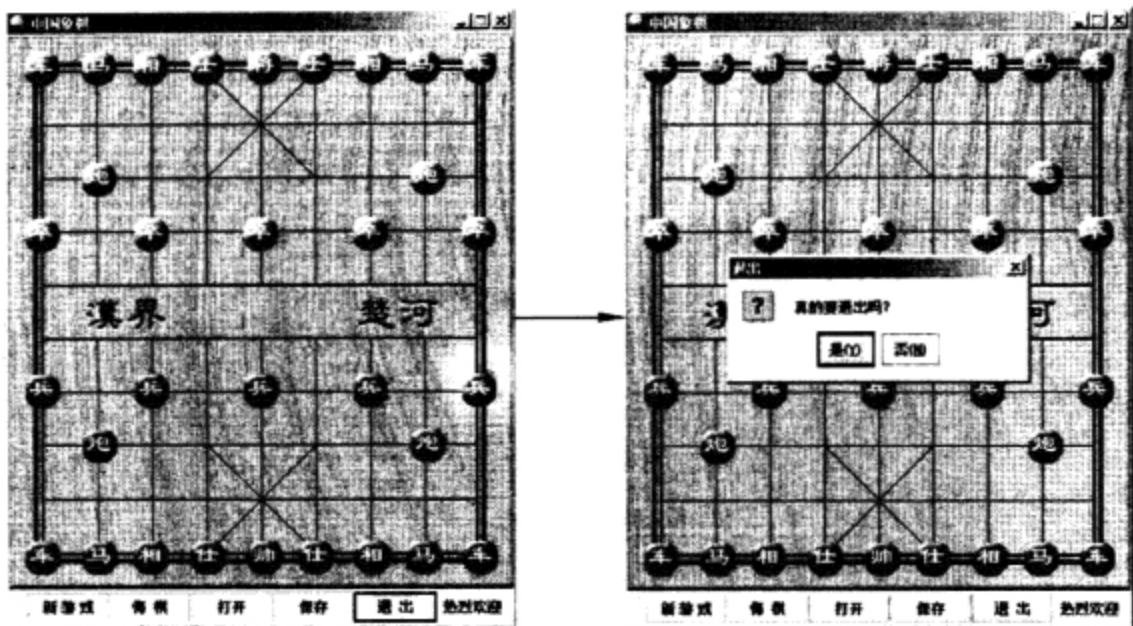


图 30.10 “退出”菜单

30.2 象棋游戏项目——象棋游戏的主类

在象棋游戏项目的主类 ChessMainFrame 中，不仅创建象棋游戏的各种成员变量，而且还会通过多线程技术实现棋子的闪烁和各种棋子的规则。该类的类图如图 30.11 所示。

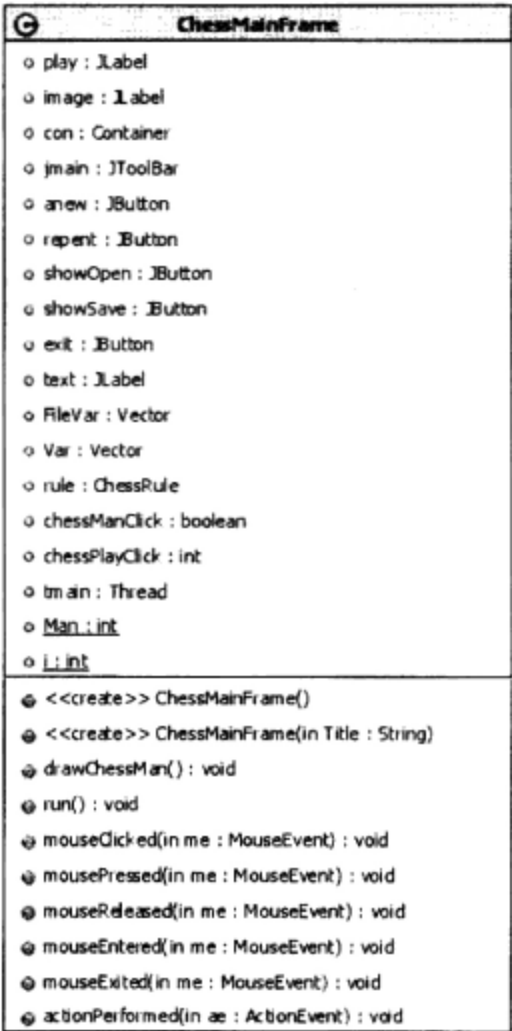


图 30.11 主类的 UML 图

30.2.1 实现象棋游戏的主界面

为了实现象棋游戏的主界面，在主类 ChessMainFrame 中创建各种成员变量，并在该

类的构造函数中对它们进行初始化, 实现主界面的内容如代码 30.1 所示。

代码 30.1 实现主界面: ChessMainFrame.java

```
class ChessMainFrame extends JFrame implements ActionListener, Mouse Listener,
    Runnable {
    //创建成员变量
    JLabel play[] = new JLabel[32];           //存储棋子的数组变量
    JLabel image;                             //棋盘的标签变量
    Container con;                             //窗格的容器变量
    JToolBar jmain;                           //工具栏的变量
    JButton anew;                             //重新开始的按钮变量
    JButton repent;                           //悔棋的按钮变量
    JButton showOpen;                         //打开的按钮变量
    JButton showSave;                         //保存的按钮变量
    JButton exit;                             //退出的按钮变量
    JLabel text;                              //当前信息的标签变量
    //创建保存当前操作的集合变量
    Vector FileVar;
    Vector Var;
    ChessRule rule;                           //规则类对象
    boolean chessManClick;                    //棋子状态的变量
    int chessPlayClick = 2;                   //玩家走棋的变量
    Thread tmain;                             //棋子闪烁的线程变量
    static int Man, i;                         //第一次单击棋子的线程响应
    ChessMainFrame() {                        //无参构造函数
    }
    ChessMainFrame(String Title) {            //带参构造函数
        con = this.getContentPane();          //为容器对象赋值
        con.setLayout(null);                 //设置布局管理器
        rule = new ChessRule();              //为规则类对象赋值
        //为保存当前操作的集合变量赋值
        FileVar = new Vector();
        Var = new Vector();
        //创建和设置工具栏
        jmain = new JToolBar();               //为工具栏变量赋值
        text = new JLabel(" 热烈欢迎");       //为变量 text 赋值并设置
        text.setToolTipText("提示信息");
        anew = new JButton(" 新 游 戏 ");     //为变量 anew 赋值并设置
        anew.setToolTipText("重新开始新的一局");
        exit = new JButton(" 退 出 ");         //为变量 exit 赋值并设置
        exit.setToolTipText("退出本程序");
        repent = new JButton(" 悔 棋 ");       //为变量 repent 赋值并设置
        repent.setToolTipText("返回到上次走棋的位置");
        showOpen = new JButton("打开");        //为变量 showOpen 赋值并设置
        showOpen.setToolTipText("打开以前棋局");
        showSave = new JButton("保存");        //为变量 showSave 赋值并设置
        showSave.setToolTipText("保存当前棋局");
        //实现把组件添加到工具栏中
        jmain.setLayout(new GridLayout(0, 6)); //设置工具栏的布局管理器
        jmain.add(anew);
        jmain.add(repent);
        jmain.add(showOpen);
        jmain.add(showSave);
```

```

jmain.add(exit);
jmain.add(text);
jmain.setBounds(0, 500, 450, 30);           //设置工具栏的大小
con.add(jmain);                             //添加工具栏到容器中
drawChessMan();                             //实现添加棋子
//对相应组件进行事件的监听
anew.addActionListener(this);
repent.addActionListener(this);
exit.addActionListener(this);
showOpen.addActionListener(this);
showSave.addActionListener(this);
//对每个棋子进行移动事件的监听
for (int i = 0; i < 32; i++) {
    con.add(play[i]);
    play[i].addMouseListener(this);
}
//实现添加并设置棋盘标签
con.add(image = new JLabel(new ImageIcon("CChess.GIF")));
image.setBounds(0, 0, 445, 498);           //设置大小
image.addMouseListener(this);             //注册监听事件
//实现注册窗体关闭监听
this.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent we) {
        System.exit(0);                    //退出系统
    }
});
//用来实现窗体的居中
//获取当前屏幕的大小
Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
Dimension frameSize = this.getSize();       //获取当前窗口的大小
//用来设置当前软件界面的大小
if (frameSize.height > screenSize.height) {
    frameSize.height = screenSize.height;
}
if (frameSize.width > screenSize.width) {
    frameSize.width = screenSize.width;
}
//用来实现窗口的居中
this.setLocation((screenSize.width - frameSize.width) / 2 - 200,
    (screenSize.height - frameSize.height) / 2 - 290);
this.setIconImage(new ImageIcon("相1.gif").getImage());
//设置当前窗口的标题图标
this.setResizable(false);                  //设置不能更改窗口大小
this.setTitle(Title);                     //设置标题
this.setSize(450, 555);                   //设置窗口大小
this.show();                              //显示窗口
}
//用来实现添加棋子的方法
public void drawChessMan() {
...                                         //省略部分内容
}
//利用线程来控制棋子闪烁的方法
public void run() {
...                                         //省略部分内容
}
//用来实现单击棋子的方法
public void mouseClicked(MouseEvent me) {

```

```

...
}
public void mousePressed(MouseEvent me) {
}
public void mouseReleased(MouseEvent me) {
}
public void mouseEntered(MouseEvent me) {
}
public void mouseExited(MouseEvent me) {
}
//编写监听按钮事件的监听器
public void actionPerformed(ActionEvent ae) {
...
}
}
//省略部分代码

```

【代码解析】

- 上述代码首先创建象棋游戏主窗口对象的成员变量，具体有表示存储棋子的数组 `play[]`；表示棋盘的变量 `image`；表示主窗口容器的 `con`；表示菜单工具栏的变量 `jmain`；表示工具栏上的按钮对象 `anew`、`repent`、`showOpen`、`showSave` 和 `exit`。提示哪方走的变量 `text`；实现保存当前操作的变量 `FileVar` 和 `Var`；表示棋子状态的变量 `chessManClick`，当其为 `true` 时棋子会闪烁，当其为 `false` 时棋子会停止闪烁表示吃棋子；表示玩家走棋的变量 `chessPlayClick`，当其值为 1 则表示持黑棋者走棋，当其值为 2 时表示持红棋者走棋，当其值为 3 时表示双方都不走棋；实现棋子闪烁线程的对象 `tmain`；实现对棋子规则类的引用变量 `rule`。
- 在上述代码中存在一个拥有窗体标题的构造函数 `ChessMainFrame(String Title)`，该方法主要用于创建象棋的窗口，该窗口的布局如图 30.12 所示。

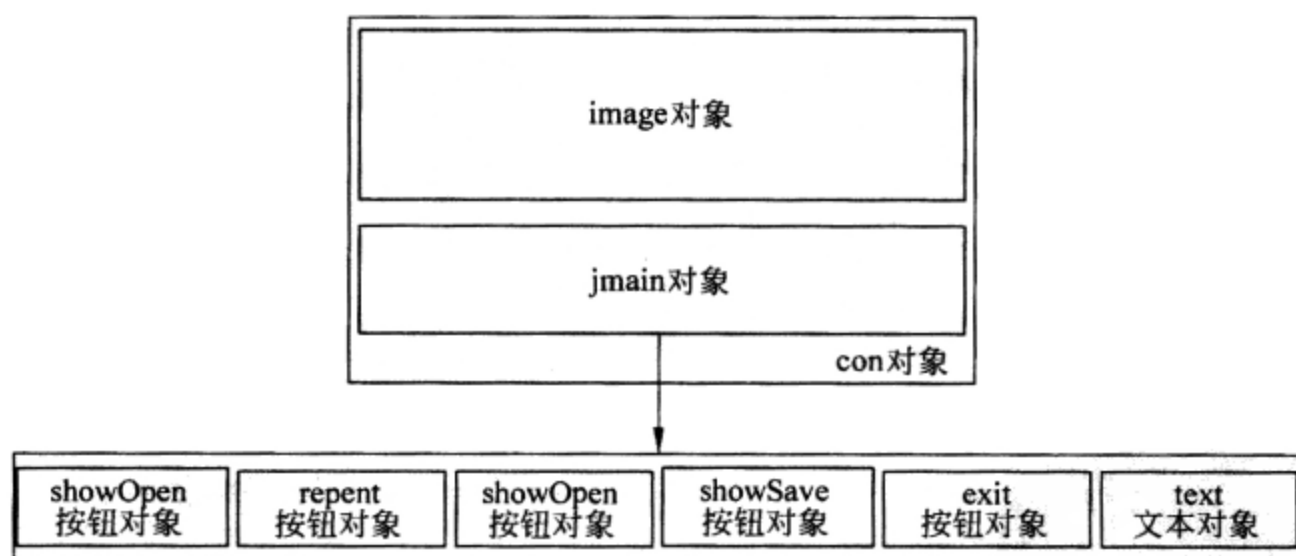


图 30.12 布局

- 在设计象棋游戏的主窗口时，不仅实现窗口居中显示，而且还为该窗口里的一些对象注册了一些事件监听器，如为工具栏上的所有按钮对象注册了动作监听器；为存储到数组表示棋子的每个 `Label` 对象注册了鼠标的监听器；为窗口注册了窗口事件监听器。

30.2.2 实现象棋游戏中添加棋子的功能

设计好象棋游戏的主界面后，如何添加棋子到主界面中呢？为了实现该功能，在主类

的构造函数中，调用了一个实现添加棋子的方法，该方法的具体内容如代码 30.2 所示。

代码 30.2 添加棋子的方法：drawChessMan()

```
class ChessMainFrame extends JFrame implements ActionListener, Mouse Listener,
    Runnable {
...                                     //省略部分代码
    //用来实现添加棋子方法
    public void drawChessMan() {
        int i, k;                                     //流程控制的变量
        Icon in;                                       //图标变量
        //添加并设置黑色棋子
        //添加车棋子
        in = new ImageIcon("车 1.GIF");               //为图标变量赋值
        for (i = 0, k = 10; i < 2; i++, k += 385) { //通过遍历存储到数组 play 中
            play[i] = new JLabel(in);
            play[i].setBounds(k, 10, 40, 40);         //设置大小
            play[i].setName("车 1");
        }
        //添加马棋子
        in = new ImageIcon("马 1.GIF");               //为图标变量赋值
        for (i = 4, k = 60; i < 6; i++, k += 305) { //通过遍历存储到数组 play 中
            play[i] = new JLabel(in);
            play[i].setBounds(k, 10, 40, 40);         //设置大小
            play[i].setName("马 1");
        }
        ...                                           //省略部分代码
        //添加并设置红色棋子
        //添加车棋子
        in = new ImageIcon("车 2.GIF");               //为图标变量赋值
        for (i = 2, k = 10; i < 4; i++, k += 385) { //通过遍历存储到数组 play 中
            play[i] = new JLabel(in);
            play[i].setBounds(k, 450, 40, 40);        //设置大小
            play[i].setName("车 2");
        }
        //添加马棋子
        in = new ImageIcon("马 2.GIF");               //为图标变量赋值
        for (i = 6, k = 60; i < 8; i++, k += 305) { //通过遍历存储到数组 play
            play[i] = new JLabel(in);
            play[i].setBounds(k, 450, 40, 40);        //设置大小
            play[i].setName("马 2");
        }
        ...                                           //省略部分代码
    }
...                                               //省略部分代码
}
```

【代码解析】

- ❑ 在上述代码中，首先创建了 3 个成员变量，即变量 i 表示各种棋子的个数，变量 k 表示棋子离主窗口左边的距离，变量 in 表示棋子的图标。
- ❑ 在具体添加棋子时，首先为该棋子的图标变量 in 赋值，然后通过循环为数组的元素赋值并设置其为主窗口中的位置。

30.2.3 实现象棋游戏中棋子闪烁的功能

设计好象棋游戏的主界面后, 如何实现当用户单击棋子时, 该对象会闪烁呢? 为了实现该功能, 下面的代码在主类中实现多线程, 具体内容如代码 30.3 所示。

代码 30.3 实现棋子闪烁的功能: run()

```
class ChessMainFrame extends JFrame implements ActionListener, MouseListener,
    Runnable {
    ...                                     //省略部分代码
    //利用线程来控制棋子闪烁的方法
    public void run() {
        while (true) {
            //第一次单击棋子时开始闪烁
            if (chessManClick) {           //当第一次单击棋子时
                play[Man].setVisible(false); //使棋子不显示
                //用来实现控制时间
                try {
                    tmain.sleep(500);
                } catch (Exception e) {
                }
                play[Man].setVisible(true); //使棋子显示
            }
            //为了避免用户看不见, 显示提示信息
            else {
                text.setVisible(false); //使提示信息不显示
                //用来实现控制时间
                try {
                    tmain.sleep(500);
                } catch (Exception e) {
                }
                text.setVisible(true); //使提示信息显示
            }
            try {
                tmain.sleep(500);
            } catch (Exception e) {
            }
        }
    }
    ...                                     //省略部分代码
}
```

【代码解析】

- 上述代码中的 run()方法为了实现闪烁现象, 首先让需要闪烁的对象不显示, 然后让当前线程休眠 500 毫秒, 最后再让对象显示出来。
- 上述代码在无限循环中首先判断是否为第一次单击棋子, 如果是则让该对象闪烁; 否则让文本框对象闪烁。最后再让当前线程休眠 500 毫秒。

30.2.4 处理单击棋子事件

对于象棋游戏来说, 如果用鼠标单击棋盘, 则表示要把处于闪烁状态的棋子走到该位

置；如果用户鼠标单击棋子，同时还有其他棋子处于闪烁状态，则表示该棋子将被闪烁棋子吃掉，否则当没有其他棋子处于闪烁状态，表示 giant 棋子将要移动，请单击其他位置。象棋游戏中的鼠标单击事件的处理内容如代码 30.4 所示。

代码 30.4 处理单击棋子事件的方法：mouseClicked()

```
class ChessMainFrame extends JFrame implements ActionListener, Mouse Listener,
    Runnable {
    ... //省略部分代码
    //用来实现单击棋子的方法
    public void mouseClicked(MouseEvent me) {
        int Ex = 0, Ey = 0; //创建表示当前坐标的变量
        //启动线程
        if (tmain == null) {
            tmain = new Thread(this); //为变量 tmain 赋值
            tmain.start(); //启动线程变量
        }
        //用来实现移动棋子的方法，即单击棋盘方法
        if (me.getSource().equals(image)) {
            //当该红棋走棋的时候
            if (chessPlayClick == 2 && play[Man].getName().charAt(1) == '2') {
                Ex = play[Man].getX();
                Ey = play[Man].getY();
                if (Man > 15 && Man < 26) { //用来实现移动卒
                    rule.armsRule(Man, play[Man], me);
                }
                ... //省略部分代码
                //判断是否走棋错误（是否在原地没有动）
                if (Ex == play[Man].getX() && Ey == play[Man].getY()) {
                    text.setText(" 红棋走棋");
                    chessPlayClick = 2;
                }
            } else {
                text.setText(" 黑棋走棋");
                chessPlayClick = 1;
            }
        }
        //当应该黑棋走棋时
        else if (chessPlayClick == 1
            && play[Man].getName().charAt(1) == '1') {
            Ex = play[Man].getX();
            Ey = play[Man].getY();
            if (Man > 15 && Man < 26) { //用来实现移动卒
                rule.armsRule(Man, play[Man], me);
            }
            ... //省略部分代码
            //是否走棋错误（是否在原地没有动）
            if (Ex == play[Man].getX() && Ey == play[Man].getY()) {
                text.setText(" 黑棋走棋");
                chessPlayClick = 1;
            }
        } else {
            text.setText(" 红棋走棋");
            chessPlayClick = 2;
        }
    }
}
```



```

    }
    chessManClick = false;           //当前没有操作（停止闪烁）时
}
//用来实现单击棋子
else {
    if (!chessManClick) {           //当第一次单击棋子，即闪烁棋子时
        for (int i = 0; i < 32; i++) {
            //获取到被单击的棋子
            if (me.getSource().equals(play[i])) {
                Man = i;             //通知线程让该棋子闪烁
                chessManClick = true; //用来实现闪烁
                break;
            }
        }
    }
    else if (chessManClick) {        //当第二次单击棋子，即吃棋子时
        chessManClick = false;       //当前没有操作（停止闪烁）
        for (i = 0; i < 32; i++) {
            //获取到被吃的棋子
            if (me.getSource().equals(play[i])) {
                //当应该红棋吃棋时
                if (chessPlayClick == 2
                    && play[Man].getName().charAt(1) == '2') {
                    Ex = play[Man].getX();
                    Ey = play[Man].getY();
                    if (Man > 15 && Man < 26) { //实现卒吃规则
                        rule.armsRule(play[Man], play[i]);
                    }
                    ... //省略部分代码
                }
                //判断是否走棋错误（是否在原地没有动）
                if (Ex == play[Man].getX()
                    && Ey == play[Man].getY()) {
                    text.setText(" 红棋走棋");
                    chessPlayClick = 2;
                    break;
                }
            }
            else {
                text.setText(" 黑棋走棋");
                chessPlayClick = 1;
                break;
            }
        }
        //当应该黑棋吃棋时
        else if (chessPlayClick == 1
            && play[Man].getName().charAt(1) == '1') {
            Ex = play[Man].getX();
            Ey = play[Man].getY();
            if (Man > 15 && Man < 26) { //实现卒吃规则
                rule.armsRule(play[Man], play[i]);
            }
            ... //省略部分代码
        }
        //是否走棋错误（是否在原地没有动）
        if (Ex == play[Man].getX()
            && Ey == play[Man].getY()) {
            text.setText(" 黑棋走棋");
            chessPlayClick = 1;
            break;
        }
    }
}

```

```

        else {
            text.setText(" 红棋走棋");
            chessPlayClick = 2;
            break;
        }
    }
}
//用来实现判断哪方胜利
if (!play[31].isVisible()) { //当黑棋胜利时
    JOptionPane.showConfirmDialog(this, "黑棋胜利", "玩家一胜利",
        JOptionPane.DEFAULT_OPTION,
        JOptionPane.WARNING_MESSAGE);
    //双方都不可以再走棋了
    chessPlayClick = 3;
    text.setText(" 黑棋胜利");
}
else if (!play[30].isVisible()) { //当红棋胜利时
    JOptionPane.showConfirmDialog(this, "红棋胜利", "玩家二胜利",
        JOptionPane.DEFAULT_OPTION,
        JOptionPane.WARNING_MESSAGE);
    chessPlayClick = 3;
    text.setText(" 红棋胜利");
}
}
}
}

```

【代码解析】

- ❑ 上述代码中首先初始化棋子的当前坐标和线程对象 `tmain`，同时启动该线程对象。然后通过判断单击对象是棋盘还是棋子来决定是进行移动棋子操作，还是单击棋子操作。最后通过判断“将”的棋子是否显示，来决定是黑方胜利还是红方胜利。
- ❑ 当获取单击的对象为棋盘时，则表示需要移动处于闪烁状态的棋子。这时首先需要判断 `chessPlayClick` 变量的值和 `play[Man].getName()` 获取到的名字中，是否包含 2 来决定应该是红方走棋还是黑方走棋。当应该红方走棋时，首先获取闪烁棋子的坐标 (Ex,Ey)，然后通过判断变量 `Man` 属于数组 `play[Man]` 中标号的范围，来决定调用 `rule` 对象中的哪个方法来实现移动。最后再通过判断移动后棋子的坐标与 (Ex,Ey) 是否相等，来决定棋子是否移动成功。当两者不等时，表示棋子移动成功，这时需要设置文本对象 `text` 的值并修改 `chessPlayClick`；否则还是原来的值。
- ❑ 当获取单击的对象为棋子时，则表示是单击棋子，该种状态有两种情况，即第一次单击棋子和吃掉棋子。对于第一种情况，首先通过循环获取单击棋子所在数组 `play` 中的标号 `i`，然后再把变量 `i` 的值赋值给变量 `man`，最后通过使变量 `chessManClick` 的值为 `true` 实现该变量 `man` 所对应的棋子闪烁。对于第二种情况，则需要通过判断闪烁棋子的名字和 `chessPlayClick` 变量的值来决定应该是红方棋子吃棋还是黑方棋子吃棋，当应该红方棋子吃棋时，首先存储闪烁棋子的当前坐标到坐标 (Ex,Ey)，然后通过判断闪烁棋子的标号变量 `man` 来调用 `rule` 对象中的方法实现吃掉 `play[i]` 对象。最后还需要判断 `play[man]` 对象的当前坐标与 (Ex,Ey)

是否相同来决定是否成功吃掉棋子, 当两个坐标相同时, 则表示红方棋子吃掉了对方棋子, 这时需要设置 text 对象的值为“黑棋走棋”及 chessPlayClick 变量的值为 2。当应该黑棋吃棋时, 则相反。

30.2.5 处理单击按钮事件

在象棋游戏的主界面下还有一行按钮, 单击这些按钮, 可以实现相应的功能, 象棋游戏中的鼠标单击按钮事件的处理内容, 如代码 30.5 所示。

代码 30.5 监听按钮动作的方法: actionPerformed()

```
//编写监听按钮动作的事件
public void actionPerformed(ActionEvent ae) {
    //当单击“新游戏”按钮时
    if (ae.getSource().equals(anew)) {
        int i, k; //创建两个变量
        //重新排列黑棋中的每个棋子位置
        for (i = 0, k = 10; i < 2; i++, k += 385) { //重新排列“车”棋
            play[i].setBounds(k, 10, 40, 40);
        }
        for (i = 4, k = 60; i < 6; i++, k += 305) { //重新排列“马”棋
            play[i].setBounds(k, 10, 40, 40);
        }
        ... //省略部分代码
        play[31].setBounds(205, 450, 40, 40); //重新排列“帅”棋
        //显示红棋先走
        chessPlayClick = 2;
        text.setText(" 红棋走棋");
        for (i = 0; i < 32; i++) { //显示全部棋子
            play[i].setVisible(true);
        }
        for (i = 0; i < Var.size(); i++) {
            Var.remove(i);
        }
    }
    //当单击“悔棋”按钮时
    else if (ae.getSource().equals(repent)) {
        try {
            //获得 setVisible 属性值
            String S = (String) Var.get(Var.size() - 4);
            //获得 x 坐标
            int x = Integer.parseInt((String) Var.get(Var.size() - 3));
            //获得 y 坐标
            int y = Integer.parseInt((String) Var.get(Var.size() - 2));
            //获得索引
            int M = Integer.parseInt((String) Var.get(Var.size() - 1));
            //显示象棋的每个棋子
            play[M].setVisible(true);
            play[M].setBounds(x, y, 40, 40);
            //用来实现判断哪一方先走
            if (play[M].getName().charAt(1) == '1') {
                text.setText(" 黑棋走棋");
                chessPlayClick = 1;
            } else {
```

```

        text.setText(" 红棋走棋");
        chessPlayClick = 2;
    }
    //实现删除用过的坐标
    Var.remove(Var.size() - 4);
    Var.remove(Var.size() - 3);
    Var.remove(Var.size() - 2);
    Var.remove(Var.size() - 1);
    chessManClick = false; //停止棋子闪烁
}
catch (Exception e) {
}
}
//当发生事件的按钮为打开按钮时
else if (ae.getSource().equals(showOpen)) {
    try {
        //创建打开对话框变量
        JFileChooser jfcOpen = new JFileChooser("打开棋局");
        int v = jfcOpen.showOpenDialog(this);
        if (v != JFileChooser.CANCEL_OPTION) {
            Var.removeAllElements(); //删除集合的所有信息
            FileVar.removeAllElements();
            //打开文件获得所有数据
            FileInputStream fileIn = new FileInputStream(jfcOpen
                .getSelectedFile());
            //封装文件流
            ObjectInputStream objIn = new ObjectInputStream(fileIn);
            FileVar = (Vector) objIn.readObject();
            fileIn.close(); //关闭文件流
            objIn.close(); //关闭对象流
            //集合内容对应的棋子坐标
            int k = 0;
            for (int i = 0; i < 32; i++) {
                play[i].setBounds(
                    ((Integer) FileVar.get(k)).intValue(),
                    ((Integer) FileVar.get(k + 1)).intValue(), 40,
                    40);
                //实现被吃掉的棋子不显示
                if (!(Boolean) FileVar.elementAt(k + 2)
                    .booleanValue()) {
                    play[i].setVisible(false); //不显示
                }
                k += 3;
            }
            //判断当前应该哪方棋子走棋
            if (((String) FileVar.lastElement()).toString().equals(
                " 红棋走棋")) {
                text.setText(((String) FileVar.lastElement())
                    .toString());
                chessPlayClick = 2;
            }
            ... //省略部分代码
        }
    }
    catch (Exception e) {
        System.out.println("ERROR ShowOpen");
    }
}

```

```

//当发生事件的按钮为保存按钮时
else if (ae.getSource().equals(showSave)) {
    try {
        //创建对话框
        JFileChooser jfcSave = new JFileChooser("保存当前棋局");
        int v = jfcSave.showSaveDialog(this);
        if (v != JFileChooser.CANCEL_OPTION) { //当单击“保存”按钮时
            FileVar.removeAllElements();
            //保存所有棋子的坐标及是否可见
            for (int i = 0; i < 32; i++) {
                FileVar.addElement(new Integer(play[i].getX()));
                FileVar.addElement(new Integer(play[i].getY()));
                FileVar.addElement(new Boolean(play[i].isVisible()));
            }
            //保存当前该哪方进行
            FileVar.add(text.getText());
            //用来实现保存到文件中
            FileOutputStream fileOut = new FileOutputStream(jfcSave
                .getSelectedFile()); //创建文件输出流
            //封装文件输出流
            ObjectOutputStream objOut = new ObjectOutputStream(fileOut);
            objOut.writeObject(FileVar); //实现输出
            //关闭相应的流
            objOut.close();
            fileOut.close();
        }
    }
    catch (Exception e) {
        System.out.println("ERROR ShowSave");
    }
}
//当发生事件的按钮为退出按钮时
else if (ae.getSource().equals(exit)) {
    //显示提示信息
    int j = JOptionPane.showConfirmDialog(this, "真的要退出吗?", "退出",
        JOptionPane.YES_OPTION, JOptionPane.QUESTION_MESSAGE);
    if (j == JOptionPane.YES_OPTION) { //当单击“是”按钮时
        System.exit(0); //退出系统
    }
}
}

```

【代码解析】

- ❑ 当发生动作的按钮为“新游戏”按钮时，首先会通过 `drawChessMan()` 方法相同功能的代码实现重新摆放每个棋子，然后设置变量 `chessPlayClick` 的值为 2 及对象 `text` 的显示文本为“红棋走棋”，最后再通过循环清空集合 `Var` 中的元素。
- ❑ 当发生动作的按钮为“悔棋”按钮时，首先获取上一步骤棋子的相关信息：是否显示信息 `s`、棋子的 `x` 坐标 `x`、`y` 坐标 `y` 和该棋子的索引 `M`。然后重新设置索引为 `M` 的棋子位置，最后通过数组 `play[M]` 元素的名字，决定文本对象 `text` 和 `chessPlayClick` 变量的值。
- ❑ 当发生动作的按钮为“打开”按钮时，首先删除当前两个集合对象 `Var` 和 `FileVar` 中的所有对象，并获取打开文件对话框的文件流对象 `fileIn`，然后通过文件流的过

滤流对象 `bojIn` 读出流中的集合对象 `FileVar`，最后通过循环把没有被吃掉的棋子显示出来。当绘制出所有的棋子后，还需要通过对象 `FileVar` 的最后一个元素来决定应该哪方走棋。

- ❑ 当发生动作的按钮为“保存”按钮时，首先保存当前棋局中所有棋子的信息到集合 `FileVar` 中。棋子信息包含棋子的 x 坐标、 y 坐标和显示信息，还有棋局当前应该哪方走棋的信息。然后通过保存文件对话框，获取输出流对象 `fileOut` 的过滤流对象 `objOut`，把集合 `FileVar` 中的信息保存起来。
- ❑ 当发生动作的按钮为“退出”按钮时，首先会出现需要用户确认的对话框，然后会根据用户的选择进行相应的操作。

📌注意：在 `Vector` 类型的 `FileVar` 集合中，存储了 32 棋子的相关信息，即一个棋子的所有信息存储到一个单位中，而单位包含三个元素，分别表示棋子的 x 坐标、 y 坐标及是否显示。该集合的最后一个元素表示应该哪方走棋。

30.3 象棋游戏项目——规则的内部类

在象棋游戏项目的主类 `ChessMainFrame` 中，创建象棋游戏的各种成员变量，并且通过多线程技术实现棋子的闪烁和各种棋子的规则。棋子规则的内部类的类图如图 30.13 所示。

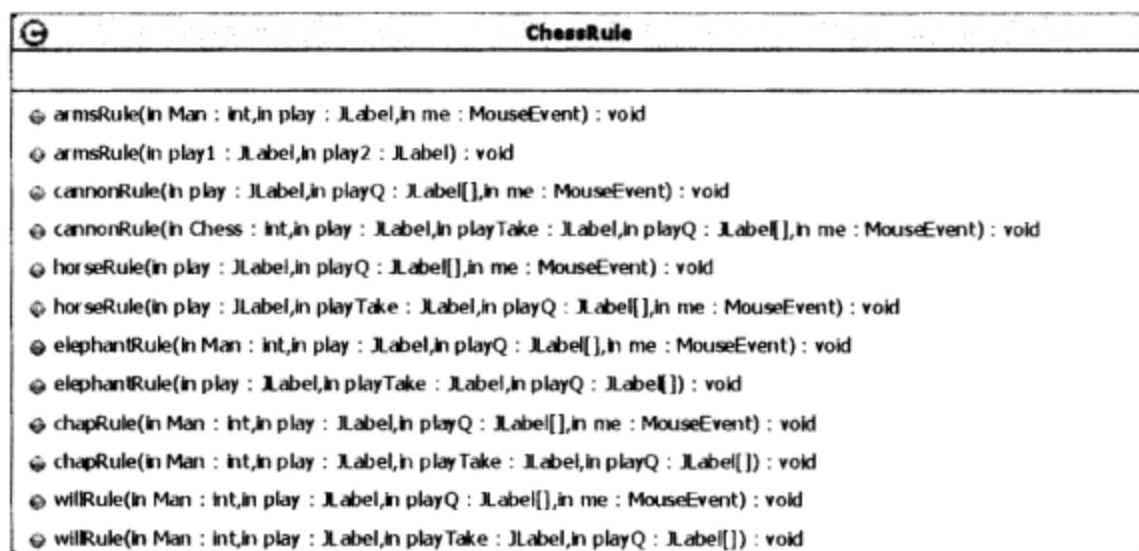


图 30.13 内部类 UML 图

30.3.1 实现卒移动和吃的方法

根据游戏规则，卒的移动范围为任何位置；移动规则是每步只能向前移动一点，过河以后即增加了向左右移动的能力，但不允许向后移动。按照上述规则，黑方棋子中的卒在不过河时只能向下移动，而红方棋子中的卒则正好相反。

在具体实现上述规则时，专门设计了 `armsRule(int Man, JLabel play, MouseEvent me)` 方法来实现卒移动的规则，其中参数 `Man` 表示正在闪烁的棋子索引，参数 `play` 表示闪烁的棋子对象，`me` 为单击事件对象。该方法的具体内容如代码 30.6 所示。

代码 30.6 卒棋移动的方法: armsRule()


```

public void armsRule(int Man, JLabel play, MouseEvent me) {
    if (Man < 21) { //黑棋阵营的卒向下
        //向下移动、得到终点的坐标模糊成合法的坐标
        if ((me.getY() - play.getY()) > 40
            && (me.getY() - play.getY()) < 80
            && (me.getX() - play.getX()) < 30
            && (me.getX() - play.getX()) > 0) {
            //当前记录添加到集合中 (用于悔棋)
            Var.add(String.valueOf(play.isVisible()));
            Var.add(String.valueOf(play.getX()));
            Var.add(String.valueOf(play.getY()));
            Var.add(String.valueOf(Man));
            //修改闪烁棋子的位置
            play.setBounds(play.getX(), play.getY() + 45, 40, 40);
        }
        //当过河后, 得到终点的坐标模糊成合法的坐标, 实现向右移动
        //当过河后, 得到终点的坐标模糊成合法的坐标, 实现向左移动
        else if (play.getY() >= 250 && (me.getX() - play.getX()) >= 30
            && (me.getX() - play.getX()) <= 90) {
            //修改闪烁棋子的位置
            play.setBounds(play.getX() + 48, play.getY(), 40, 40);
        }
        //向左移动、得到终点的坐标模糊成合法的坐标、必须过河
        else if (play.getY() >= 250 && (play.getX() - me.getX()) >= 20
            && (play.getX() - me.getX()) <= 90) {
            //修改闪烁棋子的位置
            play.setBounds(play.getX() - 48, play.getY(), 40, 40);
        }
    }
    else { //红棋阵营的卒向上
        //当前记录添加到集合 (用于悔棋)
        Var.add(String.valueOf(play.isVisible()));
        Var.add(String.valueOf(play.getX()));
        Var.add(String.valueOf(play.getY()));
        Var.add(String.valueOf(Man));
        //向上移动、得到终点的坐标模糊成合法的坐标
        if ((me.getX() - play.getX()) > 0
            && (me.getX() - play.getX()) < 30
            && (play.getY() - me.getY()) > 20
            && play.getY() - me.getY() < 70) {
            //修改闪烁棋子的位置
            play.setBounds(play.getX(), play.getY() - 48, 40, 40);
        }
        //向右移动、得到终点的坐标模糊成合法的坐标、必须过河
        else if (play.getY() <= 210 && (me.getX() - play.getX()) >= 30
            && (me.getX() - play.getX()) <= 90) {
            //修改闪烁棋子的位置
            play.setBounds(play.getX() + 50, play.getY(), 40, 40);
        }
        //向左移动、得到终点的坐标模糊成合法的坐标、必须过河
        else if (play.getY() <= 210 && (play.getX() - me.getX()) >= 20
            && (play.getX() - me.getX()) <= 60) {
            //修改闪烁棋子的位置
            play.setBounds(play.getX() - 52, play.getY(), 40, 40);
        }
    }
}
}

```

【代码解析】

- 当闪烁棋子的索引小于 21 时，则为黑方卒。根据单击点坐标与闪烁棋子坐标间的距离，即两个对象 x 轴间的距离模糊算法，可以算出和 y 轴间的距离模糊算法，可以确定卒是向下，还是过河后的向左或向右移动。当 x 轴间的距离在(0, 30)范围及 y 轴间的距离在(40, 80)时，表示向下移动；当 x 轴间的距离在(30, 90)范围及闪烁棋子 y 轴的坐标大于等于 250 时，表示向右移动；当 x 轴间的距离在(20, 90)范围及闪烁棋子 y 轴的坐标大于等于 250 时，表示向左移动。
- 在具体移动前，需要把闪烁棋子的当前信息存储到集合 `Var` 里，然后把该对象的坐标设置为单击事件对象的坐标。同理，当闪烁棋子为红方棋子中的卒时，也会经过相应的步骤进行操作。

 **注意：** `Vector` 类型的集合 `Var` 中，用于存储 32 棋子的相关信息，即一个棋子的所有信息存储到一个单位中，而单位包含 4 个元素，分别表示棋子是否显示、 x 坐标、 y 坐标和索引。该集合的最后一个元素表示应该哪方走棋。

通过卒移动规则，可以设计出卒吃棋的方法 `armsRule(JLabel play1, JLabel play2)`，其中参数 `play1` 表示闪烁的棋子，而参数 `play2` 表示当前单击的棋子，即实现棋子 `play1` 吃掉 `play2`。该方法的具体内容如代码 30.7 所示。

代码 30.7 卒棋吃棋的方法：`armsRule()`

```
public void armsRule(JLabel play1, JLabel play2) {
    //实现向右吃棋功能
    if ((play2.getX() - play1.getX()) <= 60
        && (play2.getX() - play1.getX()) >= 40
        && (play1.getY() - play2.getY()) < 10
        && (play1.getY() - play2.getY()) > -10 && play2.isVisible()
        && play1.getName().charAt(1) != play2.getName().charAt(1)) {
        //黑方阵营棋右吃棋
        if (play1.getName().charAt(1) == '1'
            && play1.getY() >= 250
            && play1.getName().charAt(1) != play2.getName().charAt(
                1)) {
            play2.setVisible(false);           //不显示被吃棋
            play1.setBounds(play2.getX(), play2.getY(), 40, 40);
                                                    //把对方的位置给自己
        }
        //红方阵营棋左吃棋
        else if (play1.getName().charAt(1) == '2'
            && play1.getY() <= 210
            && play1.getName().charAt(1) != play2.getName().charAt(
                1)) {
            play2.setVisible(false);           //不显示被吃棋
            play1.setBounds(play2.getX(), play2.getY(), 40, 40);
                                                    //把对方的位置给自己
        }
    }
    //实现向左吃棋的功能
    else if ((play1.getX() - play2.getX()) <= 60
        && (play1.getX() - play2.getX()) >= 40
        && (play1.getY() - play2.getY()) < 10
        && (play1.getY() - play2.getY()) > -10 && play2.isVisible())
```

```

        && play1.getName().charAt(1) != play2.getName().charAt(1)) {
//黑方阵营棋左吃棋
if (play1.getName().charAt(1) == '1'
    && play1.getY() >= 250
    && play1.getName().charAt(1) != play2.getName().charAt(
        1)) {
    play2.setVisible(false); //不显示被吃棋
    play1.setBounds(play2.getX(), play2.getY(), 40, 40);
    //把对方的位置给自己
}
//红方阵营棋右吃棋
else if (play1.getName().charAt(1) == '2'
    && play1.getY() <= 210
    && play1.getName().charAt(1) != play2.getName().charAt(
        1)) {
    play2.setVisible(false); //不显示被吃棋
    play1.setBounds(play2.getX(), play2.getY(), 40, 40);
    //把对方的位置给自己
}
}
//实现向前吃棋的功能
else if (play1.getX() - play2.getX() >= -10
    && play1.getX() - play2.getX() <= 10
    && play1.getY() - play2.getY() >= -70
    && play1.getY() - play2.getY() <= 70) {
//黑方阵营棋向前吃棋
if (play1.getName().charAt(1) == '1'
    && play1.getY() < play2.getY()
    && play1.getName().charAt(1) != play2.getName().charAt(
        1)) {
    play2.setVisible(false); //不显示被吃棋
    play1.setBounds(play2.getX(), play2.getY(), 40, 40);
    //把对方的位置给自己
}
//红方阵营棋向前吃棋
else if (play1.getName().charAt(1) == '2'
    && play1.getY() > play2.getY()
    && play1.getName().charAt(1) != play2.getName().charAt(
        1)) {
    play2.setVisible(false); //不显示被吃棋
    play1.setBounds(play2.getX(), play2.getY(), 40, 40);
    //把对方的位置给自己
}
}
//具体实现吃棋功能
//当前记录添加到集合中 (用于悔棋)
Var.add(String.valueOf(play1.isVisible()));
Var.add(String.valueOf(play1.getX()));
Var.add(String.valueOf(play1.getY()));
Var.add(String.valueOf(Man));
//当前记录添加到集合中 (用于悔棋)
Var.add(String.valueOf(play2.isVisible()));
Var.add(String.valueOf(play2.getX()));
Var.add(String.valueOf(play2.getY()));
Var.add(String.valueOf(i));
}

```

【代码解析】

- 当单击棋坐标与闪烁棋子的坐标间的范围：在 x 轴方向为(40, 60), y 轴方向为(-10, 10), 同时这两个对象还不是同一阵营时, 如果为黑方棋子中卒向右吃棋, 这时闪烁棋子的 y 轴坐标需要大于 250, 并且该对象的名字包含“1”字符。如果为红方棋子中卒向左吃棋, 这时闪烁棋子的 y 轴坐标需要小于 250, 并且该对象的名字包含“2”字符。
- 当闪烁棋子的坐标与单击棋子坐标间的范围：在 x 轴方向为(40, 60), y 轴方向为(-10, 10), 同时这两个对象还不是同一阵营时, 如果为黑方棋子中卒向左吃棋, 这时闪烁棋子的 y 轴坐标需要大于 250, 并且该对象的名字包含“1”字符。如果为红方棋子中卒向右吃棋, 这时闪烁棋子的 y 轴坐标需要小于 250, 并且该对象的名字包含“2”字符。
- 当闪烁棋子的坐标与单击棋子坐标间的范围：在 x 轴方向为(-10, 10), y 轴方向为(-70, 70), 如果为黑方棋子中卒向下吃棋, 这时闪烁棋子的 y 轴需要小于单击棋子的 y 轴坐标并且名字包含“1”字符, 并且与单击棋子对象不属于同一阵营。如果为红方棋子中卒向上吃棋, 这时闪烁棋子的 y 轴需要大于单击棋子的 y 轴坐标并且名字包含“2”字符, 并且要与单击棋子对象不属于同一阵营。
- 在具体实现“吃”棋功能时, 首先需要把单击的棋子设置为不可显示, 然后设置闪烁棋子的坐标为单击棋子的坐标, 最后为了能够实现“悔棋”功能, 还需要把闪烁棋子和单击棋子的坐标存储到集合对象 Var 里。

30.3.2 实现炮、车移动和吃的方法

根据游戏规则, 车和炮的移动范围都为任何位置; 但是车的移动规则是可以水平或垂直方向移动任意个无阻碍的点, 而炮的移动规则是虽然跟车很相似, 但它必须跳过一个棋子来吃掉对方的一个棋子。

在具体实现上述规则时, 专门设计了 `cannonRule(JLabel play, JLabel playQ[], MouseEvent me)` 方法来实现车和炮移动的规则, 其中参数 `play` 表示正在闪烁的棋子对象, 参数 `playQ[]` 表示其他棋子对象, `me` 为单击事件对象。该方法的具体内容如代码 30.8 所示。

代码 30.8 炮和车移动的方法: `cannonRule()`

```
public void cannonRule(JLabel play, JLabel playQ[], MouseEvent me) {
    int Count = 0;                                //表示起点和终点之间是否有棋子
    //实现上、下移动功能
    if (play.getX() - me.getX() <= 10 && play.getX() - me.getX() >= -30) {
        //指定所有模糊 y 坐标
        for (int i = 30; i <= 462; i += 48) {
            //移动的 y 坐标是否有指定坐标相近的
            if (i - me.getY() >= -10 && i - me.getY() <= 30) {
                //所有的棋子
                for (int j = 0; j < 32; j++) {
                    //找出在同一条竖线的所有棋子, 不包括自己
                    if (playQ[j].getX() - play.getX() >= -10
                        && playQ[j].getX() - play.getX() <= 10
                        && playQ[j].getName() != play.getName())
```

```

        && playQ[j].isVisible()) {
//从起点到终点 (从左到右)
for (int k = play.getY() + 50; k < i; k += 50) {
//通过大于起点、小于终点的坐标, 可以知道中间是否有棋子
    if (playQ[j].getY() < i
        && playQ[j].getY() > play.getY()) {
//中间有一个棋子, 就不能从这条竖线上过去
        Count++;
        break;
    }
}
//从起点到终点 (从右到左)
for (int k = i + 50; k < play.getY(); k += 50) {
//找起点和终点的棋子
    if (playQ[j].getY() < play.getY()
        && playQ[j].getY() > i) {
        Count++;
        break;
    }
}
}
//起点和终点没有棋子就可以移动了
if (Count == 0) {
//当前记录添加到集合 (用于悔棋)
Var.add(String.valueOf(play.isVisible()));
Var.add(String.valueOf(play.getX()));
Var.add(String.valueOf(play.getY()));
Var.add(String.valueOf(Man));
play.setBounds(play.getX(), i - 17, 40, 40);
break;
}
}
}
//左、右移动
else if (play.getY() - me.getY() >= -35
        && play.getY() - me.getY() <= 10) {
//指定所有模糊 x 坐标
for (int i = 30; i <= 420; i += 48) {
//移动的 x 坐标是否有指定坐标相近的
    if (i - me.getX() >= -35 && i - me.getX() <= 10) {
//所有的棋子
        for (int j = 0; j < 32; j++) {
//找出在同一条横线的所有棋子, 不包括自己
            if (playQ[j].getY() - play.getY() >= -10
                && playQ[j].getY() - play.getY() <= 10
                && playQ[j].getName() != play.getName()
                && playQ[j].isVisible()) {
//从起点到终点 (从上到下)
                for (int k = play.getX() + 50; k < i; k += 50) {
//通过大于起点、小于终点的坐标, 可以知道中间是否有棋子
                    if (playQ[j].getX() < i
                        && playQ[j].getX() > play.getX()) {
//中间有一个棋子就不能从这条横线上过去
                        Count++;
                        break;
                    }
                }
//从起点到终点 (从下到上)
            }
        }
    }
}
}

```



```

        for (int k = i + 50; k < play.getX(); k += 50) {
            //找起点和终点的棋子
            if (playQ[j].getX() < play.getX()
                && playQ[j].getY() > i) {
                Count++;
                break;
            }
        }
    }
}

//起点和终点没有棋子
if (Count == 0) {
    //当前记录添加到集合中（用于悔棋）
    Var.add(String.valueOf(play.isVisible()));
    Var.add(String.valueOf(play.getX()));
    Var.add(String.valueOf(play.getY()));
    Var.add(String.valueOf(Man));
    play.setBounds(i - 20, play.getY(), 40, 40);
    break;
}
}
}
}

```

【代码解析】

- ❑ 当闪烁棋子的索引小于 21 时, 则为黑方卒。根据单击点坐标与闪烁棋子坐标间的距离, 即两个对象间 x 轴间的距离模糊算法, 可以算出和 y 轴间的距离模糊算法, 可以确定卒是向下, 还是过河后的向左或向右移动。当 x 轴间的距离在 $(0, 30)$ 范围并且 y 轴间的距离在 $(40, 80)$ 时, 表示向下移动; 当 x 轴间的距离在 $(30, 90)$ 范围并且闪烁棋子 y 轴的坐标大于等于 250 时, 表示向右移动; 当 x 轴间的距离在 $(20, 90)$ 范围和闪烁棋子 y 轴的坐标大于等于 250 时, 表示向左移动。
- ❑ 在具体移动前, 需要把闪烁棋子的当前信息存储到集合 Var 里, 然后把该对象的坐标设置为单击事件对象的坐标。同理, 当闪烁棋子为红方棋子中的卒时, 也会经过相应的步骤进行操作。

🔔注意: Vector 类型的集合 Var 中, 用于存储 32 棋子的相关信息, 即一个棋子的所有信息存储到一个单位中, 而单位包含 4 个元素, 分别表示棋子是否显示、x 坐标、y 坐标和索引。该集合的最后一个元素表示该哪方走棋。

通过卒移动规则，可以设计出卒吃棋的方法 `armsRule(JLabel play1, JLabel play2)`，其中参数 `play1` 表示闪烁的棋子，而参数 `play2` 表示当前单击的棋子，即实现棋子 `play1` 吃掉 `play2`。该方法的具体内容如代码 30.9 所示。

代码 30.9 炮和车吃棋的方法: cannonRule()

```
public void cannonRule(int Chess, JLabel play, JLabel playTake,
    JLabel playQ[], MouseEvent me) {
    //起点和终点之间是否有棋子
    int Count = 0;
    //所有的棋子
    for (int j = 0; j < 32; j++) {
```



```

//找出在同一条竖线的所有棋子, 不包括自己
if (playQ[j].getX() - play.getX() >= -10
    && playQ[j].getX() - play.getX() <= 10
    && playQ[j].getName() != play.getName()
    && playQ[j].isVisible()) {
    //自己是起点, 被吃的是终点 (从上到下)
    for (int k = play.getY() + 50; k < playTake.getY(); k += 50) {
        //通过大于起点、小于终点的坐标, 可以知道中间是否有棋子
        if (playQ[j].getY() < playTake.getY()
            && playQ[j].getY() > play.getY()) {
            //计算起点和终点的棋子个数
            Count++;
            break;
        }
    }
    //自己是起点, 被吃的是终点 (从下到上)
    for (int k = playTake.getY(); k < play.getY(); k += 50) {
        //找起点和终点的棋子
        if (playQ[j].getY() < play.getY()
            && playQ[j].getY() > playTake.getY()) {
            Count++;
            break;
        }
    }
}
//找出在同一条竖线的所有棋子, 不包括自己
else if (playQ[j].getY() - play.getY() >= -10
    && playQ[j].getY() - play.getY() <= 10
    && playQ[j].getName() != play.getName()
    && playQ[j].isVisible()) {
    //自己是起点, 被吃的是终点 (从左到右)
    for (int k = play.getX() + 50; k < playTake.getX(); k += 50) {
        //通过大于起点、小于终点的坐标, 可以知道中间是否有棋子
        if (playQ[j].getX() < playTake.getX()
            && playQ[j].getX() > play.getX()) {
            Count++;
            break;
        }
    }
    //自己是起点, 被吃的是终点 (从右到左)
    for (int k = playTake.getX(); k < play.getX(); k += 50) {
        //找起点和终点的棋子
        if (playQ[j].getX() < play.getX()
            && playQ[j].getX() > playTake.getX()) {
            Count++;
            break;
        }
    }
}
}
}
//炮吃棋和车吃棋的规则
if (Count == 1 && Chess == 0
    && playTake.getName().charAt(1) != play.getName().charAt(1)) {
    //当前记录添加到集合中 (用于悔棋)
    Var.add(String.valueOf(play.isVisible()));
    Var.add(String.valueOf(play.getX()));
    Var.add(String.valueOf(play.getY()));
    Var.add(String.valueOf(Man));
    //当前记录添加到集合中 (用于悔棋)
}

```

```

        Var.add(String.valueOf(playTake.isVisible()));
        Var.add(String.valueOf(playTake.getX()));
        Var.add(String.valueOf(playTake.getY()));
        Var.add(String.valueOf(i));
        playTake.setVisible(false);
        play.setBounds(playTake.getX(), playTake.getY(), 40, 40);
    }
    //炮吃棋和车吃棋的规则
    else if (Count == 0 && Chess == 1
        && playTake.getName().charAt(1) != play.getName().charAt(1)) {
        //当前记录添加到集合中（用于悔棋）
        Var.add(String.valueOf(play.isVisible()));
        Var.add(String.valueOf(play.getX()));
        Var.add(String.valueOf(play.getY()));
        Var.add(String.valueOf(Man));

        //当前记录添加到集合中（用于悔棋）
        Var.add(String.valueOf(playTake.isVisible()));
        Var.add(String.valueOf(playTake.getX()));
        Var.add(String.valueOf(playTake.getY()));
        Var.add(String.valueOf(i));
        playTake.setVisible(false);
        play.setBounds(playTake.getX(), playTake.getY(), 40, 40);
    }
}

```

【代码解析】

- ❑ 当单击棋坐标与闪烁棋子的坐标间的范围时：在 x 轴方向为(40, 60), y 轴方向为(-10, 10)，同时这两个对象还不是同一阵营时，如果为黑方棋子中卒向右吃棋，这时闪烁棋子的 y 轴坐标需要大于 250，并且该对象的名字包含“1”字符。如果为红方棋子中卒向左吃棋，这时闪烁棋子的 y 轴坐标需要小于 250，并且该对象的名字包含“2”字符。
- ❑ 当闪烁棋子的坐标与单击棋子坐标间的范围：在 x 轴方向为(40, 60), y 轴方向为(-10, 10)，同时这两个对象还不是同一阵营时，如果为黑方棋子中卒向左吃棋，这时闪烁棋子的 y 轴坐标需要大于 250，并且该对象的名字包含“1”字符。如果为红方棋子中卒向右吃棋，这时闪烁棋子的 y 轴坐标需要小于 250，并且该对象的名字包含“2”字符。
- ❑ 当闪烁棋子的坐标与单击棋子坐标间的范围：在 x 轴方向为(-10, 10), y 轴方向为(-70, 70)，如果为黑方棋子中卒向下吃棋，这时闪烁棋子的 y 轴需要小于单击棋子的 y 轴坐标并且名字包含“1”字符，并且要与单击棋子对象不属于同一阵营。如果为红方棋子中卒向上吃棋，这时闪烁棋子的 y 轴需要大于单击棋子的 y 轴坐标并且名字包含“2”字符，并且要与单击棋子对象不属于同一阵营。
- ❑ 在具体实现“吃”棋功能时，首先需要把单击棋子设置为不可显示，然后设置闪烁棋子的坐标为单击棋子的坐标，最后为了能够实现“悔棋”功能，还需要把闪烁棋子和单击棋子的坐标存储到集合对象 `var` 中。

30.3.3 实现马移动和吃的方法

根据游戏规则，马的移动范围为任何位置；而移动规则是每步只可以水平或垂直移动

一点, 再按对角线方面向左或者右移动, 同时在移动的过程中不能够穿越障碍。即在没有障碍物的情况下只能实现 (上移, 左边)、(左移, 上边)、(下移, 右边)、(上移, 右边)、(下移, 左边)、(右移, 上边)、(右移, 下边) 和 (左移、下边) 方向的移动。

在具体实现上述规则时, 专门设计了 horseRule(JLabel play, JLabel playQ[], MouseEvent me) 方法来实现马移动的规则, 其中参数 play 表示闪烁的棋子对象, playQ[] 表示其他棋子对象, me 为单击事件对象。该方法的具体内容如代码 30.10 所示。

代码 30.10 马移动的方法: ChessMainFrame.java

```
public void horseRule(JLabel play, JLabel playQ[], MouseEvent me) {
    //创建成员变量
    int Ex = 0, Ey = 0, Move = 0;           //保存准确坐标和障碍坐标的变量
    //用来实现向 (上移, 左边) 移动
    if (play.getX() - me.getX() >= 10 && play.getX() - me.getX() <= 50
        && play.getY() - me.getY() >= 60
        && play.getY() - me.getY() <= 100) {
        //获取准确的 y 坐标
        for (int i = 30; i <= 462; i += 48) {
            //移动的 y 坐标是否有指定坐标相近的
            if (i - me.getY() >= -10 && i - me.getY() <= 30) {
                Ey = i;                       //为变量 Ey 赋值
                break;
            }
        }
        //获取准确的 x 坐标
        for (int i = 30; i <= 420; i += 48) {
            //移动的 x 坐标是否有指定坐标相近的
            if (i - me.getX() >= -35 && i - me.getX() <= 10) {
                Ex = i;                       //为变量 Ex 赋值
                break;
            }
        }
        //判断正前方是否有其他棋子
        for (int i = 0; i < 32; i++) {
            if (playQ[i].isVisible())
                && play.getX() - playQ[i].getX() <= 10
                && play.getX() - playQ[i].getX() >= -10
                && play.getY() - playQ[i].getY() >= 40
                && play.getY() - playQ[i].getY() <= 60) {
                Move = 1;                     //为变量 Move 赋值
                break;
            }
        }
        //可以移动该棋子
        if (Move == 0) {
            //当前记录添加到集合中 (用于悔棋)
            Var.add(String.valueOf(play.isVisible()));
            Var.add(String.valueOf(play.getX()));
            Var.add(String.valueOf(play.getY()));
            Var.add(String.valueOf(Move));
            //重新设置闪烁棋子的坐标
            play.setBounds(Ex - 20, Ey - 17, 40, 40);
        }
    }
    //用来实现向 (左移, 上边) 移动
```

```

else if (play.getY() - me.getY() >= 10
        && play.getY() - me.getY() <= 50
        && play.getX() - me.getX() >= 60
        && play.getX() - me.getX() <= 100) {
    ... //省略部分代码
}
    ... //省略部分代码
}
//用来实现向（下移，右边）移动
else if (me.getY() - play.getY() >= 100
        && me.getY() - play.getY() <= 130
        && me.getX() - play.getX() <= 70
        && me.getX() - play.getX() >= 30) {
    ... //省略部分代码
}
    ... //省略部分代码
}
//用来实现向（上移，右边）移动
else if (play.getY() - me.getY() >= 60
        && play.getY() - me.getY() <= 100
        && me.getX() - play.getX() <= 80
        && me.getX() - play.getX() >= 50) {
    ... //省略部分代码
}
    ... //省略部分代码
}
//用来实现向（下移，左边）移动
else if (me.getY() - play.getY() >= 100
        && me.getY() - play.getY() <= 140
        && play.getX() - me.getX() <= 50
        && play.getX() - me.getX() >= 10) {
    ... //省略部分代码
}
    ... //省略部分代码
}
//用来实现向（右移，上边）移动
else if (play.getY() - me.getY() >= 10
        && play.getY() - me.getY() <= 50
        && me.getX() - play.getX() <= 140
        && me.getX() - play.getX() >= 100) {
    ... //省略部分代码
}
    ... //省略部分代码
}
//右移下边
else if (me.getY() - play.getY() >= 60
        && me.getY() - play.getY() <= 90
        && me.getX() - play.getX() <= 130
        && me.getX() - play.getX() >= 100) {
    ... //省略部分代码
}
    ... //省略部分代码
}
//左移、下边
else if (me.getY() - play.getY() >= 50
        && me.getY() - play.getY() <= 90
        && play.getX() - me.getX() <= 100
        && play.getX() - me.getX() >= 50) {

```

```

        ...                                     //省略部分代码
    }
    ...                                     //省略部分代码
}
}

```

【代码解析】

- 当闪烁棋子的坐标与单击点坐标间的范围: 在 x 轴方向为 $[5, 10]$, y 轴方向为 $[60, 100]$ 时, 表示象棋可以实现 (上移, 左边) 移动。这时如果闪烁棋子的正前方没有其他棋子, 则可以实现移动。在具体移动时, 首先需要把象棋的信息存储到集合 Var 中, 然后再设置象棋的 x 轴坐标为准确的 x 轴坐标 Ex 减去 20, y 轴的坐标值为准确的 y 轴坐标 Ey 减去 17。
- 当闪烁棋子的坐标与单击点坐标间的范围: 在 x 轴方向为 $[60, 100]$, y 轴方向为 $[10, 50]$ 时, 表示象棋可以实现 (左移, 上边) 移动。这时如果闪烁棋子的正前方没有其他棋子, 则可以实现移动。在具体移动时, 首先需要把象棋的信息存储到集合 Var 中, 然后再设置象棋的 x 轴坐标为准确的 x 轴坐标 Ex 减去 20, y 轴的坐标值为准确的 y 轴坐标 Ey 减去 17。
- 当单击点坐标与闪烁棋子的坐标间的范围: 在 x 轴方向为 $[100, 130]$, y 轴方向为 $[30, 70]$ 时, 表示象棋可以实现 (下移, 右边) 移动。这时如果闪烁棋子的正前方没有其他棋子, 则可以实现移动。在具体移动时, 首先需要把象棋的信息存储到集合 Var 中, 然后再设置象棋的 x 轴坐标为准确的 x 轴坐标 Ex 减去 20, y 轴的坐标值为准确的 y 轴坐标 Ey 减去 17。
- 当单击点坐标与闪烁棋子的坐标间的范围: 在 x 轴方向为 $[100, 130]$, y 轴方向为 $[60, 90]$ 时, 表示象棋可以实现 (右移, 下边) 移动。这时如果闪烁棋子的正前方没有其他棋子, 则可以实现移动。在具体移动时, 首先需要把象棋的信息存储到集合 Var 中, 然后再设置象棋的 x 轴坐标为准确的 x 轴坐标 Ex 减去 20, y 轴的坐标值为准确的 y 轴坐标 Ey 减去 17。
- 当单击点坐标与闪烁棋子 x 坐标间的范围为 $[50, 80]$, 闪烁棋子坐标与单击点 y 坐标间的范围为 $[60, 100]$ 时, 表示象棋可以实现 (上移, 右边) 移动。这时如果闪烁棋子的正前方没有其他棋子, 则可以移动。在具体移动时, 首先需要把象棋的信息存储到集合 Var 中, 然后再设置象棋的 x 轴坐标为准确的 x 轴坐标 Ex 减去 20, y 轴的坐标值为准确的 y 轴坐标 Ey 减去 17。
- 当单击点坐标与闪烁棋子 x 坐标间的范围为 $[100, 140]$, 闪烁棋子坐标与单击点 y 坐标间的范围为 $[10, 50]$ 时, 表示象棋可以实现 (右移, 上边) 移动。这时如果闪烁棋子的正前方没有其他棋子, 则可以移动。在具体移动时, 首先需要把象棋的信息存储到集合 Var 中, 然后再设置象棋的 x 轴坐标为准确的 x 轴坐标 Ex 减去 20, y 轴的坐标值为准确的 y 轴坐标 Ey 减去 17。
- 当闪烁棋子坐标与单击点 x 坐标间的范围为 $[50, 100]$, 单击点坐标与闪烁棋子 y 坐标间的范围为 $[100, 140]$ 时, 表示象棋可以实现 (下移, 左边) 移动。这时如果闪烁棋子的正前方没有其他棋子, 则可以移动。在具体移动时, 首先需要把象棋的信息存储到集合 Var 中, 然后再设置象棋的 x 轴坐标为准确的 x 轴坐标 Ex 减去 20, y 轴的坐标值为准确的 y 轴坐标 Ey 减去 17。

- 当闪烁棋子坐标与单击点 x 坐标间的范围为 $[50, 100]$ ，单击点坐标与闪烁棋子 y 坐标间的范围为 $[50, 90]$ 时，表示象棋可以实现（左移，下边）移动。这时如果闪烁棋子的正前方没有其他棋子，则可以移动。在具体移动时，首先需要把象棋的信息存储到集合 Var 中，然后再设置象棋的 x 轴坐标为准确的 x 轴坐标 Ex 减去 20， y 轴的坐标值为准确的 y 轴坐标 Ey 减去 17。

通过马移动规则，可以设计出马吃棋的方法 horseRule (JLabel play, JLabel playTake, JLabel playQ[]), 其中参数 play 表示正在闪烁的棋子对象，playTake 为要吃掉的对象，参数 playQ[]表示其他棋子对象。该方法的具体内容如代码 30.11 所示。

代码 30.11 马吃棋的方法: horseRule()

```
public void horseRule(JLabel play, JLabel playTake, JLabel playQ[],
    MouseEvent me) {
    //创建成员变量
    int Move = 0; //表示闪烁棋子与被吃棋子间拥有棋子的个数
    boolean Chess = false; //表示是否可以吃棋的变量
    //实现上移、左吃功能
    if (play.getName().charAt(1) != playTake.getName().charAt(1)
        && play.getX() - playTake.getX() >= 10
        && play.getX() - playTake.getX() <= 55
        && play.getY() - playTake.getY() >= 60
        && play.getY() - playTake.getY() <= 105) {
        //判断正前方是否有其他棋子
        ...
        Chess = true; //设置变量 Chess 的值
    }
    //实现上移、右吃功能
    else if (play.getY() - playTake.getY() >= 80
        && play.getY() - playTake.getY() <= 110
        && playTake.getX() - play.getX() <= 60
        && playTake.getX() - play.getX() >= 40) {
        //判断正前方是否有其他棋子
        ...
        Chess = true; //设置变量 Chess 的值
    }
    //实现左移、上吃功能
    else if (play.getY() - playTake.getY() >= 40
        && play.getY() - playTake.getY() <= 60
        && play.getX() - playTake.getX() >= 90
        && play.getX() - playTake.getX() <= 110) {
        //判断正左方是否有其他棋子
        ...
        Chess = true; //设置变量 Chess 的值
    }
    //实现左移、下吃功能
    else if (playTake.getY() - play.getY() >= 30
        && playTake.getY() - play.getY() <= 60
        && play.getX() - playTake.getX() <= 120
        && play.getX() - playTake.getX() >= 80) {
        //判断正左方是否有其他棋子
        ...
        Chess = true; //设置变量 Chess 的值
    }
    //实现右移、上吃功能
```



```

else if (play.getY() - playTake.getY() >= 30
        && play.getY() - playTake.getY() <= 60
        && playTake.getX() - play.getX() <= 120
        && playTake.getX() - play.getX() >= 80) {
    //判断正右方是否有其他棋子
    ...
    Chess = true;                //设置变量 Chess 的值
}
//实现右移、下吃功能
else if (playTake.getY() - play.getY() >= 30
        && playTake.getY() - play.getY() <= 60
        && playTake.getX() - play.getX() <= 120
        && playTake.getX() - play.getX() >= 80) {
    //判断正右方是否其他棋子
    ...
}
    Chess = true;                //设置变量 Chess 的值
}
//实现下移、左吃功能
else if (playTake.getY() - play.getY() >= 80
        && playTake.getY() - play.getY() <= 120
        && play.getX() - playTake.getX() <= 60
        && play.getX() - playTake.getX() >= 30) {
    //判断正下方是否其他的棋子
    ...
    Chess = true;                //设置变量 Chess 的值
}
//实现下移、右吃功能
else if (playTake.getY() - play.getY() >= 80
        && playTake.getY() - play.getY() <= 120
        && playTake.getX() - play.getX() <= 60
        && playTake.getX() - play.getX() >= 40) {
    //判断正下方是否其他的棋子
    ...
    Chess = true;                //设置变量 Chess 的值
}
//实现吃棋的具体功能
if (Chess && Move == 0
    && playTake.getName().charAt(1) != play.getName().charAt(1)) {
    //添加闪烁棋子的信息到集合中 (用于悔棋)
    Var.add(String.valueOf(play.isVisible()));
    Var.add(String.valueOf(play.getX()));
    Var.add(String.valueOf(play.getY()));
    Var.add(String.valueOf(Move));
    //添加被吃棋子信息到集合中 (用于悔棋)
    Var.add(String.valueOf(playTake.isVisible()));
    Var.add(String.valueOf(playTake.getX()));
    Var.add(String.valueOf(playTake.getY()));
    Var.add(String.valueOf(i));
    playTake.setVisible(false);    //不显示被吃的棋子
    //重新设置闪烁棋子的位置
    play.setBounds(playTake.getX(), playTake.getY(), 40, 40);
}
}

```

【代码解析】

□ 当闪烁棋子的坐标与被吃棋子坐标间的范围: 在 x 轴方向为[10, 55], y 轴方向为[60,

105]时,同时在正前方没有其他棋子,表示象棋可以实现上移左吃。在具体实现吃棋过程时,首先需要判断可以吃棋的变量 `chess` 是否为 `true`,同时象棋和被吃的棋不属于同一阵营,然后才可以吃棋,即先把象棋和被吃的棋的信息存储到集合 `Var` 中,然后设置被吃的棋不显示,最后把象棋的坐标修改成被吃掉的棋的坐标。

- ❑ 当闪烁棋子的坐标与被吃棋子坐标间的范围:在 x 轴方向为[90, 110], y 轴方向为[40, 60]时,同时在正前方没有其他棋子,则表示象棋可以实现左移上吃。在具体实现吃棋过程时,首先需要判断可以吃棋的变量 `chess` 是否为 `true`,同时象棋和被吃掉的棋不属于同一阵营,然后才可以吃棋,即先把象棋和被吃的棋的信息存储到集合 `Var` 中,然后设置被吃的棋不显示,最后把象棋的坐标修改成被吃掉的棋的坐标。
- ❑ 当被吃棋子的坐标与闪烁棋子的坐标间范围:在 x 轴方向为[40, 60], y 轴方向为[-10, 10]时,同时在正前方没有其他棋子,则表示象棋可以实现下移右吃。在具体实现吃棋过程时,首先需要判断可以吃棋的变量 `chess` 是否为 `true`,同时象棋和被吃的棋不属于同一阵营,然后才可以吃棋,即先把象棋和被吃的棋的信息存储到集合 `Var` 中,然后再设置被吃的棋不显示,最后把象棋的坐标修改成被吃掉的棋的坐标。
- ❑ 当被吃棋子坐标与闪烁棋子的坐标间范围:在 x 轴方向为[100, 130], y 轴方向为[60, 90]时,同时在正前方没有其他棋子,则表示象棋可以实现右移下吃。在具体实现吃棋过程时,首先需要判断可以吃棋的变量 `chess` 是否为 `true`,同时象棋和被吃的棋不属于同一阵营,然后才可以吃棋,即先把象棋和被吃的棋的信息存储到集合 `Var` 中,然后再设置被吃的棋不显示,最后把象棋的坐标修改成被吃掉的棋的坐标。
- ❑ 当被吃棋子坐标与闪烁棋子 x 坐标间的范围为[40, 60],闪烁棋子坐标与被吃棋子 y 坐标间的范围为[80, 110]时,同时在正前方没有其他棋子,则表示象棋可以实现上移右吃。在具体实现吃棋过程时,首先需要判断可以吃棋的变量 `chess` 是否为 `true`,同时象棋和被吃的棋不属于同一阵营,然后才可以吃棋,即先把象棋和被吃棋的信息存储到集合 `Var` 中,然后再设置被吃的棋不显示,最后把象棋的坐标修改成被吃掉的棋的坐标。
- ❑ 当被吃棋子坐标与闪烁棋子 x 坐标间的范围为[80, 120],闪烁棋子坐标与被吃棋子 y 坐标间的范围为[30, 60]时,同时在正前方没有其他棋子,则表示象棋可以实现右移上吃。在具体实现吃棋过程时,首先需要判断可以吃棋的变量 `chess` 是否为 `true`,同时象棋和被吃的棋不属于同一阵营,然后才可以吃棋,即先把象棋和被吃的棋的信息存储到集合 `Var` 中,然后再设置被吃棋不显示,最后把象棋的坐标修改成被吃掉的棋的坐标。
- ❑ 当闪烁棋子坐标与被吃棋子 x 坐标间的范围为[80, 120],被吃棋子坐标与闪烁棋子 y 坐标间的范围为[-10, 10]时,同时在正前方没有其他棋子,则表示象棋可以实现左移下吃。在具体实现吃棋过程时,首先需要判断可以吃棋的变量 `chess` 是否为 `true`,同时象棋和被吃的棋不属于同一阵营,然后才可以吃棋,即先把象棋和被吃的棋的信息存储到集合 `Var` 中,然后再设置被吃的棋不显示,最后把象棋的坐标修改成被吃掉的棋的坐标。
- ❑ 当闪烁棋子坐标与被吃棋子 x 坐标间的范围为[30, 60],被吃棋子坐标与闪烁棋子 y 坐标间的范围为[80, 120]时,同时在正前方没有其他棋子,则表示象棋可以实现下

移左吃。在具体实现吃棋过程时,首先需要判断可以吃棋的变量 chess 是否为 true,同时象棋和被吃的棋不属于同一阵营,然后才可以吃棋,即先把象棋和被吃的棋的信息存储到集合 Var 中,然后再设置被吃的棋不显示,最后把象棋的坐标修改成被吃掉的棋的坐标。

30.3.4 实现象移动和吃的方法

根据游戏规则,象的移动范围只能为河界的一侧的位置;移动规则是每步只可以沿对角线方向移动两点,另外,在移动的过程中不能穿越障碍。即在没有障碍物的情况下,只能实现向左上、左下、右上和右下方向的移动。

在具体实现上述规则时,专门设计了 elephantRule(int Man, JLabel play, JLabel playQ[], MouseEvent me)方法来实现象移动的规则,其中参数 Man 表示正在闪烁的棋子索引,参数 play 表示闪烁的棋子对象,playQ[]表示其他棋子对象,me 为单击事件对象。该方法的具体内容如代码 30.12 所示。

代码 30.12 实现象移动的方法: elephantRule()

```
public void elephantRule(int Man, JLabel play, JLabel playQ[],
    MouseEvent me) {
    //创建成员变量
    int Ex = 0;                //准确的 x 轴坐标
    int Ey = 0;                //准确的 y 轴坐标
    int Move = 0;              //是否可以移动的变量
    //实现向(上左)移动
    if (play.getX() - me.getX() <= 90 && play.getX() - me.getX() >= 60
        && play.getY() - me.getY() <= 100
        && play.getY() - me.getY() >= 70) {
        //获取单击点准确的 y 坐标
        for (int i = 30; i <= 462; i += 48) {
            if (i - me.getY() >= -10 && i - me.getY() <= 30) {
                Ey = i;                //为变量 Ey 赋值
                break;
            }
        }
        //获取单击点准确的 x 坐标
        for (int i = 30; i <= 420; i += 48) {
            if (i - me.getX() >= -35 && i - me.getX() <= 10) {
                Ex = i;                //为变量 Ex 赋值
                break;
            }
        }
        //设置变量 Move 的值
        for (int i = 0; i < 32; i++) {
            if (playQ[i].isVisible()
                && play.getX() - playQ[i].getX() >= 10
                && play.getX() - playQ[i].getX() <= 50
                && play.getY() - playQ[i].getY() >= 40
                && play.getY() - playQ[i].getY() <= 60) {
                Move++;                //修改变量 Move 的值
                break;
            }
        }
    }
}
```

```

//当与红方阵营的象相近时
if (Move == 0 && Ey > 230 && Man > 9) {
    //添加闪烁棋子信息到集合 Var 中 (用于悔棋)
    Var.add(String.valueOf(play.isVisible()));
    Var.add(String.valueOf(play.getX()));
    Var.add(String.valueOf(play.getY()));
    Var.add(String.valueOf(Man));
    //重新设置闪烁棋子的坐标
    play.setBounds(Ex - 20, Ey - 15, 40, 40);
}
//当与黑方阵营的象相近时
else if (Move == 0 && Ey < 270 && Man < 10) {
    //添加闪烁棋子信息到集合 Var 中 (用于悔棋)
    Var.add(String.valueOf(play.isVisible()));
    Var.add(String.valueOf(play.getX()));
    Var.add(String.valueOf(play.getY()));
    Var.add(String.valueOf(Man));
    //重新设置闪烁棋子的坐标
    play.setBounds(Ex - 20, Ey - 15, 40, 40);
}
}
//实现向 (上右) 移动
else if (play.getY() - me.getY() <= 100
        && play.getY() - me.getY() >= 70
        && me.getX() - play.getX() >= 100
        && me.getX() - play.getX() <= 140) {
    //获取单击点准确的 Y 坐标
    ...
    //获取单击点准确的 X 坐标
    ...
    //设置变量 Move 的值
    ...
    //当与红方阵营的象相近时
    ...
    //当与黑方阵营的象相近时
    ...
}
//实现向 (下左) 移动
else if (play.getX() - me.getX() <= 100
        && play.getX() - me.getX() >= 60
        && me.getY() - play.getY() <= 130
        && me.getY() - play.getY() >= 100) {
    //获取单击点准确的 y 坐标
    for (int i = 30; i <= 462; i += 48) {
        if (i - me.getY() >= -10 && i - me.getY() <= 30) {
            Ey = i; //设置变量 Ey 的值
            break;
        }
    }
    //获取单击点准确的 x 坐标
    for (int i = 30; i <= 420; i += 48) {
        if (i - me.getX() >= -35 && i - me.getX() <= 10) {
            Ex = i; //设置变量 Ex 的值
            break;
        }
    }
    //设置变量 Move 的值
    for (int i = 0; i < 32; i++) {

```

```

        if (playQ[i].isVisible())
            && play.getX() - playQ[i].getX() >= 10
            && play.getX() - playQ[i].getX() <= 60
            && play.getY() - playQ[i].getY() <= -40
            && play.getY() - playQ[i].getY() >= -60) {
                Move++; //更新变量 Move 的值
                break;
            }
    }
    //当与红方阵营的象相近时
    if (Move == 0 && Ey > 230 && Man > 9) {
        //添加闪烁棋子信息到集合 Var 中 (用于悔棋)
        Var.add(String.valueOf(play.isVisible()));
        Var.add(String.valueOf(play.getX()));
        Var.add(String.valueOf(play.getY()));
        Var.add(String.valueOf(Man));
        //设置闪烁棋子的坐标
        play.setBounds(Ex - 20, Ey - 15, 40, 40);
    }
    //当与黑方阵营的象相近时
    else if (Move == 0 && Ey < 270 && Man < 10) {
        //添加闪烁棋子信息到集合 Var 中 (用于悔棋)
        Var.add(String.valueOf(play.isVisible()));
        Var.add(String.valueOf(play.getX()));
        Var.add(String.valueOf(play.getY()));
        Var.add(String.valueOf(Man));
        //设置闪烁棋子的坐标
        play.setBounds(Ex - 20, Ey - 15, 40, 40);
    }
}
//实现向 (下右) 移动
else if (me.getX() - play.getX() >= 100
        && me.getX() - play.getX() <= 130
        && me.getY() - play.getY() >= 100
        && me.getY() - play.getY() <= 130) {
    //获取单击点准确的 y 坐标
    ...
    //获取单击点准确的 x 坐标
    ...
    //设置变量 Move 的值
    ...
    //当与红方阵营的象相近时
    ...
    //当与黑方阵营的象相近时
    ...
}
}

```

【代码解析】

- 当闪烁棋子的坐标与单击点坐标间的范围：在 x 轴方向为 $[60, 90]$ ， y 轴方向为 $[70, 100]$ 时，表示象棋可以实现向左上移动。如果闪烁棋子的索引 man 小于 10 并且单击点的准确的 y 轴坐标 Ey 小于 270，同时闪烁棋子与单击点之间没有其他棋子时，则可以实现黑方象棋向左上移动。如果闪烁棋子的索引 man 大于 9 并且单击点的准确的 y 轴坐标 Ey 大于 230，同时闪烁棋子与单击点之间没有其他棋子时，则可以实现红方象棋向左上移动。在具体向左上移动时，首先需要把象棋的信息存储

到集合 Var 中, 然后设置象棋的 x 轴坐标为准确的 x 轴坐标 Ex 减去 20, y 轴的坐标值为准确的 y 轴坐标 Ey 减去 15。

- 当单击点坐标与闪烁棋子 x 坐标间的范围为[100, 140], 闪烁棋子坐标与单击点 y 坐标间的范围为[70, 100]时, 表示象棋可以实现向右上移动。如果闪烁棋子的索引 man 小于 10 并且单击点的准确的 y 轴坐标 Ey 小于 270, 同时闪烁棋子与单击点之间没有其他棋子时, 则可以实现黑方象棋向右上移动。如果闪烁棋子的索引 man 大于 9 并且单击点的准确的 y 轴坐标 Ey 大于 230, 同时闪烁棋子与单击点之间没有其他棋子时, 则可以实现红方象棋向右上移动。在具体向右上移动时, 首先需要把象棋的信息存储到集合 Var 中, 然后再设置象棋的 x 轴坐标为准确的 x 轴坐标 Ex 减去 20, y 轴的坐标值为准确的 y 轴坐标 Ey 减去 15。
- 当闪烁棋子的坐标与单击点 x 坐标间的范围为[60, 100], 单击点坐标与闪烁棋子 y 坐标间的范围为[100, 130]时, 表示象棋可以实现向左下移动。如果闪烁棋子的索引 man 小于 10 并且单击点的准确的 y 轴坐标 Ey 小于 270, 同时闪烁棋子与单击点之间没有其他棋子时, 则可以实现黑方象棋向左下移动。如果闪烁棋子的索引 man 大于 9 并且单击点的准确的 y 轴坐标 Ey 大于 230, 同时闪烁棋子与单击点之间没有其他棋子时, 则可以实现红方象棋向左下移动。在具体向右上移动时, 首先需要把象棋的信息存储到集合 Var 中, 然后再设置象棋的 x 轴坐标为准确的 x 轴坐标 Ex 减去 20, y 轴的坐标值为准确的 y 轴坐标 Ey 减去 15。
- 当单击点坐标与闪烁棋子的坐标间范围: 在 x 轴方向为[100, 130], y 轴方向为[100, 130]时, 表示象棋可以实现向右下移动。如果闪烁棋子的索引 man 小于 10, 并且单击点的准确的 y 轴坐标 Ey 小于 270, 同时闪烁棋子与单击点之间没有其他棋子时, 则可以实现黑方象棋向右上移动。如果闪烁棋子的索引 man 大于 9, 并且单击点的准确的 y 轴坐标 Ey 大于 230, 同时闪烁棋子与单击点之间没有其他棋子时, 则可以实现红方象棋向右上移动。在具体向右上移动时, 首先需要把象棋的信息存储到集合 Var 中, 然后再设置象棋的 x 轴坐标为准确的 x 轴坐标 Ex 减去 20, y 轴的坐标值为准确的 y 轴坐标 Ey 减去 15。

通过象移动规则, 可以设计出象吃棋的方法 elephantRule(JLabel play, JLabel playTake, JLabel playQ[]), 其中参数 play 表示正在闪烁的棋子对象, playTake 为要吃掉的对象, 参数 playQ[]表示其他棋子对象。该方法的具体内容如代码 30.13 所示。

代码 30.13 实现象吃棋的方法: elephantRule()

```
public void elephantRule(JLabel play, JLabel playTake, JLabel playQ[]) {
    //创建成员变量
    int Move = 0;                                //闪烁棋子与被吃棋子间有棋子的个数
    boolean Chess = false;                        //是否可以吃棋的变量
    //实现吃左上方的棋子功能
    if (play.getX() - playTake.getX() >= 80
        && play.getX() - playTake.getX() <= 100
        && play.getY() - playTake.getY() >= 80
        && play.getY() - playTake.getY() <= 110) {
        //判断左上方是否有棋子
        for (int i = 0; i < 32; i++) {
            if (playQ[i].isVisible()
                && play.getX() - playQ[i].getX() >= 10
```



```

        && play.getX() - playQ[i].getX() <= 50
        && play.getY() - playQ[i].getY() >= 40
        && play.getY() - playQ[i].getY() <= 60) {
            Move++; //更新变量 Move 的值
            break;
        }
    }
    Chess = true; //设置变量的值
//实现吃右上方的棋子功能
else if (playTake.getX() - play.getX() >= 110
        && playTake.getX() - play.getX() <= 80
        && play.getY() - playTake.getY() >= 80
        && play.getY() - playTake.getY() <= 110) {
//判断右上方是否有棋子
for (int i = 0; i < 32; i++) {
    if (playQ[i].isVisible())
        && playQ[i].getX() - play.getX() >= 40
        && playQ[i].getX() - play.getX() <= 90
        && play.getY() - playQ[i].getY() >= 40
        && play.getY() - playQ[i].getY() <= 60) {
            Move++; //更新变量的值
            break;
        }
    }
    Chess = true; //设置变量的值
}
//实现吃下左方的棋子的功能
else if (play.getX() - playTake.getX() >= 80
        && play.getX() - playTake.getX() <= 110
        && playTake.getY() - play.getY() >= 80
        && playTake.getY() - play.getY() <= 110) {
//判断下左方是否有棋子
for (int i = 0; i < 32; i++) {
    if (playQ[i].isVisible())
        && play.getX() - playQ[i].getX() >= 10
        && play.getX() - playQ[i].getX() <= 60
        && play.getY() - playQ[i].getY() <= -40
        && play.getY() - playQ[i].getY() >= -60) {
            Move++; //更新变量 Move 的值
            break;
        }
    }
    Chess = true; //修改变量 Chess 的值
}
//实现吃下右方的棋子的功能
else if (playTake.getX() - play.getX() >= 80
        && playTake.getX() - play.getX() <= 110
        && playTake.getY() - play.getY() >= 80
        && playTake.getY() - play.getY() <= 110) {
//判断下右方是否有棋子
for (int i = 0; i < 32; i++) {
    if (playQ[i].isVisible())
        && playQ[i].getX() - play.getX() >= 50
        && playQ[i].getX() - play.getX() <= 80
        && playQ[i].getY() - play.getY() >= 40
        && playQ[i].getY() - play.getY() <= 60) {
            Move = 1; //设置变量 Move 的值
            break;
        }
    }
}

```

```

    }
    Chess = true;                                //设置变量的值
}
//实现吃棋子的具体功能
if (Chess && Move == 0
    && playTake.getName().charAt(1) != play.getName().charAt(1)) {
    //当不属于同一阵营时
    //把闪烁棋子的信息添加到集合中（用于悔棋）
    Var.add(String.valueOf(play.isVisible()));
    Var.add(String.valueOf(play.getX()));
    Var.add(String.valueOf(play.getY()));
    Var.add(String.valueOf(Move));
    //把被吃棋子的信息添加到集合中（用于悔棋）
    Var.add(String.valueOf(playTake.isVisible()));
    Var.add(String.valueOf(playTake.getX()));
    Var.add(String.valueOf(playTake.getY()));
    Var.add(String.valueOf(i));
    playTake.setVisible(false);                    //不显示被吃的棋子
    //重新设置闪烁棋子的位置
    play.setBounds(playTake.getX(), playTake.getY(), 40, 40);
}
}

```

【代码解析】

- 当被吃棋子坐标与闪烁棋子 x 坐标间的范围为[80, 110], 闪烁棋子坐标与被吃的棋子 y 坐标间的范围为[80, 110]时, 同时在一定范围内无其他棋子, 则表示象棋可以实现向右上吃。在具体实现吃棋的过程时, 首先需要判断可以吃棋的变量 `chess` 是否为 `true` 同时象棋和被吃棋不属于同一阵营, 然后才可以吃棋, 即先把象棋和被吃棋的信息存储到集合 `Var` 中, 然后设置被吃的棋不显示, 最后把象棋的坐标修改成被吃的棋的坐标。
- 当闪烁棋子的坐标与被吃棋子坐标间的范围: 在 x 轴方向为[80, 110], y 轴方向为[80, 110]时, 同时在一定范围内无其他棋子, 则表示象棋可以实现向左上吃。在具体实现吃棋过程时, 首先需要判断可以吃棋的变量 `chess` 是否为 `true`, 同时象棋和被吃棋不属于同一阵营, 然后才可以吃棋, 即先把象棋和被吃棋的信息存储到集合 `Var` 中, 然后设置被吃的棋不显示, 最后把象棋的坐标修改成被吃的棋的坐标。
- 当闪烁棋子的坐标与被吃棋子 x 坐标间的范围为[80, 110], 被吃棋子坐标与闪烁棋子 y 坐标间的范围为[80, 110]时, 同时在一定范围内无其他棋子, 则表示象棋可以实现向左下吃。在具体实现吃棋过程时, 首先需要判断可以吃棋的变量 `chess` 是否为 `true`, 同时象棋和被吃的棋不属于同一阵营, 然后才可以吃棋, 即先把象棋和被吃的棋的信息存储到集合 `Var` 中, 然后设置被吃的棋不显示, 最后把象棋的坐标修改成被吃的棋的坐标。
- 当被吃的棋子坐标与闪烁棋子坐标间的范围: 在 x 轴方向为[80, 110], y 轴方向为[80, 110]时, 同时在一定范围内无其他棋子, 则表示象棋可以实现向下移动。在具体实现吃的棋过程时, 首先需要判断可以吃棋的变量 `chess` 是否为 `true`, 同时象棋和被吃的棋不属于同一阵营, 然后才可以吃棋, 即先把象棋和被吃棋的信息存储到集合 `Var` 中, 然后设置被吃的棋不显示, 最后把象棋的坐标修改成被吃的棋的坐标。

30.3.5 实现士移动和吃的方法

根据游戏规则, 士的移动范围只能为王宫内部的位置; 移动规则每步只可以沿对角线方向移动一点, 即只能实现向左上、左下、右上和右下方向的移动。

在具体实现上述规则时, 专门设计了 `chapRule(int Man, JLabel play, JLabel playQ[], MouseEvent me)` 方法来实现士移动的规则, 其中参数 `Man` 表示正在闪烁棋子的索引, 参数 `play` 表示闪烁的棋子对象, `playQ[]` 表示其他棋子对象, `me` 为单击事件对象。该方法的具体内容如代码 30.14 所示。

代码 30.14 士移动的方法: `chapRule()`

```
public void chapRule(int Man, JLabel play, JLabel playQ[], MouseEvent me) {
    //实现向(上、右)移动的方法
    if (me.getX() - play.getX() >= 50 && me.getX() - play.getX() <= 80
        && play.getY() - me.getY() >= 15
        && play.getY() - me.getY() <= 50) {
        //判断黑方阵营士棋所移动的坐标是否符合规则
        if (Man < 14 && me.getX() > 150 && me.getX() < 300
            && me.getY() < 150) {
            //添加闪烁棋子的信息到集合 Var 中
            Var.add(String.valueOf(play.isVisible()));
            Var.add(String.valueOf(play.getX()));
            Var.add(String.valueOf(play.getY()));
            Var.add(String.valueOf(Man));
            //重新设置闪烁棋子的位置
            play.setBounds(play.getX() + 50, play.getY() - 50, 40, 40);
        }
        //判断红方阵营士棋所移动的坐标是否符合规则
        else if (Man > 13 && me.getY() > 340 && me.getX() > 150
            && me.getX() < 300) {
            //添加闪烁棋子的信息到集合 Var 中
            Var.add(String.valueOf(play.isVisible()));
            Var.add(String.valueOf(play.getX()));
            Var.add(String.valueOf(play.getY()));
            Var.add(String.valueOf(Man));
            //重新设置闪烁棋子的位置
            play.setBounds(play.getX() + 50, play.getY() - 50, 40, 40);
        }
    }
    //实现向(上、左)移动的方法
    else if (play.getX() - me.getX() <= 50
        && play.getX() - me.getX() >= 13
        && play.getY() - me.getY() >= 15
        && play.getY() - me.getY() <= 50) {
        //判断黑方阵营士棋所移动的坐标是否符合规则
        if (Man < 14 && me.getX() > 150 && me.getX() < 300
            && me.getY() < 150) {
            //添加闪烁棋子的信息到集合 Var 中
            Var.add(String.valueOf(play.isVisible()));
            Var.add(String.valueOf(play.getX()));
            Var.add(String.valueOf(play.getY()));
            Var.add(String.valueOf(Man));
        }
    }
}
```

```

        //重新设置闪烁棋子的位置
        play.setBounds(play.getX() - 50, play.getY() - 50, 40, 40);
    }
    //判断黑方阵营士棋所移动的坐标是否符合规则
    else if (Man > 13 && me.getY() > 340 && me.getX() > 150
        && me.getX() < 300) {
        //添加闪烁棋子的信息到集合 Var 中
        Var.add(String.valueOf(play.isVisible()));
        Var.add(String.valueOf(play.getX()));
        Var.add(String.valueOf(play.getY()));
        Var.add(String.valueOf(Man));
        //重新设置闪烁棋子的位置
        play.setBounds(play.getX() - 50, play.getY() - 50, 40, 40);
    }
}
//实现向(下、左)移动的方法
else if (play.getX() - me.getX() <= 50
    && play.getX() - me.getX() >= 15
    && me.getY() - play.getY() >= 50
    && me.getY() - play.getY() <= 80) {
    //判断黑方阵营士棋所移动的坐标是否符合规则
    if (Man < 14 && me.getX() > 150 && me.getX() < 300
        && me.getY() < 150) {
        //添加闪烁棋子的信息到集合 Var 中
        Var.add(String.valueOf(play.isVisible()));
        Var.add(String.valueOf(play.getX()));
        Var.add(String.valueOf(play.getY()));
        Var.add(String.valueOf(Man));
        //重新设置闪烁棋子的位置
        play.setBounds(play.getX() - 50, play.getY() + 50, 40, 40);
    }
    //判断黑方阵营士棋所移动的坐标是否符合规则
    else if (Man > 13 && me.getY() > 340 && me.getX() > 150
        && me.getX() < 300) {
        //添加闪烁棋子的信息到集合 Var 中
        Var.add(String.valueOf(play.isVisible()));
        Var.add(String.valueOf(play.getX()));
        Var.add(String.valueOf(play.getY()));
        Var.add(String.valueOf(Man));
        //重新设置闪烁棋子的位置
        play.setBounds(play.getX() - 50, play.getY() + 50, 40, 40);
    }
}
//实现向(下、右)移动的方法
else if (me.getX() - play.getX() >= 50
    && me.getX() - play.getX() <= 80
    && me.getY() - play.getY() >= 50
    && me.getY() - play.getY() <= 80) {
    //判断黑方阵营士棋所移动的坐标是否符合规则
    if (Man < 14 && me.getX() > 150 && me.getX() < 300
        && me.getY() < 150) {
        //添加闪烁棋子的信息到集合 Var 中
        Var.add(String.valueOf(play.isVisible()));
        Var.add(String.valueOf(play.getX()));
        Var.add(String.valueOf(play.getY()));
        Var.add(String.valueOf(Man));
        //重新设置闪烁棋子的位置
        play.setBounds(play.getX() + 50, play.getY() + 50, 40, 40);
    }
}

```

```

//判断黑方阵营士棋所移动的坐标是否符合规则
else if (Man > 13 && me.getY() > 340 && me.getX() > 150
        && me.getX() < 300) {
    //添加闪烁棋子的信息到集合 Var 中
    Var.add(String.valueOf(play.isVisible()));
    Var.add(String.valueOf(play.getX()));
    Var.add(String.valueOf(play.getY()));
    Var.add(String.valueOf(Man));
    //重新设置闪烁棋子的位置
    play.setBounds(play.getX() + 50, play.getY() + 50, 40, 40);
}
}
}

```

【代码解析】

- 当单击点坐标与闪烁棋子 x 坐标间的范围为 $[-5, 80]$, 闪烁棋子坐标与单击点 y 坐标间的范围为 $[15, 50]$ 时, 表示士棋可以实现向右上移动。如果闪烁棋子的索引 man 小于 14, 并且单击点的 x 轴坐标范围在 $(150, 300)$ 间, y 轴坐标小于 150, 则可以实现黑方士棋向右上移动。如果闪烁棋子的索引 man 大于 13, 并且单击点的 x 轴坐标范围在 $(150, 300)$ 间, y 轴坐标大于 340, 则可以实现红方士棋向右上移动。在具体向右上移动时, 首先要把士棋的信息存储到集合 Var 中, 然后再把士棋的 x 轴坐标增加 50, y 轴的坐标值减去 50。
- 当闪烁棋子的坐标与单击点坐标间的范围: 在 x 轴方向为 $[13, 50]$, y 轴方向为 $[15, 50]$ 时, 表示士棋可以实现向左上移动。如果闪烁棋子的索引 man 小于 14, 并且单击点的 x 轴坐标范围在 $(150, 300)$ 间, y 轴坐标小于 150, 则可以实现黑方士棋向左上移动。如果闪烁棋子的索引 man 大于 13, 并且单击点的 x 轴坐标范围在 $(150, 300)$ 间, y 轴坐标大于 340, 则可以实现红方士棋向左上移动。在具体向左上移动时, 首先要把士棋的信息存储到集合 Var 中, 然后再把士棋的 x 轴坐标减去 50, y 轴的坐标值减去 50。
- 当闪烁棋子的坐标与单击点 x 坐标间的范围为 $[15, 50]$, 单击点坐标与闪烁棋子 y 坐标间的范围为 $[50, 80]$ 时, 表示士棋可以实现向左下移动。如果闪烁棋子的索引 man 小于 14, 并且单击点的 x 轴坐标范围在 $(150, 300)$ 间, y 轴坐标小于 150, 则可以实现黑方士棋向左下移动。如果闪烁棋子的索引 man 大于 13, 并且单击点的 x 轴坐标范围在 $(150, 300)$ 间, y 轴坐标大于 340, 则可以实现红方士棋向左下移动。在具体向左下移动时, 首先要把士棋的信息存储到集合 Var 中, 然后再把士棋的 x 轴坐标增加 50, y 轴的坐标值减去 50。
- 当单击点坐标与闪烁棋子的坐标间范围: 在 x 轴方向为 $[50, 80]$, y 轴方向为 $[50, 80]$ 时, 表示士棋可以实现向右下移动。如果闪烁棋子的索引 man 小于 14, 并且单击点的 x 轴坐标范围在 $(150, 300)$ 间, y 轴坐标小于 150, 则可以实现黑方士棋向右下移动。如果闪烁棋子的索引 man 大于 13, 并且单击点的 x 轴坐标范围在 $(150, 300)$ 间, y 轴坐标大于 340, 则可以实现红方士棋向右下移动。在具体向右下移动时, 首先要把士棋的信息存储到集合 Var 中, 然后再把士棋的 x 轴坐标增加 50, y 轴的坐标值增加 50。

通过士移动规则, 可以设计出士吃棋的方法 `chapRule(int Man, JLabel play, JLabel playTake, JLabel playQ[])`, 其中参数 Man 表示闪烁棋子的索引, 参数 $play$ 表示正在闪烁的

棋子对象, playTake 为要吃掉的对象, 参数 playQ[] 表示所有的其他棋子对象。该方法的具体内容如代码 30.15 所示。

代码 30.15 士吃棋的方法: chapRule()

```
public void chapRule(int Man, JLabel play, JLabel playTake,
    JLabel playQ[]) {
    boolean Chap = false; //表示是否可以吃棋的变量
    //实现(上、右)方向吃棋的功能
    if (playTake.getX() - play.getX() >= 30
        && playTake.getX() - play.getX() <= 60
        && play.getY() - playTake.getY() >= 30
        && play.getY() - playTake.getY() <= 60) {
        //当黑方阵营的士吃棋时
        if (Man < 14 && playTake.getX() > 150 && playTake.getX() < 300
            && playTake.getY() < 150 && playTake.isVisible()) {
            Chap = true; //设置变量 Chap 的值
        }
        //当红方阵营的士吃棋时
        else if (Man > 13 && playTake.getX() > 150
            && playTake.getX() < 300 && playTake.getY() > 340
            && playTake.isVisible()) {
            Chap = true; //设置变量 Chap 的值
        }
    }
    //实现(上、左)方向吃棋的功能
    else if (play.getX() - playTake.getX() <= 60
        && play.getX() - playTake.getX() >= 30
        && play.getY() - playTake.getY() >= 30
        && play.getY() - playTake.getY() <= 60) {
        //当黑方阵营的士吃棋时
        if (Man < 14 && playTake.getX() > 150 && playTake.getX() < 300
            && playTake.getY() < 150 && playTake.isVisible()) {
            Chap = true; //设置变量 Chap 的值
        }
        //当红方阵营的士吃棋时
        else if (Man > 13 && playTake.getX() > 150
            && playTake.getX() < 300 && playTake.getY() > 340
            && playTake.isVisible()) {
            Chap = true; //设置变量 Chap 的值
        }
    }
    //实现(下、左)方向吃棋的功能
    else if (play.getX() - playTake.getX() <= 60
        && play.getX() - playTake.getX() >= 30
        && playTake.getY() - play.getY() >= 30
        && playTake.getY() - play.getY() <= 60) {
        //当黑方阵营的士吃棋时
        if (Man < 14 && playTake.getX() > 150 && playTake.getX() < 300
            && playTake.getY() < 150 && playTake.isVisible()) {
            Chap = true; //设置变量 Chap 的值
        }
        //当红方阵营的士吃棋时
        else if (Man > 13 && playTake.getX() > 150
            && playTake.getX() < 300 && playTake.getY() > 340
            && playTake.isVisible()) {
            Chap = true; //设置变量 Chap 的值
        }
    }
}
```



```

    }
}
//实现(下、右)方向吃棋的功能
else if (playTake.getX() - play.getX() >= 30
        && playTake.getX() - play.getX() <= 60
        && playTake.getY() - play.getY() >= 30
        && playTake.getY() - play.getY() <= 60) {
    //当黑方阵营的士吃棋时
    if (Man < 14 && playTake.getX() > 150 && playTake.getX() < 300
        && playTake.getY() < 150 && playTake.isVisible()) {
        Chap = true; //设置变量 Chap 的值
    }
    //当红方阵营的士吃棋时
    else if (Man > 13 && playTake.getX() > 150
        && playTake.getX() < 300 && playTake.getY() > 340
        && playTake.isVisible()) {
        Chap = true; //设置变量 Chap 的值
    }
}
//具体实现吃棋功能
if (Chap
    && playTake.getName().charAt(1) != play.getName().charAt(1)) {
    //当闪烁棋子和被吃棋子不属于同一阵营时
    //添加闪烁棋子信息到集合 Var 中
    Var.add(String.valueOf(play.isVisible()));
    Var.add(String.valueOf(play.getX()));
    Var.add(String.valueOf(play.getY()));
    Var.add(String.valueOf(Man));
    //添加被吃棋子信息到集合 Var 中
    Var.add(String.valueOf(playTake.isVisible()));
    Var.add(String.valueOf(playTake.getX()));
    Var.add(String.valueOf(playTake.getY()));
    Var.add(String.valueOf(i));
    playTake.setVisible(false); //不显示被吃的棋子
    //设置闪烁棋子坐标为被吃的棋子坐标
    play.setBounds(playTake.getX(), playTake.getY(), 40, 40);
}
}

```

【代码解析】

- 当被吃棋子坐标与闪烁棋子 x 坐标间的范围为 $[30, 60]$, 闪烁棋子坐标与被吃的棋子 y 坐标间的范围为 $[30, 60]$ 时, 表示士棋可以实现向右上吃。如果闪烁棋子的索引 man 小于 14, 并且被吃棋子的 x 轴坐标范围在 $(150, 300)$ 间, y 轴坐标小于 150, 则可以实现黑方士棋向右上吃。如果闪烁棋子的索引 man 大于 13, 并且单击点的 x 轴坐标范围在 $(150, 300)$ 间, y 轴坐标大于 340, 则可以实现红方士棋向右上吃。在具体向上移动前, 首先需要判断可以吃棋的变量 $Chap$ 是否为 `true`, 同时士棋和被吃的棋不属于同一阵营, 然后才可以吃棋, 即先把士棋和被吃的棋的信息存储到集合 Var 中, 然后设置被吃的棋不显示, 最后把士棋的坐标修改成被吃的棋的坐标。
- 当闪烁棋子的坐标与被吃棋子坐标间的范围: 在 x 轴方向为 $[30, 60]$, y 轴方向为 $[30, 60]$ 时, 表示士棋可以实现向左上吃。如果闪烁棋子的索引 man 小于 14, 并且被吃棋子的 x 轴坐标范围在 $(150, 300)$ 间, y 轴坐标小于 150, 则可以实现黑方士棋向左上吃。如果闪烁棋子的索引 man 大于 13, 并且单击点的 x 轴坐标范围在 $(150,$

300) 间, y 轴坐标大于 340, 则可以实现红方士棋向左上吃。在具体向左移动前, 首先需要判断可以吃棋的变量 Chap 是否为 true, 同时士棋和被吃的棋不属于同一阵营, 然后才可以吃棋, 即先把士棋和被吃的棋的信息存储到集合 Var 中, 然后设置被吃的棋不显示, 最后把士棋的坐标修改成被吃的棋的坐标。

- 当闪烁棋子的坐标与被吃棋子 x 坐标间的范围为[30, 60], 被吃棋子的坐标与闪烁棋子 y 坐标间的范围为[30, 60]时, 表示士棋可以实现向左下吃。如果闪烁棋子的索引 man 小于 14, 并且被吃棋子的 x 轴坐标范围在 (150, 300) 间, y 轴坐标小于 150, 则可以实现黑方士棋向左下吃。如果闪烁棋子的索引 man 大于 13, 并且单击点的 x 轴坐标范围在 (150, 300) 间, y 轴坐标大于 340, 则可以实现红方士棋向右吃。在具体向左下移动前, 首先需要判断可以吃棋的变量 Chap 是否为 true, 同时士棋和被吃的棋不属于同一阵营, 然后才可以吃棋, 即先把士棋和被吃的棋的信息存储到集合 Var 中, 然后设置被吃的棋不显示, 最后把士棋的坐标修改成被吃的棋的坐标。
- 当被吃棋子坐标与闪烁棋子的坐标间范围: 在 x 轴方向为[30, 60], y 轴方向为[30, 60]时, 表示士棋可以实现向下移动。如果闪烁棋子的索引 man 小于 14, 并且被吃棋子的 x 轴坐标范围在 (150, 300) 间, y 轴坐标小于 150, 则可以实现黑方士棋向右下吃。如果闪烁棋子的索引 man 大于 13, 并且单击点的 x 轴坐标范围在 (150, 300) 间, y 轴坐标大于 340, 则可以实现红方士棋向下吃。在具体向右下移动前, 首先需要判断可以吃棋的变量 Chap 是否为 true, 同时士棋和被吃棋不属于同一阵营, 然后才可以吃棋, 即先把士棋和被吃的棋的信息存储到集合 Var 中, 然后设置被吃的棋不显示, 最后把士棋的坐标修改成被吃的棋的坐标。

30.3.6 实现将移动和吃的方法

根据游戏规则, 将的移动范围只能为王宫内部的位置; 而移动规则每步只可以水平或垂直移动一点。

在具体实现上述规则时, 专门设计了 willRule(int Man, JLabel play, JLabel playQ[], MouseEvent me)方法来实现将棋移动的规则, 其中参数 Man 表示正在闪烁的棋子的索引, 参数 play 表示闪烁的棋子对象, playQ[]表示其他棋子对象, me 为单击事件对象。该方法的具体内容如代码 30.16 所示。

代码 30.16 将移动的方法: willRule()

```
public void willRule(int Man, JLabel play, JLabel playQ[], MouseEvent me) {
    //实现向上移动功能
    if (me.getX() - play.getX() >= -5 && me.getX() - play.getX() <= 35
        && play.getY() - me.getY() <= 50
        && play.getY() - me.getY() >= 15) {
        //判断黑方阵营将棋所移动的坐标是否符合规则
        if (Man == 30 && me.getX() > 150 && me.getX() < 300
            && me.getY() < 150) {
            //添加闪烁棋子信息到集合中(用于悔棋)
            Var.add(String.valueOf(play.isVisible()));
            Var.add(String.valueOf(play.getX()));
            Var.add(String.valueOf(play.getY()));
        }
    }
}
```

```

        Var.add(String.valueOf(Man));
        //设置闪烁棋子的坐标为单击点的坐标
        play.setBounds(play.getX(), play.getY() - 50, 40, 40);
    }
    //判断红方阵营将棋所移动的坐标是否符合规则
    else if (Man == 31 && me.getY() > 340 && me.getX() > 150
        && me.getX() < 300) {
        //添加闪烁棋子信息到集合中 (用于悔棋)
        Var.add(String.valueOf(play.isVisible()));
        Var.add(String.valueOf(play.getX()));
        Var.add(String.valueOf(play.getY()));
        Var.add(String.valueOf(Man));
        //设置闪烁棋子的坐标为单击点的坐标
        play.setBounds(play.getX(), play.getY() - 50, 40, 40);
    }
}
//实现向左移动功能
else if (play.getX() - me.getX() >= 15
    && play.getX() - me.getX() <= 50
    && me.getY() - play.getY() <= 40
    && me.getY() - play.getY() >= -5) {
    //判断黑方阵营将棋所移动的坐标是否符合规则
    if (Man == 30 && me.getX() > 150 && me.getX() < 300
        && me.getY() < 150) {
        //添加闪烁棋子信息到集合中 (用于悔棋)
        Var.add(String.valueOf(play.isVisible()));
        Var.add(String.valueOf(play.getX()));
        Var.add(String.valueOf(play.getY()));
        Var.add(String.valueOf(Man));
        //设置闪烁棋子的坐标为单击点的坐标
        play.setBounds(play.getX() - 50, play.getY(), 40, 40);
    }
    //判断红方阵营将棋所移动的坐标是否符合规则
    else if (Man == 31 && me.getY() > 340 && me.getX() > 150
        && me.getX() < 300) {
        //添加闪烁棋子信息到集合中 (用于悔棋)
        Var.add(String.valueOf(play.isVisible()));
        Var.add(String.valueOf(play.getX()));
        Var.add(String.valueOf(play.getY()));
        Var.add(String.valueOf(Man));
        //设置闪烁棋子的坐标为单击点的坐标
        play.setBounds(play.getX() - 50, play.getY(), 40, 40);
    }
}
//实现向右移动的功能
else if (me.getX() - play.getX() >= 50
    && me.getX() - play.getX() <= 85
    && me.getY() - play.getY() <= 40
    && me.getY() - play.getY() >= -5) {
    //判断黑方阵营将棋所移动的坐标是否符合规则
    if (Man == 30 && me.getX() > 150 && me.getX() < 300
        && me.getY() < 150) {
        //添加闪烁棋子信息到集合中 (用于悔棋)
        Var.add(String.valueOf(play.isVisible()));
        Var.add(String.valueOf(play.getX()));
        Var.add(String.valueOf(play.getY()));
        Var.add(String.valueOf(Man));
        //设置闪烁棋子的坐标为单击点的坐标
        play.setBounds(play.getX() + 50, play.getY(), 40, 40);
    }
}

```

```

    }
    //判断红方阵营将棋所移动的坐标是否符合规则
    else if (Man == 31 && me.getY() > 340 && me.getX() > 150
            && me.getX() < 300) {
        //添加闪烁棋子信息到集合中（用于悔棋）
        Var.add(String.valueOf(play.isVisible()));
        Var.add(String.valueOf(play.getX()));
        Var.add(String.valueOf(play.getY()));
        Var.add(String.valueOf(Man));
        //设置闪烁棋子的坐标为单击点的坐标
        play.setBounds(play.getX() + 50, play.getY(), 40, 40);
    }
}
//实现向下移动的功能
else if (me.getX() - play.getX() >= -5
        && me.getX() - play.getX() <= 35
        && me.getY() - play.getY() <= 85
        && me.getY() - play.getY() >= 50) {
    //判断黑方阵营将棋所移动的坐标是否符合规则
    if (Man == 30 && me.getX() > 150 && me.getX() < 300
            && me.getY() < 150) {
        //添加闪烁棋子信息到集合中（用于悔棋）
        Var.add(String.valueOf(play.isVisible()));
        Var.add(String.valueOf(play.getX()));
        Var.add(String.valueOf(play.getY()));
        Var.add(String.valueOf(Man));
        //设置闪烁棋子的坐标为单击点的坐标
        play.setBounds(play.getX(), play.getY() + 50, 40, 40);
    }
    //判断红方阵营将棋所移动的坐标是否符合规则
    else if (Man == 31 && me.getY() > 340 && me.getX() > 150
            && me.getX() < 300) {
        //添加闪烁棋子信息到集合中（用于悔棋）
        Var.add(String.valueOf(play.isVisible()));
        Var.add(String.valueOf(play.getX()));
        Var.add(String.valueOf(play.getY()));
        Var.add(String.valueOf(Man));
        //设置闪烁棋子的坐标为单击点的坐标
        play.setBounds(play.getX(), play.getY() + 50, 40, 40);
    }
}
}
}

```

【代码解析】

- ❑ 当单击点坐标与闪烁棋子的坐标间范围：在 x 轴方向为 $(-5, 35)$ ， y 轴方向为 $(15, 50)$ 时，表示将棋可以实现向上移动。如果闪烁棋子的索引 man 为 30，并且单击点的 x 轴坐标范围在 $(150, 300)$ 间， y 轴坐标小于 150，则可以实现黑方将棋向上移动。如果闪烁棋子的索引 man 为 31，并且单击点的 x 轴坐标范围在 $(150, 300)$ 间， y 轴坐标大于 340，则可以实现红方将棋向上移动。在具体向上移动时，首先需要把将棋的信息存储到集合 Var 中，然后再把将棋 y 轴坐标值减去 50。
- ❑ 当单击点坐标与闪烁棋子的坐标间范围：在 x 轴方向为 $(15, 50)$ ， y 轴方向为 $(-5, 40)$ 时，表示将棋可以实现向左移动。如果闪烁棋子的索引 man 为 30，并且单击点的 x 轴坐标范围在 $(150, 300)$ 间， y 轴坐标小于 150，则可以实现黑方将棋向左移动。

如果闪烁棋子的索引 `man` 为 31, 并且单击点的 x 轴坐标范围在(150, 300)间, y 轴坐标大于 340, 则可以实现红方将棋向左移动。在具体向左移动时, 首先需要把将棋的信息存储到集合 `Var` 中, 然后再把将棋的 x 轴坐标值减去 50。

- 当单击点坐标与闪烁棋子的坐标间范围: 在 x 轴方向为(50, 85), y 轴方向为(-5, 40) 时, 表示将棋可以实现向右移动。如果闪烁棋子的索引 `man` 为 30, 并且单击点的 x 轴坐标范围在(150, 300)间, y 轴坐标小于 150, 则可以实现黑方将棋向右移动。如果闪烁棋子的索引 `man` 为 31, 并且单击点的 x 轴坐标范围在(150, 300)间, y 轴坐标大于 340, 则可以实现红方将棋向右移动。在具体向右移动时, 首先需要把将棋的信息存储到集合 `Var` 中, 然后再把将棋的 x 轴坐标值加上 50。
- 当单击点坐标与闪烁棋子的坐标间范围: 在 x 轴方向为(-5, 35), y 轴方向为(50, 85) 时, 表示将棋可以实现向下移动。如果闪烁棋子的索引 `man` 为 30, 并且单击点的 x 轴坐标范围在(150, 300)间, y 轴坐标小于 150, 则可以实现黑方将棋向下移动。如果闪烁棋子的索引 `man` 为 31, 并且单击点的 x 轴坐标范围在(150, 300)间, y 轴坐标大于 340, 则可以实现红方将棋向下移动。在具体向左移动时, 首先需要把将棋的信息存储到集合 `Var` 中, 然后再把将棋的 y 轴坐标值加上 50。

通过将移动规则, 可以设计出将吃棋的方法 `willRule(int Man, JLabel play, JLabel playTake, JLabel playQ[])`, 其中参数 `Man` 表示闪烁棋子的索引, 参数 `play` 表示正在闪烁的棋子对象, `playTake` 为要吃掉的对象, 参数 `playQ[]` 表示其他棋子对象。该方法的具体内容如代码 30.17 所示。

代码 30.17 将吃棋的方法: `willRule()`

```
public void willRule(int Man, JLabel play, JLabel playTake,
    JLabel playQ[]) {
    boolean will = false;           //是否能够实现吃的变量
    //实现向上吃棋功能
    if (play.getX() - playTake.getX() >= -10
        && play.getX() - playTake.getX() <= 10
        && play.getY() - playTake.getY() >= 40
        && play.getY() - playTake.getY() <= 60
        && playTake.isVisible()) {
        //当被吃的棋子与黑方阵营中的将棋相近时
        if (Man == 30 && playTake.getX() > 150 && playTake.getX() < 300
            && playTake.getY() < 150) {
            will = true;           //设置变量 will 的值
        }
        //当被吃的棋子与黑方阵营中的将棋相近时
        else if (Man == 31 && playTake.getY() > 340
            && playTake.getX() > 150 && playTake.getX() < 300) {
            will = true;           //设置变量 will 的值
        }
    }
    //实现向左吃棋功能
    else if (play.getX() - playTake.getX() >= 40
        && play.getX() - playTake.getX() <= 60
        && playTake.getY() - play.getY() <= 10
        && playTake.getY() - play.getY() >= -10
        && playTake.isVisible()) {
```



```

//当被吃的棋子与黑方阵营中的将棋相近时
if (Man == 30 && playTake.getX() > 150 && playTake.getX() < 300
    && playTake.getY() < 150) {
    will = true; //设置变量 will 的值
}
//当被吃的棋子与黑方阵营中的将棋相近时
else if (Man == 31 && playTake.getY() > 340
    && playTake.getX() > 150 && playTake.getX() < 300) {
    will = true; //设置变量 will 的值
}
}
//实现向右吃棋的功能
else if (playTake.getX() - play.getX() >= 40
    && playTake.getX() - play.getX() <= 60
    && playTake.getY() - play.getY() <= 10
    && playTake.getY() - play.getY() >= -10
    && playTake.isVisible()) {
//当被吃的棋子与黑方阵营中的将棋相近时
if (Man == 30 && playTake.getX() > 150 && playTake.getX() < 300
    && playTake.getY() < 150) {
    will = true; //设置变量 will 的值
}
//当被吃的棋子与黑方阵营中的将棋相近时
else if (Man == 31 && playTake.getY() > 340
    && playTake.getX() > 150 && playTake.getX() < 300) {
    will = true; //设置变量 will 的值
}
}
//实现向下吃棋的功能
else if (playTake.getX() - play.getX() >= -10
    && playTake.getX() - play.getX() <= 10
    && playTake.getY() - play.getY() <= 60
    && playTake.getY() - play.getY() >= 40
    && playTake.isVisible()) {
//当被吃的棋子与黑方阵营中的将棋相近时
if (Man == 30 && playTake.getX() > 150 && playTake.getX() < 300
    && playTake.getY() < 150) {
    will = true; //设置变量 will 的值
}
//当被吃的棋子与黑方阵营中的将棋相近时
else if (Man == 31 && playTake.getY() > 340
    && playTake.getX() > 150 && playTake.getX() < 300) {
    will = true; //设置变量 will 的值
}
}
//具体的吃棋功能
if (playTake.getName().charAt(1) != play.getName().charAt(1)
    && will) { //当不属于同一阵营时
//添加闪烁棋子信息到集合中（用于悔棋）
Var.add(String.valueOf(play.isVisible()));
Var.add(String.valueOf(play.getX()));
Var.add(String.valueOf(play.getY()));
Var.add(String.valueOf(Man));
//添加被吃棋子信息到集合中（用于悔棋）
Var.add(String.valueOf(playTake.isVisible()));
Var.add(String.valueOf(playTake.getX()));

```



```

        Var.add(String.valueOf(playTake.getY()));
        Var.add(String.valueOf(i));
        playTake.setVisible(false);           //使被吃棋子不显示
        //设置闪烁棋子的坐标为被吃的棋子的坐标
        play.setBounds(playTake.getX(), playTake.getY(), 40, 40);
    }
}

```

【代码解析】

- 当闪烁棋子的坐标与被吃棋子坐标间的范围：在 x 轴方向为 $(-10, 10)$ ， y 轴方向为 $(40, 60)$ 时，表示将棋可以实现向上吃。如果闪烁棋子的索引 `man` 为 30，并且被吃棋子的 x 轴坐标范围在 $(150, 300)$ 间， y 轴坐标小于 150，则可以实现黑方将棋向上吃。如果闪烁棋子的索引 `man` 为 31，并且单击点的 x 轴坐标范围在 $(150, 300)$ 间， y 轴坐标大于 340，则可以实现红方将棋向上吃。在具体向上移动前，首先需要判断可以吃棋的变量 `will` 是否为 `true`，同时将棋和被吃的棋不属于同一阵营，然后才可以吃棋，即先把将棋和被吃的棋的信息存储到集合 `Var` 中，然后设置被吃的棋不显示，最后把将棋的坐标修改成被吃的棋的坐标。
- 当闪烁棋子的坐标与被吃棋子的坐标间范围：在 x 轴方向为 $(40, 60)$ ， y 轴方向为 $(-10, 10)$ 时，表示将棋可以实现向左吃。如果闪烁棋子的索引 `man` 为 30 并且单击点的 x 轴坐标范围在 $(150, 300)$ 间， y 轴坐标小于 150，则可以实现黑方将棋向左吃。如果闪烁棋子的索引 `man` 为 31，并且单击点的 x 轴坐标范围在 $(150, 300)$ 间， y 轴坐标大于 340，则可以实现红方将棋向左吃。在具体向左移动前，首先需要判断可以吃棋的变量 `will` 是否为 `true`，同时将棋和被吃的棋不属于同一阵营，然后才可以吃棋，即先把将棋和被吃的棋的信息存储到集合 `Var` 中，然后设置被吃的棋不显示，最后把将棋的坐标修改成被吃的棋的坐标。
- 当被吃棋子坐标与闪烁棋子的坐标间范围：在 x 轴方向为 $(40, 60)$ ， y 轴方向为 $(-10, 10)$ 时，表示将棋可以实现向右吃。如果闪烁棋子的索引 `man` 为 30 并且单击点的 x 轴坐标范围在 $(150, 300)$ 间， y 轴坐标小于 150，则可以实现黑方将棋向右吃。如果闪烁棋子的索引 `man` 为 31 并且单击点的 x 轴坐标范围在 $(150, 300)$ 间， y 轴坐标大于 340，则可以实现红方将棋向右吃。在具体向右移动前，首先需要判断可以吃棋的变量 `will` 是否为 `true`，同时将棋和被吃的棋不属于同一阵营，然后才可以吃棋，即先把将棋和被吃的棋的信息存储到集合 `Var` 中，然后设置被吃的棋不显示，最后把将棋的坐标修改成被吃的棋的坐标。
- 当被吃棋子坐标与闪烁棋子的坐标间范围：在 x 轴方向为 $(-10, 10)$ ， y 轴方向为 $(40, 60)$ 时，表示将棋可以实现向下移动。如果闪烁棋子的索引 `man` 为 30 并且单击点的 x 轴坐标范围在 $(150, 300)$ 间， y 轴坐标小于 150，则可以实现黑方将棋向下吃。如果闪烁棋子的索引 `man` 为 31 并且单击点的 x 轴坐标范围在 $(150, 300)$ 间， y 轴坐标大于 340，则可以实现红方将棋向下吃。在具体向下移动前，首先需要判断可以吃棋的变量 `will` 是否为 `true`，同时将棋和被吃的棋不属于同一阵营，然后才可以吃棋，即先把将棋和被吃的棋的信息存储到集合 `Var` 中，然后设置被吃的棋不显示，最后把将棋的坐标修改成被吃的棋的坐标。

30.4 小 结

本章主要介绍一个完整的中国象棋游戏项目，该项目模拟了现实生活中的象棋游戏，基于 Java 语言构建而成。在具体实现中国象棋游戏项目时，不仅详细讲解了该系统涉及的象棋对象和该对象的一些特殊效果，而且还详细讲解了各种棋子的规则，如卒的移动和吃的规则、炮的移动和吃的规则、车的移动和吃的规则、马的移动和吃的规则、象的移动和吃的规则、士的移动和吃的规则及将的移动和吃的规则等。

第 31 章 俄罗斯方块游戏网络版（Swing+多线程+网络编程）

本章将综合 J2SE 的基础知识来实现一个名为俄罗斯方块的游戏，该游戏包含所有的基础知识有面向对象思想、多线程、GUI 中的事件模型、网络编程等。为了让读者能够掌握和编写出该游戏，本章将通过 MVC 三层结构的方式来讲解该项目。

本章的学习目标如下：

- 掌握 J2SE 中基础知识和面向对象的思想；
- 熟悉网络编程；
- 了解俄罗斯方块游戏的总体设计思路及逐步实现过程；
- 培养把实际问题转换成代码的能力。

31.1 俄罗斯方块游戏项目原理

俄罗斯方块游戏是一款风靡全球的电视游戏机和掌上游戏机游戏，它由俄罗斯人阿列克谢·帕基特诺夫发明，故得此名。该游戏虽然只有几种动作（例如移动、旋转、摆放和自动输出的各种方块，使之排列成完整的一行或多行并且消除得分），但是其上手简单、老少皆宜，因此家喻户晓，风靡世界。本节不仅讲解该游戏的原理，还将演示如何玩该游戏。

31.1.1 基本原理


虽然俄罗斯方块游戏项目很简单，但是如果编写程序则不是一件简单的事情。在具体设计和实现前，需要了解该游戏的基本原理。俄罗斯方块游戏项目的原理如下。

（1）游戏界面：一个用于摆放小型正方形的平面虚拟场地，其标准大小：行宽为 15 单位，列高为 20 单位，每个单位为一个正方形。

（2）俄罗斯方块：一组由 4 个小型正方形组成的规则图形，英文称为 Tetromino，中文为方块。其共有 7 种类型，分别以 S、Z、L、J、I、O、T 这 7 个字母的形状来命名。而每种类型的方块最多只有 4 种状态。

（3）在具体玩时，游戏界面顶部会出现由随机发生器不断输出地单个方块，该方块会自动向下移动，在移动的过程中玩家可以通过键盘控制旋转（选择状态）、左右移动来摆放方块。

(4) 当摆放的结果将游戏界面的一行或多行完全填满时, 则组成这些行的所有小正方形将被消除, 并且以此来换取一定的积分或者其他形式的奖励。而未被消除的方块会一直累积, 并对后来的方块摆放造成各种影响。如果未被消除的方块堆放的高度超过场地所规定的最大高度(20 单位), 则游戏结束。

 **注意:** 具体到每一款不同的游戏, 其中的细节规则可能有差别, 但是基本规则是相同的。

31.1.2 项目结构框架分析

对于俄罗斯方块游戏项目, 主要利用 MVC 模式来实现。所谓 MVC 模式, 就是把整个项目的功能分成模型层、表示层和控制层。俄罗斯方块游戏项目目录如图 31.1 所示, 各个包的功能如下。

- ❑ `com.cjgong.client` 包: 该包里包含了俄罗斯方块游戏项目中, 实现客户端连接服务器端的程序。
- ❑ `com.cjgong.server` 包: 该包里包含了俄罗斯方块游戏项目中, 实现服务器端连接客户端的程序。
- ❑ `com.cjgong.tetris` 包: 该包里包含了俄罗斯方块游戏项目中, 实现各种业务逻辑的类。
- ❑ `com.cjgong.view` 包: 该包里包含了俄罗斯方块游戏项目中的各种视图, 如关于面板、连接对方面板、分数报告面板、设置级别面板、警告面板和对话框、游戏结束面板和对话框。

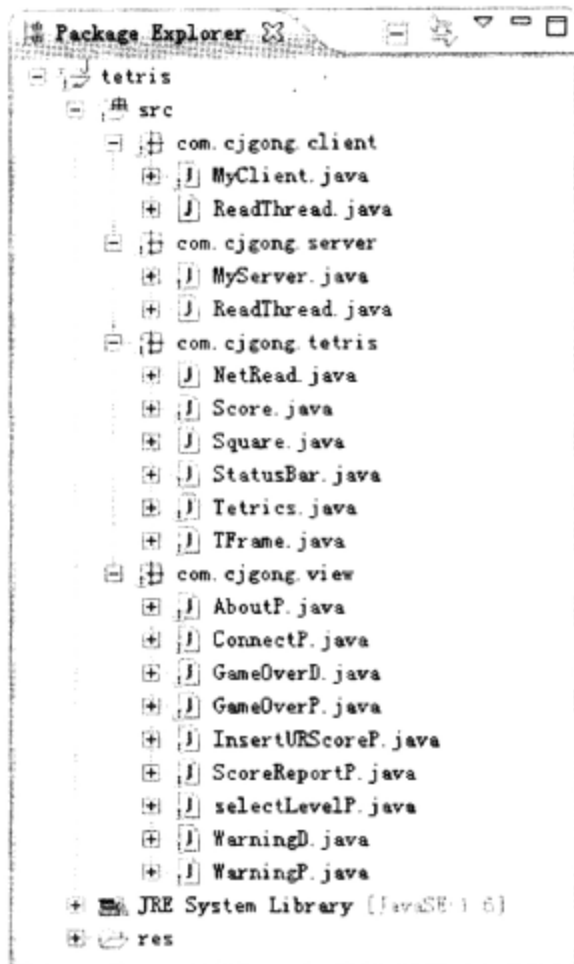


图 31.1 项目目录

31.1.3 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括俄罗斯方块的初始化界面、查看已存在的文件和目录属性及查看不存在的文件和目录属性。

1. 初始化界面

当运行俄罗斯方块游戏项目中的 `TFrame` 类后, 会出现如图 31.2 所示的初始界面——俄罗斯方块界面。

2. “游戏”菜单

当出现俄罗斯方块初始化界面后, 选择“游戏”|“开始游戏”菜单命令就可以开始一个新游戏, 具体过程如图 31.3 所示。

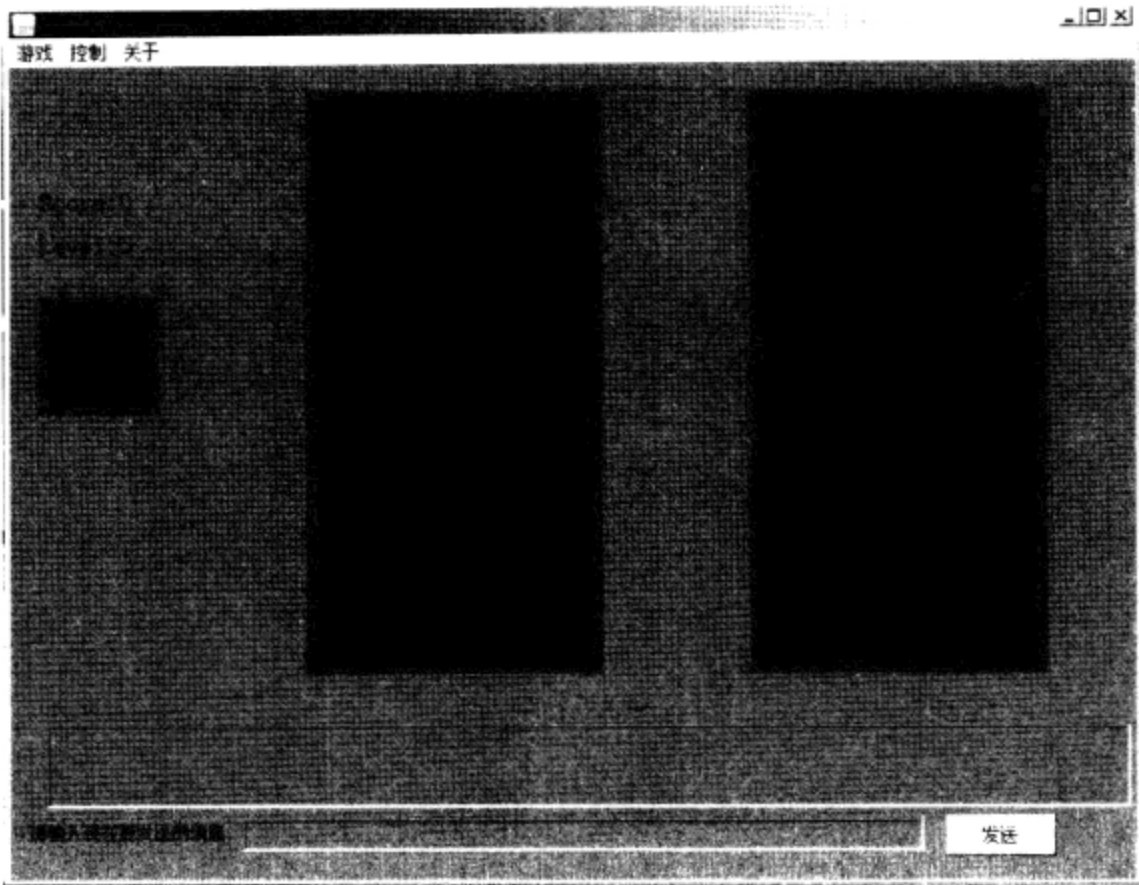


图 31.2 初始化界面

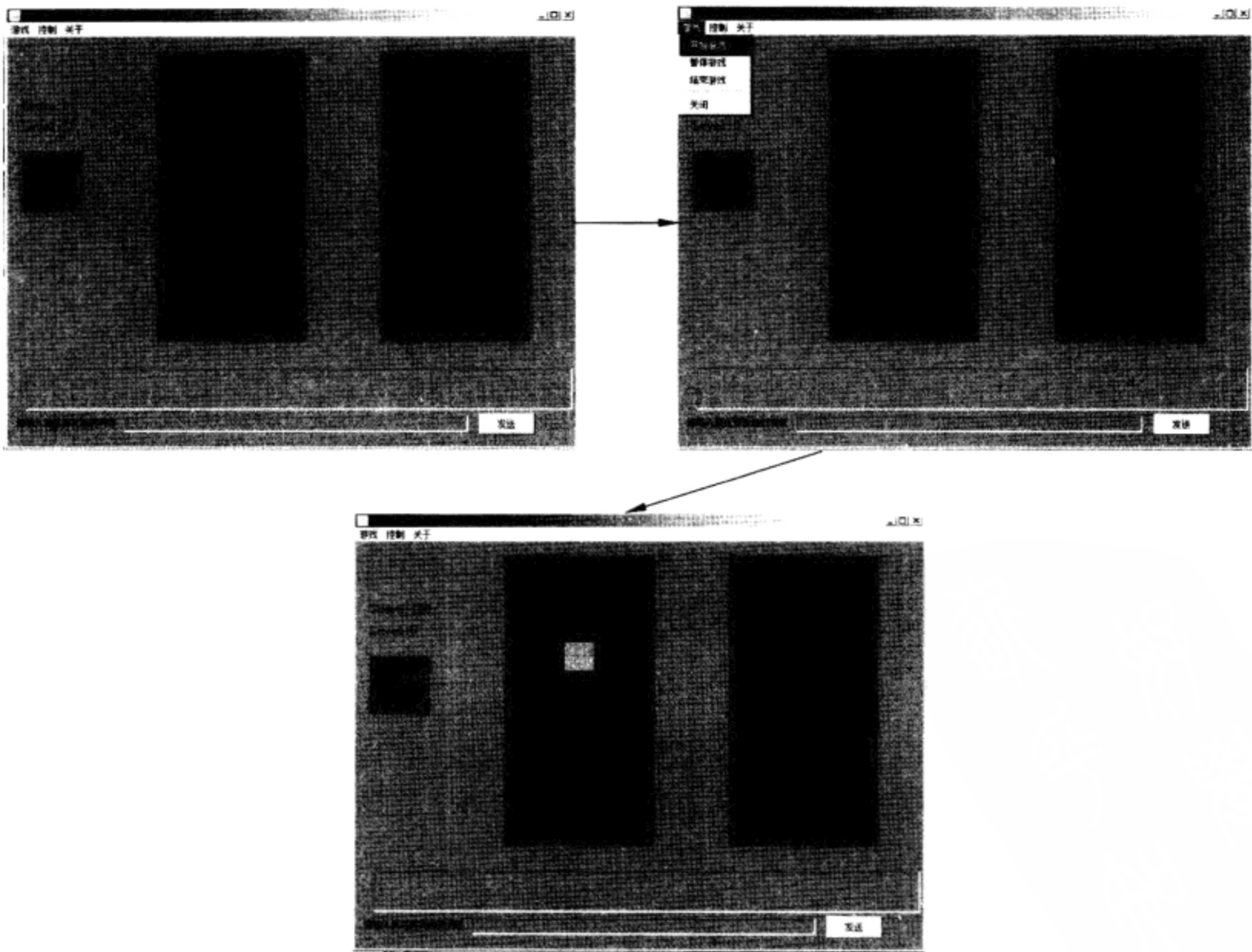


图 31.3 “开始游戏”菜单

在玩游戏的过程中，如果选择“游戏”|“暂停游戏”菜单命令后，游戏就会处于暂停状态。这时如果再次选择“游戏”|“开始游戏”菜单命令，游戏就会处于运行状态，具体过程如图 31.4 所示。

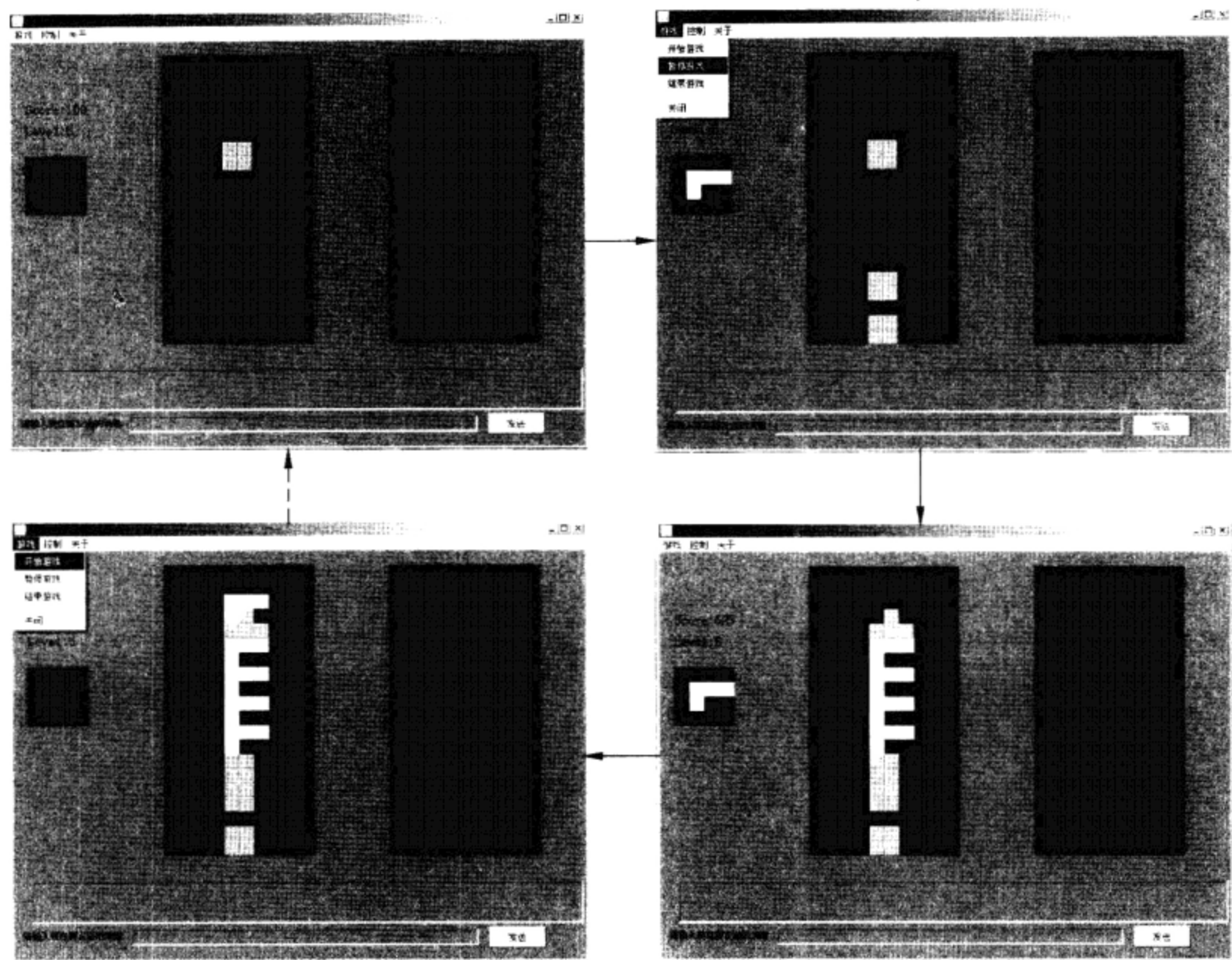


图 31.4 “暂停游戏”菜单

当游戏处于运行状态时，如果选择“游戏”|“结束游戏”菜单命令，游戏就结束，具体过程如图 31.5 所示。

当游戏处于运行状态时，如果选择“游戏”|“关闭”菜单命令或者单击窗口右上角的关闭按钮，这时游戏就会结束，具体过程如图 31.6 所示。

3. “控制”菜单

当出现初始化界面后，如果想设置俄罗斯方块游戏的难度级别，可以选择“控制”|“设置级别”菜单命令，这时就会出现“设置级别”对话框。在该对话框中通过滚动条选择相应的游戏难易级别，然后单击“确定”按钮，该游戏的左边就会显示出具体的游戏级别，具体过程如图 31.7 所示。

当出现初始化界面后，如果想设置运行的俄罗斯方块游戏为服务器端，可以选择“控制”|“等待对方连接”菜单命令，这时在界面下方的文本域中就会出现“Welcome to the server!”字符串，具体过程如图 31.8 所示。

当再次运行程序出现初始化界面, 如果想设置运行的俄罗斯方块游戏为客户端时, 可以选择“控制”|“连接对方”菜单命令, 会出现“连接对方”对话框。在该对话框中输入服务器端的 IP 地址或服务器名, 单击“确定”按钮, 在服务器端界面下方的文本域中会显示“Has been conected”字符串, 具体过程如图 31.9 和图 31.10 所示。

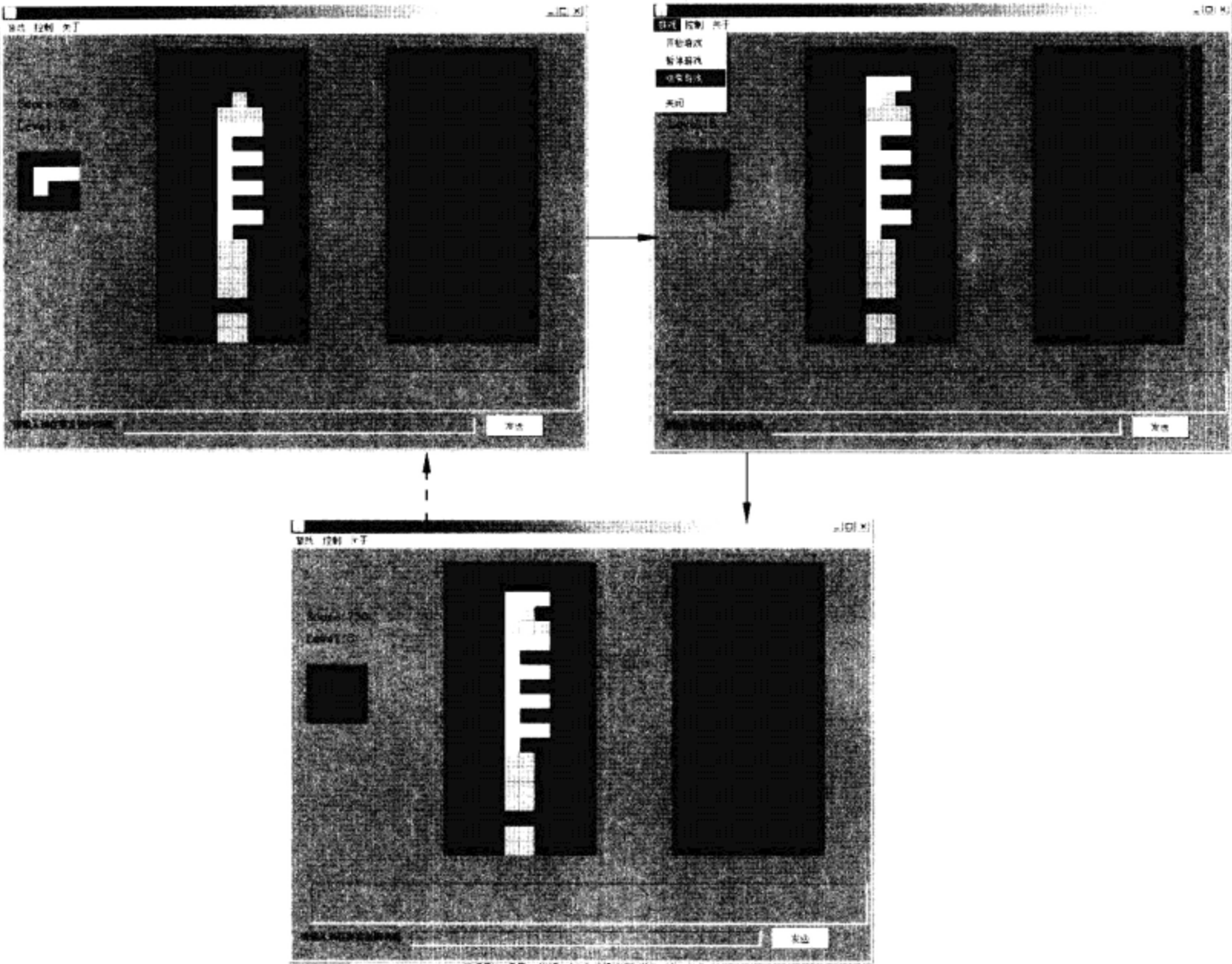


图 31.5 “结束游戏”菜单

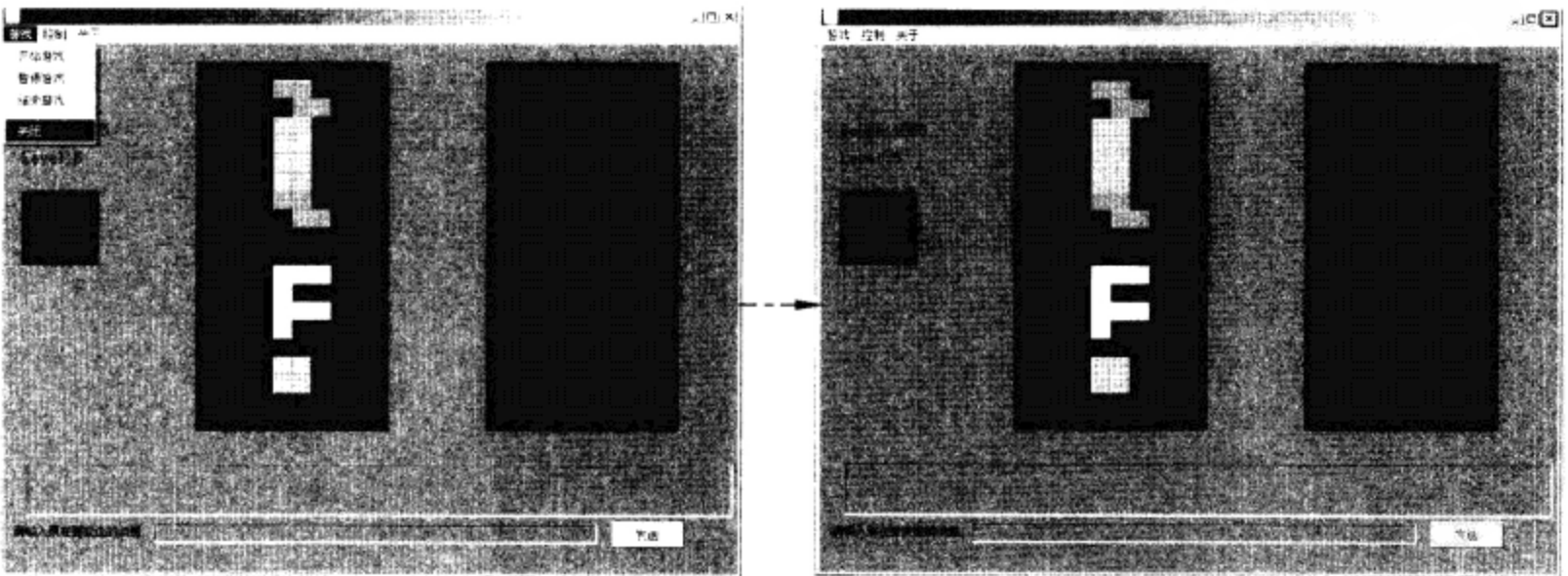


图 31.6 关闭游戏

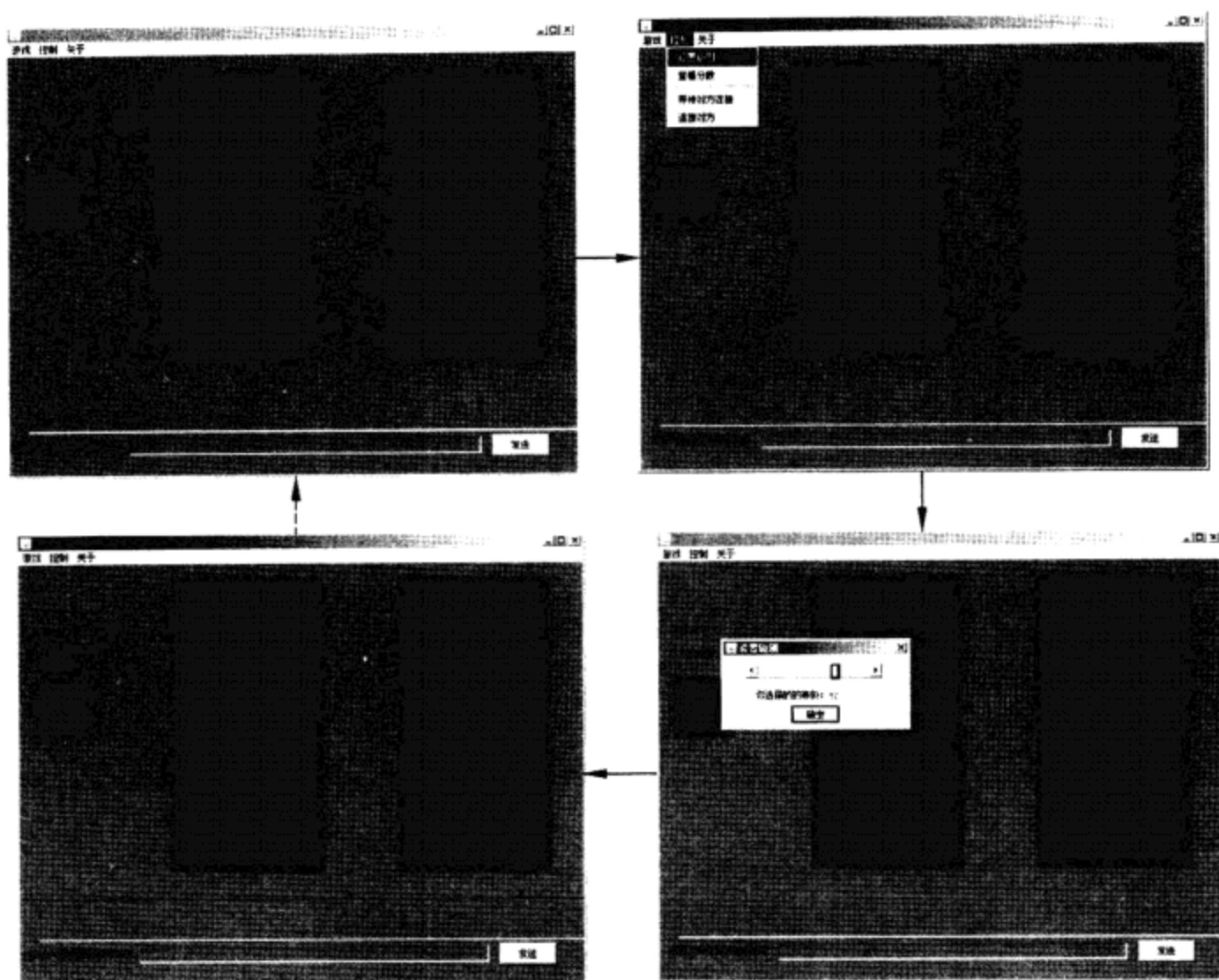


图 31.7 设置游戏级别

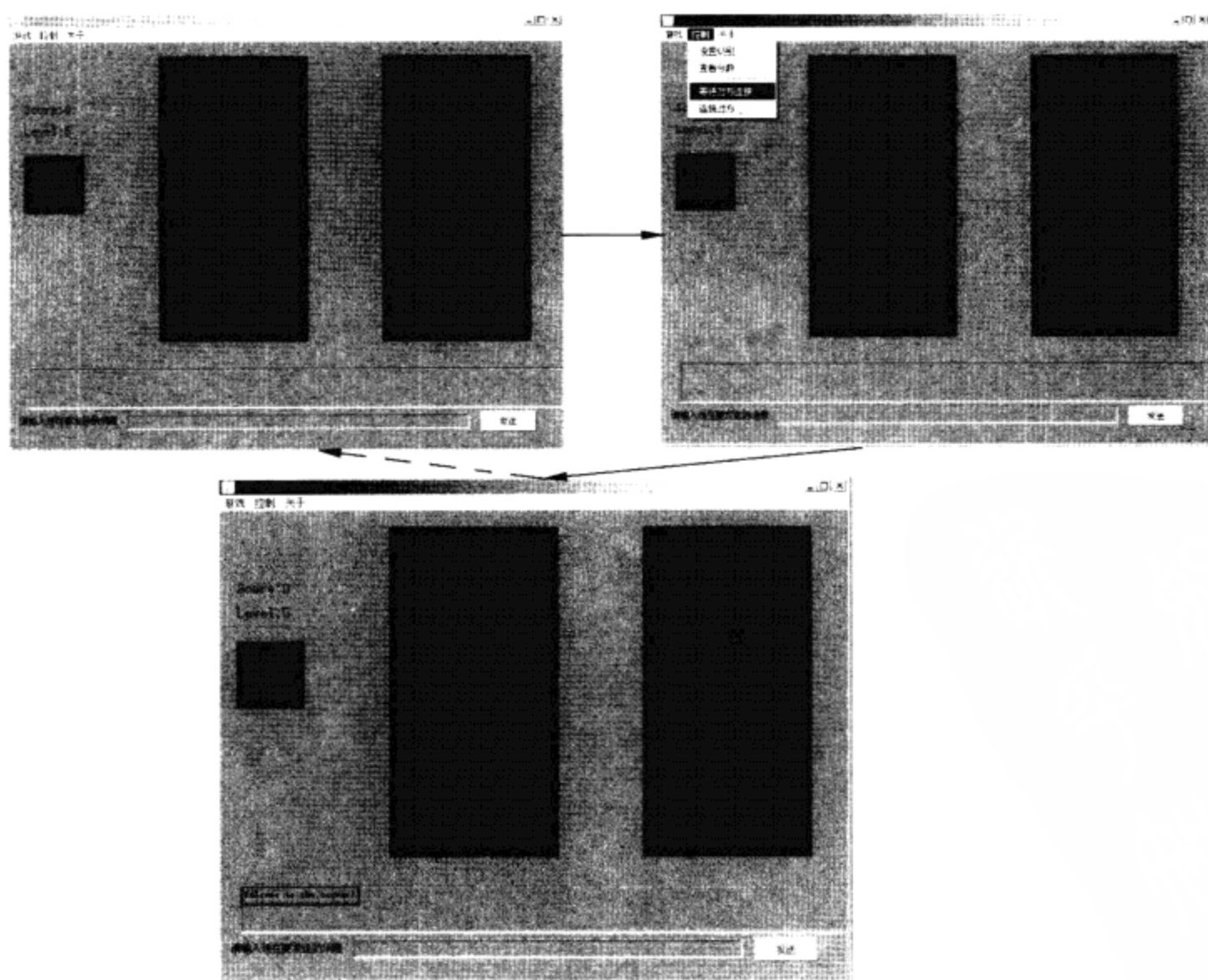


图 31.8 设置服务器

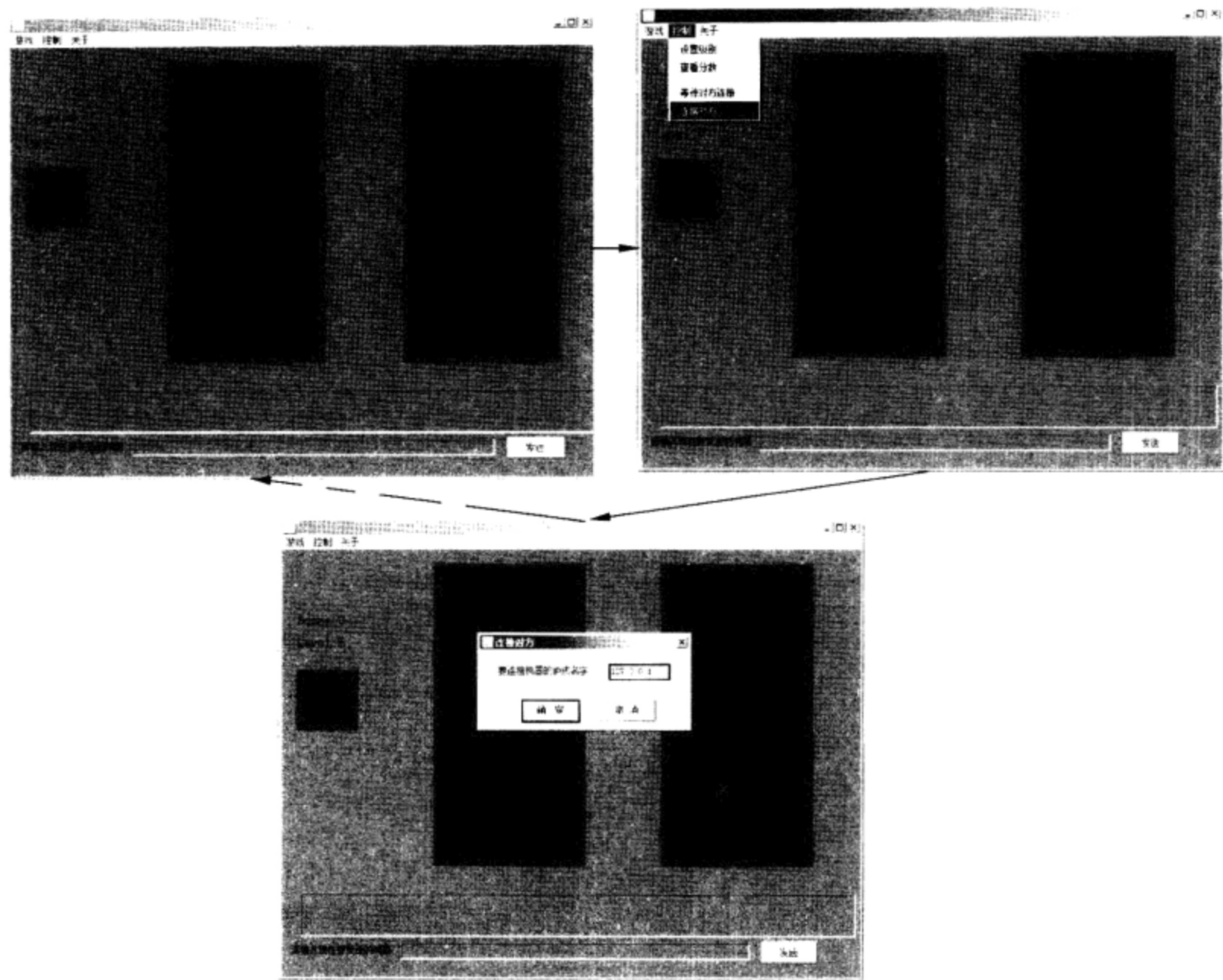


图 31.9 客户端的设置过程

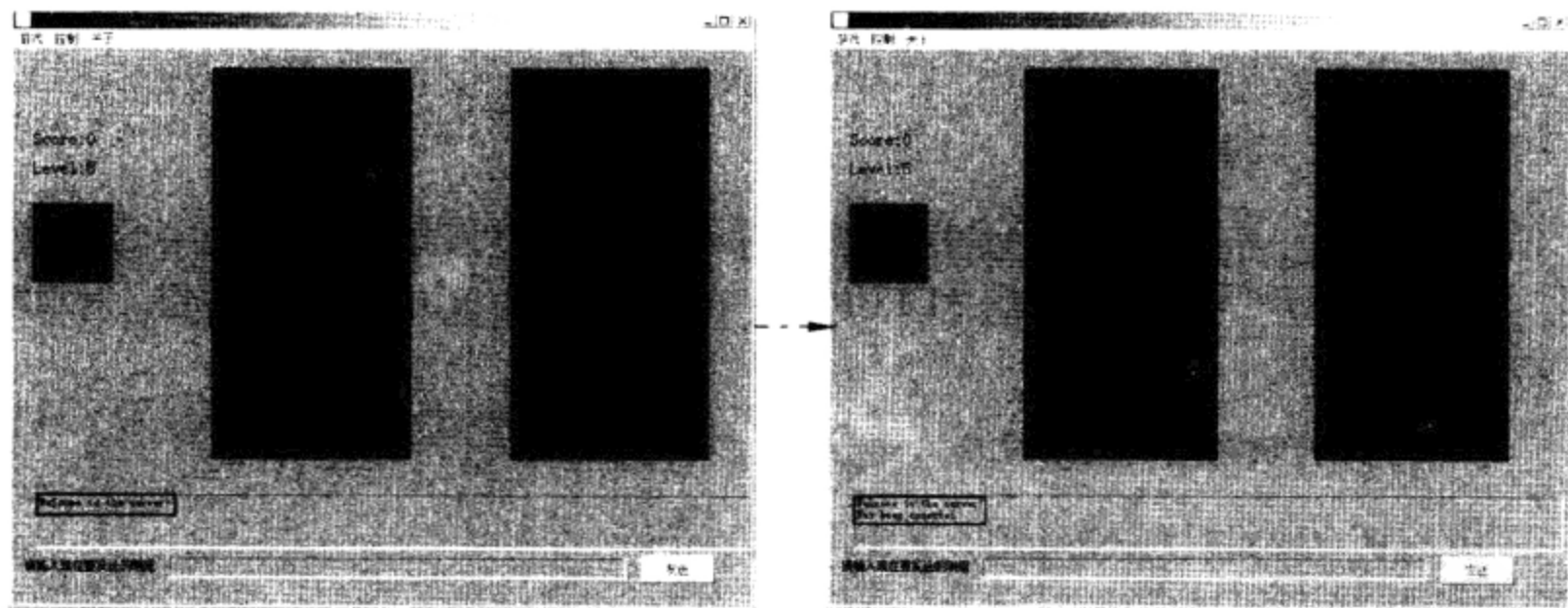


图 31.10 服务器端的变化过程

当俄罗斯方块游戏结束时，如果得分在前 10 名的范围内，则会出现“恭喜”对话框。在该对话框的“请输入你的名字”文本框中输入相应的名字，然后单击“确定”按钮就会存储到分数报告—Top10 里，具体过程如图 31.11 所示。如果选择“控制”|“查看分数”菜单命令，会出现“分数报告—Top10”对话框，具体过程如图 31.12 所示。

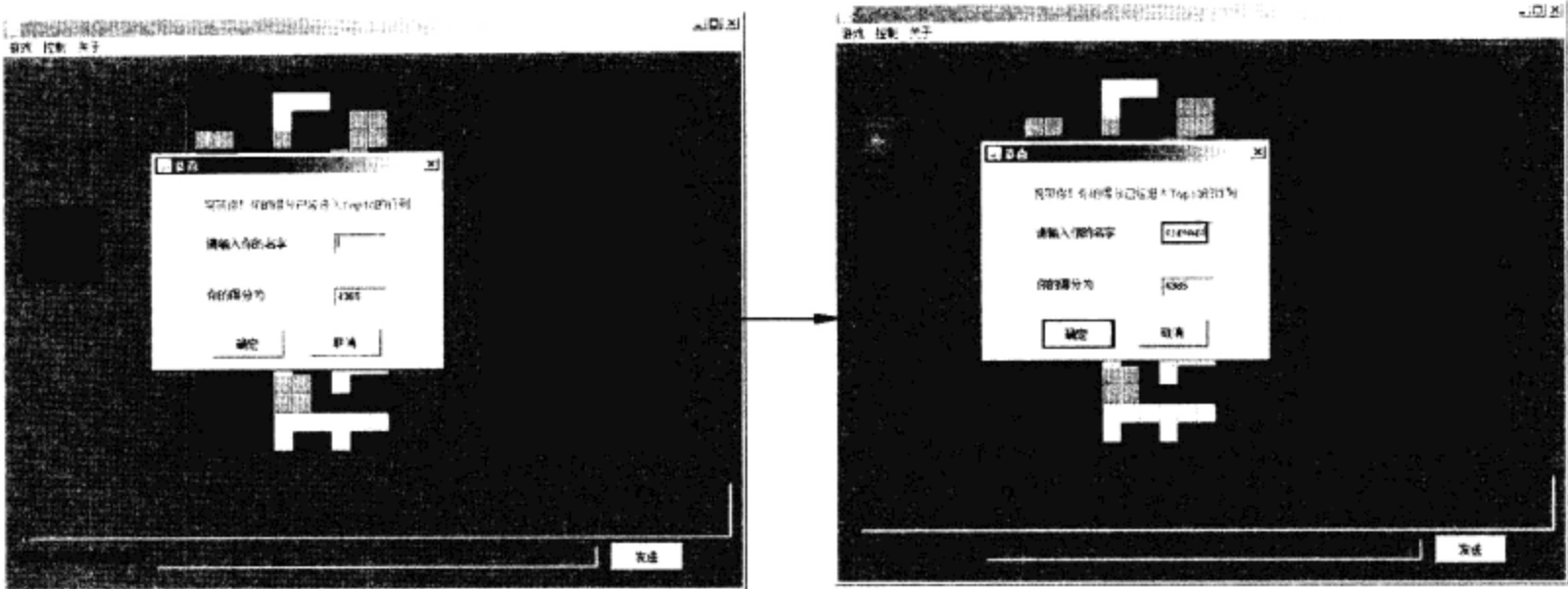


图 31.11 存储分数报告

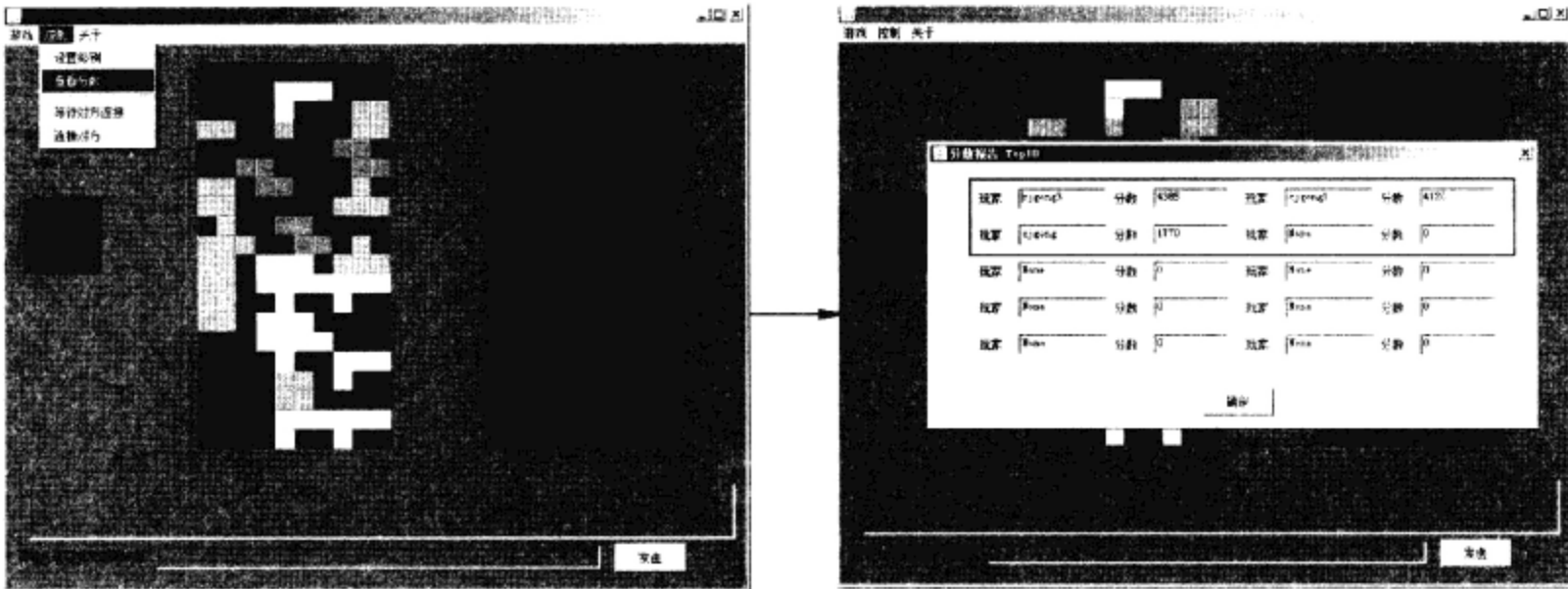


图 31.12 查看分数报告

4. “关于”菜单

当出现初始化界面后，如果想查看俄罗斯方块游戏的信息，可以选择“关于”菜单选项，会出现“关于”对话框，如图 31.13 所示。在该对话框中单击“确定”按钮，对话框就会消失。

5. 俄罗斯方块游戏的网络对战

当运行两个俄罗斯方块游戏时，设置其中一个为服务器端同时设置另一个为客户端后，两个运行程序就可以实现网络对战。在俄罗斯方块游戏里，只有服务器端有开始游戏、暂停游戏和结束游戏权限，如果在客户端通过菜单“游戏”|“开始游戏”命令启动游戏时，则会出现如图 31.14 所示的对话框。当在服务器端通过菜单“游戏”|“开始游戏”命令启动游戏后，就会实现网站游戏对战，具体过程如图

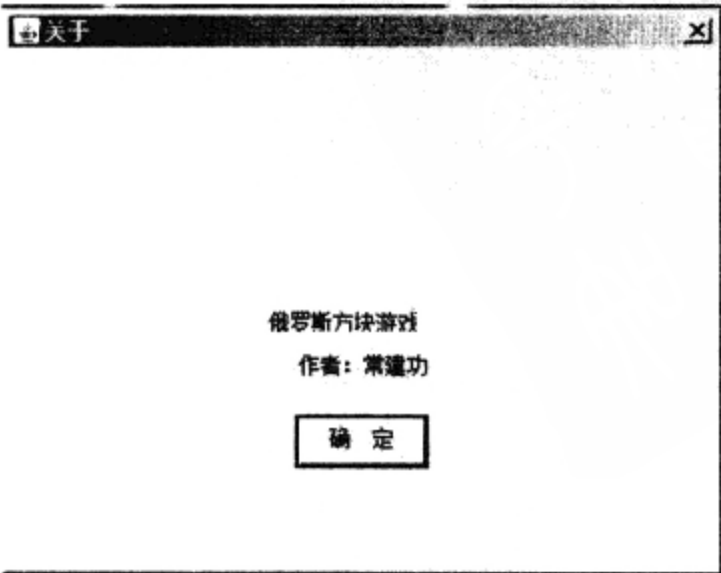


图 31.13 “关于”对话框

31.15 所示。当网络对战结束后，就会在双方界面出现不同的分数，具体过程如图 31.16 所示。

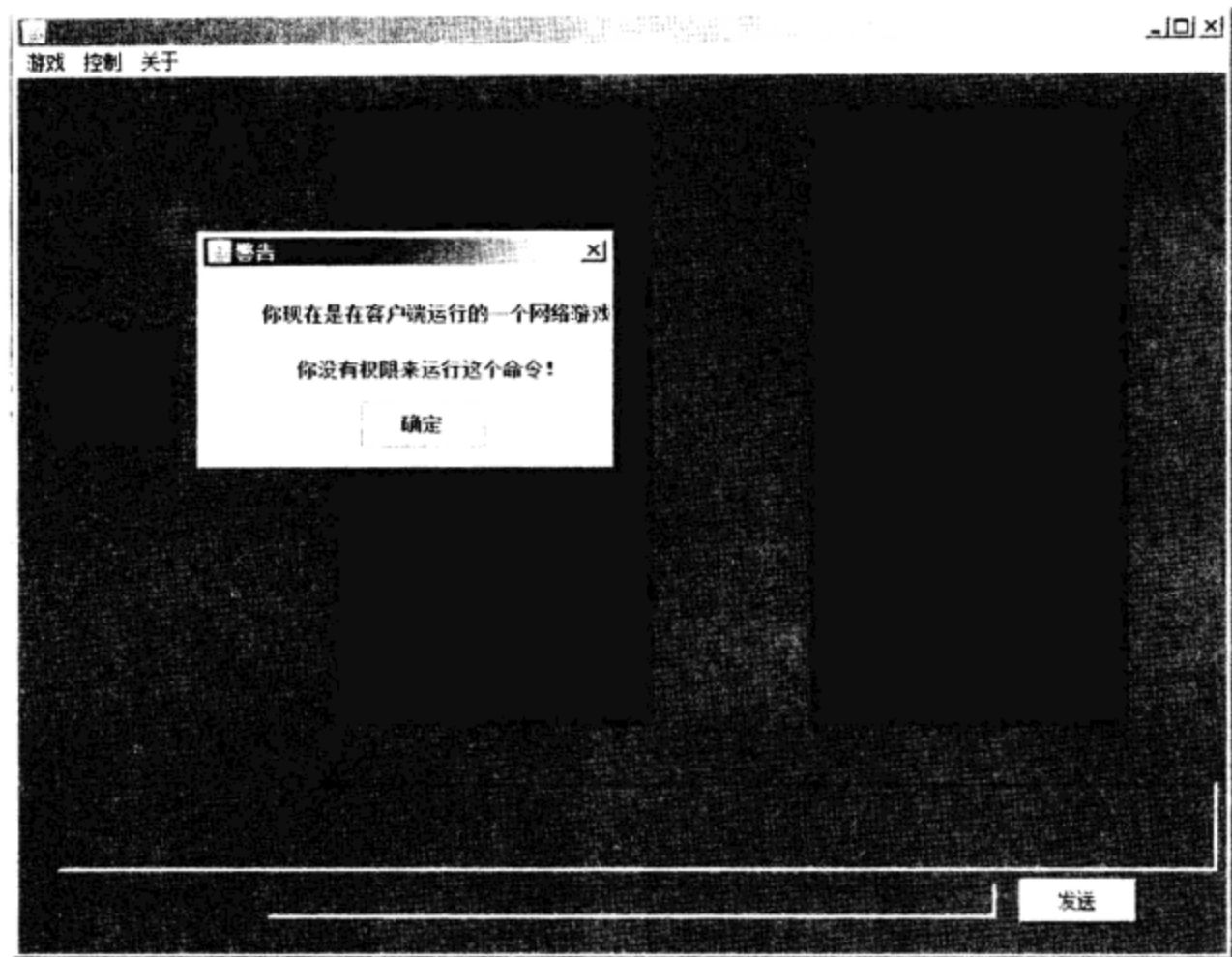


图 31.14 “警告”对话框

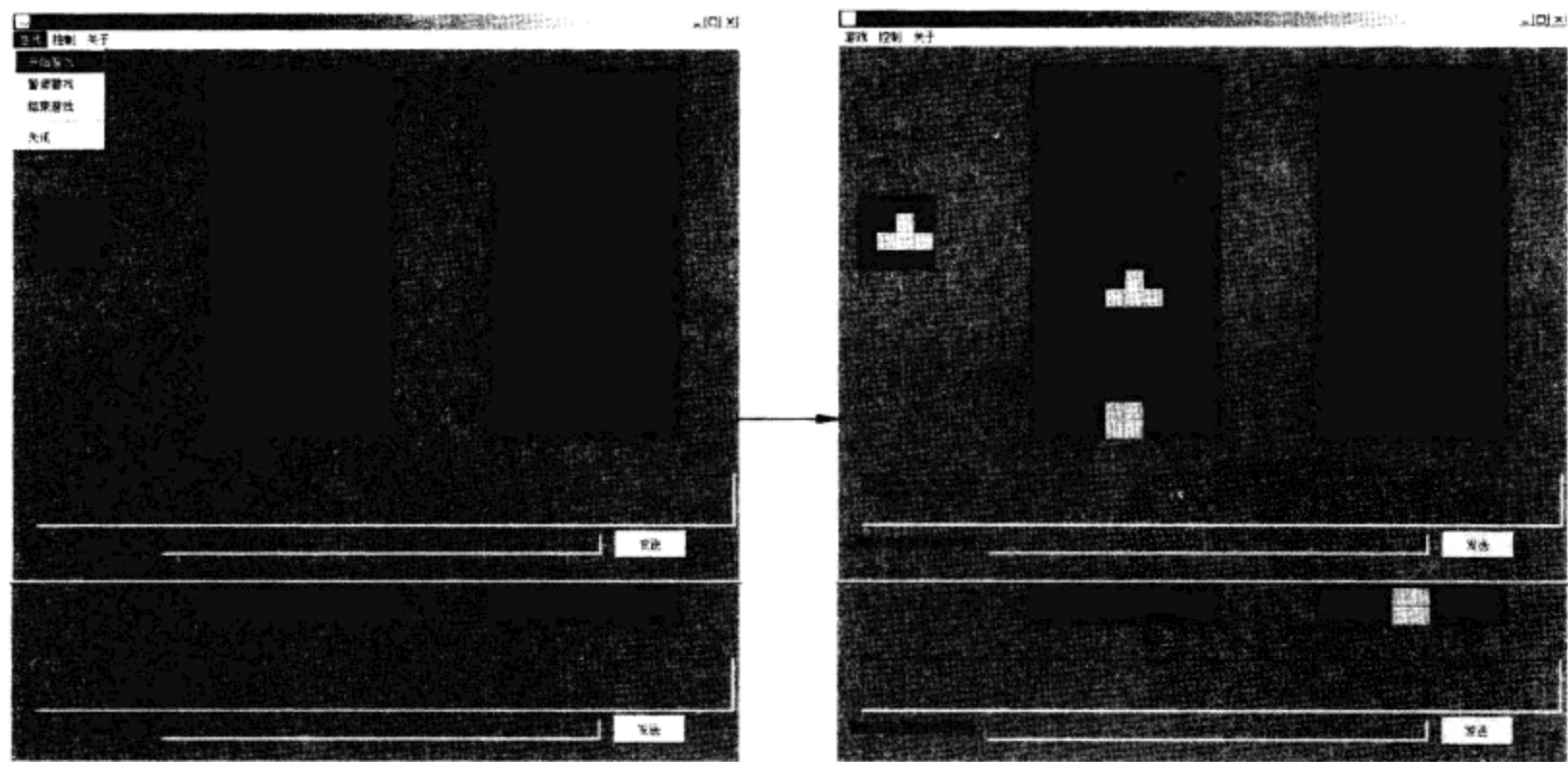


图 31.15 开始网络对战

在具体网络对战时，服务器端和客户端双方可以互相通信进行交流。例如当服务器端和客户端刚建立连接后，客户端首先可以在界面下方的“请输入现在要发送的消息”文本框中输入“Hello,ni hao!”文本，然后单击“发送”按钮，具体过程如图 31.17 所示。客户端的文本域就会显示“Hello,ni hao!”文本，而服务器端的文本域同样会多显示出“对手：

Hello,ni hao!”，具体过程如图 31.18 所示。

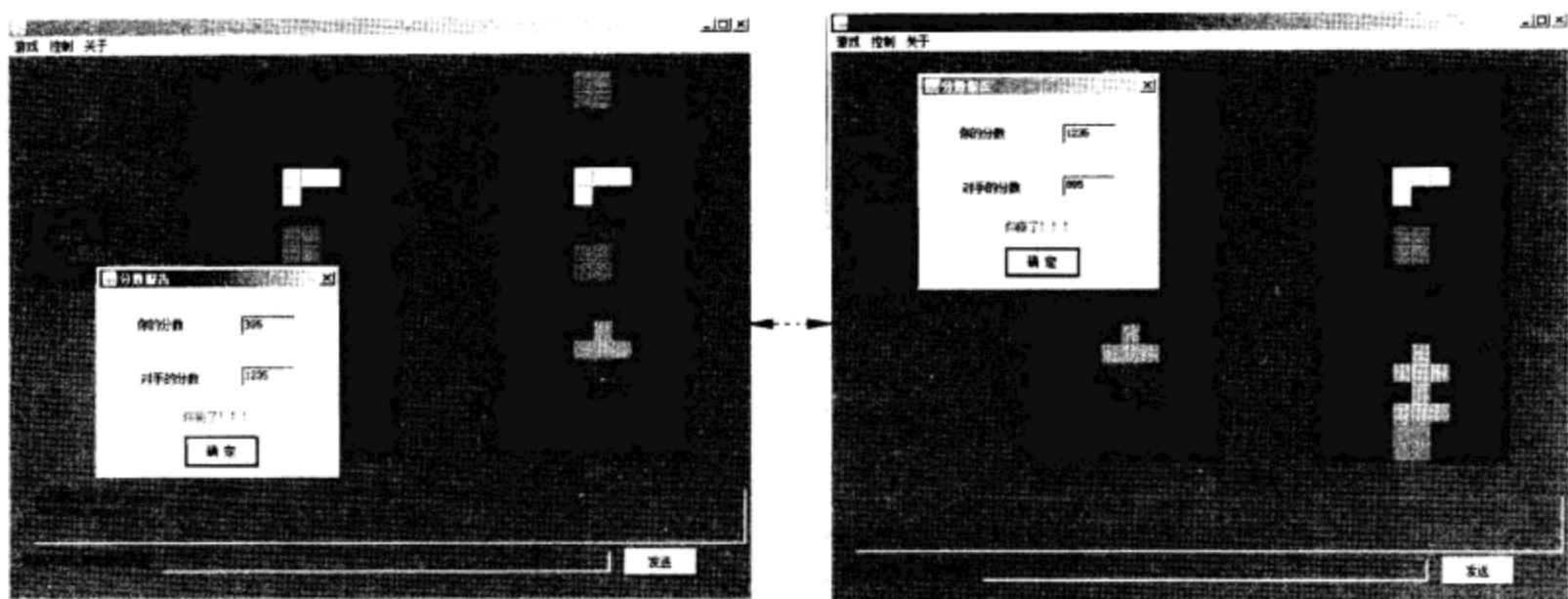


图 31.16 网络对战结束

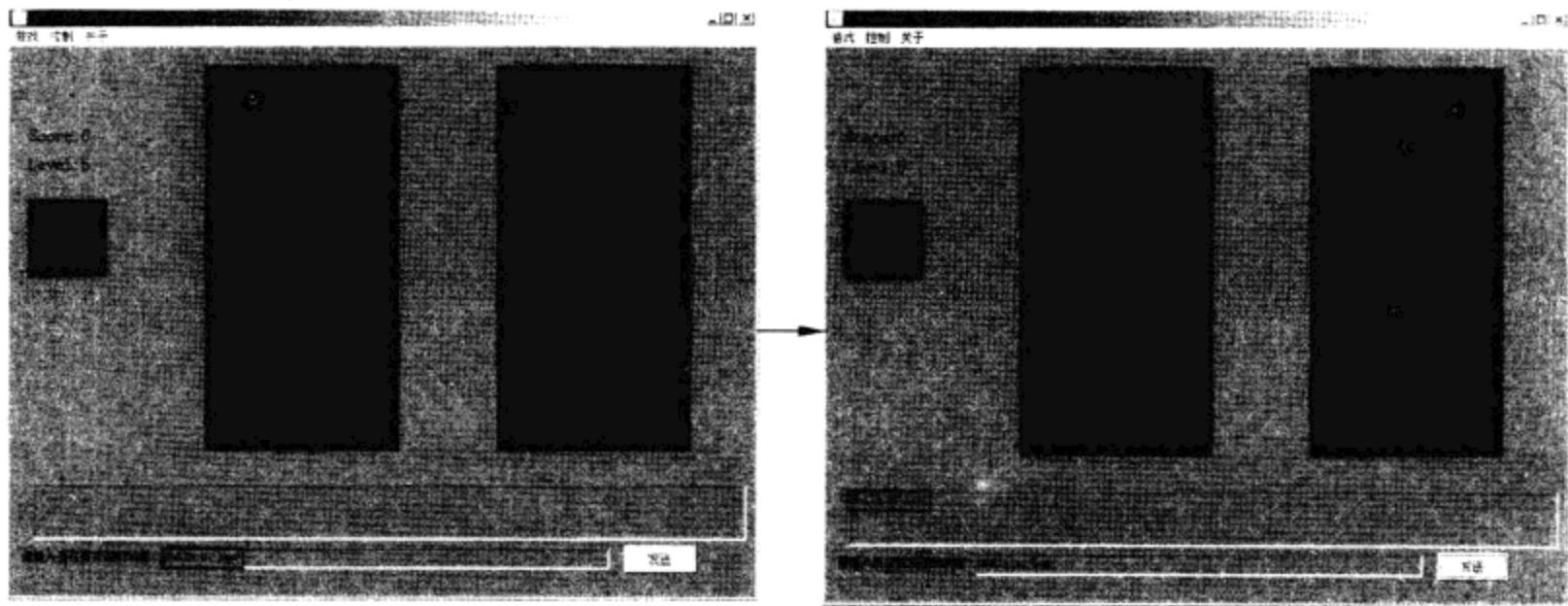


图 31.17 客户端发送信息的过程

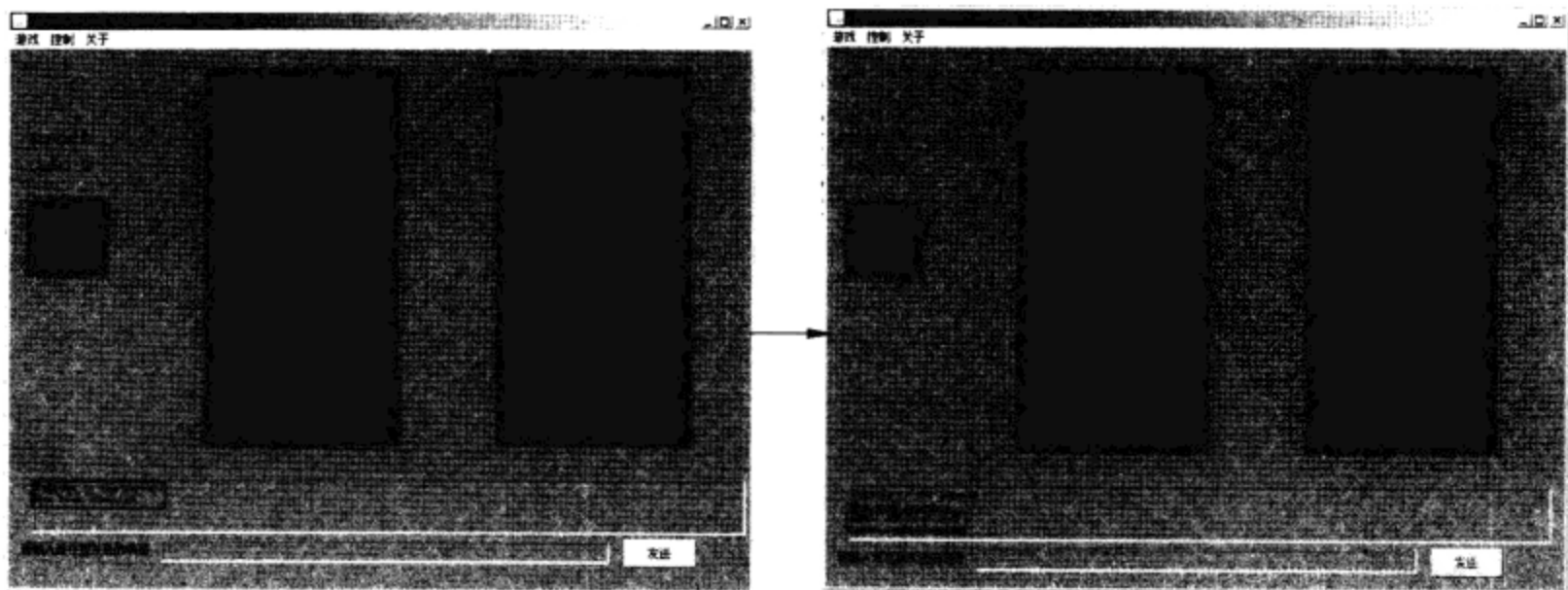


图 31.18 服务器端接收信息

⚠注意：在玩游戏时，如果想控制方向，可以通过方向键来实现，“↑”方向键用来实现方块图形的旋转。

31.2 俄罗斯方块游戏项目——初步设计涉及的对象

根据面向对象的思想，一个项目中涉及的对象都应该通过类来表示。那么俄罗斯方块游戏项目中涉及哪些类呢？每个类中都有什么属性和方法呢？通过分析俄罗斯方块游戏项目，可以发现该项目涉及两个对象，即正方形对象和俄罗斯方块对象。

31.2.1 正方形类

Shape 为俄罗斯方块游戏项目中方块图形对象的组成类，该类包含了正方形所具有的属性和动作的方法，具体内容如代码 31.1 所示，其 UML 如图 31.19 所示。

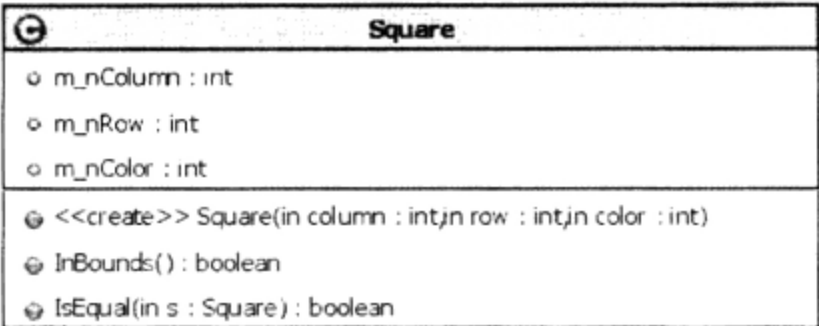


图 31.19 Shape 类图

代码 31.1 正方块图形的类: Square.java

```
class Square {
    //创建成员变量
    int m_nColumn;           //行数变量
    int m_nRow;              //列数变量
    int m_nColor;            //颜色变量
    Square(int column, int row, int color) { //构造函数
        //初始化成员变量
        this.m_nColumn = column;
        this.m_nRow = row;
        this.m_nColor = color;
    }
    boolean InBounds() { //判断是否出界
        return (m_nColumn >= 0 && m_nColumn < Tetrics.m_nCols && m_nRow >=
        0 && m_nRow < Tetrics.m_nRows + 4);
    }
    boolean IsEqual(Square s) { //判断是否相同的正方形
        return m_nColumn == s.m_nColumn && m_nRow == s.m_nRow
        && m_nColor == s.m_nColor;
    }
}
```

【代码解析】

在上述代码中，首先创建正方形的特性属性：表示行数变量的属性 m_nColumn、表示列数变量的属性 m_nRow 和表示颜色变量的属性 m_nColor，具体含义如图 31.20 所示。然后在构造函数

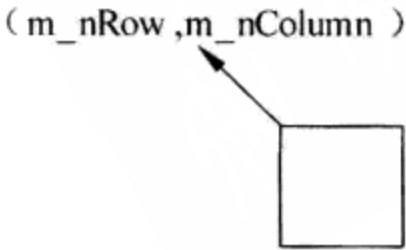


图 31.20 正方形属性

中初始化成员变量，最后创建判断是否出界的 InBounds()方法及判断是否相同的正方形的 IsEqual()方法。

31.2.2 俄罗斯方块类

通过 31.2.1 节的介绍可以知道，俄罗斯方块游戏项目涉及正方形对象和俄罗斯方块对象，那么创建完正方形类后，如何设计俄罗斯方块类呢？Tetricks 表示俄罗斯方块游戏项目中的方块对象，该类包含了俄罗斯方块游戏中方块对象所具有的属性 and 动作的方法，其 UML 如图 31.21 和图 31.22 所示。

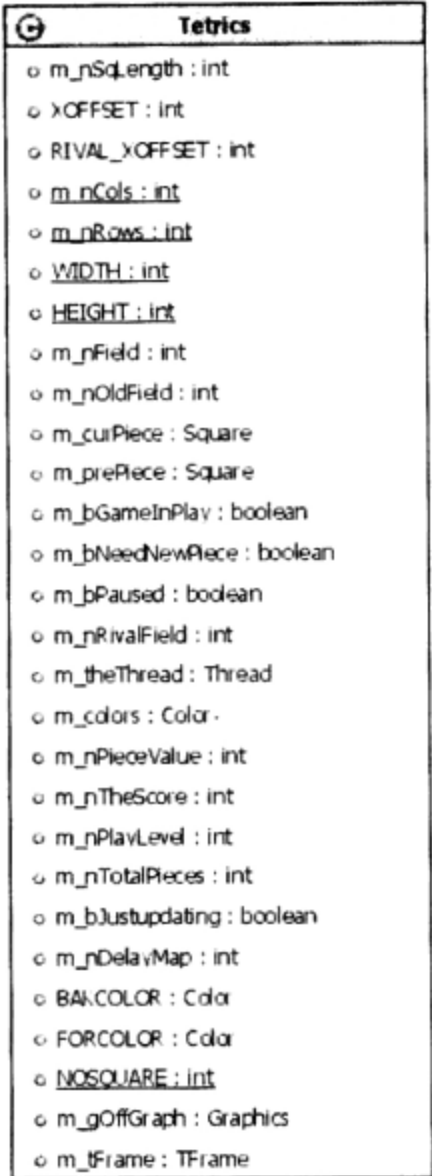


图 31.21 Tetricks 类的属性

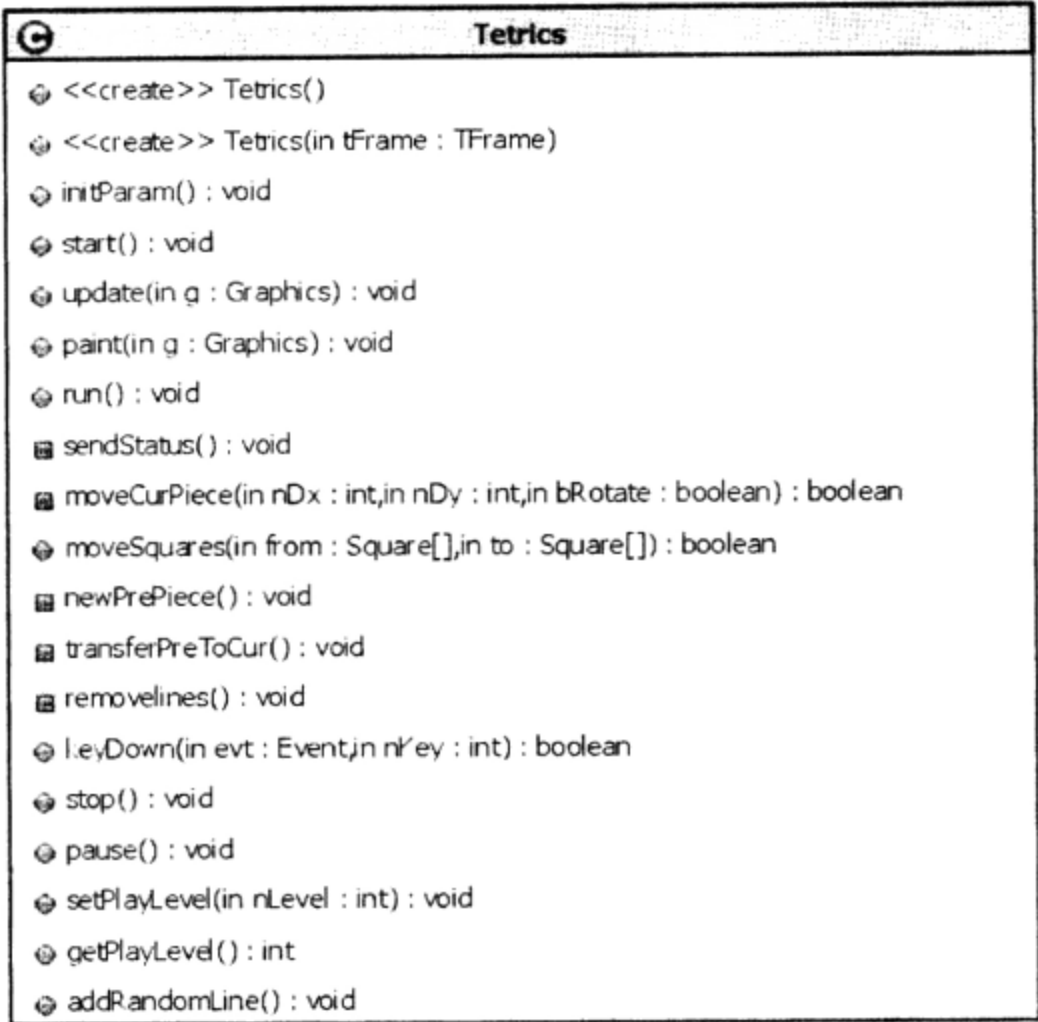


图 31.22 Tetricks 类的方法

1. 表示出俄罗斯方块

在具体设计俄罗斯方块游戏项目时，可以把显示俄罗斯方块的游戏界面看作是一个大大的表格，如图 31.23 所示。该表格中的一个格子就是组成俄罗斯方块图形的基本单位，其中一个格子可以表示为正方形对象（利用格子的坐标来表示正方形的相关属性），而几个连接在一起的格子就可以表示为俄罗斯方块对象。

通过分析可以知道，在具体实现方块图形对象时，可以通过一个存储正方形对象的数组来表示出俄罗斯方块对象，即用存储 4 个方阵格子的正方形对象来表示方块图形，如图 31.24 所示。

根据俄罗斯方块游戏规则可以知道, 每个方块图形对象都有多种状态。玩家在玩游戏时, 只会针对方块图形的一种状态, 而旋转操作则是显示该对象的下一种状态。既然已经表示出方阵图形的一种状态, 那么如何表示出多种状态呢? 具体的俄罗斯方块状态如图 31.25 所示。

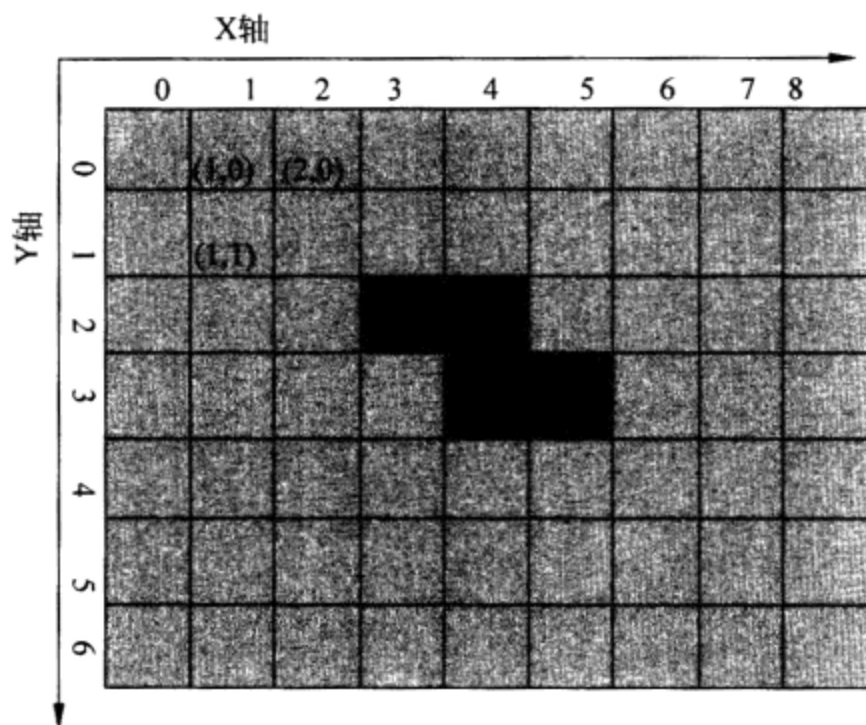


图 31.23 界面表格化

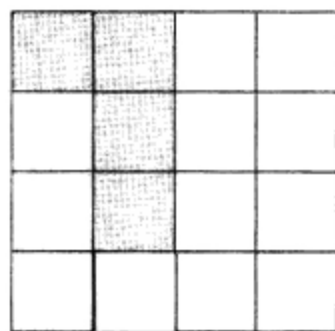


图 31.24 俄罗斯方阵

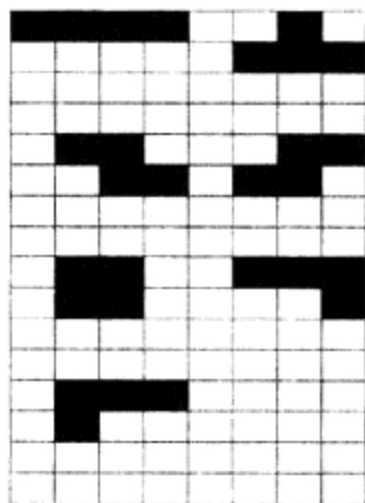


图 31.25 俄罗斯方块的多态

通过分析可以知道, 在具体实现方块图形对象时, 可以通过一个存储 4 个连接在一起的格子 (正方形) 表示出俄罗斯方块对象。为了能够显现出俄罗斯方块, 首先需要通过相应方法获取格子的相关坐标, 然后在相应的位置绘制正方形, 具体内容如代码 31.2 所示。

代码 31.2 俄罗斯方块的类: Tetrics.java

```
public class Tetrics extends Panel implements Runnable {
    //创建成员变量
    int m_nSqlLength; //正方块的大小: 宽, 高
    //信息区大小
    final int XOFFSET = 200;
    final int RIVAL_XOFFSET = 500;
    static int m_nCols;
    static int m_nRows;
    public static int WIDTH = 800;
    public static int HEIGHT = 450;
```

```

//当前的界面情况
int m_nField[][];
int m_nOldField[][];
//当前移动的方块
Square m_curPiece[] = new Square[4];
Square m_prePiece[] = new Square[4];
boolean m_bGameInPlay;
boolean m_bNeedNewPiece;
boolean m_bPaused = false;
//对手的情况
int m_nRivalField[][];
Thread m_theThread = null;
Color m_colors[];
int m_nPieceValue, m_nTheScore = 0;
int m_nPlayLevel;
int m_nTotalPieces;
boolean m_bJustupdating = false;
int m_nDelayMap[] = { 600, 600, 600, 600, 500, 400, 310, 250, 200, 150,
100 };
final Color BAKCOLOR = new Color(80, 123, 166);
final Color FORCOLOR = Color.black;
public static int NOSQUARE = 0;
Graphics m_gOffGraph;
TFrame m_tFrame;
public Tetrics() { //无参构造函数
    super();
    setBackground(BAKCOLOR); //设置背景颜色
    initParam(); //调用初始化方法
}
public Tetrics(TFrame tFrame) { //带参构造函数
    super();
    m_tFrame = tFrame; //初始化对象 m_tFrame
    setBackground(BAKCOLOR);
    initParam();
}
public void initParam() { //初始化方法
    //设置初始参数
    m_nSqLength = 20;
    m_nCols = 10;
    m_nRows = 20;
    m_nField = new int[m_nCols][m_nRows + 4];
    m_nOldField = new int[m_nCols][m_nRows + 4];
    //对手的状态
    m_nRivalField = new int[m_nCols][m_nRows + 4];
    m_nPlayLevel = 5; //游戏级别变量
    //定义8种颜色
    m_colors = new Color[8]; //定义存储颜色的数组对象 m_colors
    ... //省略部分代码
}
public synchronized void paint(Graphics g) { //重新绘制方法
    g.setFont(new Font("宋体", 0, 18)); //设置字体
    //输出 Score 和 level 的位置
    int gx = m_nSqLength;
    int gy = m_nSqLength * m_nRows / 4;
    //清除相应边框并输出分数
    g.clearRect(gx, gy - 25, XOFFSET - gx, 25);
    g.drawString("Score:" + m_nTheScore, gx, gy);
    gy += 31; //重新设置 gy 值
}

```

```

//清除相应边框并输出分数
g.clearRect(gx, gy - 25, XOFFSET - gx, 25);
g.drawString("Level:" + m_nPlayLevel, gx, gy);
//绘制预览的方块
int middle = m_nCols / 2;
int top = m_nRows;
gy += 31; //重新设置 gy
g.setColor(Color.black); //设置颜色
//绘制预览背景
g.fillRect(gx, gy, m_nSqLength * 4, m_nSqLength * 4);
if (m_bGameInPlay) { //当游戏处于运行状态
    for (int i = 0; i < 4; i++) { //通过循环绘制方块对象
        //设置颜色
        g.setColor(m_colors[m_prePiece[i].m_nColor]);
        //绘制俄罗斯方块对象
        g.fill3DRect((m_prePiece[i].m_nColumn - middle + 2)
            * m_nSqLength + gx, gy - (m_prePiece[i].m_nRow - top)
            * m_nSqLength, m_nSqLength, m_nSqLength, true);
    }
}
//用来画自己的游戏区域
Image img1 = createImage(m_nSqLength * 10, m_nSqLength * 20);
Graphics g1 = img1.getGraphics();
//用来画对手的游戏区域
Image img2 = createImage(m_nSqLength * 10, m_nSqLength * 20);
Graphics g2 = img2.getGraphics();
for (int i = 0; i < m_nCols; i++)
    for (int j = 0; j < m_nRows; j++) {
        //-1 代表游戏刚刚开始
        g1.setColor(m_colors[m_nField[i][m_nRows - 1 - j]]);
        g1.fill3DRect(m_nSqLength * i, m_nSqLength * j,
            m_nSqLength,
            m_nSqLength, true);
        //来画对手的情况
        g2.setColor(m_colors[m_nRivalField[i][m_nRows - 1 - j]]);
        g2.fill3DRect(m_nSqLength * i, m_nSqLength * j,
            m_nSqLength,
            m_nSqLength, true);
    }
//绘制自己和对方的图像
g.drawImage(img1, XOFFSET, m_nSqLength, this);
g.drawImage(img2, RIVAL_XOFFSET, m_nSqLength, this);
m_bJustupdating = false;
}
private void newPrePiece() { //产生一个新的方块用来做预览
    int middle = m_nCols / 2;
    int top = m_nRows;
    switch ((int) (Math.random() * 7)) {
    case 0:
        m_nPieceValue = 100;
        m_prePiece[0] = new Square(middle - 1, top - 1, 1);
        m_prePiece[1] = new Square(middle - 2, top - 1, 1);
        m_prePiece[2] = new Square(middle, top - 1, 1);
        m_prePiece[3] = new Square(middle + 1, top - 1, 1);
        break;
    ... //省略部分代码
    }
}

```

```

private void transferPreToCur() {           //将预览的方块转变成正在动的方块
    Square old[] = new Square[4];
    old[0] = old[1] = old[2] = old[3] = new Square(-1, -1, 0);
    for (int i = 0; i < 4; i++) {
        m_curPiece[i] = m_prePiece[i];
    }
    m_bGameInPlay = moveSquares(old, m_curPiece);
    if (!m_bGameInPlay && m_tFrame.m_nNetStatus != TFrame.NOCONNECT)
        m_tFrame.sendStr("GameOver:" + m_nTheScore);
    else if (!m_bGameInPlay && m_tFrame != null)
        m_tFrame.insertScoreReport(m_nTheScore);
}

public void run() {                          //实现 run() 方法
    while (m_bGameInPlay) {                  //循环
        //线程休眠时间 t
        try {
            int t;
            if (m_nPlayLevel > 10)
                t = 75;
            else
                t = m_nDelayMap[m_nPlayLevel];
            Thread.sleep(t);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        //判断是否需要新的俄罗斯方块对象
        if (m_bNeedNewPiece) {
            if (m_nPieceValue > 0) {
                m_nTheScore += m_nPieceValue;
                m_nTotalPieces += 1;
                if (m_nTotalPieces % 31 == 0)
                    m_nPlayLevel++;
            }
            removelines();
            transferPreToCur();
            newPrePiece();
            m_bNeedNewPiece = false;
        } else {
            m_bNeedNewPiece = !moveCurPiece(0, -1, false);
            if (!m_bNeedNewPiece)
                m_nPieceValue -= 5;
        }
        repaint();
        sendStatus();
    }
    m_theThread = null;
}

...                                         //省略部分代码
}

```

【代码解析】

在上述代码中，首先创建了各种成员变量，然后又构造了两种构造函数，最后为了便于操作又专门创建了初始化参数方法 `iParam()`、重新绘制方法 `paint()`、产生预览俄罗斯方块方法 `newPrePiece()` 和将预览俄罗斯方块转换成正在动的俄罗斯方块。

2. 俄罗斯方块移动

通过上面的内容已经可以表示出俄罗斯方块图形对象，如何实现方块图形对象的移动

功能呢? 由于方块图形对象会自动向下移动, 所以方块图形对象里需要有以下几个表示其位置的变量: 变量 `left` 表示方块图形到左边界的距离, 变量 `top` 表示方块图形到上边界的距离。图形的移动实际上就是改变变量 `left` 和 `top` 的值, 如图 31.26 所示。

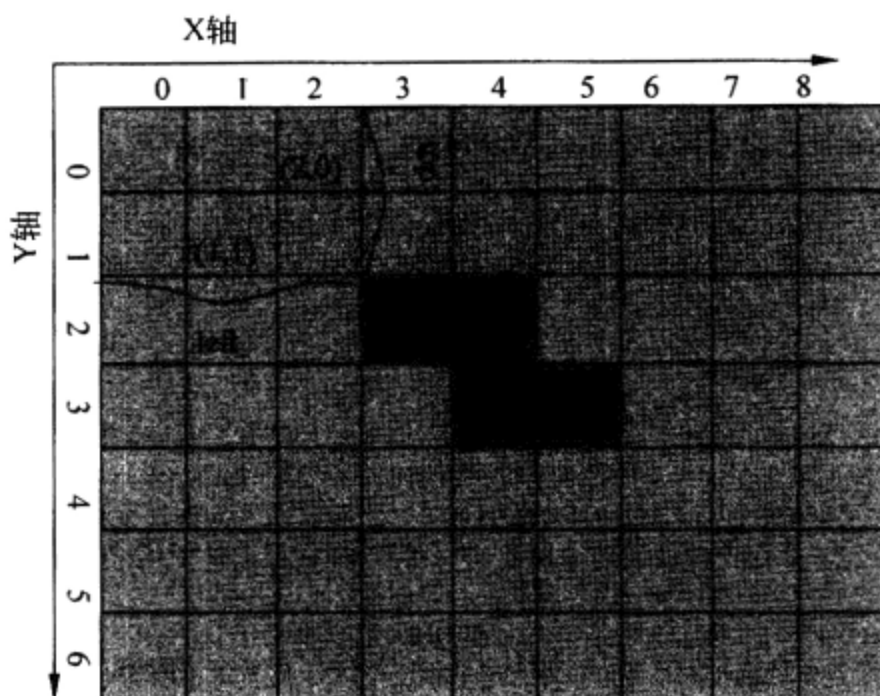


图 31.26 位置的表示

如果要把方块图形对象在游戏显示面板上画出来, 即把组成图形的格子画出来 (即把 4×4 方阵中标志为 1 的格子画出来, 标志为 0 的格子不画出来), 则需要知道这些格子的位置, 如何获取这些格子的位置呢? 可以通过如下表达式来获取:

`x=left+格子的 x 相对坐标`
`y=top+格子的 y 相对坐标`

通过分析可以知道, 只有不断地重新绘制新坐标的方块才可以实现俄罗斯方块对象的移动, 俄罗斯方块移动的具体内容如代码 31.3 所示。

代码 31.3 俄罗斯方块的移动: Tetricks.java

```
public class Tetricks extends Panel implements Runnable {
    //实现移动方块
    private synchronized boolean moveCurPiece(int nDx, int nDy, boolean
    bRotate) {
        Square newpos[] = new Square[4];
        for (int i = 0; i < 4; i++) {
            if (bRotate) {
                int dx = m_curPiece[i].m_nColumn - m_curPiece[0].m_nColumn;
                int dy = m_curPiece[i].m_nRow - m_curPiece[0].m_nRow;
                newpos[i] = new Square(m_curPiece[0].m_nColumn - dy,
                    m_curPiece[0].m_nRow + dx, m_curPiece[i].m_nColor);
            } else {
                newpos[i] = new Square(m_curPiece[i].m_nColumn + nDx,
                    m_curPiece[i].m_nRow + nDy, m_curPiece[i].m_nColor);
            }
        }
        if (moveSquares(m_curPiece, newpos) == false)
            return false;
        m_curPiece = newpos;
        return true;
    }
}
```

```

}
//移动方块, 如果不能移动, 则返回假
boolean moveSquares(Square from[], Square to[]) {
    //判断是否能移动
    outerlable: for (int i = 0; i < to.length; i++) {
        if (to[i].InBounds() == false)
            return false;
        //如果不在可玩区域, 则返回假
        if (m_nField[to[i].m_nColumn][to[i].m_nRow] != 0) {
            for (int j = 0; j < from.length; j++)
                if (to[i].IsEqual(from[j]))
                    continue outerlable;
            return false;
        }
    }
    //移动
    for (int i = 0; i < from.length; i++)
        if (from[i].InBounds())
            m_nField[from[i].m_nColumn][from[i].m_nRow] = 0;

    for (int i = 0; i < to.length; i++)
        m_nField[to[i].m_nColumn][to[i].m_nRow] = to[i].m_nColor;
    return true;
}

public boolean keyDown(Event evt, int nKey) { //键盘按钮的处理方法
    //使游戏处于运行状态
    if (!m_bGameInPlay)
        return true;
    if (m_bPaused)
        return true;
    switch (nKey) { //键盘的编号
        case 'a': //当按键为“a、←”
            case Event.LEFT:
                //实现向左移动
                moveCurPiece(-1, 0, false);
                m_bNeedNewPiece = false;
                repaint();
                break;
        case 'd': //当按键为“d、→”
            case Event.RIGHT:
                //实现向右移动
                moveCurPiece(1, 0, false);
                m_bNeedNewPiece = false;
                repaint();
                break;
        case 'w': //当按键为“w、↑”
            case Event.UP:
                //实现向上移动
                moveCurPiece(0, 0, true);
                repaint();
                break;
        case 's': //当按键为“s、↓”
            case Event.DOWN:
                //实现不停地向下移动
                while (moveCurPiece(0, -1, false))
                    ;
                repaint();
                break;
    }
}

```

```

        return true;
    }
    private void removelines() {
        outerlabel: for (int j = 0; j < m_nRows; j++) { //去掉可以消失的行
            for (int i = 0; i < m_nCols; i++)
                if (m_nField[i][j] == 0)
                    continue outerlabel;
            for (int k = j; k < m_nRows - 1; k++)
                for (int i = 0; i < m_nCols; i++)
                    m_nField[i][k] = m_nField[i][k + 1];
            j -= 1;
            m_tFrame.sendStr("RemoveLine");
        }
    }
    ...
}
//省略部分代码

```

【代码解析】

在上述代码中，首先创建了实现移动功能的 `moveCurPiece()` 方法，然后又创建了判断是否能够移动的 `moveSquares()` 方法，最后还创建了处理按键的 `keyDown()` 方法和消除可消失行的 `removelines()` 方法。

3. 俄罗斯方块游戏控制

由于在俄罗斯游戏界面里存在游戏开始、停止和暂停的菜单，所以需要创建菜单项处理的方法，游戏控制的方法如代码 31.4 所示。

代码 31.4 俄罗斯方块游戏控制的方法: Tetrics.java

```

public class Tetrics extends Panel implements Runnable {
    public synchronized void start() { //开始方法
        if (m_theThread != null)
            //游戏是被暂停，而不是重新开始
        {
            m_bPaused = false;
            m_theThread.resume(); //恢复线程对象
            return;
        }
        repaint(); //重绘
        //重新开始赋上游戏的状态
        for (int i = 0; i < m_nCols; i++) {
            for (int j = 0; j < m_nRows + 4; j++) {
                m_nField[i][j] = 0;
                m_nOldField[i][j] = -1;
                m_nRivalField[i][j] = 0;
            }
        }
        m_nTheScore = 0;
        m_nTotalPieces = 0;
        m_bNeedNewPiece = true;
        m_bGameInPlay = true;
        m_theThread = new Thread(this); //创建线程对象 m_theThread
        newPrePiece(); //产生新的方块做预览
        m_theThread.start(); //启动线程
        requestFocus(); //请求组件获取焦点
    }
}

```

```

public synchronized void stop() { //停止方法
    if (m_theThread != null)
        m_theThread.stop(); //线程停止
    m_theThread = null;
}
public synchronized void pause() { //暂停方法
    if (m_theThread != null) {
        try {
            m_theThread.suspend();
            m_bPaused = true;
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
}

```

【代码解析】

在上述代码中，start()方法控制游戏启动，stop()方法控制游戏结束，pause()方法控制游戏暂停。

4. 俄罗斯方块游戏背景网格

在俄罗斯方块游戏的游戏区域中，为了便于游戏玩家的操作，专门创建了网格。创建网格的方法如代码 31.5 所示。

代码 31.5 绘制网格方法：Tetricks.java

```

public synchronized void addRandomLine() { //绘制表格
    int nRandom[] = new int[m_nCols]; //创建数组
    boolean bAllZero = true; //创建布尔类型变量 bAllZero
    boolean bNoZero = true; //创建布尔类型变量 bNoZero
    for (int i = 0; i < m_nCols; i++) {
        nRandom[i] = (int) (7 * Math.random());
        if (nRandom[i] != 0)
            bAllZero = false;
        else
            bNoZero = false;
    }
    if (bAllZero) {
        nRandom[(int) (m_nCols * Math.random())] = (int) (Math.random()
            * 6 + 1);
    } else if (bNoZero) {
        nRandom[(int) (m_nCols * Math.random())] = 0;
    }
    for (int nCol = 0; nCol < m_nCols; nCol++)
        for (int nRow = m_nRows + 3; nRow > 0; nRow--) {
            m_nField[nCol][nRow] = m_nField[nCol][nRow - 1];
        }
    for (int nCol = 0; nCol < m_nCols; nCol++)
        m_nField[nCol][0] = nRandom[nCol];
    for (int i = 0; i < 4; i++) {
        m_curPiece[i].m_nRow++;
    }
}
public void setPlayLevel(int nLevel) {
    //属性 playLevel 的 getter() 和 setter() 方法
}

```

```

m_nPlayLevel = nLevel;                //初始化变量 m_nPlayLevel
Graphics g = getGraphics();           //获取图形变量 g
g.setFont(new Font("宋体", 0, 18));   //设置字体
//初始化变量 int gx 和 int gy
int gx = m_nSqLength;
int gy = m_nSqLength * m_nRows / 4 + 31;
//调用清除方法
g.clearRect(gx, gy - 25, XOFFSET - gx, 25);
//输出游戏级别信息
g.drawString("Level:" + m_nPlayLevel, gx, gy);
}
public int getPlayLevel() {
    return m_nPlayLevel;
}

```

【代码解析】

在上述代码中, 为了能够设置和获取游戏级别, 专门创建了 `setPlayLevel()` 和 `getPlayLevel()` 方法。同时还创建了添加网格的 `addRandomLine()` 方法。

31.2.3 俄罗斯方块游戏项目的 TOP10 分数对象

在俄罗斯方块游戏项目中还涉及分数对象, 该对象主要用来存储玩家分数的 TOP 记录, 该对象的 `Score` 类的具体内容如代码 31.6 所示, 该类的 UML 如图 31.27 所示。

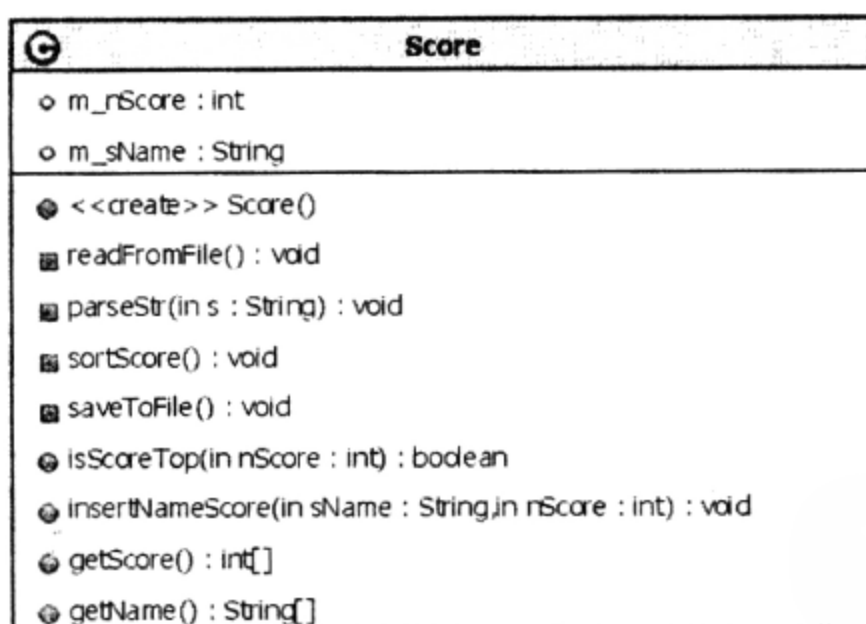


图 31.27 `Score` 类的类图

代码 31.6 TOP10 分数对象: `Score.java`

```

public class Score {
    //创建成员变量
    int[] m_nScore = new int[10];           //创建分数数组对象 m_nScore
    String[] m_sName = new String[10];      //创建游戏玩家姓名数组对象 m_sName
    public Score() {                        //构造函数
        for (int i = 0; i < 10; i++) {      //初始化数组 m_nScore 和 m_sName
            m_nScore[i] = 0;
            m_sName[i] = "None";
        }
    }
}

```

```

    }
    readFromFile(); //调用方法 readFromFile()
}
private void readFromFile() { //从文件中读取信息
    String ObjStr = ""; //创建空字符串对象
    try {
        Reader in = new FileReader("res/a.txt"); //获取文件输入流对象 in
        char[] buff = new char[4096]; //创建字符数组对象 buff
        int nch; //创建整型变量 nch
        //通过循环读取文件里的信息对象
        while ((nch = in.read(buff, 0, buff.length)) != -1) {
            ObjStr = ObjStr + (new String(buff, 0, nch));
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    parseStr(ObjStr);
}
private void saveToFile() { //保存分数到文件中
    sortScore(); //调用分数排序方法
    String sStr = ""; //创建空字符串对象 sStr
    for (int i = 0; i < 10; i++) { //通过循环创建字符串对象 sStr
        if (m_sName[i].trim().equals(""))
            m_sName[i] = "none";
        sStr += m_sName[i].trim() + "@" + m_nScore[i] + "|";
    }
    try {
        File f = new File("res/a.txt"); //创建文件对象 f
        if (!f.exists()) //当文件不存在时
            f.createNewFile(); //则创建文件
        //文件的输出流对象
        Writer wr = new FileWriter(f);
        wr.write(sStr); //把字符串写入文件中
        wr.flush(); //清空缓存区
    } catch (Exception e) {
        e.printStackTrace();
    }
}
private void parseStr(String s) { //解析字符串
    //创建对象 st
    StringTokenizer st = new StringTokenizer(s, "|");
    for (int i = 0; i < 10; i++) {
        if (st.hasMoreTokens()) {
            String sStr = st.nextToken();
            StringTokenizer stt = new StringTokenizer(sStr, "@");
            m_sName[i] = stt.nextToken(); //初始化数组对象 m_sName
            try {
                //初始化数组对象 m_nScore
                m_nScore[i] = Integer.parseInt(stt.nextToken());
            } catch (Exception e) {
                m_nScore[i] = 0;
            }
        } else {
            //数组 m_sName 和 m_nScore 的默认值
            m_sName[i] = "None";
            m_nScore[i] = 0;
        }
    }
}

```



```

    }
    private void sortScore() { //排序分数
        //创建成员变量 nTempScore 和 sTempName
        int nTempScore = 0;
        String sTempName = "None";
        for (int i = 0; i < 10; i++) { //实现分数排序
            for (int j = i; j < 10; j++) {
                if (m_nScore[i] < m_nScore[j]) {
                    nTempScore = m_nScore[j];
                    sTempName = m_sName[j];
                    m_nScore[j] = m_nScore[i];
                    m_sName[j] = m_sName[i];
                    m_nScore[i] = nTempScore;
                    m_sName[i] = sTempName;
                }
            }
        }
    }
    public boolean isScoreTop(int nScore) { //判断是否为前 10 的分数
        boolean bIsTop = false; //创建布尔变量 bIsTop
        for (int i = 0; i < 10; i++) { //通过循环实现判断
            if (m_nScore[i] < nScore)
                bIsTop = true;
        }
        return bIsTop; //返回对象 bIsTop
    }
    //实现游戏玩家和分数的插入
    public void insertNameScore(String sName, int nScore) {
        if (!isScoreTop(nScore))
            return;
        sortScore(); //排序
        //插入用户名和密码
        m_nScore[9] = nScore;
        m_sName[9] = sName;
        sortScore();
        saveToFile(); //保存字符到文件中
    }
    public int[] getScore() { //属性 m_nScore 的 getter() 和 setter() 方法
        return m_nScore;
    }
    public String[] getName() {
        return m_sName;
    }
}

```

【代码解析】

- ❑ 在上述代码中为了便于操作，专门创建了两个成员变量 `m_nScore` 和 `m_sName`，并同时创建了这两个成员变量的操作方法 `getScore()` 和 `getName()`。
- ❑ 当玩家玩完游戏后，就会在相应对话框中显示出玩家的分数。如果该分数通过 `isScoreTop()` 方法判断为 TOP10，则通过 `insertNameScore()` 方法插入 TOP10 里，同时通过 `sortScore()` 方法实现排序，最后再通过 `saveToFile()` 方法写入相应文件中。
- ❑ 当玩家想查看 TOP10 时，则可以通过 `readFromFile()` 方法读取相应文件中的信息，并通过 `parseStr()` 方法解析读取的字符串。

31.3 俄罗斯方块游戏项目——服务器端和客户端

在俄罗斯方块游戏项目中，为了能够实现游戏的网络对战，专门通过网络编程实现了服务器端和客户端。

通过 31.2 节的介绍可以知道俄罗斯方块游戏项目涉及的几个对象，以及这些对象涉及的属性和方法。那么如何实现这些对象呢？即如何设计这些对象的属性并实现这些对象的方法呢？本节将详细介绍如何表示出俄罗斯方块游戏项目中涉及的对象。

31.3.1 表示出俄罗斯方块游戏项目的服务器端

为了能够实现俄罗斯方块游戏的网络对战，俄罗斯方块游戏的服务器端必须能够不停地监听俄罗斯方块游戏的客户端，并且能够转发消息到俄罗斯方块游戏的客户端。为了实现上述功能，专门创建了一个名为 **MyServer** 的类，该类的具体内容如代码 31.7 所示，该类的 UML 如图 31.28 所示。

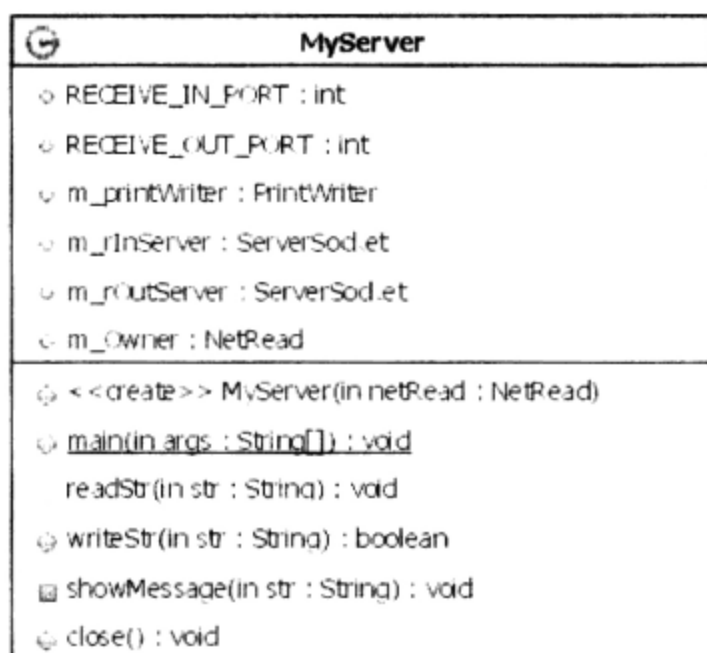


图 31.28 服务器类图

代码 31.7 服务器端后台程序：MyServer.java

```

public class MyServer {
    //成员变量
    final int RECEIVE_IN_PORT = 9090;           //接收信息端口号
    final int RECEIVE_OUT_PORT = 9091;          //信息发送端口号
    PrintWriter m_printWriter;                  //输出对象 m_printWriter
    ServerSocket m_rInServer, m_rOutServer;      //服务器端套接字
    NetRead m_Owner;                            //创建连接对象

    public MyServer(NetRead netRead) {          //构造函数
        m_Owner = netRead;                      //初始化对象 m_Owner
        //初始化对象 m_rInServer 和 m_rOutServer
        m_rInServer = null;
        m_rOutServer = null;
    }
  
```

```

Socket socketIn = null, socketOut = null; //用户请求的套接字
Thread readThread = null; //创建对象 readThread
try {
    //为对象 m_rInServer 和 m_rOutServer 赋值
    m_rInServer = new ServerSocket(RECEIVE_IN_PORT);
    m_rOutServer = new ServerSocket(RECEIVE_OUT_PORT);
    //显示相关信息
    showMessage("Welcome to the server!");
    //输出相关信息
    System.out.println("Welcome to the server!");
    System.out.println(new Date());
    System.out.println("The server is ready!");
    System.out.println("Port: " + RECEIVE_IN_PORT);
    System.out.println("Local machine's name:"
        + InetAddress.getLocalHost());
    //等待用户连接请求
    socketIn = m_rInServer.accept();
    socketOut = m_rOutServer.accept();
    //显示相关信息
    showMessage("Has been conected");
    //生成 serverThread 的实例
    readThread = new ReadThread(socketIn, this);
    //启动 serverThread 线程
    readThread.start();
    //获取对象 writer
    OutputStreamWriter writer = new OutputStreamWriter(socketOut
        .getOutputStream());
    m_printWriter = new PrintWriter(writer, true);
} catch (IOException e) {
    System.out.println(e.getMessage());
}
}

protected void readStr(String str) { //读取信息方法
    m_Owner.readStr(str);
}

public boolean writeStr(String str) { //写入信息方法
    if (m_printWriter == null)
        return false;

    try { //信息的写入
        m_printWriter.println(str);
        m_printWriter.flush();
    } catch (Exception e) {
        return false;
    }
    return true;
}

private void showMessage(String str) { //显示信息方法
    m_Owner.showMessage(str);
}

public void close() { //关闭方法
    try {
        //关闭相关对象
        m_rInServer.close();
        m_rOutServer.close();
        m_printWriter.close();
    } catch (Exception e) {
    }
}
}

```

【代码解析】

- ❑ 在上述代码中,首先创建了两个服务器端套接字对象 `m_rInServer` 和 `m_rOutServer`,并同时通过 `accept()`方法等待用户的要求,然后类对象 `readThread` 通过套接字 `socketIn` 不停地监听客户端,最后通过 `socketOut` 对象的输出流输出获取的信息。
- ❑ 为了便于开发,专门创建了读取数据的 `readStr()`方法和写数据的 `writeStr()`方法,同时还实现了关闭相关对象的 `close()`方法。

为了实现俄罗斯方块游戏的服务器端不停地监听客户端,专门创建了一个名为 `ReadThread` 的类,该类的具体内容如代码 31.8 所示,该类的 UML 如图 31.29 所示。

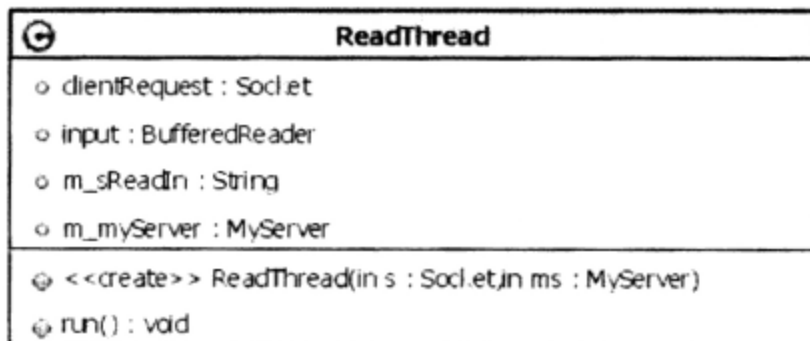


图 31.29 监听线程类图

代码 31.8 不停监听客户端线程类: `ReadThread.java`

```
class ReadThread extends Thread {
    //创建成员变量
    Socket clientRequest; //用户连接的通信套接字
    BufferedReader input; //输入流
    String m_sReadIn = ""; //读取字符串
    MyServer m_myServer; //创建 MyServer 对象
    public ReadThread(Socket s, MyServer ms) { //serverThread 的构造器
        //接收 receiveServer 传来的套接字
        this.clientRequest = s;
        this.m_myServer = ms;
        //创建输入流对象 reader
        InputStreamReader reader;
        try {
            //初始化输入
            reader = new InputStreamReader(clientRequest.getInputStream());
            input = new BufferedReader(reader);
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
    public void run() { //线程的执行方法
        boolean done = false; //创建布尔变量 done
        while (!done && m_myServer != null) {
            try {
                //接收客户端信息
                m_sReadIn = input.readLine();
                m_myServer.readStr(m_sReadIn);
            } catch (IOException e) {
                System.out.println(e.getMessage());
            }
        }
    }
}
```

```
try {
    clientRequest.close(); //关闭套接字
} catch (IOException e) {
    System.out.println(e.getMessage());
}
}
```

【代码解析】

在上述代码中，通过构造函数实现初始化方法，即首先通过输入服务器端套接字的 `getInputStream()` 方法获取对象 `reader`，然后再包装对象 `reader` 成对象 `input`，最后在线程的 `run()` 方法中通过循环不停地读取客户端发送过来的信息。

31.3.2 表示出俄罗斯方块游戏项目的客户端

为了能够实现俄罗斯方块游戏的网络对战，俄罗斯方块游戏的客户端必须能够不停地监听俄罗斯方块游戏的服务器端，并且能够转发消息到俄罗斯方块游戏的服务器端。为了实现上述功能，专门创建了一个名为 `MyClient` 的类，该类的具体内容如代码 31.9 所示，该类的 UML 如图 31.30 所示。

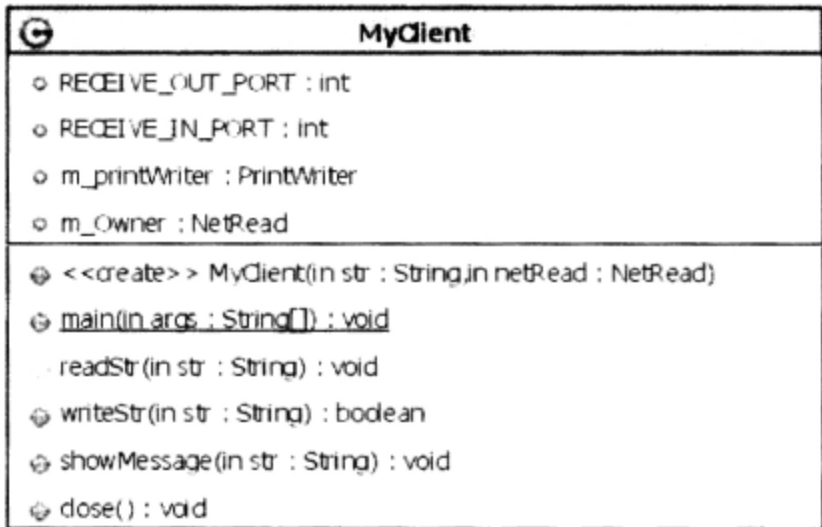


图 31.30 客户端类图

代码 31.9 客户端后台程序: MyClient.java

```
public class MyClient {
    //创建成员变量
    //相关端口成员变量
    final int RECEIVE_OUT_PORT = 9090;
    final int RECEIVE_IN_PORT = 9091;
    PrintWriter m_printWriter; //输出流对象
    NetRead m_Owner; //m_Owner 对象
    public MyClient(String str, NetRead netRead) { //构造函数
        //初始化成员变量
        m_Owner = netRead;
        Socket socketIn = null, socketOut = null;
        Thread readThread = null;
        try {
            //初始化对象 socketOut 和 socketIn
            socketOut = new Socket(str, RECEIVE_OUT_PORT);
```

```
        socketIn = new Socket(str, RECEIVE_IN_PORT);
        //生成 serverThread 的实例
        readThread = new ReadThread(socketIn, this);
        readThread.start(); //启动 serverThread 线程
        //获取输出流对象 writer
        OutputStreamWriter writer = new OutputStreamWriter(socketOut
            .getOutputStream());
        //初始化对象 m_printWriter
        m_printWriter = new PrintWriter(writer, true);
    } catch (Exception e) {
        System.out.println("can't connect to the server");
    }
}

protected void readStr(String str) { //读取字符串对象的方法
    System.out.println(str);
    m_Owner.readStr(str);
}

public boolean writeStr(String str) { //写入字符串对象的方法
    if(m_printWriter == null)
        return false;
    try{
        m_printWriter.println(str);
        m_printWriter.flush();
    } catch (Exception e) {
        return false;
    }
    return true;
}

public void showMessage(String str) { //用来显示不是对方发过来的消息
    m_Owner.showMessage(str);
}

public void close() { //关闭方法
    m_printWriter.close();
}
}
```

【代码解析】

- ❑ 在上述代码中，首先在构造函数中初始化端套接字对象 socketIn 和 socketOut，然后类对象 readThread 通过套接字 socketIn 不停地监听服务器端，最后通过 m_printWriter 对象的输出流输出获取的信息。
- ❑ 为了便于开发，专门创建了读取数据的 readStr()方法和写数据的 writeStr()方法，同时还实现了关闭相关对象的 close()方法。

为了实现俄罗斯方块游戏的客户端不停地监听服务器端，专门创建了一个名为 ReadThread 的类，该类的具体内容如代码 31.10 所示，该类的 UML 如图 31.31 所示。

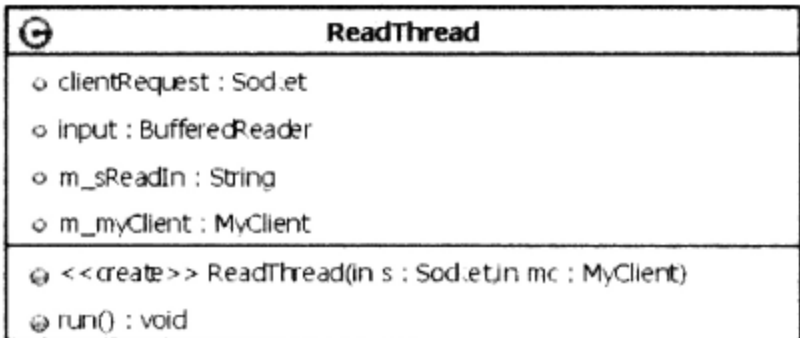


图 31.31 监听线程类图

代码 31.10 不停监听服务器端线程类: ReadThread.java

```

class ReadThread extends Thread {
    //创建成员变量
    Socket clientRequest;                //用户连接的通信套接字
    BufferedReader input;                //输入流对象
    String m_sReadIn = "";               //读进来的字符串
    MyClient m_myClient;                 //MyClient 类对象
    public ReadThread(Socket s, MyClient mc) { //serverThread 的构造器
        //初始化成员变量
        this.clientRequest = s;
        this.m_myClient = mc;
        InputStreamReader reader;
        try {                            //接收 receiveServer 传来的套接字
            //初始化输入、输出流
            reader = new InputStreamReader(clientRequest.getInputStream());
            input = new BufferedReader(reader);
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
    public void run() {                  //线程的执行方法
        boolean done = false;           //创建布尔型变量
        while (!done) {
            try {
                //接收服务器端的信息
                m_sReadIn = input.readLine();
                m_myClient.readStr(m_sReadIn);
                System.out.println(m_sReadIn);
            } catch (IOException e) {
                System.out.println(e.getMessage());
            }
            m_sReadIn = m_sReadIn.trim().toLowerCase();
            if (m_sReadIn == null || m_sReadIn.equals("quit"))
                done = true;
        }
        try {
            clientRequest.close();        //关闭套接字
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}

```

【代码解析】

在上述代码中,通过构造函数实现初始化方法,即首先通过套接字的 `getInputStream()` 方法获取对象 `reader`,然后再包装对象 `reader` 成 `input` 对象,最后在线程的 `run()`方法中通过循环不停地读取服务器端发送过来的信息。

31.4 俄罗斯方块游戏项目——游戏主界面

俄罗斯方块游戏项目中为了能够更方便玩家使用,设计了非常友好的游戏界面。这些界面不仅包含俄罗斯方块游戏的主界面 `TFrame`,而且还包括俄罗斯方块游戏的状态工具栏。

31.4.1 俄罗斯方块游戏的主界面

为了让罗斯方块游戏更具有友好性，该游戏的主界面必须简易、方便并能够吸引玩家的眼光。俄罗斯方块游戏设主界面的内容如代码 31.11 所示，该类的 UML 如图 31.32 所示。

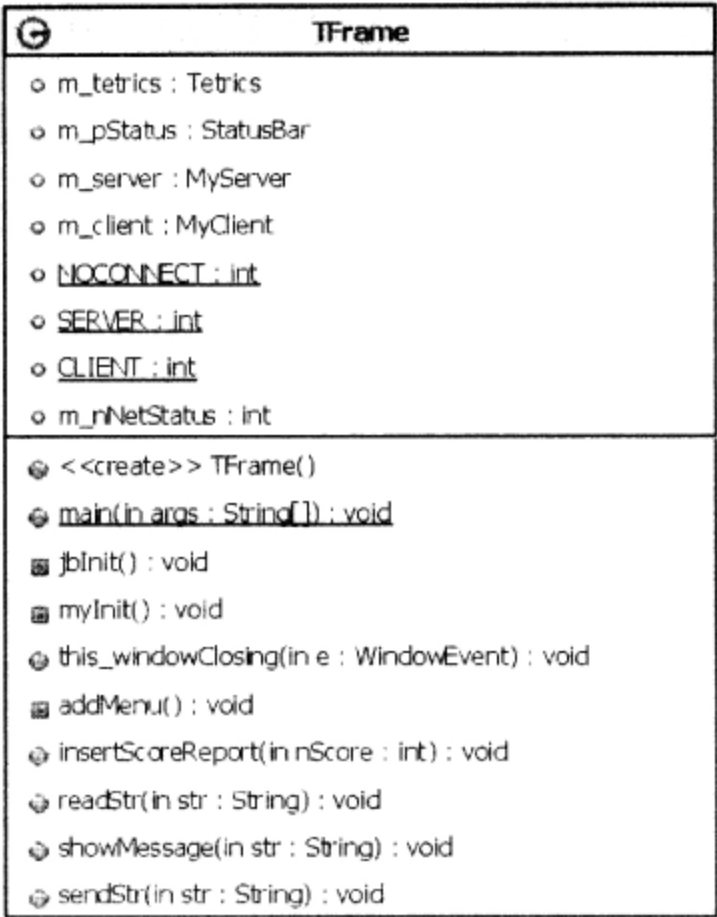


图 31.32 主界面类图

代码 31.11 俄罗斯方块游戏主界面的类：TFrame.java

```
public class TFrame extends Frame implements NetRead {
    //创建成员变量
    public Tetrics m_tetrics; //创建 m_tetrics 对象
    public StatusBar m_pStatus; //创建 m_pStatus 对象
    public MyServer m_server; //创建 m_server 对象
    public MyClient m_client; //创建 m_client 对象
    //定义表示现在网络状态的常量
    //NOCONNECT 表示单机运行
    //SERVE 表示这个游戏用作服务器
    //CLIENT 表示这个游戏用作客户端
    public static final int NOCONNECT = 0, SERVER = 1, CLIENT = 2;
    public int m_nNetStatus = NOCONNECT; //表示单机运行状态
    public TFrame() { //构造函数
        super();
        try {
            jbInit(); //调用 jbInit() 方法
            myInit(); //调用 myInit() 方法
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

public static void main(String[] args) { //主方法
    TFrame mframe = new TFrame(); //创建对象 mframe
    mframe.show(); //显示窗口
}

private void jbInit() throws Exception { //按钮监听器
    this.addWindowListener(new java.awt.event.WindowAdapter() {
        public void windowClosing(WindowEvent e) { //实现窗口关闭的方法
            this.windowClosing(e);
        }
    });
}

private void myInit() { //构造函数
    setSize(750, 580); //设置窗口大小
    setLocation(500, 400); //设置窗口位置
    addMenu(); //调用添加菜单的方法
    Panel gameP = new Panel(); //创建面板对象
    gameP.setLayout(null); //设置对象 gameP 的布局管理器
    m_tetrics = new Tetrics(this); //初始化对象 m_tetrics
    //设置 m_tetrics 的大小
    m_tetrics.setBounds(0, 0, Tetrics.WIDTH, Tetrics.HEIGHT);
    m_pStatus = new StatusBar(this); //初始化对象 m_pStatus
    //设置 m_pStatus 的大小
    m_pStatus.setBounds(0, Tetrics.HEIGHT, Tetrics.WIDTH,
        Tetrics.HEIGHT + 131);
    gameP.add(m_tetrics); //添加对象 m_tetrics 到 gameP 中
    gameP.add(m_pStatus); //添加对象 m_pStatus 到 gameP 中
    add(gameP); //添加对象 gameP 到 TFrame 中
}

void this_windowClosing(WindowEvent e) { //关闭窗口方法
    if (m_nNetStatus == SERVER)
        m_server.close();
    if (m_nNetStatus == CLIENT)
        m_client.close();
    System.exit(0); //退出窗口
}

private void addMenu() { //添加菜单方法
    MenuBar menuBar = new MenuBar(); //创建工具栏
    MenuListener menuListener = new MenuListener(this); //创建菜单监听器
    MenuShortcut ms = new MenuShortcut(KeyEvent.VK_S); //创建热键对象
    //创建 3 个菜单对象
    Menu menu1 = new Menu("游戏"); //“游戏”菜单
    Menu menu2 = new Menu("控制"); //“控制”菜单
    Menu menu3 = new Menu(""); //“关于”菜单
    //添加菜单对象到工具栏对象 menuBar 中
    menuBar.add(menu1);
    menuBar.add(menu2);
    menuBar.add(menu3);
    //创建和设置菜单对象 menu1
    MenuItem menuItem1_1 = new MenuItem("开始游戏");
    //创建菜单项对象 menuItem1_1
    MenuItem menuItem1_2 = new MenuItem("暂停游戏");
    //创建菜单项对象 menuItem1_2
    MenuItem menuItem1_3 = new MenuItem("结束游戏");
    //创建菜单项对象 menuItem1_3

```

```

MenuItem menuItem1_4 = new MenuItem("关闭");
//创建菜单项对象 menuItem1_4

//添加菜单项到菜单对象 menu1 中
menu1.add(menuItem1_1);
menu1.add(menuItem1_2);
menu1.add(menuItem1_3);
menu1.addSeparator(); //添加分割条
menu1.add(menuItem1_4);
//为菜单项添加监听器
menuItem1_1.addActionListener(menuListener);
menuItem1_2.addActionListener(menuListener);
menuItem1_3.addActionListener(menuListener);
... //省略部分代码
setMenuBar(menuBar); //设置工具栏
}
... //省略部分代码
}

```

【代码解析】

在上述代码中，为了便于开发专门创建了添加菜单的 `addMenu()` 方法、实现初始化功能的 `myInit()` 方法及实现关闭功能的 `jbInit()` 方法，最后在构造函数中调用 `jbInit()` 方法和 `myInit()` 方法来实现俄罗斯方块游戏的主界面。

在 `TFrame` 类中除了创建设计俄罗斯方块游戏的主界面方法外，还会创建一些其他方法，这些方法的具体内容如代码 31.12 所示。

代码 31.12 主窗口中涉及的一些方法

```

public void insertScoreReport(int nScore) { //实现添加分数到“分数报告”
    Score score = new Score(); //创建分数对象
    if (score.isScoreTop(nScore)) {
        Dialog d = new Dialog(this, "恭喜"); //创建对话框
        //创建对象 ius
        InsertURScoreP ius = new InsertURScoreP(nScore, d);
        d.add(ius); //添加对象 ius 到对话框中
        d.setSize(316, 231); //设置对话框大小
        d.setLocation(400, 310); //设置对话框位置
        d.show(); //显示对话框
    }
}

public void readStr(String str) { //从对方读来字符串时，调用这个方法
    //如果读来的数据是对方的状态信息
    if (str.startsWith("Status:")) {
        int[] nRivalField = new int[Tetris.m_nCols * (Tetris.m_nRows
            + 4)];
        str = str.substring(7, str.length());
        StringTokenizer st = new StringTokenizer(str, "|");
        int i = 0;
        try {
            while (st.hasMoreTokens()) {
                nRivalField[i] = Integer.parseInt(st.nextToken());
                i++;
            }
        } catch (Exception e) {
        }
        i = 0;
    }
}

```

```

        for (int col = 0; col < Tetrics.m_nCols; col++)
            for (int row = 0; row < Tetrics.m_nRows; row++) {
                m_tetrics.m_nRivalField[col][row] = nRivalField[i];
                i++;
            }
    }
    //如果读来的数据是对方消去一行的消息
    else if (str.equals("RemoveLine")) {
        m_tetrics.addRandomLine();
    }
    //如果读来的数据是对方开始游戏的信息
    else if (str.equals("StartGame")) {
        m_tetrics.start();
    }
    //如果读来的数据是对方暂停游戏的信息
    else if (str.equals("PauseGame")) {
        m_tetrics.pause();
    }
    //如果读来的数据是对方中止游戏的信息
    else if (str.equals("StopGame")) {
        m_tetrics.stop();
    }
    //如果读来的数据是对方要求改变游戏级别的信息
    else if (str.startsWith("Level:")) {
        str = str.substring(6, str.length());
        try {
            m_tetrics.setPlayLevel(Integer.parseInt(str));
        } catch (Exception e) {
        }
    }
    //如果读来的数据是对方与你谈话的内容
    else if (str.startsWith("Talk:")) {
        str = str.substring(5, str.length());
        m_pStatus.appendStr("对手: " + str + "\n");
    }
    //如果读来的数据是对方的游戏已经结束的信息
    else if (str.startsWith("GameOver:")) {
        str = str.substring(9, str.length());
        if (m_tetrics.m_bGameInPlay) {
            m_tetrics.m_bGameInPlay = false;
            m_tetrics.stop();
            sendStr("GameOver:" + m_tetrics.m_nTheScore);
        }
        try {
            int nRivalScore = Integer.parseInt(str);
            new GameOverD(this, m_tetrics.m_nTheScore, nRivalScore);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
//用来在 StatusBar 里显示现在的联网状况
public void showMessage(String str) {
    m_pStatus.appendStr(str + "\n");
}
//将信息发给对方
public void sendStr(String str) {
    switch (m_nNetStatus) {
        case SERVER:
            if (m_server != null)

```

```

        m_server.writeStr(str + "\n");
        break;
    case CLIENT:
        if (m_client != null)
            m_client.writeStr(str + "\n");
        break;
    }
}

```

【代码解析】

- 在上述代码中,首先实现了接口 NetRead 中的相关方法 readStr()和 showMessage(), readStr()方法主要用来处理接受到字符串,而 showMessage()方法主要用来显示联网状况。
- 为了便于开发,还创建了发送信息的方法 sendStr()和插入分数到 TOP10 分数对话框的方法 insertScoreReport()。

31.4.2 俄罗斯方块游戏的事件处理类

在俄罗斯方块游戏的主界面,当单击相应菜单时就会实现相应功能。处理菜单的事件方法为类 MenuListener,该类的具体内容如代码 31.13 所示,该类的 UML 如图 31.33 所示。

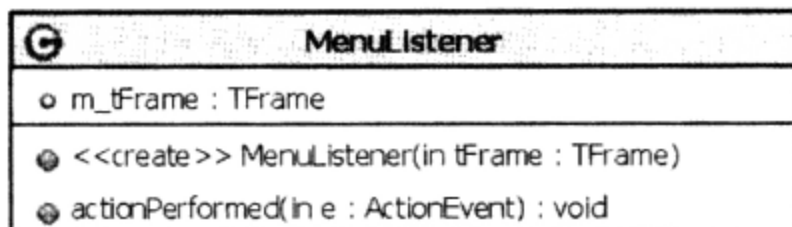


图 31.33 事件处理类图

代码 31.13 菜单的监听类: MenuListener.java

```

//菜单的监听事件
private class MenuListener implements ActionListener {
    TFrame m_tFrame; //创建 TFrame 类对象
    public MenuListener(TFrame tFrame) { //构造函数
        m_tFrame = tFrame; //初始化对象 m_tFrame
    }
    public void actionPerformed(ActionEvent e) {
        //获取命令的相关字符串
        String sCommand = e.getActionCommand();
        if (sCommand.equals("开始游戏")) { //开始玩游戏
            if (m_tFrame.m_nNetStatus == TFrame.CLIENT) {
                new WarningD(m_tFrame);
                return;
            }
            if (m_tFrame.m_nNetStatus == TFrame.SERVER) {
                m_tFrame.sendStr("StartGame");
                m_tFrame.sendStr("Level:"
                    + m_tFrame.m_tetris.getPlayLevel());
            }
            m_tetris.start(); //启动俄罗斯方块游戏
        } else if (sCommand.equals("结束游戏")) { //控制结束游戏
            if (m_tFrame.m_nNetStatus == TFrame.CLIENT) {

```



```

        new WarningD(m_tFrame);
        return;
    }
    if (m_tFrame.m_nNetStatus == TFrame.SERVER)
        m_tFrame.sendStr("StopGame");
    m_tetris.stop();
} else if (sCommand.equals("暂停游戏")) { //控制暂停游戏
    if (m_tFrame.m_nNetStatus == TFrame.CLIENT) {
        new WarningD(m_tFrame);
        return;
    }
    if (m_tFrame.m_nNetStatus == TFrame.SERVER)
        m_tFrame.sendStr("PauseGame");
    m_tetris.pause();
} else if (sCommand.equals("关闭游戏")) { //控制关闭游戏
    dispose();
} else if (sCommand.equals("设置级别")) { //实现设置级别
    if (m_tFrame.m_nNetStatus == TFrame.CLIENT) {
        new WarningD(m_tFrame);
        return;
    }
    Dialog d = new Dialog(m_tFrame, "设置级别");
    selectLevelP slp = new selectLevelP(m_tFrame, d);
    d.add(slp);
    d.setSize(252, 126);
    d.setLocation(400, 310);
    d.show();
} else if (sCommand.equals("查看分数")) { //实现查看分数
    //控制设置级别的代码加到这里
    Dialog d = new Dialog(m_tFrame, "分数报告-Top10");
    ScoreReportP srp = new ScoreReportP(d);
    d.add(srp);
    d.setSize(643, 310);
    d.setLocation(400, 310);
    d.show();
} else if (sCommand.equals("等待对方连接")) { //服务器功能, 等待连接
    if (m_tFrame.m_nNetStatus == TFrame.CLIENT) {
        new WarningD(m_tFrame);
        return;
    }
    m_tFrame.m_server = new MyServer(m_tFrame);
    System.out.println("开始监听");
    m_nNetStatus = SERVER;
} else if (sCommand.equals("连接对方")) { //客户端, 连接服务器
    Dialog d = new Dialog(m_tFrame, "连接对方");
    d.add(new ConnectP(d, m_tFrame));
    d.setSize(284, 131);
    d.setLocation(400, 310);
    d.show();
} else if (sCommand.equals("")) { //对话框
    Dialog d = new Dialog(m_tFrame, "");
    d.add(new AboutP(d));
    d.setSize(400, 320);
    d.setLocation(400, 310);
    d.show();
}
}
}
}

```

【代码解析】

在上述代码中，首先创建了一个 TFrame 类型成员变量 m_tFrame，同时初始化该成员变量。然后通过分支结构来决定各种菜单的处理方法：处理开始游戏菜单方法、处理结束游戏方法、处理暂停游戏菜单方法、处理关闭游戏菜单方法、处理设置级别菜单方法、处理查看分数菜单方法、处理等待对方连接菜单方法、处理连接对方菜单方法和处理菜单方法。

31.4.3 俄罗斯方块游戏的状态工具栏

玩家在具体玩俄罗斯方块游戏时，如果是网络对战状态，则需要交流、互通消息。为了能够更好地实现上述功能，专门创建了状态工具栏来实现玩家交流。俄罗斯方块游戏状态工具栏的内容如代码 31.14 所示，该类的 UML 如图 31.34 所示。

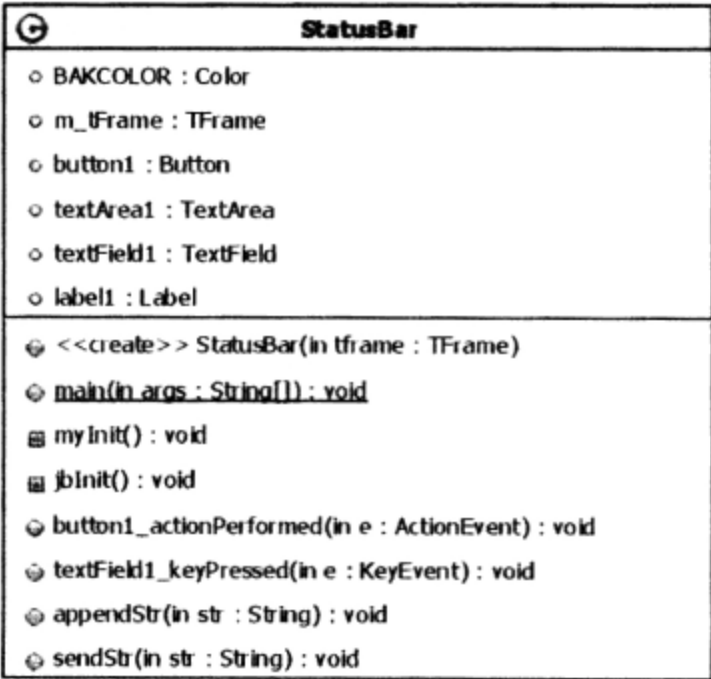


图 31.34 状态工具栏类图

代码 31.14 状态工具栏面板：StatusBar.java

```
public class StatusBar extends Panel {
    final Color BAKCOLOR = new Color(80, 123, 166);    //设置背景颜色
    TFrame m_tFrame;                                //创建对象 m_tFrame
    Button button1 = new Button();                    //创建按钮对象 button1
    //创建文本域对象
    TextArea textArea1 = new TextArea("", 3, 0, TextArea.SCROLLBARS_NONE);
    TextField textField1 = new TextField();            //创建文本框对象
    Label label1 = new Label();                        //创建标签对象
    public StatusBar(TFrame tframe) {                  //构造函数
        m_tFrame = tframe;                            //初始化对象 m_tFrame
        try {
            jbInit();                                  //调用 jbInit() 方法
            myInit();                                  //调用 myInit() 方法
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

public static void main(String[] args) { //主方法
    Frame f = new Frame(); //创建窗口对象
    //创建 StatusBar 类对象
    StatusBar statusBar1 = new StatusBar(new JFrame());
    f.add(statusBar1); //添加对象 statusBar1 到 f
    f.show(); //显示窗口
}
private void myInit() { //初始化功能
    this.setBackground(BAKCOLOR); //设置背景颜色
    textArea1.setBackground(BAKCOLOR); //设置文本域颜色
    textField1.setBackground(BAKCOLOR); //设置文本框颜色
}
private void jbInit() throws Exception { //初始化界面
    label1.setText("请输入现在要发送的消息"); //设置标签对象 label1 的内容
    //设置标签对象的大小
    label1.setBounds(new Rectangle(11, 71, 133, 18));
    //按钮对象添加监听器
    button1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
            button1_actionPerformed(e); //调用事件处理方法
        }
    });
    button1.setBackground(SystemColor.menu); //设置按钮的颜色
    button1.setLabel("发送"); //设置按钮的内容
    //设置按钮的大小
    button1.setBounds(new Rectangle(632, 68, 75, 29));
    this.setLayout(null); //设置布局管理器
    textArea1.setRows(3); //设置文本域的列数
    //设置文本域的大小
    textArea1.setBounds(new Rectangle(25, 6, 732, 57));
    //设置文本框的大小
    textField1.setBounds(new Rectangle(157, 71, 460, 22));
    //为文本框对象添加监听器
    textField1.addKeyListener(new java.awt.event.KeyAdapter() {
        public void keyPressed(KeyEvent e) {
            textField1_keyPressed(e); //调用事件处理方法
        }
    });
    this.setBackground(SystemColor.menu); //设置背景颜色
    this.add(textArea1, null); //添加对象 textArea1 到 StatusBar 中
    this.add(label1, null); //添加对象 label1 到 StatusBar 中
    this.add(button1, null); //添加对象 button1 到 StatusBar 中
    this.add(textField1, null); //添加对象 textField1 到 StatusBar 中
}
void button1_actionPerformed(ActionEvent e) { //按钮监听器
    String str = textField1.getText().trim(); //获取文本框中的内容
    textArea1.append(str + "\n"); //追加到文本域内容后
    sendStr("Talk:" + str); //实现信息发送
}
void textField1_keyPressed(KeyEvent e) { //键盘监听器
    if (e.getKeyCode() == KeyEvent.VK_ENTER) { //当为 Enter 键时
        String str = textField1.getText().trim(); //获取文本框中的内容
        textArea1.append(str + "\n"); //追加到文本域内容后
        sendStr("Talk:" + str); //实现信息发送
    }
}

```

```
    }
    public void appendStr(String str) {                                //实现信息追加
        textArea1.append(str);                                        //信息的追加
    }
    public void sendStr(String str) {                                    //实现发送信息的方法
        m_tFrame.sendStr(str);                                        //实现信息发送
    }
}
```

【代码解析】

在上述代码中，首先创建了实现初始化功能的方法 `jbInit()`和 `myInit()`，然后在构造函数中调用这两个方法。最后为了便于操作，还创建了字符串的操作 `appendStr()`和 `sendStr()`、按键处理方法 `textField1_keyPressed()`、动作处理方法 `button1_actionPerformed()`。

31.5 俄罗斯方块游戏项目——其他界面的设计

每当打开俄罗斯方块游戏项目时，玩家首先会看到该游戏的主界面和状态工具栏，选择相应菜单项就会出现相对应的面板对话框。本节主要讲解该项目的“关于”面板、连接对方面板和游戏结束面板。

31.5.1 “关于”面板

为了让罗斯方块游戏更具有友好性，玩家可以通过罗斯方块游戏的关于面板来获取该游戏的一些信息。“关于”面板的内容如代码 31.15 所示，该类的 UML 如图 31.35 所示。

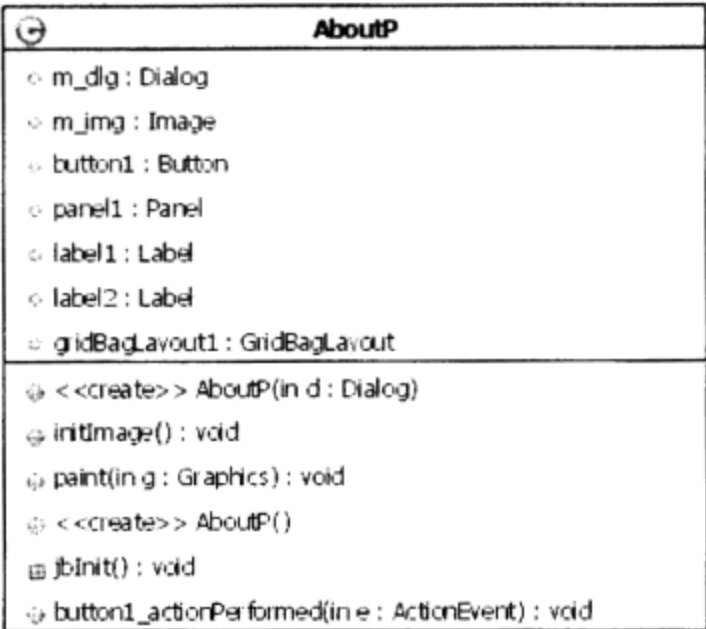


图 31.35 “关于”面板类图

代码 31.15 “关于”面板: AboutP.java

```
public class AboutP extends Panel {
    //创建成员变量
    Dialog m_dlg;                                //关于对话框对象 m_dlg
    Image m_img;                                //关于图像对象 m_img
}
```

```

Button button1 = new Button();           //关于按钮对象 button1
Panel panel1 = new Panel();              //关于面板对象 panel1
Label label1 = new Label();              //关于标签对象 label1
Label label2 = new Label();              //关于标签对象 label2
//关于布局对象 gridBagLayout1
GridBagLayout gridBagLayout1 = new GridBagLayout();
public AboutP() {                         //无参构造函数
    try {
        jbInit();                       //调用 jbInit() 方法
    } catch (Exception e) {
        e.printStackTrace();
    }
}
public AboutP(Dialog d) {                 //带参构造函数
    super();
    m_dlg = d;                           //初始化对象 m_dlg
    try {
        jbInit();                       //调用 jbInit() 方法
        initImage();                   //调用 initImage() 方法
    } catch (Exception e) {
    }
}
void initImage() {                       //初始化图像
    URL url = null;                      //创建对象 url
    try {
        //初始化对象 url
        url = Class.forName("TFrame").getResource("about.gif");
    } catch (Exception e) {
    }
    m_img = getToolkit().getImage(url);  //初始化对象 m_img
    MediaTracker mt = new MediaTracker(this); //创建对象 mt
    mt.addImage(m_img, 1);               //加载图像对象 m_img
    try {                                //查看图像的加载
        mt.wait();
        mt.checkAll();
    } catch (Exception e) {
    }
}
public void paint(Graphics g) {           //重写 paint() 方法
    g.drawImage(m_img, 0, 0, this);       //重绘图像
    super.paint(g);                      //绘制
}
private void jbInit() throws Exception { //初始化功能
    button1.setLabel("确定");             //设置按钮标识
    //设置按钮的位置
    button1.setBounds(new Rectangle(28, 65, 71, 29));
    //添加按钮事件
    button1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
            button1_actionPerformed(e);
            //调用 button1_actionPerformed(e) 方法
        }
    });
    this.setLayout(gridBagLayout1);       //设置对话框布局管理器
    panel1.setLayout(null);               //设置面板布局管理器
    label1.setText("俄罗斯方块游戏");     //设置 label1 标签的文本
    label2.setText("作者: 常建功 (cjgong)"); //设置 label2 标签的文本

```

```

//设置 label2 标签的位置
label2.setBounds(new Rectangle(26, 31, 73, 18));
//添加 label1 标签到对话框对象中
this.add(label1, new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0,
    GridBagConstraints.WEST, GridBagConstraints.NONE, new
    Insets(
        149, 142, 133, 166), 8, 0));
//添加 panel1 标签到对话框对象中
this.add(panel1, new GridBagConstraints(0, 0, 1, 1, 1.0, 1.0,
    GridBagConstraints.CENTER, GridBagConstraints.BOTH, new
    Insets(
        142, 133, 56, 147), 119, 101));
panel1.add(label2, null);           //添加对象 label2 到 panel1 标签中
panel1.add(button1, null);         //添加对象 button1 到 panel1 标签中
}
void button1_actionPerformed(ActionEvent e) {           //事件处理方法
    m_dlg.dispose();                                     //对话框消失
}
}

```

【代码解析】

在上述代码中，由于“关于对话框”中的内容是面板对象 AboutP，所以继承了 Panel 类。在该对象中包含了一个标签对象 label1 和一个面板对象 panel1。在面板对象 panel1 中，包含了 label2 标签对象和 button1 按钮对象，该面板的布局如图 31.36 所示。

31.5.2 连接对方面板

在玩俄罗斯方块游戏的网络对战时，客户端玩家需要连接到服务器端玩家，那如何实现连接呢？这就需要连接对方面板，该面板的内容如代码 31.16 所示，该类的 UML 如图 31.37 所示。

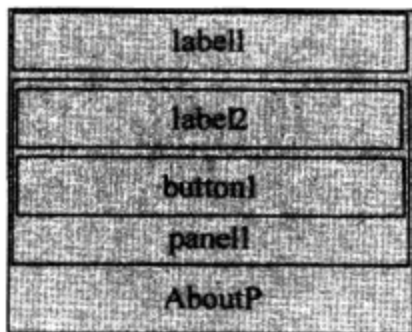


图 31.36 布局

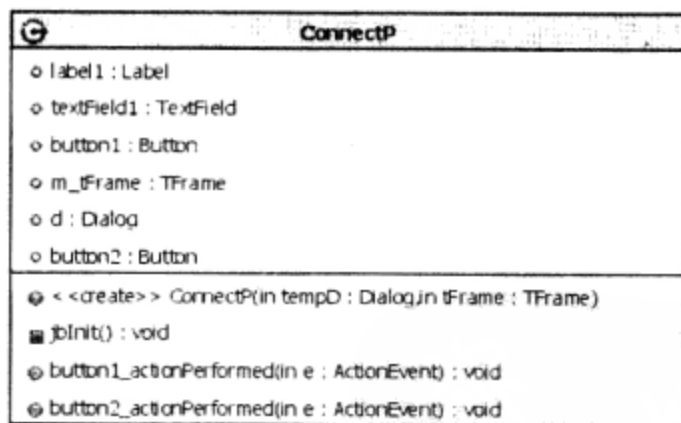


图 31.37 连接对方面板类图

代码 31.16 连接对方面板：ConnectP.java

```

public class ConnectP extends Panel {
    //创建成员变量
    Label label1 = new Label();           //标签对象 label1
    TextField textField1 = new TextField(); //文本对象 textField1
    Button button1 = new Button();         //按钮对象 button1
    TFrame m_tFrame;                      //窗口对象 m_tFrame
    Dialog d;                             //对话框对象 d
}

```



```

Button button2 = new Button(); //按钮对象 button2
public ConnectP(Dialog tempD, TFrame tFrame) { //带参数构造函数
    super(); //调用构造函数
    d = tempD; //初始化对象 d
    m_tFrame = tFrame; //初始化对象 m_tFrame
    try {
        jbInit(); //调用方法 jbInit()
    } catch (Exception e) {
        e.printStackTrace();
    }
}
private void jbInit() throws Exception { //实现初始化方法
    label1.setText("要连接机器的 IP 或名字"); //设置标签对象 label1 的文本
    //设置标签对象 label1 的位置
    label1.setBounds(new Rectangle(22, 20, 137, 18));
    this.setLayout(null); //设置布局管理器
    //设置文本对象的大小
    textField1.setBounds(new Rectangle(169, 19, 83, 22));
    button1.setLabel("确 定"); //设置按钮对象的内容
    //设置按钮对象的大小
    button1.setBounds(new Rectangle(56, 66, 75, 29));
    //为按钮对象添加监听器
    button1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
            button1_actionPerformed(e); //调用事件处理方法
        }
    });
    button2.setLabel("取 消"); //设置按钮对象的内容
    //设置按钮对象的大小
    button2.setBounds(new Rectangle(157, 66, 75, 29));
    button2.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
            button2_actionPerformed(e); //调用事件处理方法
        }
    });
    this.add(label1, null); //添加对象 label1 到对象 ConnectP 中
    this.add(textField1, null); //添加对象 textField1 到对象 ConnectP 中
    this.add(button1, null); //添加对象 button1 到对象 ConnectP 中
    this.add(button2, null); //添加对象 button2 到对象 ConnectP 中
}
void button1_actionPerformed(ActionEvent e) { //实现事件处理方法
    m_tFrame.m_client = new MyClient(textField1.getText().trim(),
    m_tFrame);
    System.out.println("连接对方"); //输出相应信息
    m_tFrame.m_nNetStatus = TFrame.CLIENT;
    d.dispose(); //对话框消失
}
void button2_actionPerformed(ActionEvent e) { //实现事件处理方法
    d.dispose(); //对话框消失
}
}

```

【代码解析】

在上述代码中,俄罗斯方块游戏的“连接对方”面板是 infoPanel 对象,在该对象中包含了一个标题标签对象 infoTitleLabel 和一个文本域对象 infoTextArea。该面板的布局如图

31.38 所示。

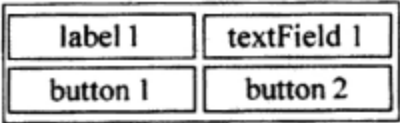


图 31.38 布局

31.5.3 分数报告面板

当网络对战游戏结束后，如果赢家的得分比较高属于 TOP10 范围，则会出现插入分数记录面板。在该面板中只需要输入玩家的名字，就会把包含“玩家名字+分数”的记录插入到分数报告面板里。插入分数记录面板的内容如代码 31.17 所示，其 UML 如图 31.39 所示。

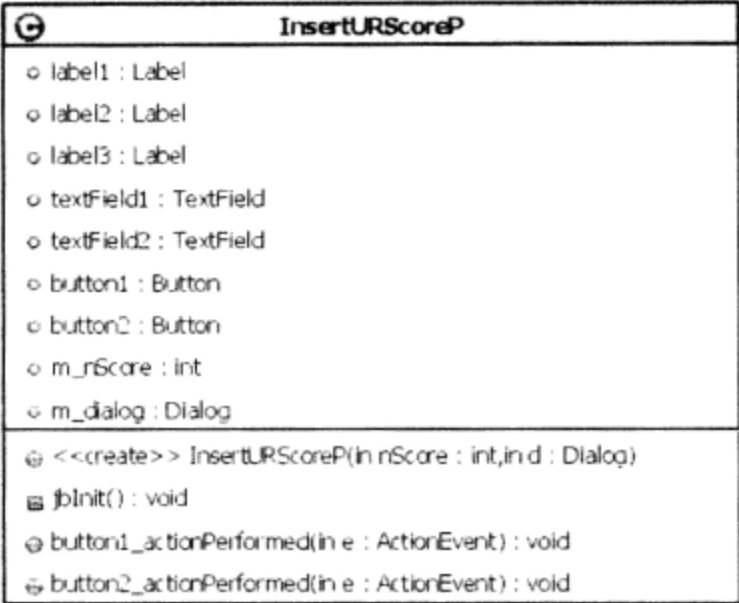


图 31.39 插入分数记录面板类图

代码 31.17 警告对话框：InsertURScoreP.java

```
public class InsertURScoreP extends Panel {
    //创建成员变量
    Label label1 = new Label(); //对象 label1
    Label label2 = new Label(); //对象 label2
    Label label3 = new Label(); //对象 label3
    TextField textField1 = new TextField(); //对象 textField1
    TextField textField2 = new TextField(); //对象 textField2
    Button button1 = new Button(); //对象 button1
    Button button2 = new Button(); //对象 button2
    int m_nScore; //玩家的得分
    Dialog m_dialog; //对话框
    public InsertURScoreP(int nScore, Dialog d) { //构造函数
        m_dialog = d; //初始化对话框对象 m_dialog
        m_nScore = nScore; //初始化对象 m_nScore
        try {
            jbInit(); //调用 jbInit() 方法
            textField2.setText("" + nScore); //设置文本框内容
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void jbInit() throws Exception {
        label1.setText("请输入你的名字"); //设置对象 label1 的内容
```

```

//设置对象 label1 的大小
label1.setBounds(new Rectangle(52, 64, 116, 18));
label2.setText("你的得分为"); //设置对象 label2 的内容
//设置对象 label2 的大小
label2.setBounds(new Rectangle(52, 118, 83, 18));
this.setLayout(null); //设置布局管理器
label3.setForeground(Color.red); //设置颜色
//设置对象 label3 的内容
label3.setText("祝贺你! 你的得分已经进入 Top10 的行列");
//设置对象 label3 的大小
label3.setBounds(new Rectangle(49, 25, 220, 18));
textField1.setColumns(10); //设置 textField1 的长度
//设置 textField1 的大小
textField1.setBounds(new Rectangle(186, 63, 56, 22));
textField2.setColumns(10); //设置 textField2 的长度
textField2.setEditable(false); //设置 textField2 不可编辑
//设置 textField2 的大小
textField2.setBounds(new Rectangle(186, 120, 56, 22));
button1.setLabel("确定"); //设置按钮 button1 的文本
button1.setBounds(new Rectangle(59, 165, 75, 29)); //设置按钮 button1 的大小

//为按钮 button1 注册事件监听器
button1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        button1_actionPerformed(e); //调用事件处理方法
    }
});
button2.setLabel("取消"); //设置按钮 button2 的文本
//设置按钮 button2 的大小
button2.setBounds(new Rectangle(160, 164, 75, 29));
//为按钮 button2 注册事件监听器
button2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        button2_actionPerformed(e); //调用事件处理方法
    }
});
this.add(label3, null); //添加对象 label3 到 InsertURScoreP 中
this.add(textField1, null); //添加对象 textField1 到 InsertURScoreP 中
this.add(label1, null); //添加对象 label1 到 InsertURScoreP 中
this.add(label2, null); //添加对象 label2 到 InsertURScoreP 中
this.add(textField2, null); //添加对象 textField2 到 InsertURScoreP 中
this.add(button1, null); //添加对象 button1 到 InsertURScoreP 中
this.add(button2, null); //添加对象 button2 到 InsertURScoreP 中
}

void button1_actionPerformed(ActionEvent e) { //创建事件处理方法
    Score score = new Score(); //创建 Score 类对象
    //调用 insertNameScore() 方法
    score.insertNameScore(textField1.getText().trim(), m_nScore);
    m_dialog.dispose(); //对话框消失
}

void button2_actionPerformed(ActionEvent e) { //创建事件处理方法
    m_dialog.dispose(); //对话框消失
}
}

```

【代码解析】

在上述代码中，俄罗斯方块游戏的“插入分数记录面板”是 InsertURScoreP 对象，在该对象中包含了 3 个面板对象 label3、label1 和 label2、两个输入文本框对象 textField1 和 textField2，以及两个按钮对象 button1 和 button2，该面板的布局如图 31.40 所示。

对于一款游戏来说，分数排名是非常重要的。俄罗斯方块游戏同样具有实现排名功能的分数报告面板，通过该面板，游戏玩家可以查看 TOP10 范围的分数。分数报告面板的内容如代码 31.18 所示，其 UML 如图 31.41 所示。

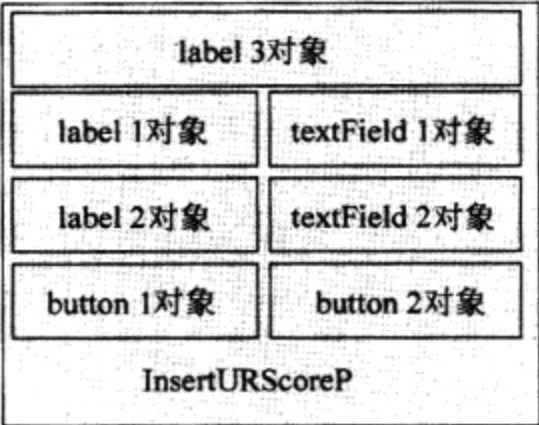


图 31.40 布局

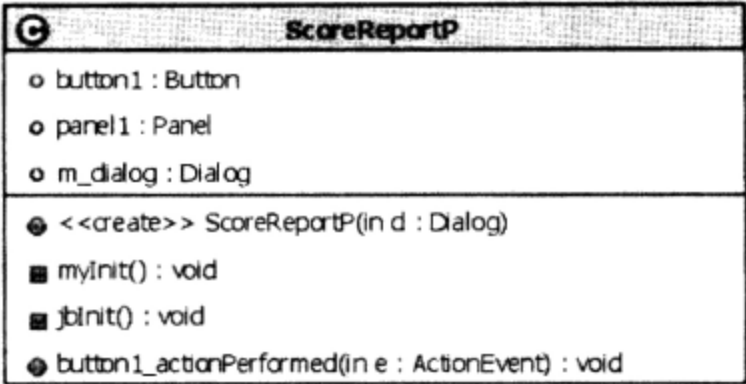


图 31.41 分数报告面板类图

代码 31.18 分数报告面板：ScoreReportP.java

```
public class ScoreReportP extends Panel {
    //创建成员变量
    Button button1 = new Button(); //按钮 button1 对象
    Panel panel1 = new Panel(); //面板 panel1 对象
    Dialog m_dialog; //对话框 m_dialog 对象
    public ScoreReportP(Dialog d) { //构造函数
        m_dialog = d; //初始化对象 m_dialog
        try {
            jbInit(); //调用 jbInit() 方法
            myInit(); //调用 myInit() 方法
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    private void myInit() throws Exception {
        TextField[] m_textScore = new TextField[10]; //分数输入文本框数组
        TextField[] m_textName = new TextField[10]; //姓名输入文本框数组
        Score score = new Score(); //创建 Score 类对象
        int[] nScore = score.getScore(); //初始化分数值数组
        String[] sName = score.getName(); //初始化姓名数组
        //通过循环来实现布局
        for (int i = 0; i < sName.length; i++) {
            Panel p = new Panel(); //创建面板对象 p
            System.out.println(sName[i].trim()); //输出相应信息
            //初始化数组对象 m_textScore 和 m_textName
            m_textName[i] = new TextField(sName[i], 10);
            m_textScore[i] = new TextField("" + nScore[i], 8);
            m_textName[i].setEditable(false); //设置不可编辑
            m_textScore[i].setEditable(false); //设置不可编辑
        }
    }
}
```

```
//创建面板对象 l1 和 l2
Label l1 = new Label("玩家");
Label l2 = new Label("分数");
p.add(l1); //添加对象 l1 到对象 p 中
p.add(m_textName[i]); //添加对象 m_textName[i] 到对象 p 中
p.add(l2); //添加对象 l2 到对象 p 中
p.add(m_textScore[i]); //添加对象 m_textScore[i]到对象 p 中
panel1.add(p); //添加对象 p 到对象 panel1 中
}
}
private void jbInit() throws Exception {
    button1.setLabel("确定"); //设置按钮文本
    //设置按钮大小
    button1.setBounds(new Rectangle(285, 234, 75, 29));
    //为按钮添加事件监听器
    button1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
            button1_actionPerformed(e); //调用事件处理方法
        }
    });
    this.setLayout(null); //设置布局管理器
    //设置面板对象 panel1 的大小
    panel1.setBounds(new Rectangle(24, 16, 596, 207));
    this.add(panel1, null); //添加对象 panel1 到对象 ScoreReportP 中
    this.add(button1, null); //添加对象 button1 到对象 ScoreReportP 中
}
void button1_actionPerformed(ActionEvent e) { //事件处理方法
    m_dialog.dispose(); //对话框消失
}
}
```

【代码解析】

在上述代码中，俄罗斯方块游戏的“分数报告”面板是 ScoreReportP 对象，在该对象中包含了一个面板对象 panel1 和一个按钮对象 button1。在面板对象 panel1 中包含了另一个面板 p，而在面板 p 中则包含 l1、l2、m_textScore 和 m_textName 对象，其中 m_textScore 和 m_textName 对象都属于 TextField 类型，该面板的布局如图 31.42 所示。

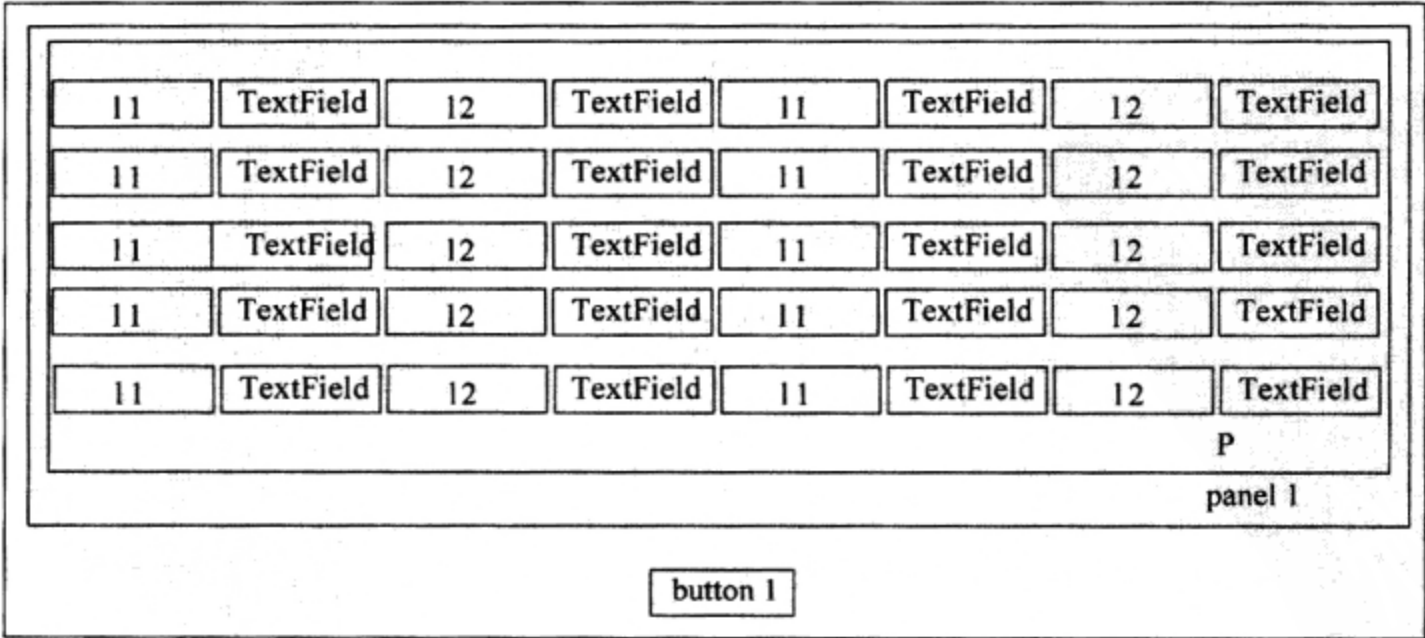


图 31.42 布局

31.5.4 设置级别面板

在玩俄罗斯方块游戏之前，玩家可以通过级别面板设置该游戏的难易程度——游戏的级别。在设置级别面板中只需要移动滚动条，就会改变并显示出新的游戏级别。设置级别面板的内容如代码 31.19 所示，其 UML 如图 31.43 所示。

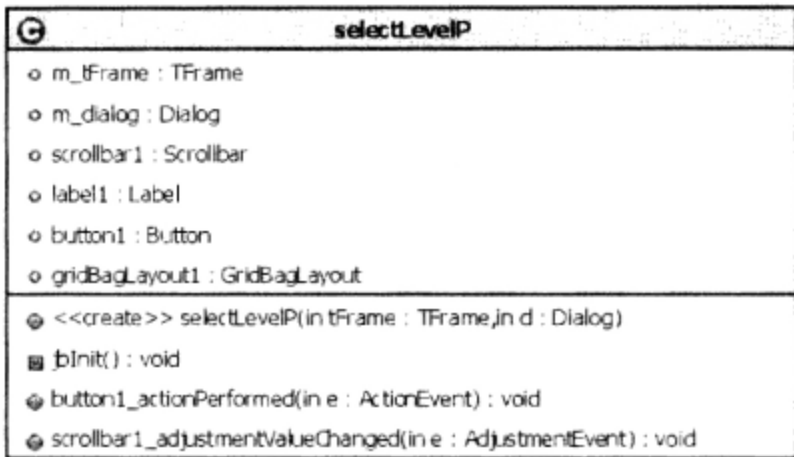


图 31.43 设置级别面板类图

代码 31.19 设置级别面板：selectLevelP.java

```
public class selectLevelP extends Panel {
    //创建成员变量
    TFrame m_tFrame; //TFrame 类对象
    Dialog m_dialog; //Dialog 类对象
    Scrollbar scrollbar1 = new Scrollbar(); //滚动条对象 scrollbar1
    Label label1 = new Label(); //标签对象 label1
    Button button1 = new Button(); //按钮对象 button1
    GridBagLayout gridBagLayout1 = new GridBagLayout(); //布局对象 gridBagLayout1

    public selectLevelP(TFrame tFrame, Dialog d) { //构造函数
        //初始化成员对象
        m_tFrame = tFrame;
        m_dialog = d;
        try {
            jbInit(); //调用方法 jbInit()
            //输出相应信息
            System.out.println(tFrame.m_tetris.getPlayLevel());
            //设置滚动条的值
            scrollbar1.setValue(tFrame.m_tetris.getPlayLevel());
            //设置标签文本
            label1.setText("你选择的的等级: " + scrollbar1.getValue());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void jbInit() throws Exception {
        this.setLayout(gridBagLayout1); //设置布局管理器
        scrollbar1.setMaximum(15); //设置最大值
        scrollbar1.setMinimum(1); //设置最小值
        scrollbar1.setOrientation(0); //设置滚动条方向
        scrollbar1.setPageIncrement(1); //设置滚动条的单位增量
    }
}
```



```

scrollbar1.setValue(5); //设置滚动条的默认值
scrollbar1.setVisibleAmount(1); //设置滚动条的可视量
scrollbar1
    .addAdjustmentListener(new java.awt.event. Adjustment-
        Listener() {
            public void adjustmentValueChanged(AdjustmentEvent e) {
                //调用事件处理方法
                scrollbar1_adjustmentValueChanged(e);
            }
        });
label1.setText("你选择的的等级: 5"); //设置标签的内容
button1.setLabel("确定"); //设置按钮内容
button1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        button1_actionPerformed(e); //调用事件处理方法
    }
});
//添加 scrollbar1 对象到 selectLevelP 中
this.add(scrollbar1, new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0,
    GridBagConstraints.CENTER, GridBagConstraints.BOTH, new
    Insets(
        14, 39, 0, 45), 118, 3));
//添加 button1 对象到 selectLevelP 中
this.add(button1, new GridBagConstraints(0, 2, 1, 1, 0.0, 0.0,
    GridBagConstraints.CENTER, GridBagConstraints.NONE, new
    Insets(
        0, 84, 10, 85), 22, -3));
//添加 label1 对象到 selectLevelP 中
this.add(label1, new GridBagConstraints(0, 1, 1, 1, 0.0, 0.0,
    GridBagConstraints.WEST, GridBagConstraints.NONE, new
    Insets(
        9, 50, 0, 62), 35, 8));
}
void button1_actionPerformed(ActionEvent e) { //事件处理方法
    m_dialog.dispose(); //对话框消失
}
void scrollbar1_adjustmentValueChanged(AdjustmentEvent e) { // 事件处理方法
    //设置标签内容
    label1.setText("你选择的等级: " + scrollbar1.getValue());
    int nLevel = scrollbar1.getValue(); //获取滚动条的值
    m_tFrame.m_tetris.setPlayLevel(nLevel);
    if (m_tFrame.m_nNetStatus == TFrame.SERVER)
        m_tFrame.sendStr("Level:" + nLevel);
}
}

```

【代码解析】

在上述代码中, 俄罗斯方块游戏的“设置级别”面板是 selectLevelP 对象, 在该对象中包含了 3 个对象, 分别是 scrollbar1 对象、label1 对象和 button1 对象, 该面板的布局如图 31.44 所示。

31.5.5 警告面板和对话框

当玩俄罗斯方块游戏的网络对战时, 对于客户端游戏玩家来说, 有些菜单选项是不可使用的。玩家如果单击了这些游戏选项, 则会出现包含警告面板的警告对话框, 警告面板

的内容如代码 31.20 所示，其 UML 如图 31.45 所示。



图 31.44 布局

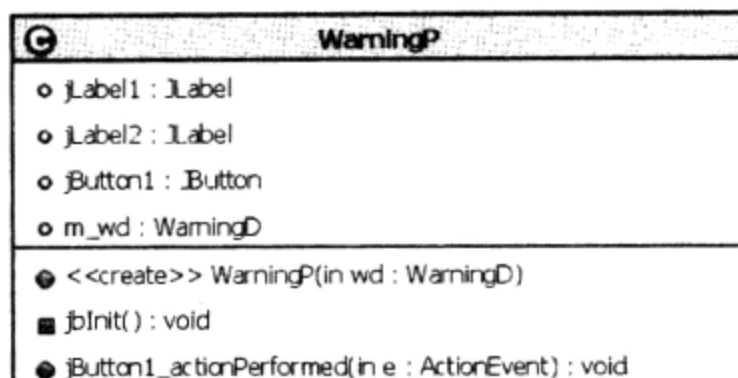


图 31.45 警告面板类图

代码 31.20 警告面板: WarningP.java

```
public class WarningP extends Panel {
    //创建成员变量
    JLabel jLabel1 = new JLabel();
    JLabel jLabel2 = new JLabel();
    JButton jButton1 = new JButton();
    WarningD m_wd;
    public WarningP(WarningD wd) {
        m_wd = wd;
        try {
            jbInit();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    private void jbInit() throws Exception {
        jLabel1.setForeground(Color.red); //设置标签对象 jLabel1 的字体颜色
        //设置标签对象文本
        jLabel1.setText("你现在是在客户端运行的一个网络游戏。");
        jLabel1.setBounds(new Rectangle(37, 15, 247, 31));
        //设置标签对象的大小
        this.setLayout(null); //设置布局管理器
        jLabel2.setForeground(Color.red); //设置标签对象 jLabel2 的字体颜色
        jLabel2.setText("你没有权限来运行这个命令!"); //设置标签对象文本
        jLabel2.setBounds(new Rectangle(59, 51, 189, 31));
        //设置标签对象的大小
        jButton1.setText("确定"); //设置按钮对象的文本
        jButton1.setBounds(new Rectangle(99, 87, 79, 29));
        //设置按钮对象的大小
        //设置按钮对象的事件监听器
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(ActionEvent e) {
                jButton1_actionPerformed(e); //调用事件处理方法
            }
        });
        //添加相应对象到 WarningP 对象中
        this.add(jButton1, null);
        this.add(jLabel1, null);
        this.add(jLabel2, null);
    }
    void jButton1_actionPerformed(ActionEvent e) { //事件处理方法
```

```
        m_wd.dispose(); //对话框消失
    }
}
```

【代码解析】

在上述代码中，俄罗斯方块游戏的警告面板是 WarningP 对象，在该对象中包含 3 个对象，即 Label1 对象、Label2 对象和 Button1 对象，该面板的布局如图 31.46 所示。

在具体玩俄罗斯方块游戏网络对战时，当客户端玩家选择不能使用的菜单选项时显示的是警告对话框，而不是警告面板。游戏警告对话框其实很简单，只是添加了游戏警告面板，对话框的内容如代码 31.21 所示，其 UML 如图 31.47 所示。

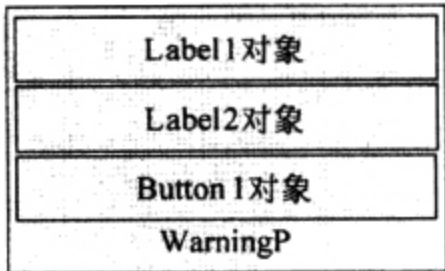


图 31.46 布局

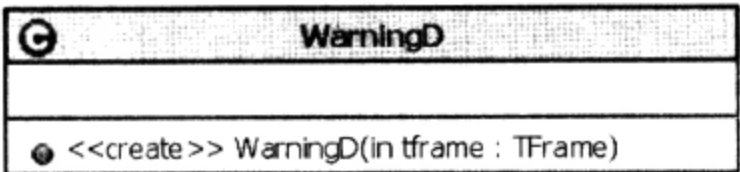


图 31.47 警告对话框

代码 31.21 警告对话框：WarningD.java

```
public class WarningD extends Dialog {
    public WarningD(TFrame tframe) { //构造函数
        super(tframe);
        add(new WarningP(this)); //添加警告对话框
        setTitle("警告"); //设置对话框标题
        setSize(264, 154); //设置对话框大小
        show(); //显示对话框
    }
}
```

【代码解析】

在上述代码中，主要在构造函数中实现警告对话框的初始化功能，首先设置该对话框的标题和大小，然后添加警告面板到该对话框中，最后通过 show()方法显示该对话框。

31.5.6 游戏结束面板和对话框

当网络对战结束游戏后，客户端和服务端就会出现包含结束面板的结束对话框。在结束面板中不仅会显示客户端和服务端的成绩，而且还会显示出“输赢”情况。结束面板的内容如代码 31.22 所示，其 UML 如图 31.48 所示。

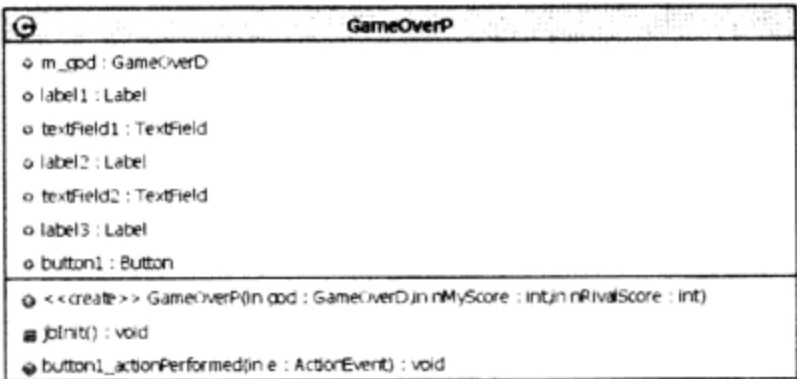


图 31.48 游戏结束面板类图

代码 31.22 游戏结束面板: GameOverP.java

```

public class GameOverP extends Panel {
    //创建成员变量
    GameOverD m_god; //m_god 对象
    Label label1 = new Label(); //label1 对象
    TextField textField1 = new TextField(); //textField1 对象
    Label label2 = new Label(); //label2 对象
    TextField textField2 = new TextField(); //textField2 对象
    Label label3 = new Label(); //label3 对象
    Button button1 = new Button(); //button1 对象
    //构造函数
    public GameOverP(GameOverD god, int nMyScore, int nRivalScore) {
        m_god = god; //初始化对象 m_god
        try {
            jbInit(); //调用 jbInit() 方法
            textField1.setText("" + nMyScore); //设置文本
            textField2.setText("" + nRivalScore); //设置文本
            if (nRivalScore > nMyScore) //判断输赢
                label3.setText("你输了!!!");
            else
                label3.setText("你赢了!!!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    private void jbInit() throws Exception { //初始化方法
        label1.setText("你的分数"); //设置 label1 文本
        //设置 label1 的大小
        label1.setBounds(new Rectangle(36, 29, 73, 18));
        this.setLayout(null); //设置布局管理器
        textField1.setEditable(false); //设置 textField1 不可编辑
        //设置 textField1 的大小
        textField1.setBounds(new Rectangle(145, 29, 56, 22));
        label2.setText("对手的分"); //设置 label2 文本
        label2.setBounds(new Rectangle(40, 85, 81, 18)); //设置 label2 大小
        textField2.setEditable(false); //设置 textField2 不可编辑
        //设置 textField2 的大小
        textField2.setBounds(new Rectangle(145, 82, 56, 22));
        label3.setAlignment(1); //设置 label3 的对齐方式
        label3.setForeground(Color.red); //设置 label3 的颜色
        label3.setText("label3"); //设置 label3 的文本
        //设置 label3 的大小
        label3.setBounds(new Rectangle(24, 125, 195, 18));
        button1.setLabel("确 定"); //设置 button1 的内容
        button1.setBounds(new Rectangle(87, 157, 75, 29)); //设置 button1 的大小
        //为按钮 button1 注册监听器
        button1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(ActionEvent e) {
                button1_actionPerformed(e); //调用事件处理方法
            }
        });
        this.add(label2, null); //添加 label2 到 GameOverP 中
        this.add(label1, null); //添加 label1 到 GameOverP 中
    }
}

```

```

        this.add(textField1, null);           //添加 textField1 到 GameOverP 中
        this.add(textField2, null);           //添加 textField2 到 GameOverP 中
        this.add(label3, null);               //添加 label3 到 GameOverP 中
        this.add(button1, null);              //添加 button1 到 GameOverP 中
    }
    void button1_actionPerformed(ActionEvent e) { //实现事件处理方法
        m_god.dispose();                        //对话框消失
    }
}

```

【代码解析】

在上述代码中，俄罗斯方块游戏的结束面板是 GameOverP 对象，在该对象中包含了一个标题标签对象 infoTitleLabel 和一个文本域对象 infoTextArea。该面板的布局如图 31.49 所示。

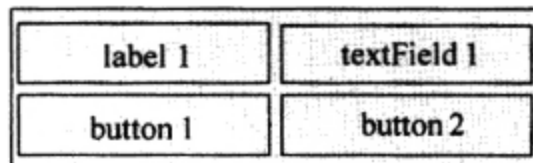


图 31.49 布局

当网络对战游戏结束后，显示的是游戏结束对话框，而不是游戏结束面板。游戏结束对话框其实很简单，只是添加了游戏结束面板，对话框的内容如代码 31.23 所示，其 UML 如图 31.50 所示。

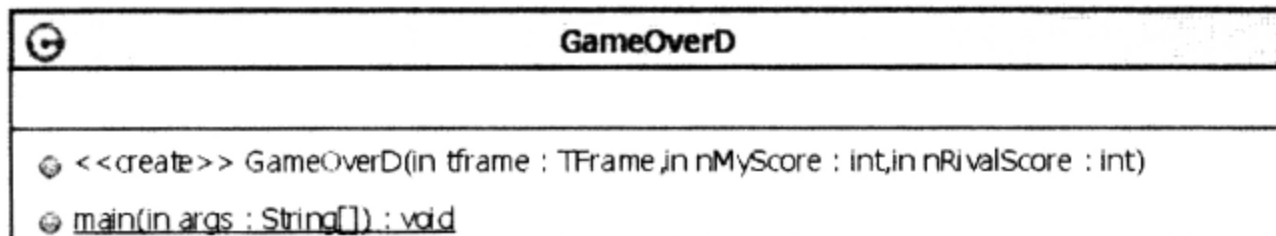


图 31.50 游戏结束对话框类图

代码 31.23 游戏结束对话框: GameOverD.java

```

public class GameOverD extends Dialog {
    //构造函数
    public GameOverD(TFrame tframe, int nMyScore, int nRivalScore) {
        super(tframe);           //调用构造函数
        setTitle("分数报告");     //设置标题
        setSize(251, 223);        //设置大小
        setLocation(400, 310);    //设置对话框位置
        //添加对话框面板到对话框中
        add(new GameOverP(this, nMyScore, nRivalScore));
        show();
    }
    public static void main(String args[]) { //主方法
        new GameOverD(null, 1, 2);          //创建对象 GameOverD
    }
}

```

【代码解析】

在上述代码中，主要在构造函数中实现游戏结束对话框的初始化功能，首先设置该对话框的标题和大小，然后添加游戏结束面板到该对话框中，最后通过 show()方法显示该对话框。

31.6 小 结

本章主要介绍了一个完整的实现网络对战功能的俄罗斯方块游戏项目，该项目与流行的俄罗斯方块游戏相比虽然游戏界面比较粗糙，但是却实现了该游戏的所有功能，同时完全基于 Java 语言构建而成。

在具体实现俄罗斯方块游戏项目时，由于需要实现网络对战功能，所以除了实现该游戏的基本功能外，还通过 Java 语言的网络编程知识实现了该游戏服务器端和客户端的连接。“俄罗斯方块游戏”项目通过 MVC 模型来实现，由于整个游戏比较小，所以 M 和 C 层放在一个包（tetris）里，同时由于该项目主要讲解 Java 语言的网络编程，所以分成了两个包，server 包主要用来实现服务器端连接客户端，client 包主要用来实现客户端连接服务器端。

第 32 章 图书管理系统项目 (GUI+Oracle 数据库)

“图书管理系统”项目是典型的信息管理系统(MIS)，该系统的开发主要包括后台数据库的建立和维护，以及前端应用程序的开发两个方面。对于前者主要要求数据的一致性和完整性，而对于后者则要求应用程序功能完备、易用等。本章将通过 Swing 组件实现图书管理系统界面；通过 Oracle 数据库来存储和管理该项目的数据。

本章的学习目标如下：

- 掌握组件和面板的使用方法；
- 掌握自定义组件及注册的相关事件；
- 了解连接 Oracle 的方法。

32.1 图书管理系统原理

图书管理系统是一个很大的信息管理系统，本章的系统只是定位于小型图书馆的应用。根据要求，图书管理系统针对图书管理员进行添加新书、执行借书、还书、查看图书的操作，另外，进入该系统的读者能查看当前图书馆的藏书情况并能执行查询操作，还可以通过该系统注册成为会员。

32.1.1 项目结构框架分析

对于图书管理系统项目，主要利用 Swing 组件实现图形用户界面和 Oracle 数据库的操作，该项目目录如图 32.1 所示。

32.1.2 项目功能业务分析

本节将向读者介绍整个项目要实现的功能。这些功能包括图书馆的初始化界面、用户管理、书籍管理、借书管理和退出管理。

1. 初始化界面

当运行“图书馆”项目中的 MainWindow 类后，就会出现

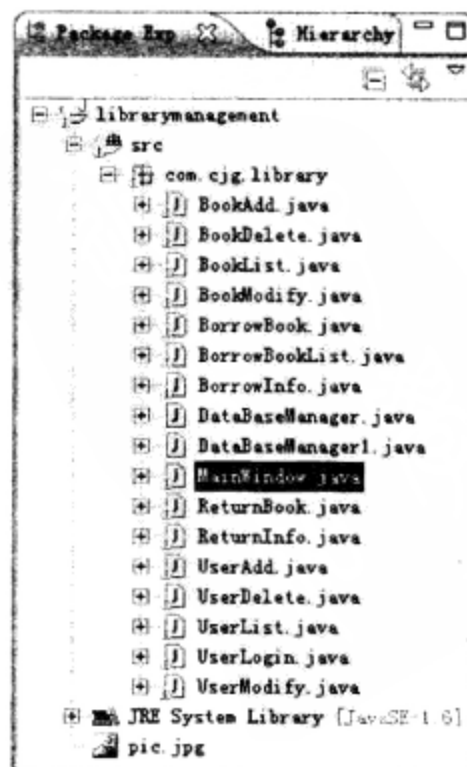


图 32.1 项目目录

如图 32.2 所示的初始界面——图书馆界面。



图 32.2 初始化界面

2. 系统管理菜单——用户登录

当出现初始化界面后，单击“系统管理 | 用户登录”菜单就会出现用户登录界面，在该界面中输入用户名和密码就会进入系统，同时会根据登录用户的权限设置菜单的可用性，具体过程如图 32.3 所示。

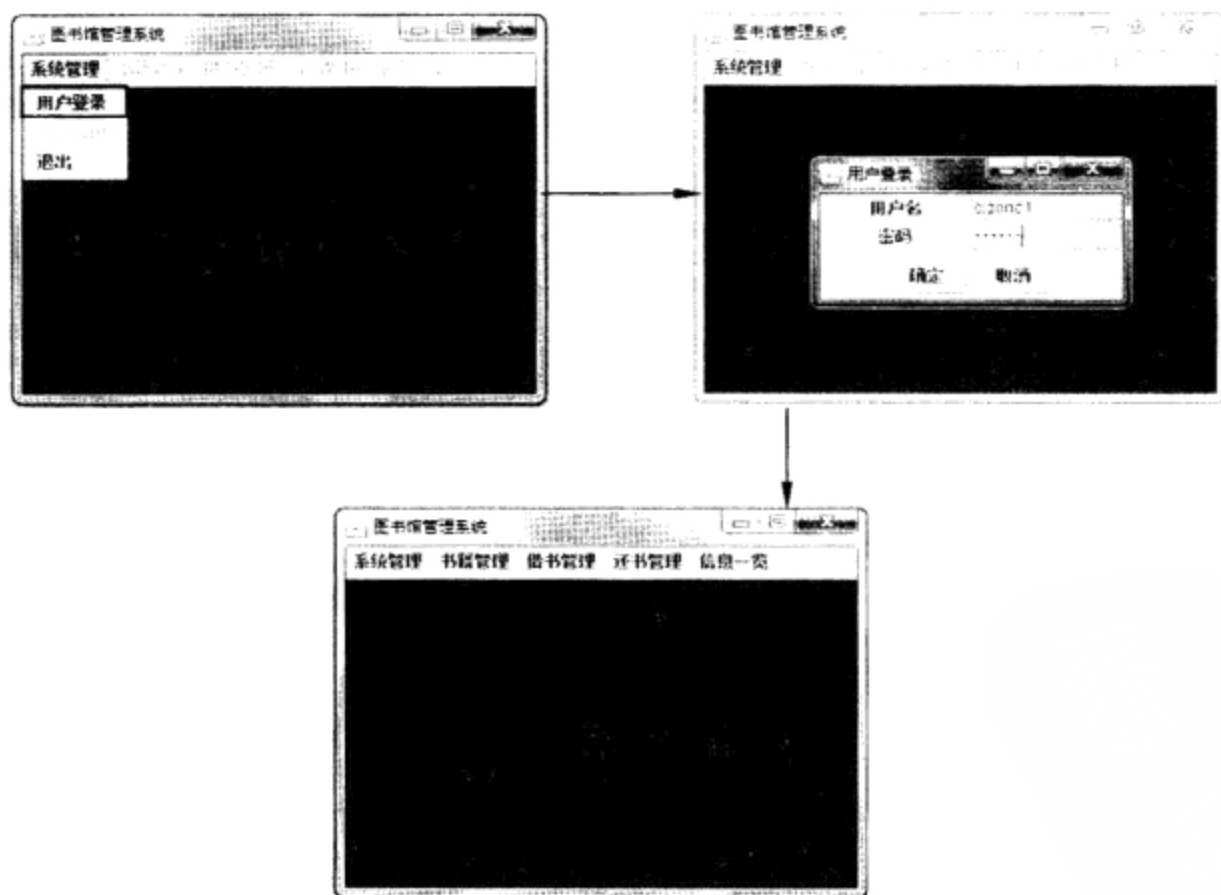


图 32.3 管理员登录

3. 用户管理菜单——用户管理

由于该项目初始化只有一个管理员用户，所以当出现初始化界面后，首先需要通过系统管理员账户添加新的用户名，即可以选择菜单“系统管理” | “用户管理” | “添加用户”命令打开“添加用户”对话框。在该对话框中添加相应信息后，单击“确定”按钮就可以

实现用户的添加功能，具体过程如图 32.4 所示。

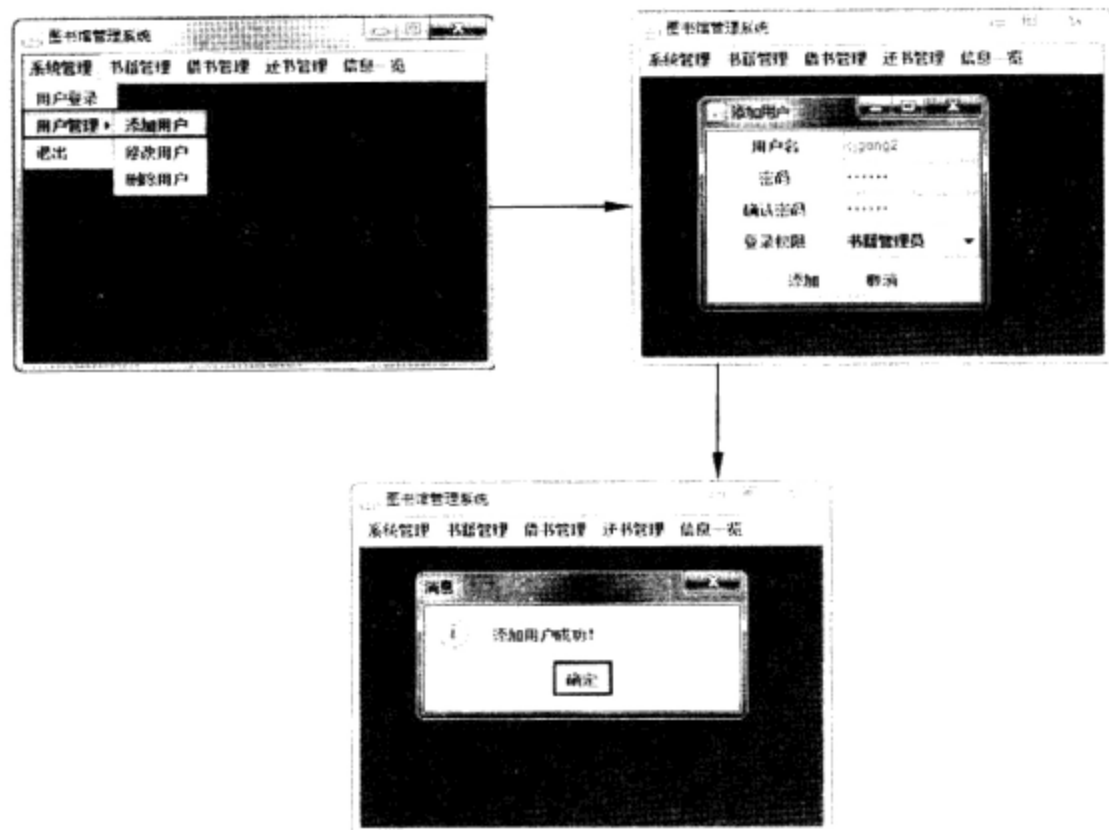


图 32.4 添加书籍管理员用户的过程

注意：为了便于后面的操作，又添加了两个用户，这两个用户的信息如图 32.5 和图 32.6 所示。



图 32.5 cjgong3 用户信息



图 32.6 test 用户信息

对于已经存在的用户，如果想修改用户信息，可以选择菜单“系统管理”|“用户管理”|“修改用户”命令打开“更改密码”对话框。在该对话框中修改相应信息后，单击“更新”按钮就可以实现用户的修改功能，具体过程如图 32.7 所示。

对于已经存在的用户，如果想删除用户信息，可以选择菜单“系统管理”|“用户管理”|“删除用户”命令打开“删除用户”对话框。在该对话框中添加相应信息后，单击“确定”按钮就可以实现删除用户的功能，具体过程如图 32.8 所示。

如果想查看所有用户的信息，可以选择菜单“信息一览”|“用户列表”命令打开“用户列表一览”对话框，该对话框会显示所有用户的信息，具体过程如图 32.9 所示。

注意：在没有删除用户之前，“用户列表一览”对话框如图 32.10 所示。



图 32.7 修改用户信息

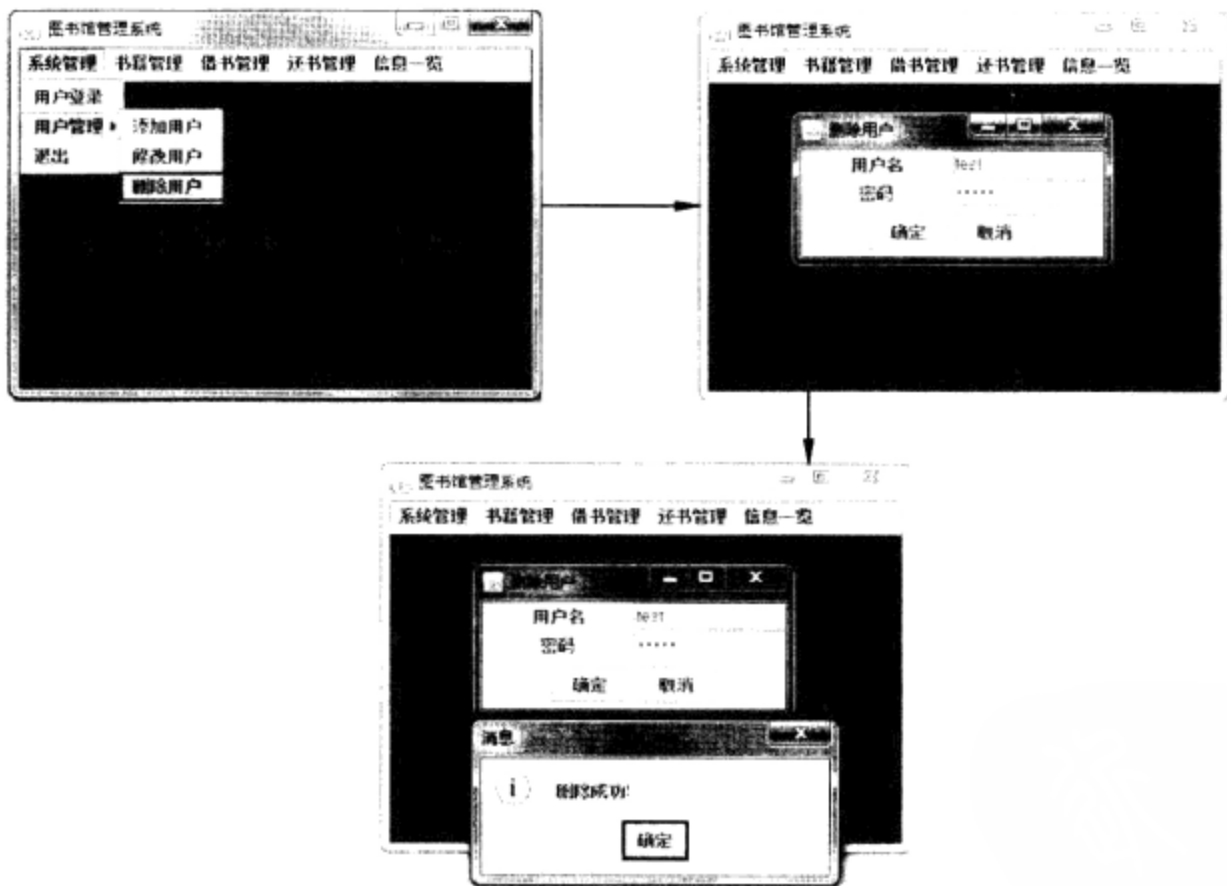


图 32.8 删除用户



图 32.9 用户信息一览

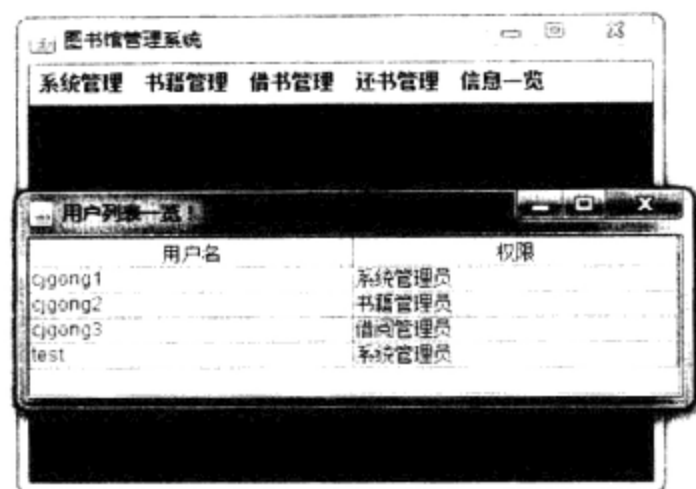


图 32.10 没有删除用户之前的“用户列表一览”对话框

4. 书籍管理——添加书籍

如果想使用“书籍管理”菜单，需要通过书籍管理员用户 cjgong2 来登录。选择菜单“书籍管理”|“添加书籍”命令打开“添加书籍信息”对话框。在该对话框中添加相应信息后，单击“添加”按钮就可以实现添加书籍功能，具体过程如图 32.11 所示。

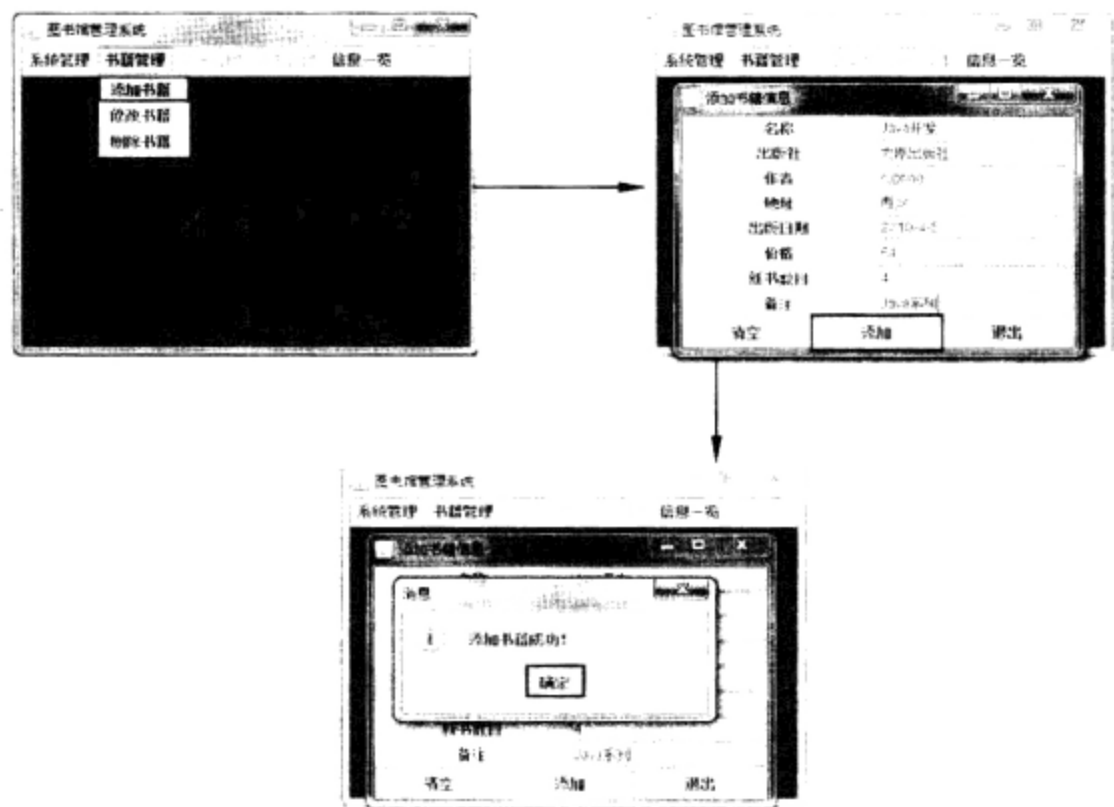


图 32.11 添加书籍功能

注意：为了便于后面的操作，又添加了两本书籍，这两本书籍的信息如图 32.12 和图 32.13 所示。

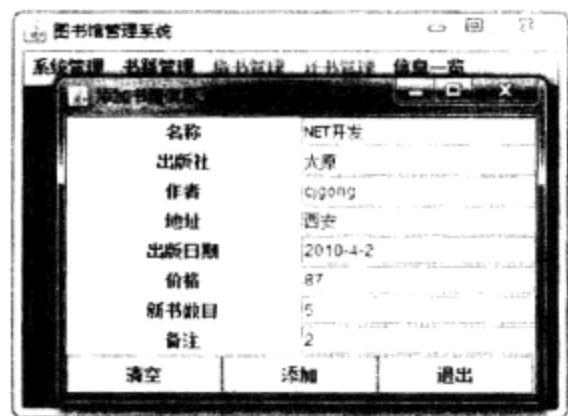


图 32.12 NET 开发书籍

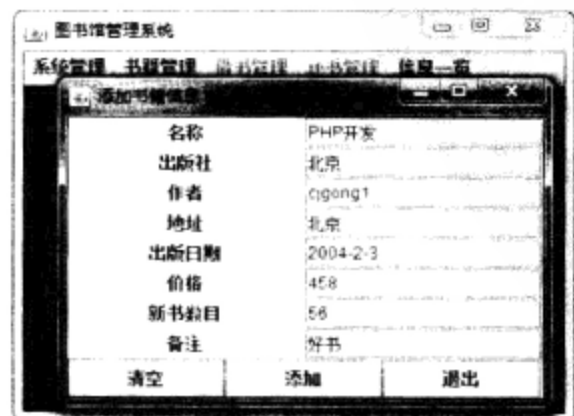


图 32.13 PHP 开发书籍

如果想查看所有书籍的信息，可以选择菜单“信息一览”|“书籍列表”命令打开“书籍信息一览”对话框，该对话框会显示所要查找的图书信息。当在“名称”文本框中输入“NET 开发”后，单击“查询”按钮就会在下面的 table 组件中显示出包含该书名的图书信息，具体过程如图 32.14 所示。当在“作者”文本框中输入“cjgong1”后，单击“查询”按钮就会在下面的 table 组件中显示出包含该作者的图书信息，具体过程如图 32.15 所示。当在“出版社”文本框中输入“太原”后，单击“查询”按钮就会在下面的 table 组件中显示出包含该出版社的图书信息，具体过程如图 32.16 所示。

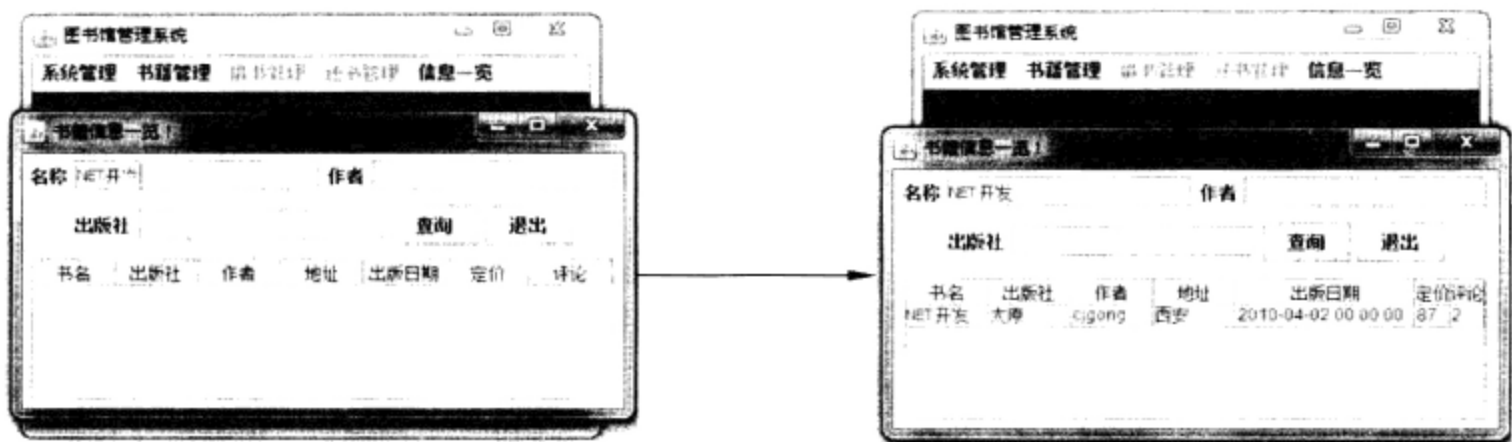


图 32.14 通过书名查看图书信息

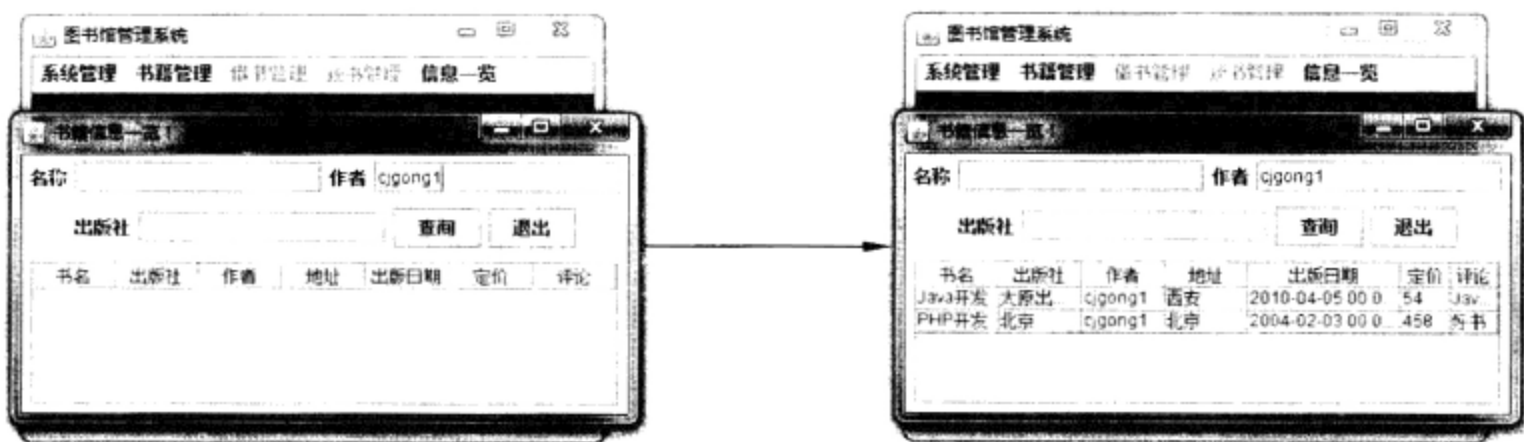


图 32.15 通过作者查看图书信息

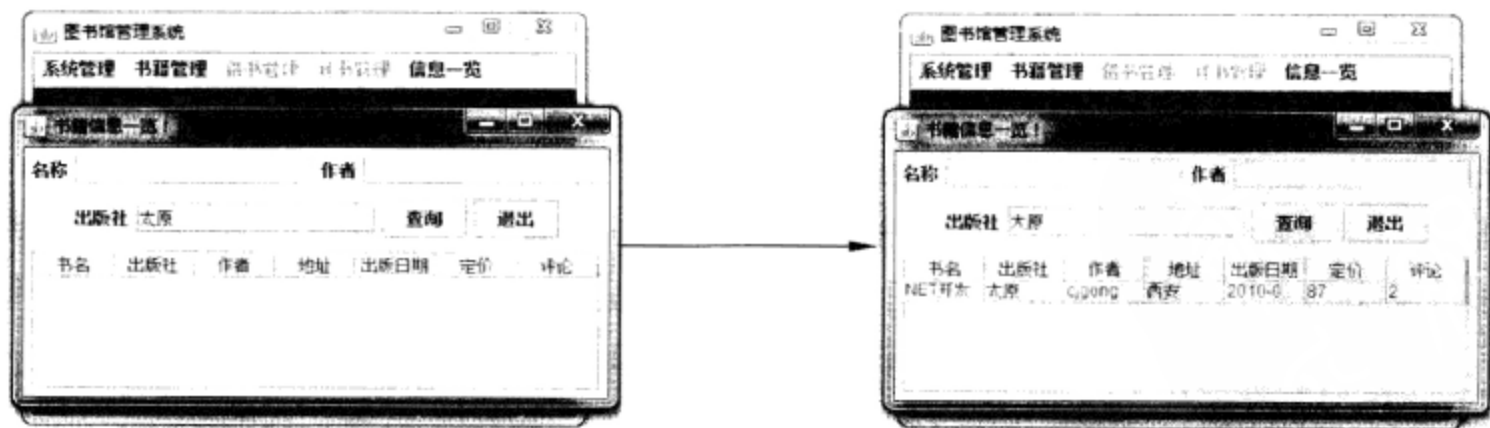


图 32.16 通过出版社查看图书信息

对于已经存在的书籍，如果想修改书籍信息，可以选择菜单“书籍管理”|“修改书籍”命令打开“修改书籍信息”对话框。在该对话框中输入相应信息后，单击“确定”按钮会显示出该书籍的所有信息。修改相应信息后单击“更新”按钮，可以实现书籍信息的修改功能，具体过程如图 32.17 所示。

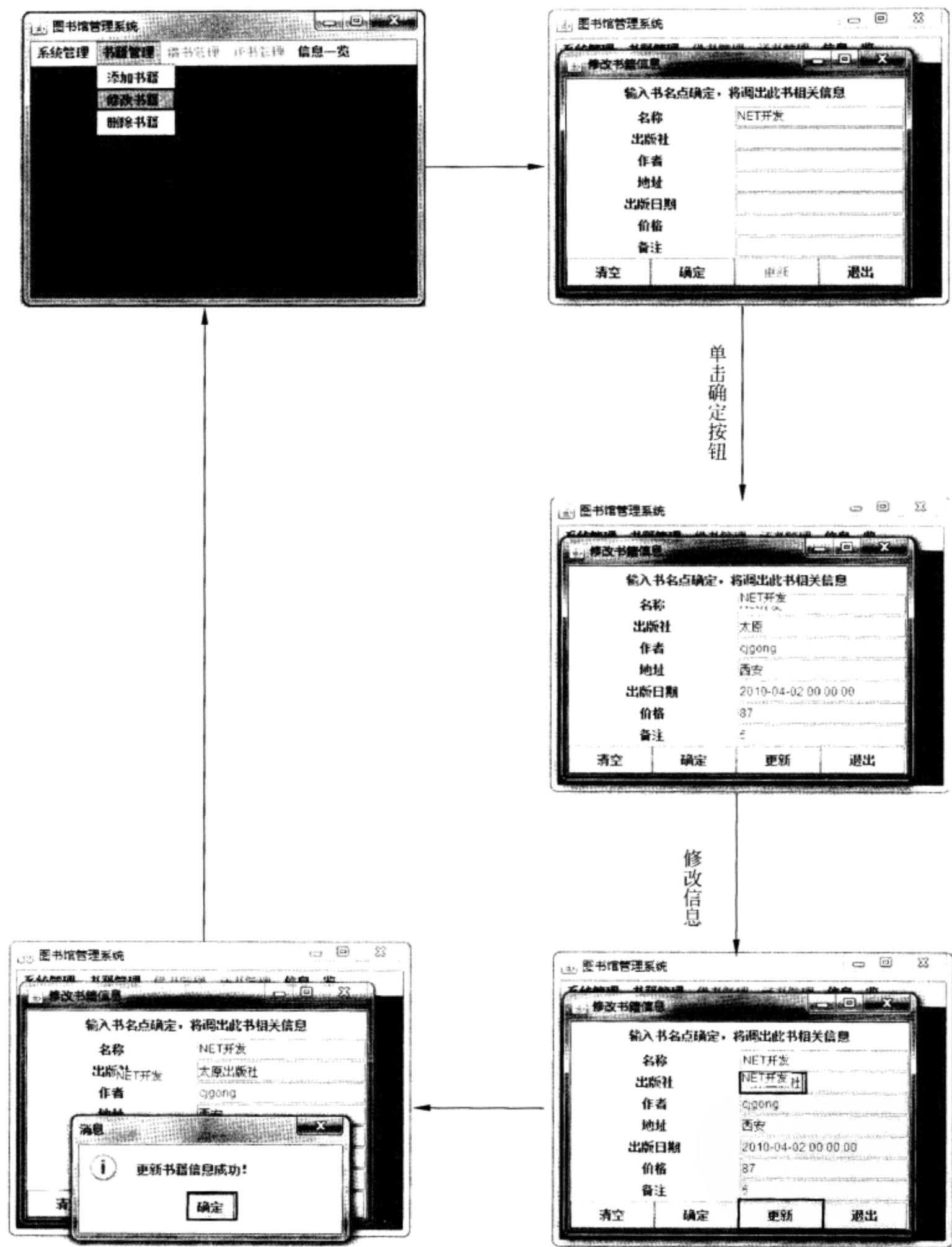


图 32.17 修改书籍信息

对于已经存在的书籍，如果想删除书籍信息，可以选择菜单“书籍管理”|“用户管理”|“删除用户”命令打开“删除用户”对话框。在该对话框中添加相应信息后，单击“确定”按钮可以实现书籍的删除功能，具体过程如图 32.18 所示。如果这时候再通过“信息一览”菜单查询该书籍，则不会显示该书籍信息，如图 32.19 所示。

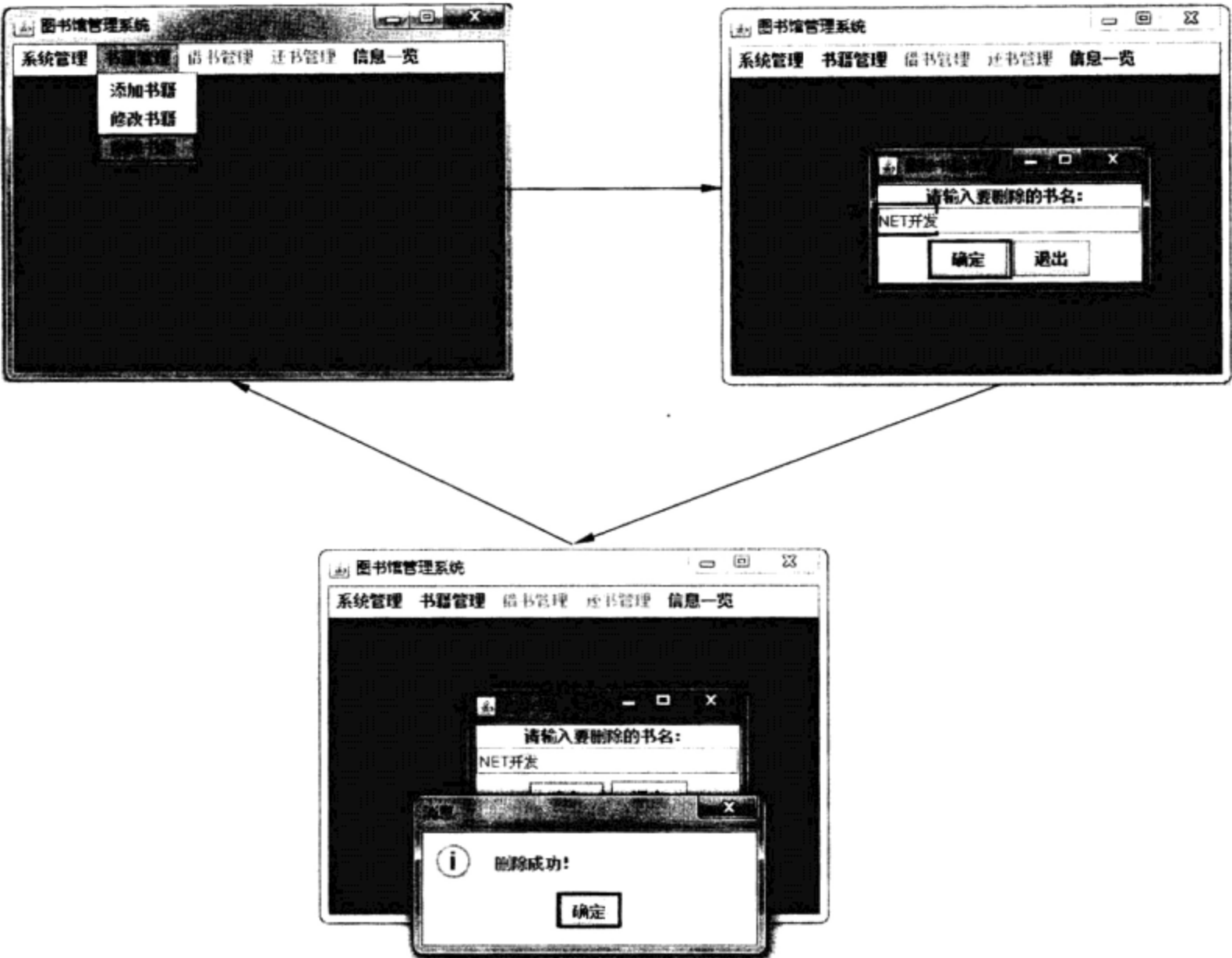


图 32.18 删除图书



图 32.19 无图书信息

5. 借书管理

如果想使用“借书管理”菜单，需要通过书籍管理员用户 `cjgong3` 来登录。选择菜单“借书管理”|“书籍出借”命令打开“书籍出借”对话框。在该对话框中添加相应信息后，单击“确定”按钮可以实现出借书籍的功能，具体过程如图 32.20 所示。

注意：为了便于后面的操作，又添加了两个出借图书的记录，这两个用户的信息如图 32.21 和图 32.22 所示。

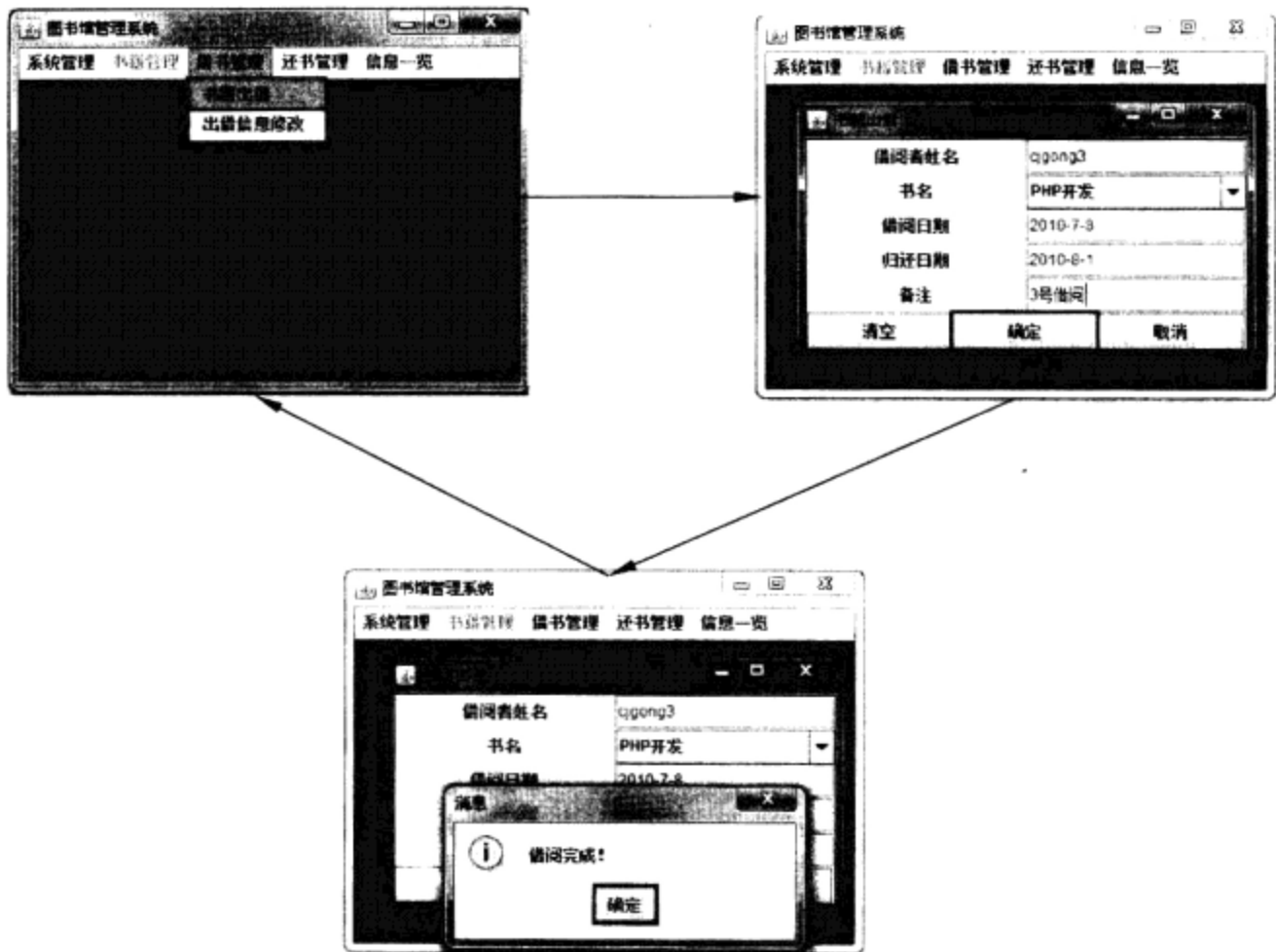


图 32.20 出借书籍的过程



图 32.21 添加出借记录

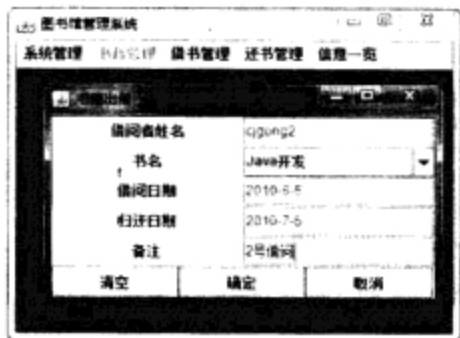


图 32.22 添加出借记录

如果想查看出借书籍的信息，可以选择菜单“信息一览”|“借阅情况表”命令打开“书籍借阅一览”对话框。在该对话框中输入相应信息后，单击查询按钮会显示相应信息，具体过程如图 32.23 所示。

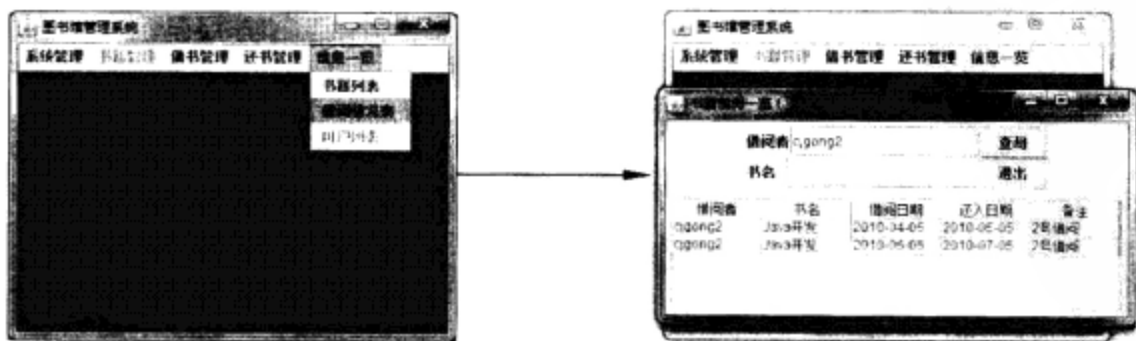


图 32.23 浏览借阅信息

对于已经存在的借阅记录，如果想修改该记录信息，可以选择菜单“借书管理”|“出借信息修改”命令打开“修改书籍出借信息”对话框。在该对话框中添加相应信息后，单

击“确定”按钮会显示该记录信息。修改相应信息后单击“更新”按钮，会实现出借信息修改功能，具体过程如图 32.24 所示。

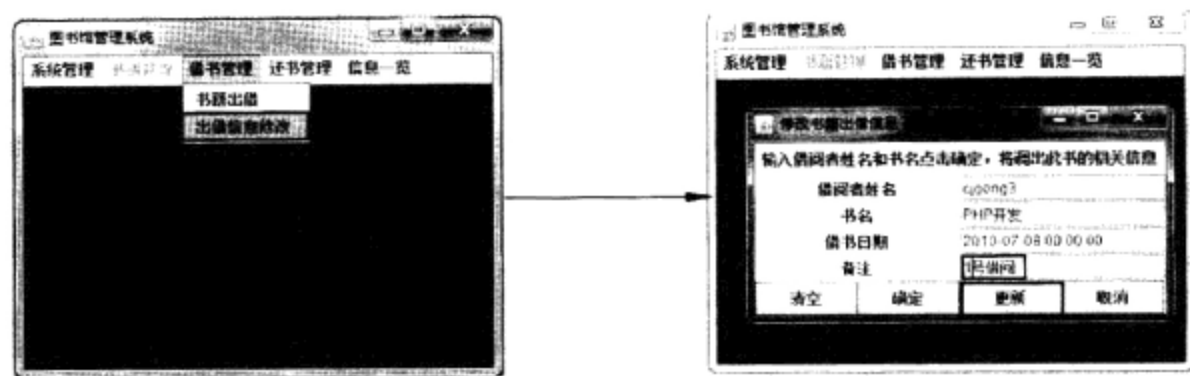


图 32.24 修改书籍出借信息

注意：当通过菜单出借信息修改完出借记录后，如果想查看修改后的出借记录，可以通过“信息一览”|“借阅情况表”菜单来实现，具体信息如图 32.25 所示。



图 32.25 修改出借记录信息

6. 退出功能


当出现初始界面后，如果想实现退出功能，可以通过单击右上角的图标来实现，如图 32.26 所示。也可以通过菜单“系统管理”|“退出”命令实现退出功能，如图 32.27 所示。



图 32.26 实现退出功能



图 32.27 通过菜单退出

32.2 图书管理系统项目——图书的操作

图书管理系统项目中对图书实现了 CRUD（增删改查）操作，即添加图书操作、删除图书操作、修改图书信息操作和浏览图书信息操作，各种操作所对应的类如图 32.28 所示。

32.2.1 实现添加图书功能的类

BookAdd 类为图书管理系统项目中图书操作的添加图书操作，该类不仅继承了 JFrame

类，而且还实现了监听各个组件相应动作的功能，该类的类图如图 32.29 所示，具体内容如代码 32.1 所示。

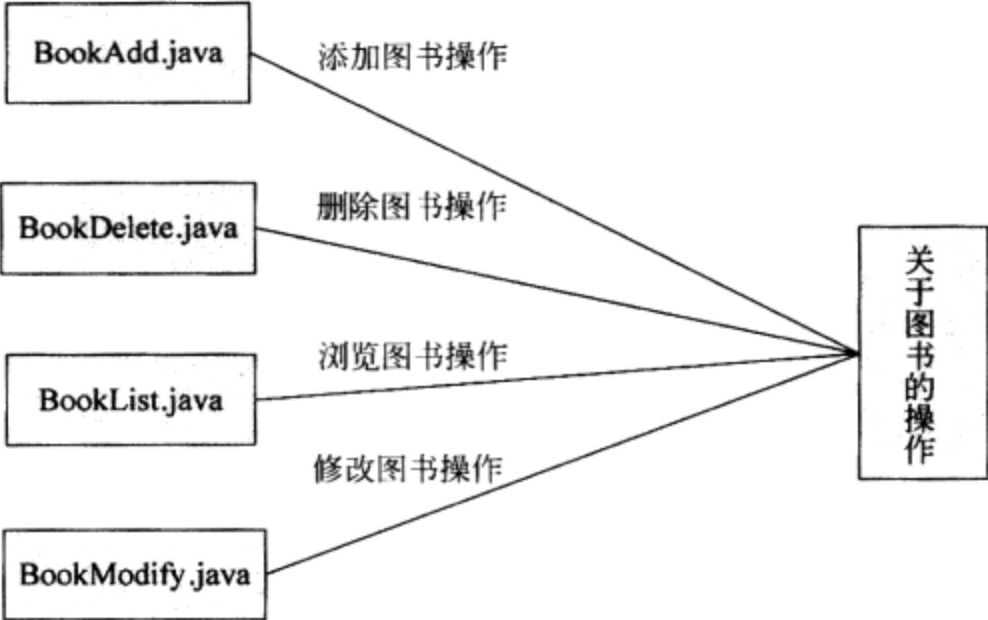


图 32.28 程序关系图



图 32.29 BookAdd 类的类图

代码 32.1 添加书类：BookAdd.java

```
public class BookAdd extends JFrame implements ActionListener {
    DataBaseManager db = new DataBaseManager(); //创建DataBaseManager类
    ResultSet rs; //数据集对象
    JPanel panel1, panel2; //创建面板对象
    //创建标签对象
    JLabel BookNameLabel, PressNameLabel, AuthorLabel, AddressLabel,
        PressDateLabel, PriceLabel, BookCountLabel, CommentLabel;
    //创建输入文本框对象
    JTextField BookNameTextField, PressNameTextField, AuthorTextField,
        AddressTextField, PressDateTextField, PriceTextField,
        BookCountTextField, CommentTextField;
    Container c; //容器对象 c
    JButton ClearBtn, AddBtn, ExitBtn; //按钮对象
    public BookAdd() { //构造函数
        super("添加图书信息"); //设置标签
        c = getContentPane(); //为对象 c 赋值
        c.setLayout(new BorderLayout()); //设置布局管理器
        //为标签对象赋值
        BookNameLabel = new JLabel("名称", JLabel.CENTER);
        PressNameLabel = new JLabel("出版社", JLabel.CENTER);
        AuthorLabel = new JLabel("作者", JLabel.CENTER);
```

```

AddressLabel = new JLabel("地址", JLabel.CENTER);
PressDateLabel = new JLabel("出版日期", JLabel.CENTER);
PriceLabel = new JLabel("价格", JLabel.CENTER);
BookCountLabel = new JLabel("新书数目", JLabel.CENTER);
CommentLabel = new JLabel("备注", JLabel.CENTER);
//为面板对象赋值
BookNameTextField = new JTextField(15);
PressNameTextField = new JTextField(15);
AuthorTextField = new JTextField(15);
AddressTextField = new JTextField(15);
PressDateTextField = new JTextField(15);
PriceTextField = new JTextField(15);
BookCountTextField = new JTextField(15);
CommentTextField = new JTextField(15);
panel1 = new JPanel(); //为对象 panel1 赋值
panel1.setLayout(new GridLayout(8, 2)); //设置布局
//添加各种标签对象和面板对象到对象 panel1 里
panel1.add(BookNameLabel);
... //省略部分代码
panel2 = new JPanel(); //为对象 panel2 赋值
panel2.setLayout(new GridLayout(1, 3)); //设置布局管理器
//为各种按钮赋值并注册监听事件
ClearBtn = new JButton("清空");
ClearBtn.addActionListener(this);
AddBtn = new JButton("添加");
AddBtn.addActionListener(this);
ExitBtn = new JButton("退出");
ExitBtn.addActionListener(this);
//添加各种按钮到对象 panel2 中
panel2.add(ClearBtn);
panel2.add(AddBtn);
panel2.add(ExitBtn);
//添加对象 panel1 和 panel2 到对象 c 中
c.add(panel1, BorderLayout.CENTER);
c.add(panel2, BorderLayout.SOUTH);
}
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == ExitBtn) {
        db.closeConnection();
        this.dispose();
    } else if (e.getSource() == ClearBtn) {
        BookNameTextField.setText("");
        PressNameTextField.setText("");
        AuthorTextField.setText("");
        AddressTextField.setText("");
        PressDateTextField.setText("");
        PriceTextField.setText("");
        BookCountTextField.setText("");
        CommentTextField.setText("");
    } else if (e.getSource() == AddBtn) {
        if (BookNameTextField.getText().trim().equals("")) {
            JOptionPane.showMessageDialog(null, "书名不能为空!");
        } else if (PressNameTextField.getText().trim().equals("")) {
            JOptionPane.showMessageDialog(null, "出版社不能为空!");
        } else if (AuthorTextField.getText().trim().equals("")) {
            JOptionPane.showMessageDialog(null, "作者不能为空!");
        } else if (BookCountTextField.getText().trim().equals("")) {
            JOptionPane.showMessageDialog(null, "新书数目不能为空!");
        }
    }
}

```



```

    } else {
        try {
            String strSQL = "insert into books(bookname,press,
            author,address,pressDate,price,books_count,com)
            values('"
                + BookNameTextField.getText().trim()
                + "','"
                + PressNameTextField.getText().trim()
                + "','"
                + AuthorTextField.getText().trim()
                + "','"
                + AddressTextField.getText().trim()
                + "','"
                + PressDateTextField.getText().trim()
                + "','"
                + PriceTextField.getText().trim()
                + "','"
                + BookCountTextField.getText().trim()
                + "','"
                + CommentTextField.getText().trim() + "')";
            if (db.updateSql(strSQL)) {
                JOptionPane.showMessageDialog(null, "添加图书成功!");
                this.dispose();
            } else {
                JOptionPane.showMessageDialog(null, "添加图书失败!");
                this.dispose();
            }
            db.closeConnection();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}
}
}
}

```

【代码解析】

- 上述代码的构造函数主要用来实现图书管理系统中添加图书窗口，该用户界面涉及的具体容器、对象和布局如图 32.30 所示。

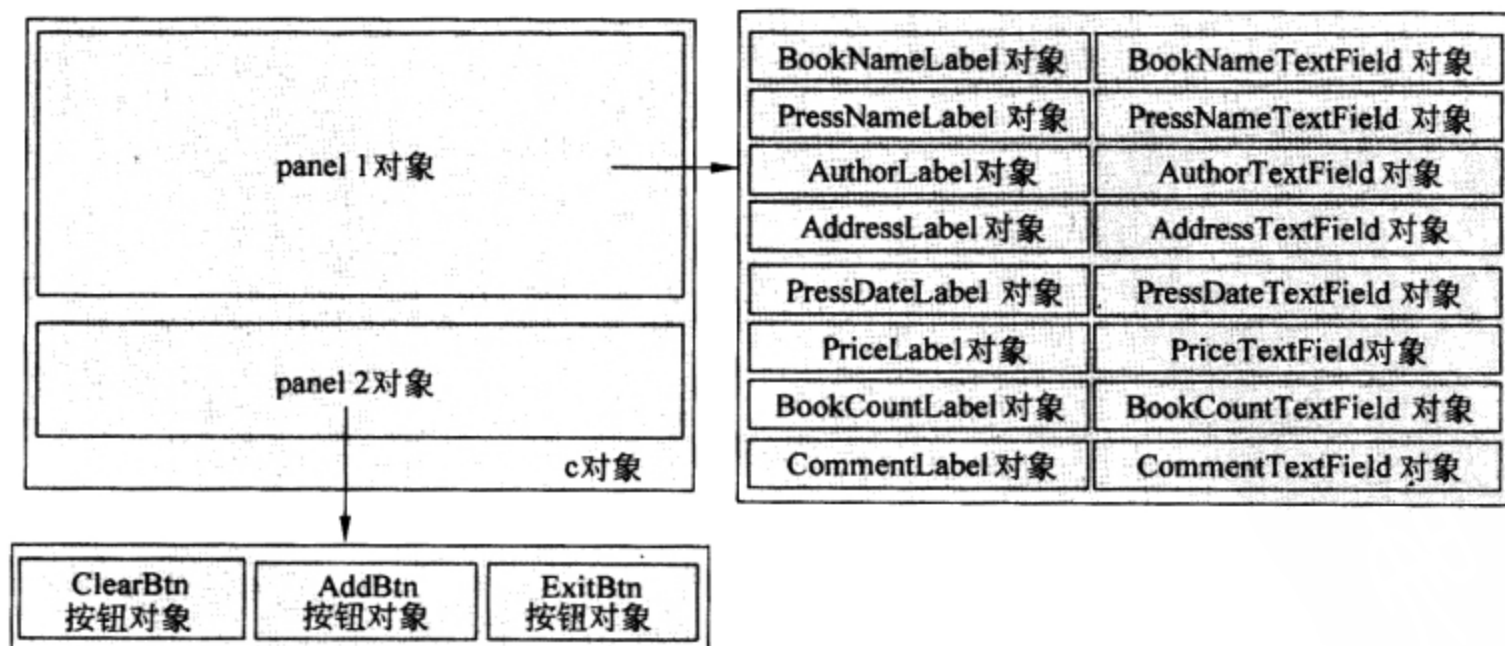


图 32.30 布局

- 在上述代码的处理监听事件方法中，如果发生动作的按钮为“退出”按钮，该窗口将不显示；如果发生动作的按钮为“清空”按钮，则该窗口中所有文本输入框的内容为空；如果发生动作的按钮为“添加”按钮，首先判断文本输入框的内容是否为空，并根据文本输入框的内容创建 SQL 插入语句 strSQL，然后再通过 updateSql 方法执行该语句。如果 SQL 语句的执行结果为 true，则显示“添加书籍成功”的信息框，否则显示“添加书籍失败”的信息框。

32.2.2 实现修改图书功能的类

BookModify 类为图书管理系统项目中修改图书操作的类，同样，该类不仅继承了 JFrame 类，而且还实现了监听各个组件相应动作的功能。该类的类图如图 32.31 所示，具体内容如代码 32.2 所示。



图 32.31 BookModify 类的类图

代码 32.2 修改图书的类: BookModify.java

```

public class BookModify extends JFrame implements ActionListener {
    DataBaseManager db = new DataBaseManager(); //创建DataBaseManager类
    ResultSet rs; //创建数据集类
    JPanel panel1, panel2, panel3; //创建各种面板类对象
    //创建提示标签类对象并初始化其内容
    JLabel TipLabel = new JLabel("输入书名点确定，将调出此书相关信息");
    //创建各种标签对象
    JLabel BookNameLabel, PressNameLabel, AuthorLabel, AddressLabel,
        PressDateLabel, PriceLabel, CommentLabel;
    //创建各种输入文本框对象
    JTextField BookNameTextField, PressNameTextField, AuthorTextField,
        AddressTextField, PressDateTextField, PriceTextField,
        CommentTextField;
    Container c; //创建 Container 对象
    JButton ClearBtn, YesBtn, UpdateBtn, ExitBtn; //创建按钮对象
    public BookModify() { //构造函数
        super("修改图书信息"); //设置标题
        c = getContentPane(); //为对象 c 赋值
        c.setLayout(new BorderLayout()); //设置布局管理器
        panel3 = new JPanel(); //为对象 panel3 赋值
        //添加对象 TipLabel 到对象 panel3 中
        panel3.add(TipLabel);
        c.add(panel3, BorderLayout.NORTH); //添加对象 panel3 到北部
        //创建各种标签对象
        BookNameLabel = new JLabel("名称", JLabel.CENTER);
        PressNameLabel = new JLabel("出版社", JLabel.CENTER);
        AuthorLabel = new JLabel("作者", JLabel.CENTER);
        AddressLabel = new JLabel("地址", JLabel.CENTER);
    }
}
  
```

```

PressDateLabel = new JLabel("出版日期", JLabel.CENTER);
PriceLabel = new JLabel("价格", JLabel.CENTER);
CommentLabel = new JLabel("备注", JLabel.CENTER);
//创建各种输入文本框对象
BookNameTextField = new JTextField(15);
PressNameTextField = new JTextField(15);
AuthorTextField = new JTextField(15);
AddressTextField = new JTextField(15);
PressDateTextField = new JTextField(15);
PriceTextField = new JTextField(15);
CommentTextField = new JTextField(15);
panel1 = new JPanel(); //为对象 panel1 赋值
panel1.setLayout(new GridLayout(7, 2)); //设置对象 panel1 布局管理器
//添加各种标签对象和输入文本框对象到对象 panel1 中
panel1.add(BookNameLabel);
... //省略部分代码
panel2 = new JPanel(); //为对象 panel2 赋值
panel2.setLayout(new GridLayout(1, 4)); //设置对象 panel2 的布局管理器
//创建各种按钮对象
ClearBtn = new JButton("清空");
YesBtn = new JButton("确定");
UpdateBtn = new JButton("更新");
ExitBtn = new JButton("退出");
//添加按钮对象到对象 panel2 中
panel2.add(ClearBtn);
panel2.add(YesBtn);
panel2.add(UpdateBtn);
panel2.add(ExitBtn);
//为各种按钮对象注册事件监听器
ClearBtn.addActionListener(this);
YesBtn.addActionListener(this);
UpdateBtn.addActionListener(this);
ExitBtn.addActionListener(this);
UpdateBtn.setEnabled(false);
//添加对象 panel1 和 panel2 到对象 c 中
c.add(panel1, BorderLayout.CENTER);
c.add(panel2, BorderLayout.SOUTH);
}
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == ExitBtn) {
        this.dispose();
    } else if (e.getSource() == ClearBtn) {
        BookNameTextField.setText("");
        PressNameTextField.setText("");
        AuthorTextField.setText("");
        AddressTextField.setText("");
        PressDateTextField.setText("");
        PriceTextField.setText("");
        CommentTextField.setText("");
    } else if (e.getSource() == YesBtn) {
        try {
            String strSQL = "select * from books where bookName='"
                + BookNameTextField.getText().trim() + "'";
            ... //省略部分代码
        } catch (NullPointerException upe) {
            System.out.println(upe.toString());
        } catch (SQLException sqle) {
            System.out.println(sqle.toString());
        }
    }
}

```

```
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    } else if (e.getSource() == UpdateBtn) {
        try {
            String strSQL = "update books set bookName='"
                + BookNameTextField.getText().trim() + "',press='"
                + PressNameTextField.getText().trim() + "',author='"
                + AuthorTextField.getText().trim() + "',address='"
                + AddressTextField.getText().trim() + "',pressDate='"
                + PressDateTextField.getText().trim() + "',price='"
                + PriceTextField.getText().trim() + "',com='"
                + CommentTextField.getText().trim() + "'";
            //省略部分代码
            ...
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}
```

【代码解析】

- 上述代码在该类的构造函数中实现了图书管理系统中修改图书信息窗口的功能，该用户界面涉及的具体容器、对象和布局如图 32.32 所示。
- 在上述代码的处理监听事件方法中，如果发生动作的按钮为“退出”按钮，该窗口将不显示。如果发生动作的按钮为“清空”按钮，则该窗口中所有文本输入框的内容为空。如果发生动作的按钮为“确定”按钮，首先会检查“文件名”文本框中的内容是否为空。如果不为空，则根据该文本框内容创建查询语句 strSQL，然后执行该语句；如果执行结果为空，则出现“此书没有在书库中”的信息框，否则会把查询结果显示在相应文本框中，最后设置更新按钮为可用。如果发生动作的按钮为“更新”按钮，则首先根据文本框的内容创建 SQL 更新语句 strSQL，然后再通过 updateSql()方法执行该语句，如果 SQL 语句的执行结果为 true，则显示“更新书籍信息成功”的信息框，否则显示“更新书籍信息失败”的信息框。

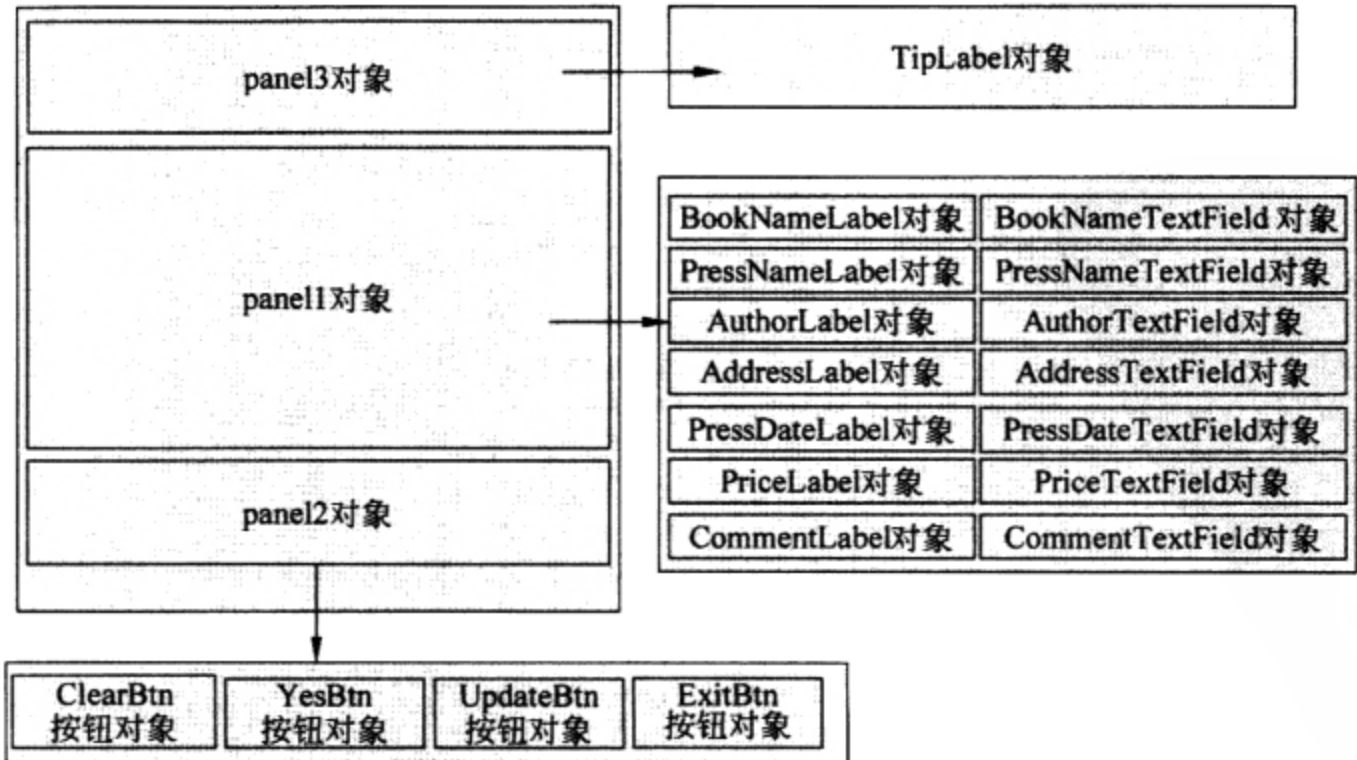


图 32.32 布局

32.2.3 实现浏览图书信息的类

BookList 类为图书管理系统项目中浏览图书信息操作的类，该类不仅继承了 JFrame 类，而且还实现了监听各个组件相应动作的功能，该类的类图如图 32.33 所示，具体内容如代码 32.3 所示。

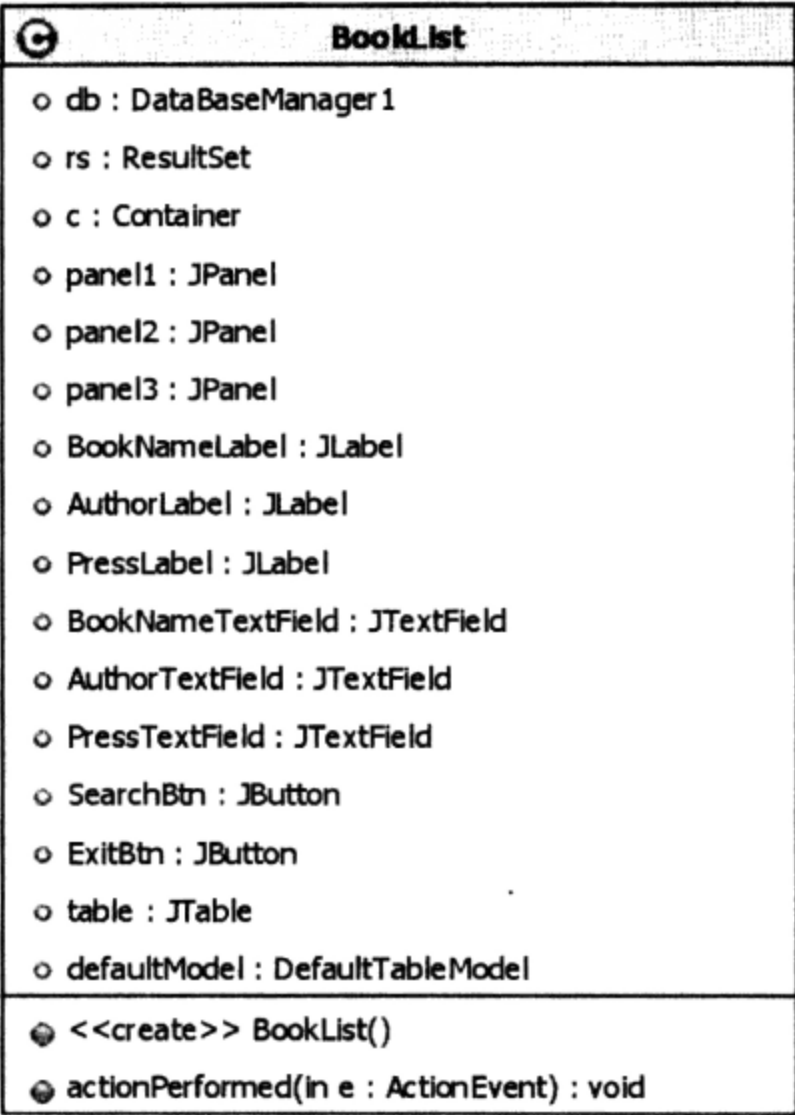


图 32.33 BookList 类的类图

代码 32.3 浏览图书信息：BookList.java

```
public class BookList extends JFrame implements ActionListener {
    DataBaseManager db = new DataBaseManager();           //创建连接数据库类
    ResultSet rs;                                           //数据集对象
    Container c;                                           //容器对象
    //创建各种面板对象
    JPanel panel1, panel2, panel3;
    JLabel BookNameLabel, AuthorLabel, PressLabel;        //创建各种标签对象
    //创建各种输入文本框对象
    JTextField BookNameTextField, AuthorTextField, PressTextField;
    JButton SearchBtn, ExitBtn;                            //创建按钮对象
    JTable table = null;                                   //创建表格对象
    DefaultTableModel defaultModel = null;                 //创建表格模式变量
    public BookList() {                                    //构造函数
        super("图书信息一览!");                           //设置窗口的标题
        c = getContentPane();                             //获取容器
    }
}
```



```

c.setLayout(new BorderLayout()); //设置布局管理器
//为标签对象赋值
BookNameLabel = new JLabel("名称", JLabel.CENTER);
AuthorLabel = new JLabel("作者", JLabel.CENTER);
PressLabel = new JLabel("出版社", JLabel.CENTER);
//为输入文本框对象赋值
BookNameTextField = new JTextField(15);
AuthorTextField = new JTextField(15);
PressTextField = new JTextField(15);
//为按钮对象赋值
SearchBtn = new JButton("查询");
ExitBtn = new JButton("退出");
//为按钮对象注册事件
SearchBtn.addActionListener(this);
ExitBtn.addActionListener(this);
//创建和设置 panel1 对象
panel1 = new JPanel(); //创建 panel1 对象
//添加相应对象到对象 panel1 中
panel1.add(BookNameLabel);
panel1.add(BookNameTextField);
panel1.add(AuthorLabel);
panel1.add(AuthorTextField);
//创建和设置 panel3 对象
panel3 = new JPanel(); //创建 panel3 对象
//添加相应对象到对象 panel3 中
panel3.add(PressLabel);
panel3.add(PressTextField);
panel3.add(SearchBtn);
panel3.add(ExitBtn);
//创建标题的字符串
String[] name = { "书名", "出版社", "作者", "地址", "出版日期",
"定价", "评论" };
String[][] data = new String[0][0]; //创建数组
defaultModel = new DefaultTableModel(data, name); //为表格模式对象赋值
table = new JTable(defaultModel); //设置表格对象的模式
//设置表格的大小
table.setPreferredSize(new Dimension(400, 80));
JScrollPane s = new JScrollPane(table); //创建表格的滚动面板
//创建和设置 panel2 对象
panel2 = new JPanel(); //创建 panel2 对象
panel2.add(s); //添加对象 s 到面板对象 panel2
//添加 panel1、panel2 和 panel3 到容器中
c.add(panel1, BorderLayout.NORTH);
c.add(panel3, BorderLayout.CENTER);
c.add(panel2, BorderLayout.SOUTH);
}

public void actionPerformed(ActionEvent e) { //注册事件
    if (e.getSource() == SearchBtn) { //当发生事件的按钮为 SearchBtn
        //创建实现查询功能的 SQL 语句
        String strSQL = "select bookName,press,author,address,
        pressDate,price,com from books";
        String strSql = null; //创建一个字符串
        //根据输入框的内容来创建实现查询功能的 SQL 语句
        if (BookNameTextField.getText().trim().equals("")
            && AuthorTextField.getText().trim().equals("")
            && PressTextField.getText().trim().equals("")) {

```



```

        strSql = strSql;
    } else if (BookNameTextField.getText().trim().equals("")
        && AuthorTextField.getText().trim().equals("")) {
        strSql = strSql + " where press='"
            + PressTextField.getText().trim() + "'";
    } else if (BookNameTextField.getText().trim().equals("")
        && PressTextField.getText().trim().equals("")) {
        strSql = strSql + " where author='"
            + AuthorTextField.getText().trim() + "'";
    }
    ... //省略部分代码
    try { //通过表格显示查询到的结果
        //删除表格中原有的数据
        int rowCount = defaultModel.getRowCount() - 1;
        //获取的 table 中的数据行
        int j = rowCount; //创建一个变量
        //通过循环删除表格原有的数据
        for (int i = 0; i <= rowCount; i++) {
            defaultModel.removeRow(j); //删除 rowCount 行的数据
            defaultModel.setRowCount(j); //重新设置行数
            j = j - 1;
        }
        rs = db.getResult(strSql); //获取执行查询语句的结果
        //遍历执行结果
        while (rs.next()) {
            Vector data = new Vector(); //创建集合变量
            //添加执行结果到集合中
            data.addElement(rs.getString(1));
            data.addElement(rs.getString(2));
            data.addElement(rs.getString(3));
            data.addElement(rs.getString(4));
            data.addElement(rs.getString(5));
            data.addElement(rs.getString(6));
            data.addElement(rs.getString(7));
            defaultModel.addRow(data); //设置表格模式
        }
        table.revalidate(); //更新表格
    } catch (SQLException sqle) {
        System.out.println(sqle.toString());
    } catch (Exception ex) {
        System.out.println(ex.toString());
    }
} else if (e.getSource() == ExitBtn) { //当发生事件的按钮为 ExitBtn
    db.closeConnection(); //关闭数据库连接
    this.dispose(); //退出系统
}
}
}

```

【代码解析】

- ❑ 上述代码的构造函数主要用来实现图书管理系统中浏览图书信息窗口，该用户界面涉及的具体容器、对象和布局如图 32.34 所示。
- ❑ 在上述代码的处理监听事件方法中，如果发生动作的按钮为“退出”按钮，则该窗口将不显示同时关闭连接。如果发生动作的按钮为“查询”按钮，则首先根据文本框的内容创建 SQL 查询语句 strSql，然后重新设置表格模式对象 defaultModel，

最后把 strSQL 语句的执行结果存储到集合 data 中并在 table 对象中显示出来。

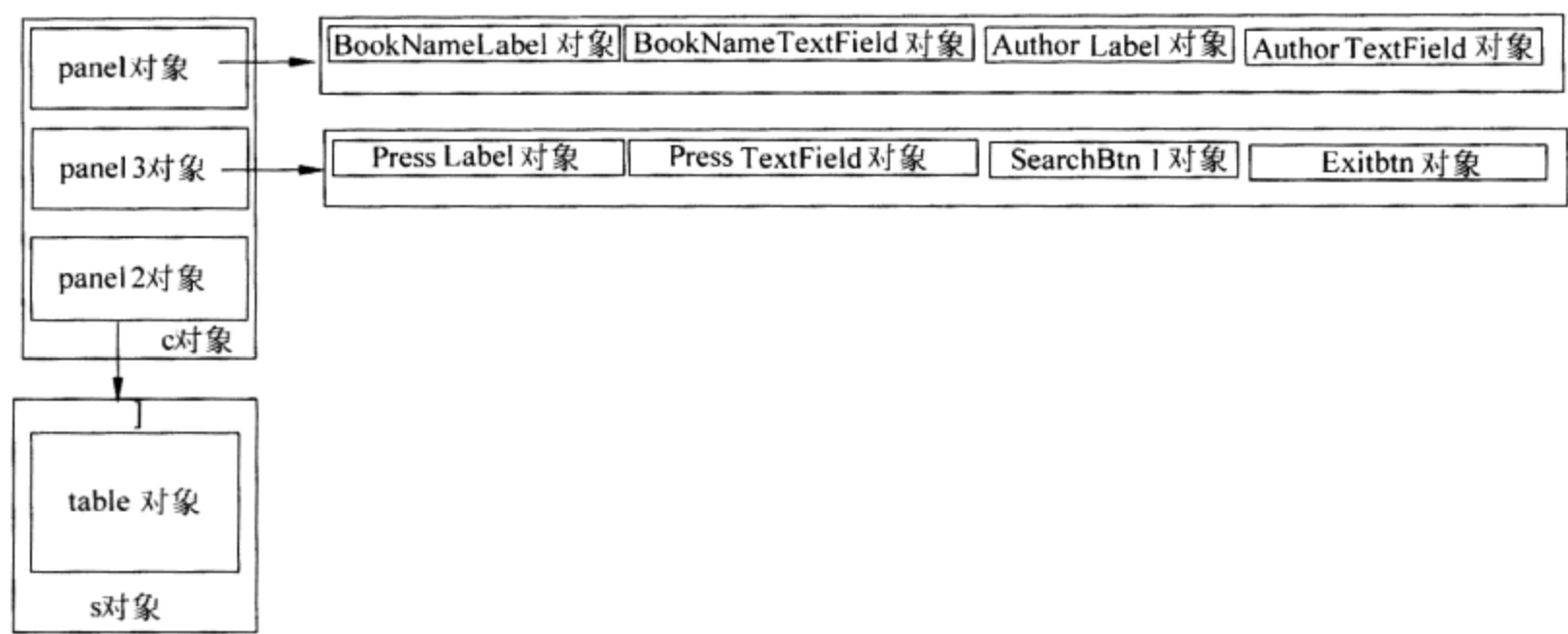


图 32.34 布局

32.2.4 实现删除图书信息的类

BookDelete 类为图书管理系统项目中删除图书操作的类，该类不仅继承了 JFrame 类，还实现了监听各个组件相应动作的功能，该类的类图如图 32.35 所示，具体内容如代码 32.4 所示。

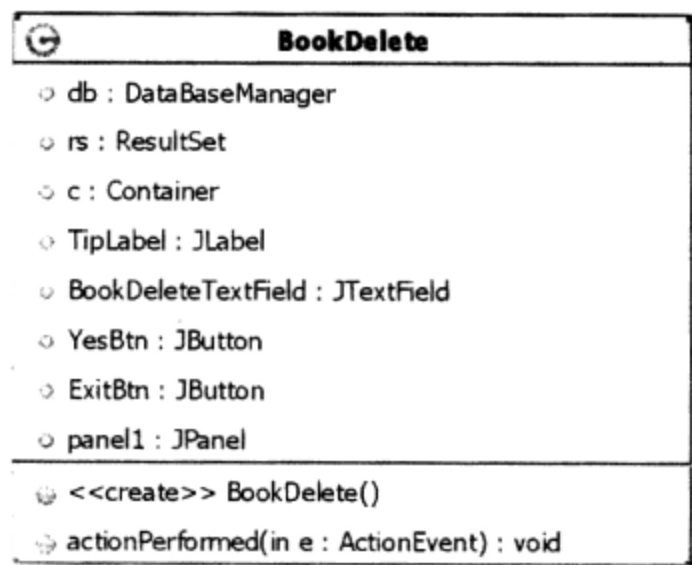


图 32.35 BookDelete 类的类图

代码 32.4 删除图书信息：BookDelete.java

```
public class BookDelete extends JFrame implements ActionListener {
    //创建各种成员变量
    DataBaseManager db = new DataBaseManager();           //创建操作数据对象
    ResultSet rs;                                           //创建数据集对象
    Container c;                                           //创建容器对象
    //创建标签对象
    JLabel TipLabel = new JLabel("请输入要删除的书名: ", JLabel.CENTER);
    JTextField BookDeleteTextField = new JTextField(15);  //创建文本框对象
    //创建按钮对象
```

```

JButton YesBtn, ExitBtn;
JPanel panell = new JPanel(); //创建面板对象
public BookDelete() { //构造函数
    super("删除书籍信息"); //设置标题
    c = getContentPane(); //获取容器对象
    c.setLayout(new BorderLayout()); //设置布局管理器
    c.add(TipLabel, BorderLayout.NORTH); //添加对象 TipLabel 到容器中
    //添加对象 BookDeleteTextField 到容器中
    c.add(BookDeleteTextField, BorderLayout.CENTER);
    //为按钮对象赋值
    YesBtn = new JButton("确定");
    ExitBtn = new JButton("退出");
    //为按钮对象注册事件
    YesBtn.addActionListener(this);
    ExitBtn.addActionListener(this);
    //添加按钮对象到对象 panell 中
    panell.add(YesBtn);
    panell.add(ExitBtn);
    c.add(panell, BorderLayout.SOUTH); //添加对象 panell 到容器中
}

public void actionPerformed(ActionEvent e) { //事件监听器
    if (e.getSource() == ExitBtn) { //当发生事件的对象为 ExitBtn
        this.dispose(); //退出系统
    } else if (e.getSource() == YesBtn) { //当发生事件的对象为 YesBtn
        try {
            //创建实现查询功能的 SQL 语句
            String strSQL = "select borrowed_count from books where
                bookName='"
                + BookDeleteTextField.getText().trim() + "'";
            rs = db.getResult(strSQL); //获取查询结果
            if (!rs.first()) { //根据查询结果进行判断
                //显示出错信息
                JOptionPane.showMessageDialog(null, "书库里没有你要删除
                    的书!");
            } else {
                //创建实现删除功能的信息
                String strSql = "delete from books where bookName='"
                    + BookDeleteTextField.getText().trim()
                    + "' and borrowed_count=0";
                rs.first(); //设置游标
                int count = rs.getInt(1); //获取表示数目的变量
                if (!(count == 0)) {
                    //显示相应信息
                    JOptionPane.showMessageDialog(null,
                        "此书还有学生没有还! \n 现在还不能从书库中删
                        除...");
                    this.dispose(); //退出系统
                } else if (db.updateSql(strSql)) { //根据查询结果进行判断
                    //当删除成功时
                    JOptionPane.showMessageDialog(null, "删除成功!");
                    db.closeConnection();
                    this.dispose();
                } else {
                    //当删除不成功时
                    JOptionPane.showMessageDialog(null, "删除失败!");
                    db.closeConnection();
                }
            }
        } catch (Exception e1) {
            //异常处理
        }
    }
}

```

```
        this.dispose();
    }
}
} catch (SQLException sqle) {
    System.out.println(sqle.toString());
} catch (Exception ex) {
    System.out.println(ex.toString());
}
}
}
```

【代码解析】

- 上述代码的构造函数主要用来实现图书管理系统中的删除图书窗口，该用户界面涉及的具体容器、对象和布局如图 32.36 所示。
- 在上述代码的处理监听事件方法中，如果发生动作的按钮为“退出”按钮，该窗口将不显示。如果发生动作的按钮为“确定”按钮，则首先根据“图书名字”文本框的内容创建查询语句 strSQL 并执行该语句，如果为 false，则出现“书库里没有你要删除的书”的信息框，否则根据该文本框的内容创建删除已经归还图书的 strSQL 语句。最后，如果该语句的执行结果为 true，则显示“删除成功”信息框，否则显示“删除失败”信息框。

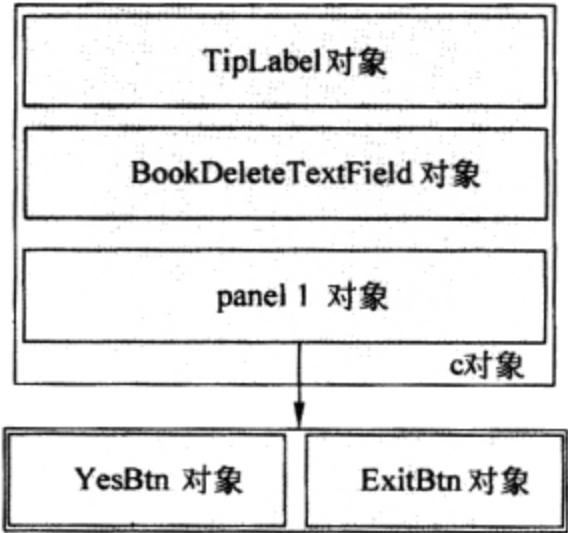


图 32.36 布局

32.3 图书管理系统项目——用户的操作

图书管理系统项目中对用户实现了 CRUD（增、删、改、查）操作，即添加用户操作、删除用户操作、修改用户信息操作和查看用户信息操作，同时还实现了用户登录的功能，各种操作所对应的类如图 32.37 所示。

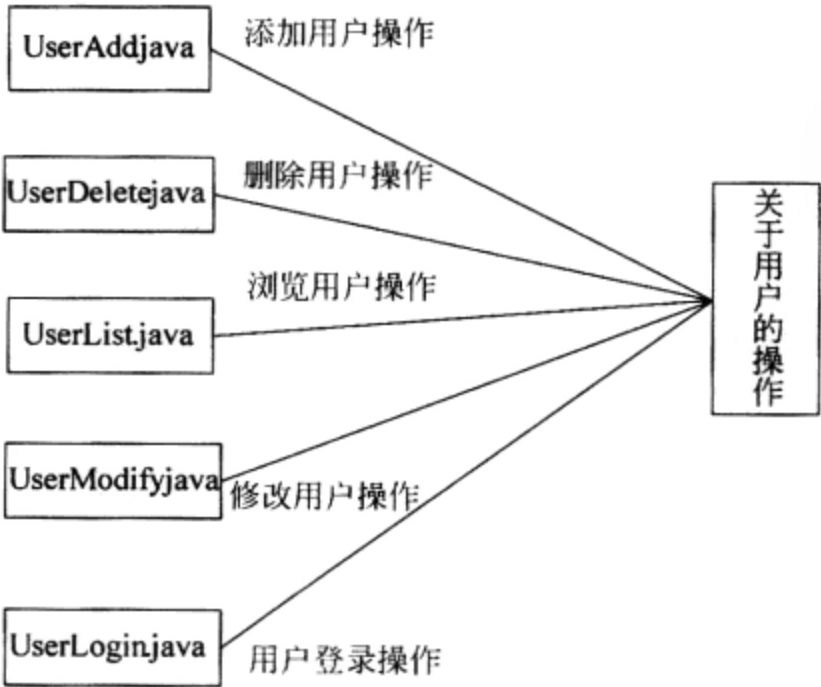


图 32.37 程序关系图

32.3.1 实现添加用户功能的类

UserAdd 类为“图书管理系统”项目中用户操作的添加用户操作，该类不仅继承了 JFrame 类，而且还实现了监听各个组件相应动作的功能，该类的类图如图 32.38 所示，具体内容如代码 32.5 所示。



图 32.38 类 UserAdd 的类图

代码 32.5 添加用户类：UserAdd.java

```

public class UserAdd extends JFrame implements ActionListener {
    //创建各种成员变量
    DataBaseManager db = new DataBaseManager(); //创建数据库连接变量
    ResultSet rs; //创建数据库集变量
    Container c; //创建容器变量
    JPanel panel1, panel2; //创建各种面板对象
    //创建各种提示标签类对象
    JLabel UserLabel, PasswordLabel, PasswordConfirmLabel, LoginPrivelegeLabel;
    JTextField UserTextField; //创建输入文本框对象
    //创建密码和确认密码对象
    JPasswordField PasswordTextField, PasswordConfirmTextField;
    JComboBox LoginPrivelegeComboBox; //创建选择框对象
    JButton AddBtn, CancelBtn; //创建按钮对象
    public UserAdd() { //构造方法
        super("添加用户");
        c = getContentPane(); //获取容器对象
        c.setLayout(new BorderLayout()); //设置容器的布局管理器
        UserLabel = new JLabel("用户名", JLabel.CENTER); //为 UserLabel 对象赋值
    }
}
  
```

```

PasswordLabel = new JLabel("密码", JLabel.CENTER);
//为 PasswordLabel 对象赋值

//为 PasswordConfirmLabel 对象赋值
PasswordConfirmLabel = new JLabel("确认密码", JLabel.CENTER);
//为 LoginPrivelegeLabel 对象赋值
LoginPrivelegeLabel = new JLabel("登录权限", JLabel.CENTER);
//为各种文本框对象赋值
UserTextField = new JTextField(10);
PasswordTextField = new JPasswordField(10);
PasswordConfirmTextField = new JPasswordField(10);
//为 LoginPrivelegeComboBox 对象赋值
LoginPrivelegeComboBox = new JComboBox();
LoginPrivelegeComboBox.addItem("系统管理员");
LoginPrivelegeComboBox.addItem("图书管理系统员");
LoginPrivelegeComboBox.addItem("借阅管理员");
//为各种按钮对象赋值
AddBtn = new JButton("添加");
CancelBtn = new JButton("取消");
AddBtn.addActionListener(this);
CancelBtn.addActionListener(this);
//为对象 panel1 赋值并添加各种对象
panel1 = new JPanel();
panel1.setLayout(new GridLayout(4, 2)); //设置布局管理器
//添加各种对象到对象 panel1 中
panel1.add(UserLabel);
... //省略部分代码
c.add(panel1, BorderLayout.CENTER); //添加对象 panel1 到容器 c 中
//为对象 panel2 赋值并添加各种对象
panel2 = new JPanel();
panel2.add(AddBtn);
panel2.add(CancelBtn);
c.add(panel2, BorderLayout.SOUTH); //添加对象 panel2 到容器 c 中
setSize(300, 300);
}

public void actionPerformed(ActionEvent e) { //注册事件
    if (e.getSource() == CancelBtn) { //当发生事件的组件为 CancelBtn 时
        db.closeConnection(); //调用 closeConnection() 方法
        this.dispose();
    } else if (e.getSource() == AddBtn) { //当发生事件的组件为 AddBtn 时
        try {
            //创建 SQL 语句
            String strSQL = "select * from userTable where userName='"
                + UserTextField.getText().trim() + "'";
            //判断 UserTextField 组件里的内容
            if (UserTextField.getText().trim().equals("")) {
                JOptionPane.showMessageDialog(null, "用户名不能为空!");
            }
            //判断 PasswordTextField 组件里的内容
            } else if (PasswordTextField.getText().trim().equals("")) {
                JOptionPane.showMessageDialog(null, "密码不能为空!");
            }
            //判断两个密码框的内容是否一致
            } else if (!PasswordTextField.getText().trim().equals(
                PasswordConfirmTextField.getText().trim())) {
                JOptionPane.showMessageDialog(null, "两次输入的密码不一致!");
            } else {
                //根据 SQL 语句的查询结果显示相应信息
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}

```



```
        if (db.getResult(strSQL).first()) {
            JOptionPane
                .showMessageDialog(null, "此用户已经存在, 请重新输入用户名!");
        } else {
            //创建插入数据库 SQL 语句
            strSQL = "insert into userTable(Username,Password,Power)
            values('"
                + UserTextField.getText().trim()
                + "','"
                + PasswordTextField.getText().trim()
                + "','"
                + LoginPrivelegeComboBox.getSelectedItem()
                + "')";
            if (db.updateSql(strSQL)) { //根据执行结果显示相应信息
                this.dispose();
                JOptionPane.showMessageDialog(null, "添加用户成功!");
            } else {
                JOptionPane.showMessageDialog(null, "添加用户失败!");
            }
        }
    }
} catch (SQLException sqle) {
    System.out.println(sqle.toString());
} catch (Exception ex) {
    System.out.println(ex.toString());
}
}
```

【代码解析】

□ 上述代码的构造函数主要用来实现了图书管理系统中的添加用户窗口，该用户界面涉及的具体容器、对象和布局如图 32.39 所示。

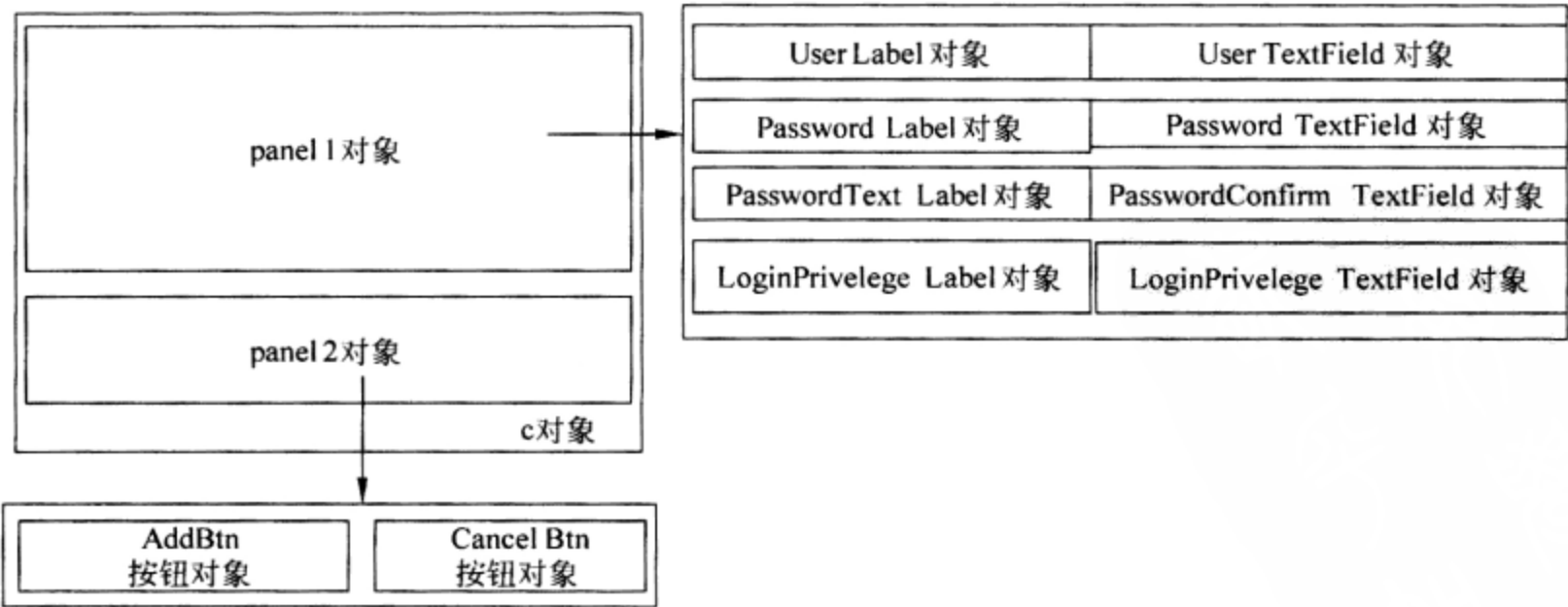


图 32.39 布局

□ 在上述代码的处理监听事件方法中，如果发生动作的按钮为“取消”按钮，该窗

口将不显示并同时关闭数据库连接；如果发生动作的按钮为“添加”按钮，首先判断“用户名”文本框的内容是否为空，以及“密码”文本框和“确认密码”文本框的内容是否一致，然后根据用户名文本框的内容查找是否已经存在该用户。如果存在，则弹出“此用户已经存在，请重新输入用户名！”的信息框，否则创建实现插入用户名和密码的 SQL 语句 strSQL。最后，如果 strSQL 语句的执行结果为 true，则显示“添加用户成功”的信息框，否则显示“添加用户失败”的信息框。

32.3.2 实现删除用户功能的类

UserDelete 类为图书管理系统项目中删除用户操作的类，该类不仅继承了 JFrame 类，而且还实现了监听各个组件相应动作的功能，该类的类图如图 32.40 所示，具体内容如代码 32.6 所示。

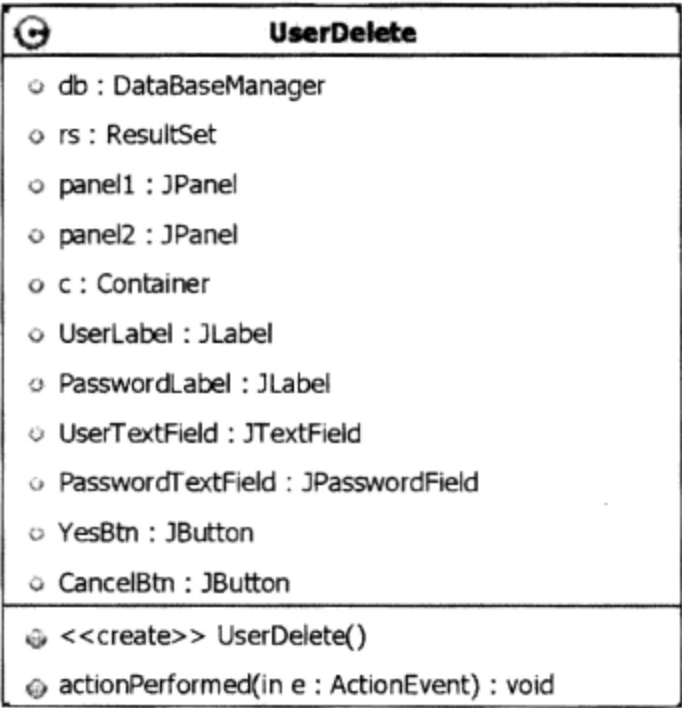


图 32.40 UserDelete 类的类图

代码 32.6 删除用户类：UserDelete.java

```
public class UserDelete extends JFrame implements ActionListener {
    //创建各种成员变量
    DataBaseManager db = new DataBaseManager();           //创建数据库操作对象
    ResultSet rs;                                           //创建数据集对象
    //创建各种面板对象
    JPanel panel1, panel2;
    Container c;                                           //创建容器对象
    //创建各种提示标签组件对象
    JLabel UserLabel, PasswordLabel;
    JTextField UserTextField;                             //创建文件框对象
    JPasswordField PasswordTextField;                     //创建密码框对象
    JButton YesBtn, CancelBtn;                             //创建按钮对象
    public UserDelete() {                                 //构造函数
        super("删除用户");                                //设置标题
    }
}
```

```

c = getContentPane(); //获取容器
c.setLayout(new BorderLayout()); //设置布局管理器
//为各种成员变量赋值
UserLabel = new JLabel("用户名", JLabel.CENTER);
PasswordLabel = new JLabel("密码", JLabel.CENTER);
UserTextField = new JTextField(10);
PasswordTextField = new JPasswordField(10);
//为按钮对象赋值
YesBtn = new JButton("确定");
CancelBtn = new JButton("取消");
//为按钮对象注册事件
YesBtn.addActionListener(this);
CancelBtn.addActionListener(this);
//为对象 panel1 赋值并添加对象
panel1 = new JPanel();
panel1.setLayout(new GridLayout(2, 2)); //设置布局管理器
//为对象 panel1 添加各种对象
panel1.add(UserLabel);
panel1.add(UserTextField);
panel1.add(PasswordLabel);
panel1.add(PasswordTextField);
//为对象 panel2 赋值并添加对象
panel2 = new JPanel();
panel2.add(YesBtn);
panel2.add(CancelBtn);
//添加对象 panel1 和 panel2 到容器 c 中
c.add(panel1, BorderLayout.CENTER);
c.add(panel2, BorderLayout.SOUTH);
setSize(300, 300); //设置对象大小
}
public void actionPerformed(ActionEvent e) { //创建事件监听器
    try {
        if (e.getSource() == CancelBtn) { //发生事件的对象为 CancelBtn
            db.closeConnection(); //调用 closeConnection() 方法
            this.dispose();
        } else if (e.getSource() == YesBtn) { //发生事件的对象为 YesBtn
            //获取密码框内容
            char[] password = PasswordTextField.getPassword();
            //创建密码字符串
            String passwordSTR = new String(password);
            //创建查询 SQL 语句
            String strSQL = "select * from userTable where userName='"
                + UserTextField.getText().trim() + "' and "
                + "password='"
                + passwordSTR + "'";
            //判断用户名
            if (UserTextField.getText().trim().equals("")) {
                JOptionPane.showMessageDialog(null, "用户名不能为空!");
            } //判断密码
            } else if (PasswordTextField.equals("")) {
                JOptionPane.showMessageDialog(null, "密码不能为空!");
            } else if (db.getResult(strSQL).first()) {
                //创建删除 SQL 语句
                strSQL = "delete from userTable where userName='"
                    + UserTextField.getText().trim() + "'";
                //判断是否删除成功
                if (db.updateSql(strSQL)) {

```

```

        JOptionPane.showMessageDialog(null, "删除成功!");
        this.dispose();
    } else {
        JOptionPane.showMessageDialog(null, "删除失败!");
        this.dispose();
    }
    db.closeConnection(); //调用关闭连接方法
} else {
    JOptionPane.showMessageDialog(null, "不存在此用户或者密码错误!");
}
}
} catch (SQLException sqle) {
    System.out.println(sqle.toString());
} catch (Exception ex) {
    System.out.println(ex.toString());
}
}
}

```

【代码解析】

- 上述代码的构造函数主要用来实现图书管理系统中的删除用户窗口，该用户界面涉及的具体容器、对象和布局如图 32.41 所示。

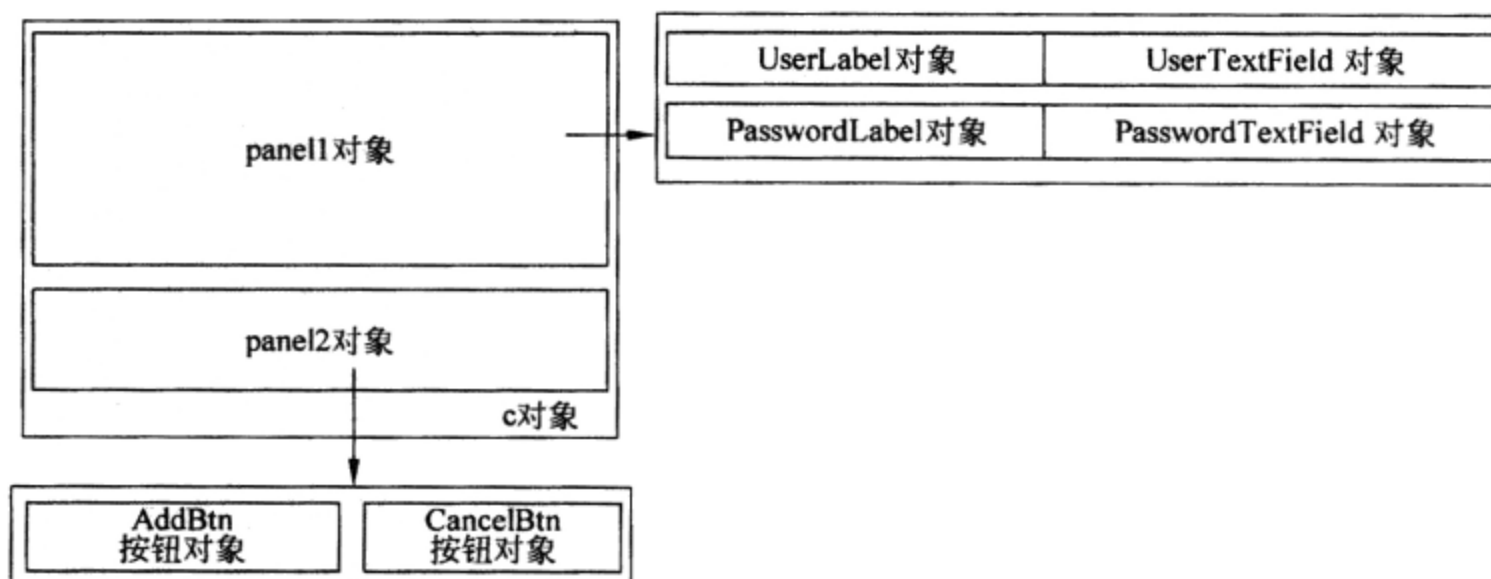


图 32.41 布局

- 在上述代码的处理监听事件方法中，如果发生动作的按钮为“取消”按钮，该窗口将不显示并关闭该数据库连接。如果发生动作的按钮为“确定”按钮，则首先判断“用户名”文本框和“密码”框的内容是否为空，同时根据这两个文本框创建查询的 SQL 语句 strSQL，然后执行该 SQL 语句。如果执行结果不为空，则根据“用户名”文本框创建 SQL 删除语句 strSQL。最后，如果 strSQL 的执行结果为 true，则显示“删除成功”信息框，否则显示“删除失败”信息框。

32.3.3 实现修改用户功能的类

UserModify 类为图书管理系统项目中的修改用户操作的类，同样该类不仅继承了 JFrame 类，而且还实现了监听各个组件相应动作的功能，该类的类图如图 32.42 所示，具体内容如代码 32.7 所示。

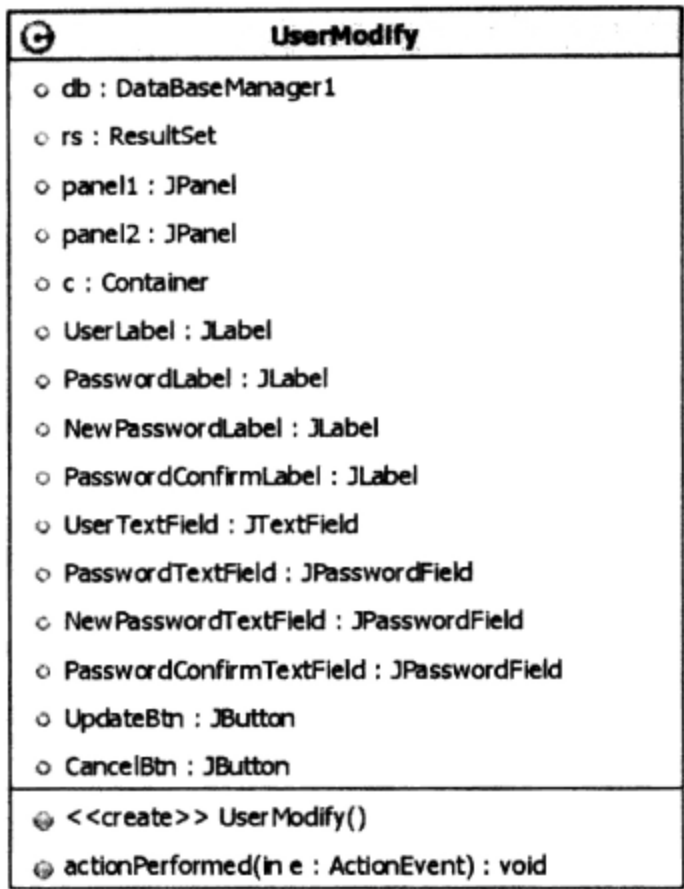


图 32.42 UserModify 类的类图

代码 32.7 修改用户类：UserModify.java

```
public class UserModify extends JFrame implements ActionListener {
    //创建各种成员变量
    DataBaseManager db = new DataBaseManager();           //创建数据库操作对象
    ResultSet rs;                                         //创建数据集对象
    //创建各种面板对象
    JPanel panel1, panel2;
    Container c;                                           //创建容器对象
    //创建各种标签组件对象
    JLabel UserLabel, PasswordLabel, NewPasswordLabel, PasswordConfirmLabel;
    JTextField UserTextField;                             //创建文本框对象
    //创建各种密码框对象
    JPasswordField PasswordTextField, NewPasswordTextField,
        PasswordConfirmTextField;
    JButton UpdateBtn, CancelBtn;                         //创建按钮对象
    public UserModify() {                                 //构造函数
        super("更改密码");
        c = getContentPane();
        c.setLayout(new BorderLayout());
        UserLabel = new JLabel("用户名", JLabel.CENTER);
        PasswordLabel = new JLabel("原密码", JLabel.CENTER);
        NewPasswordLabel = new JLabel("新密码", JLabel.CENTER);
        PasswordConfirmLabel = new JLabel("确认新密码", JLabel.CENTER);
        UserTextField = new JTextField(10);
        PasswordTextField = new JPasswordField(10);
        NewPasswordTextField = new JPasswordField(10);
        PasswordConfirmTextField = new JPasswordField(10);
        UpdateBtn = new JButton("更新");
        CancelBtn = new JButton("取消");
        UpdateBtn.addActionListener(this);
        CancelBtn.addActionListener(this);
    }
}
```

```

panel1 = new JPanel();
panel1.setLayout(new GridLayout(4, 2));
...
panel2 = new JPanel();
panel2.add(UpdateBtn);
panel2.add(CancelBtn);
c.add(panel1, BorderLayout.CENTER);
c.add(panel2, BorderLayout.SOUTH);
setSize(300, 300);
}
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == CancelBtn) {
        db.closeConnection();
        this.dispose();
    } else if (e.getSource() == UpdateBtn) {
        try {
            char[] password = PasswordTextField.getPassword();
            String passwordSTR = new String(password);
            char[] newPassword = NewPasswordTextField.getPassword();
            String newPasswordSTR = new String(newPassword);
            char[] confirmPassword = PasswordConfirmTextField.
            getPassword();
            String confirmPasswordSTR = new String(confirmPassword);
            String strSQL = "select * from userTable where userName='"
                + UserTextField.getText().trim() + "'and
                password='"
                + passwordSTR + "'";
            if (UserTextField.getText().trim().equals("")) {
                JOptionPane.showMessageDialog(null, "用户名不能为空!");
            } else if (passwordSTR.equals("")) {
                JOptionPane.showMessageDialog(null, "原密码不能为空!");
            } else if (!newPasswordSTR.equals(confirmPasswordSTR)) {
                JOptionPane.showMessageDialog(null, "两次输入的新密码不
                一致!");
            } else {
                if (!db.getResult(strSQL).first()) {
                    JOptionPane.showMessageDialog(null, "此用户不存在或
                    者原密码不正确!");
                } else {
                    strSQL = "update userTable set password='"
                        + newPasswordSTR + "'where userName='"
                        + UserTextField.getText().trim() + "'";
                    if (db.updateSql(strSQL)) {
                        JOptionPane.showMessageDialog(null, "更新密码成
                        功!");
                        this.dispose();
                    } else {
                        JOptionPane.showMessageDialog(null, "更新密码失
                        败!");
                        this.dispose();
                    }
                }
                db.closeConnection();
            }
        } catch (SQLException sqle) {
            System.out.println(sqle.toString());
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}

```

//省略部分代码


```
    }  
    }  
}
```

【代码解析】

□ 上述代码的构造函数主要用来实现图书管理系统中的修改用户信息窗口，该用户界面涉及的具体容器、对象和布局如图 32.43 所示。

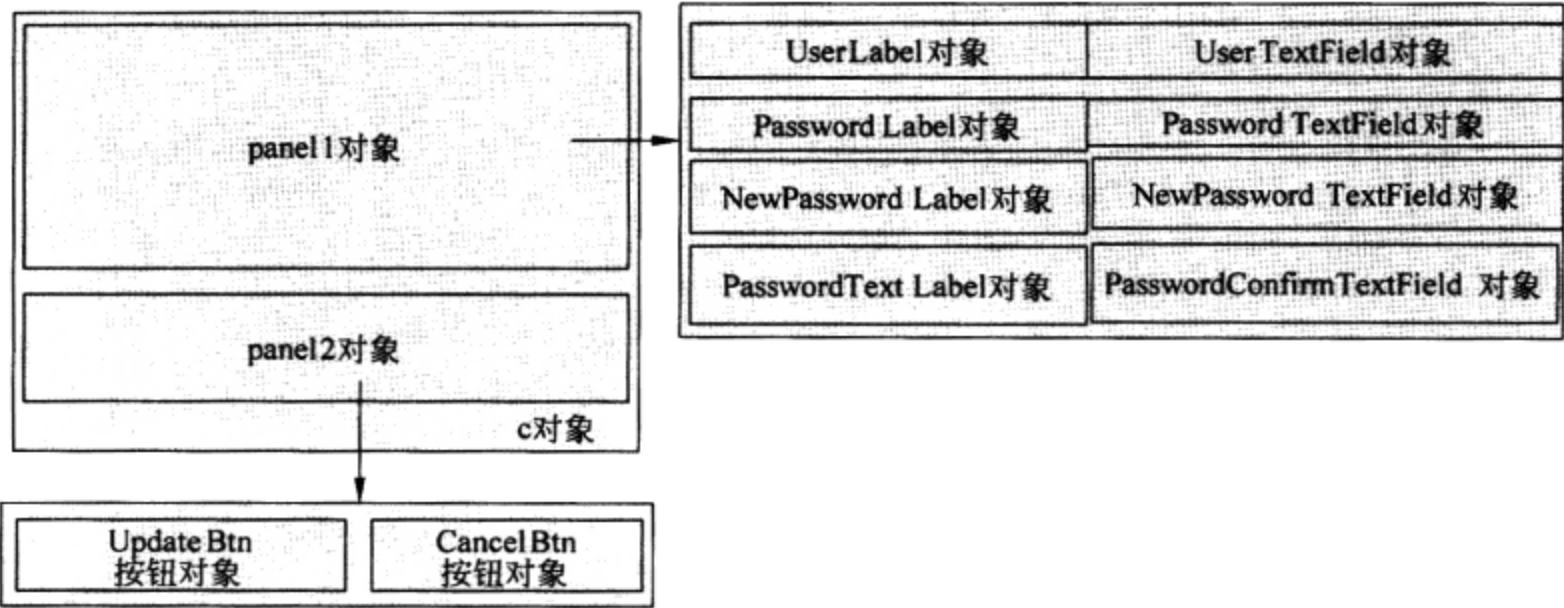


图 32.43 布局

□ 在上述代码的处理监听事件方法中，如果发生动作的按钮为“取消”按钮，该窗口将不显示同时关闭数据库连接。如果发生动作的按钮为“更新”按钮，则首先判断“用户名”输入框和“密码”框是否为空，同时判断新密码和确认密码是否相同，如果符合要求，则根据用户名输入框和密码框内容创建查询的 SQL 语句 strSQL。然后判断 strSQL 的执行结果，如果为空，则弹出“此用户不存在或者原密码不正确！”的信息框，否则根据“用户名”文本框和“新密码”文本框的内容创建更新语句。最后执行更新语句，如果执行结果为 true，则显示“更新密码成功”的信息框，否则显示“更新密码失败”的信息框。

32.3.4 实现用户登录功能的类

UserLogin 类为图书管理系统项目中用户登录操作的类，该类不仅继承了 JFrame 类，而且还实现了监听各个组件相应动作的功能，该类的类图如图 32.44 所示，具体内容如代码 32.8 所示。

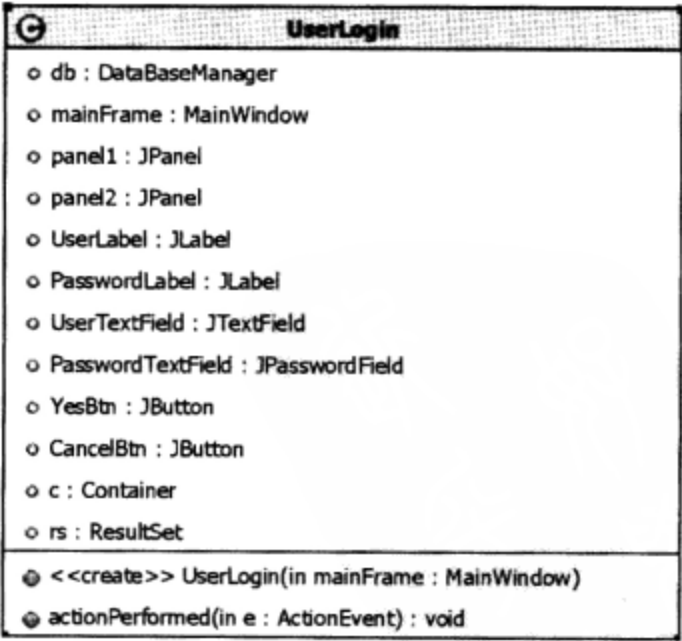


图 32.44 UserLogin 类的类图

代码 32.8 用户登录的类：UserLogin.java

```
public class UserLogin extends JFrame implements ActionListener {  
    //创建各种成员变量
```

```

DataBaseManager db = new DataBaseManager();           //创建数据库操作对象
Container c;                                           //创建容器对象
ResultSet rs;                                         //创建数据集对象
MainWindow mainFrame;                                //创建主窗口
//创建面板对象
JPanel panel1, panel2;
//创建标签对象
JLabel UserLabel, PasswordLabel;
JTextField UserTextField;                             //创建文本输入框对象
JPasswordField PasswordTextField;                    //创建密码框对象
//创建按钮对象
JButton YesBtn, CancelBtn;
public UserLogin(MainWindow mainFrame) {              //带参构造函数
    super("用户登录");                                //设置标题
    this.mainFrame = mainFrame;                       //为变量 mainFrame 赋值
    //为标签对象赋值
    UserLabel = new JLabel("用户名", JLabel.CENTER);
    PasswordLabel = new JLabel("密码", JLabel.CENTER);
    //为输入框对象赋值
    UserTextField = new JTextField(10);
    PasswordTextField = new JPasswordField(10);
    //为按钮对象赋值
    YesBtn = new JButton("确定");
    CancelBtn = new JButton("取消");
    //为按钮对象注册事件
    YesBtn.addActionListener(this);
    CancelBtn.addActionListener(this);
    //为对象 c 赋值并设置布局管理器
    c = getContentPane();
    c.setLayout(new BorderLayout());
    //创建和设置对象 panel1
    panel1 = new JPanel();                             //创建面板对象
    panel1.setLayout(new GridLayout(2, 2));            //设置布局管理器
    //添加各种对象到对象 panel1
    panel1.add(UserLabel);
    panel1.add(UserTextField);
    panel1.add(PasswordLabel);
    panel1.add(PasswordTextField);
    c.add(panel1, BorderLayout.CENTER);                 //添加对象 panel1 到容器 c 中
    //创建和设置对象 panel2
    panel2 = new JPanel();                             //创建面板对象
    //添加按钮对象到面板 panel2
    panel2.add(YesBtn);
    panel2.add(CancelBtn);
    c.add(panel2, BorderLayout.SOUTH);                 //添加对象 panel2 到容器 c 中
    setSize(300, 300);                                //设置窗口大小
}
public void actionPerformed(ActionEvent e) {          //事件监听器
    if (e.getSource() == CancelBtn) {                //当发生事件的组件为 CancelBtn
        mainFrame.setEnabled("else");
        this.dispose();                               //退出系统
    } else {                                           //当发生事件的组件为 YesBtn
        //创建密码字符数组
        char[] password = PasswordTextField.getPassword();
        String passwordSTR = new String(password);    //将字符数组转换成字符串
    }
}

```

```

//判断用户名是否为空
if (UserTextField.getText().trim().equals("")) {
    JOptionPane.showMessageDialog(null, "用户名不可为空!");
    return;
}
//判断密码是否为空
if (passwordSTR.equals("")) {
    JOptionPane.showMessageDialog(null, "密码不可为空!");
    return;
}
String strSQL; //定义查询的字符串变量
//为变量 strSQL 赋值
strSQL = "select * from userTable where UserName='"
        + UserTextField.getText().trim() + "'and Password='"
        + passwordSTR + "'";
rs = db.getResult(strSQL); //获取执行结果
boolean isExist = false; //创建一个布尔型变量
try {
    isExist = rs.first(); //为变量 isExist 赋值
} catch (SQLException sqle) {
    System.out.println(sqle.toString());
}
if (!isExist) { //当无此用户名和密码时
    //显示出错信息
    JOptionPane.showMessageDialog(null, "用户名不存在或者密码不正确!");
    mainFrame.setEnabled("else");
} else {
    try {
        rs.first();
        mainFrame.setEnabled(rs.getString("power").trim());
        db.closeConnection(); //关闭数据库连接
        this.dispose();
    } catch (SQLException sqle2) {
        System.out.println(sqle2.toString());
    }
}
}
}
}

```

【代码解析】

- 上述代码的构造函数主要用来实现图书管理系统中的用户登录窗口，该用户界面涉及的具体容器、对象和布局如图 32.45 所示。
- 在上述代码的处理监听事件方法中，如果发生动作的按钮为“取消”按钮，该窗口将不显示同时主窗口对象获取焦点。如果发生动作的按钮为“确定”按钮，则首先判断“用户名”文本框和“密码”文本框是否为空，如果符合要求，则根据“用户名”文本框和“密码”文本框的内容创建查询的 SQL 语句 strSQL。然后判断 strSQL 的执行结果，如果为空，则弹出“此用户不存在或者原密码不正确！”的信息框，否则根据执行结果获取该用户的权限，设置主窗口的可用组件并同时使该窗口消失。

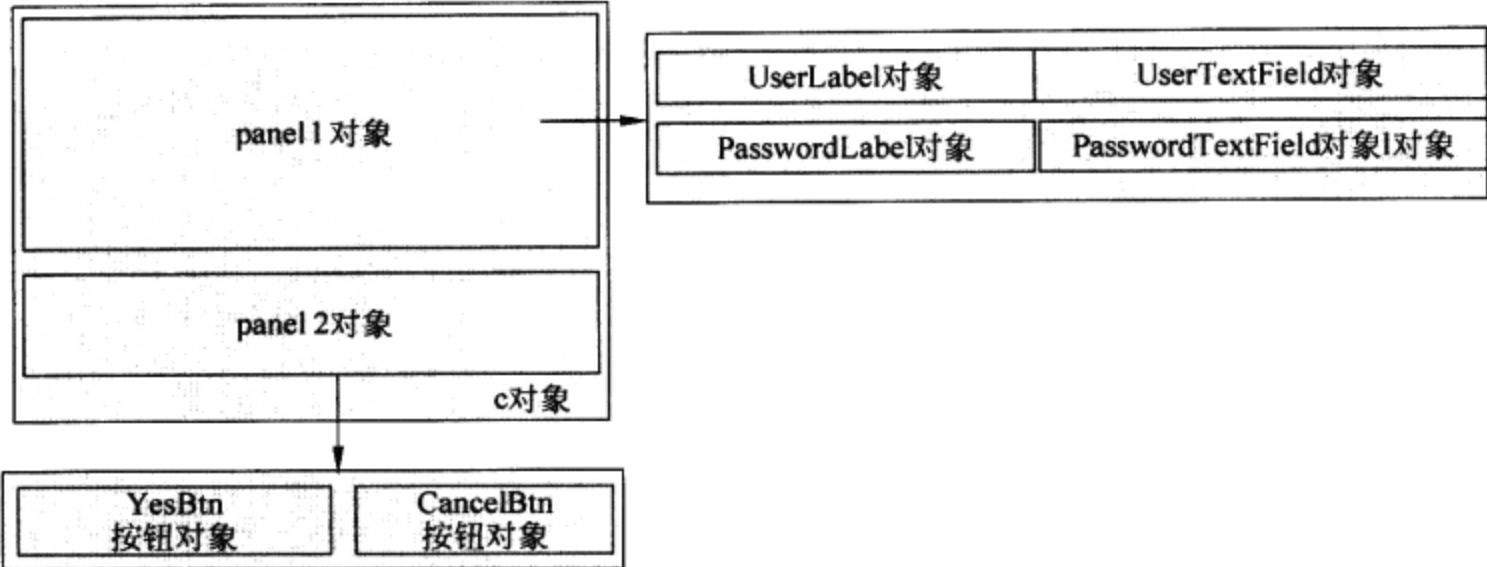


图 32.45 布局

32.3.5 实现用户列表功能的类

UserList 类为图书管理系统项目中显示用户信息操作的类，该类不仅继承了 JFrame 类，而且还实现了监听各个组件相应动作的功能，该类的类图如图 32.46 所示，具体内容如代码 32.9 所示。

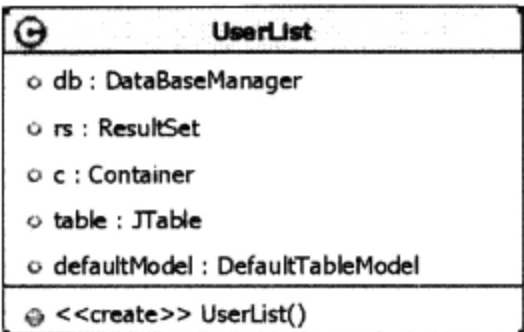


图 32.46 UserList 类的类图

代码 32.9 用户列表的类: UserList.java

```
public class UserList extends JFrame {
    //创建成员变量
    DataBaseManager db = new DataBaseManager();
    ResultSet rs;
    Container c;
    JTable table = null;
    DefaultTableModel defaultModel = null;
    public UserList() {
        super("用户列表一览!");
        c = getContentPane();
        c.setLayout(new BorderLayout());
        //创建用户名和权限的数组
        String[] name = { "用户名", "权限" };
        String[][] data = new String[0][0];
        defaultModel = new DefaultTableModel(data, name); //为表格模式赋值
        //为表格对象 table 赋值
        table = new JTable(defaultModel);
        //设置表格对象 table
        table.setPreferredSize(new Dimension(400, 80));
        JScrollPane s = new JScrollPane(table); //创建滚动面板对象 s
        c.add(s); //添加对象 s 到容器 c 中
        try {
            //创建实现查询功能的 SQL 语句
            String strSql = "select userName,power from userTable";
            rs = db.getResult(strSql); //获取执行结果
            while (rs.next()) { //遍历执行结果
```

```

        Vector insertRow = new Vector();    //创建集合对象 insertRow
        //为集合对象 insertRow 添加数据
        insertRow.addElement(rs.getString(1));
        insertRow.addElement(rs.getString(2));
        defaultModel.addRow(insertRow);    //设置表格模式
    }
    table.revalidate();                    //更新表格
} catch (SQLException sqle) {
    System.out.println(sqle.toString());
} catch (Exception ex) {
    System.out.println(ex.toString());
}
}
}

```

【代码解析】

□ 上述代码的构造函数主要用来实现图书管理系统中的浏览用户信息窗口,该用户界面涉及的具体容器、对象和布局如图 32.47 所示。

□ 在上述代码中对于表格 table 中的数据,首先创建表格 userTable 的查询语句,然后执行该 SQL 语句,最后把执行结果通过循环存储到集合 insertRow 中并把该集合作为 defaultModel 的数据。

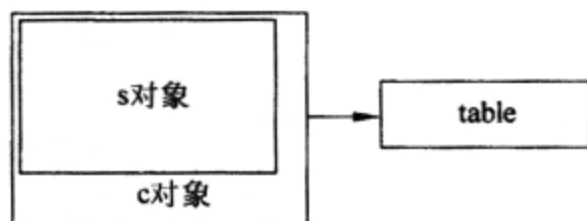


图 32.47 布局

32.4 图书管理系统项目——出借图书的操作

图书管理系统项目中为了实现借书的功能,分别实现了图书出借类、修改 s 出借图书信息类和浏览出借图书信息的类,各种操作所对应的类如图 32.48 所示。

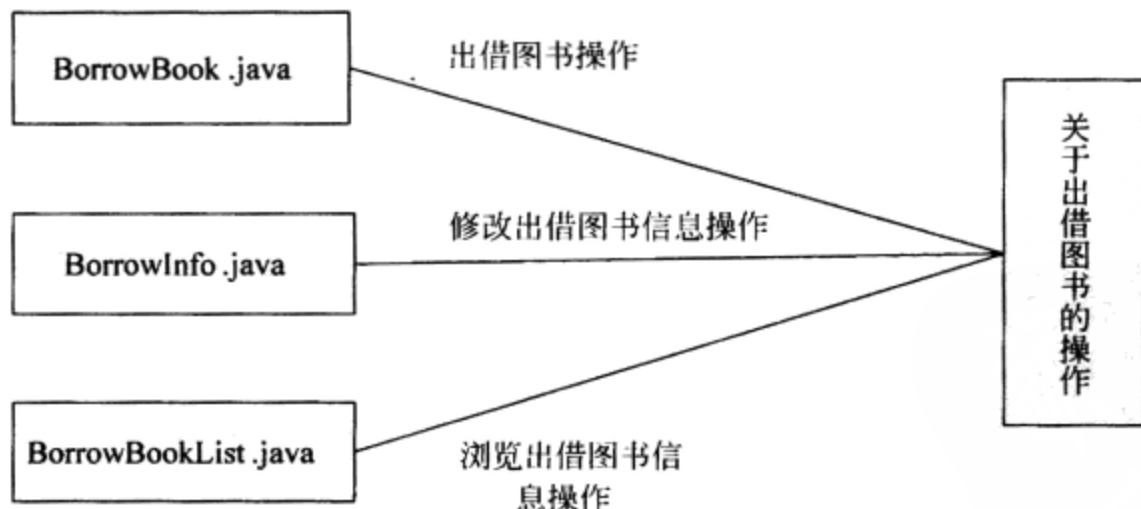


图 32.48 程序关系图

32.4.1 出借图书操作的类

BorrowBook 类为图书管理系统项目中实现出借图书操作的类,该类不仅继承了 JFrame 类,而且还实现了监听各个组件相应动作的功能,该类的类图如图 32.49 所示,具体内容

如代码 32.10 所示。



图 32.49 BorrowBook 类的类图

代码 32.10 借书方法: BorrowBook.java

```
public class BorrowBook extends JFrame implements ActionListener {
    //创建各种成员变量
    DataBaseManager db = new DataBaseManager(); //创建数据库操作对象
    ResultSet rs; //数据集对象
    //创建面板对象
    JPanel panel1, panel2;
    Container c; //创建容器对象
    //创建各种标签对象
    JLabel BorrowedBookStudentLabel, BorrowedBookNameLabel, BorrowedDateLabel,
        ReturnDateLabel, BorrowedCommentLabel;
    //创建各种输入文本框对象
    JTextField BorrowedBookStudentTextField, BorrowedDateTextField,
        ReturnDateTextField, BorrowedCommentTextField;
    //创建各种按钮对象
    JButton ClearBtn, YesBtn, CancelBtn;
    JComboBox BookNameComboBox = new JComboBox(); //创建选择框对象
    public BorrowBook() { //构造函数
        super("图书出借"); //设置标题
        c = getContentPane(); //为容器对象 c 赋值
        c.setLayout(new BorderLayout()); //设置布局管理器
        //为各种标签对象和输入文本框对象赋值
        BorrowedBookStudentLabel = new JLabel("借阅者姓名", JLabel.CENTER);
        BorrowedBookNameLabel = new JLabel("书名", JLabel.CENTER);
```



```

BorrowedDateLabel = new JLabel("借阅日期", JLabel.CENTER);
ReturnDateLabel = new JLabel("归还日期", JLabel.CENTER);
BorrowedCommentLabel = new JLabel("备注", JLabel.CENTER);
BorrowedBookStudentTextField = new JTextField(15);
BorrowedDateTextField = new JTextField(15);
ReturnDateTextField = new JTextField(15);
BorrowedCommentTextField = new JTextField(15);
try {
    //创建查询的 SQL 语句
    String strSQL = "select bookName from books where books_
count>borrowed_count";
    rs = db.getResult(strSQL);           //获取到执行 SQL 语句的结果
    //遍历执行结果
    while (rs.next()) {
        //添加信息到下拉列表
        BookNameComboBox.addItem(rs.getString(1));
    }
} catch (SQLException sqle) {
    System.out.println(sqle.toString());
} catch (Exception ex) {
    System.out.println(ex.toString());
}
//创建和设置面板对象 panel1
panel1 = new JPanel();                 //为对象 panel1 赋值
panel1.setLayout(new GridLayout(5, 2)); //设置面板布局管理
//添加各种对象到面板 panel1 中
panel1.add(BorrowedBookStudentLabel);
...                                   //省略部分代码
c.add(panel1, BorderLayout.CENTER);    //添加 panel1 到容器 c 中
//创建和设置面板对象 panel2
panel2 = new JPanel();                 //为对象 panel2 赋值
panel2.setLayout(new GridLayout(1, 3)); //设置面板布局管理
//为按钮对象赋值
ClearBtn = new JButton("清空");
YesBtn = new JButton("确定");
CancelBtn = new JButton("取消");
//为按钮对象注册事件监听器
ClearBtn.addActionListener(this);
YesBtn.addActionListener(this);
CancelBtn.addActionListener(this);
//添加各种按钮对象到 panel2 对象中
panel2.add(ClearBtn);
panel2.add(YesBtn);
panel2.add(CancelBtn);
c.add(panel2, BorderLayout.SOUTH);      //添加对象 panel2 到容器 c 中
}

public void actionPerformed(ActionEvent e) { //事件监听器
    if (e.getSource() == CancelBtn) { //发生事件的组件为 CancelBtn
        this.dispose();               //退出
    } else if (e.getSource() == ClearBtn) { //发生事件的组件为 ClearBtn
        //清空相应组件的内容
        BorrowedBookStudentTextField.setText("");
        BorrowedDateTextField.setText("");
        BorrowedCommentTextField.setText("");
    } else if (e.getSource() == YesBtn) { //发生事件的组件为 YesBtn
        //判断相应输入框中的内容是否为空
        if (BorrowedBookStudentTextField.getText().trim().equals("")) {

```

```

        JOptionPane.showMessageDialog(null, "请输入借阅者的姓名...");
    } else if (BookNameComboBox.getSelectedItem().equals("")) {
        JOptionPane.showMessageDialog(null, "对不起,现在书库里没有书,\n你现在不能借书!");
    } else {
        try {
            //创建插入信息的 SQL 语句
            String strSQL = "insert into bookBrowse(studentname,bookname,borrowdate,returndate,com) values('"
                + BorrowedBookStudentTextField.getText().trim()
                + "','"
                + BookNameComboBox.getSelectedItem()
                + "','"
                + BorrowedDateTextField.getText().trim()
                + "','"
                + ReturnDateTextField.getText().trim()
                + "','"
                + BorrowedCommentTextField.getText().trim()
                + "')";
            if (db.updateSql(strSQL)) { //根据执行结果判断是否借阅成功
                //显示相应的信息
                JOptionPane.showMessageDialog(null, "借阅完成!");
                //更新数据库信息
                strSQL = "update books set borrowed_count=
                    borrowed_count+1 where bookname='" + BookNameComboBox.getSelectedItem() + "'";
                db.updateSql(strSQL); //执行 SQL 语句
                db.closeConnection(); //关闭连接
                this.dispose();
            } else { //当借阅不成功时
                //显示借阅失败信息
                JOptionPane.showMessageDialog(null, "借阅失败!");
                db.closeConnection();
                this.dispose();
            }
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}
}
}
}

```

【代码解析】

- ❑ 上述代码的构造函数主要用来实现图书管理系统中的出借图书窗口，该用户界面涉及的具体容器、对象和布局如图 32.50 所示。
- ❑ 在上述代码的处理监听事件方法中，如果发生动作的按钮为“取消”按钮，该窗口将不显示。如果发生动作的按钮为“清空”按钮，该窗口中所有文本输入框的内容为空。如果发生动作的按钮为“确定”按钮，首先判断“借阅者”文本框的内容及书库中的书是否为空，然后根据各个文本框的内容创建 SQL 插入语句 strSQL 并执行该语句。最后，如果插入 SQL 语句的执行结果为 true，则显示“借阅完成”的信息框，同时更新 books 表格中的数据，否则显示“借阅失败”的信息框。

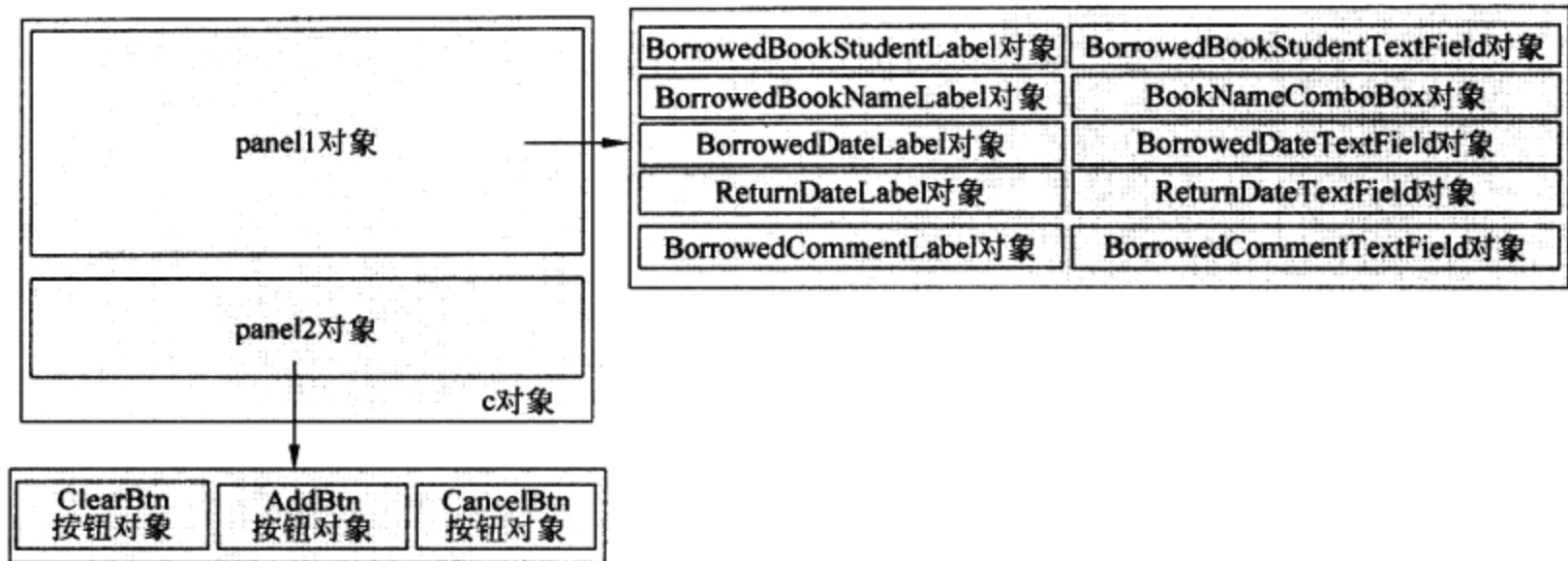


图 32.50 布局

32.4.2 借书列表方法

BorrowBookList 类为图书管理系统项目中，实现图书出借操作的查看出借图书列表操作的类，该类不仅继承了 JFrame 类，而且还实现了监听各个组件相应动作的功能，该类的类图如图 32.51 所示，具体内容如代码 32.11 所示。

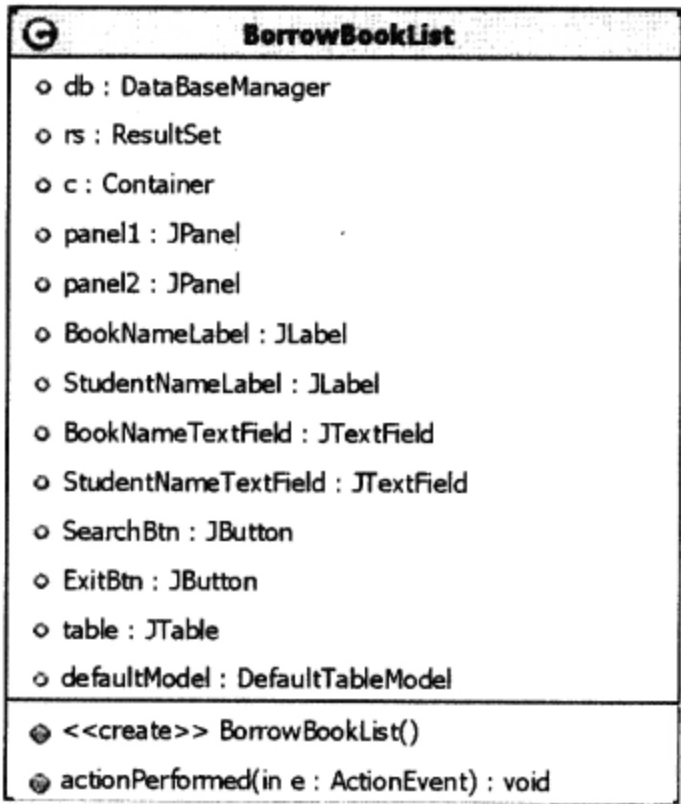


图 32.51 BorrowBookList 类的类图

代码 32.11 查看出借图书列表方法: BorrowBookList.java

```
public class BorrowBookList extends JFrame implements ActionListener {
    //创建各种成员变量
    DataBaseManager db = new DataBaseManager(); //创建操作数据库对象
    ResultSet rs; //创建数据集对象
    Container c; //创建容器对象
    //创建各种面板对象
```

```

JPanel panel1, panel2;
//创建各种标签对象
JLabel BookNameLabel, StudentNameLabel;
//创建各种文本框对象
JTextField BookNameTextField, StudentNameTextField;
//创建各种按钮对象
JButton SearchBtn, ExitBtn;
JTable table = null;                                //创建表格对象
DefaultTableModel defaultModel = null;              //创建表格模式
public BorrowBookList() {                            //构造函数
    super("图书借阅一览!");                          //设置标题
    c = getContentPane();                            //为对象 c 赋值
    c.setLayout(new BorderLayout());                  //设置对象 c 的布局管理器
    //为标签对象和文本框对象赋值
    BookNameLabel = new JLabel("书名", JLabel.CENTER);
    StudentNameLabel = new JLabel("借阅者", JLabel.CENTER);
    BookNameTextField = new JTextField(15);
    StudentNameTextField = new JTextField(15);
    //为按钮对象赋值
    SearchBtn = new JButton("查询");
    ExitBtn = new JButton("退出");
    //为按钮对象注册事件
    SearchBtn.addActionListener(this);
    ExitBtn.addActionListener(this);
    //创建和设置 box1 容器对象
    Box box1 = Box.createHorizontalBox();              //创建 box1 容器对象
    //向 box1 对象添加对象
    box1.add(StudentNameLabel);
    box1.add(StudentNameTextField);
    box1.add(SearchBtn);
    //创建和设置 box2 容器对象
    Box box2 = Box.createHorizontalBox();              //创建 box2 容器对象
    //向 box2 对象添加对象
    box2.add(BookNameLabel);
    box2.add(BookNameTextField);
    box2.add(ExitBtn);
    //创建和设置 boxH 容器对象
    Box boxH = Box.createVerticalBox();                //创建 box2 容器对象
    //向 boxH 对象添加对象
    boxH.add(box1);
    boxH.add(box2);
    boxH.add(Box.createVerticalGlue());
    //创建和设置 panel1 对象
    panel1 = new JPanel();                            //为 panel1 对象赋值
    panel1.add(boxH);                                //添加对象 boxH 到 panel1 对象中
    panel2 = new JPanel();                            //为 panel2 对象赋值
    //创建名字字符串数组
    String[] name = { "借阅者", "书名", "借阅日期", "还入日期", "备注" };
    String[][] data = new String[0][0];              //创建时间的数组
    //为对象 defaultModel 赋值
    defaultModel = new DefaultTableModel(data, name);
    table = new JTable(defaultModel);                //为对象 table 赋值
    //设置对象 table
    table.setPreferredScrollableViewportSize(new Dimension(400, 80));
    JScrollPane s = new JScrollPane(table);          //创建 table 对象的滚动面板

```

```

panel2.add(s); //添加对象 s 到面板 panel2 中
//添加对象 panel1 和 panel2 到容器 c 中
c.add(panel1, BorderLayout.NORTH);
c.add(panel2, BorderLayout.SOUTH);
}
public void actionPerformed(ActionEvent e) { //事件监听器
    if (e.getSource() == ExitBtn) { //发生事件的组件为 ExitBtn 时
        db.closeConnection(); //关闭数据库连接对象
        this.dispose(); //退出系统
    } else if (e.getSource() == SearchBtn) { //发生事件的组件为 SearchBtn 时
        //创建查询的 SQL 语句
        String strSQL = "select studentname,bookname, borrowdate,
        returndate,com from bookbrowse";
        String strSql = null; //创建字符串对象
        //设置具体的实现查询功能的 SQL 语句
        if (StudentNameTextField.getText().trim().equals("")
            && BookNameTextField.getText().trim().equals("")) {
            //当学生名和书名为空时
            strSql = strSQL;
        } else if (StudentNameTextField.getText().trim().equals("")) {
            //当学生名为空时
            strSql = strSQL + " where bookName='"
                + BookNameTextField.getText().trim() + "'";
        } else if (BookNameTextField.getText().trim().equals("")) {
            //当书名为空时
            strSql = strSQL + " where studentName='"
                + StudentNameTextField.getText().trim() + "'";
        } else {
            strSql = strSQL + " where studentName='"
                + StudentNameTextField.getText().trim()
                + "'and bookName='"
                + BookNameTextField.getText().trim() + "'";
        }
    }
    try { //设置 table 对象中的数据
        //删除 table 对象中的数据
        int rowCount = defaultModel.getRowCount() - 1;
        //获取 table 对象中的数据行
        int j = rowCount; //为变量 j 赋值
        //通过遍历删除数据
        for (int i = 0; i <= rowCount; i++) {
            defaultModel.removeRow(j); //删除 rowCount 行的数据
            defaultModel.setRowCount(j); //重新设置行数
            j = j - 1;
        }
        rs = db.getResult(strSql); //获取执行结果
        //通过遍历把执行结果存储到对象 data 中
        while (rs.next()) {
            Vector data = new Vector();
            data.addElement(rs.getString(1));
            data.addElement(rs.getString(2));
            data.addElement(rs.getString(3));
            data.addElement(rs.getString(4));
            data.addElement(rs.getString(5));
            defaultModel.addRow(data); //设置对象 defaultModel
        }
        table.revalidate(); //刷新对象 table
    } catch (SQLException sqle) {

```

```

        System.out.println(sqle.toString());
    } catch (Exception ex) {
        System.out.println(ex.toString());
    }
}
}
}

```

【代码解析】

- 上述代码的构造函数主要用来实现图书管理系统中的查看出借图书信息窗口，该用户界面涉及的具体容器、对象和布局如图 32.52 所示。

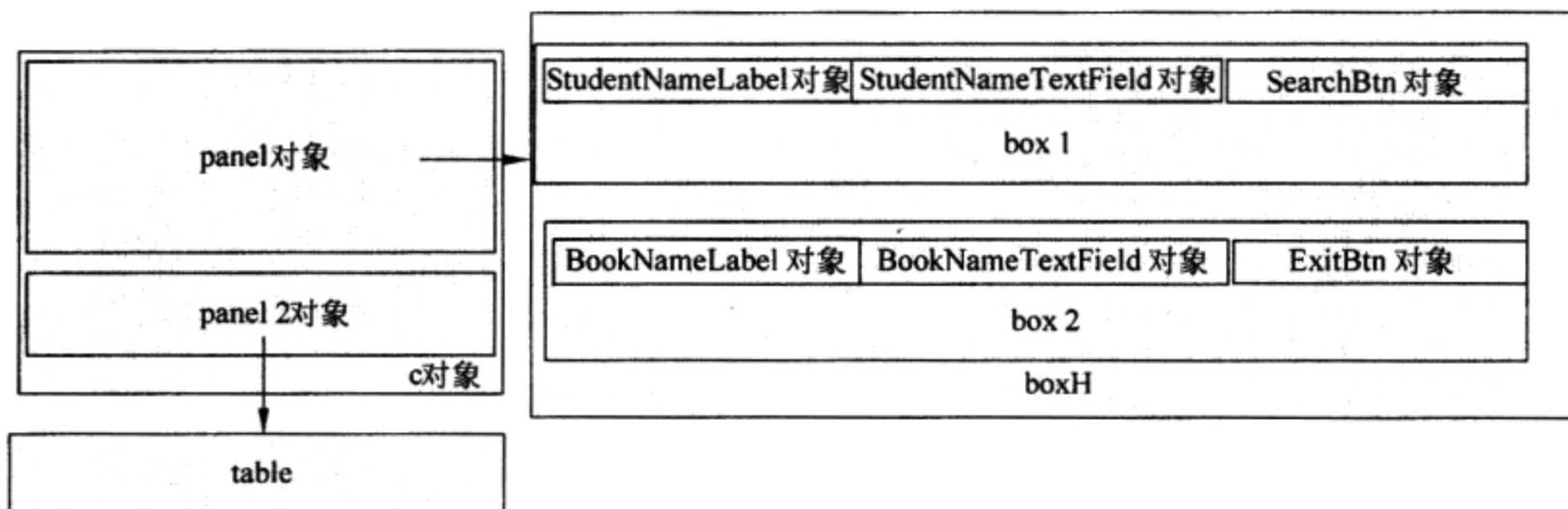


图 32.52 布局

- 在上述代码的处理监听事件方法中，如果发生动作的按钮为“退出”按钮，该窗口将不显示。如果发生动作的按钮为“查询”按钮，首先根据文本框的内容是否为空设置查询 SQL 语句 strSQL，然后把执行该 SQL 语句的结果存储到集合 data 中，最后如果再删除表格对象 table 中的值，同时并把集合 data 的值在该表格中显示出来。

32.4.3 修改出借图书信息方法

BorrowInfo 类为图书管理系统项目中，实现图书出借操作的修改出借图书信息操作的类，该类不仅继承了 JFrame 类，而且还实现了监听各个组件相应动作的功能，该类的类图如图 32.53 所示，具体内容如代码 32.12 所示。

代码 32.12 修改出借图书信息：BorrowInfo.java

```

public class BorrowInfo extends JFrame implements ActionListener {
    //创建成员变量
    DataBaseManager db = new DataBaseManager();           //创建操作数据库对象
    ResultSet rs;                                           //创建数据集对象
    //创建各种面板对象
    JPanel panell1, panel2, panel3;
    Container c;                                           //创建容器对象
    //创建各种标签组件对象
    JLabel TipLabel = new JLabel("输入借阅者姓名和书名单击确定，将调出此书的相关信息");
}

```

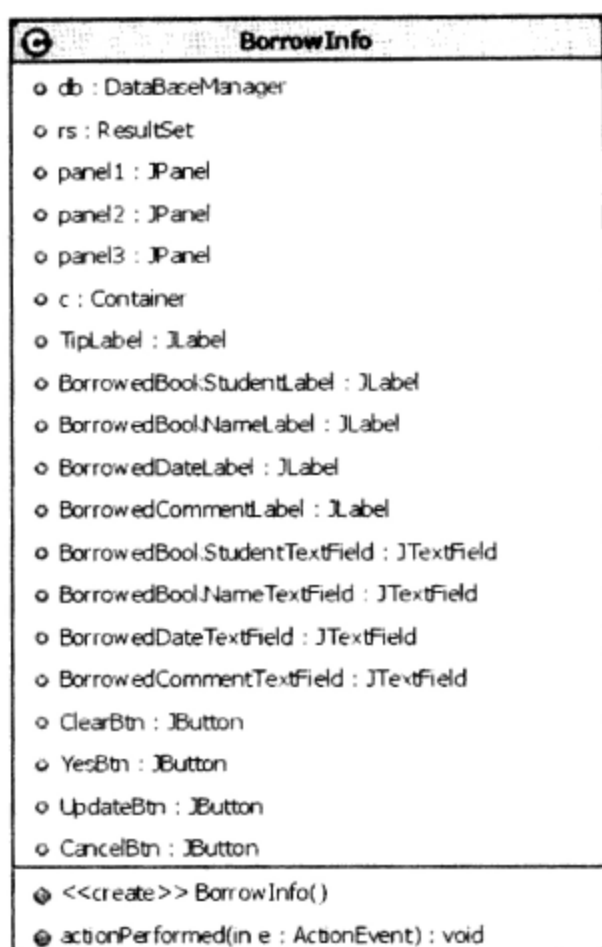



图 32.53 BorrowInfo 类的类图

```

JLabel BorrowedBookStudentLabel, BorrowedBookNameLabel, Borrowed-
DateLabel, BorrowedCommentLabel;
//创建各种文本框对象
JTextField BorrowedBookStudentTextField, BorrowedBookNameTextField,
    BorrowedDateTextField, BorrowedCommentTextField;
//创建各种按钮对象
JButton ClearBtn, YesBtn, UpdateBtn, CancelBtn;
public BorrowInfo() {                                     //构造函数
    super("修改图书出借信息");                             //设置标题
    c = getContentPane();                                 //为容器赋值
    c.setLayout(new BorderLayout());                         //设置布局管理器
    //创建和设置 panel3 对象
    panel3 = new JPanel();
    panel3.add(TipLabel);
    c.add(panel3, BorderLayout.NORTH);                       //添加对象 panel3 到容器 c 中
    //创建和设置 panel1 对象
    //为各种标签组件对象赋值
    BorrowedBookStudentLabel = new JLabel("借阅者姓名", JLabel.CENTER);
    BorrowedBookNameLabel = new JLabel("书名", JLabel.CENTER);
    BorrowedDateLabel = new JLabel("借书日期", JLabel.CENTER);
    BorrowedCommentLabel = new JLabel("备注", JLabel.CENTER);
    //为各种文本框对象赋值
    BorrowedBookStudentTextField = new JTextField(15);
    BorrowedBookNameTextField = new JTextField(15);
    BorrowedDateTextField = new JTextField(15);
    BorrowedCommentTextField = new JTextField(15);
    panel1 = new JPanel();                                   //为对象 panel1 赋值
    panel1.setLayout(new GridLayout(4, 2));                 //设置 panel1 布局管理器
    //添加各种标签对象和输入文本框对象到 panel1 中
    panel1.add(BorrowedBookStudentLabel);
    ...                                                     //省略部分代码
    c.add(panel1, BorderLayout.CENTER);                       //添加对象 panel1 到容器 c 中
  
```

```

//创建和设置 panel2 对象
panel2 = new JPanel(); //为对象 panel2 赋值
panel2.setLayout(new GridLayout(1, 4)); //设置布局管理器
//为各个按钮对象赋值
ClearBtn = new JButton("清空");
YesBtn = new JButton("确定");
UpdateBtn = new JButton("更新");
CancelBtn = new JButton("取消");
//为按钮对象注册事件
ClearBtn.addActionListener(this);
YesBtn.addActionListener(this);
UpdateBtn.addActionListener(this);
UpdateBtn.setEnabled(false);
CancelBtn.addActionListener(this);
//添加按钮对象到对象 panel2 中
panel2.add(ClearBtn);
panel2.add(YesBtn);
panel2.add(UpdateBtn);
panel2.add(CancelBtn);
c.add(panel2, BorderLayout.SOUTH); //添加对象 panel2 到容器 c 中
}

public void actionPerformed(ActionEvent e) { //事件监听器
    if (e.getSource() == ClearBtn) { //当发生事件的按钮为 ClearBtn
        //设置各个文本框的内容为空
        BorrowedBookStudentTextField.setText("");
        BorrowedBookNameTextField.setText("");
        BorrowedDateTextField.setText("");
        BorrowedCommentTextField.setText("");
    } else if (e.getSource() == CancelBtn) { //当发生事件的按钮为 CancelBtn 时
        this.dispose(); //退出系统
    } else if (e.getSource() == YesBtn) { //当发生事件的按钮为 YesBtn 时
        try {
            //创建查询 SQL 语句变量
            String strSQL = "select studentName,bookName, borrowDate,
com from BookBrowse where studentName='"
                + BorrowedBookStudentTextField.getText().trim()
                + "'and bookName='"
                + BorrowedBookNameTextField.getText().trim() + "'";
            rs = db.getResult(strSQL); //获取执行 SQL 语句的结果
            if (!rs.first()) { //当没有借相应书时
                //显示相应的信息
                JOptionPane.showMessageDialog(null, "此学生没有借过书!
或者没有此书!");
            } else {
                //为各种文本框的内容赋值
                BorrowedBookStudentTextField.setText(rs.getString(1));
                BorrowedBookNameTextField.setText(rs.getString(2));
                BorrowedDateTextField.setText(rs.getString(3));
                BorrowedCommentTextField.setText(rs.getString(4));
                UpdateBtn.setEnabled(true);
            }
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    } else if (e.getSource() == UpdateBtn) {
        //当发生事件的按钮为 UpdateBtn 时
        //创建 SQL 语句变量

```

```

String strSQL = "update bookBrowse set borrowDate=TO_DATE('"
    + BorrowedDateTextField.getText().trim() + "','',com='"
    + BorrowedCommentTextField.getText().trim()
    + "' where studentName='"
    + BorrowedBookStudentTextField.getText().trim()
    + "'and bookName='"
    + BorrowedBookNameTextField.getText().trim() + "'";
if (db.updateSql(strSQL)) {                                //判断执行结果
    //显示更新成功的信息
    JOptionPane.showMessageDialog(null, "更新成功!");
    db.closeConnection();                                //关闭数据库操作
    this.dispose();                                    //退出系统
} else {
    //显示更新失败的信息
    JOptionPane.showMessageDialog(null, "更新失败!");
    db.closeConnection();                                //关闭数据库操作
    this.dispose();                                    //退出系统
}
}
}
}

```

【代码解析】

- 上述代码的构造函数，主要用来实现图书管理系统中修改出借图书信息窗口，该用户界面涉及的具体容器、对象和布局如图 32.54 所示。

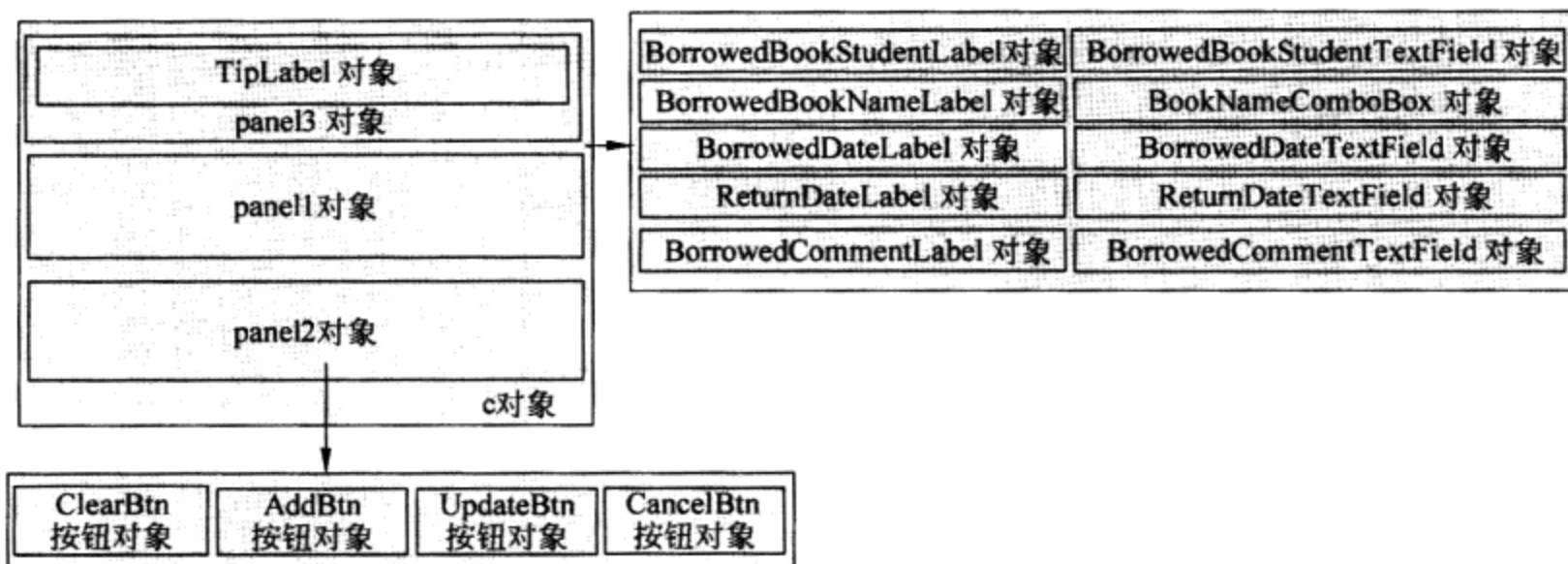


图 32.54 布局

- 在上述代码的处理监听事件方法中，如果发生动作的按钮为“清空”按钮，则相应的文本框内容将为空。如果发生动作的按钮为“取消”按钮，该窗口将不显示。如果发生动作的按钮为“确定”按钮，首先根据“借书学生”文本框和“出借书”文本框的内容创建查询 SQL 语句，然后执行该语句。如果结果为空则显示“此学生没有借过书！或者没有此书！”信息框，否则根据执行结果的内容设置各个文本框并设置“更新”按钮可用。如果发生动作的按钮为“更新”按钮，首先根据各个文本框的内容创建查更新 SQL 语句，如果该语句的执行结果为 true，则显示“更新成功”的信息框，否则显示“更新失败”的信息框。

32.5 图书管理系统项目——归还图书的操作

图书管理系统项目中为了实现还书的功能，分别实现了归还图书和修改归还图书信息操作，各种操作所对应的类如图 32.55 所示。

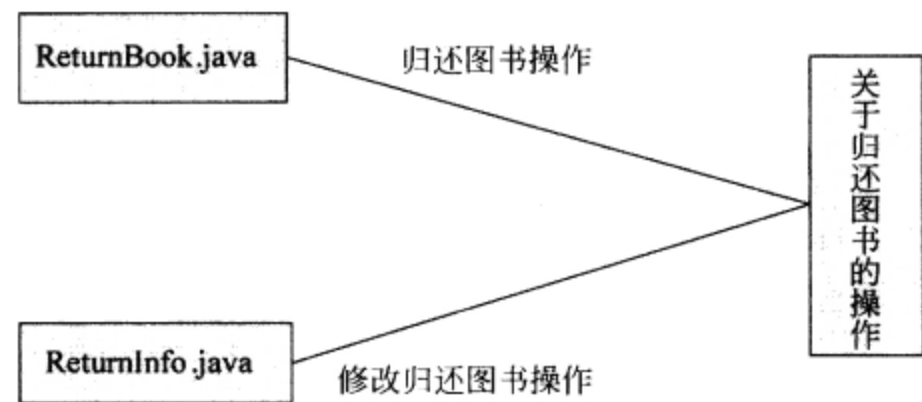


图 32.55 程序关系图

32.5.1 归还图书类

ReturnBook 类为图书管理系统项目中，实现归还图书操作的归还图书操作的类，该类不仅继承了 JFrame 类，而且还实现了监听各个组件相应动作的功能，该类的类图如图 32.56 所示，具体内容如代码 32.13 所示。



图 32.56 ReturnBook 类的类图

代码 32.13 还书方法：ReturnBook.java

```
public class ReturnBook extends JFrame implements ActionListener {
    //创建各种成员变量
```

```

DataBaseManager db = new DataBaseManager();           //创建操作数据库对象
ResultSet rs;                                         //创建数据集对象
//创建面板对象
JPanel panel1, panel2;
Container c;                                         //创建容器 c 对象
//创建各种标签对象
JLabel ReturnedBookStudentLabel, ReturnedBookNameLabel, Returned-
DateLabel,
    ReturnedCommentLabel;
//创建各种文本框对象
JTextField ReturnedBookStudentTextField, ReturnedDateTextField,
    ReturnedCommentTextField;
//创建按钮对象
JButton ClearBtn, YesBtn, CancelBtn;
JComboBox BookNameComboBox = new JComboBox();       //创建选择框对象
public ReturnBook() {                               //构造函数
    super("图书还入");                               //设置标题
    c = getContentPane();                           //为容器对象 c 赋值
    c.setLayout(new BorderLayout());                 //设置布局管理器
    //为各种标签和文本框对象赋值
    ReturnedBookStudentLabel = new JLabel("还书者姓名", JLabel.CENTER);
    ReturnedBookNameLabel = new JLabel("书名", JLabel.CENTER);
    ReturnedDateLabel = new JLabel("日期", JLabel.CENTER);
    ReturnedCommentLabel = new JLabel("备注", JLabel.CENTER);
    ReturnedBookStudentTextField = new JTextField(15);
    ReturnedDateTextField = new JTextField(15);
    ReturnedCommentTextField = new JTextField(15);
    try {                                             //获取相关 SQL 语句执行结果
        //创建 SQL 语句
        String s = "";                               //创建字符串
        //创建查询 SQL 语句
        String strSQL = "select bookName from bookBrowse where
is_returned='否'";
        rs = db.getResult(strSQL);                 //获取执行结果
        while (rs.next()) {                         //遍历执行结果
            //添加内容到下拉菜单中
            BookNameComboBox.addItem(rs.getString(1));
        }
    } catch (SQLException sqle) {
        System.out.println(sqle.toString());
    } catch (Exception ex) {
        System.out.println(ex.toString());
    }
    //创建和设置对象 panel1
    panel1 = new JPanel();                           //为对象 panel1 赋值
    panel1.setLayout(new GridLayout(4, 2));         //设置布局管理器
    //添加对象到对象 panel1 中
    panel1.add(ReturnedBookStudentLabel);
    ...                                              //省略部分代码
    c.add(panel1, BorderLayout.CENTER);             //添加对象 panel1 到容器 c 中
    //创建和设置对象 panel2
    panel2 = new JPanel();                           //为对象 panel2 赋值
    panel2.setLayout(new GridLayout(1, 3));         //设置布局管理器
    //为按钮对象赋值
    ClearBtn = new JButton("清空");

```

```

YesBtn = new JButton("确定");
CancelBtn = new JButton("取消");
//为按钮对象注册事件
ClearBtn.addActionListener(this);
YesBtn.addActionListener(this);
CancelBtn.addActionListener(this);
//添加按钮对象到对象 panel2 中
panel2.add(ClearBtn);
panel2.add(YesBtn);
panel2.add(CancelBtn);
c.add(panel2, BorderLayout.SOUTH);    //添加对象 panel2 到容器 c 中
}

public void actionPerformed(ActionEvent e) {    //事件监听器
    if (e.getSource() == CancelBtn) {    //当发生事件的组件为 CancelBtn 时
        this.dispose();    //退出系统
    } else if (e.getSource() == ClearBtn) {    //当发生事件的组件为 ClearBtn 时
        //各个文本框的内容为空
        ReturnedBookStudentTextField.setText("");
        ReturnedDateTextField.setText("");
        ReturnedCommentTextField.setText("");
    } else if (e.getSource() == YesBtn) {    //当发生事件的组件为 YesBtn 时
        //判断相应文本框的内容是否为空
        if (ReturnedBookStudentTextField.getText().trim().equals(""))

            JOptionPane.showMessageDialog(null, "请输入还书者的姓名...");
        } else if (BookNameComboBox.getSelectedItem().equals("")) {
            JOptionPane.showMessageDialog(null, "图书馆没有出借过书!");
        } else {    //当不为空时
            try {
                //创建实现更新功能的 SQL 语句
                String strSQL = "update bookBrowse set returnDate='"
                    + ReturnedDateTextField.getText().trim()
                    + "',com='"
                    + ReturnedCommentTextField.getText().trim()
                    + "',is_returned='是' where studentName='"
                    + ReturnedBookStudentTextField.getText().trim()
                    + "'and bookName='"
                    + BookNameComboBox.getSelectedItem() + "'";
                if (db.updateSql(strSQL)) {    //根据更新结果显示相应信息
                    //显示还书成功信息
                    JOptionPane.showMessageDialog(null, "还书完成!");
                    //创建更新数据库信息
                    strSQL = "update books set borrowed_count=
                        borrowed_count-1 where bookname='"
                        + BookNameComboBox.getSelectedItem() + "'";
                    db.updateSql(strSQL);    //执行数据库更新操作
                    db.closeConnection();    //关闭连接
                    this.dispose();    //退出系统
                } else {
                    //显示还书失败信息
                    JOptionPane.showMessageDialog(null, "还书失败!");
                    db.closeConnection();
                    this.dispose();
                }
            } catch (Exception ex) {
                System.out.println(ex.toString());
            }
        }
    }
}

```



```
    }  
    }  
    }  
}
```

【代码解析】

- 上述代码的构造函数主要用来实现图书管理系统中的归还图书窗口，该用户界面涉及的具体容器、对象和布局如图 32.57 所示。

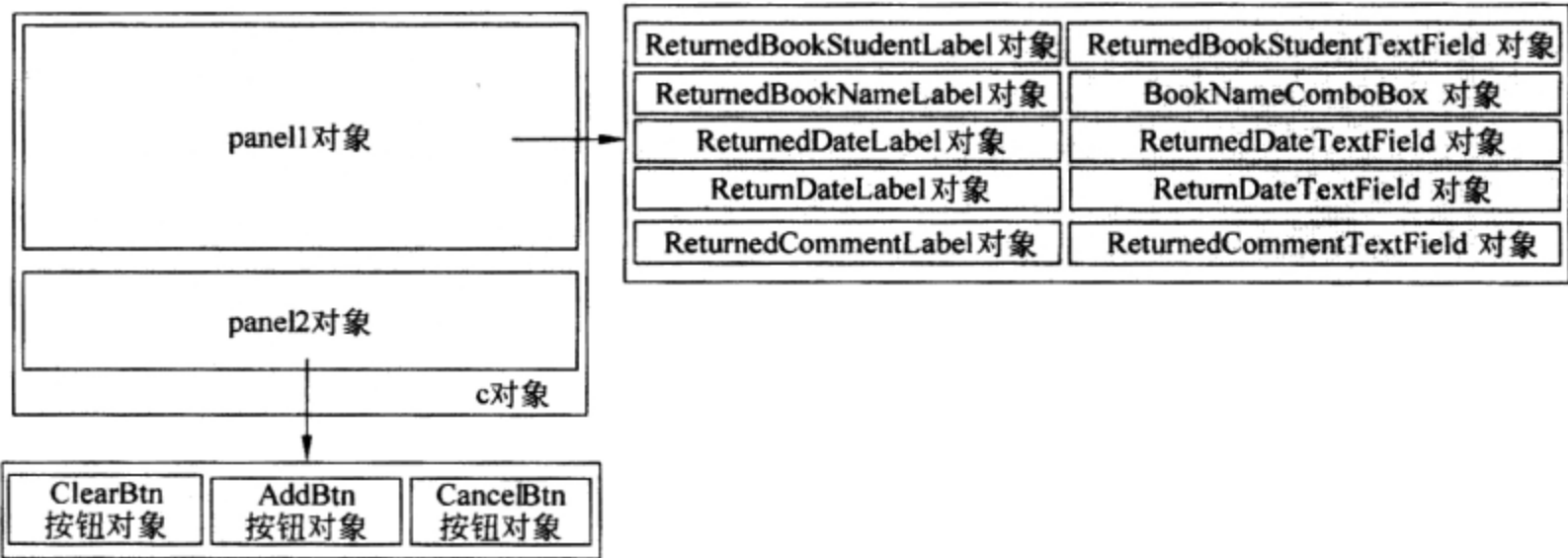


图 32.57 布局

- 在上述代码的处理监听事件方法中，如果发生动作的按钮为“取消”按钮，该窗口将不显示。如果发生动作的按钮为“清空”按钮，该窗口中相应文本框的内容为空。如果发生动作的按钮为“确定”按钮，首先判断“归还者”文本框的内容和书库中出借的书是否为空，然后根据各个文本框的内容创建 SQL 更新语句 strSQL 并执行该语句。如果更新 SQL 语句的执行结果为 true，则显示“还书完成”的信息框，同时更新 books 表格中的数据，否则显示“还书失败”的信息框。

32.5.2 修改归还图书信息类

ReturnInfo 类为“图书管理系统”项目中，实现归还图书操作的修改归还图书信息操作的类，该类不仅继承了 JFrame 类，而且还实现了监听各个组件相应动作的功能，该类的类图如图 32.58 所示，具体内容如代码 32.14 所示。

代码 32.14 修改还书信息方法：ReturnInfo.java

```
public class ReturnInfo extends JFrame implements ActionListener {  
    //创建各种成员变量  
    DataBaseManager db = new DataBaseManager();           //创建数据库操作对象  
    ResultSet rs;                                           //创建数据集对象  
    //创建面板对象  
    JPanel panel1, panel2;  
    Container c;                                           //创建容器对象  
    //创建各种标签对象  
    JLabel TipLabel = new JLabel("输入还书者姓名和书名单击确定，将调出此书的相关
```

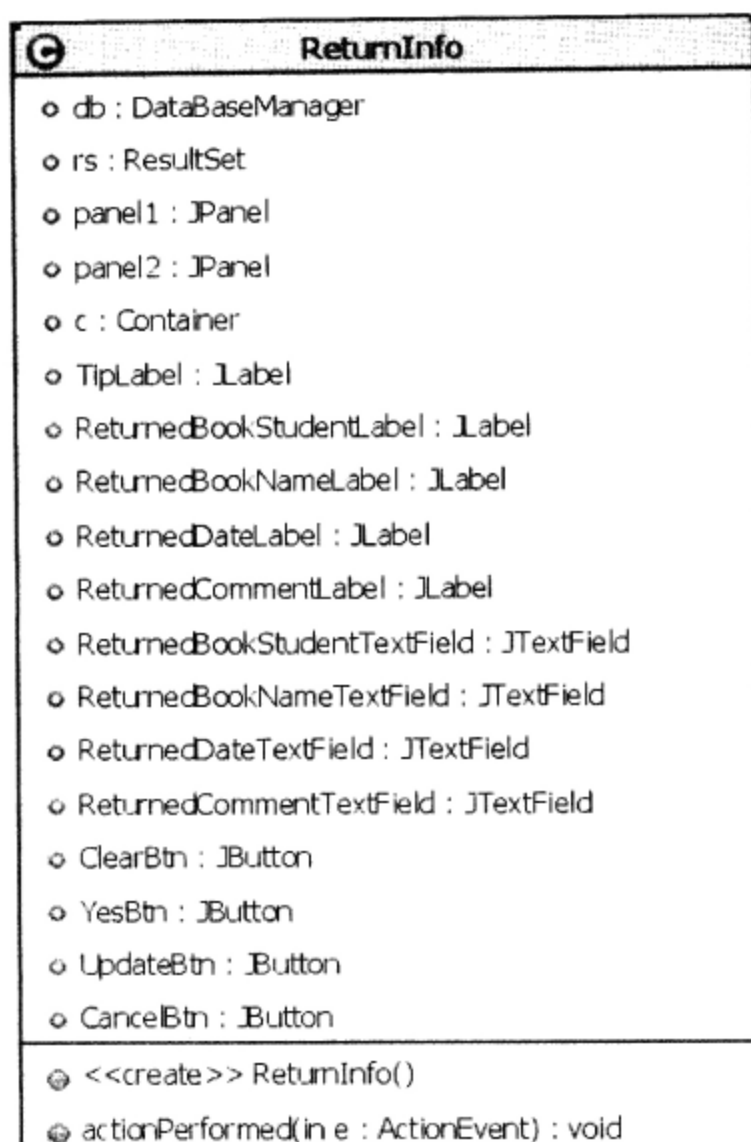


图 32.58 ReturnInfo 类的类图

信息");

JLabel ReturnedBookStudentLabel, ReturnedBookNameLabel, ReturnedDateLabel,

ReturnedCommentLabel;

//创建文本框对象

JTextField ReturnedBookStudentTextField, ReturnedBookNameTextField, ReturnedDateTextField, ReturnedCommentTextField;

//创建按钮对象

JButton ClearBtn, YesBtn, UpdateBtn, CancelBtn;

public ReturnInfo() {

//构造函数

super("修改图书还入信息");

//设置标题

c = getContentPane();

//获取容器对象 c

c.setLayout(new BorderLayout());

//设置布局管理器

//添加对象 TipLabel 到容器 c 中

c.add(TipLabel, BorderLayout.NORTH);

//为标签对象和文本框对象赋值

ReturnedBookStudentLabel = new JLabel("还书者姓名", JLabel.CENTER);

ReturnedBookNameLabel = new JLabel("书名", JLabel.CENTER);

ReturnedDateLabel = new JLabel("还书日期", JLabel.CENTER);

ReturnedCommentLabel = new JLabel("备注", JLabel.CENTER);

ReturnedBookStudentTextField = new JTextField(15);

ReturnedBookNameTextField = new JTextField(15);

ReturnedDateTextField = new JTextField(15);

ReturnedCommentTextField = new JTextField(15);

//创建和设置对象 panel1

panel1 = new JPanel();

//创建 panel1 对象

panel1.setLayout(new GridLayout(4, 2));

//设置布局管理器

```

//添加标签和文本框对象到对象 panel1 中
panel1.add(ReturnedBookStudentLabel);
...
c.add(panel1, BorderLayout.CENTER); //省略部分代码 //添加对象 panel1 到容器 c 中
//创建并设置 panel2 对象
panel2 = new JPanel(); //创建 panel2 对象
panel2.setLayout(new GridLayout(1, 4)); //设置布局管理器
//为按钮对象赋值
ClearBtn = new JButton("清空");
YesBtn = new JButton("确定");
UpdateBtn = new JButton("更新");
CancelBtn = new JButton("取消");
//为按钮对象注册事件
ClearBtn.addActionListener(this);
YesBtn.addActionListener(this);
UpdateBtn.addActionListener(this);
UpdateBtn.setEnabled(false);
CancelBtn.addActionListener(this);
//添加按钮到对象 panel2 中
panel2.add(ClearBtn);
panel2.add(YesBtn);
panel2.add(UpdateBtn);
panel2.add(CancelBtn);
c.add(panel2, BorderLayout.SOUTH); //添加 panel2 对象到容器 c 中
}

public void actionPerformed(ActionEvent e) { //事件监听器
    if (e.getSource() == ClearBtn) { //当发生事件的按钮为 ClearBtn 时
        //清空各个文本框里的内容
        ReturnedBookStudentTextField.setText("");
        ReturnedBookNameTextField.setText("");
        ReturnedDateTextField.setText("");
        ReturnedCommentTextField.setText("");
    } else if (e.getSource() == CancelBtn) {
        //当发生事件的按钮为 CancelBtn
        this.dispose(); //退出系统
    } else if (e.getSource() == YesBtn) { //当发生事件的按钮为 YesBtn
        try { //修改借出书的信息
            //创建查询 SQL 语句
            String strSQL = "select studentName,bookName, borrowDate,
com from BookBrowse where studentName='"
                + ReturnedBookStudentTextField.getText().trim()
                + "'and bookName='"
                + ReturnedBookNameTextField.getText().trim() + "'";
            rs = db.getResult(strSQL); //获取执行结果
            if (!rs.first()) { //当执行结果中无记录时
                //显示出错信息
                JOptionPane.showMessageDialog(null, "此学生没有借过书!
或者没有此书!");
            } else { //当执行结果中存在记录时
                //为相应的文本框赋值
                ReturnedBookStudentTextField.setText(rs.getString(1));
                ReturnedBookNameTextField.setText(rs.getString(2));
                ReturnedDateTextField.setText(rs.getString(3));
                ReturnedCommentTextField.setText(rs.getString(4));
                UpdateBtn.setEnabled(true);
            }
        } catch (Exception ex) {

```

```

        System.out.println(ex.toString());
    }
} else if (e.getSource() == UpdateBtn) {
    //当发生事件的按钮为 UpdateBtn 时
    //创建 SQL 语句
    String strSQL = "update bookBrowse set returnDate=TO_DATE('"
        + ReturnedDateTextField.getText().trim() + "','',com='"
        + ReturnedCommentTextField.getText().trim()
        + "' where studentName='"
        + ReturnedBookStudentTextField.getText().trim()
        + "'and bookName='"
        + ReturnedBookNameTextField.getText().trim() + "'";
    if (db.updateSql(strSQL)) { //根据执行结果进行判断
        //当更新成功后
        JOptionPane.showMessageDialog(null, "更新成功!");
        db.closeConnection();
        this.dispose();
    } else {
        //当更新失败后
        JOptionPane.showMessageDialog(null, "更新失败!");
        db.closeConnection();
        this.dispose();
    }
}
}
}
}

```

【代码解析】

- 上述代码的构造函数主要用来实现图书管理系统中修改出借图书信息的窗口，该用户界面涉及的具体容器、对象和布局如图 32.59 所示。

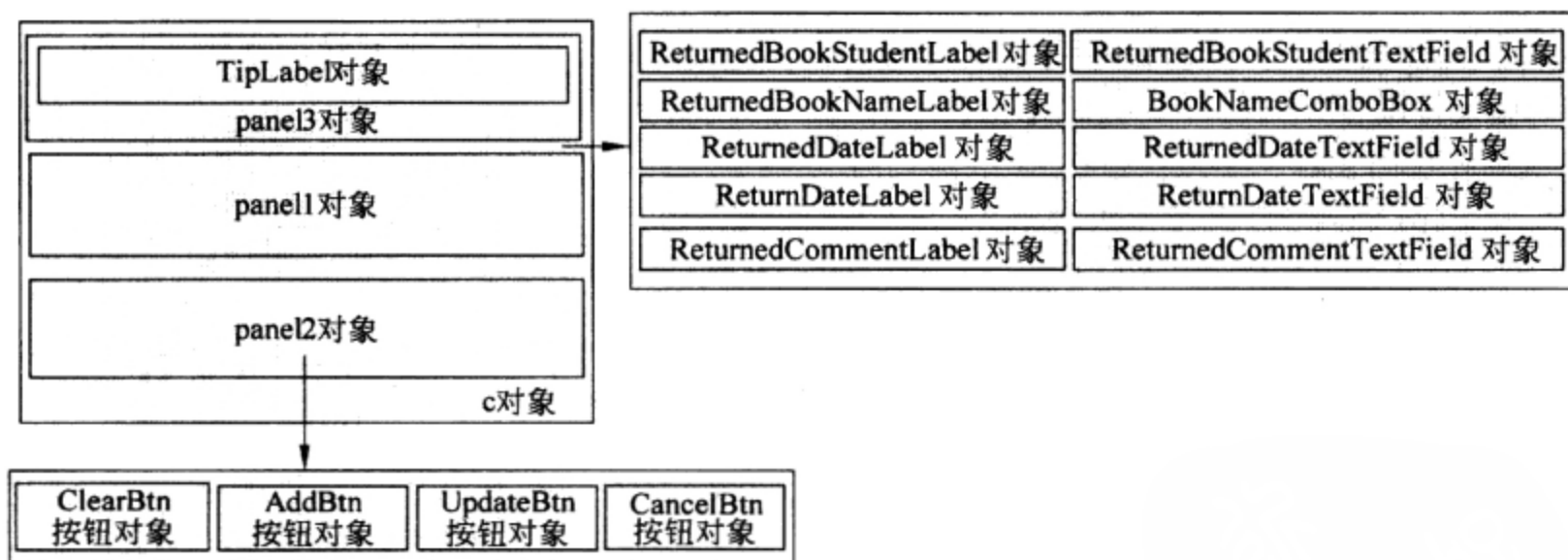


图 32.59 布局

- 在上述代码的处理监听事件方法中，如果发生动作的按钮为“清空”按钮，则相应文本框内容将为空。如果发生动作的按钮为“取消”按钮，该窗口将不显示。如果发生动作的按钮为“确定”按钮，首先根据“还书学生”文本框和“归还书”文本框的内容创建查询 SQL 语句，然后执行该语句。如果结果为空则显示“此学生没有借过书！或者没有此书！”信息框，否则根据执行结果的内容设置各个文本框并设置“更新”按钮可用；如果发生动作的按钮为“更新”按钮，首先根据各个文本框的内容创建并更新 SQL 语句，如果该语句的执行结果为 true，则显示

“更新成功”的信息框，否则显示“更新失败”的信息框。

32.6 图书管理系统项目——该项目的其他类

前面的内容中讲解了图书管理系统项目的所有逻辑功能，例如图书操作的功能、用户操作的功能、出借图书的功能和归还图书的功能。那么如何把这些功能联系在一起呢？这就涉及该项目的主界面。对用户、图书等进行操作时，都需要操作数据，即与数据连接并执行相应的 SQL 语句。为了便于编写，该项目创建了一个数据库连接和操作的工具类。

32.6.1 主窗口类

MainWindow 类为“图书管理系统”项目的主界面，该类不仅继承了 JFrame 类，而且还实现了监听各个组件相应动作的功能，该类的类图如图 32.60 所示。由于该类的内容比较多，所以分成 3 个步骤来讲解，分别如下。

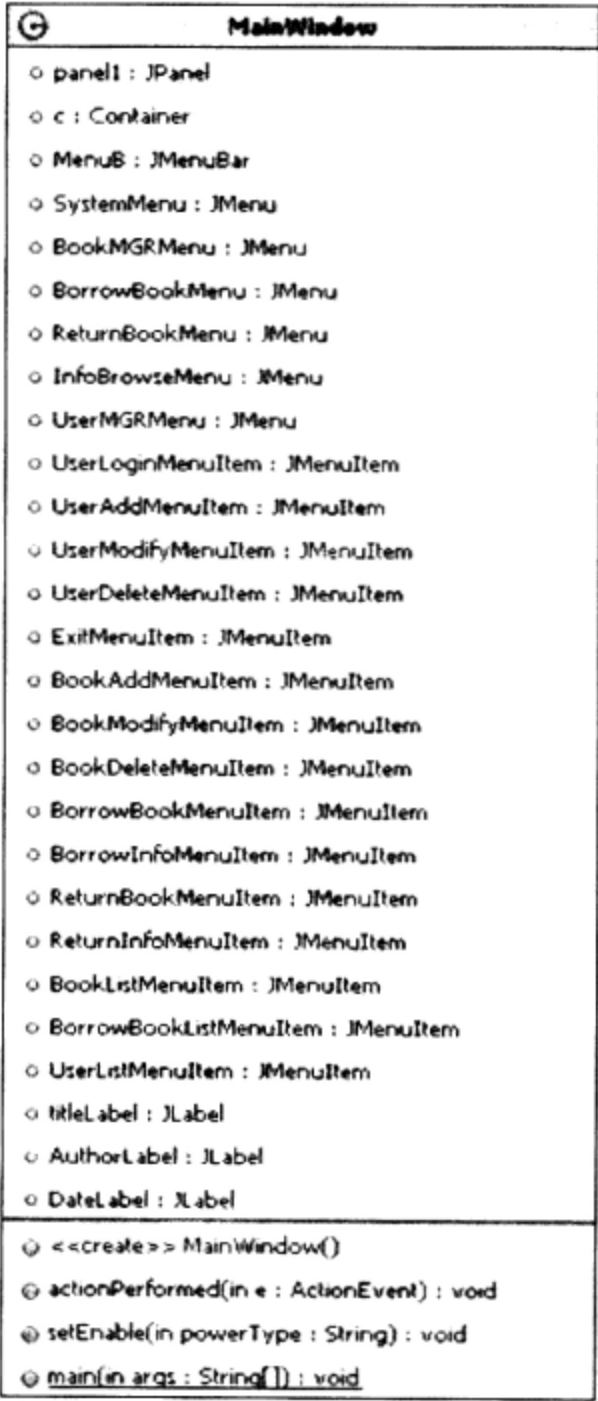


图 32.60 MainWindow 类的类图

(1) 在主界面类中实现了图书管理项目的界面，界面的内容如代码 32.15 所示。

代码 32.15 界面的主窗口类：MainWindow.java

```
public class MainWindow extends JFrame implements ActionListener {
    //创建成员变量
    JPanel panell;                                //创建面板对象
    Container c;                                  //创建容器对象
    JMenuBar MenuB;                               //创建菜单工具栏对象
    //创建菜单对象
    JMenu SystemMenu, BookMGRMenu, BorrowBookMenu, ReturnBookMenu,
        InfoBrowseMenu, UserMGRMenu;
    //创建菜单选项对象
    JMenuItem UserLoginMenuItem, UserAddMenuItem, UserModifyMenuItem,
        UserDeleteMenuItem, ExitMenuItem, BookAddMenuItem,
        BookModifyMenuItem, BookDeleteMenuItem, BorrowBookMenuItem,
        BorrowInfoMenuItem, ReturnBookMenuItem, ReturnInfoMenuItem,
        BookListMenuItem, BorrowBookListMenuItem, UserListMenuItem;
    //创建标签对象
    JLabel titleLabel, AuthorLabel, DateLabel;
    public MainWindow() {                          //构造函数
        super("图书馆管理系统");                  //设置窗口标题
        //创建并设置系统管理菜单
        MenuB = new JMenuBar();                  //为菜单工具栏赋值
        //为菜单变量赋值
        SystemMenu = new JMenu("系统管理");
        UserMGRMenu = new JMenu("用户管理");
        //为菜单选项赋值
        UserLoginMenuItem = new JMenuItem("用户登录");
        UserAddMenuItem = new JMenuItem("添加用户");
        UserModifyMenuItem = new JMenuItem("修改用户");
        UserDeleteMenuItem = new JMenuItem("删除用户");
        ExitMenuItem = new JMenuItem("退出");
        //为菜单 SystemMenu 添加菜单选项
        SystemMenu.add(UserLoginMenuItem);
        SystemMenu.add(UserMGRMenu);
        SystemMenu.add(ExitMenuItem);
        //为菜单 UserMGRMenu 添加菜单选项
        UserMGRMenu.add(UserAddMenuItem);
        UserMGRMenu.add(UserModifyMenuItem);
        UserMGRMenu.add(UserDeleteMenuItem);
        //为菜单选项注册监听器
        UserLoginMenuItem.addActionListener(this);
        UserAddMenuItem.addActionListener(this);
        UserModifyMenuItem.addActionListener(this);
        UserDeleteMenuItem.addActionListener(this);
        ExitMenuItem.addActionListener(this);
        MenuB.add(SystemMenu);                  //为菜单工具栏注册监听器
        //创建和设置书籍管理菜单
        BookMGRMenu = new JMenu("书籍管理");      //为菜单变量 BookMGRMenu 赋值
        //为菜单项赋值
        BookAddMenuItem = new JMenuItem("添加书籍");
        BookModifyMenuItem = new JMenuItem("修改书籍");
        BookDeleteMenuItem = new JMenuItem("删除书籍");
        //添加菜单项到菜单变量 BookMGRMenu 中
        BookMGRMenu.add(BookAddMenuItem);
    }
}
```



```

BookMGRMenu.add(BookModifyMenuItem);
BookMGRMenu.add(BookDeleteMenuItem);
//为菜单项注册事件
BookAddMenuItem.addActionListener(this);
BookModifyMenuItem.addActionListener(this);
BookDeleteMenuItem.addActionListener(this);
MenuB.add(BookMGRMenu);           //添加菜单 BookMGRMenu 到菜单工具栏中
//创建并设置借书管理菜单
BorrowBookMenu = new JMenu("借书管理"); //为菜单变量 BorrowBookMenu 赋值
//为各种菜单选项赋值
BorrowBookMenuItem = new JMenuItem("书籍出借");
BorrowInfoMenuItem = new JMenuItem("出借信息修改");
//为菜单变量注册事件
BorrowBookMenu.add(BorrowBookMenuItem);
BorrowBookMenu.add(BorrowInfoMenuItem);
//为菜单选项注册事件
BorrowBookMenuItem.addActionListener(this);
BorrowInfoMenuItem.addActionListener(this);
MenuB.add(BorrowBookMenu); //添加菜单变量 BorrowBookMenu 到工具栏中
//创建并设置还书管理菜单
ReturnBookMenu = new JMenu("还书管理"); //为菜单变量赋值
...                                     //省略部分代码
//初始化界面
titleLabel = new JLabel(new ImageIcon(".\\pic.jpg"));
c = getContentPane();                //获取容器
c.setLayout(new BorderLayout());      //设置布局管理器
//创建并设置面板 panell
panell = new JPanel();
panell.setLayout(new BorderLayout()); //设置布局管理器
panell.add(titleLabel, BorderLayout.CENTER);
...                                     //省略部分代码
c.add(panell, BorderLayout.CENTER); //添加面板对象 panell 到容器对象 c 中
setBounds(100, 50, 400, 300);      //设置窗口大小
show();                             //显示窗口
//设置初始功能
UserMGRMenu.setEnabled(false);
BookMGRMenu.setEnabled(false);
BorrowBookMenu.setEnabled(false);
ReturnBookMenu.setEnabled(false);
InfoBrowseMenu.setEnabled(false);
}
...                                     //省略部分代码
public static void main(String args[]) { //主函数
    MainWindow mainFrame = new MainWindow(); //创建 MainWindow 对象
}
}

```

【代码解析】

上述代码主要实现了图书管理项目程序的入口,即该项目的主界面,该主界面涉及的具体容器、对象和布局如图 32.61 所示。

(2) 接着为主界面中的所有菜单注册动作事件,并创建处理该事件的方法,具体内容如代码 32.16 所示。

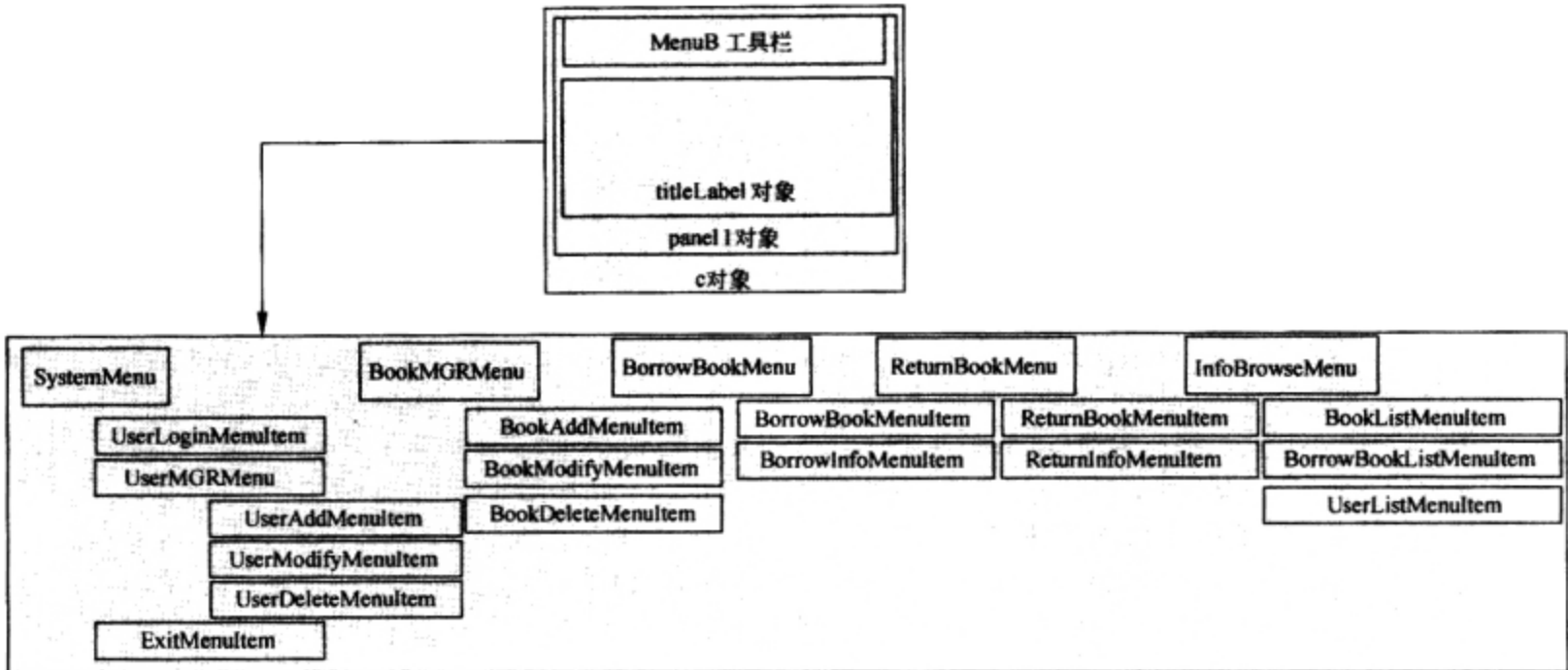


图 32.61 布局

代码 32.16 实现事件注册的主窗口类: MainWindow.java

```

public class MainWindow extends JFrame implements ActionListener {
    ...
    //创建和设置每个菜单点击后出现的窗口和窗口显示的位置
    public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand() == "用户登录") {
            //创建用户登录界面
            UserLogin UserLoginFrame = new UserLogin(this);
            //获取用户登录界大小
            Dimension FrameSize = UserLoginFrame.getPreferredSize();
            //获取主界大小
            Dimension MainFrameSize = getSize();
            Point loc = getLocation(); //获取窗口显示位置的坐标点
            //设置登录窗口的位置
            UserLoginFrame.setLocation((MainFrameSize.width -
                FrameSize.width)
                / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                + loc.y);
            //显示登录窗口
            UserLoginFrame.pack();
            UserLoginFrame.show();
        } else if (e.getActionCommand() == "添加用户") {
            //创建用户添加界面
            UserAdd UserAddFrame = new UserAdd();
            //获取用户添加界面的大小
            Dimension FrameSize = UserAddFrame.getPreferredSize();
            //获取主界面的大小
            Dimension MainFrameSize = getSize();
            Point loc = getLocation(); //获取窗口显示位置的坐标点
            //设置用户添加窗口的位置
            UserAddFrame.setLocation((MainFrameSize.width - FrameSize.
                width)
                / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                + loc.y);
            //显示用户添加窗口
        }
    }
}

```

```

        UserAddFrame.pack();
        UserAddFrame.show();
    } else if (e.getActionCommand() == "修改用户") {
        //创建用户修改界面
        UserModify UserModifyFrame = new UserModify();
        //获取用户修改界面的大小
        Dimension FrameSize = UserModifyFrame.getPreferredSize();
        //获取主界面的大小
        Dimension MainFrameSize = getSize();
        Point loc = getLocation(); //获取窗口显示位置的坐标点
        //设置用户修改窗口的位置
        UserModifyFrame.setLocation((MainFrameSize.width -
            FrameSize.width)
            / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
            + loc.y);
        //显示用户修改窗口
        UserModifyFrame.pack();
        UserModifyFrame.show();
    } else if (e.getActionCommand() == "删除用户") {
        //创建用户删除界面
        UserDelete UserDeleteFrame = new UserDelete();
        //获取用户删除界面的大小
        Dimension FrameSize = UserDeleteFrame.getPreferredSize();
        //获取主界面的大小
        Dimension MainFrameSize = getSize();
        Point loc = getLocation(); //获取窗口显示位置的坐标点
        //设置用户删除窗口的位置
        UserDeleteFrame.setLocation((MainFrameSize.width -
            FrameSize.width)
            / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
            + loc.y);
        //显示用户删除窗口
        UserDeleteFrame.pack();
        UserDeleteFrame.show();
    } else if (e.getActionCommand() == "添加书籍") {
        //创建用户添加界面
        BookAdd BookAddFrame = new BookAdd();
        //获取用户添加界面的大小
        Dimension FrameSize = BookAddFrame.getPreferredSize();
        //获取主界面的大小
        Dimension MainFrameSize = getSize();
        Point loc = getLocation(); //获取窗口显示位置的坐标点
        //设置用户添加窗口的位置
        BookAddFrame.setLocation((MainFrameSize.width - FrameSize.
            width)
            / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
            + loc.y);
        //显示用户添加窗口
        BookAddFrame.pack();
        BookAddFrame.show();
    } else if (e.getActionCommand() == "修改书籍") {
        //创建书籍修改界面
        BookModify BookModifyFrame = new BookModify();
        //获取书籍修改界面的大小
        Dimension FrameSize = BookModifyFrame.getPreferredSize();
        //获取主界面的大小
        Dimension MainFrameSize = getSize();

```

```

        Point loc = getLocation(); //获取窗口显示位置的坐标点
        //设置书籍修改窗口的位置
        BookModifyFrame.setLocation((MainFrameSize.width -
            FrameSize.width)
            / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
            + loc.y);
        //显示书籍修改窗口
        BookModifyFrame.pack();
        BookModifyFrame.show();
    }
    ... //省略部分代码
    else if (e.getActionCommand() == "退出") {
        //退出系统
        this.dispose();
        System.exit(0);
    }
}
... //省略部分代码
}

```

【代码解析】

- ❑ 在上述代码中主要用来实现当菜单被单击后就会出现相应的窗口，并且显示窗口的位置在主界面的中间。
- ❑ 当选择“用户登录”菜单项时，就会出现 UserLogin 类的窗口对象。当选择“添加用户”菜单项时，就会出现 UserAdd 类的窗口对象。当选择“修改用户”菜单项时，就会出现 UserModify 类的窗口对象。当选择“删除用户”菜单项时，就会出现 UserDelete 类的窗口对象；当选择“添加书籍”菜单项时，就会出现 BookAdd 类的窗口对象。当选择“修改书籍”菜单项时，就会出现 BookModify 类的窗口对象。当选择“删除书籍”菜单项时，就会出现 BookDelete 类的窗口对象。当选择“书籍出借”菜单项时，就会出现 BorrowBook 类的窗口对象。当选择“出借信息修改”菜单项时，就会出现 BorrowInfo 类的窗口对象。当选择“书籍还入”菜单项时，就会出现 ReturnBook 类的窗口对象。当选择“书籍还入信息修改”菜单项时，就会出现 ReturnInfo 类的窗口对象。当选择“书籍列表”菜单项时，就会出现 BookList 类的窗口对象。当选择“借阅情况表”菜单项时，就会出现 BorrowBookList 类的窗口对象。当选择“用户列表”菜单项时，就会出现 UserList 类的窗口对象。当选择“退出”菜单项时，就会实现主窗口不显示的功能。

(3) 最后设置拥有不同权限的用户登录时，可以操作不同的菜单，具体内容如图 32.17 所示。

代码 32.17 实现权限设置的主窗口类：MainWindow.java

```

public class MainWindow extends JFrame implements ActionListener {
    ... //省略部分代码
    // 创建和设置登录用户的权限
    public void setEnable(String powerType) {
        if (powerType.trim().equals("系统管理员")) { //当为系统管理员时
            //设置其可以使用的功能
            UserMGRMenu.setEnabled(true);
            BookMGRMenu.setEnabled(true);
            BorrowBookMenu.setEnabled(true);
        }
    }
}

```

```
ReturnBookMenu.setEnabled(true);
InfoBrowseMenu.setEnabled(true);
UserListItem.setEnabled(true);
} else if (powerType.trim().equals("书籍管理员")) { //当为书籍管理员时
    //设置其可以使用的功能
    UserMGRMenu.setEnabled(false);
    BookMGRMenu.setEnabled(true);
    BorrowBookMenu.setEnabled(false);
    ReturnBookMenu.setEnabled(false);
    InfoBrowseMenu.setEnabled(true);
    UserListItem.setEnabled(false);
} else if (powerType.trim().equals("借阅管理员")) { //当为借阅管理员时
    //设置其可以使用的功能
    UserMGRMenu.setEnabled(false);
    BookMGRMenu.setEnabled(false);
    BorrowBookMenu.setEnabled(true);
    ReturnBookMenu.setEnabled(true);
    InfoBrowseMenu.setEnabled(true);
    UserListItem.setEnabled(false);
} else if (powerType.trim().equals("else")) { //当为其他成员时
    //设置其可以使用的功能
    UserMGRMenu.setEnabled(false);
    BookMGRMenu.setEnabled(false);
    BorrowBookMenu.setEnabled(false);
    ReturnBookMenu.setEnabled(false);
    InfoBrowseMenu.setEnabled(false);
}
...
}
```

//省略部分代码

【代码解析】

上述代码主要用来实现用户的权限，即当“系统管理员”登录系统时，所有菜单都能使用；当其他权限的用户登录时，则只能使用部分菜单。

32.6.2 数据库操作

在图书管理系统项目中为了方便操作，创建了一个数据库操作的类 DataBaseManager，其类图如图 32.62 所示。在该类中不仅实现了数据库的连接，而且还是实现了数据库的操作，具体内容如代码 32.18 所示。

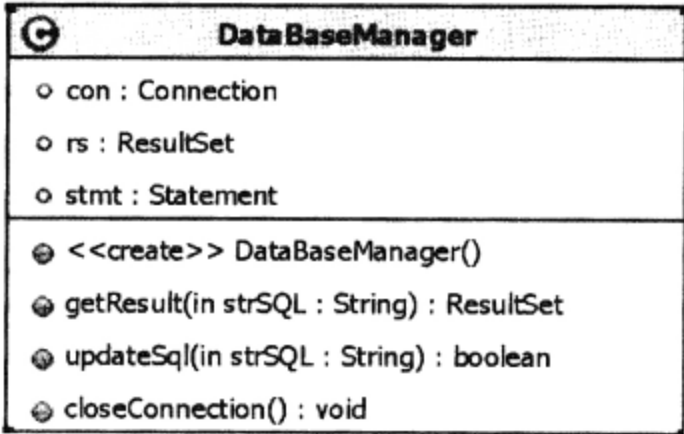


图 32.62 数据库操作类 UML 图

代码 32.18 数据库的基本操作: DataBaseManager.java

```

public class DataBaseManager {
    //创建成员变量
    Connection con;                //数据库连接对象
    ResultSet rs;                  //数据集对象
    Statement stmt;                //数据状态对象
    public DataBaseManager() {      //构造函数
        try {
            //加载驱动器
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con = DriverManager.getConnection(
                "jdbc:oracle:thin:@192.168.1.102:1521:orcl", "scott",
                "root");            //获取数据库连接对象
            //数据状态对象
            stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_UPDATABLE);
        } catch (ClassNotFoundException cnfex) {
            //抛出异常信息
            System.err.println("Failed to load JDBC/ODBC driver.");
            cnfex.printStackTrace();
            System.exit(1);
        } catch (SQLException sqle) {
            System.out.println(sqle.toString()); //抛出异常信息
        }
    }
    public ResultSet getResult(String strSQL) { //获取数据集对象
        try {
            rs = stmt.executeQuery(strSQL);      //获取数据集对象
            return rs;                          //返回数据集对象
        } catch (SQLException sqle) {
            System.out.println(sqle.toString());
            return null;
        }
    }
    public boolean updateSql(String strSQL) { //实现更新数据方法
        try {
            stmt.executeUpdate(strSQL);          //执行更新操作
            con.commit();                        //执行提交
            return true;
        } catch (SQLException sqle) {
            System.out.println(sqle.toString());
            return false;
        }
    }
    public void closeConnection() { //执行关闭连接方法
        try {
            con.close();                        //执行关闭连接
        } catch (SQLException sqle) {
            System.out.println(sqle.toString());
        }
    }
}

```

【代码解析】

上述代码中首先创建了3个成员变量: 数据库连接对象 con、数据集对象 rs 和数据状态对象 stmt, 然后在该类的构造函数中初始化数据库连接对象和数据状态对象。为了能够

方便其他类的使用,在该类中还创建了获取数据集的 `getResult()` 方法。对于数据库,还经常需要用到更新数据库的 `updateSql()` 方法和关闭数据库连接的 `closeConnection()` 方法。

32.7 小 结

本章主要介绍了一个完整的图书管理项目,该项目实现了流行的图书管理系统的基本功能,基于 Java 语言和数据库 Oracle 构建而成。在具体实现图书管理项目时,详细讲解了该系统涉及的 4 个对象和这些对象的相关操作,如图书对象和该对象的相关操作、用户对象和该对象的相关操作、出借图书的相关操作和归还图书操作。最后还详细介绍了该项目的用户界面类和数据库操作类。